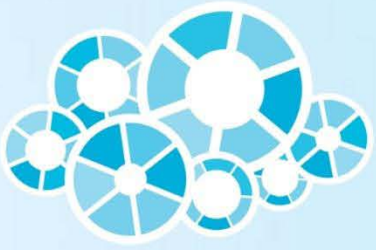
A WebMatrix csapat és Balássy György

Honlapépítés a XXI. században



Kezdőknek és haladóknak

devPortal

Honlapépítés a XXI. században

A Microsoft ASP.NET Web Pages egy ingyenes webfejlesztő technológia, melyet úgy terveztek meg, hogy a világ legjobb fejlesztői élményét nyújtsa webfejlesztők számára. Ezen könyv segítségével megtanulhatjuk az internet és a Web működését, továbbá képesek leszünk XXI. századi, korszerű dinamikus webes tartalmak készítésére az ASP.NET és a Razor szintaxis segítségével.

Megjegyzés: Ez a dokumentum a Microsoft WebMatrix and ASP.NET Web Pages Beta 3 kiadásához készült. A termék a fejlesztés során némileg változhat. A legújabb információkért látogassunk el a http://www.asp.net/webmatrix oldalra.

A Microsoftos fejlesztői közösség WebMatrix-os oldalát az alábbi linken találjuk: http://webmatrix.devportal.hu

Weblapfejlesztéssel, WebMatrix-szal, Razor nyelvvel kapcsolatos szakami kérdéseinket az alábbi fórumon tehetjük fel: http://devportal.hu/forums/348.aspx

A Facebookon a WebMatrix magyarországi rajongó táborához való csatlakozással értesülhetünk a legújabb hírekről! WebMatrix Magyarország

A könyv fordítása és lektorálása során a kiadó a legnagyobb gondossággal járt el. Az esetleges hibákat és észrevételeket a jos@jos.hu címen szívesen fogadjuk.

Az 1. fejezet Balássy György munkája
Internet: http://www.msdnkk.hu/Users/balassy

Szerkesztette és szakmailag lektorálta: Szabó Vince

Anyanyelvi lektor: Dr. Bonhardtné Hoffman Ildikó

© Kiadó: Jedlik Oktatási Stúdió Kft.
1212 Budapest, Táncsics Mihály u. 92.
Internet: http://www.jos.hu
E-mail: jos@jos.hu
Felelős kiadó: a Jedlik Oktatási Stúdió Kft. ügyvezetője

Tartalomjegyzék

1. fejezet – Balássy György: Bevezetés a web programozásába

Az internet működése

Avagy honnan jön a vezeték és mi történik benne?

Az ASP.NET WebPages technológia és a Microsoft WebMatrix segítségével nagyon egyszerűen és hatékonyan készíthetünk új weboldalakat, interaktív webhelyeket. A hatékonyság és az egyszerűség egyik kulcsa, hogy a rendszer számos terhet levesz a vállunkról, azaz bizonyos problémákat nem nekünk kell megoldanunk, hanem készen kapjuk a működő megoldást. Néha azonban előfordul, hogy a legjobb igyekezetünk ellenére valami mégsem működik, biztos mindannyian találkoztunk már a *„nincs internet”,* a *„nem jön be az oldal”* vagy épp a *„nem töltődik le a fájl”* jelenségekkel. Ezek mindennapos problémák egy webfejlesztő életében, melyek megoldásához nélkülözhetetlen, hogy értsük, hogyan is működik az internet.

Ennek a fejezetnek a célja, hogy megismerjük, mi történik a háttérben, miközben egy weboldalt nézünk a böngészőben. A fejezet végére egyértelmű lesz, hogy a gépünkből kilógó kábel hova fut a világban, és hogyan találja meg a böngésző a webszervert a beírt webcím alapján.

Géptől gépig

Ahhoz, hogy az önálló számítógépeket *világhálóba* tudjuk szervezni, természetesen valahogy fizikailag is össze kell tudnunk kapcsolni őket. Az internet nélkülözhetetlenségének köszönhetően a mai számítógépeknek ma már szinte kötelező tartozékuk az erre a célra szolgáló *hálózati kártya* (más néven *hálózati csatoló*, angolul *network interface card*, vagy egyszerűen csak *NIC*). Ha megnézzük a gépünk csatlakozóit, valami ehhez hasonlót biztosan találunk rajta, róla van szó:

1. ábra - A hálózati csatlakozó a számítógép hátoldalán

A hálózati kártya régebben a nevéhez hűen valóban önálló, kártya formájú, opcionális része volt a számítógépnek, ma már a számítógép nélkülözhetetlen részévé vált. Sőt, ha hordozható számítógépünk van, akkor valószínűleg egy *vezeték nélküli hálózati kártya* (*wireless network card*) is található a gépben. Mivel ehhez nem csatlakozik kábel, legfeljebb annyit fogunk észrevenni belőle, hogy sok gépen egy külön kapcsolóval szabályozhatjuk, hogy mikor legyen be- vagy kikapcsolva a vezeték nélküli hálózati kártyánk, ezzel is csökkentve a laptopunk fogyasztását.

A hálózati kártyába kell bedugnunk a *hálózati kábel*t. A csatlakozója (hivatalosan *RJ45*-ként szokták feltüntetni) nagyon hasonló a telefonkábel csatlakozójához (amit pontosan *RJ11*-nek hívnak), csak egy kicsit szélesebb, ezért ha nem erőltetjük, nem tudjuk rossz helyre dugni. Ha ilyen kábelre van szükségünk, a boltban *UTP patch kábel* néven vásárolhatjuk meg, különböző hosszúságokban.

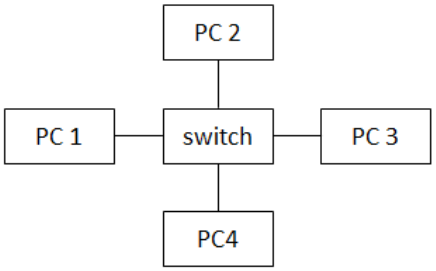
2. ábra - Hálózati kábel

> **MEGJEGYZÉS:** Ebben a könyvben a *hálózat* szó alatt mindenhol a számítógép-hálózatokat értjük, ezt még véletlenül se keverjük össze a 230V-os elektromos hálózattal! A *hálózati kábel*eket (*network cable* és nem *power cable*) és eszközöket gyengeáramú jelek továbbítására tervezték, és rossz néven veszik, ha a fali konnektorba erőszakoljuk őket.

A nagy kérdés persze az, hogy a gépünkből induló kábel hova megy tovább. Az internet olyan óriási hálózat, hogy nem lehet minden egyes számítógépet egyetlen központi csomóponthoz csatlakoztatni, ezért inkább hálózatok hálózataként valósították meg. Egy-egy kisebb alhálózatban (*local area network* [LAN], vagy egyszerűen csak *subnet*) a gépeket legtöbbször csillag alakzatban rendezik el, azaz van egy helyi központi eszköz, amelyhez az összes környéken lévő számítógép csatlakozik, és amely képes a gépek között zajló forgalomban rendet tartani. Erre a feladatra egy aktív hálózati eszköz, a *hálózati kapcsoló* (amit magyarul is inkább *switch*nek szoktunk hívni) szolgál.

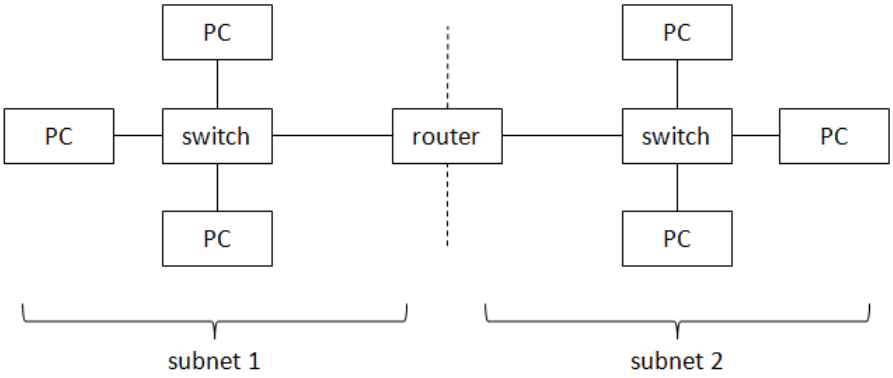
3. ábra - Egy 48 portos switch

Egy-egy switchre 8, 16, 24, 48 számítógépet lehet csatlakoztatni attól függően, hogy hány csatlakozója (azaz *port*ja) van, a switch pedig a csillag közepén gondoskodik arról, hogy a gépek megtalálják egymást.



4. ábra - Négy számítógép összekötése egy switch segítségével

De mi van akkor, ha egy számítógép egy olyan számítógéppel kíván kommunikálni, amelyik nem erre a switchre csatlakozik? Itt egy újabb hálózati eszközre van szükség, ami okosabb, mint a switch és képes egy switchre csatlakoztatott összes gépet a világháló többi részéhez kapcsolni. Ez az eszköz az útvonalválasztó (*router* vagy *default gateway*).



5. ábra - Két alhálózat összekapcsolása routerrel

Ha csak egyetlen számítógéppel szeretnénk az internetre csatlakozni, akkor előfordulhat, hogy otthon sem switchcsel, sem routerrel nem fogunk találkozni. Ha azonban több gépet szeretnénk egymással és a világhálóval összekötni, akkor vásárolhatunk olyan eszközt, amely egyben képes ellátni a router és a switch funkcióit is.

Számok a gépeknek

Miután megértettük, hogy az interneten lévő gépek fizikailag hogyan kapcsolódnak egymáshoz, ejtsünk pár szót arról, hogy ezeken a kábeleken hogyan vándorolnak az adatok!

Fizikai cím

A gépek egymást számokkal azonosítják, azaz minden egyes gépnek van egy önálló *címe*. A hálózati kártyák gyártói minden egyes kártyának adnak egy egyedi címet, ún. *Media Access Control address*t, röviden *MAC-address*t, vagy más néven fizikai címet. Ez egy 6 bájtos cím, egyedi a világon, az én gépemben lévő hálózati kártyáé például ez: **00-1C-7E-1D-B8-B5**.

> Mivel ezt a címet a rendszergazdák és az internet szolgáltatók beállításnál és hibakeresésnél gyakran szokták kérni, ezért a laptopok gyártói gyakran ráírják egy matricára a laptop alján. Ha laptopod van, fordítsd meg a gépet, hátha a tiédre is ráírták!
>
> Akár ráírták, akár nem, a gép meg is tudja mondani magáról, hogy mi az ő fizikai címe. Indítsd el a **Start menü**ből a **Parancssor**t (Command Prompt), és írd be, hogy **`ipconfig /all`**! Az elszaladó hosszú listában a **Fizikai cím** (Physical Address) sorban találod a géped egyedi címét.

IP cím

A fizikai címmel több probléma is van, melyek közül a legfontosabb, hogy semmilyen jelentést nem hordoz. Bár kiderül belőle, hogy pontosan melyik cég gyártotta az adott hálózati kártyát, de azt például már nem tudja, hogy földrajzilag hol található a gép, vagy hogy melyek a szomszédos gépek.

Ugorjunk tehát egy magasabb absztrakciós szintre, és a fizikai cím helyett használjunk egy másik címet, ami több információt hordoz, ez pedig az *IP cím* (*Internet Protocol address*). Az IP cím egy 4 bájtból álló cím, aminek a 4 elemét (angolul *octet*jét) ponttal elválasztva szoktuk leírni. Ez például egy nagyon barátságos IP cím: **152.66.188.13**.

> Kíváncsi vagy a sajátodra? Indítsd el a **Start menü**ből a **Parancssor**t (Command Prompt), és írd be, hogy **`ipconfig`**, majd üss Entert! Az **IPv4 cím** (IPv4 Address) sorban találod a géped által használt IP címet.
>
> Ha pedig ki szeretnéd próbálni, hogy a géped „lát-e” egy másik gépet a címe alapján, akkor szintén Parancssorban írd be a **`ping`** parancsot, majd utána a másik gép IP címét!

Az IP címek szintén egyediek a világon, az internet szolgáltatók gondoskodnak róla, hogy a körülbelül 4 milliárd cím közül mindenkinek olyan jusson, amit éppen más nem használ. Mivel az internet szolgáltatók földrajzilag helyhez kötöttek (például egy adott országban lévő ügyfeleknek nyújtanak szolgáltatást), ezért az IP címekből az is kiderül, hogy az IP címet használó számítógép hol található.

Az IP cím mellett még szükség van az ún. *alhálózati maszk* (*subnet mask*) beállításra is. Az alhálózati maszk segítségével egy számítógép, amely adatot akar küldeni egy másik számítógép számára a hálózaton, meg tudja állapítani, hogy a címzettel azonos alhálózaton tartózkodik-e. Ez azért fontos, mert ha egy alhálózaton vannak, akkor közvetlenül kommunikálhatnak egymással, ha viszont nem, akkor a feladó a router felé küldi a csomagot, a router pedig gondoskodik a csomag továbbításáról

más alhálózatok felé. Innen a router másik neve, hívják ugyanis *alapértelmezett átjáró*nak (*default gateway*) is.

Tehát legalább három címet kell beállítanunk minden számítógépen, ha azt szeretnénk, hogy a számítógép adatokat tudjon küldeni és fogadni az interneten:

- Az *IP címet*, ami minden számítógép esetén egyedi cím.
- Az *alhálózati maszkot*, amivel a feladó eldöntheti, hogy a címzett vele azonos alhálózaton van-e.
- Az *alapértelmezett átjáró címét*, ami a routernek az adott alhálózatra kapcsolódó hálózati kártyájának címe.

> **TIPP:** Parancssorban kiadva az `ipconfig` parancsot, az **IPv4 cím** (IPv4 Address), az **Alhálózati Maszk** (Subnet Mask) és az **Alapértelmezett átjáró** (Default Gateway) sorokban látható a három cím. Windows 7 alatt a **Hálózati Kapcsolat Részletek** ablakban tekinthetjük át kényelmesen az összes beállítást.

A kérdés már csak az, hogy honnan tudjuk, pontosan milyen értékeket kell beállítanunk? A kényelem érdekében a legtöbb esetben nem szükséges ennek a három értéknek a kézi beállítása, a számítógép tudja őket „magától". Ez valójában azt jelenti, hogy a hálózaton üzemel egy ún. *DHCP kiszolgáló* (*Dynamic Host Configuration Protocol*), amelytől a számítógép képes lekérdezni a szükséges beállításokat. Ezért fordulhat elő, hogy a legtöbb nyilvános hálózaton elég rácsatlakoznunk a hálózatra, nem kell a beállításokkal bajlódnunk.

> **JÓ TUDNI:** Webfejlesztőként érdemes tudnunk, hogy van egy speciális IP cím, ami mindig a saját gépünket jelöli, ez pedig a **127.0.0.1**. Tehát ha egy böngésző címsorába beírjuk, hogy `http://127.0.0.1`, akkor mindig az ugyanazon a gépen lévő webkiszolgálóhoz fogunk csatlakozni, így akár hálózati kapcsolat nélkül is készíthetünk weboldalakat.

Összefoglalva eddigi tudásunkat, ha a mi gépünknek helyesen beállítjuk a három cím paraméterét és ismerjük a címzett gép IP címét, akkor a két gép tud egymással kommunikálni. Ez a gyakorlatban valóban így működik, ám ha hús-vér felhasználó is ül a gép előtt, akkor a címek használata nem túl praktikus. A gépek jól működnek számokkal, de az emberek inkább neveket tudnak megjegyezni. A következő fejezetben annak járunk utána, hogyan lesznek a nevekből számok.

Nevek az embereknek

Mikor kedvenc böngészőnk címsorába beírunk egy webcímet, például azt, hogy http://devportal.hu, akkor azt a böngészőnek le kell fordítania egy IP címre, hiszen az előző fejezetben láttuk, hogy a gépeknek címekre van szükségük a sikeres kommunikációhoz. Ezt a folyamatot *névfeloldás*nak hívjuk, és a *Domain Name System (DNS)* szolgáltatás a felelős érte.

DNS

A DNS szolgáltatás kétféleképpen tud működni. Dinamikus névfeloldás esetén az előző fejezetben bemutatott három címen kívül a gépünk automatikusan megkapja a DHCP kiszolgálótól a *DNS kiszolgáló* címét is. A DNS kiszolgáló egy olyan szolgáltatás a hálózaton, amelynek a hálózat gépei kéréseket küldenek, és ő visszaadja a beküldött névhez tartozó IP címet (vagy akár fordítva, az IP címhez tartozó nevet). Egyetlen DNS kiszolgáló nem tudja a világhálóra kapcsolt összes számítógép nevét és IP címét, ezért ha ő nem tud válaszolni a kérdésre, akkor továbbítja azt más DNS kiszolgálóknak. Ha egy rendszergazda új gépet köt a hálózatra, és azt szeretné, hogy azt más gépek név szerint is megtalálják, akkor be kell jegyeznie az új gépet a helyi DNS kiszolgáló adatbázisába. Mivel a DNS kiszolgálók kapcsolódnak egymáshoz, egy távoli gépről érkező névfeloldási kérés előbb-

utóbb el fog jutni az adott alhálózatért felelős DNS kiszolgálóhoz, aki így képes lesz megadni a névhez tartozó IP címet. Ennek a módszernek az az előnye, hogy semmit nem kell konfigurálni az egyes számítógépeken, a központi szolgáltatás beállításával az összes kliens számítógép el fogja tudni érni az adatokat.

> **TIPP:** Parancssorban kiadva az **`ipconfig /all`** parancsot, a **DNS kiszolgálók** sorokban láthatóak a számítógép számára beállított névszerverek. Ezeket is be lehet állítani kézzel, de többnyire a DHCP szervertől kapja meg a gépünk a DNS szerverek IP címeit is. Több névszervert is beállíthatunk, de érdemes tudni, hogy először mindig az elsődleges DNS szerverhez fordul a gépünk, a másodlagoshoz csak akkor, ha az elsődleges nem válaszol.

Hosts fájl

Egy másik lehetséges megoldás a statikus névfeloldás. Statikus névfeloldás esetén nincs központi szolgáltatás, hanem minden egyes gépen nyilvántartjuk az elérni kívánt gépek név-IP cím párosait. Ez az adatbázis alapértelmezés szerint a `C:\Windows\System32\drivers\etc\hosts` fájlban található, ami egy egyszerű szöveges állomány, bármilyen szövegszerkesztővel, például Jegyzettömbbel is szerkeszthető. Ennek a megoldásnak az előnye, hogy egy IP címhez tetszőleges nevet vehetünk fel, és azzal a névvel hivatkozhatunk az adott gépre még akkor is, ha az nincs bejegyezve a központi DNS szolgáltatásba.

A gyakorlatban a statikus névfeloldás nem önállóan, hanem a dinamikus névfeloldással együtt szokott működni, tehát a két módszer kiegészíti egymást.

Webfejlesztés közben nagyon jó szolgálatot tehet a `hosts` fájl. Előfordulhat, hogy a gépünkön futó webszerveren több webhelyet is szeretnénk futtatni, ekkor a rendszer például a név alapján különböztetheti meg őket. A `hosts` fájlban felvehetünk több nevet, és mindegyik mutathat a `127.0.0.1` IP címre.

Egy másik gyakori eset, amikor praktikus a `hosts` fájl használata: ha még nincs bejegyezve a hivatalos *tartomány (domain) név* egy internet szolgáltatónál, de az alkalmazásnak mindenképp szüksége van rá. Ebben az esetben ideiglenesen felvehetjük a nevet a saját `hosts` fájlunkba, később pedig amikor megtörténik a hivatalos névbejegyzés, akkor könnyen törölhetjük onnan.

> **TIPP:** A névfeloldás működésének tesztelésére kiválóan használható az **`nslookup`** parancs. Nyissunk egy Parancssort, és írjuk be, hogy **`nslookup devportal.hu`**, majd üssünk Entert. A válaszban látni fogjuk a névfeloldást végző kiszolgáló nevét és a devportal.hu IP címét is. Az **`nslookup`** fordítva is tud működni, azaz ha egy név helyett IP címet írunk be utána, akkor megmondja a címhez tartozó nevet (feltéve, hogy az adott tartomány rendszergazdája nem csak a névhez tartozó IP címet regisztrálta, hanem az IP címhez tartozó nevet is).

Dinamikus DNS

A fenti megoldások mind azt feltételezik, hogy az IP cím, amire a névbejegyzés hivatkozik, állandó. Csakhogy a mai világban az internet szolgáltatók többsége az otthoni felhasználók számára dinamikus IP címet ad, azaz bizonyos időközönként az IP cím megváltozik. Mivel a DNS szerverek erről a címváltozásról automatikusan nem értesülnek, ezért az ilyen dinamikus IP címre mutató névbejegyzések nagyon hamar elavulnának.

Erre a problémára jelent megoldást a *dinamikus DNS szolgáltatás* (nem összekeverendő a *dinamikus névfeloldással*). A dinamikus DNS működése megegyezik a hagyományos DNS szerverek működésével, azzal az egyetlen különbséggel, hogy ha egy számítógép IP címe változik, akkor a számítógép értesíti a DNS szervert, és automatikusan frissíti a saját névbejegyzéséhez tartozó IP címet.

A világon számos ingyenes dinamikus DNS szolgáltató van, a legismertebbek a DynDns.com és a no-ip.com, de létezik több magyar szolgáltató is, például a myip.hu és a dynpro.hu. Ezekről a weboldalakról a regisztráció után egy kliens programot kell letöltenünk, amely a gépünkön folyamatosan figyeli a hálózati kapcsolatot, és ha IP cím változást tapasztal, automatikusan frissíti a DNS szerveren a névbejegyzést. Így megoldható, hogy például az otthoni gépünket, amely akár naponta többször új IP címet kap, név szerint el tudjuk érni távolról.

DNS gyorsítótár

Gyakori eset, hogy egy futó programnak egymás után többször van szüksége egy adott névhez tartozó IP címre. Mivel a névfeloldás időigényes folyamat (el kell mennie a kérésnek a hálózaton keresztül a névszerverig és vissza), ezért a válaszok egy ún. *gyorsítótárba* (*cache*) kerülnek. Ez azt jelenti, hogy a számítógép egy ideig megjegyzi a DNS szervertől kapott választ, és ez alatt az idő alatt ha egy program ismét erre kíváncsi, akkor nem fordul a névkiszolgálóhoz, hanem „fejből” azonnal nyomja a választ.

A választ annyi időre gyorsítótárazhatja a kliens, amennyi időre a DNS kiszolgáló engedi. A válaszban tehát nemcsak a kérdezett névhez tartozó IP címet küldi vissza a névszerver, hanem egy ún. *time-to-live (TTL)* értéket is, ami azt jelzi, hogy mennyi ideig őrizheti a kliens a gyorsítótárban a választ.

A DNS gyorsítótár hasznos szolgáltatás, a webfejlesztőknél azonban néha okoz egy kis galibát. Képzeljük el az alábbi esetet:

1. Fejlesztőként készítünk egy weboldalt, és kitesszük egy szerverre.
2. Beírjuk a kedvenc Internet Explorer böngészőnk címsorába a szerver nevét, de keserűen tapasztaljuk, hogy nem jön be semmi ☹.
3. Kipróbáljuk úgy, hogy a szerverre név helyett IP címmel hivatkozunk, és úgy minden működik. Hmm.
4. Némi gondolkodás után a homlokunkra csapunk, hogy hoppá, elfelejtettük regisztrálni a nevet, ezért irány a hosts fájl vagy a DNS kiszolgáló, ahova felvesszük a szerver nevéhez tartozó IP címet.
5. Visszatérünk a böngészőnkhöz, frissítjük az oldalt, de az eredmény ugyanaz. Fehér oldal szép nagy hibaüzenettel: A kiszolgáló nem található.

Hol a hiba?

Ebbe a szituációba előbb-utóbb minden webfejlesztő belekeveredik, ezért érdemes ismernünk a megoldást: a problémát a DNS gyorsítótár okozza.

Mikor a 2. lépésben a böngészőben megpróbáljuk megnyitni az oldalt, a gépünk a DNS kiszolgálóhoz fordul, és megkísérli elvégezni a névfeloldást. Mivel azonban a DNS kiszolgálón nincs bejegyezve a webkiszolgáló neve, ezért a DNS kiszolgáló azt fogja mondani, hogy *„ilyen nevű szerver nincs”*. Ez a válasz – legyen bármennyire használhatatlan – ugyanúgy bekerül a DNS gyorsítótárba, mint a sikeres válaszok.

JEGYEZZÜK MEG: a negatív válasz is válasz.

Ezek után hiába frissítjük a névszerver adatbázisát, attól még nem fog kikerülni a bejegyzés a számítógép DNS gyorsítótárából. Ha nem akarjuk kivárni, amíg a gyorsítótárból magától kikerül ez a

bejegyzés – ez akár több nap múlva is lehet –, akkor parancssorból kiadhatjuk az `ipconfig /flushdns` parancsot, ami elvégzi az operációs rendszer DNS gyorsítótárának törlését.

Ám ettől még valószínűleg nem fog megjavulni minden, ugyanis a mai böngészők általában maguk is tartalmaznak egy DNS gyorsítótárat. A böngészők gyorsítótárát legegyszerűbben a böngésző újraindításával törölhetjük. Ha ezt a fenti 5. lépés előtt megtesszük, akkor valószínűleg már sikeresen meg fog jelenni az új oldalunk.

Programok egymás között

Az előző fejezetben láttuk, hogy két gép hogyan találja meg egymást IP cím és név alapján. De mi van akkor, ha egy gépen több program is található? Tegyük fel, hogy van egy szerver gépünk, amin fut egy webkiszolgáló és egy FTP kiszolgáló, illetve van egy ide kapcsolódó kliens számítógépünk, amin fut egy webböngésző és egy FTP kliens program is. A két gépen két-két program üzenget egymásnak, hogy lehet, hogy mégsem keverednek össze az üzenetek, hogyan talál be egy üzenet egy gépen belül a megfelelő alkalmazáshoz?

Ezt a problémát oldják meg az ún. *port*ok. A port nem más, mint egy egész szám 0 és 65535 között, ami a gépen belül azonosít egy alkalmazást. Amikor egy program egy másik gépen futó alkalmazásnak kíván üzenetet küldeni, akkor nemcsak a célgép IP címét kell megadnia, hanem a célgépen futó alkalmazás port számát is, így lesz egyértelmű, hogy pontosan melyik programnak kell feldolgoznia az üzenetet.

Vannak nagyon gyakran használt alkalmazások – mint például a webkiszolgáló –, melyeknek a port számát az egyszerűség érdekében szabványosították. Például a webkiszolgáló mindig a 80-as porton figyelve várja a bejövő HTTP kéréseket, a titkosított HTTPS kéréseket pedig a 443-as porton. Ezért aztán nem is szükséges ezeket a port számokat kiírnunk, a böngésző automatikusan tudni fogja, hogy ha a webcím `http://`-vel kezdődik, akkor a kiszolgáló 80-as portjára, ha `https://`-sel, akkor pedig a kiszolgáló 443-as portjára kell küldenie a kérést.

> A portoknak nagy szerepük van a tűzfalak beállításában, melyekkel korlátozhatjuk egy számítógép kimenő és bejövő forgalmát. Például ha a webszerverünk elé beteszünk egy tűzfalat, de azon nem nyitjuk ki a 80-as portot, akkor a webszerver nem lesz elérhető.

Előfordulhat, hogy egy alkalmazásból több példányt is szeretnénk futtatni, vagy például egy webszervert üzemeltetünk a gépünkön, amely több webhelynek ad otthont. Bármennyire is igaz, hogy a `http://` protokollhoz a szabvány szerint a 80-as port tartozik, két program vagy két webhely nem osztozhat ugyanazon a porton, hiszen ha egy böngésző címsorába beírnánk, hogy `http://localhost/default.cshtml`, akkor a webkiszolgáló nem tudná eldönteni, hogy melyik webhelyre gondolunk. Ilyenkor kénytelenek vagyunk a két webhelyet más-más porton üzemeltetni, és a nem alapértelmezett portszámokat a kérésben kiírni. Az egyik webhely kezdőlapja lehet a `http://localhost/index.php`, a másiké pedig például a `http://localhost:8080/default.cshtml`.

> Mikor egy WebMatrixban készülő webhelyet futtatunk, akkor a WebMatrix elindítja az *IIS Expresst*, ami egy kifejezetten fejlesztői célra használható webkiszolgáló. Az IIS Express képes egyszerre több webhelyet is futtatni, mégpedig úgy, hogy mindegyiket önálló port számon teszi elérhetővé. Figyeljük meg a böngésző címsorát, és látni fogjuk a port számot.
>
> A Windows beépített *Internet Information Services* webkiszolgálójával (ami az előbb említett IIS Express nagy testvére) úgy is üzemeltethetünk több webhelyet, hogy mindegyik a HTTP protokollhoz tartozó 80-as porton figyel. Ez akkor lehetséges, ha a webhelyek önálló néven érhetőek el, az egyik például a **`http://forum.example.com`**, a másik pedig a

`http://aruhaz.example.net` címen. Ilyenkor a webkiszolgáló a név alapján képes eldönteni, hogy a bejövő kérés pontosan melyik webhelyre vonatkozik.

A HyperText Transfer Protocol

Avagy mi történik a böngésző színfalai mögött?

A *HyperText Transfer Protocol*, azaz a HTTP a webböngészők által a weboldalak megjelenítésére használt információátviteli protokoll a világhálón, ezért amikor egy webalkalmazást készítünk, akkor a legfontosabb elvárás a kódunkkal szemben, hogy HTTP protokollon keresztül lehessen vele kommunikálni. Bár a legtöbb szerver oldali programozási környezet (pl. ASP.NET) jórészt elfedi a HTTP protokoll részleteit, mindenképp érdemes megismerkednünk a protokoll működésével, mert ez a tudás sokat segíthet a programjaink megtervezésében és a fejlesztés során a hibakeresésben is.

A HTTP aktuális verziója az 1.1 verzió, melyet pontosan az 1999 júniusában kiadott (nem egy mai darab, de működik ☺), 176 oldalas RFC 2616 specifikál. Ebben a fejezetben csak a legfontosabb részeket érintjük, akit további részletek érdekelnek, az eredeti dokumentumból tájékozódhat.

A fejezet végére pontosan ismerni fogjuk a HTTP protokoll működését, ami nagy segítség lesz a weboldalak hibáinak feltárásában.

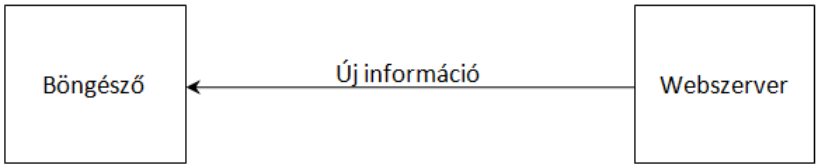
Főbb jellemzők

A HTTP protokollal két szoftver beszélget egymással, egy webes kliens és egy webszerver. A *webes kliens* (angolul *client*, amit néha *ügyfél*nek is fordítanak) legtöbbször a felhasználó által elindított webböngésző program (*webbrowser*), de lehet bármilyen más szoftver is, ami érti a HTTP protokollt (például egy RSS olvasó program), éppen ezért általánosságban a klienseket összefoglaló néven *user agent*nek szokták nevezni. A párbeszéd másik résztvevője a *webszerver* (más néven *webkiszolgáló*), amely a felhasználó által elérni kívánt tartalmat vagy szolgáltatást biztosítja.

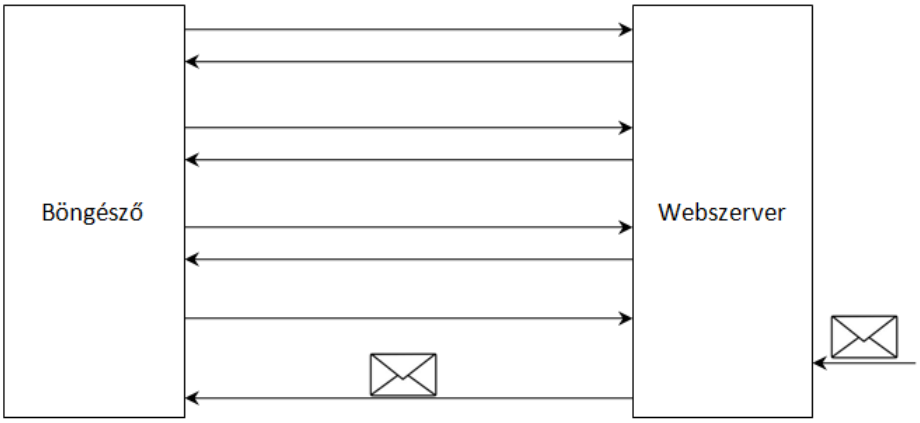
A HTTP legfontosabb jellemzője, hogy egy ún. *kérés-válasz* protokoll a kliens és a szerver között. Ez azt jelenti, hogy a kommunikációt mindig (értsd *MINDIG!*) a kliens kezdeményezi azzal, hogy elküld egy *HTTP kérést* (*request*) a szervernek, aki erre egy *HTTP válasszal* (*response*) válaszol. Fontos, hogy a webszerver nagyon jólnevelt, ha nem kérdezik, nem beszél. A kommunikáció során tehát mindig kérés-válasz párok követik egymást, valahogy így:

6. ábra - A HTTP kérések és válaszok

Ebből az következik, hogy a HTTP protokoll nem biztosít lehetőséget arra, hogy a szerveren megjelenő új információ azonnal megjelenjen a felhasználó böngészőjében. A szerver nem képes arra, hogy elküldjön egy olyan új információt a kliensnek, amit a kliens nem kért. Ilyen tehát nincs:



Érdekes módon felhasználóként mégis nagyon hasonlóval találkozunk, amikor a Facebookot alkalmazva azt látjuk, hogy a mások által írt bejegyzések „maguktól” megjelennek az üzenőfalunkon, vagy amikor a GMail arra figyelmeztet, hogy új levelünk érkezett. Ez csak úgy működhet, hogy miközben mi a weboldalt nézegetjük, a böngésző a háttérben (a felhasználó számára nem látható módon) bizonyos időnként (például percenként) kéréseket küld a szervernek. Ezekre a kérésekre a legtöbb esetben a szerver azt fogja válaszolni, hogy nincs semmi újdonság, amikor azonban a szerveren új információ keletkezik (például érkezik egy levelünk), akkor a következő kérés alkalmával erről a kliens is értesülni is fog. Ezt hívják a szerver *pollozás*ának, a háttérben történő kommunikációt pedig *Ajax*nak.



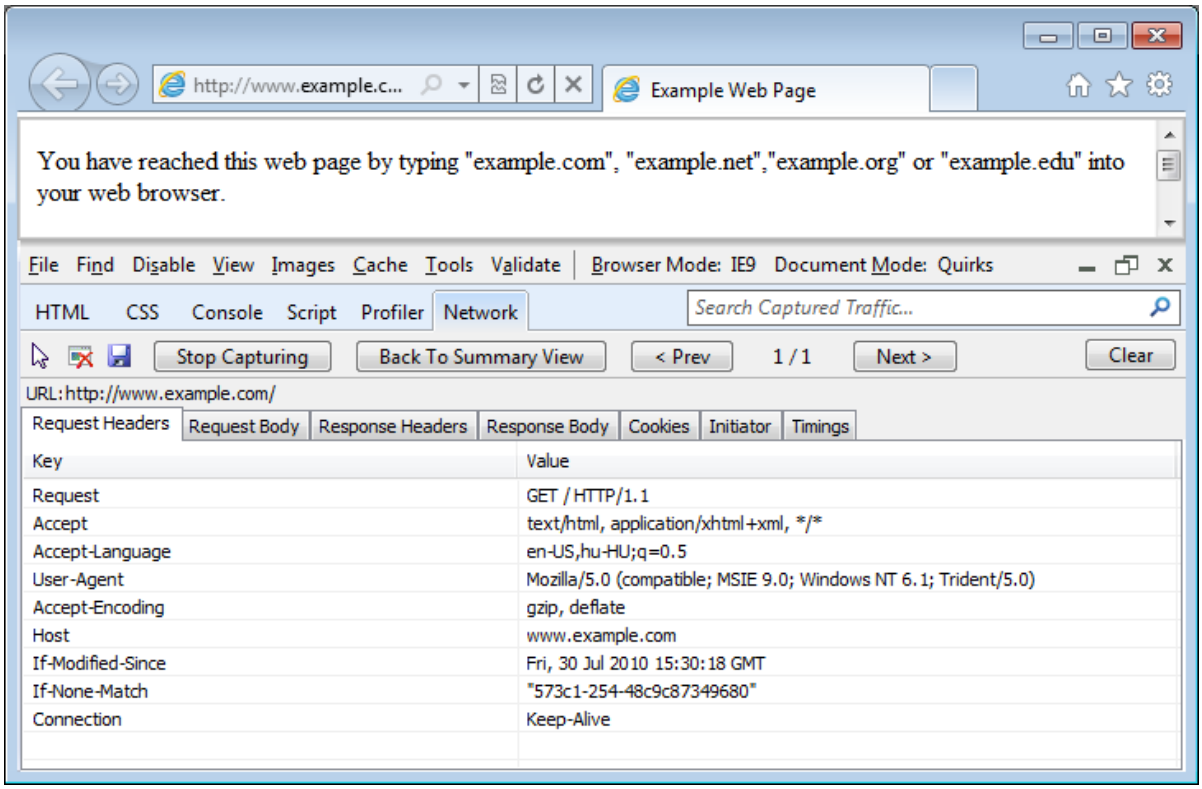
A HTTP másik jellemzője, hogy ún. *állapotmentes* protokoll. Állapotnak tekinthető az, hogy egy webáruházban hány termék van éppen a kosarunkban, vagy hogy egy közösségi oldalra éppen be vagyunk-e jelentkezve. Ezek mind pillanatnyi állapotok, és miközben használjuk a weboldalt, ezek az állapotok megváltozhatnak (például az előbb még üres volt a kosarunk, most már van benne egy karalábé). Mivel a HTTP állapotmentes, ezért ezek az állapotok és azoknak a változása protokoll szinten nem jelenik meg sehol, kár is keresni. Ahogy később látni fogjuk, ezt a problémát máshogy kell megoldanunk.

A kérés és a válasz

Miután így megbeszéltük a HTTP legfontosabb jellemzőit, ássunk egy kicsit mélyebbre, és nézzük meg, hogy pontosan mi történik, miközben a kliens és a szerver beszélget egymással!

A webfejlesztők szerszámosládájának egyik legfontosabb eleme a saját (de tényleg!) HTTP forgalmunk elemzésére szolgáló eszköz. Ezeket a szoftvereket összefoglaló néven *HTTP debugging proxy*nak hívják, és úgy működnek, hogy beállnak a böngésző és a hálózat közé, és a rajtuk átmenő forgalmat megmutatják. Az egyik legkönnyebben használható ilyen eszköz a Fiddler, de nagyon elterjedt a Firefox böngészőbe beépülő Firebug kiegészítés is. Bátran telepítsük fel őket, majd nézzük meg például a www.example.com oldalt kedvenc böngészőnkben, és figyeljük meg, mit mutatnak!

Még egyszerűbb és kényelmesebb a helyzet, ha Internet Explorer 9-et használunk, ott ugyanis a beépített **Fejlesztői Eszköztár** (*Developer Toolbar*, F12) **Hálózat** (*Network*) fülén azonnal látszik a HTTP kérés és a válasz minden részlete, nincs szükség külső eszköz telepítésére:



7. ábra – Az Internet Explorer 9 Developer Toolbar Network fülén a forgalom minden részlete látható

Miközben ellátogatunk egy oldalra, a böngésző az alábbihoz hasonló HTTP kérést küldi el a szervernek:

```
GET http://www.example.com/ HTTP/1.1
Accept: */*
Accept-Language: en-US,hu-HU;q=0.5
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64;
          Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET
          CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E; MS-RTC
          LM 8)
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
Host: www.example.com
Pragma: no-cache
```

Ahogy látható, a HTTP egy szöveges protokoll, így a kommunikáció nagyon könnyen értelmezhető. A kérés első része az ún. *metódus* (*method*), amivel azt mondjuk meg, hogy milyen műveletet szeretnénk elvégezni. Ez leggyakrabban a letöltéshez használt `GET`, vagy a feltöltéshez használt `POST`.

A metódust követi a kért erőforrás webcíme (itt: `http://www.example.com/`), majd a protokoll verziószáma, ami szinte mindig `HTTP/1.1`.

A következő sorok ún. fejléc sorok, amellyel a böngésző további információkat juttathat el a szerverhez. A fenti részletben például egy `Accept-Language` fejlécben tudatja a böngésző a szerverrel, hogy elsősorban angol (`en-US`), másodsorban magyar (`hu-HU`) nyelvű tartalmat szeretne kapni.

A fejléc sorokat egy üres sor követi, majd utána következhet a kérés törzse (*body*). Ilyet főként HTTP `POST` kérésekben lehet látni (a fenti példában tehát nem), például egy űrlapba beírt szövegek a kérésnek ebben a részében jutnak el a szerverhez.

A szerver erre a kérésre egy HTTP választ küld vissza:

```
HTTP/1.1 200 OK
```

```
Date: Fri, 01 Oct 2010 10:42:07 GMT
Server: Apache
Last-Modified: Fri, 30 Jul 2010 15:30:18 GMT
ETag: "573c1-254-48c9c87349680"
Accept-Ranges: bytes
Content-Length: 596
Connection: close
Content-Type: text/html; charset=UTF-8

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
(…még folytatódik…)
```

A válasz első sora a státusz sor, amiben a szerver által támogatott verziószám (`HTTP/1.1`) után egy státusz kód (`200`) és egy angol nyelvű indoklás (`OK`) található. A kéréshez hasonlóan ezután a fejléc, majd egy üres sor után a válasz törzse következik. A válasz törzsében kapjuk vissza a kért weboldalt, képet vagy videót, amit le akartunk tölteni.

Ezek tehát a kérés és a válasz általános formái, a következő fejezetekben egy kicsit részletesebben is megnézzük, hogy az egyes részek mire jók, milyen értékekkel találkozhatunk ott.

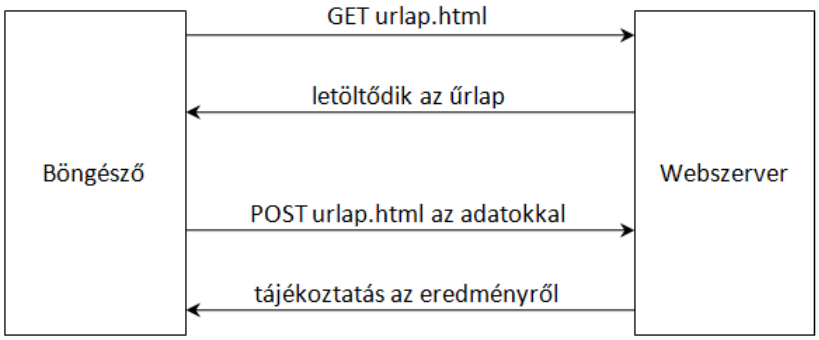
GET és POST

Az előző fejezetben láttuk, hogy a HTTP kérés legelső eleme a metódus, amivel meghatározzuk, hogy milyen műveletet akarunk elvégezni a szerveren.

A leggyakoribb metódus a `GET`, amivel egy oldalt, képet, fájlt, vagy általános néven *erőforrást* tudunk lekérni a szerverről. Egy másik nagyon gyakori metódus a `POST`, amivel feltölteni tudunk; például egy űrlap kitöltése után megadott értékek legtöbbször HTTP `POST` kéréssel utaznak fel a szerverre a kérés törzsében.

> Egy webalkalmazás készítése során néha döntés elé kerülünk, hogy egy adott funkciót egy GET vagy egy POST művelettel valósítsunk meg. Például egy webes levelező oldalon a nem kívánt levelek törlése elviekben bármelyik módon megoldható, melyiket válasszuk? Hasonló probléma jön akkor is elő, amikor a felhasználói felületet tervezzük. A Törlés funkciót egy hiperlink vagy egy gomb (button) valósítsa meg? A hiperlink ugyanis alapértelmezés szerint egy HTTP GET, míg a gomb egy HTTP POST kérést küld a szervernek.
>
> Ha technikailag bármelyiket választhatjuk, akkor célszerű aszerint dönteni, hogy a GET kérések általában biztonságosak, azaz nem változtatják meg a szerver állapotát, és többszöri végrehajtásuk ugyanazt az eredményt váltja ki, mint az egyszeri. Például egy kerti törpe katalógust többször is letölthetek a szerverről, vagy a Bingben is lefuttathatom ugyanazt a keresést többször, nem fáj senkinek. Adatok lehívására tehát kiváló a GET. Ha azonban adatok felküldéséről vagy állapotváltozásról van szó, amik ráadásul többször végrehajtva mást eredményeznek, akkor inkább használjunk POST-ot. Egy levél letörlése a Hotmailen megváltoztatja az állapotot, egy banki átutalás pedig többször megismételve a kívánt mértéknél jobban apasztja a bankszámlánkat, ezért ezekben az esetekben inkább POST javasolt.

Egy webes űrlap (például kérdőív) kitöltése során több HTTP kérésre van szükség, amik között előfordulnak `GET` és `POST` kérések is:



8. ábra - Egy webes űrlap kitöltésének folyamata

A kért erőforrás címe

A kérés második része a kért erőforrás (oldal, kép, dokumentum stb.) pontos címe. A szabvány szerint itt egy egységes erőforrás azonosítót, ún. URI-t (*Uniform Resource Identifier*) kell megadnunk, ami formailag így néz ki:

```
[séma]:[séma függő rész]
```

Először tehát meg kell adnunk egy sémát, majd a kettőspont után azt a szöveget, amit a séma ismeretében tudunk értelmezni. Lássunk rá példát! Ez itt vajon micsoda?

```
4633714
```

Csak egy szép nagy szám, de akárhogy nézem, nem derül ki belőle, hogy mit kell kezdeni vele, és talán már az sem emlékszik rá, aki ideírta. Lehet egy hangya cipőmérete, de akár három fej kelkáposzta ára is (persze más mértékegységben). De ha így írom le, mindjárt értelmet nyer:

```
tel:4633714
```

A `tel` séma egyértelműen megmondja, hogy ez egy telefonszám, erről pedig tudjuk, hogy mit kell csinálnunk vele: be kell nyomkodnunk a telefonunkba. A kettőspont *előtti* rész tehát megmondja, hogy mire jó a kettőspont *utáni* rész.

Ezenkívül az URI még egy dolgot ír elő, mégpedig azt, hogy ez *egyértelműen* azonosítson valamit. A fenti telefonszám egyértelműen azonosít egy készüléket, ami csörögni fog, ha felhívjuk. Itt egy másik URI példa, ahol a sémából az derül ki, hogy ez egy e-mail cím, azaz egyértelműen azonosít egy postaládát:

```
mailto:gipszjakab@example.com
```

Ez alapján már értjük az alábbi URI-t is, ami azt mondja, hogy ez egy a HTTP protokoll számára értelmezhető cím:

```
http://balassygyorgy.wordpress.com/2010/05/21/facebook-like-button-xss
```

Csakhogy ennek a címnek van még egy érdekessége is: nemcsak hogy egyedileg azonosít egy weboldalt, de azt is meghatározza, hogy *hol található* ez az oldal az interneten. Konkrétan a `balassygyorgy.wordpress.com` szerveren a `2010/05/21` mappában, és úgy hívják, hogy `facebook-like-button-xss`. Ezért tudja megtalálni a böngésző, mert tudja, hogy a gigantikus méretű interneten *hol* kell keresnie. Mivel itt nemcsak azonosításról, hanem a dokumentum lelőhelyének meghatározásáról is szó van, ezért az URI elnevezés helyett gyakran használjuk az *URL*, azaz *Uniform Resource Locator* elnevezést.

Az URL általános formája a következő:

```
protocol://username:password@FQDN:port/path/file?
          variable1=value1&variable2=value2#name
```

Elöl van tehát a protokoll, amit opcionálisan egy felhasználónév és egy jelszó követhet, amit egy kukac (`@`) karakter választ el az FQDN-től. Az *FQDN* a *Fully Qualified Domain Name*, azaz a tartomány teljes neve, például **balassygyorgy.wordpress.com**. Az FQDN után jöhet a port száma a szerveren, amit többnyire elhagyunk, mert a böngésző tudja, hogy a HTTP protokoll a 80-as portra, a HTTPS protokoll a 443-as portra kapcsolódik. A portszám után jöhetnek mappák és almappák (*path*), majd végül a lekérni kívánt fájl neve. A kérdőjel (`?`) után következő ún. *query string*ben további információkat adhatunk át a szervernek, amit a szerver tetszőleges módon dolgozhat fel. Ezek a paraméterek `név=érték` formában, egymástól `&` jellel elválasztva adhatók meg. Az URL legvégén a `#` karakter után egy név következhet, ami az oldalon belüli hivatkozásra utalhat.

> Egy webalkalmazás készítése során gyakran előforduló feladat, hogy egy URL-t kell darabjaira szednünk vagy éppen két URL egyezőségét kell vizsgálnunk. Fontos tudni, hogy bár a fenti az URL-ek „szép” általános formája van, sajnos egy URL lehet teljesen „álcázott” is, mert a szabvány megengedi a karakterek kódolását. Például ez is egy érvényes webcím, a böngésző megérti, ha beírjuk neki:
>
> %77%77%77%2E%67%6F%6F%67%6C%65%2E%63%6F%6D
>
> Éppen ezért egy URL feldolgozását manuálisan, hagyományos string műveletekkel megoldani nagyon kockázatos, mert könnyen hibázhatunk. A legtöbb webalkalmazás programozói keretrendszer erre biztosít beépített lehetőséget, például ASP.NET platformon a **`System.Uri`** osztály segítségével egy csapásra megkaphatjuk az URL bármely részét.

Az URL-ek két nagyobb csoportba sorolhatóak. Az *abszolút URL*-ek nem függnek semmitől, önmagukban egyértelműen meghatározzák a kért erőforrást. Például ez a cím mindig az *MSDN Kompetencia Központ* honlapján található *ASP.NET Induló Készlet* oldalára mutat:

```
http://www.msdnkk.hu/Pages/InduloKeszlet/ASPNET/Default.aspx
```

Ez a kezdőlap (a `Default.aspx` oldal) tartalmaz egy képet, amire a hivatkozás így történik:

```
ASPNET_Indulo_Keszlet.jpg
```

Ez a webcím önmagában nem teljes, csak a kezdőlap címéhez viszonyítva értelmezhető, ezért ezt *relatív URL*-nek hívjuk. Az ebből képzett abszolút URL ez lesz:

```
http://www.msdnkk.hu/Pages/InduloKeszlet/ASPNET/ASPNET_Indulo_Keszlet.jpg
```

Ha éppen egy másik oldalon lennénk, és ott hivatkoznánk ugyanezzel a relatív URL-lel erre a képre, akkor már valószínűleg rossz lenne a hivatkozás, és a szerver nem találná meg a fájlt. Részben ezért szoktak ún. *gyökér relatív cím*eket (*root-relative URL*) használni. Például az alábbi egy ilyen URL:

```
/Storage/balassy/Avatar/avatar.jpg
```

A sima relatív URL-ekhez képest az a nagy különbség, hogy egy „/” jellel kezdődik, ez jelzi a böngészőnek, hogy ne az aktuális oldal mappájához csapja hozzá ezt a címet, hanem a szerver neve után illessze oda. Ez lesz az abszolút cím:

```
http://www.msdnkk.hu/Storage/balassy/Avatar/avatar.jpg
```

Általában jobban szeretünk relatív címeket használni, mert rövidebbek, olvashatóbbak, és ha a weboldalunkat másik szerverre költöztetjük, akkor nem kell a szerver nevét mindenhol átírnunk.

Fejléc mezők

Ahogy a HTTP kérés és válasz általános formájának bemutatásánál láttuk, a böngésző és a szerver is küld fejléc sorokat, melyekben a kért tartalmon kívül további információk utaznak. A fejléc sorok

formailag azonosak, először az adott fejléc mező neve áll, amit egy kettőspont után a hozzá tartozó érték követ.

A következőkben a teljesség igénye nélkül álljon itt néhány érdekesebb fejléc mező jelentése!

Az `Accept` fejléccel a kliens határozhatja meg, hogy milyen típusú fájlt szeretne letölteni, amire mintegy válaszul a szerver a `Content-Type` fejlécben adja meg, hogy milyen típusú a fájl, amit visszaküld. Szintén megadja hozzá a szerver, hogy a letölteni kívánt fájl mekkora (`Content-Length` fejléc) és hogy ez a szám miben értendő (`Accept-Ranges`).

A böngésző a kommunikáció során elég sokat elárul magáról, például a `User-Agent` mezőben elküldi a saját típusát, verziószámát és még az operációs rendszer verziószámát is, ami alapján gyönyörű statisztikák készíthetőek szerver oldalon. Szintén elküldi az `Accept-Language` mezőben, hogy a felhasználó hogyan állította be a nyelvi opciókat a böngészőjében, így ez alapján a szerver eldöntheti, hogy milyen nyelvű tartalmat adjon vissza. Sőt, ha egy oldalra úgy jutottunk el, hogy egy másik oldalon kattintottunk egy linkre, akkor a böngésző a hivatkozó oldal címét egy `Referer` (igen, a szabvány szerint így kell írni) nevű fejléc mezőben elárulja a hivatkozott oldalnak.

A szerver is tud bőbeszédű lenni, például a `Server` sorban megmondja az ő típusát, a `Date` sorban pedig a szerver pillanatnyi dátumát.

Biztos tapasztaltuk már, hogy egy weboldal első alkalommal kicsit lassabban töltődik be, mint a későbbi alkalmakkor. Ez gyakran azért van, mert az oldal egyes részeit (például a képeket) a böngésző a saját gyorsítótárában eltárolja, így azokat legközelebb nem kell letöltenie. Hogy mit és mennyi ideig tárolhat, azt a szerver határozza meg a `Cache-Control` és az `Expires` fejléc mezőkkel, sőt ilyen esetekben azt is elküldi, hogy az egyes fájlokat mikor módosították utoljára a szerveren (`Last-Modified` fejléc). Ezt felhasználva a következő alkalommal a böngésző egy `If-Modified-Since` fejléc mezővel tudja jelezni a szerver felé, hogy csak akkor kéri az adott fájlt, ha az a megadott idő után változott. Ha azt tapasztaljuk, hogy bár a webszerveren módosítottunk valamit, de az nem jelenik meg a böngészőben, akkor vizsgáljuk meg a HTTP forgalmat, és figyeljünk ezekre a fejléc mezőkre!

> **JÓ TUDNI:** A böngészők próbálják minimalizálni a hálózati forgalmat – ezzel is gyorsítva az oldal megjelenítését –, ezért a saját gyorsítótárukban igyekeznek eltárolni a szervertől kapott válaszokat (vagy legalább egyes részeit, például a képeket). Ha teljesen frissíteni szeretnénk az oldalt a böngészőben, akkor ne egyszerűen az `F5` gombot nyomjuk meg, hanem a `CTRL+F5` billentyűket, vagy a `CTRL` lenyomása közben kattintsunk a *Frissítés* (*Refresh*) gombra!

Státusz kódok

A HTTP válasz legfontosabb részei kétségkívül a státusz kód és a hozzá tartozó szöveges indoklás. A hibakódok háromjegyű számok, melyek az első számjegy alapján csoportokba sorolhatóak. Íme a leggyakoribb hibakódok:

- `1xx: Information` – A HTTP kapcsolattal kapcsolatos tájékoztató üzenetek, a gyakorlatban nagyon ritkán találkozhatunk velük.
- `2xx: Successful` – Sikeres kérés.
  - `200 OK` – Ez a legjobb és szerencsére leggyakoribb eset.
- `3xx: Redirect` – Átirányítás, a kért tartalom máshol található, amit a `Location` fejléc mezőben ad meg a szerver.
  - `301 Moved permanently` – A kért tartalom végérvényesen átköltözött egy másik címre, a böngésző akár meg is jegyezheti az új címet.

    - `302 Found (temporary move)` – A kért tartalom ideiglenesen máshol található.
    - `304 Not modified` – A kért tartalom az utolsó letöltés óta nem változott. Ezt válaszolhatja a szerver, ha a kliens az `If-Modified-Since` fejléc mezővel jelzi, hogy mi a nála tárolt utolsó változat. Ekkor a kért fájl tartalma nem fog ismét letöltődni a szerverről.
- `4xx: Client Error` – Hiba történt, ami valószínűleg a kliens hibája.
    - `401 Unauthorized` – A kliens olyan oldalt próbál elérni, amihez nincs joga. Ennek az üzenetnek a hatására a böngésző fel fogja dobni a felhasználónév és a jelszó bekérésére szolgáló ablakot.
    - `403 Forbidden` – A kliensnek nincs jogosultsága a kért tartalom elérésére.
    - `404 Not found` – Hibás a cím, a megadott címen nem található tartalom. Célszerű ellenőrizni, hogy a kért címen valóban található fájl a szerveren, vagy elgépeltük az URL-t.
- `5xx: Server Error` – Hiba történt a szerveren.
    - `500 Internal server error` – A szerveren futó webalkalmazásban belső hiba történt. Ezt általában a programozó vagy az üzemeltető tudja kijavítani.
    - `503 Service unavailable` – A szerveren leállt a szolgáltatás, ami vagy a túl sok hiba (ld. 500-as hiba), vagy a túl nagy forgalom (túlterhelés) eredménye.

Ha egy weboldal készítésekor azt tapasztaljuk, hogy valami nem egészen úgy történik, ahogy terveztük, akkor célszerű megnézni a HTTP forgalmat és megkeresni a hibakódot. Ha például egy oldalon nem jelenik meg egy kép, akkor az valószínűleg annak köszönhető, hogy elírtuk a kép URL-jét, amire a szerver 404-es hibával fog válaszolni. Ettől még az oldal meg fog jelenni, a böngésző nem fog hibát jelezni, de ez a kép hiányozni fog. A HTTP válaszban lévő hibakódból egyértelműen kiderül, hogy miért.

Biztonságos kommunikáció

Ahogy az előző fejezetben láttuk, a HTTP protokoll egy nyílt szöveges protokoll, ennek köszönhető, hogy könnyen tudtuk értelmezni a kommunikációt, ami a gyakorlatban nagyban segíti a fejlesztő munkáját. Sajnos ugyanez a nyílt kommunikáció a rosszindulatú támadók munkáját is ugyanígy segíti, ugyanis lehetővé teszi a kommunikáció egyszerű lehallgatását.

Ha illetéktelenek elől védeni szeretnénk a HTTP forgalmat, akkor a HTTP protokoll mellett egy másik protokollt is be kell vetnünk. Ha a HTTP-vel együtt használjuk a *Secure Sockets Layer (SSL)* protokollt is, akkor az eredmény a *HTTPS* protokoll, ami az alábbi két előnnyel bír:

- Titkosítja a forgalmat, így az átküldött információt nem lehet lehallgatni.
- Azonosítja a szervert, így a felhasználó számára egyértelmű, hogy kinek küldi el például a bankkártyája adatait.

A nagy kérdés persze az, hogy amikor felhasználóként megnyitunk egy weboldalt, ami például X banknak mondja magát, akkor honnan tudhatjuk, hogy az valóban *annak* a banknak az oldala? Csak azért, mert az oldal annak mondja magát, nyilván nem. (Ennyi erővel bárki mondhatná magáról, hogy ő személyesen a Batman.) Az URL alapján? Az sem jó, hiszen az adott bank tetszőleges webcímet választhat magának. Akkor mi a megoldás?

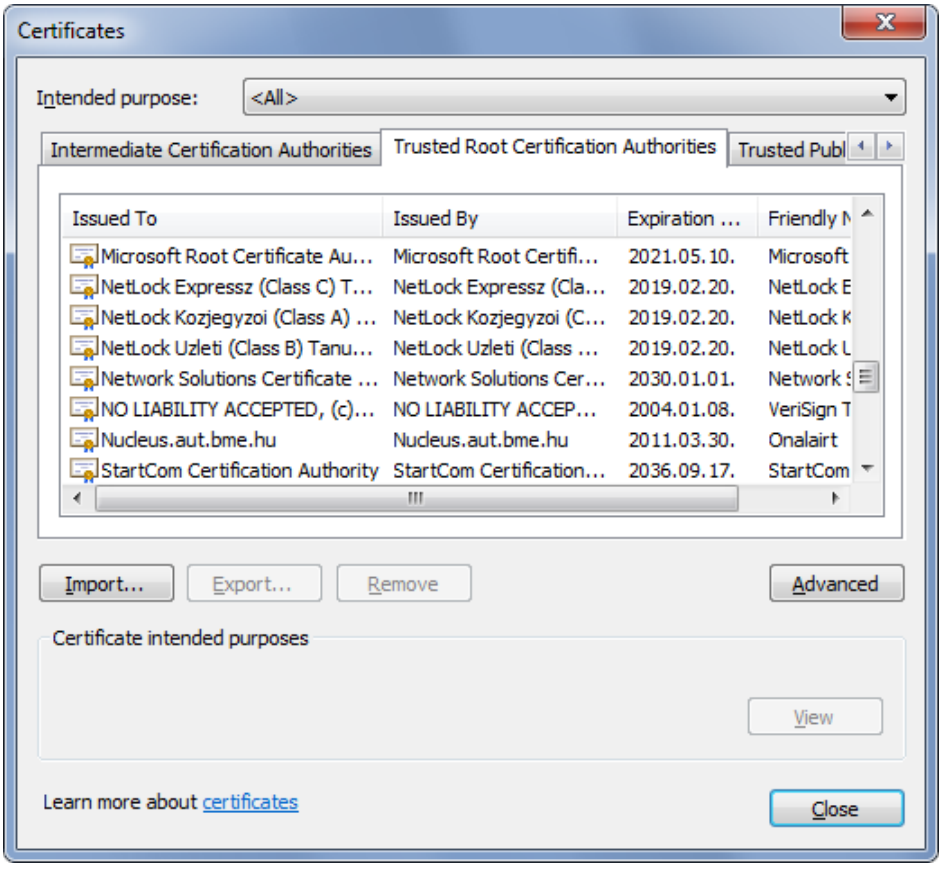
A megoldás az, hogy választunk egy megbízható harmadik felet, akiben a kommunikáció mindkét szereplője megbízik, a szerver is és a kliens is. Ha ez a harmadik fél azt állítja, hogy az adott weboldal tényleg annak a banknak a honlapja, akkor a felhasználó ezt elhiheti.

Ez a gyakorlatban úgy történik, hogy az adott cég vásárol egy tartomány nevet (pl. `xbank.example.com`), majd a céget és a tartományt igazoló dokumentumokkal elmegy egy ilyen megbízható harmadik félhez, *a tanúsítvány kiadóhoz (certification authority)*. A tanúsítvány kiadó a lehető legalaposabban ellenőrzi a dokumentumokat, majd kiállít egy *tanúsítványt* (*certificate*), amely igazolja, hogy a tanúsítvány tulajdonosa valóban az, akinek mondja magát. A cégnek ezek után fel kell telepítenie ezt a tanúsítványt a webszerverre, és innen kezdve megy minden magától.

A böngésző, amikor `https://` kezdetű URL-t lát, akkor automatikusan HTTP helyett HTTPS protokollt fog használni, és a szerver 80-as portja helyett a HTTPS protokollnak megfelelő 443-as portjára fog csatlakozni. Lekéri a szerver tanúsítványát, és ellenőrzi a benne szereplő adatokat:

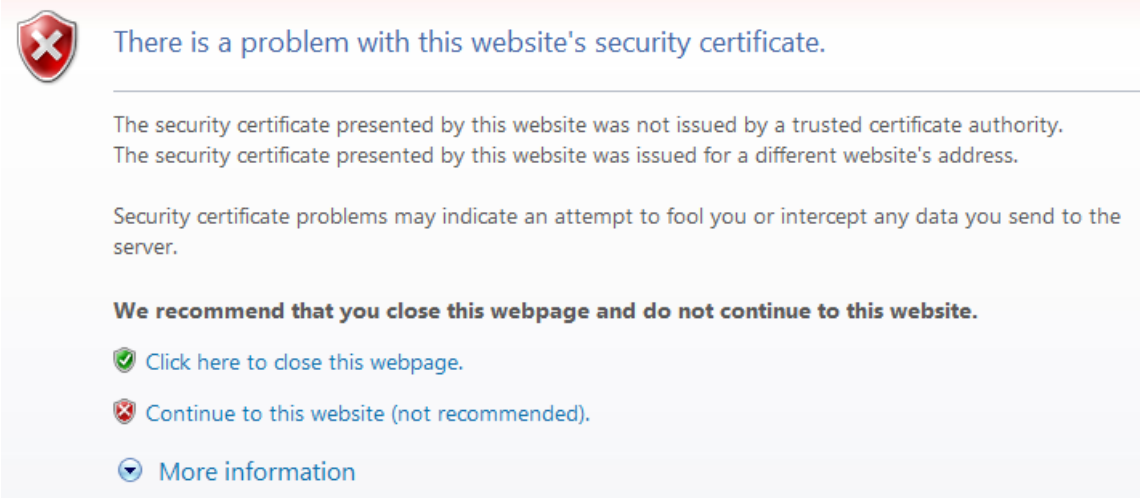
1. Ellenőrzi, hogy tényleg azt a webcímet nézzük-e éppen, amire a tanúsítványt kiállították. Nem mindegy, hogy az `example.com` vagy a `www.example.com` oldalról van-e szó.
2. Ellenőrzi, hogy a tanúsítvány nem járt-e le (általában 1 évre szól).
3. Ellenőrzi, hogy a tanúsítványt nem vonták-e vissza. Ha egy tanúsítvány rossz kezekbe kerül, akkor a tanúsítvány kiadója visszavonhatja azt.
4. Ellenőrzi, hogy a tanúsítványt olyan tanúsítvány kiadó állította ki, akiben a böngésző megbízik.

A megbízható tanúsítványkiadók listája az Internet Explorer *Internet Options* ablakának *Content* fülén, a *Certificates* gombra kattintva, a felugró ablak *Trusted Root Certification Authorities* oldalán tekinthető meg:



Az *Import* gombra kattintva lehet olyan tanúsítványkiadókat hozzáadni a rendszerhez, melyeket a böngésző beépítetten nem ismer.

Ha ezek közül bármelyik nem teljesül, akkor a tanúsítványt érvénytelennek tekinti a böngésző, és figyelmeztetést fog megjeleníteni:



9. ábra - Figyelmeztetés az Internet Explorerben érvénytelen tanúsítvány esetén

Az érvénytelen tanúsítvány azt jelenti, hogy a felhasználó nem tudhatja biztosan, hogy a webszerver valóban az-e, akinek mondja magát.

A tanúsítvány másik felhasználása, hogy a benne szereplő információk felhasználásával a kommunikáció két szereplője titkosítani tudja a forgalmat. Ha egy tanúsítvány nem érvényes, attól még a forgalom titkosítására lehet alkalmas.

A tanúsítványt egy *hiteles* tanúsítvány kiadótól kell vásárolni, ha azt szeretnénk, hogy a böngésző megbízhatónak tekintse. Ha ennek a költségeit nem szeretnénk kifizetni, akkor van lehetőség a tanúsítvány házi elkészítésére is, az ilyet hívják *önaláírt tanúsítvány*nak (*self-signed certificate*). Ebben az esetben a fenti ellenőrző lista 4. pontja nem fog teljesülni, így a böngésző a tanúsítványt érvénytelennek fogja tekinteni, és a felhasználó sem tudhatja biztosan, hogy kinek a szerverével kommunikál, kivel osztja meg a bizalmas adatait.

Bár a kommunikáció ebben az esetben is lehet titkosított, önaláírt tanúsítvány esetén a rosszindulatú támadó nagyon könnyen kijátszhatja a felhasználók figyelmetlenségét, és belehallgathat a hálózati forgalomba, ezért önaláírt tanúsítvány helyett *mindig célszerű* hiteles kiadótól származó tanúsítványt vásárolni.

Állapotkezelés

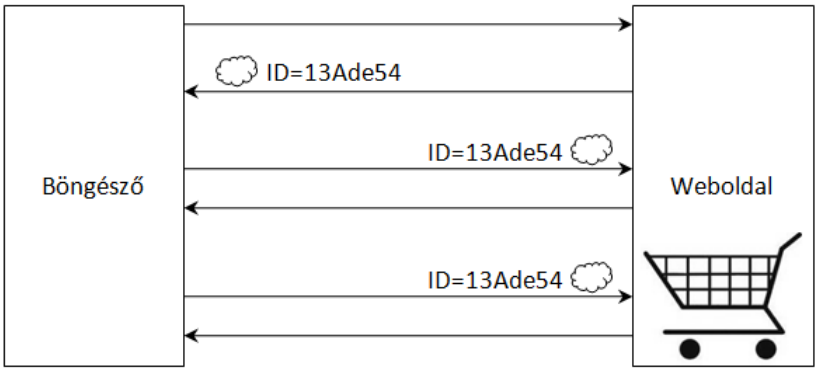
Ahogy arról korábban már szó volt, a HTTP állapotmentes protokoll, ami azt jelenti, hogy az egyes kérés-válasz párok között nincs kapcsolat. Ez valószínűleg nem is okozott különösebb gondot a web megjelenésekor, de mára már elengedhetetlenné vált, hogy a webáruházakban meg tudjuk tömni a kosarunkat, vagy hogy be tudjunk jelentkezni a kedvenc közösségi oldalunkra. Azaz a webalkalmazásokban szükségessé vált a *felhasználói munkamenet* (angolul *session*) követése.

A megoldást *Lou Montulli* találta ki a *sütik*, azaz angolul a *HTTP cookie*-k formájában.

Montullitól azóta sokszor megkérdezték, hogy miért pont a *cookie* nevet adta ennek a technológiának, amire ő azt szokta válaszolni, hogy nincs ebben semmi különös, a program

által kapott és változatlanul visszaküldött információdarabkára akkoriban elterjedt elnevezés volt a *magic cookie*.

A *cookie*-k olyan információdarabkák, amiket az első kérésre válaszolva a szerver elküld a böngészőnek, a böngésző pedig minden egyes további HTTP kérésben visszaküldi a szervernek. Az átküldött információ bármi lehet, de mivel a sütik mérete korlátozott, ezért az a bevett megoldás, hogy a webalkalmazás szerver oldalon tárolja az összes felhasználó munkamenetéhez tartozó adatait, és a sütiben csak egy *munkamenet azonosító* (*session ID*) utazik.



10. ábra - A munkamenet azonosító a HTTP kérésekben és válaszokban

A gyakorlatban a cookie HTTP fejléc sorokban utazik, először a szerver egy `Set-Cookie` nevű fejléc mezőben állítja be a cookie értékét:

```
HTTP/1.1 200 OK
Set-Cookie: lsd=2EDQr; path=/; domain=.facebook.com
```

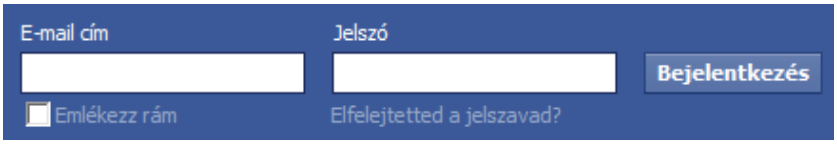
A következő kéréseknél pedig a böngésző egy `Cookie` nevű fejléc mezőben visszaküldi azt a szervernek:

```
GET http://www.facebook.com/home.php? HTTP/1.1
Cookie: lsd=2EDQr;
```

Fontos, hogy a böngészőknek a szabvány szerint úgy kell kezelniük a sütiket, hogy csak azoknak a weboldalaknak küldjék vissza őket, ahonnan kapták, így nem lehetséges az, hogy az egyik webhelyről kapott sütink eljut egy másik webhelyhez.

Kétféle süti létezik:

- A *session cookie* csak a munkamenet idejére létezik, és a böngésző memóriájában tárolódik. Ha a munkamenetünk véget ér vagy a böngészőt bezárjuk, a süti elveszik.
- Az állandó sütit (angolul *permanent* vagy *persistent cookie*) viszont a böngésző a diszkre menti, így meglesz akkor is, amikor újraindítjuk a böngészőt vagy akár az egész gépünket. Amikor egy weboldalra bejelentkezve bejelöljük az *Emlékezz Rám* (jegyezd meg a jelszavam) opciót, akkor tipikusan ilyen állandó süti keletkezik. Az operációs rendszerben szerencsére minden böngészőnek és minden felhasználónak önálló süti tára van, így nem fordulhat elő, hogy a mi gépünkre bejelentkezve egy másik felhasználó a mi munkamenetünket folytatja.



11. ábra - A bejelentkezés és az Emlékezz Rám opció a Facebookon

A süti egy egyszerű szöveges állomány, ahogy az a fenti példából is látszik. Önmagában nincs benne semmi veszélyes, ezért rögtön el is oszlathatjuk azokat a tévhiteket, miszerint a sütik vírusokat, felugró ablakokat, kéretlen leveleket vagy reklámokat terjesztenének – a cookie-k erre nem képesek. Az viszont igaz, hogy egy süti tartalmazhat személyes adatot vagy egy sütinek pusztán a létezése a gépünkön utalhat arra, hogy meglátogattunk egy weboldalt. Ha nem szeretnénk, hogy a süti a hálózaton átutazva illetéktelen kezekbe kerüljön, akkor mindenképpen használjunk biztonságos HTTPS protokollt!

> A sütikkel kapcsolatos összes kérdésünkre választ kaphatunk az Unofficial Cookie FAQ oldalán.

A HyperText Markup Language

Avagy milyen nyelven értenek a böngészők?

Miután megfejtettük, hogyan kommunikálnak egymással a számítógépeink az interneten keresztül, és megértettük a kommunikáció során használt HTTP protokoll minden részletét, a következő kérdés az lehet, hogy mit kell visszaküldeni a webszervernek a böngésző számára ahhoz, hogy a felhasználó számára az eredmény egy weboldal legyen? A HTTP válasz törzsében a webszervernek egy *HyperText Markup Language* (az elnevezés magyarul *hiperszöveges jelölőnyelvet* jelent), azaz HTML formátumú dokumentumot kell visszaküldenie, ez ugyanis a böngészők nyelve.

A HTML gyökerei a nyolcvanas évek végére nyúlnak vissza, amikor a svájci CERN[1]-ben dolgozó fizikus, *Tim Berners-Lee* azzal a problémával szembesült, hogy a korabeli dokumentum formátumok nem megfelelőek a fizikusok által előállított kutatási eredmények megjelenítésére. Az egyik fájó pont a képek, ábrák, illusztrációk és általánosságban a gazdag formázási lehetőségek hiánya volt, a másik pedig az, hogy a kutatási eredmények általában egymásra épülnek, kapcsolódnak egymáshoz, amit az akkori fájlformátumok nem tudtak kezelni. Ezen problémák megoldására Tim Berners-Lee két ötlettel állt elő, melyek a mai napig meghatározzák a HTML nyelvet:

1. A dokumentum ne csak egyszerű szöveg legyen, hanem a tartalmat hordozó részt lássuk el címkékkel, melyek kiegészítő információkat (*metaadatokat*) adnak a szöveghez. Hasonló célú egyszerűbb jelöléseket a nyomdatechnikában már korábban is használtak (pl. egyszeres aláhúzás: dőlt, kettős aláhúzás: félkövér betű), itt a nagy újdonság a formalizmus és címkekészlet megalkotása volt, így jött létre a HTML mint *jelölőnyelv* (*markup language*).
2. A dokumentumok „mutassanak túl” önmagukon, azaz olyan *hiperszöveg*eket (*hypertext*) tartalmazzanak, amelyek bármely részlete egy másik dokumentumra hivatkozhat. Ez az, amit ma úgy ismerünk, hogy egy weboldalon rákattinthatunk hivatkozásokra (linkekre), aminek hatására betöltődik egy másik weboldal.

E két fő irányvonal mentén született meg a HTML nyelv első leírása 1991-ben, melynek ma a továbbfejlesztett és szabványosított 4.01-es, hamarosan pedig 5-ös verzióját használjuk.

A HTML nyelv elemei

A HTML nyelv jelölőrendszere elemekből és attribútumokból épül fel. Az elemekkel tudjuk felcímkézni a szöveg egyes részeit, az attribútumokkal pedig ezeknek a címkéknek a tulajdonságait tudjuk meghatározni.

---

[1] A CERN, a *Conseil Européenne pour la Recherche Nucléaire* (magyarul *Nukleáris Kutatások Európai Tanácsa*) a részecskefizikai kutatások európai szervezete, a világ legnagyobb részecskefizikai laboratóriuma, mely a francia-svájci határon helyezkedik el, Genftől kissé északra. Forrás: Wikipedia.

Az *elemek* (angolul *element*) a szöveget formailag egy *kezdő címke* (*start tag*) és egy *záró címke* (*end tag*) közé zárják:

```
<címke>szöveg</címke>
```

Az elem elején a kezdő címkét „kacsacsőrök” közé kell tennünk; a záró címkét szintén, de ott még egy per-jelre is szükség van, hogy egyértelművé tegyük, az az elem vége. A címkék természetesen nem fognak megjelenni a weboldalon, hanem csak kiegészítő információkat hordoznak, amiket a böngésző értelmez.

A címke viselkedését attribútumokkal tudjuk meghatározni, melyekből több is kapcsolódhat egy elemhez:

```
<címke attribútum1="érték1" attribútum2="érték2">szöveg</címke>
```

Az attribútumokat és értékeiket a kezdő címkénél adhatjuk meg, méghozzá úgy, hogy az értéket idézőjelek (`"érték"`) vagy aposztrófok (`'érték'`) közé tesszük.

A legtöbb elem a fenti formában egy szöveg formáját vagy szerepét (pl. címsor) határozza meg, de vannak más jellegű elemek is (pl. sortörés, kép hivatkozás). Ezért formailag léteznek ún. *önbezáró címkék* (*self-closing tag*), amelyeknek ilyen egyszerű a formájuk:

```
<címke />
```

Természetesen ehhez is kapcsolódhatnak attribútumok:

```
<címke attribútum1="érték1" attribútum2="érték2" />
```

Van lehetőség arra, hogy a HTML kódba megjegyzéseket (*comment*) tegyünk annak érdekében, hogy később emlékezzünk arra, mit miért csináltunk, vagy hogy éppen a weboldal melyik részének a kódját látjuk:

```
<!-- Itt következik a lábléc -->
```

Természetesen ezek a megjegyzések sem jelennek meg a böngészőben, de bárki számára láthatóak, aki a böngészőben kiválasztja a *View Source* menüpontot, ezért bizalmas információkat még véletlenül se írjunk ide.

> **JÓ TUDNI:** A HTML nyelv eredeti szabványa a fentieknél „lazább” szintaktikát is megenged. Például bizonyos esetekben a záró címke elhagyható (pl. új bekezdés kezdete egyben az előző végét is jelenti), az attribútumok értékeit nem kötelező idézőjelek közé tenni és az önbezáró elemeket sem kötelező jelölni. Ezek a lazán formázott HTML dokumentumok azonban sok bosszúságot okoznak a HTML szerkesztő programoknak és a HTML kódot értelmező böngészőknek is, ezért ma már inkább XHTML kódot szokás írni, amely az *XML* (*eXtensible Markup Language*) nyelv szigorú formai szabályait követi. A fent ismertetett szintaktika megfelel ezeknek az előírásoknak.

Fontosabb HTML elemek

Miután áttekintettük az elemek, címkék és attribútumok formáját, ismerkedjünk meg a legfontosabb HTML elemek jelentésével és használatával! Fontos látni, hogy a HTML nyelvben az elemek nevei rögzítettek, nem találhatunk ki találomra újabb elemeket, azokat kell használnunk, amik a szabványban megtalálhatóak.

A HTML dokumentum struktúrája

A HTML dokumentum első sora az ún. *Document Type Definition*, vagyis a *DTD*. A DTD határozza meg a böngésző számára, hogy a dokumentum pontosan milyen szabványt követ, így a böngésző

pontosan tudni fogja, hogy a szabványnak abban a verziójában milyen HTML elemek megengedettek, és melyiknek mi a jelentése.

A HTML 4.01 háromféle DTD-t ismer:

1. HTML 4.01 Strict: nem engedélyezi formázó elemek használatát, azok helyett mindenképp *Cascading Style Sheets* stíluslapok segítségével kell a formázást elvégezni. Ez a legelterjedtebb DTD:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
          "http://www.w3.org/TR/html4/strict.dtd">
```

2. HTML 4.01 Transitional: engedélyezi néhány korábbi, a szabványból később kikerült elem használatát is:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
          "http://www.w3.org/TR/html4/loose.dtd">
```

3. HTML 4.01 Frameset: a fentieken kívül keretek (*frame*-ek) használatát is támogatja:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
          "http://www.w3.org/TR/html4/frameset.dtd">
```

Hasonlóan létezik háromféle (strict, transitional, frameset) DOCTYPE deklaráció az XHTML 1.0 szabványhoz is, az XHTML 1.1 verzióhoz azonban csak ez az egy:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
          "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

A közelgő HTML5 verzióban (ami gyakorlatilag egyben az XHTML5 verzió is) szerencsére lényegesen egyszerűsödik ez a sor:

```
<!DOCTYPE html>
```

Fontos, hogy a HTML fájlunk elején szerepeljen ez a sor, különben előfordulhat, hogy a böngésző nem úgy jeleníti meg az oldalt, ahogy szeretnénk.

A `!DOCTYPE` után következik az oldal ún. *gyökér eleme* (*root element*), a `html` elem, ami tartalmazza a dokumentum fejlécét (`head`) és törzsét (`body`):

```
<!DOCTYPE html>
<html>
  <head>
    <!-- Ez a fejléc-->
  </head>

  <body>
    <!-- Ez a szövegtörzs -->
  </body>
</html>
```

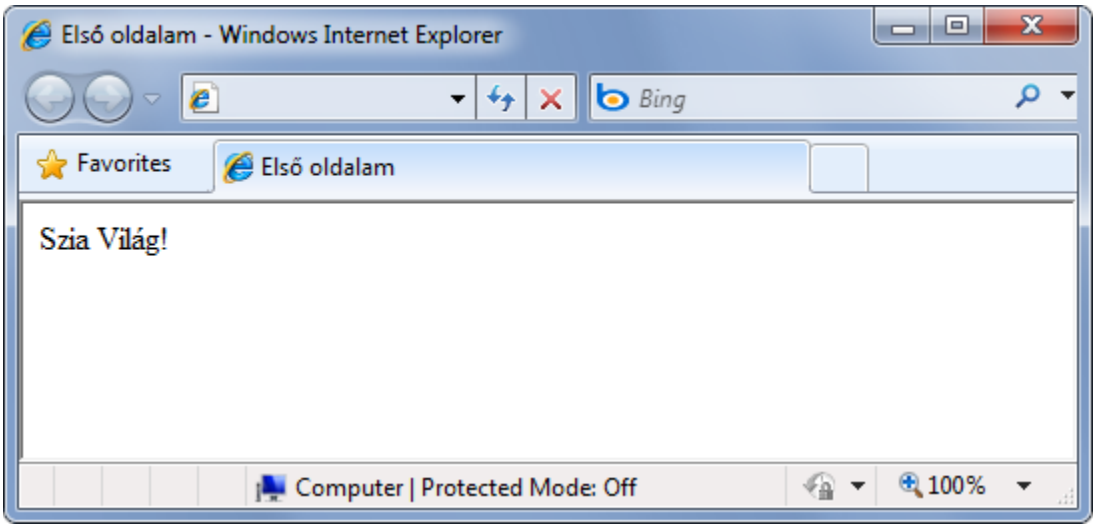
A fejlécben kap helyet a dokumentum címe, amit a böngésző az ablak címsorban megjelenít (`title` elem) és további leíró információk (`meta` elem), amik viszont nem jelennek meg az oldalon, például:

```
<head>
  <title>Bevezetés az internet programozásába</title>
  <meta name="author" content="Balássy György" />
  <meta http-equiv="Content-Language" content="hu" />
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
</head>
```

Ennyi tudással akár el is készíthetjük az első HTML oldalunkat! Készítsünk egy új szövegfájlt, nyissuk meg a Jegyzettömbbel (Notepad), és írjuk ezt bele:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Első oldalam</title>
  </head>
  <body>
    Szia Világ!
  </body>
</html>
```

Mentsük el a fájlt `.htm` kiterjesztéssel, majd kattintsunk rá duplán! Meg fog nyílni az elsődleges böngészőnk, és így fog megjelenni az oldalunk:



12. ábra - Az első HTML oldalunk

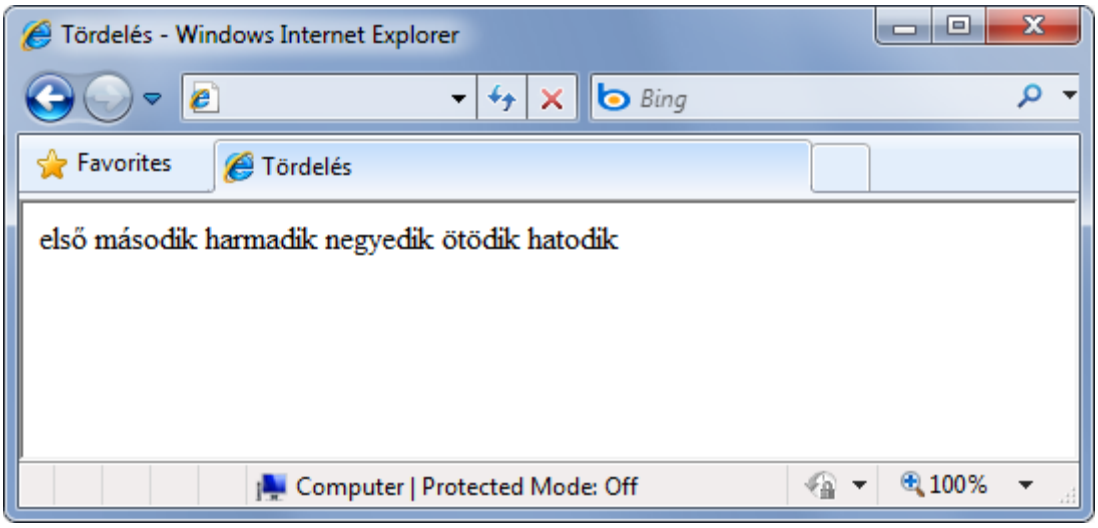
Látható, hogy a dokumentum törzse (`body` elem) tartalmazza azokat a HTML elemeket, amelyek alapján a böngésző megjeleníti az oldal tartalmát. A következő fejezetekben ezekkel az elemekkel foglalkozunk.

A szöveg struktúrája

Ha egy HTML dokumentumban rendezni, formázni szeretnénk a szöveget, akkor ezt kizárólag HTML elemekkel tehetjük meg. Azaz hiába teszünk sortöréseket a HTML kódba, vagy hiába teszünk egymás után sok szóközt, a böngészőben csak egyetlen szóköz fog megjelenni helyette. Például átírhatjuk az előző példánk `body` elemének kódját így:

```
<body>
  első második
  harmadik              negyedik

  ötödik  hatodik
</body>
```

De hiába is erőlködünk a sok szóközzel, sortöréssel és tabulátorral, a böngészőben úgy fog megjelenni, mintha csak egyetlen szóközt írtunk volna:



13. ábra - A HTML forráskód tördelése nem jelenik meg az oldalon

Ha másként szeretnénk, erre szolgáló címkéket kell használnunk.

Egy hosszabb szöveget a `p` (*paragraph*) elem segítségével tagolhatunk bekezdésekre:

```
<p>Első bekezdés</p>
<p>Második bekezdés</p>
```

Ha egy bekezdésen belül új sort szeretnénk kezdeni, akkor azt a `br` (*break*) elemmel tehetjük meg. A `br` elem önállóan áll, nincs sem záró címkéje, sem pedig attribútumai, mindössze ennyi:

```
<br />
```

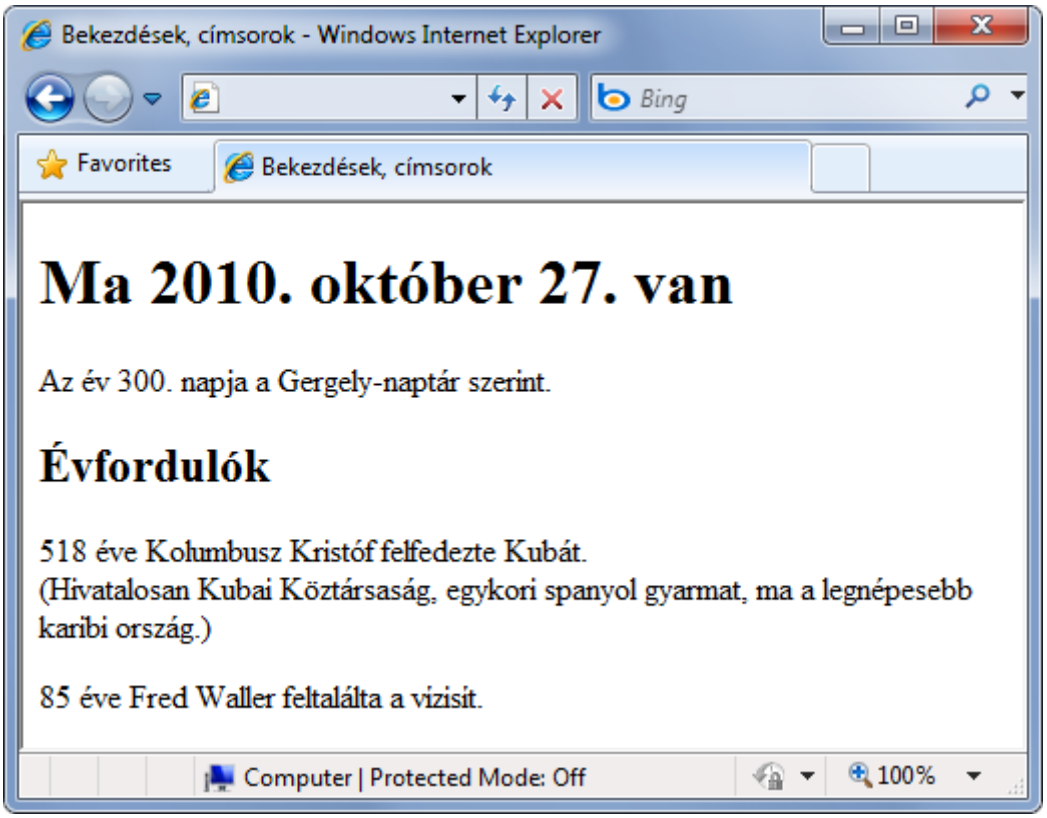
A bekezdések közé címsorokat tehetünk, méghozzá hat méretben, melyeket a `h1..h6` (*heading*) elemekkel jelölhetünk:

```
<h1>Főcím</h1>
<h2>Alcím</h2>
```

Rakjuk össze ezeket egyetlen példába:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Bekezdések, címsorok</title>
  </head>
  <body>
    <h1>Ma 2010. október 27. van</h1>
    Az év 300. napja a Gergely-naptár szerint.

    <h2>Évfordulók</h2>
    <p>
      518 éve Kolumbusz Kristóf felfedezte Kubát.<br />
      (Hivatalosan Kubai Köztársaság, egykori spanyol gyarmat,
           ma a legnépesebb karibi ország.)
    </p>
    <p>85 éve Fred Waller feltalálta a vízisít.</p>
  </body>
</html>
```

Mindez így jelenik meg a böngészőben:



14. ábra - Címsorok és bekezdések az oldalon

Érdemes megfigyelni, hogy a böngésző nagyobb betűkkel jeleníti meg a címsorokat, a bekezdések között automatikusan hagy távolságot, sőt a hosszú sorok tördeléséről is gondoskodik. Minden HTML elemre igaz, hogy van egy alapértelmezett megjelenési módja, amit természetesen megváltoztathatunk, ha mi másként szeretnénk. Erről részletesebben a *Cascading Style Sheets* c. fejezetben lesz szó.

Ha listákra van szükségünk, akkor kétféle elemet használhatunk attól függően, hogy számozott vagy csak egyszerű felsorolásos listát szeretnénk megjeleníteni az oldalon. A számozott listák keretét `ol` (*ordered list*) elemek jelentik, amin belül az egyes lista tételeket `li` (*list item*) címkék jelzik:

```
<ol>
  <li>Első</li>
  <li>Második</li>
  <li>Harmadik</li>
</ol>
```

Ha nincs szükségünk a sorszámokra, akkor az `ul` (*unordered list*) elemmel egyszerű felsorolásos listát készíthetünk:

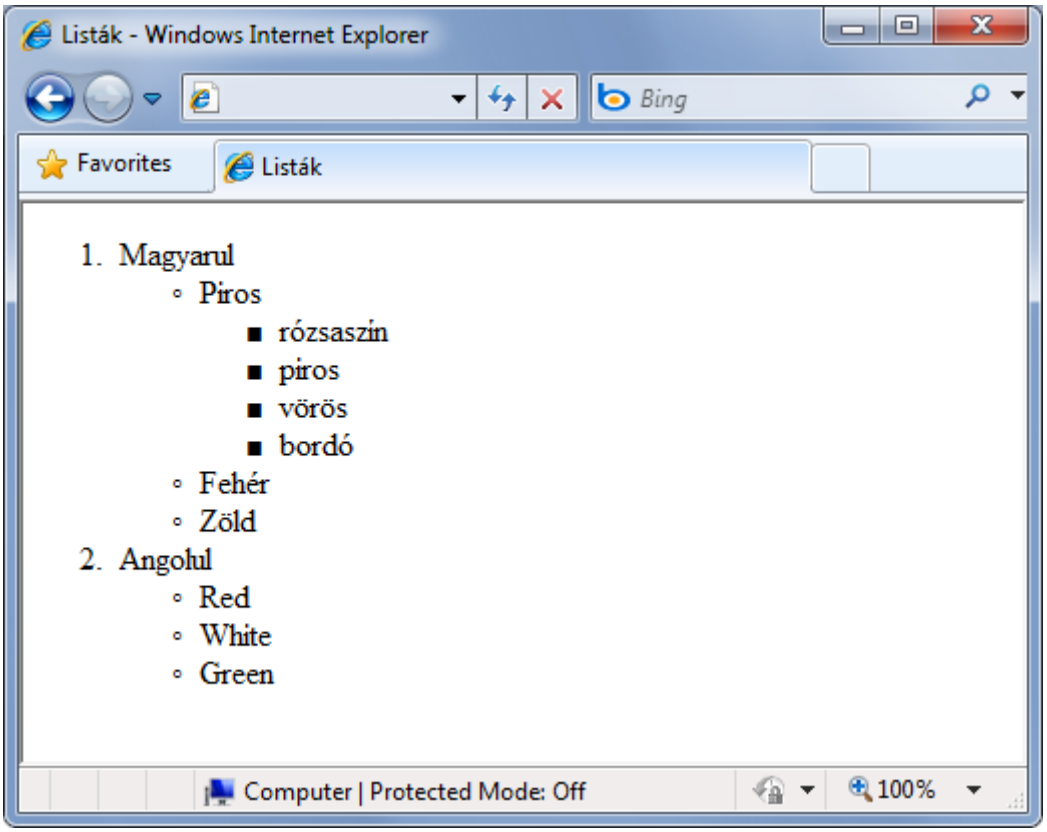
```
<ul>
  <li>Piros</li>
  <li>Fehér</li>
  <li>Zöld</li>
</ul>
```

Igény szerint a felsorolásokat és számozott listákat több szint mélyen akár egymásba is ágyazhatjuk, a böngésző automatikusan gondoskodni fog a megfelelő sorszám vagy lista ikon megjelenítéséről. Ez például egy többszintű lista:

```
<ol>
  <li>Magyarul
    <ul>
```

```
        <li>Piros
          <ul>
            <li>rózsaszín</li>
            <li>piros</li>
            <li>vörös</li>
            <li>bordó</li>
          </ul>
        </li>
        <li>Fehér</li>
        <li>Zöld</li>
      </ul>
    </li>
    <li>Angolul
      <ul>
        <li>Red</li>
        <li>White</li>
        <li>Green</li>
      </ul>
    </li>
  </ol>
```

Amit a böngésző így mutat meg:



15. ábra - Többszintű lista, automatikus számozással, ikonokkal, behúzással

A szöveg tagolására használható még a `hr` (*horizontal rule*) elem is, amely egy vízszintes vonallal választja el a felette és alatta lévő tartalmat. A `br` elemhez hasonlóan a `hr` is önbezáró:

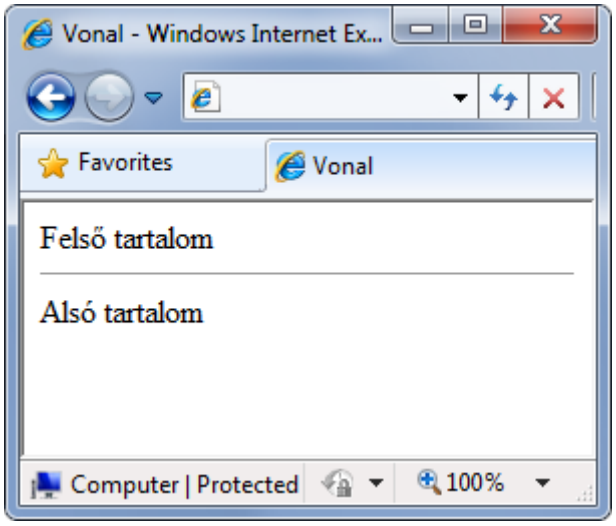
```
<hr />
```

A használata nagyon egyszerű:

```
Felső tartalom
<hr />
```

```
Alsó tartalom
```

A böngésző tudja a dolgát:



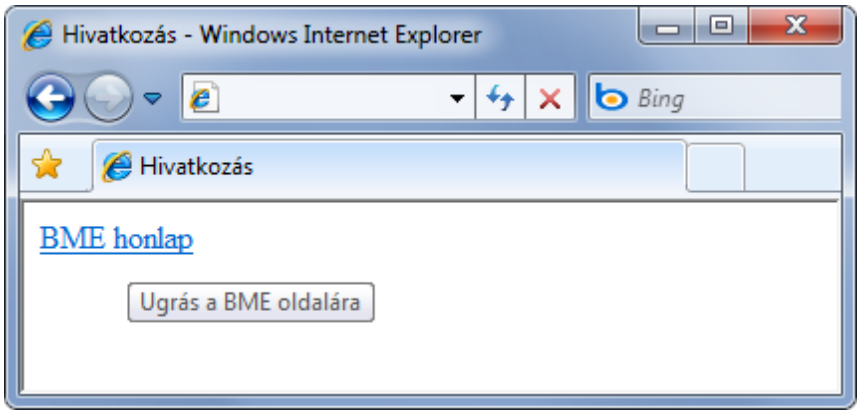
16. ábra - Így jelenik meg a <hr /> elem az oldalon

Hivatkozások

Az egyes weboldalaink között a kapcsolatot *hiperhivatkozás*okkal (*hyperlink*) teremthetjük meg, amit az `a` (*anchor*=horgony) elem valósít meg:

```
<a href="http://www.bme.hu"
   title="Ugrás a BME oldalára"
   target="_blank">BME honlap</a>
```

A nyitó és a záró címke között szerepel az a szöveg, ami meg fog jelenni a weboldalon, a `href` attribútumban pedig azt a webcímet kell megadnunk, amire ezzel a hivatkozással ugrani lehet. A `title` attribútumba olyan segítség írható, amely megjelenik a böngészőben, amikor a felhasználó a hivatkozás fölé viszi az egeret (ún. *tooltip*). A `target` attribútumban azt adhatjuk meg, hogy hova töltődjön be a hivatkozott weboldal. Ha az értéke `_blank`, akkor a böngésző új ablakban fogja megnyitni az oldalt.



17. ábra - A fenti hivatkozás kód így jelenik meg az oldalon

Szintén az `a` elem használható oldalon belüli ugrások, például tartalomjegyzék vagy az oldal tetejére mutató hivatkozás létrehozására. Ehhez meg kell jelölnünk a hivatkozni kívánt részt a `name` attribútummal:

```
<a name="LapTeteje" />
```

Majd az erre mutató linket be kell szúrnunk a kívánt helyre:

```
<a href="#LapTeteje">Ugrás az oldal tetejére</a>
```

Az a elemen belül tetszőleges elem állhat, például ha egy képet szeretnénk kattinthatóvá tenni, akkor a kép megjelenítésére szolgáló img elemet tehetjük a link belsejébe:

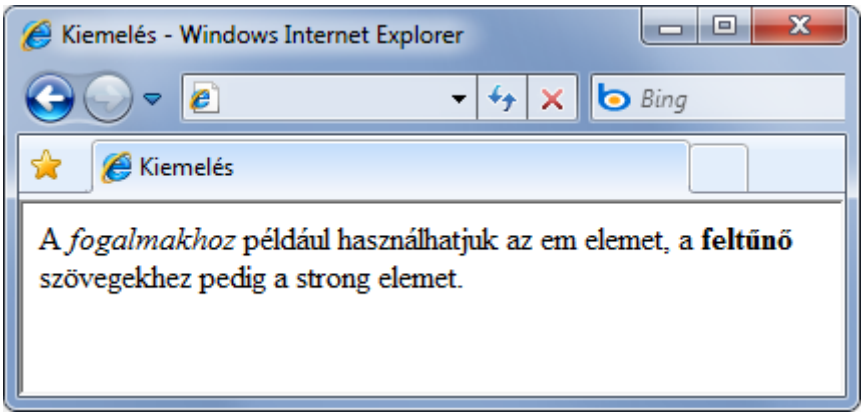
```
<a href="http://www.bme.hu">
  <img src="logo.gif" />
</a>
```

Szemantikai elemek

Az eddig bemutatott elemek többnyire a szöveg megjelenésén változtattak. A továbbiakban lássunk pár olyan HTML elemet, amelyek jelentést (szemantikát) is társítanak a szöveghez! Láttunk már erre példát a címsorok esetén, hiszen a <h1> elem azonkívül, hogy megnagyobbítja a szöveget még azt is jelenti, hogy az a pár szó az oldal címe.

Hasonlóan az és a elem nemcsak dőlten és félkövéren jeleníti meg a tartalmazott szöveget, hanem egyben azt is jelenti, hogy ez hangsúlyozott illetve kiemelt tartalom:

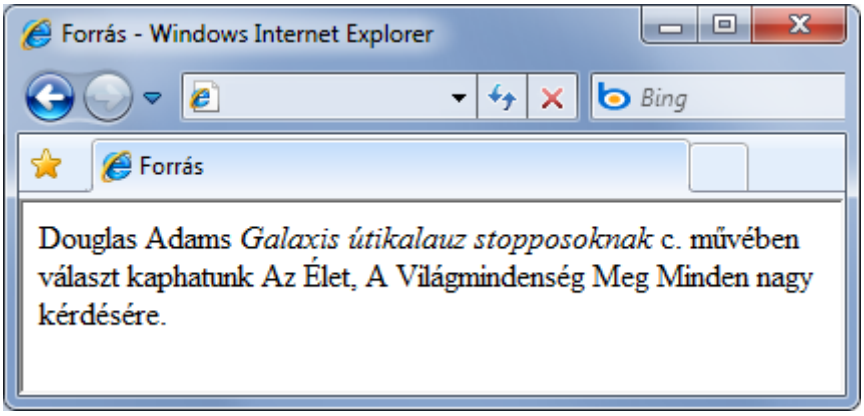
```
A <em>fogalmakhoz</em> például használhatjuk az em elemet, a
        <strong>feltűnő</strong> szövegekhez pedig a strong elemet.
```



18. ábra - Kiemelések a szövegben

A <cite> elem nemcsak hogy alapértelmezés szerint sok böngészőben dőlten jeleníti meg a tartalmazott szöveget, de azt is jelenti, hogy egy hivatkozott forrásról van szó:

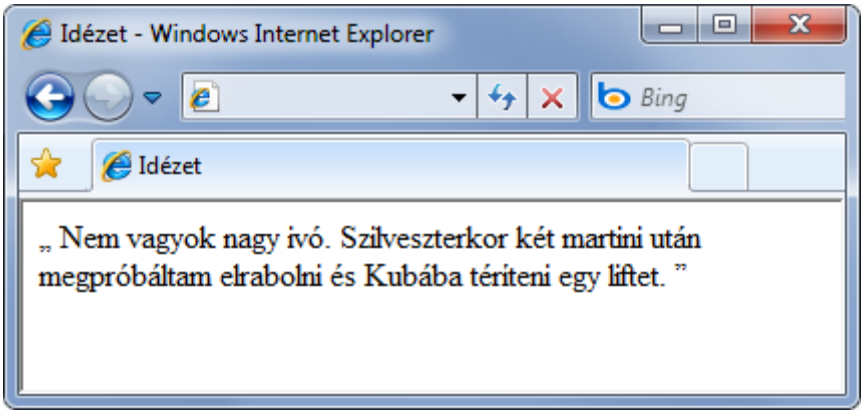
```
Douglas Adams <cite>Galaxis útikalauz stopposoknak</cite> c. művében
        választ kaphatunk Az Élet, A Világmindenség Meg Minden nagy
        kérdésére.
```



19. ábra - Forrás megjelölése

A külső forrásból átvett rövidebb idézeteket a `<q>` elemmel, a hosszabbakat pedig a `<blockquote>` elemmel jelezhetjük (ez utóbbiban több bekezdés is lehet, `<p>` elemekkel jelölve), mindkét esetben az opcionális `cite` attribútummal hivatkozhatunk a forrás URL-jére:

```
<q cite="http://hu.wikiquote.org/wiki/Woody_Allen">
          Nem vagyok nagy ivó. Szilveszterkor két martini után
          megpróbáltam elrabolni és Kubába téríteni egy liftet.
</q>
```



20. ábra - A szöveg köré automatikusan odakerültek az idézőjelek

A rövidítésekhez és mozaikszavakhoz az `<abbr>` és `<acronym>` elemeket használhatjuk. Ezeknek az elemeknek a törzse mindkét esetben a rövid verzió, a kifejtést a `title` attribútumban adhatjuk meg. A `title` attribútumra általában az jellemző, hogy a böngészők egy felugró tipp ablakban jelenítik meg a tartalmát, így ha a rövidítés fölé viszi a felhasználó az egeret, akkor rögtön látja a rövidítés feloldását is:

```
<acronym title="National Aeronautics and Space
          Administration">NASA</acronym>
```

Ezek a szemantikai HTML elemek (és a többi, amit itt nem tudtunk felsorolni) nagyban hozzájárulnak ahhoz, hogy a weboldalaink ne csak az emberi szem számára szépen megjelenő szövegek legyenek, hanem a szöveg egyes részeinek szerepe a feldolgozó programok számára is egyértelmű legyen. Például a vakok és gyengénlátók által használt képernyőolvasó programok szóban jelzik, ha címet, idézetet vagy rövidítést találnak.

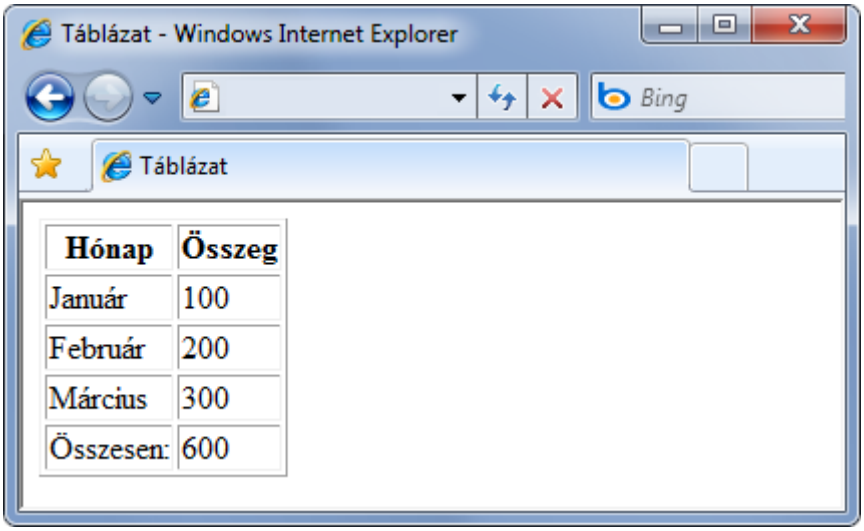
Táblázatok

A HTML nyelvben egy egész sor elem szolgál arra, hogy táblázatokat tudjunk megjeleníteni az oldalainkon. Egy három soros és két oszlopos táblázatot így készíthetünk el:

```
<table border="1" cellpadding="1" cellspacing="2" summary="Kiadások">
  <thead>
    <tr>
      <th>Hónap</th>
      <th>Összeg</th>
    </tr>
  </thead>
  <tfoot>
    <tr>
      <td>Összesen:</td>
      <td>600</td>
    </tr>
  </tfoot>
  <tbody>
    <tr>
```

```
      <td>Január</td>
      <td>100</td>
    </tr>
    <tr>
      <td>Február</td>
      <td>200</td>
    </tr>
    <tr>
      <td>Március</td>
      <td>300</td>
    </tr>
  </tbody>
</table>
```

A táblázatot a `<table>` elem jelzi, melyben az egyes sorokat `<tr>` (*table row*), azon belül az egyes cellákat pedig `<td>` (*table data*) elemek definiálják. A böngésző itt is ad egy alapértelmezett stílust az egyes elemeknek:



21. ábra - Egyszerű táblázat

Ha több cellát szeretnénk összevonni vízszintesen vagy függőlegesen, akkor azt a `<td>` elem `colspan` és `rowspan` attribútumaival tehetjük meg, ahol értékként azt kell megadnunk, hogy hány cellát kívánunk összevonni:

```
<table>
  <!-- Első sor két cellával -->
  <tr>
    <td>első</td>
    <td>második</td>
  </tr>

  <!-- Második sor egy cellával -->
  <tr>
    <td colspan="2">összevonva</td>
  </tr>
</table>
```

A cellák között a `<th>` (*table header*) elemekkel különböztethetjük meg a fejléc cellákat. Ha precízen jelölni szeretnénk, hogy sor vagy oszlop fejlécről van szó, akkor a `scope="col"` vagy `scope="column"` attribútumot kell használnunk.

Ha nagyobb táblázatról van szó, akkor azon belül célszerű jeleznünk a fejlécet, a láblécet és a táblázat törzsét, ami például az oldal kinyomtatásakor lehet hasznos információ a böngészőnek. A táblázatnak ezeket a részeit a `<thead>`, a `<tbody>` és a `<tfoot>` elemek jelölik. Elsőre kicsit szokatlan lehet, hogy a `<tfoot>` elemnek a `<tbody>` *előtt* kell állnia, hogy a böngésző a feldolgozás során időben hozzáférjen az abban szereplő információkhoz.

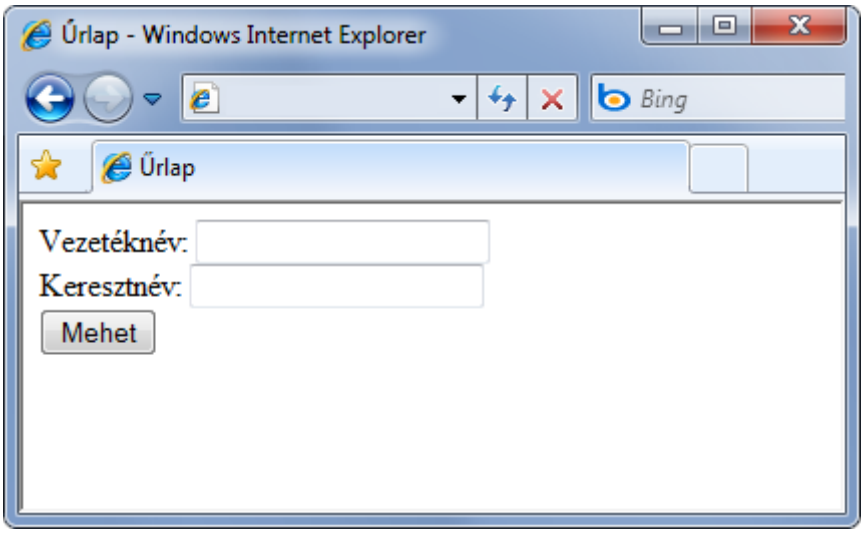
Űrlapok

A weboldalak nemcsak információk megjelenítésére használhatóak, hanem adatok bekérésére is szolgálhatnak. Ilyenkor űrlapokat kell készítenünk, amiket a felhasználó kitölt, majd az adatokat a böngésző egy HTTP POST kéréssel eljuttatja a szerverre. Hogy a szerver oldalon mi történik az elküldött adatokkal, az a webszerveren futó alkalmazás feladata.

Íme, egy egyszerű űrlap:

```
<form action="urlap.aspx" method="post">
  <label for="veznev">Vezetéknév:</label>
  <input id="veznev" type="text" />
  <br />
  <label for="kernev">Keresztnév:</label>
  <input id="kernev" type="text" />
  <br />
  <input type="submit" value="Mehet" />
</form>
```

Ami így jelenik meg a felhasználó számára:



22. ábra - Egyszerű űrlap két szövegdobozzal és egy gombbal

A `<form>` elem jelzi a böngésző számára, hogy itt egy űrlapról van szó. A `<form>` elem `action` attribútumában azt az URL-t kell megadnunk, ahova a böngésző a megadott adatokat el fogja küldeni, a `method` attribútumban pedig azt, hogy `HTTP GET` vagy `POST` formában várja a szerver az adatokat. Ezek nélkül az űrlap nem sokat ér.

Az űrlapon belül a mezőket és a mezőkhöz tartozó címkéket célszerű összekapcsolni egymással, hogy logikailag is legyen kapcsolat, ne csak a megjelenítéskor kerüljenek egymás mellé. A címkéket a `<label>` elem hordozza, a mezőkhöz tartozó beviteli vezérlők többségét pedig az `<input>` elem jeleníti meg. E kettő között úgy lehet kapcsolatot teremteni, hogy az `<input>` elem `id` attribútumában megadunk egy tetszőleges, az oldalon belüli egyedi azonosítót és erre hivatkozunk a `<label>` elem `for` attribútumában.

Azonosítóként bármit használhatunk, de a bevált gyakorlat szerint olyan azonosítót választunk, ami rövid, egyértelmű, illetve nem tartalmaz ékezeteket és szóközöket.

Az `<input>` elem `type` attribútumában kell megadnunk, hogy pontosan milyen beviteli mezőre van szükségünk. Ezek a lehetséges értékek:

- `text`: szövegdoboz
- `checkbox`: jelölőnégyzet
- `radio`: rádiógomb
- `password`: jelszó mező (szövegdoboz, de nem látszanak a beírt karakterek)
- `file`: fájl feltöltés
- `hidden`: rejtett

Az input vezérlővel gombokat is tehetünk az űrlapunkra, szintén a `type` attribútum beállításával:

- `submit`: elküldi az űrlap tartalmát a `<form>` elemben megadott címre
- `reset`: kitörli az összes űrlap mezőbe beírt értéket
- `button`: egyszerű nyomógomb, JavaScript kódot kell írnunk, amivel tetszőleges feladatot bízhatunk a gombra

Az input vezérlőnek van még két hasznos attribútuma. A `title` attribútumba olyan súgó szöveget írhatunk, ami akkor jelenik meg, ha a felhasználó a vezérlő fölé viszi az egeret. Az `accesskey` attribútumban pedig egy billentyűt határozhatunk meg, amivel a felhasználó (ha az `ALT` gombbal együtt, vagy más böngészőkben a `SHIFT+ALT` gombbal együtt nyom le) gyorsan „beleugorhat” a szöveg kurzorral a mezőbe. Így az űrlap nemcsak egérrel lesz kezelhető, hanem a billentyűzet használatával is mozoghatnak a felhasználók az egyes mezők között.

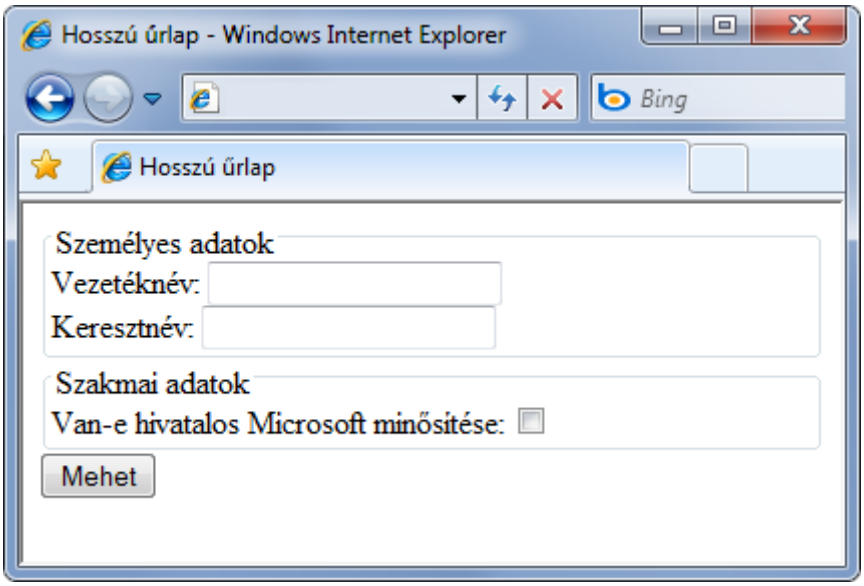
Az input vezérlőn kívül gyakran használunk még legördülő listákat, hogy a felhasználónak ne kelljen gépelnie, hanem előre meghatározott értékek közül választhasson ki egyet. A lehetséges értékeket a `<select>` elemen belül elhelyezett `<option>` elemekkel kell felsorolni:

```
<select id="szinek">
  <option value="P">Piros</option>
  <option value="F">Fehér</option>
  <option value="Z">Zöld</option>
</select>
```

Mikor a felhasználó választ egy elemet és elküldi az űrlapon megadott adatokat a szerverre, az `<option>` elem `value` attribútumában megadott értéket fogja megkapni a szerver, ezért fontos ennek az attribútumnak az egyértelmű, program által könnyen feldolgozható értékekkel történő kitöltése.

Ha egy űrlap hosszú, akkor célszerű azt részekre bontani, erre szolgál a `<fieldset>` elem, amin belül az egyes részeknek a `<legend>` elemmel adhatunk címet:

```
<form action="urlap.aspx" method="post">
  <fieldset>
    <legend>Személyes adatok</legend>
    <!-- Ide jönnek a személyes adatok mezői -->
  </fieldset>
  <fieldset>
    <legend>Szakmai adatok</legend>
    <!-- Ide jönnek a szakmai adatok mezői -->
  </fieldset>
</form>
```



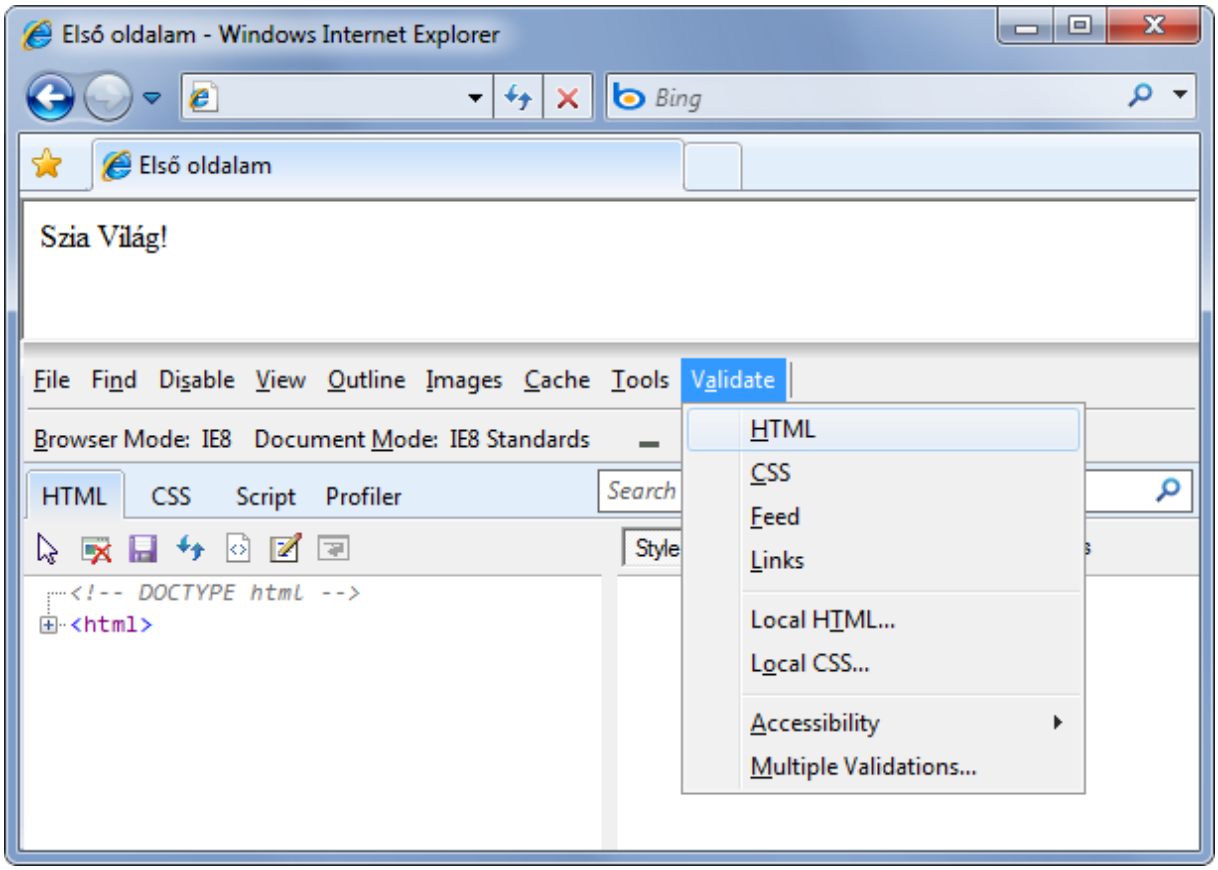
23. ábra - Hosszú űrlap több részre bontva

Validálás

Az előző fejezetekben számos elem szerepét és lehetőségeit tekintettük át, ám ezek mellett a HTML nyelv még sok, ritkábban használt elemet definiál. Azon kívül, hogy a HTML nyelvben csak ezeket az előre definiált elemeket használhatjuk, van még egy fontos megkötés, mégpedig az elemek egymáshoz viszonyított helyzetére vonatkozóan: a HTML szabvány szigorúan meghatározza, hogy az egyes elemeknek mely elemek lehetnek a gyermekei. Például nem lehet `<p>` elemeket egymásba ágyazni – szerencsére nem is lenne értelme bekezdésen belül bekezdést írni.

Sajnos nem mindenhol ennyire egyszerű és logikus a helyzet, ezért célszerű valamilyen programmal ellenőrizni, hogy az általunk elkészített HTML kód helyes-e. Erre kiválóan használható a W3C által biztosított *Markup Validation Service* (http://validator.w3.org). Ez egy ingyenes és nyilvános szolgáltatás, amelyre feltölthetjük az elkészített HTML oldalunkat, és válaszul egy részletes jelentést találunk a benne található hibákról. Ha a kódunk helyes, akkor pedig instrukciókat kapunk arra vonatkozóan, hogyan jelezhetjük oldalunk kiválóságát egy erre szolgáló logó segítségével.

Ha az oldalunkat már feltöltöttük egy nyilvános webszerverre, akkor a HTML kód validálásához használhatjuk *az Internet Explorer Fejlesztői Eszköztárán* (*Developer Toolbar*, F12) található *Validate* menüpontot:



24. ábra - Validálás közvetlenül Internet Explorerből

Ha az oldalunk még nem nyilvános, akkor el kell látogatnunk a *Markup Validation Service* weboldalára, ahol közvetlenül feltölthetjük az ellenőrizendő HTML kódot.

Sajnos a gyakorlatban nem mindig elég, hogy az oldalunk megfelel a szabványoknak, hiszen az oldalunk megjelenése leginkább a böngészőktől függ és így attól, hogy a böngészők hogyan implementálják a szabványokat. Ezért a szintaktikai tesztelés után mindig célszerű ellenőriznünk, hogy az oldalunk az elképzelésünk szerint jelenik-e meg az összes böngészőben. Ennél a tesztelésnél az alábbi eszközök segíthetnek:

- A *Microsoft Expression SuperPreview* képes egyszerre egymás mellett több böngészőben is megjeleníteni a tesztelni kívánt oldalt és az oldal egyes részeinek összehasonlításában segíteni. (http://www.microsoft.com/expression/products/SuperPreview_Overview.aspx)
- Az *IE Tester* egy ingyenesen letölthető alkalmazás, amely a különböző ablakaiban az Internet Explorer különböző verzióit képes betölteni, így ennek a böngészőnek az újabb és régebbi változatai alatt tesztelhetjük az oldalunkat (http://www.my-debugbar.com/wiki/IETester/HomePage).
- A *BrowserShots* ingyenes online szolgáltatásnak egy webcímet kell megadnunk, és közel 70, különböző platformon futó böngésző és böngésző verzió közül választhatjuk ki, hogy melyikkel akarjuk a tesztelést elvégeztetni. A szolgáltatás ezek után a háttérben megnyitja az oldalt a kiválasztott böngészőkben és készít róluk egy-egy képernyőfotót. (http://browsershots.org/)

Cascading Style Sheets stíluslapok

Avagy mitől lesz szép az oldal?

Egy weboldal készítésekor természetesen alapvető elvárás, hogy az oldalnak olyan arculatot adhassunk, amilyet csak szeretnénk. Nemcsak strukturálni szeretnénk a tartalmat, hanem formázni is, azaz szeretnénk megadni, hogy az egyes részletek, címek, bekezdések, képek hol és hogyan jelenjenek meg. A HTML nyelvben erre természetesen már a kezdetektől fogva van lehetőség. Egyes elemek tartalmazhatnak megjelenítésre vonatkozó attribútumokat (pl. `align`: igazítás), vagy például a `font` elemmel megadhatjuk a betűtípust, betűméretet és a színt. Íme egy példa:

```
<p align="center">
  <img src="logo.jpg" border="1" />
  <font size="10" color="red">
    Ez itt egy nagy piros szöveg középre igazítva.
  </font>
</p>
```

Ez a megközelítés működőképes, de nem túl szép, ugyanis a tartalom és a megjelenés nagyon keveredik egymással, amiből számos kellemetlenség következik:

- Egy teljes oldal forráskódját nagyon nehéz átlátni, hiszen a tiszta tartalmat felszabdalják a megjelenítésért felelős címkék és attribútumok.
- Nehéz következetes dizájnt készíteni, hiszen például a címsorokat minden esetben helyben kell megformáznunk, ügyelve arra, hogy mindenhol következetesen ugyanúgy nézzenek ki.
- Mivel minden oldalon újra és újra definiáljuk ugyanazokat az arculati beállításokat, ezért a hálózati forgalom jelentősen megnövekedhet.
- Nem lehet a webhely arculatát egyszerűen megváltoztatni, hiszen a dizájn részletei a tartalom között bújnak meg.

A nyilvánvaló megoldás az lenne, ha a HTML kódból ki tudnánk emelni a megjelenítésért felelős elemeket és attribútumokat, helyettük a szövegszerkesztőkben megszokott módon stílusokat definiálnánk, és a szövegben csak az így kialakított stílusokra hivatkoznánk. Szerencsére erre van megoldás, pont erre szolgál a *Cascading Style Sheets (CSS)* technológia, sőt ma már ez a javasolt és elvárt megközelítés a dizájn és az egységes arculat kialakítására.

Attribútumok

Cascading Style Sheets stíluslapokkal dolgozva a beállításokat *attribútum név-attribútum érték* párosokkal adhatjuk meg. Az attribútumok neve és értéktartománya rögzített, és gyakorlatilag az összes megjelenéssel kapcsolatos területet (méretek, távolságok, színek, pozíciók stb.) lefedik, így az ide vonatkozó HTML attribútumokra már nincs is szükség. A HTML és CSS attribútumok elnevezése néhol teljesen egyező, máshol csak hasonló. Például a fenti kódrészletben használt `color` attribútum létezik CSS-ben is, az `align="center"` HTML attribútum helyett viszont a `text-align` CSS attribútumot kell használnunk:

```
text-align: center;
color: red;
```

Formailag annyi megkötés van, hogy itt nincs szükség idézőjelekre, az egyenlőségjel helyett pedig kettőspontot kell használnunk. Egyszerre több CSS attribútumnak is adhatunk értéket, ebben az esetben pontosvesszővel kell elválasztanunk a név-érték párosokat egymástól.

Ha megjegyzést szeretnénk írni a CSS kódunkba, akkor azt `/*` és `*/` jelek között tehetjük meg:

```
/* Kiemelés */
color: red;
font-weight: bold;
```

Elhelyezés

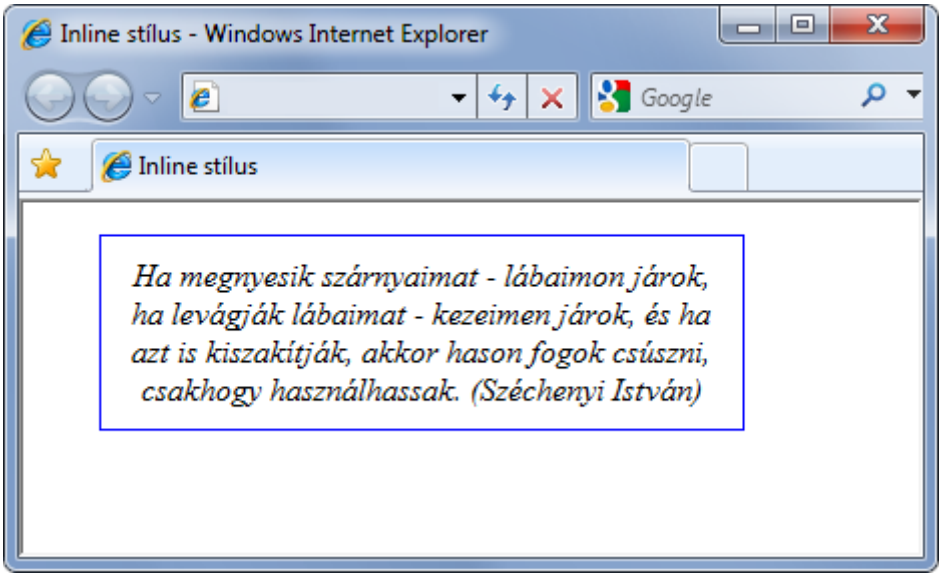
A következő kérdés nyilvánvalóan az, hogy ezeket a CSS beállításokat hogyan kapcsoljuk a HTML kódunkhoz. Erre háromféle lehetőségünk van:

Használhatunk ún. *inline stílust*, ekkor a formázandó HTML elem `style` attribútumába kell írnunk a CSS attribútumokat és azok értékeit:

```
<p style="font-style: italic; border: 1px solid blue; margin-left: 30px;
          width: 300px; padding: 10px; text-align: center;">
          Ha megnyesik szárnyaimat - lábaimon járok,
          ha levágják lábaimat - kezeimen járok,
          és ha azt is kiszakítják, akkor hason fogok csúszni,
          csakhogy használhassak. (Széchenyi István)
</p>
```

Az inline stílusnak megvan az az előnye, hogy bárhol használhatjuk, minden komolyabb előkészítés nélkül. A hátránya pedig természetesen az, hogy így a stílusbeállításaink továbbra is több helyen lesznek az oldalon, többszörösen növelik az oldal méretét, nekünk kell szinkronban tartani őket stb.

A fenti stílusbeállításoknak köszönhetően az egyszerű bekezdés így jelenik meg a böngészőben:



25. ábra - Egy bekezdés kerettel, margóval, rögzített szélességgel, középre igazítva

Egy másik lehetőség a CSS kód elhelyezésére a stílus blokk használata. Ebben az esetben tipikusan a HTML oldal `head` elemében hozunk létre egy `style` elemet, és oda írjuk a CSS kódunkat. Ekkor persze felmerül az a kérdés, hogy honnan fogják tudni az oldal egyes részei, hogy mely beállítások vonatkoznak rájuk, de erre még később visszatérünk. Egyelőre csak a `style` blokk szintaktikáját akarjuk megismerni:

```
<style type="text/css">
          /* A body elem és a gyermek elemeinek stílusa */
          body
          {
           font-family: Verdana, Arial, Sans-Serif;
           font-size: 75%;
           background-color: #f4e7bb;
          }
</style>
```

A stílus blokk lehetőséget ad arra, hogy a HTML oldalban egy helyre koncentráljuk a megjelenítésre vonatkozó részeket, így ha esetleg át kell szabnunk az oldal arculatát, egy központi helyen tehetjük

meg azt a CSS kód átírásával. Az a hátrány azonban így is megmarad, hogy a webhely által használt összes oldal között nekünk kell szinkronban tartani a CSS blokkokat.

Ezen segít a harmadik megoldás, amikor nem a HTML fájlban helyezzük el a CSS kódot, hanem egy külső fájlban, aminek tipikusan a `.css` kiterjesztést szoktuk adni. Ekkor a HTML oldal `head` elemében egy `link` elemmel hivatkozhatunk az oldalhoz tartozó stíluslapra:

```
<link href="style.css" rel="stylesheet" type="text/css" />
```

Ez a legpraktikusabb megoldás, hiszen ilyenkor a teljes webhelyünkhöz tartozó összes stílusbeállítás egyetlen (vagy néhány) fájlban van, erre hivatkozik az összes oldal, így ha módosítani kell valamit, akkor azt elég egyetlen helyen megtenni.

Jogosan teheti fel bárki a kérdést, hogy mi van akkor, ha mind a három megoldást egyszerre használom? Ebben az esetben a böngésző alkalmazni fogja az összes stílusbeállítást, mégpedig a következő sorrendben, az általánostól a specifikusabb irányba haladva:

1. Külső CSS fájl
2. Oldalon belüli stílus blokk
3. Az elemhez tartozó stílusbeállítások

Ez azt jelenti, hogy ha egy elemre a külső CSS fájlban megadunk egy beállítást, majd ugyanarra az elemre és ugyanarra a tulajdonságára (például háttérszín) az oldalon belül megadunk egy másik beállítást, akkor az oldalon belül definiált felülírja a külső fájlban megadott értéket. Úgy is mondhatnánk, hogy a közelebbi nyer. Viszont ha egy tulajdonságot csak a külső CSS fájlban adunk meg és az oldalon belül nem adunk meg erre a tulajdonságra másik értéket, akkor természetesen a korábbi beállítás érvényre fog jutni.

> Tehát lehet egy webhelyen belül egyszerre több helyen is stílusokat megadni, ám a javasolt megközelítés az, hogy lehetőleg csak külső CSS fájlt használjunk, mert ez a legrugalmasabb és legjobban karbantartható megoldás.

A CSS szelektorok

Az előző fejezetben a stílus blokknál már érintettük azt a problémát, hogy meg kell adnunk, hogy egy definiált stílus pontosan melyik HTML elemre vagy elemekre vonatkozzon. Erre szolgálnak a CSS *szelektorok* (CSS *selectors*).

Ha azt akarjuk, hogy egy beállítás az összes HTML elemre érvényes legyen, akkor a `*` (csillag) szelektort kell használnunk. Például a keretet (`border`), a külső- (`margin`) és a belső margót (`padding`) így szüntethetjük meg (a vastagságuk ill. a szélességük méretét kell számmal megadni):

```
* { border: 0; margin: 0; padding: 0; }
```

Látható, hogy formailag annyi követelmény van, hogy miután egy szelektorral meghatároztuk a kiválasztandó elemek körét, a rá vonatkozó stílusbeállításokat kapcsos zárójelek között kell megadnunk.

Hivatkozhatunk a HTML elem nevére is, ebben az esetben az összes helyen, ahol az adott elemet használjuk, automatikusan a megadott stílussal fog megjelenni. Például a második szintű címsor (`h2`) elé (fölé) tehetünk 25 pixel távolságot (`padding-top`) és aláhúzhatjuk (`text-decoration`) a szöveget:

```
h2 { text-decoration: underline; padding-top: 25px; }
```

A harmadik szelektor lehetőség, hogy egy konkrét elemre hivatkozunk az `id` attribútumán keresztül. Ehhez a HTML kódban egyedi értéket kell adnunk az `id` attribútumnak, és ezt kell megadnunk a CSS-ben is egy `#` jel után. Az alábbi példában egy adott címsort kis kapitális stílussal jelenítünk meg:

```
<style type="text/css">
  #idezetek { font-variant: small-caps; }
</style>
<h2 id="idezetek">Idézetek</h2>
```

Könnyen lehet, hogy nem az elem összes előfordulását, és nem is egy konkrét elemet akarunk formázni, hanem csak néhány kitüntetett helyen akarunk alkalmazni egy stílust. Ebben az esetben célszerű létrehoznunk egy saját stílus osztályt, aminek tetszőleges nevet adhatunk, a lényeg az, hogy pontot írjunk elé. Például definiálhatunk egy `szerzo` osztályt, ami meghatározza, hogy a szöveg legyen dőlt (`font-style`) és kisebb (`font-size`):

```
<style type="text/css">
  .szerzo {
    font-style: italic;
    font-size: 80%;
  }
</style>
```

> Ebben a példában a betűméretet %-ban, azaz a normál mérethez viszonyított arányban adjuk meg. Hasonló relatív megadási mód az **em** egység használata: **1em** az aktuális betűtípus méretével egyezik meg. Van lehetőség abszolút méret megadására is, ha képpontban, azaz *pixel*ben (**px**) vagy *point*ban (**pt**) adjuk meg az értéket.

Ezután azon a helyen, ahol használni akarjuk, a HTML elem `class` attribútumában tudunk rá hivatkozni:

```
<p class="szerzo">(Deák Ferenc)</p>
```

Így az oldalon több helyen is hivatkozhatunk erre a CSS osztályra, akár különböző HTML elemekben is.

> **JÓTANÁCS:** Célszerű olyan elnevezéseket választani a CSS osztályainkhoz, amik arra utalnak, hogy mire szolgál az adott osztály, nem pedig arra, hogy hogyan néz ki. Például a fontos dolgok hangsúlyozására szolgáló osztálynál a **kiemeles** jó elnevezés, a **nagypiros** viszont nem, még ha a lényeges részeket az oldalon nagyobb, piros betűvel jelenítjük is meg. Ha követjük ezt a bevált gyakorlatot, akkor nagyon könnyű lesz a dizájnon módosítani, például a **kiemeles** osztályt bátran átszínezhetjük kékre, a neve továbbra is arra utal, hogy a fontos részek megjelenítésénél használjuk. A **nagypiros** osztályban is átírhatjuk a színezést más színre, működni fog, csak éppen nem fogjuk később érteni, hogy miért hívjuk **nagypiros**nak, ha kéken jelenik meg, azaz nehéz lesz a kódunkat karbantartani.

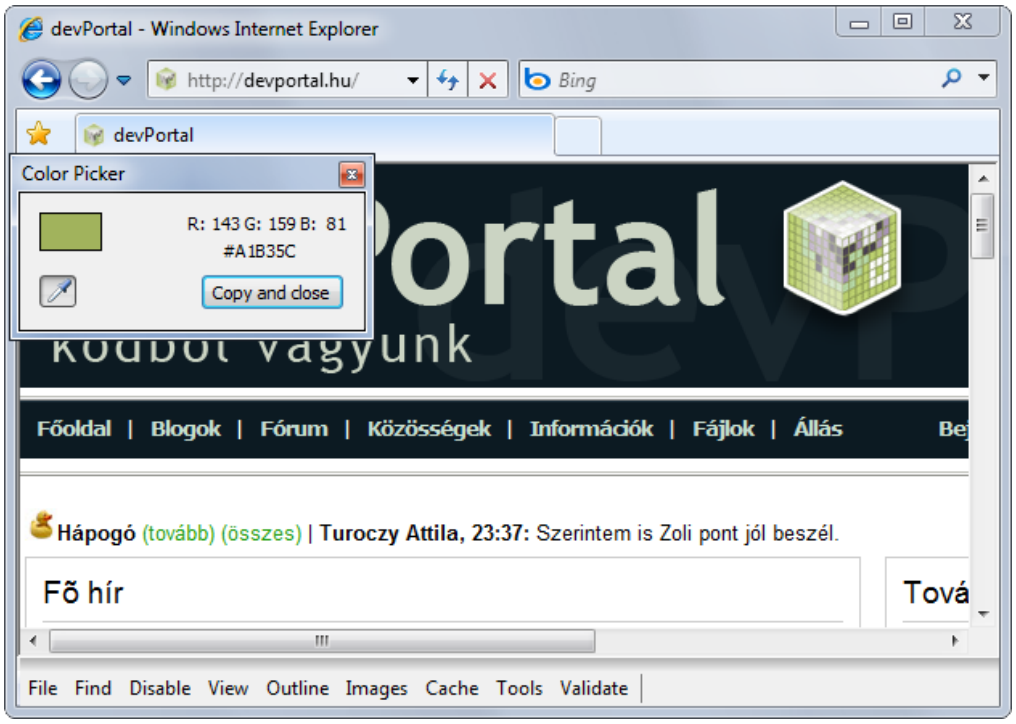
Ez a négy (*, elem, id, class) a leggyakrabban használt szelektor típus. Ezen kívül vannak még további speciális szelektorok, amelyek szintén hasznosak lehetnek munkánk során. Az ún. *pszeudo-osztályok* (*pseudo-class*) lehetővé teszik speciális tulajdonságokkal bíró elemek kiválasztását és ahhoz stílus definiálását. A leggyakoribb példa erre a hiperhivatkozás: HTML szinten mindegyik egy-egy `a` elemnek látszik, de az oldalon megjelenve lesz különbség köztük aszerint, hogy fölé visszük-e az egeret (`hover`), vagy hogy meglátogattuk-e már (`visited`) a hivatkozott oldalt. Ezekre az esetekre megadhatunk más formátumot CSS-ben:

```
a:hover   { color: #be2d00; }
a:visited { color: Black; }
```

> Ebben a példában az is látszik, hogy a színekre CSS-ben kétféleképpen hivatkozhatunk: a nevükkel vagy a kódjukkal. Név szerint meglehetősen sok színre tudunk hivatkozni

AliceBlue-tól **YellowGreen**ig, de természetesen csak a gyakoribbakra. A precíz megoldás a színkódok használata, ami a **#** jel után 6 karakter megadását jelenti. Ez a 6 karakter valójában 3×2 karakter, mindhárom helyen az RGB színskálának megfelelő 0-255 értékek hexadecimális változatai szerepelnek. Tehát például a piros színre hivatkozhatunk a nevével (**Red**) vagy a kódjával (**#ff0000**) is. Ha nem akarjuk magunk elvégezni az átváltást, akkor használhatjuk az **rgb** makrót, aminél zárójelben az RGB színkomponensek értékeit tízes számrendszerben kell megadnunk, például: **rgb(255,0,0)**

Tipp: ha tetszik valahol egy szín és szeretnénk megtudni a kódját, akkor használhatjuk az *Internet Explorer Developer Toolbar* (F12) *Tools* menüjének *Show Color Picker* (**CTRL+K**) eszközét. A megjelenő pipettával egyszerűen mutassunk rá a weboldalon a szimpatikus színre, és a kis ablakban máris látjuk a színhez tartozó HTML kódot:



Arra is van mód, hogy ún. *pszeudo-elemekkel* (*pseudo-element*) meghatározzuk egy elem speciális tulajdonságokkal bíró részének a formáját. Például az összes bekezdés első betűjét (`p:first-letter`) felnagyítva iniciálé hatást érhetünk el:

```
p:first-letter { font-size: 130%; }
```

Ezekkel a szelektorokkal a HTML dokumentum bármely elemét el tudjuk érni és meg tudjuk formázni. Sőt, a HTML elemek egymásba ágyazását kihasználva egészen specifikus szelektorokat is használhatunk. Az összes olyan képre (`img`), amely egy bekezdésen (`p`) belül található, így hivatkozhatunk:

```
p img { margin-right: 10px; }
```

Ezt persze fokozhatjuk akár több szint mélységben, sőt osztályokat és `id`-kat is bevonhatunk, azonban a gyakorlatban célszerű kerülni a szelektorok túlbonyolítását, mert a böngészőnek is tovább tart megjeleníteni, és mi sem fogjuk megérteni később, hogy miért így csináltuk, valamint hogy ezek pontosan hogyan is jutnak érvényre.

Span és div

Mi van akkor, ha nincs is a kívánt helyen semmilyen HTML elem? Ebben a kódrészletben például szerepel a nemkívánatos (elavult, nem javasolt a használata, de a böngészők megértik) `font` elem a lényeg pirossal történő kiemelésére:

```
Ez itt a <font color="red">lényeges</font> rész.
```

Ha az ajánlásoknak megfelelően mellőzzük a `font` elemet, akkor marad a csupasz szöveg, de vajon milyen szelektorral tudunk egyetlen szóra hivatkozni?

Sajnos semmilyennel, mindenképp szükségünk van egy HTML elemre. Ha ilyen problémával találkozunk, hogy az oldalon belül egy kisebb részt szeretnénk megformázni, akkor beszúrhatunk egy `span` elemet, és arra alkalmazhatjuk a stílust a korábban már megismert módokon:

```
<style type="text/css">
  .fontos {
    color: red;
  }
</style>

Ez itt egy <span class="fontos">lényeges</span> rész.
```

Ha az oldalnak egy nagyobb részét (például fejléc, jobb hasáb, friss hírek blokk stb.) szeretnénk formázni, akkor hasonlóan használhatjuk a `div` (*division*) elemet. Mindkét elemnek van `style`, `class` és `id` attribútuma, a formázás tehát hasonlóan megy mindkét esetben.

A nagy különbség, hogy a `span` ún. *inline* elem, a `div` pedig *blokk* elem. Ez első megközelítésben annyit jelent, hogy a `span` nem változtat az oldal elrendezésén, a `div` után viszont alapértelmezés szerint bekerül egy sortörés, ezzel is jelezve, hogy ott egy újabb nagyobb blokk következik. Ennél azonban fontosabb, hogy a `span` csak további inline elemeket tartalmazhat (pl. `strong`, `em`, `abbr`, de `div` vagy `p` nem), míg a `div` szinte bármilyen HTML elemnek lehet a szülő eleme, így akár bekezdéseket vagy további `div` és `span` elemeket is tartalmazhat. `Span`t majdnem bármilyen másik elembe tehetünk, a `div` azonban szigorúbb, például `span`be vagy `p` elembe nem ágyazhatjuk.

Öröklés

A CSS egyik legérdekesebb és leghasznosabb tulajdonsága, hogy a beállítások öröklődnek. Ennek köszönhető, hogy nem szükséges minden beállítást minden elemre definiálnunk, hanem elég a legkülső elemre megadnunk, onnan öröklődni fog a gyermek elemekre.

Lehet például egy bekezdésünk egy kiemelt résszel:

```
<p>
  Jártál-e már <em>Makkoshotykán</em>?
</p>
```

Ha a bekezdést kékkel írjuk, automatikusan a kiemelés is kék lesz, nem szükséges arra külön megadnunk ugyanezt a stílust:

```
p { color: blue; }
```

Szerencsére nem minden beállítás öröklődik, hanem csak azok, amelyeknek tipikusan kívánatos az öröklődése. A keretezés (`border`) például nem öröklődik. Ha öröklődne, akkor a bekezdésre alkalmazott kereten belül megjelenne egy külön keret Makkoshotyka körül is.

Ha szeretnénk kikényszeríteni az öröklést, akkor az `inherit` értéket adhatjuk meg az attribútumnak:

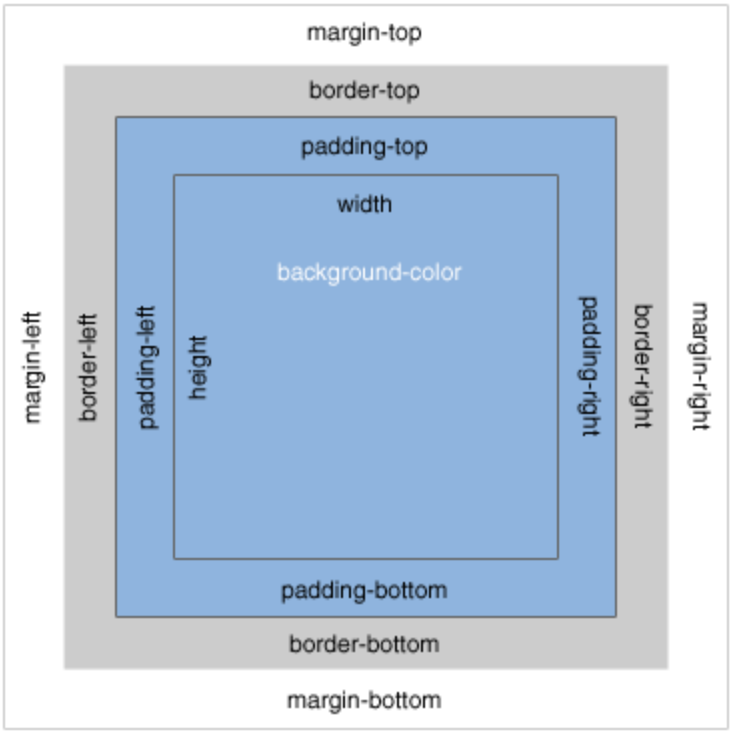
```
border: inherit;
```

A CSS-ben általában igaz az, hogy a specifikusabb beállítások fontosabbak, mint az általánosabb beállítások, ezért előfordulhat, hogy a szülőtől öröklődő értéket egy gyermek elemre megadott stílus felülírja. Ha ezt nem szeretnénk, használhatjuk az `!important` kiegészítést:

```
em { color: red; }
p { color: blue !important; }
```

Doboz modell

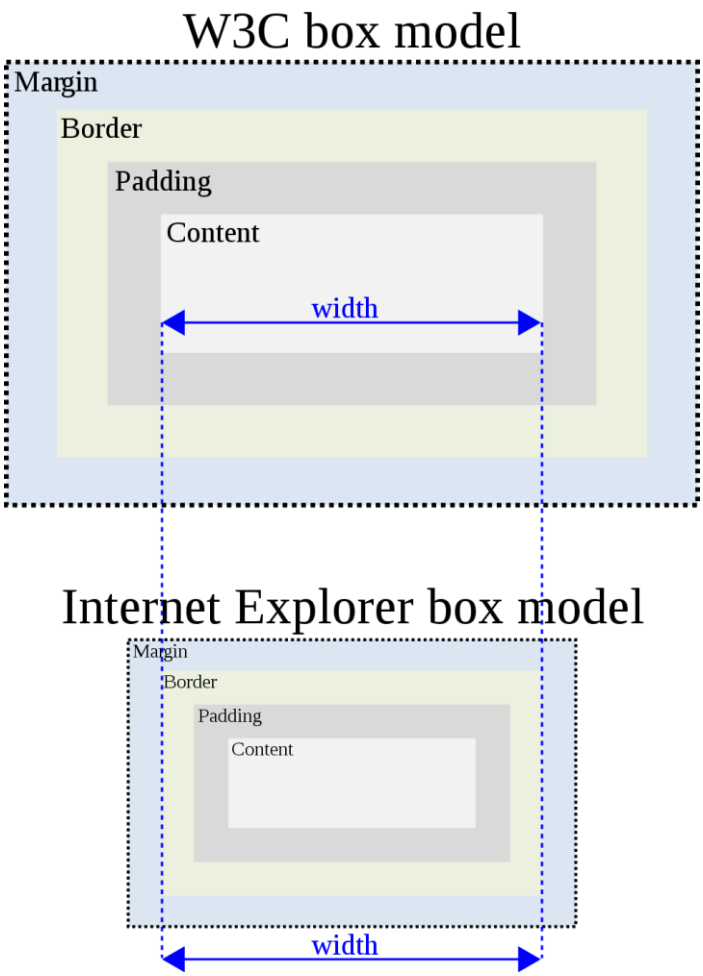
Az elemek méretének meghatározásához és helyes pozicionálásához érdemes megismerkednünk a *CSS doboz modellel* (*CSS box model*). Az alábbi ábra szemlélteti, hogy egy adott HTML elem (leggyakrabban `div`) esetén pontosan hol is állítjuk a belső margó (`padding`), keret (`border`) és külső margó (`margin`) értékeket:



26. ábra: A CSS doboz modell

Bár a doboz modell logikusnak és egyértelműnek látszik, történeti okokból sajnos a böngésző gyártók nem egyformán valósították meg, emiatt könnyen előfordulhat, hogy ugyanazt az oldalt különböző böngészőkben megtekintve pár pixel különbséget tapasztalunk. Ez többnyire nem okoz gondot, azonban ha nagyon kicentizzük az oldalt és mindent pixelre kikalkulálunk, akkor lehet, hogy kellemetlen élményben lesz részünk, például az egyik böngészőben egymás mellé kerül két elem, míg egy másik böngészőben egymás alá tördelődnek, mert ott éppen nem férnek ki egy sorba.

Az alábbi ábra például a W3C szabványnak megfelelő és az Internet Explorer régebbi változatai által alkalmazott doboz modell közötti különbséget szemlélteti:



27. ábra: IE és W3C doboz modell (Forrás: Wikipedia)

> **JÓTANÁCS:** Bár a szabványok és a böngészők egyre jobban közelednek egymáshoz, egyelőre sajnos nem a doboz modell a CSS egyetlen böngészőfüggő része (lásd QuirksMode.org). A váratlan problémák elkerülése érdekében ezért célszerű az oldalunkat minden böngészőben ellenőrizni, amit a webhelyen támogatni szeretnénk. Az oldal megjelenésének teszteléséhez jól használható a Microsoft Expression SuperPreview, az ingyenesen letölthető IE Tester és az online szolgáltatásként elérhető BrowserShots.

Oldalelrendezések

A webhely arculatának kialakításakor az egyik első feladatunk az oldalak elrendezésének kialakítása lesz. Ki kell találnunk, hogy milyen széles és milyen magas lesz az oldal, lesz-e fejléc, lábléc, esetleg menü – és persze azt is, hogy mindez hol és mekkora méretben fog megjelenni.

Hagyományosan erre létezik egy nagyon egyszerű megoldás, a táblázat alapú oldalelrendezés (*table-based layout*). Ennek a lényege, hogy a HTML nyelv által biztosított táblázat elemekkel (`table`, `td`, `tr`) valójában az egész oldalt egy nagy, láthatatlan keretekkel megrajzolt táblázatként képzeljük el. Például a bal hasáb a táblázat első oszlopa, a jobb hasáb a táblázat utolsó oszlopa, a felső menü pedig egy nagy cella, összevonva a táblázat összes oszlopát. Ha a nagy külső táblázat valamelyik celláját tovább kell osztanunk, akkor abban újabb táblázatot helyezünk el, ahol a sorokat és oszlopokat az igényeknek megfelelően vonjuk össze. A végeredmény egy gigantikus táblázat rengeteg, amiből szerencsére a végfelhasználó semmit nem vesz észre.

A táblázatos elrendezésnek a „súlyosbított” fajtája az ún. *spacer GIF*-ek alkalmazása. Ennél a megoldásnál az oldal egyes elemeinek pozicionálását átlátszó (azaz rejtett) képfájlok tetszőleges átméretezésével oldják meg, amelyek így épp a kívánatos mértékben tolják szét az oldalon lévő rejtett táblázatot.

Ennek a megközelítésnek azonban számos hátránya van:

- A HTML nyelv táblázattal kapcsolatos elemeit táblázatos *adatok* megjelenítésére találták ki. Egy táblázatnak lehet fejléc sora, lábléc sora és benne az egyes cellákban adatok szoktak megjelenni, és ezeket az adatokat lehet soronként vagy oszloponként olvasni. Egy szó mint száz, a `table` elem arra való, hogy a felhasználónak egy a hagyományos értelemben vett táblázatot jelenítsünk meg.
- A `table` elem szemantikai jelentését bizonyos programok kihasználják, például a látássérültek által használt képernyőolvasók vagy a keresőmotorok. Ezek az alkalmazások speciálisan értelmezik a táblázatot, ezzel segítve a felhasználót.
- A táblázat alapú oldalelrendezés nagyon sok HTML kódot eredményez, mely nemcsak az oldal letöltését és betöltését lassítja, hanem ráadásul nagyon nehezen átlátható, azaz nehezen tudunk később belejavítani.

> *„A pokolba vezető út átlátszó képekkel és egymásba ágyazott, rejtett táblázatokkal van kirakva.” - Bill Merikallio, Adam Pratt*

Ezek a megoldások a maguk idejében sem voltak ideálisak, ma már kifejezetten idejétmúltak, sőt nem túlzás azt állítani, hogy napjainkban igen ciki ezeket használni pozicionálásra.

A korrekt megoldás a *táblázatmentes oldalelrendezés* (*tableless layout*) alkalmazása. A táblázatmentes elrendezésnél az alapelv, hogy mindent stílusbeállításokkal próbálunk a megfelelő helyen és a megfelelő formában megjeleníteni. Ez a legtöbbször azt jelenti, hogy az oldalon létrehozunk számos `div` elemet (pl. fejléc, menü, kereső doboz, lábléc stb.) és ezeket CSS-sel méretezzük és pozicionáljuk.

A következő példában az egyik leggyakoribb oldalelrendezést készítjük el táblázatok nélkül. A cél egy olyan oldal, aminek van egy 50 pixel magas fejléce, egy 20 pixel magas lábléce, a kettő közötti rész pedig függőlegesen 30%-70% arányban osztott, és mindez összesen 950 pixel szélességben középre rendezve jelenik meg az oldalon, valahogy így:

<table>
<tr><td colspan="2">Fejléc</td></tr>
<tr><td>Menü</td><td>Tartalom</td></tr>
<tr><td colspan="2">Lábléc</td></tr>
</table>

Mivel az oldal egyes részei önmagukban is összetettek lehetnek (képek, bekezdések, felsorolások stb.), ezért minden egyes részt önálló `div` elembe zárunk, amiket pedig az `id` attribútumon keresztül egyedi névvel látunk el:

```
<div id="keret">
  <div id="fejlec">Fejléc helye</div>
  <div id="balmenu">Bal menü helye</div>
  <div id="tartalom">Tartalom helye</div>
  <div id="lablec">Lábléc helye</div>
</div>
```

A CSS fájlban ezekre az egyedi azonosítókra hivatkozva állítjuk be az oldal egyes részeinek megjelenését. Az egész oldal legyen 950 pixel széles és középre rendezett:

```
#keret{
  width: 950px;
  margin-left: auto;
  margin-right: auto;
}
```

Majd jöjjön az 50 pixel magas fejléc:

```
#fejlec{
  height: 50px;
}
```

A következő két `div` a `balmenu` és a `tartalom` egymás mellett:

```
#balmenu{
  width: 30%;
  float: left;
}

#tartalom{
  width: 70%;
  float: right;
}
```

Végül pedig a lábléc, ami a teljes szélességet kitölti és 20 pixel magas:

```
#lablec{
  height: 20px;
  clear: both;
}
```

Ezzel készen is vagyunk. Látható, hogy a HTML kód nagyon egyszerű maradt, és az azonosítóknak köszönhetően jól olvasható, hogy az egyes blokkok az oldal melyik részét jelentik. A CSS kód sem bonyolult, ráadásul nagyon rugalmas, hiszen könnyen vehetjük magasabbra a fejlécet vagy éppen szélesebbre a menüt.

Több stíluslap

Van lehetőségünk arra, hogy egy HTML oldalhoz több CSS stíluslapot rendeljünk. Ehhez mindössze annyit kell tennünk, hogy a HTML oldal `head` részébe több `link` elemet teszünk és több külső `.css` fájlra hivatkozunk. Ez több esetben is hasznos lehet.

Az egyik gyakori eset több stíluslap alkalmazására az ún. *reset CSS*. A böngészők az egyes HTML elemekhez rendelnek egy alapértelmezett megjelenítést, például a címsorokat nagyobb betűvel jelenítik meg, a bekezdések között hagynak egy kis helyet stb. Sajnos ez nem egészen egyezik az egyes böngészőkben, éppen ezért bevált gyakorlat egy önálló CSS fájl alkalmazása, amely eltörli ezeket a különbségeket. Eric Meyer például ezt javasolja (részlet):

```
html, body, div, span, applet, object, iframe, h1, h2, h3, h4, h5, h6, p,
          blockquote, pre, a, abbr, acronym, address, big, cite, code,
          del, dfn, em, font, img, ins, kbd, q, s, samp, small, strike,
```

```
            strong, sub, sup, tt, var, b, u, i, center, dl, dt, dd, ol, ul,
            li, fieldset, form, label, legend, table, caption, tbody, tfoot,
            thead, tr, th, td {
  margin: 0;
  padding: 0;
  border: 0;
  outline: 0;
  font-size: 100%;
  vertical-align: baseline;
  background: transparent;
}
body {
  line-height: 1;
}
ol, ul {
  list-style: none;
}
```

Egy másik érdekes lehetőség az ún. *alternatív stíluslapok* (*alternate style sheet*) alkalmazása. Itt is arról van szó, hogy egy weboldalhoz több CSS fájlt rendelünk hozzá, csakhogy ezek nem egyszerre alkalmazódnak, hanem mindig csak az, amit a felhasználó kiválaszt. Így az oldal szokásos megjelenítésén kívül készíthetünk például gyengénlátóknak szánt erősen kontrasztos változatot. Ennek a megvalósítása is a `head` elembe tett `link` elemekkel lehetséges, csak éppen még nevet is kell adnunk a stíluslapunknak a `title` attribútumon keresztül:

```
<!-- Ez mindig alkalmazódni fog -->
<link rel="stylesheet" type="text/css" href="reset.css" />

<!-- Ez lesz az alapértelmezett, de lecserélhető -->
<link rel="stylesheet" type="text/css" href="normal.css"
          title="Normál változat" />

<!-- Alternatív stíluslap -->
<link rel="alternate stylesheet" type="text/css" href="large.css"
          title="Gyengénlátó változat" />
```

Az alapértelmezett és az alternatív változatok között az oldal látogatói a böngésző beépített menüpontjai segítségével válthatnak (Firefoxban például *View* menü *Page style* menüpontjával).

A több stíluslap lehetőségének legpraktikusabb kihasználása kétségkívül a nyomtatóbarát változat megvalósítása. Amikor kinyomtatunk egy weboldalt, akkor a böngésző megpróbálja pontosan azt megjeleníteni a nyomtatón, amit a képernyőn is látunk, ezzel azonban számos probléma lehet: előfordulhat, hogy az oldal szélesebb, mint a papír, a szöveg sötét alapon fehér betűkkel szedett, az oldal bal hasábja (ami biztosan ráfér a papírra) csak menüt és reklámot tartalmaz stb. Ezeknek a problémáknak az elkerülésére célszerű az oldalhoz egy nyomtatóbarát stíluslapot is készíteni. A `link` elem `media` attribútumában adhatjuk meg, hogy az adott stíluslapot a böngésző a képernyőn használja (`screen`) vagy nyomtatáskor (`print`):

```
<link rel="stylesheet" href="main.css" type="text/css" media="screen" />
<link rel="stylesheet" href="print.css" type="text/css" media="print" />
```

A nyomtatóbarát változat számos ponton eltérhet az oldal szokásos megjelenítésétől:

- Fehér alapon fekete betűkkel jelenik meg.
- Felesleges képernyő elemek (például menü, reklám) eltűnnek.
- Extra információk jelenhetnek meg (például szerzői jogok, linkek URL-jei).
- Serif (talpas) betűt használ (nyomtatásban jobban olvasható).
- Abszolút helyett relatív méretezést alkalmaz (például 950px szélesség helyett 100%).

A leggyakoribb feladat a nemkívánatos blokkok elrejtése, például az oldalelrendezéseknél látott bal oldali menüt így rejthetjük el:

```
#balmenu{
  display: none;
}

#tartalom{
  width: 100%;
}
```

A nyomtatáshoz használt CSS beállítások kialakítása időigényes feladat, de megéri, mert a felhasználó sokkal elégedettebb lesz, amikor a jól bevált módon, a böngésző *Nyomtatás* ikonjára kattintva kinyomtatja az oldalt, ezért általában megéri ezzel fáradnunk.

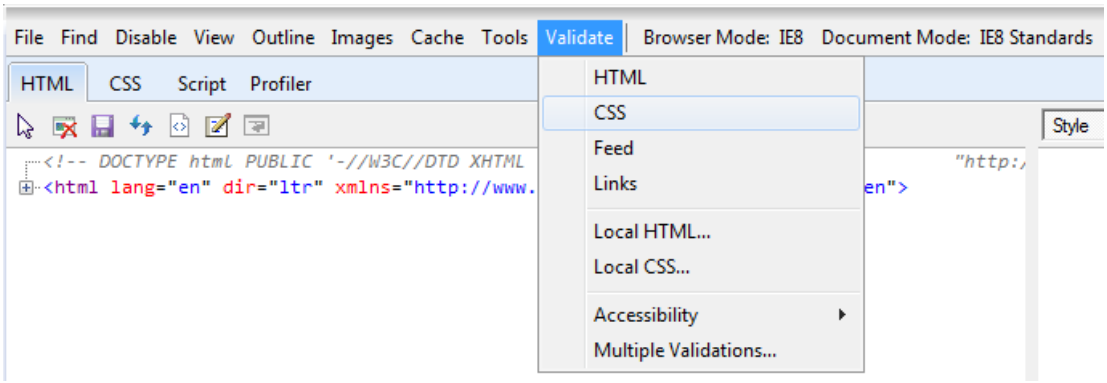
Ellenőrzés

Miután elkészültünk az oldalunkkal és a hozzá tartozó CSS fájllal, célszerű lefuttatnunk egy ellenőrzést, hogy elkészült kódunk megfelel-e a szabványnak. Erre legegyszerűbben a *W3C CSS Validation* szolgáltatását használhatjuk, amely a http://jigsaw.w3.org/css-validator/ címen ingyenesen érhető el.

Itt megadhatjuk egy nyilvánosan elérhető webhelynek a címét, feltölthetünk a gépünkről egy CSS fájlt, vagy akár egy nagy szövegdobozba is bemásolhatjuk a CSS kódunkat, majd a *Check* gombra kattintva azonnal megkapjuk az elemzés eredményét.

28. ábra: A W3C CSS Validation szolgáltatása

A W3C-nek ezt a hivatalos szolgáltatását számos webfejlesztő program és böngésző kiegészítés beépítetten tartalmazza, például megtalálható az *Internet Explorer Developer Toolbar*on is a *Validate* menüpont alatt:



29. ábra: CSS validator az IE Developer Toolbaron

Ennek a fejezetnek a segítségével megértettük az internet működését, valamint tisztában vagyunk az alapvető technológiákkal és protokollokkal. Itt az ideje, hogy elmélyedjünk a dinamikus weboldalak készítésének rejtelmeiben.

2. fejezet – Bevezetés a WebMatrixba és az ASP.NET weboldalakba

Ebben a fejezetben megismerhetjük a Microsoft WebMatrixot, egy ingyenes webfejlesztő technológiát, mely a legjobb környezetet nyújtja webfejlesztők számára.

A fejezetből megtudhatjuk:

- Mi az a WebMatrix?
- Hogyan telepíthetjük fel?
- Hogyan kezdjünk el egyszerű weboldalakat létrehozni a WebMatrix segítségével?
- Hogyan készíthetünk dinamikus weboldalakat a WebMatrixszal?
- Hogyan programozhatunk weboldalakat a Visual Studio segítségével, hogy még többet kihozhassunk az oldalunkból?

Mi az a WebMatrix?

A WebMatrix egy ingyenes, könnyen átlátható fejlesztőkörnyezet, mely többféle webes eszközt magába foglalva teszi gyerekjátékká weboldalak elkészítését. A csomag az IIS Expresst (egy fejlesztői webszervert), az ASP.NET-et (egy webes futtatókörnyezetet) és az SQL Server Compactot (egy beépített adatbázist) tartalmazza. Ezen felül találunk egy egyszerű eszközt, mellyel hatékonyabban fejleszthetjük és hamarabb elindíthatjuk weboldalunkat különféle nyílt forráskódú modulok felhasználásával. A most elsajátított tudásunkat és WebMatrixban fejlesztett kódunkat könnyedén felhasználhatjuk a Visual Studio vagy SQL Server használata során is.

A WebMatrix segítségével készült weboldalak dinamikusak is lehetnek – felhasználói interakció vagy más események, például az adatbázis változása esetén megváltozhat a lapok tartalma vagy megjelenése. ASP.NET alapú dinamikus weboldalak készítéseskor Razor szintaxist, C# vagy Visual Basic programozási nyelvet is használhatunk.

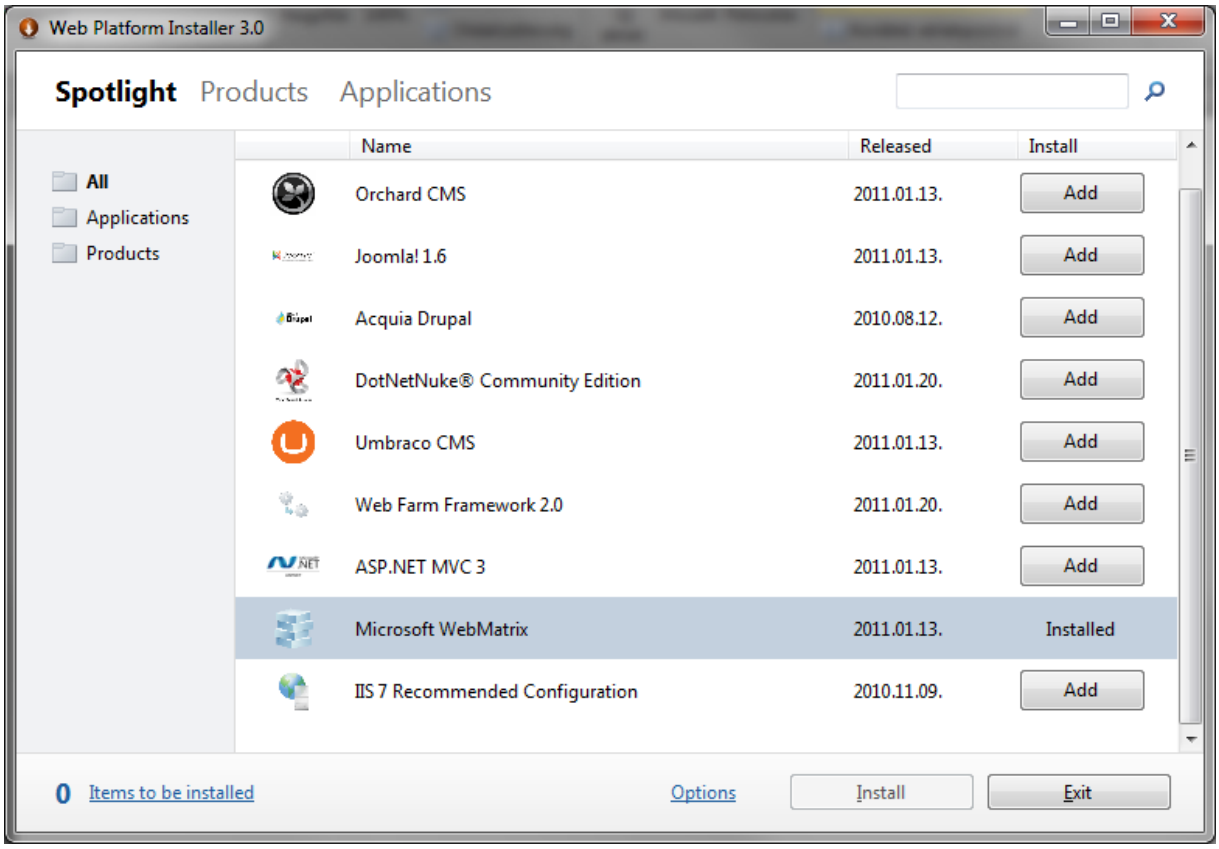
Ha már használunk valamilyen programozói eszközt, kipróbálhatjuk a WebMatrix eszközöket, vagy készíthetünk saját eszközeinkkel olyan weboldalakat, melyek az ASP.NET-et használják.

Ebben a fejezetben megtudhatjuk, hogyan segít a WebMatrix abban, hogy webhelyeket hozzunk létre, és dinamikus weboldalakat készítsünk.

Webmatrix telepítése

A WebMatrix telepítéséhez használhatjuk a Microsoft ingyenes Web Platform Installerét, mellyel egyszerűen telepíthetjük és konfigurálhatjuk a webbel kapcsolatos technológiákat.

1. Ha még nem rendelkezünk a Web Platform Installerrel, letölthetjük a következő címről: http://go.microsoft.com/fwlink/?LinkID=205867
2. Indítsuk el a Web Platform Installert, válasszuk a *Products* fület, és kattintsunk a WebMatrix felirat melletti *Add* gombra a telepítéshez!



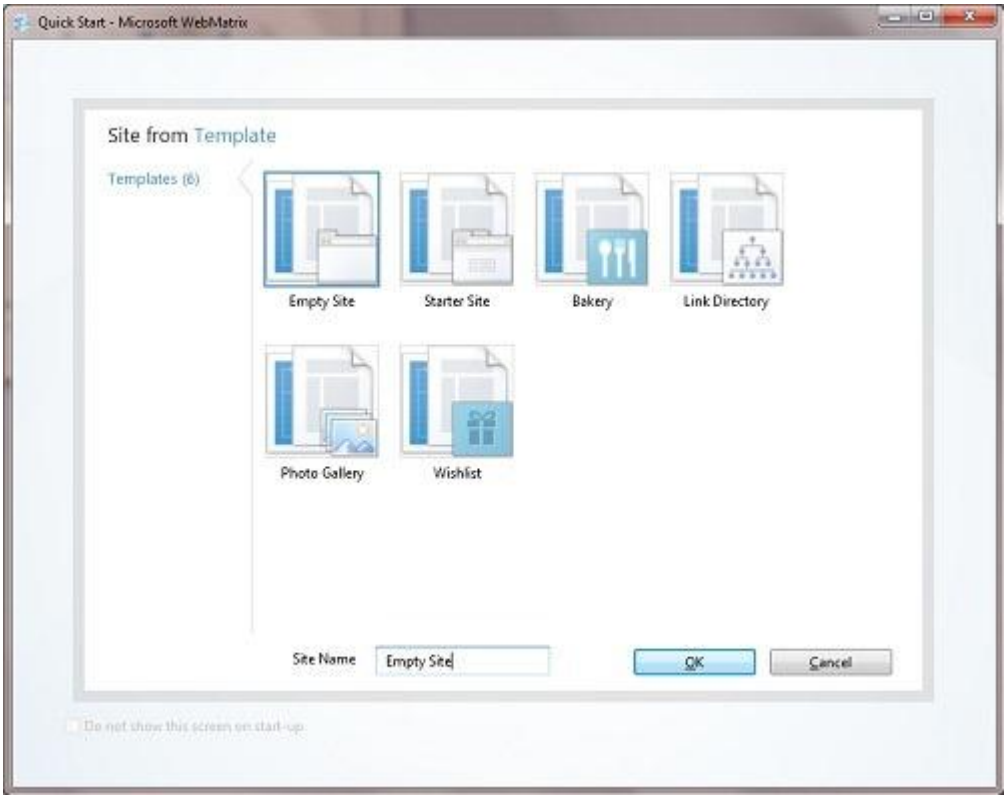
Megjegyzés: Ha már korábban feltelepítettük a WebMatrix Beta verzióját, a Web Platform Installer frissíti azt a végleges verzióra. Ebben az esetben előfordulhat, hogy a korábban készített webhelyek nem jelennek meg a *My Sites* listában, amikor először megnyitjuk a WebMatrixot. Egy korábban készített webhely megnyitásához kattintsunk a *Site From Folder* ikonra, válasszuk ki a kívánt webhelyet, és kattintsunk az Open gombra! A WebMatrix következő megnyitásakor a webhely már látszani fog a *My Sites* listában.

Első lépések a WebMatrixszal

Elsőként hozzunk létre egy új webhelyet és egy egyszerű weboldalt!

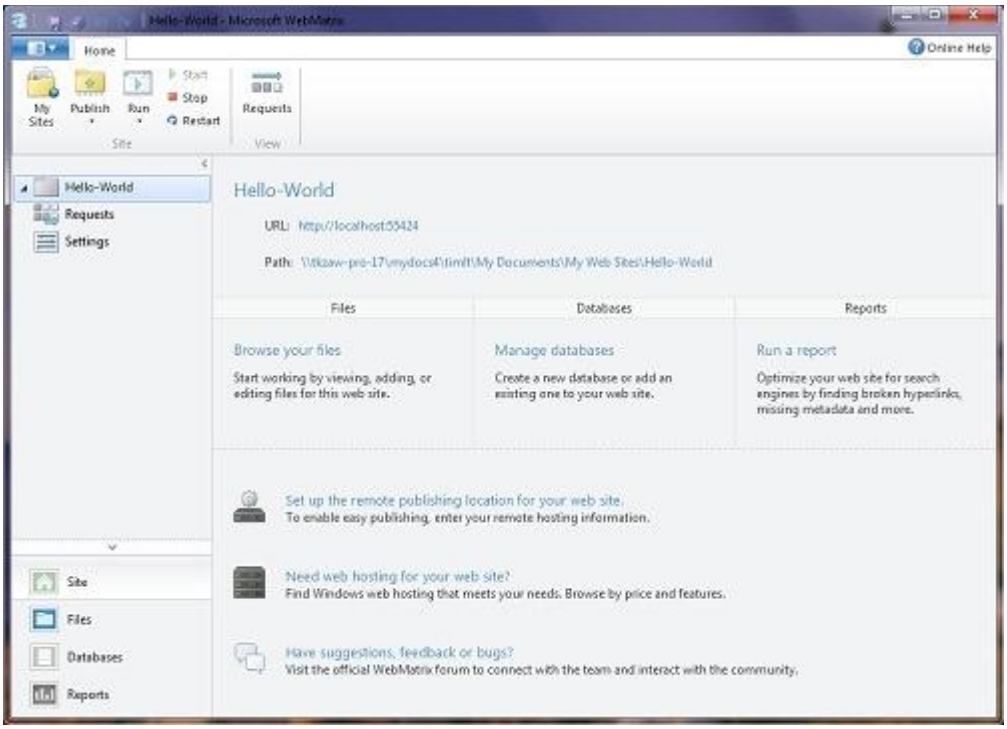
1. Indítsuk el a WebMatrixot!

2. Válasszuk a *Site From Template* lehetőséget! A sablonok (Templates) előre megírt fájlokat és oldalakat tartalmaznak különböző típusú webhelyekhez.



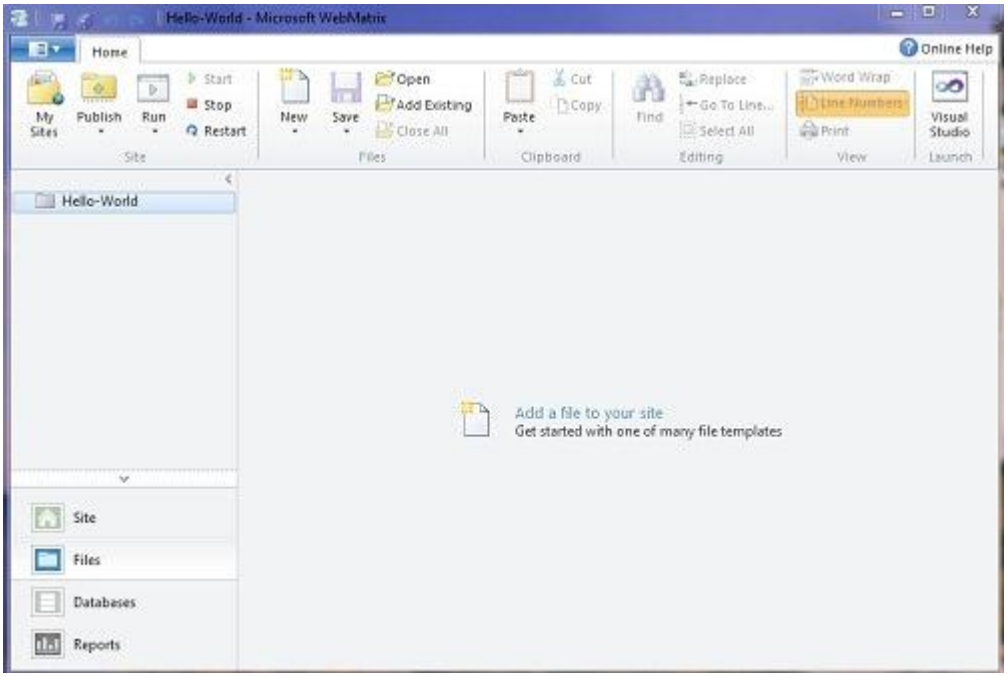
3. Válasszuk az *Empty Site* lehetőséget, és nevezzük *Hello-World*-nek!
4. Kattintsunk az *OK* gombra. A WebMatrix létrehozza és megnyitja az új webhelyünket.

Az ablak tetején találjuk a Gyorselérési eszköztárat és a szalagot, ahogyan a Microsoft Office 2010 programjaiban megszokhattuk. A bal alsó sarokban láthatjuk a munkafelület-választót (workspace selector), ahol a különböző gombokra kattintva változtathatjuk, hogy mi jelenjen meg felettünk a bal oldali panelon. Jobb oldalon találjuk a tartalom panelt, ahol például jelentéseket tekinthetünk meg, vagy fájlokat módosíthatunk. Végül az alsó sorban az értesítési sávot találjuk, ahol az épp aktuális tudnivalókról értesülhetünk.

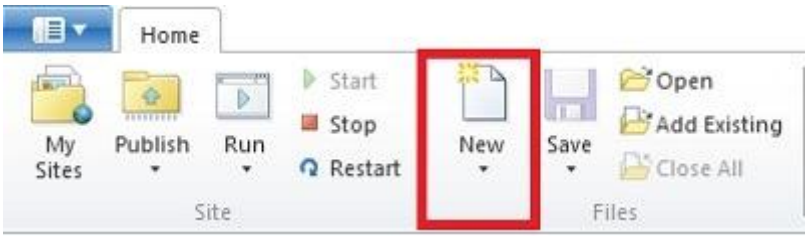


Weboldal létrehozása

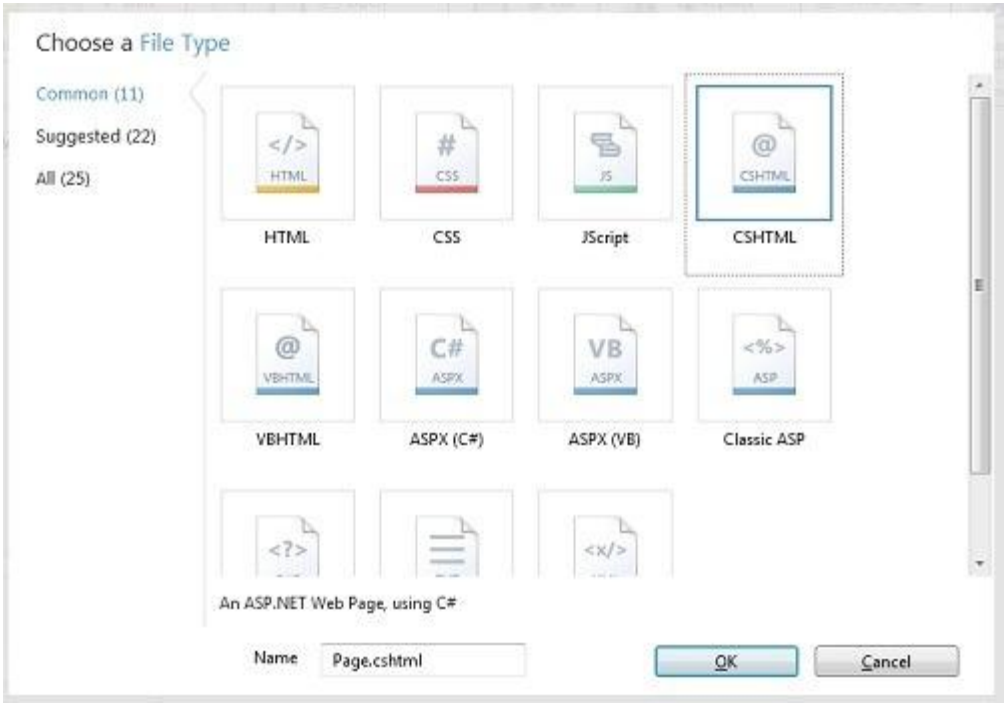
Válasszuk a WebMatrixban a *Files* munkafelületet! Ezzel a felülettel kezelhetjük fájljainkat és mappáinkat. A baloldali panelban láthatjuk a webhely fájljainak struktúráját.



1. Kattintsunk a szalagon a *New* gombra, azon belül a *New File* lehetőségre!



A WebMatrix különféle fájltípusokat ajánl fel. Legtöbbjük, például a HTML, CSS vagy a TXT már ismerős lehet.



2. Válasszuk a CSHTML-t, és a *Name* felirat mellé írjuk be a fájl nevét: *default.cshtml*. A CSHTML oldalak olyan speciális oldalak a WebMatrixban, melyek tartalmazhatnak hagyományos kódokat, mint HTML vagy JavaScript, de ezenfelül programkódokat is írhatunk weboldalunkhoz. (A CSHTML fájlokkal később részletesebben megismerkedünk.)
3. Kattintsunk az *OK* gombra! A WebMatrix létrehozza a fájlt, és megnyitja a szerkesztőben.

Láthatjuk, hogy ez egy hagyományos HTML kód.

4. Adjuk hozzá a következő címet, fejlécet és bekezdést az oldalhoz:

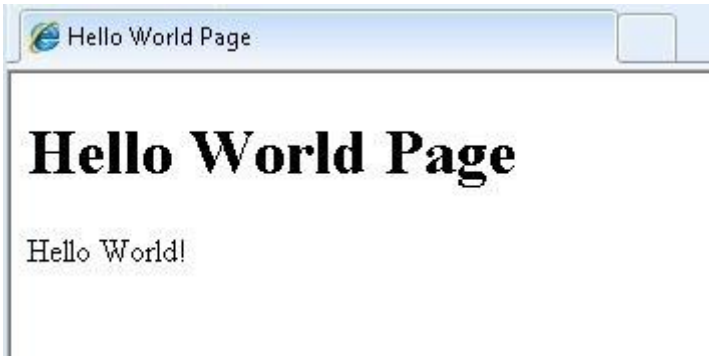
```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8" />
        <title>Hello World Page</title>
    </head>
    <body>
    <h1>Hello World Page</h1>
    <p>Hello World!</p>
    </body>
</html>
```

5. Gyorselérési eszköztáron mentsük munkánkat a *Save* gombra kattintva!

Kattintsunk a szalagon a *Run* gombra.

Megjegyzés: Mielőtt a *Run* gombra kattintva futtatjuk a weboldalt, győződjünk meg róla, hogy a futtatni kívánt weboldalunk van kiválasztva a *Files* felületen! A WebMatrix mindig azt az oldalt indítja el, amelyik ki van jelölve bal oldalon, akkor is, ha épp valamelyik másik oldalt szerkesztjük. Ha egy oldalt sem jelöltünk ki, a WebMatrix megpróbálja az oldal kezdőlapját (*default.cshtml*) futtatni, de ha nem talál ilyet, a böngésző hibát fog jelezni.

A WebMatrix elindít egy webszervert (IIS Express), melynek segítségével weboldalaink futását tesztelhetjük. Az oldal az alapértelmezett böngészőnkben nyílik meg.



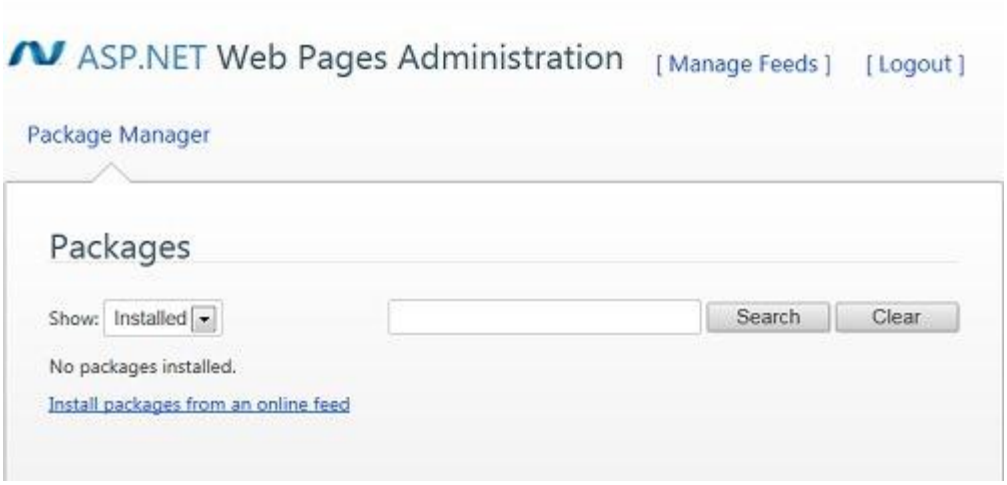
Helperek telepítése a Package Manager segítségével

Miután feltelepítettük a WebMatrixot, és létrehoztuk első webhelyünket, érdemes feltelepítenünk az ASP.NET Web Helpers Libraryt. Ez a gyűjtemény olyan helpereket (komponenseket) tartalmaz, melyek egyszerűbbé teszik a gyakori programozási feladatokat, és gyakran lesznek segítségünkre az útmutató során. (Néhány helper már automatikusan feltelepül az ASP.NET Web Pages telepítésével, de néhány továbbit kell telepítenünk más gyűjteményekből). Az elérhető helperek listáját a függelékben találjuk. A következő lépésekből megtudhatjuk, hogyan használjuk a *Package Manager* eszközt a Helper csomagok telepítéséhez.

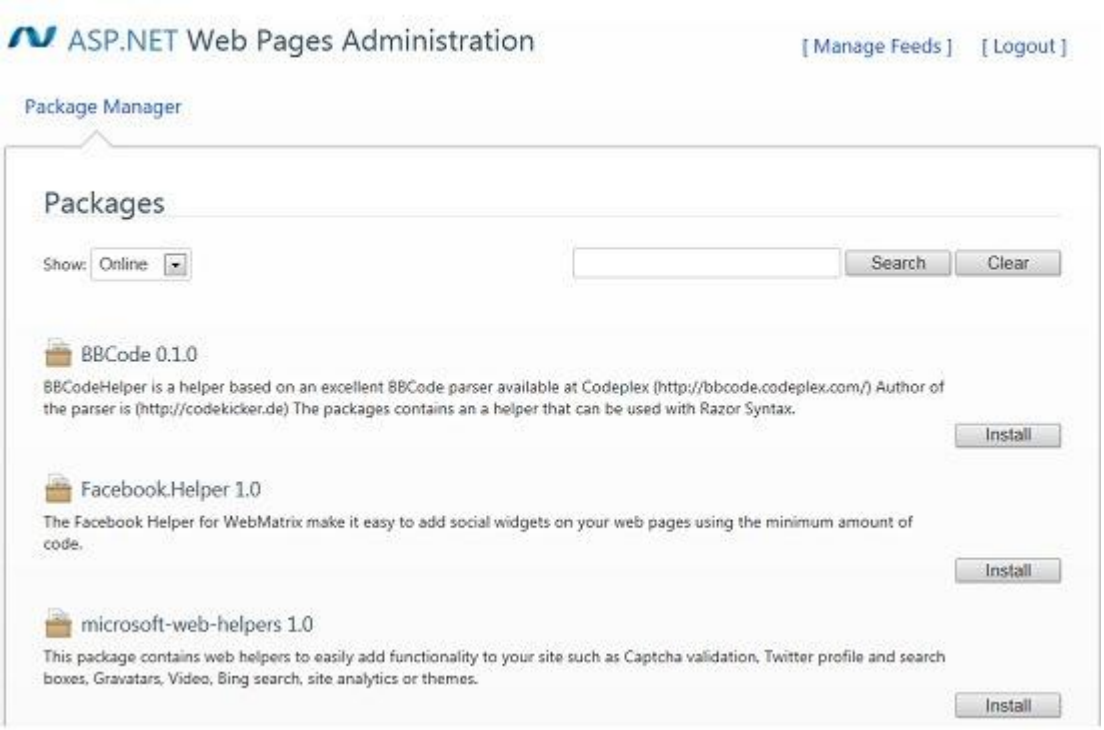
1. Futtassuk a korábban elkészített *default.cshtml* oldalunkat!
2. A böngésző címsorában töröljük ki a cím végéről a *default.cshtml* feliratot, és írjuk át *_Admin*-ra! Az eredmény legyen ehhez hasonló (nem baj, ha a port száma nem 8080):

http://localhost:8080/_Admin

3. Nyomjuk meg az Entert a cím megnyitásához! Mivel ez az első alkalom, hogy belépünk az *_Admin* oldalra, létre kell hoznunk egy jelszót.
4. Hozzuk létre a jelszót! Ha ezzel kész vagyunk, az *_Admin* oldal bejelentkezik, és láthatjuk a *Package Manager* oldalt. Alapértelmezésként a *Package Manager* minden telepített csomagot (package) mutat.



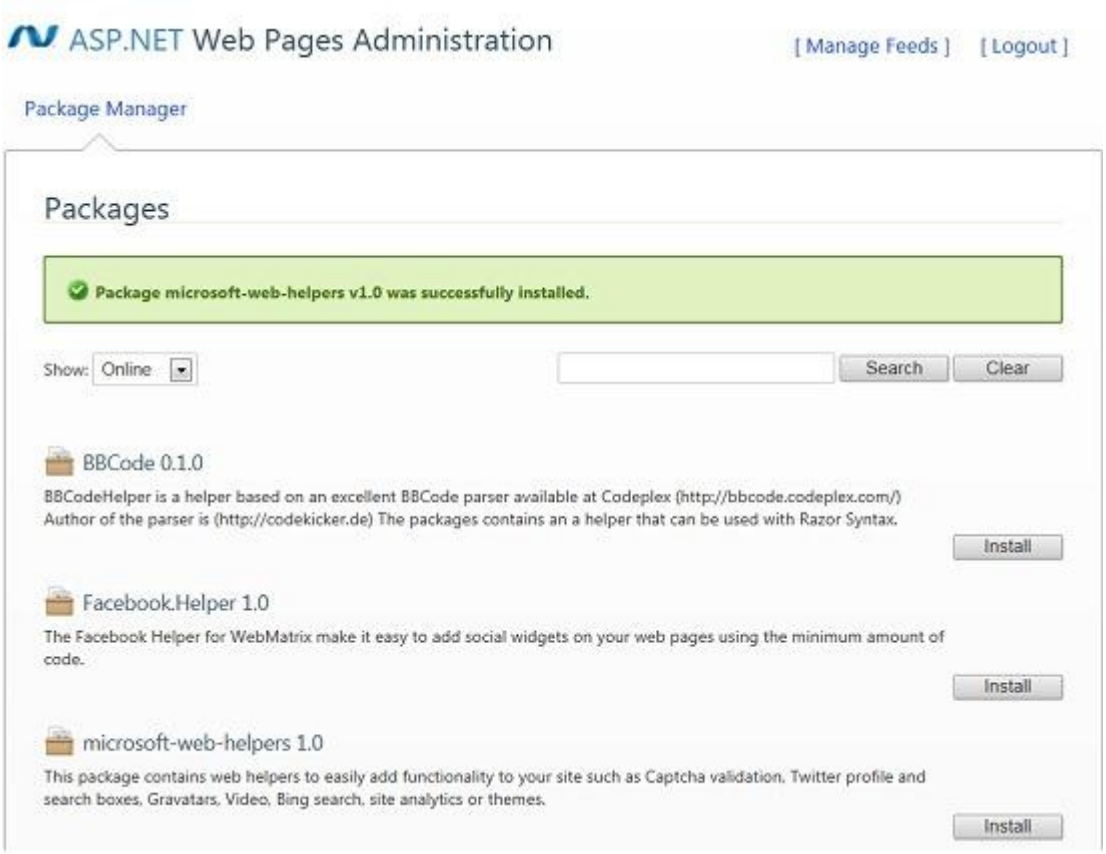
5. Kattintsunk a *Show* felirat melletti legördülő menüre, és válasszuk az *Online* lehetőséget! Erre a lehetőségre kattintva egy listát kapunk a telepíthető csomagokról. (Ha szeretnénk további forrásokat megtekinteni, kattintsunk a *Manage Feeds* linkre, ahol hozzáadhatunk, módosíthatunk vagy törölhetünk forrásokat.)



Megjegyzés: Válasszuk a legördülő menüben az *Updates* lehetőséget a már telepített csomagok elérhető frissítéseinek megjelenítéséhez!

6. Kattintsunk a *microsoft-web-helpers* felirat melletti *Install* gombra az ASP.NET Web Helpers Library telepítéséhez. A megnyíló oldalon részletesebb információkat kapunk a csomagról, illetve láthatjuk a csomag licencszerződését.
7. A részleteket leíró oldalon kattintsunk ismét az *Install* gombra a telepítéshez!

   Amikor a csomag feltelepült, a Package Manager jelzi a változást. Ha valami oknál fogva el szeretnénk távolítani az ASP.NET Helpers Libraryt, ezen az oldalon tehetjük meg.



8. Ismételjük meg a 6. és 7. lépéseket a Facebook.Helper telepítéséhez! A Facebook helperrel később részletesebben megismerkedünk.

   A következő fejezetben megtudhatjuk, milyen egyszerűen alakíthatjuk *default.cshtml* fájlunkat egy dinamikus weboldallá.

ASP.NET weboldal kódok használata

Most olyan oldalt fogunk készíteni, melyben egyszerű kódok segítségével jeleníthetjük meg a szerveren aktuális dátumot és időt az oldalon. Ez a példa bemutatja a Razor szintaxis használatát, mellyel kódokat illeszthetünk az ASP.NET weboldalak HTML kódjába. (Ezzel részletesebben a következő fejezetben foglalkozunk.) A most következő kód bemutatja az egyik helpert, melyről korábban olvashattunk.

1. Nyissuk meg a *default.cshtml* fájlt!
2. Készítsük el az oldalunkat a példának megfelelően:

```
<!DOCTYPE html>
<html lang="en">
    <head>
```

```
        <meta charset="utf-8" />
        <title>Hello World Page</title>
    </head>
    <body>
    <h1>Hello World Page</h1>
    <p>Hello World!</p>
    <p>The time is @DateTime.Now</p>
    </body>
</html>
```

Az oldal megegyezik a hagyományos HTML jelölésrendszerrel, egy kivétellel: a @ karakter jelöli az ASP.NET programkódokat.

3. Mentsük el az oldalt, és indítsuk el a böngészőben! Láthatjuk a pontos dátumot és időt.

Ez az egyszerű kódsor elvégezte az összes munkát: kiderítette a pontos időt, kiválasztotta a megjelenés formátumát, és elküldte azt a böngészőbe. (A formátum egyszerűen módosítható, ez csak az alapértelmezett.)

Tegyük fel, hogy bonyolultabb kódot szeretnénk beszúrni, például, hogy görgessük folyamatosan egy kiválasztott Twitter felhasználó bejegyzéseit. Ehhez használhatunk helpert, ami, mint korábban olvashattuk, egyszerűbbé teszi a gyakori feladatokat – ebben az esetben a Twitter bejegyzések begyűjtését és helyes megjelenítését.

1. Hozzunk létre egy új CSHTML fájlt *TwitterFeed.cshtml* néven!
2. Az új oldal tartalma legyen a következő:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8" />
        <title>Twitter Feed</title>
    </head>
    <body>
        <h1>Twitter Feed</h1>
        <form action="" method="POST">
         <div>
            Enter the name of another Twitter feed to display:
             
            <input type="text" name="TwitterUser" value=""/>
             
            <input type="submit" value="Submit" />
         </div>
         <div>
            @if (Request["TwitterUser"].IsEmpty()) {
                @Twitter.Search("microsoft")
```

```
            }
            else {
                @Twitter.Profile(Request["TwitterUser"])
            }
        </div>
    </form>
</body>
</html>
```

Ez a HTML kód olyan oldalt ír le, ahol egy beviteli mezőt találunk a felhasználó nevének beírásához, és egy *Submit* gombot a küldéshez. Ez az első címkepáros között található.

A következő páros között találunk néhány sor kódot. (Az ASP.NET weboldalaknál a @ karakterrel jelöljük a programkódokat). Amikor először jelenik meg a weboldal, vagy a felhasználó üresen hagyja a szövegdobozt, a Request["TwitterUser"].IsEmpty feltétel érvényesülni fog. Ebben az esetben az oldal a „microsoft” keresésre adott bejegyzéseket fogja listázni. A többi esetben a megadott felhasználónévhez tartozó bejegyzések láthatók.

3. Futtassuk az oldalt a böngészőnkben! A Twitterolvasó a „microsoft” kifejezést tartalmazó bejegyzéseket listázza.

4. Írjunk be egy Twitter felhasználónevet, és kattintsunk a Submit gombra! Megjelennek a felhasználó bejegyzései. (Ha nem létező felhasználónevet írunk be, a Twitterolvasó megjelenik, de üres lesz.)

Ez a példa egy kis betekintést adott abba, hogy hogyan használhatjuk a WebMatrixot, és hogyan készíthetünk ASP.NET alapú dinamikus weboldalakat a Razor szintaxis használatával.

A következő fejezet részletesebben foglalkozik a programkóddal. Az azt következő fejezetekben pedig megtudhatjuk, hogyan használhatunk programkódot különféle weboldalakkal kapcsolatos feladatok elvégzéséhez.

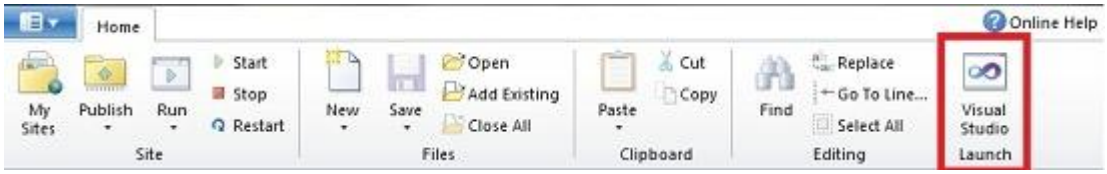
ASP.NET Razor oldalak programozása Visual Studióban

Az ASP.NET Razor oldalakat programozhatjuk a WebMatrix segítségével, de használhatjuk a Visual Studio 2010-et is, annak teljes verzióját, vagy az ingyenes Visual Web Developer Express editiont. Ha a Visual Studiot vagy a Visual Web Developert használjuk, két programozói eszköz teheti hatékonyabbá munkánkat – az IntelliSense és a debugger. Az IntelliSense a szerkesztőben működve jelenít meg a tartalomhoz kapcsolódó lehetőségeket. Ha például egy HTML elementet írunk be, az IntelliSense felajánlja az elementhez kapcsolódó attribútumokat, és azt is megmutatja, milyen értékekek adhatók meg az egyes elementekhez. Az Intellisense segítségünkre van HTML, JavaScript, C# és Visual Basic kód írásakor (Ezeken a nyelveken írjuk az ASP.NET weboldalakat.)

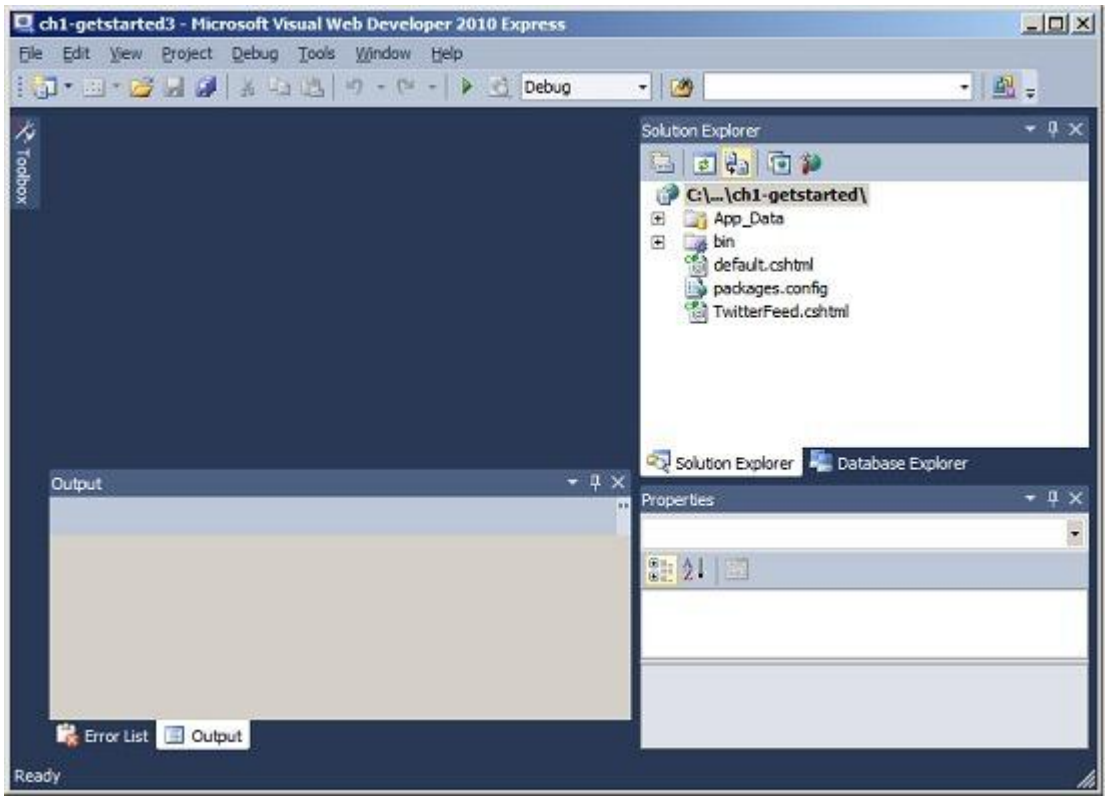
A debugger segítségével megállíthatunk egy programot futás közben. Ellenőrizhetjük, hogy az egyes változók milyen értékeket vettek fel, és soronként futtathatjuk a programot, hogy lássuk annak működését.

Ha a számítógépre fel van telepítve a Visual Studio, a WebMatrixban szerkesztett webhelyeket megnyithatjuk a Visual Studióban, hogy kihasználhassuk az Intellisense és a debugger nyújtotta lehetőségeket.

1. Nyissuk meg az ebben a fejezetben készített webhelyünket, és válasszuk a *Files* felületet!
2. Kattintsunk a szalagon a Visual Studio Launch gombra!



Miután a webhely megnyílik a Visual Studióban, láthatjuk annak felépítését a Visual Studio ablak Solution Explorer paneljában. A következő illusztrációban a Visual Web Developer 2010 Expresst láthatjuk, benne a létrehozott webhelyünkkel.



A Visual Studióban szerkesztett ASP.NET Razor oldalakhoz használható IntelliSenseről és debuggerről részletesebben olvashatunk a függelékben (ASP.NET Weboldalak programozása Visual Studioban).

ASP.NET oldalak létrehozása és tesztelése saját szövegszerkesztővel

Nem muszáj a WebMatrix Editort használnunk ASP.NET weboldalak létrehozásához és teszteléséhez. Egy oldal létrehozásához bármilyen szövegszerkesztőt használhatunk, beleértve a Notepadet. Az egyetlen, amire oda kell figyelnünk, hogy *.cshtml* kiterjesztéssel mentsük az oldalakat (Visual Basic esetén *.vbhtml* kiterjesztéssel).

A .cshtml oldalak tesztelésének legegyszerűbb módja az, hogyha elindítunk egy webszervert (IIS Express) a WebMatrixban található *Run* gomb segítségével. Ha nem szeretnénk használni a WebMatrixot, elindíthatjuk a webszervert a parancssorból is, és hozzárendelhetünk egy adott portszámot. Ezután be kell írnunk a portszámot a címsorba, amikor *.cshtml* fájlokat szeretnénk megtekinteni a böngészőben.

A Windowsban nyissuk meg a parancssort rendszergazdai jogosultságokkal, és lépjünk a következő mappába:

C:\Program Files\IIS Express

64-bites rendszer esetén a következő mappát nyissuk meg:

C:\Program Files (x86)\IIS Express

Írjuk be a következő parancsot, a webhelyünk elérési útjának megfelelően!

iisexpress.exe /port:35896 /path:C:\BasicWebSite

Tetszőleges portot választhatunk, csupán arra kell figyelnünk, hogy ne olyan legyen, melyet más program már használ. (Az 1024 fölötti portszámok általában szabadok.)

A path értékét úgy adjuk meg, hogy a tesztelni kívánt *.cshtml* fájlokat tartalmazó mappára mutasson!

A parancs lefuttatása után megnyithatjuk és böngészhetjük *.cshtml* fájljainkat, ha a böngésző címsorába beírjuk:

http://localhost:35896/default.cshtml

Az IIS Express parancssori funkcióinak megismeréséhez írjuk be a parancssorba:

iisexpress.exe /?

3. fejezet – Bevezetés az ASP.NET webszerkesztésbe Razor szintaxissal

Ebben a fejezetben megismerkedhetünk az ASP.NET weboldalak programozásával és a Razor szintaxissal. Az ASP.NET a Microsoft által kifejlesztett technológia webszervereken futó dinamikus weboldalakhoz.

A fejezet tartalma:

- 8 tipp az induláshoz.
- Az alapvető programozási fogalmak, amelyekre szükségünk lesz a továbbiakban.
- Mi az ASP.NET szerverkód és a Razor szintaxis lényege?

8 fő programozási tipp

Ebben a részben összefoglaltuk a legfontosabb tudnivalókat ahhoz, ahogy elkezdhessünk ASP.NET szerver kódot írni Razor szintaxissal.

> **Megjegyzés**: A Razor syntax a C# programozási nyelvre épül, amelyet ebben a könyvben végig használunk. Azonban a Razor syntax a Visual Basic nyelvet is támogatja, így a könyvben leírtakat Visual Basicben is használhatunk. Részletesebb információt a függelék Visual Basic progamozási nyelv és szintaxisa része tartalmaz.

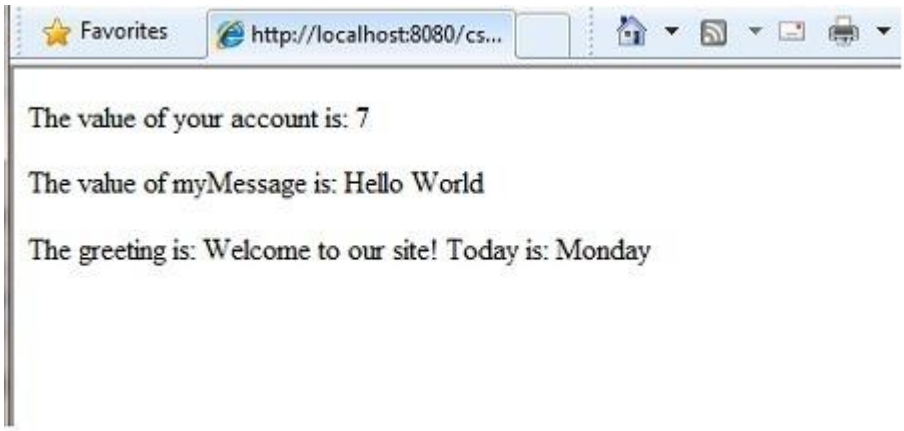
Ezen programozási technikák többségéről további részleteket találhatunk a fejezetben tovább haladva.

1. A @ karakter jelöli a kódrészleteket

A @ jel segítségével beágyazott kifejezéseket és egy- vagy többsoros parancsblokkok kezdetét jelezzük.

```
<!-- Egysoros parancsblokkok  -->
@{ var total = 7; }
@{ var myMessage = "Hello World"; }

<!-- Beágyazott kifejezések -->
<p>The value of your account is: @total </p>
<p>The value of myMessage is: @myMessage</p>

<!-- Többsoros parancsblokkok -->
@{
    var greeting = "Welcome to our site!";
    var weekDay = DateTime.Now.DayOfWeek;
    var greetingMessage = greeting + " Today is: " + weekDay;
}
<p>The greeting is: @greetingMessage</p>
```

Az eredmény így jelenik meg a böngészőben:



HTML kódolás

Ahogy az előző példákban láthattuk, a @ jel használatakor az ASP.NET ú.n. HTML kódolja a kimenetet: A HTML-ben speciális jelentést hordozó karaktereket (mint például a <,> és az &) lecseréli a HTML szabványban rögzített helyettesítő karaktersorozatokra, így azokat a böngésző kirajzolja a képernyőre, és nem kódnak értelmezi. HTML kódolás nélkül a kódunk kimenete bizonyos esetekben nem jelenne meg jól, sőt, akár biztonsági rés is keletkezhetne a weboldalunkon.

Ha HTML kódot szeretnénk megjeleníteni (pl `<p></p>` a bekezdésekhez vagy `<em></em>`, hogy kiemeljünk egy szöveget), lásd: Szöveg, Markup és Kód egyesítése a Kódblokkban.

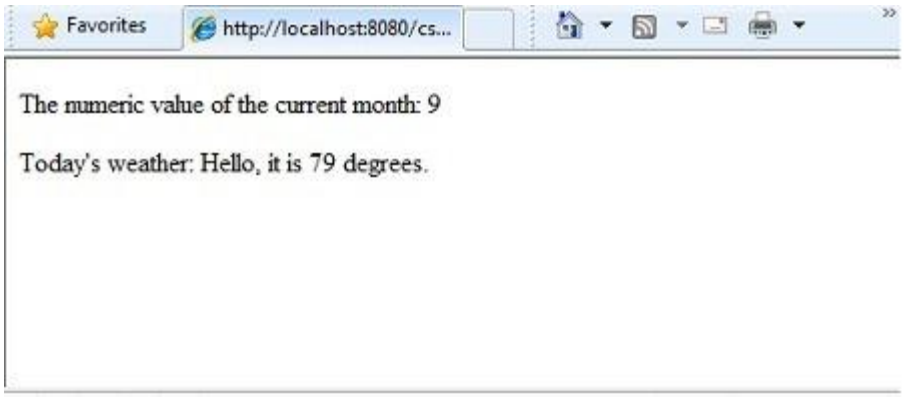
Részletesebb információt a HTML kódolásról a 5. fejezetben (Munka az űrlapokkal) olvashatunk.

2. A kódblokkokat kapcsos zárójelek közé zárjuk

Egy kódblokk egy vagy több utasítást tartalmaz, és ezeket kapcsos zárójelek közé írjuk

```
<!-- Egysoros parancsblokk.  -->
@{ var theMonth = DateTime.Now.Month; }
<p>The numeric value of the current month: @theMonth</p>

<!-- Többsoros parancsblokk. -->
@{
    var outsideTemp = 79;
    var weatherMessage = "Hello, it is " + outsideTemp + " degrees.";
}
<p>Today's weather: @weatherMessage</p>
```

Az eredmény így jelenik meg a böngészőben:



3. Az utasításokat pontosvesszővel zárjuk le

Blokkon belül minden utasítás pontosvesszőre kell, hogy végződjön. A beágyazott kifejezések nem végződnek pontosvesszővel.

```
<!-- Egysoros parancsblokk.  -->
@{ var theMonth = DateTime.Now.Month; }

<!-- Többsoros parancsblokk. -->
@{
    var outsideTemp = 79;
    var weatherMessage = "Hello, it is " + outsideTemp + " degrees.";
}

<!-- Sorközi kifejezés, itt nincsen pontosvessző. -->
<p>Today's weather: @weatherMessage</p>
```

4. Értékek tárolására változókat használunk

Értékeket a változókban tárolhatunk. Ezek lehetnek stringek (szövegek), számok, dátumok stb. Új változót a `var` kulcsszóval hozhatunk létre a parancsblokkokban. A @ jel segítségével a változó értékét egyszerűen beágyazhatjuk a HTML kódba.

```
<!-- String tárolása -->
@{ var welcomeMessage = "Welcome, new members!"; }
<p>@welcomeMessage</p>

<!-- Dátum tárolása -->
@{ var year = DateTime.Now.Year; }

<!-- Egy változó megjelnítése -->
<p>Welcome to our new members who joined in @year!</p>
```

Az eredmény így jelenik meg a böngészőben:

5. A szöveget tartalmazó stringeket kettős idézőjelek közé írjuk

A *string* egy karaktersorozat, amit szövegként kezelünk. Ahhoz, hogy megkülönböztessük a szöveget a programkódtól, idézőjelek közé kell tegyük:

```
@{ var myString = "This is a string literal"; }
```

A C# kódban így elhelyezett szöveg (*string literal*) esetén a visszaperjelnek (\) és dupla idézőjelnek speciális jelentése van. Ezért ha visszapert tartalmazó szöveget szeretnénk elhelyezni a C# kódban, akkor a *verbatim string literal* jelölést kell használunk: az első idézőjel elé egy @ karaktert kell elhelyezni.

```
<!-- Visszaperjel beágyazása stringbe -->
@{ var myFilePath = @"C:\MyFolder\"; }
<p>The path is: @myFilePath</p>
```

Ha a szöveg maga idézőjelet tartalmaz, azt meg kell különbözetnünk a string literalt lezáró idézőjeltől. Verbatim string literal jelölés esetén két egymást követő idézőjel felel meg egy „szó szerinti" idézőjelnek.

```
<!-- Dupla idézőjel beszúrása egy stringbe-->
@{ var myQuote = @"The person said: ""Hello, today is Monday."""; }
<p>@myQuote</p>
```

Az eredmény így jelenik meg a böngészőben:

Megjegyzés: A @ karakternek több jelentése van: az ASP.NET oldalakban a programkód kezdetét jelöljük vele, míg a C# programkódon belül a *verbatim string literal* kezdetét jelöli. Erről bővebben az alábbi oldalon tájékozódhatunk: http://msdn.microsoft.com/en-us/library/aa691090(VS.71).aspx

6. A kód érzékeny a kisbetűkre és nagybetűkre.

A C#-ben a kulcsszavak (`var`, `true`, `if`) és a változó nevek érzékenyek a kis- és nagybetűkre. A következő kódsorok két különböző változót hoznak létre: `lastName` és `LastName`.

```
@{
var lastName = "Smith";
var LastName = "Jones";
}
```

Ha egy változót `var lastName = "Smith"` utasítással deklaráljuk később a `@LastName` kifejezéssel próbálunk hivatkozni rá, akkor hibát kapunk, a `LastName` nem felismerhető.

Megjegyzés: Visual Basic nyelv nem érzékeny a kis-nagybetűkre. További információkat a függelékben (A Visual Basic programozási nyelv és szintaxisa) olvashatunk.

7. A kód nagy része objektumokkal foglalkozik

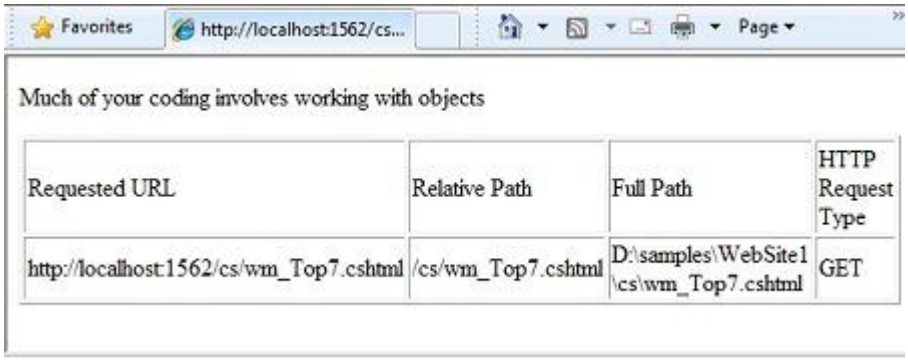
Objektumokkal reprezentáljuk azokat a dolgokat, amiket a programunkkal manipulálni szeretnénk (pl. oldal, szövegdoboz, fájl, kép, webes lekérés, e-mail, az datbázis egy sora stb.) Egy objektum jellemzőit a tulajdonságain keresztül tudhatjuk meg. Például egy szövegdoboznak van többek közt egy `Text` (szöveg) tulajdonsága; egy lekérésnek van egy `Url` tulajdonsága; egy e-mailnek van `From` (feladó) tulajdonsága; egy ügyfél objektumnak pedig `FirstName` (keresztnév) tulajdonsága. Az objektumok továbbá rendelkeznek metódusokkal is, ezek a műveletek, melyekre az objektum képes. Erre jó példa a fájl objektum `Save (mentés)` metódusa, egy kép objektum `Rotate` (elforgatás) metódusa, vagy egy e-mail objektum `Send` (küldés) metódusa.

Sokszor fogjuk használni a `Request` (lékérés) objektumot, ami egy elküldött űrlap mezőiben található értékeket adja vissza; esetleg azt, hogy milyen típusú böngésző indította a lekérést; a lekért oldal

URL-jét, vagy éppen azonosítja a felhasználót, és még sok minden másra is képes. A következő példa bemutatja, hogyan érjük el a `Request` objektum tulajdonságait, és hogyan hívjuk meg a `MapPath` nevű metódusát. Ezzel megtudhatjuk egy lap szerveroldali teljes elérési útját.

```
<table border="1">
<tr>
    <td>Requested URL</td>
    <td>Relative Path</td>
    <td>Full Path</td>
    <td>HTTP Request Type</td>
</tr>
<tr>
    <td>@Request.Url</td>
    <td>@Request.FilePath</td>
    <td>@Request.MapPath(Request.FilePath)</td>
    <td>@Request.RequestType</td>
</tr>
</table>
```

Az eredmény így jelenik meg a böngészőben:



8. A kódban elágazásokat helyezhezünk el

A dinamikus weboldalak egyik kulcs tulajdonsága, hogy képesek vagyunk eldönteni, mi történjen különböző helyzetekben. Ennek a legáltalánosabb módja az `if` parancs (és az opcionális `else` parancs)

```
@{
   var result = "";
   if(IsPost)
   {
      result = "This page was posted using the Submit button.";
   }
   else
   {
      result = "This was the first request for this page.";
   }
}

<!DOCTYPE html>
<html>
    <head>
        <title></title>
    </head>
<body>
<form method="POST" action="" >
  <input type="Submit" name="Submit" value="Submit"/>
  <p>@result</p>
</form>
</body>
</html>
```

```
    </body>
</html>
```

Az `if(IsPost)` utasítás az `if(IsPost == true)` rövidítése. Az `if` utasítással sokféle módon tesztelhetünk különféle eseteket, ismételhetünk kód blokkokat. Ezekről a későbbiekben többet megtudhatunk.

Az eredmény egy böngészőben megjelenítve (a *Submit* gomb megnyomása után):

HTTP GET és POST metódusok és az IsPost tulajdonság

A weblapok lekéréséhez használt HTTP protokoll szerint a böngésző alapvetően kétféleképpen fordulhat a szerverhez. GET esetén elsősorban tartalmat kér le (pl. egy weboldalt, egy kép fájlt), míg POST esetén tartalmat küld a szervernek (pl. egy kitöltött űrlap tartalmát). Általában, amikor a felhasználó először lekér egy oldalt, a lekérdezés a GET metódust használja. Ha a felhasználó kitölt egy űrlapot és ráklikkel a *Küldés* gombra, akkor a böngésző általában egy POST kérést küld a szervernek.

A web programozásban nem árt tudnunk, hogy egy oldalt éppen GET vagy POST metódussal kérnek le, hogy eldönthessük, hogyan dolgozzuk fel az oldalt. Az ASP.NET weboldalaknál az `IsPost` tulajdonságból tudjuk kiolvasni, hogy a lekérés GET vagy POST típusú. Ha a lekérés POST típusú, akkor az `IsPost` tulajdonság visszatérési értéke true (igaz) lesz. Ebben az esetben fogjuk kiolvasni az űrlapmezők tartalmát a lekérésből. Ebben a könyvben sok olyan példát találunk, ami bemutatja, hogyan dolgozzunk fel egy weboldalt az `IsPost` tulajdonságtól függően.

Egyszerű példakódok

A következő példákból megismerhetjük az alapszintű programozási technikákat. Ebben a példában egy olyan weboldalt készítünk, amiben a felhasználó megad két számot, majd az oldal összeadja azokat, és megjeleníti az eredményt.

1. Hozzunk létre egy *AddNumbers.cshtml* fájlt!
2. Másoljuk bele a következő kódot és jelöléseket, felülírva bármit, amit az oldal tartalmazott!

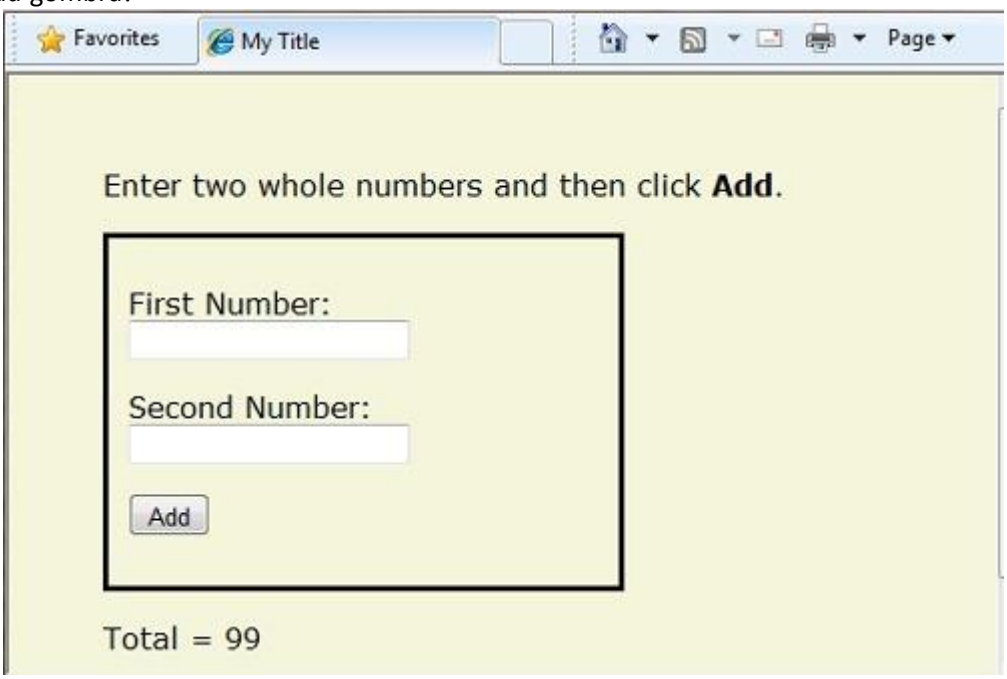
```
@{
    var total = 0;
    var totalMessage = "";
    if(IsPost) {

        // Retrieve the numbers that the user entered.
        var num1 = Request["text1"];
        var num2 = Request["text2"];

        // Convert the entered strings into integers numbers and add.
        total = num1.AsInt() + num2.AsInt();
```

```
        totalMessage = "Total = " + total;
    }
}

<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Add Numbers</title>
    <meta charset="utf-8" />
    <style type="text/css">
      body {background-color: beige; font-family: Verdana, Arial;
            margin: 50px; }
      form {padding: 10px; border-style: solid; width: 250px;}
    </style>
  </head>
<body>
  <p>Enter two whole numbers and then click <strong>Add</strong>.</p>
  <form action="" method="post">
    <p><label for="text1">First Number:</label>
      <input type="text" name="text1" />
    </p>
    <p><label for="text2">Second Number:</label>
      <input type="text" name="text2" />
    </p>
    <p><input type="submit" value="Add" /></p>
  </form>

  <p>@totalMessage</p>

</body>
</html>
```

Figyeljünk oda a következőkre:

- A @ karakter kezdi az első kódblokkot az oldalon, és megtalálható az oldal alján lévő `totalMessage` változó előtt is.
- Az oldal tetején lévő blokkot kapcsos zárójelek veszik körül.
- A fenti blokkban az összes sor pontosvesszőre végződik.
- A `total`, `num1`, `num2`, és `totalMessage` változók számokat, illetve egy stringet tárolnak.
- A `totalMessage` változóban eltároladnó szöveget idézőjelek közé tesszük
- Mivel a kód érzékeny a kis- és nagybetűkre, az oldal alján található `totalMessage` változó nevét pontosan ugyanúgy írjuk, mint a lap tetején!
- A `num1.AsInt() + num2.AsInt()` kifejezés megmutatja, hogyan dolgozzunk objektumokkal és metódusokkal. Az `AsInt` metódus egész számmá (integer) alakítja a felhasználó által bevitt karaktersorozatot (string), így végezhetünk vele matematikai műveleteket.
- A `<form>` címke tartalmazza a `method="post"` attríbútumot. Ez utasítja a böngészőt, hogy amikor a felhasználó az *Add* gombra kattint, az oldalt HTTP POST módszerrel küldje el a szervernek. A szerver oldalon ezért az `if(IsPost)` teszt igaz értéket ad, és a megadott kód lefut, amely megjeleníti a számok összeadásából adódó eredményt.

3. Mentsük el az oldalt és futtassuk böngészőben! (Győződjünk meg róla, ez a kijelölt oldal a *Files* munkaterületen, mielőtt futtatnánk.) Vigyünk be két egész számot, majd kattintsunk az *Add* gombra!



Programozási alapfogalmak

Amint azt az 2. fejezetben is láttuk, nem kell kétségbe esni, ha ezelőtt még sosem programoztunk. A WebMatrixszal, ASP.NET weboldalakkal és a Razor szintaxissal gyorsan tudunk kifinomult dinamikus weboldalakat készíteni, és nem kell túl sok kódot írnunk a kívánt eredményhez.

Ez a fejezet áttekintést ad az ASP.NET webprogramozásról. Ez nem egy részletes elemzés, csak egy gyors áttekintés a leginkább használt programozási fogalmakról. A most megszerzett tudás lefedi azokat a területeket, amire szükségünk lesz a könyv olvasása során.

Először ismerjük meg a technikai hátteret!

A Razor Syntax, Server Code és ASP.NET

A Razor szintaxis egy egyszerű programozási szintaxis, mellyel szerver oldali kódot ágyazhatunk be egy weboldalba. Egy, a Razor szintaxisban megírt weboldal kétféle tartalomból áll: a kliens oldalnak szánt tartalomból és a szerver oldalon futtatandó kódból. A kliensoldali tartalom azokat a dolgokat tartalmazza, amiket már megszokhattunk a weboldalaknál: HTML tagek, stílus információk (pl. CCS), szkriptek (pl. JavaScript), és egyszerű szövegek.

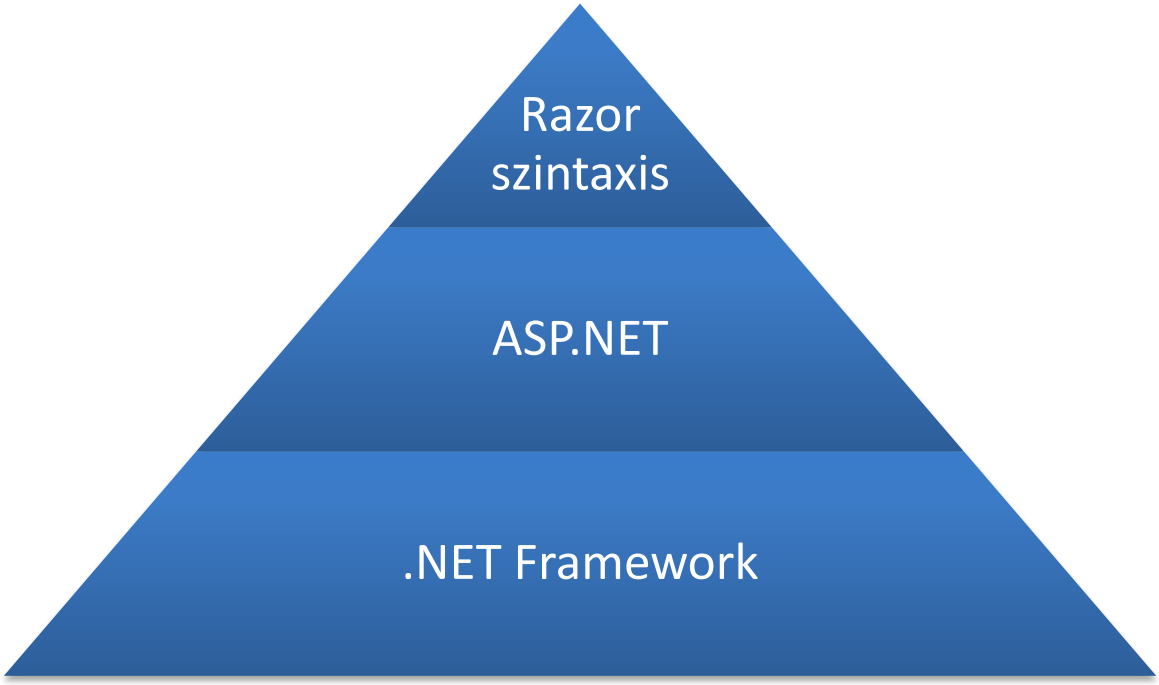
A Razor szintaxis lehetővé teszi, hogy szerverkódot szúrjunk a kliensoldali tartalom közé. Ha van szerverkód az oldalon. akkor a szerver először végigfuttatja a kódot, és csak azután küldi el az oldalt a böngészőbe. Azáltal hogy szerveren is fut, a kód sokkal összetettebb feladatokat tud végrehajtani, mint a kliensoldali tartalom önmagában, például a szerveren lévő adatbázishoz hozzáfér. Fontos, hogy a szerverkód dinamikusan tud kliens tartalmat készíteni – HTML kódot vagy más tartalmat tud generálni, amit később küld el a böngészőnek a statikus HTML tartalmakkal együtt. A böngésző szemszögéből nézve a szerverkód által generált kliens tartalom nem fog különbözni más kliens tartalmaktól. Mint láthatjuk, a szükséges szerverkód elég egyszerű.

A Razor szintaxissal működő ASP.NET weboldalaknak speciális kiterjesztésük van (.cshtml vagy .vbhtml). A szerver ezeket a kiterjesztéseket felismeri, futtatja a Razor szintaxissal megjelölt kódot, és csak ez után küldi el az oldalt a böngészőnek.

Mihez hasonlíthatjuk az ASP.NET-et?

A Razor szintaxis egy, a Microsoft által kidolgozott technikán, az ASP.NET-en alapul, aminek alapja a Microsoft.NET Framework. A .NET Framework egy nagy, átfogó programozó keretrendszer, amit a Microsoft fejlesztett ki, hogy szinte bármilyen számítógépes alkalmazást fejleszthessünk vele. Az ASP.NET a .NET Framework része, amelyet web alkalmazások fejlesztéséhez terveztek. A világ számos fejlesztője használja óriási, nagy forgalmú webhelyek készítéséhez az ASP.NET-et. (Ha egy webhelyen az *.aspx* kiterjesztést látjuk, tudjuk, hogy az oldal ASP.NET segítségével készült)

A Razor szintaxissal olyan hatékonyan dolgozhatunk, mint az ASP.NET-tel, de az egyszerűbb szintaxisnak köszönhetően kezdők számára könnyebben elsajátítható a használata, a gyakorlottabb felhasználók munkája pedig felgyorsul. Ez az egyszerű szintaxis egy családba tartozik az ASP.NET-tel és a .NET Frameworkkel, így amikor a webhelyünk összetettebbé válik, rendelkezésre állnak a nagy keretrendszerek megoldásai és szolgáltatásai.



Osztályok és példányok

Az ASP.NET kód objektumokkal dolgozik. Az objektumokkal kapcsolatos alapfogalom az osztályok (class). Az osztály az objektum definíciója vagy sablonja. Például egy alkalmazásban lehet olyan `Customer` (ügyfél) osztály, amely meghatározza az ügyfelekkel kapcsolatos tulajdonságokat és metódusokat.

Egy-egy ügyfél adatait leíró objektum az Ügyfél osztály egy-egy példánya. Minden példány ugyanazokat a tulajdonságokat és metódusokat tartalmazza, de a tulajdonságok értékei példányról példányra különböznek, hiszen minden ügyfélobjektum egyedi. Az egyik felhasználó objektumnak a `LastName` (vezetéknév) tulajdonsága lehet „Smith”, egy másikban pedig „Jones”.

Ehhez hasonlóan, bármely különálló weblap a webhelyen egy `Page` (Oldal) objektum, ami a `Page` osztály egyik példánya. A lapon található gomb egy `Button` (Gomb) objektum ami a `Button` osztály egyik példánya, és így tovább. Minden példánynak megvan a „maga jelleme”, de mindegyik az osztályában meghatározott „kaptafára készült”.

Nyelv és szintaxis

Az előző fejezetben egy egyszerű példán keresztül láthattuk, hogyan kell ASP.NET weboldalt készíteni, és hogyan adhatunk hozzá szerver oldali kódot a HTML kódhoz. A könyv ezen részében megtanuljuk, hogyan írjunk Razor szintaxist használó ASP.NET szerver kódot – és hogy mik a programnyelv szabályai.

Ha már van némi tapasztalatunk a programozás terén (főként, ha már programoztunk C-ben, C++-ban, C#-ban, Visual Basicben, vagy esetleg JavaScript-ben.) akkor az itt olvasható anyag nagyrészt ismerős lehet. Ha így van, akkor valószínűleg csak azt kell megszoknunk, hogy a *.cshtml* fájlokhoz hogyan adhatunk WebMatrix kódot.

Alap szintaxis

Szöveg, HTML és programkód keveréke a kódblokkban

A szerver oldali kódblokkokban gyakran szöveget vagy HTML kódot szeretnénk generálni a böngésző számára. Ha a kódblokkunk tartalmaz szöveget, melyet nem kódként szeretnénk felhasználni, akkor az ASP.NET-nek meg kell tudnia különböztetni a szöveget a kódtól. Ezt különböző módokon tehetjük lehetővé.

- Tegyük a szöveget HTML elemek közé, mint például `<p></p>` vagy `<em></em>`:

```
@if(IsPost) {
    // Ebben a sorban az egész szöveg páros <p> címkék között van.
    <p>Hello, the time is @DateTime.Now and this page is a
postback!</p>
} else {
    // A szöveg címkepárok között van, ezeket követi a szerverkód.
    <p>Hello <em>stranger</em>, today is: <br /> </p>  @DateTime.Now
}
```

  Egy HTML elem tartalmazhat szöveget, további HTML elemeket, és szerverkód kifejezéseket. Ha az ASP.NET érzékeli a nyitó HTML címkét, akkor lekódol mindent, amit az elem tartalmaz, és elküldi a böngészőnek (és persze végrehajtja a szerverkód utasításokat).

- Használjuk a `@:` operátort vagy a `<text>` elemet. A `@:` segítségével egy sor szöveget vagy párosítatlan HTML címkéket írathatunk ki. A `<text>` elem segítségével több sort is kiírathatunk. Ezek a módszerek akkor hasznosak, ha egy HTML elemet nem végrehajtani, hanem megjeleníteni szeretnénk.

```
@if(IsPost) {
    // Sima szöveget párosítatlan HTML címke és szerverkód követ.
    @: The time is: <br /> @DateTime.Now
    // Szerverkód, egyszerű szöveg, páros címkék, majd ismét szöveg.
    @DateTime.Now @:is the <em>current</em> time.
}
```

  Ha több sort, vagy párosítatlan HTML címkéket szeretnénk kiírni, minden sort a `@:` segítségével folytathatunk vagy a `<text>` elem segítségével csatolhatunk. Az ASP.NET a `@:` operátort, és a `<text>` címkét egyaránt szövegek felismeréséhez használja, ezek sosem jelennek meg a böngészőben.

```
@if(IsPost) {
    // Ismételje meg az előző példát, de most használja a <text>
címkét.
    <text>
    The time is: <br /> @DateTime.Now
    @DateTime.Now is the <em>current</em> time.
```

```
    </text>
}

@{
    var minTemp = 75;
    <text>It is the month of @DateTime.Now.ToString("MMMM"), and
    it's a <em>great</em> day! <br /><p>You can go swimming if it's
at
    least @minTemp degrees. </p></text>
}
```

A fenti első példa megegyezik az előzővel, csak itt `<text>` címkét használunk. A második példában használunk szöveget, párosítatlan HTML címkéket (`<br />`), programkódot és párosított HTML címkéket, amiket `<text>` és `</text>` közé zárunk. Természetesen a sorokat külön-külön írva is működne a kód, csak használni kellene a `@:` operátort minden sor előtt.

Megjegyzés: Ha a szöveget az itt leírtak szerint íratunk ki (kód blokkba helyezett HTML taggel, `@:` operátorral vagy `<text>` taggel), akkor az ASP.NET nem végez HTML kódolást (ahogyan korábban láthattuk).

Szóközök

Az extra szóközök a programkódban (az idézőjelek közötti string literalokon kívül) nem változtatják annak lefutását.

```
@{ var lastName = "Smith"; }
```

Az utasításban a sorköznek nincs hatása a utasításra, a jó olvashatóság érdekében tördelhetjük a sorokat A következő utasítások mind egyenértékűek a fentivel:

```
@{ var theName =
"Smith"; }

@{
    var
    personName
    =
    "Smith"
    ;
}
```

Arra vigyázzunk, hogy a string szövegeket nem tördelhetjük több sorba. A következő példa ezt a hibát mutatja be.

```
@{ var test = "This is a long
string"; } // Nem működik!
```

Ha esetleg mégis szükség lenne arra, hogy egy hosszú szöveget több sorba tördeljünk, kétféleképpen tehetjük meg. Használhatjuk a plusz (+) jelet, erről a fejezet egy későbbi részében olvashatunk. Használhatjuk a @ karaktert, hogy *verbatim string literalt* hozzunk létre, ahogy azt már korábban láttuk a fejezetben. Ezek tartalmazhatnak sortörést.

```
@{ var longString = @"This is a
long
string";
}
```

Megjegyzések a HTML és programkódban

A kódjainkban elhelyezhetünk megjegyzéseket magunknak vagy más fejlesztőknek. Ha egy kódrészletet átmenetileg ki akaránk törölni, átalakíthatjuk megjegyzéssé. Így a hatása nem érvényesül, de később bármikor újra felhasználhatjuk.

A HTML és a Razor kódban különbözőképpen jelöljük a megjegyzést. A Razor kód a szerveroldalon fut, ezért a megjegyzések még szerver oldalon eltávolításra kerülnek, így a felhasználók nem láthatják azokat a böngészőjükben.

ASP.NET Razor megjegyzést @* és *@ közé írunk, ezek lehetnek egysorosak vagy többsorosak:

```
@*  Egysoros megjegyzés. *@

@*
    Többsoros megjegyzés.
    Bármennyi sor hosszúságú lehet.
*@
```

Itt egy megjegyzés egy kódblokkban:

```
@{
    @* Ez egy megjegyzés. *@
    var theVar = 17;
}
```

Itt van ugyanaz a kódblokk úgy, hogy a kód sorát megjegyzésként jelöltük meg, ezért ez nem fog futni:

```
@{
    @* Ez egy megjegyzés. *@
    @* var theVar = 17;  *@
}
```

Egy kódblokk belsejében, a Razor megjegyzés szintaxisa mellett használhatjuk más programozó nyelv megjegyzés szintaxisát, például a C#-ét:

```
@{
    // Ez egy megjegyzés.
    var myVar = 17;
    /* Ez egy többsoros megjegyzés
    mely a C# megjegyzés szintaxisát használja. */
}
```

A C#-ban // karakter kezdi az egysoros megjegyzéseket, a többsoros megjegyzéseket pedig /* és */ közé írjuk. (Ahogy a Razor megjegyzéseket, úgy a C# megjegyzések sem jutnak el a böngészőig.)

HTML kódhoz készíthetünk HTML megjegyzést is:

```
<!-- Ez egy megjegyzés. -->
```

A HTML megjegyzéseket `<!--` és `-->` közé írjuk. A HTML megjegyzéseket nem csak szöveghez használhatjuk, de HTML kódhoz is. Ez a HTML megjegyzés az összes tartalmat és címkét eltűnteti, amit tartalmaz:

```
<!-- <p>Ez egy paragrafus.</p> -->
```

A Razor megjegyzésekkel ellentétben a HTML megjegyzéseket megkapja a böngésző (de nem jeleníti meg), így a felhasználók el tudják olvasni őket az oldal forrását megtekintve.

Változók

A változó egy adatok/értékek tárolására alkalmas objektum. Egy változó neve akármi lehet, csak arra kell figyelni, hogy betűvel kezdődjön, nem tartalmazhat szóközt és speciális jelentésű karaktereket.

Változók és adattípusok

A változóknak lehet egy megadott adattípusuk, ami megadja, hogy milyen adatokat tárolhatunk bennük. A string változókba szöveget tárolhatunk (pl. „Helló világ”), az integer változókban egész számokat tárolhatunk (pl. 3 vagy 79), és a date változókba dátum értékeket tárolhatunk különböző

formátumokban (pl. 4/12/2010 vagy Március 2009). Természetesen még sokféle adattípust használhatunk. Azonban egy változónak nem kötelező típust megadni. A legtöbb esetben az ASP.NET kitalálja a változóról, hogy milyen adatot tárolunk benne. (Esetenként meg kell adnunk egy típust, a könyvben található példákból kiderül, hogy mikor.)

Ahhoz, hogy változót hozzunk létre, használjuk a `var` kulcsszót (ha nem akarjuk meghatározni a típusát) vagy használjuk a típus nevét:

```
@{
    // Szöveget adunk meg változónak.
    var greeting = "Welcome!";

    // Számot adunk meg változónak.
    var theCount = 3;

    // Kifejezést adunk meg változónak.
    var monthlyTotal = theCount + 5;

    // Dátumot adunk meg változónak.
    var today = DateTime.Today;

    // A jelenlegi oldal Url-jét adjuk meg változónak.
    var myPath = this.Request.Url;

    // A változókat explicit változókkal deklaráljuk.
    string name = "Joe";
    int count = 5;
    DateTime tomorrow = DateTime.Now.AddDays(1);
}
```

A következő példában olyan beágyazott kifejezéseket láthatunk, amik felhasználják a weboldal változóit.

```
@{
    // Változók értékeinek beágyazása HTML markupba.
    <p>@greeting, friends!</p>

    // Változók használata beágyazott kifejezésben.
    <p>The predicted annual total is: @( monthlyTotal * 12)</p>

    // Az oldal URL-jének megjelenítése változóval.
    <p>The URL to this page is: @myPath</p>
}
```

Az eredmény így jelenik meg a böngészőben:

Adattípusok konvertálása és tesztelése

Annak ellenére, hogy az ASP.NET általában automatikusan meg tudja határozni az adattípusokat, néha mégsem sikerül. Ilyenkor besegítünk egy explicit átkonvertálással. Még akkor is, ha nincs szükség átkonvertálásra, néha hasznos lehet, ha tudjuk, hogy milyen típusú adatokkal dolgozunk.

A legáltalánosabb eset, hogy string-et konvertálunk át más típussá, például integerre, vagy date-re. A következő példa egy tipikus esetet mutat be, amiben string-ből számot csinálunk.

```
@{
    var total = 0;

    if(IsPost) {
        // A beírt számok beolvasása.
        var num1 = Request["text1"];
        var num2 = Request["text2"];
        // A stringek átalakítása számmá, majd azok összeadása.
        total = num1.AsInt() + num2.AsInt();
    }
}
```

A felhasználó minden adatot string-ként ad meg. Még akkor is, ha a felhasználótól számot kérünk, és ők azt is írtak be, az elküldött adat string lesz. Ebből adódik, hogy a számmá kell konvertálnunk. Mivel az ASP.NET nem tud két string-et összeadni, ha a példánkban össze akarnánk adni az értékeinket anélkül, hogy, átkonvertálnánk őket, a következő hibaüzenetet kapnánk.

Cannot implicitly convert type 'string' to 'int'.

Az értékek integerré konvertálásához az `AsInt` metódust használjuk. Ha a konvertálás sikeres, össze tudjuk adni a két számot.

A következő táblázat tartalmazza a változók gyakori konvertálásait a tesztelési metódusait.

Metódus	**Leírás**	**Példa**
`AsInt()`, `IsInt()`	Egy string-et ami egy egész számnak felel meg átkonvertál integerré.	`var myIntNumber = 0;` `var myStringNum = "539";` `if(myStringNum.IsInt()==true){` `myIntNumber =` `myStringNum.AsInt();` `}`
`AsBool()`, `IsBool()`	Átkonvertál egy igaz vagy hamis string-et boolean típussá.	`var myStringBool = "True";` `var myVar =` `myStringBool.AsBool();`
`AsFloat()`, `IsFloat()`	Egy string-ből ami egy tizedes törtet tartalmaz (pl. "1.3") float-ot (lebegő pontos) konvertál.	`var myStringFloat = "41.432895";` `var myFloatNum =` `myStringFloat.AsFloat();`
`AsDecimal()`, `IsDecimal()`	Átkonvertál egy tizedes tört string-et (1.3) egy decimális számmá. (Az ASP.NET-ben a decimális szám pontosabb, mint a float.	`var myStringDec = "10317.425";` `var myDecNum =` `myStringDec.AsDecimal();`
`AsDateTime()`, `IsDateTime()`	Átkonvertál egy string-et ami dátum és idő értékű	`var myDateString = "12/27/2010";` `var newDate =` `myDateString.AsDateTime();`

	az ASP.NET DateTime típussá.	
`ToString()`	Bármilyen adatot string-gé konvertál.	`int num1 = 17;` `int num2 = 76;` `// myString is set to 1776` `string myString =` `num1.ToString() +` `num2.ToString();`

Operátorok

Az operátor egy olyan kulcsszó vagy karakter, ami megmondja az ASP.NET-nek hogy milyen parancsot hajtson végre egy-egy kifejezésben. A C# nyelv (és az erre épülő Razor szintaxis is) sokféle operátort támogat, de csak párat kell ismernünk ahhoz, hogy, elkezdhessünk ASP.NET weboldalakat készíteni. A következő táblázatban megismerhetjük a leggyakoribb operátorokat.

Operátor	**Leírás**	**Példák**
`.`	Pont. Az objektum neve és az elérni kívánt tulajdonság vagy metódus neve közé írjuk.	`var myUrl = Request.Url;` `var count =` `Request["Count"].AsInt();`
`()`	Zárójelek. Kifejezéseket csoportosíthatunk velük és paramétereket adhatunk át metódusoknak.	`@(3 + 7)` `@Request.MapPath(Request.FilePa` `th);`
`[]`	Szögletes zárójelek. Tömbök, listák, gyűjtemények elemeit címezhetjük meg.	`var income =` `Request["AnnualIncome"];`
`=`	Értékadás. A kifejezés jobb oldalán található értéket a kifejezés bal oldalán található változónak értékül adjuk. (Vigyázat! Az = és az == operátor mást jelent!)	`var age = 17;`
`!`	Tagadás. Az igaz (true) értéket megfordítja hamis-ra (false) és fordítva. Általában gyors módszerként használjuk hamisság vizsgálatára.(vagyis tagadva igaz).	`bool taskCompleted = false;` `// Processing.` `if(!taskCompleted) {` `// Continue processing` `}`
`==`	Egyenlőség vizsgálata. Visszatérési értéke igaz (true), ha az értékek egyenlők.	`var myNum = 15;` `if (myNum == 15) {` `// Do something.` `}`
`!=`	Egynlőtlenség vizsgálata.	`var theNum = 13;` `if (theNum != 15) {`

	Visszatérési értéke (true), ha az értékek egynlők.	`// Do something.` `}`
< > <= >=	Kevesebb, mint, Nagyobb, mint, Kevesebb vagy egyenlő és Nagyobb vagy egyenlő.	`if (2 < 3) {` `// Do something.` `}` `var currentCount = 12;` `if(currentCount >= 12) {` `// Do something.` `}`
+	Matematikai operátorok, számolásoknál használjuk őket.	`@(5 + 13)` `@{ var netWorth = 150000; }` `@{ var newTotal = netWorth * 2;` `}` `@(newTotal / 2)`
+	Összeláncolás. String-eket fűzünk össze vele. Az ASP.NET az adat típusa szerint felismeri a különbséget eközött és a matematikai operátor között.	`// The displayed result is "abcdef".` `@("abc" + "def")`
&& ||	Logikai ÉS és VAGY művelet, amikkel feltételeket kötünk össze.	`bool myTaskCompleted = false;` `int totalCount = 0;` `// Processing.` `if(!myTaskCompleted &&` `totalCount < 12) {` `// Continue processing.` `}`
+= -=	A növelés és csökkentés operátorok. A jobb oldalon szereplő kifejezés értékét hozzáadjuk (kivonjuk) a bal oldalon szereplő változó érétkéhez.	`int theCount = 0;` `theCount += 1; // Adds 1 to count`

Fájlok és mappák használata a kódban

A kódjainkban sűrűn fogunk használni fájl és mappa elérési utakat. Ebben a példában egy weboldal fizikai mappastruktúráját láthatjuk, ahogy megjelenhet a weblapot fejlesztő számítógépen.

```
C:\WebSites\MyWebSite
    default.cshtml
    datafile.txt
    \images
        Logo.jpg
    \styles
        Styles.css
```

Egy webszerveren minden weboldalnak van egy virtuális mappa struktúrája, ami megfelel az oldalunk fizikai mappastruktúrájának. Alapértelmezetten a virtuális és a fizikai mappanevek megegyeznek. A virtuális gyökérkönyvtárat egy perjel (/) jelöli, pont úgy, mint ahogy a számítógépünkön C: meghajtó gyökérkönyvtárát fordított perjel (\) jelöli. (A virtuális mappa elérési utakat mindig sima perjellel jelöljük.) A következő példában bemutatjuk a virtuális és fizikai elérési útját az előző példában látható *Styles.css* fájlnak.

- Fizikai elérési út: *C:\WebSites\MyWebSiteFolder\styles\Styles.css*
- Virtuális elérési út (a virtuális gyökérkönyvtártól /): */styles/Styles.css*

Amikor fájlokkal és mappákkal dolgozunk a kódunkban, attól függően, hogy milyen objektumokat használunk, néha virtuális, néha fizikai elérési utakat használunk. Az ASP.NET-ben sokféle módszer áll lehetőségünkre hogy fájlokat és mappákat használjunk a kódunkban. Ezek lehetnek: a ~ operátor, a `Server.MapPath` metódus és a `Href` metódus.

A ~ operátor: a virtuális gyökérkönyvtár beolvasása

Hogy a kódunkban megadjuk a fájljaink és mappáink virtuális gyökérét, használjuk a ~ operátort. Ez azért hasznos, mert így szabadon mozgathatjuk a weboldalunkat, anélkül hogy a kódban szereplő elérési utakat megzavarnánk.

```
@{
    var myImagesFolder = "/images";
    var myStyleSheet = "/styles/StyleSheet.css";
}
```

A Server.MapPath metódus: A virtuális elérési út fizikaivá konvertálása

A Server.MapPath metódus egy virtuális elérési útból(pl */default.cshtm*) teljes elérési utat (pl *C:\WebSites\MyWebSiteFolder\default.cshtml*) konvertál. Ezt a metódust olyan feladatok elvégzésére használhatjuk, amik teljesítéséhez fizikai elérési útra van szükség, mint például egy, a webszerverünkön található szövegfájl olvasása vagy írása.(Egy kiszolgáló szerveren általában nem ismerjük weboldalunk teljes elérési útját.). A metódus úgy működik, hogy egy fájl vagy mappa virtuális elérési útját átadjuk neki és visszatérési értékként megadja a teljes utat.

```
@{
    var dataFilePath = "/dataFile.txt";
}
<!-- Megjeleníti a fizikai elérési utat:
C:\Websites\MyWebSite\datafile.txt -->
<p>@Server.MapPath(dataFilePath)</p>
```

A Href metódus: Elérési utak létrehozása az oldal forrásaihoz

A `WebPage` objektum `Href` metódusa átkonvertálja azokat az elérési utakat, amiket a szerverkódunkban hozunk létre (tartalmazhat ~ operátort), olyan elérési utakká amiket a böngészőnk értelmezni tud. (A böngésző nem tudja értelmezni a ~ operátort mivel az szigorúan ASP.NET operátor. Arra is használhatjuk még a `Href` metódust, hogy elérési utakat készítsünk olyan forrásokhoz, mint például képfájlok, más weboldal, és CSS fájlok. Például használhatjuk ezt a metódust arra, hogy egy HTML parancsban attribútumokat adjuk egy `<img>`,`<link>`, és egy `<a>` elemnek.

```
@{
    var myImagesFolder = "/images";
    var myStyleSheet = "/styles/StyleSheet.css";
}

<!-- Ez a kód ezt az elérési utat hozza létre "../images/Logo.jpg” az
src attribútumban. -->
<img src="@Href(myImagesFolder)/Logo.jpg" />

<!-- Ugyanaz, csak itt ~-t haszálunk -->
<img src="@Href("/images")/Logo.jpg" />

<!—Ez egy hivatkozást készít a CSS fájlra. -->
<link rel="stylesheet" type="text/css" href="@Href(myStyleSheet)" />
```

Logikai feltételek és ciklusok

ASP.NET kódban írhatunk olyan parancsokat, amik bizonyos feltételektől függenek, vagy akár olyan kódot, ami egy meghatározott alkalommal megismétli önmagát.

Feltételek tesztelése

Hogy egyszerű feltételeket teszteljünk, használjuk az `If` parancsot. Ennek visszatérési érteke lehet igaz vagy hamis a feltételünktől függően.

```
@{
  var showToday = true;
  if(showToday)
  {
    @DateTime.Today;
  }
}
```

A teszt blokk az `if` kulcsszóval kezdődik. A valódi tesztelés (feltétel) zárójelben van, ennek lehet a visszatérési értéke igaz vagy hamis. Az `if` parancs egy `else` ágat is tartalmazhat, arra az esetre, ha feltételünk visszatérési értéke hamis lenne.

```
@{
  var showToday = false;
  if(showToday)
  {
    @DateTime.Today;
  }
  else
  {
    <text>Sorry!</text>
  }
}
```

Egy `else if` blokk használatával akár több feltételt is megadhatunk.

```
@{
    var theBalance = 4.99;
    if(theBalance == 0)
    {
        <p>You have a zero balance.</p>
    }
    else if (theBalance  > 0 && theBalance <= 5)
    {
        <p>Your balance of $@theBalance is very low.</p>
    }
    else
    {
        <p>Your balance is: $@theBalance</p>
    }
}
```

A következő példában láthatjuk, hogy ha az `If` águnk feltétele nem teljesül, akkor az `else if` águnk feltételét vizsgáljuk meg. Ha ez a feltétel teljesül az `else if` ág fog lefutni, ha azonban egyik feltétel sem teljesül, akkor az `else` ág fog lefutni. Egy `If` ág mellé végtelenszámú `else if` ágat használhatunk, az a lényeg, hogy egy `else` ággal zárjuk le, ugyanis ez felel meg a ''bármi más'' feltételnek.

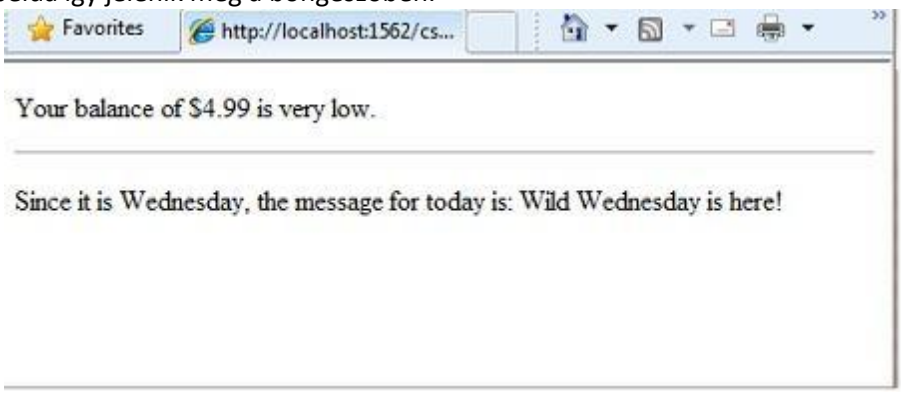
Sok feltétel esetén használjuk a `switch` blokkot:

```
@{
    var weekday = "Wednesday";
    var greeting = "";

    switch(weekday)
    {
        case "Monday":
```

```
                greeting = "Ok, it's a marvelous Monday";
                break;
            case "Tuesday":
                greeting = "It's a tremendous Tuesday";
                break;
            case "Wednesday":
                greeting = "Wild Wednesday is here!";
                break;
            default:
                greeting = "It's some other day, oh well.";
                break;
        }

        <p>Since it is @weekday, the message for today is: @greeting</p>
    }
```

Az ellenőrizendő változót zárójelezni kell (a példánkban a weekday változó). Minden esetben egy `case` kulcsszót használunk, amit egy kettőspont (:) követ. Ha valamelyik érték megegyezik a letesztelt értékkel, akkor abban a `case` ágban levő kód fut le. Minden `case` kulcsszót egy `break` kulcsszóval zárunk le. (Ha elfelejtjük beírni a `break` kulcsszót valamelyik `case` blokknál, akkor a következő `case` ág is le fog futni.) Egy `switch` blokknak rendszerint van `default` ága, hiszen a „bármi más” utolsó feltétel teljesül ha más eset nem lesz igaz.

Az utolsó két példa így jelenik meg a böngészőben:



Kód ismétlése (Ciklusok)

Sűrűn van arra szükség, hogy egy-egy parancsot többször is lefuttatassunk. Ezt ciklusok használatával érjük el. Például gyakran kell egy parancsot egy adattömb minden elemére lefuttassunk. Ha tudjuk, hányszor szeretnénk ismételni, akkor használjuk a `for` ciklust, mivel ez a ciklus nagyon hasznos ha előre vagy hátrafele szeretnénk számolni.

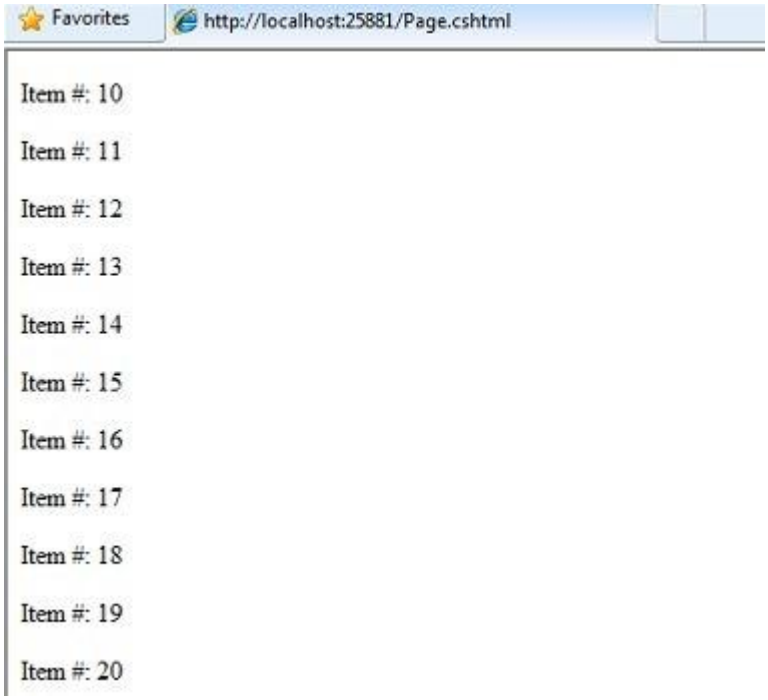
```
@for(var i = 10; i < 21; i++)
{
    <p>Line #: @i</p>
}
```

A ciklus a `for` kulcsszóval kezdődik, majd három zárójelben lévő, pontosvesszőre végződő parancs követi.

- A zárójelben az első parancsot (`var i=10;`) egy számlálót hoz létre és a 10-es értékre állítja be kezdésnek. A számlálót nem kell `i`-nek hívnunk – bármilyen legális változó nevet adhatunk neki. Amikor a `for` ciklus fut, a számláló automatikusan növekszik.
- A második paranccsal (`i < 21;`) azt állítjuk be, hogy meddig szeretnénk számolni. Ebben az esetben azt szeretnénk, hogy maximum 20-ig számoljon (azaz, hogy addig működjön, amíg a számláló kisebb, mint 21)

- A harmadik parancs (`i++`) egy inkremens operátort használ, ami egyszerűen eggyel növeli a számlálót minden lefutáskor.

A zárójeleken belül az a kód van, amely a ciklus minden iterációját lefutja. A kód minden alkalommal létrehoz egy új paragrafust (`<p>` elem) és kijelzi az `i` (számláló) jelenlegi értékét. Ha megnyitjuk ezt a példát, akkor 11 sornyi szöveget fogunk látni, amiben minden sor a változó egyesével növelt értékét mutatja.



Ha tömbökkel vagy listákkal dolgozunk, akkor sűrűn fogunk `foreach` ciklust használni. A lista hasonló objektumok gyűjteménye, és a `foreach` segítségével feladatokat hajthatunk végre minden elemén. Ez a típusú ciklus azért hasznos a listáknál, mert nincs szükség számlálóra, vagy beállított határra. Ehelyett a `foreach` végigmegy a listán elejétől a végéig.

A következőpélda visszaadja a `Request.ServerVariables` gyűjtemény elemeit (melyek információk a webszerverről). Arra használjuk a `foreach` ciklust, hogy a gyűjtemény minden elemének a nevét megjelenítsük egy HTML listában egy új `<li>` elemként.

```
<ul>
@foreach (var myItem in Request.ServerVariables)
{
    <li>@myItem</li>
}
</ul>
```

A `foreach` kulcsszót zárójelek követik, melyekben meghatározhatjuk a gyűjtemény egy elemét (a példában `var myItem`), melyet az in kulcsszó követ, majd a gyűjtemény, melyet végig akarunk vizsgálni. A `foreach` ciklus törzsében felhasználhatjuk a korábban deklarált változót annak kijelzésére.

Az általános felhasználású ciklusokhoz használjuk a `while` kifejezést:

```
@{
    var countNum = 0;
    while (countNum < 50)
    {
        countNum += 1;
        <p>Line #@countNum: </p>
    }
}
```

A `while` ciklus a `while` kulcsszóval kezdődik, majd a zárójelekben megadjuk, hogy a ciklus meddig folytatódjon (itt addig, amíg a `countNum` kevesebb, mint 50). A ciklusok egy számolásra használt változót vagy objektumot tipikusan vagy növelnek, vagy csökkentnek. A példánkban akárhányszor lefut a ciklus a `+=` operátor mindig hozzáad 1-et a `countNum` változóhoz.(Ha csökkenteni szeretnénk a változót akkor a `-=` operátort kéne használni.)

Objektumok és gyűjtemények

Az ASP.NET weboldalak szinte minden része objektum, még maga a weboldal is. Ebből a részből megismerhetjük a fontosabb objektumokat.

Oldal objektumok

Az ASP.NET –ben a legalapvetőbb objektum az oldal. Egy oldal tulajdonságait közvetlenül elérhetjük, anélkül, hogy bármilyen más objektumot kéne használnunk, a következő kód a `Request` objektum segítségével beolvassa az oldal elérési útját:

```
@{
    var path = Request.FilePath;
}
```

Hogy tisztázzuk, hogy a jelenlegi oldalon lévő objektumra utalunk, opcionálisan használhatjuk a `this` kulcsszót. Itt az előző kód példa, a `this` kulcsszóval:

```
@{
    var path = this.Request.FilePath;
}
```

A Page objektum tulajdonságaival sok információhoz juthatunk hozzá, mint például:

- `Request` (Kérelem): Ahogy már láthattuk ,ez a gyűjtemény információkat tárol az éppen aktuális lekérdezésről, azt is beleértve, hogy milyen böngésző indította a lekérdezést, mi a megtekinteni kívánt oldal URL-je, ki a felhasználó… stb.
- `Response` (Válasz:.Ez azoknak az információknak a gyűjteménye, amiket visszaküld az oldal a böngészőnek, amikor lefutott a szerveroldali kód. Használhatjuk ezt a tulajdonságot például arra, hogy információkat helyezünk el a válaszba.

```
@{
    // Access the page's Request object to retrieve the Url.
    var pageUrl = this.Request.Url;
}
<a href="@pageUrl">My page</a>
```

Gyűjtemény objektumok(Listák,tömbök)

Az ugyanolyan típusú objektumok csoportját nevezzük *gyűjteménynek*. Erre tökéletes példa a `Customer` (ügyfelek) objektumok gyűjteménye egy adatbázisból. Az ASP.NET-ben sok beépített gyűjteményt találhatunk, mint például a `Request.Files` gyűjtemény.

Két megszokott gyűjtemény típus a *tömb* és a „*szótárérték*”. A tömb akkor hasznos ha hasonló dolgokat akarunk együtt tárolni, de nem akarunk minden elemhez külön változót létrehozni.

```
@* Tömbblokk 1: Új tömb deklarálása kapcsoszárójelekkel. *@
@{
    <h3>Team Members</h3>
    string[] teamMembers = {"Matt", "Joanne", "Robert", "Nancy"};
    foreach (var person in teamMembers)
    {
        <p>@person</p>
    }
}
```

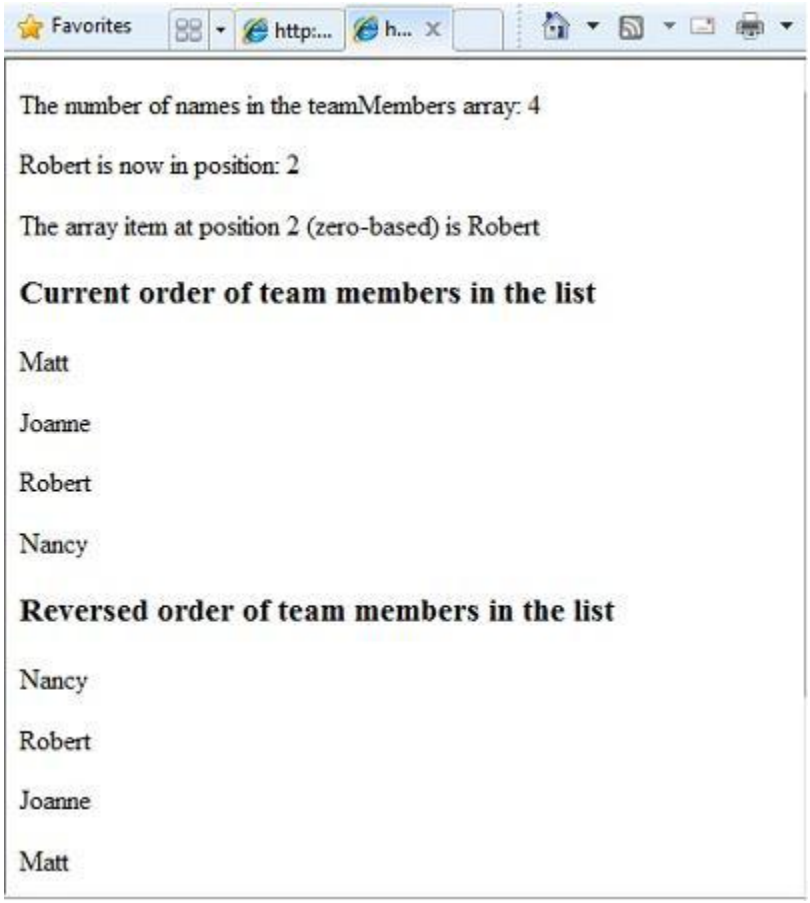
Minden tömbnek van egy megadott adattípusa, mint például `string`, `integer`, vagy `DateTime`. Hogy utaljunk arra, hogy tömbbel dolgozunk, szögletes zárójeleket adunk a deklaráláshoz (mint például `string[]` vagy `int[]`). A tömbben tárolt elemeket az indexüket használva érhetjük el, vagy egy `foreach` ciklussal. A tömbök indexei 0-val kezdődnek, vagyis a tömb első elemének az indexe 0 a másodiknak 1, és így tovább.

```
@{
    string[] teamMembers = {"Matt", "Joanne", "Robert", "Nancy"};
    <p>The number of names in the teamMembers array:
@teamMembers.Length </p>
    <p>Robert is now in position: @Array.IndexOf(teamMembers,
"Robert")</p>
    <p>The array item at position 2 (zero-based) is
@teamMembers[2]</p>
    <h3>Current order of team members in the list</h3>
    foreach (var name in teamMembers)
    {
        <p>@name</p>
    }
    <h3>Reversed order of team members in the list</h3>
```

```
        Array.Reverse(teamMembers);
        foreach (var reversedItem in teamMembers)
        {
            <p>@reversedItem</p>
        }
    }
```

Egy tömb hosszát, vagyis hogy hány elemű a `Length` tulajdonságából tudhatjuk meg. Ha szeretnénk egy megadott érték helyét meghatározni a tömbben akkor az `Array.IndexOf` metódust használjuk. De ha úgy akarjuk meg is fordíthatjuk a tömb tartalmát (`Array.Reverse` metódus) vagy akár rendezhetjük is (`Array.Sort` metódus).

A string tömbünk így jelenik meg a böngészőben:



A tár(szótárérték) nem más, mint kulcs/érték párok gyűjteménye, ezekben elég megadni a kulcsot (vagy nevet), hogy beállítsuk vagy beolvassuk a hozzátartozó értéket:

```
@{
    var myScores = new Dictionary<string, int>();
    myScores.Add("test1", 71);
    myScores.Add("test2", 82);
    myScores.Add("test3", 100);
    myScores.Add("test4", 59);
}
<p>My score on test 3 is: @myScores["test3"]%</p>
@(myScores["test4"] = 79)
<p>My corrected score on test 4 is: @myScores["test4"]%</p>
```

A `new` kulcsszóval jelezzük, hogy új tár (szótárérték) objektumot akarunk létrehozni.A `var` kulcsszóval egy változóhoz fűzhetjük a tárunkat. A tárban használt adattípusokat törtzárójelek (`<` `>`) közé kell

tennünk, és a deklarálás végére még egy zárójel párt kell tenni, mivel ez az, ami tulajdonképpen létrehoz egy új tárat.

Ahhoz hogy új elemeket adjunk hozzá a tárunkhoz, meghívhatjuk a tár `Add` metódusát ez változónál (esetünkben `myScores`), majd megadhatjuk hozzá a kulcs-érték párt. A következő példában láthatjuk az értékadás egy alternatíváját, amikor a kulcsot szögletes zárójelbe írjuk:

```
myScores["test4"] = 79;
```

Hogy értéket olvassunk ki a tárunkból, szögletes zárójelezzük az elérni kívánt kulcsot:

```
var testScoreThree = myScores["test3"];
```

Paraméterekkel rendelkező metódusok hívása

Ahogy a fejezet korábbi részeiben olvashattuk, az objektumok metódusokkal rendelkezhetnek. Például a `Database` objektum tartalmazhat egy `Database.Connect` metódust. Léteznek olyan metódusok, amiknek egy vagy több paramétere is van. A *paraméter* egy olyan érték, amit azért adunk át egy metódusnak hogy az sikeresen letudjon futni. Erre jó példa a `Request.MapPath` metódus, ami három paraméterrel is rendelkezik.

```
public string MapPath(string virtualPath, string baseVirtualDir, bool
allowCrossAppMapping);
```

Ez a metódus visszaadja a megadott virtuális elérési útnak megfelelő fizikai elérési utat. A metódus három paramétere a `virtualPath`, `baseVirtualDir`, és az `allowCrossAppMapping`. (Megjegyzés: a deklarálásnál a paraméterek azokkal az adattípusokkal jelennek meg, amiket elfogadnak.) Amikor meghívjuk ezt a metódust, mind három paraméterét meg kell adnunk.

A Razor szinakszissal kétféle módon adhatunk át paramétereket: *helyzeti* és *névszerinti* paraméterekként. Hogy helyzeti paraméterek használatával hívjunk meg egy metódust, a metódus deklarálásának megfelelő sorrendben kell megadni a paramétereket (a metódus dokumentációjából ezt megtudhatjuk). Ehhez a sorrendhez mindenképp tartanunk kell magunkat, nem cserélhetjük fel a paramétereket, és nem hagyhatunk ki paramétereket (ha mégis szükséges, akkor `null` értéket vagy üres stringet ("")adjunk neki).

A következő példa feltételezi, hogy rendelkezünk a webhelyünkön egy *script* mappával. A kód meghívja a `Request.MapPath` metódust, átadja a három paramétert a megfelelő sorrendben,majd megjeleníti a végeredményt.

```
// Helyzeti paraméterek átadása.
var myPathPositional = Request.MapPath("/scripts", "/", true);
<p>@myPathPositional</p>
```

Amikor egy metódus sok paraméterrel rendelkezik, névszerinti paraméterek használatával letisztultabb, és jobban megfejthető lesz a kódunk. Hogy névszerinti paraméterekkel hívjunk meg egy metódust, adjuk meg a paraméter nevét, majd egy kettőspontot (:), ez után adjuk meg az értékét. A névszerinti paraméterek előnye, hogy akármilyen sorrendben megadhatjuk őket. (Hátránya, hogy a metódushívás sokkal több helyet foglal)

Az előző példában látható metódust most névszerinti paraméterekkel hívjuk meg:

```
// Névszerinti paraméterek átadása.
var myPathNamed = Request.MapPath(baseVirtualDir: "/", allowCrossAppMapping: true, virtualPath:
"/scripts");
<p>@myPathNamed</p>
```

Mint láthatjuk, a paramétereket más sorrendben adtuk át, de ennek ellenére, ha lefuttatjuk az előző két példát, ugyanazt az értéket fogják visszaadni.

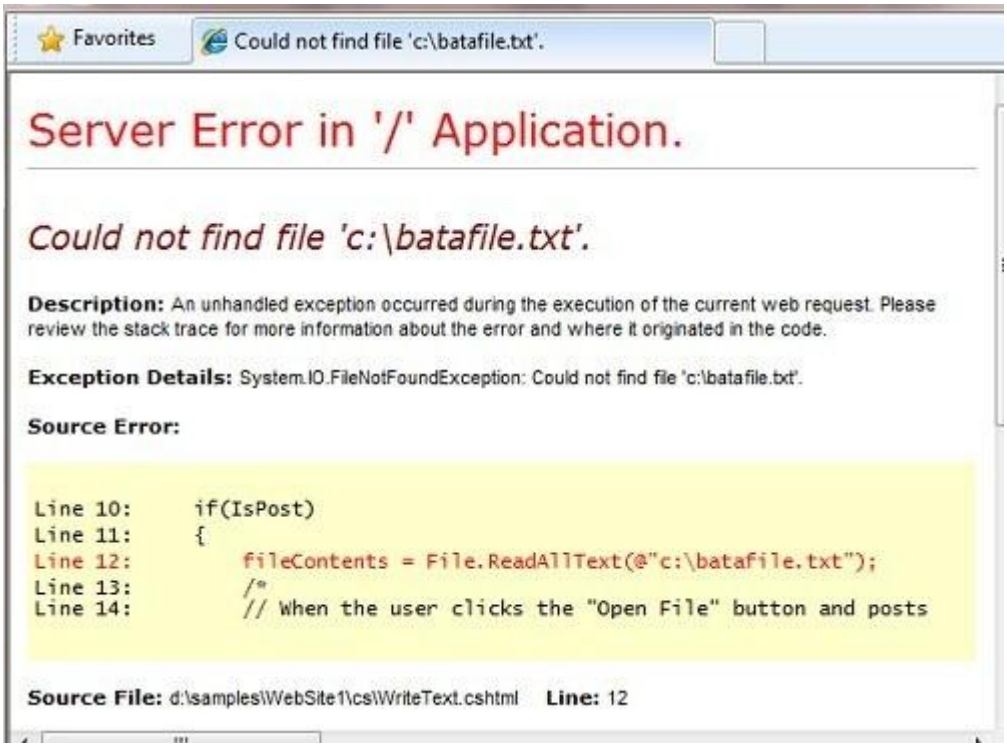
Hibák kezelése

Try-Catch utasítások

Lehetséges, hogy a kódunk tartalmaz olyan utasításokat, amik valamiféle hibát generálnak. Például:

- Sokféle hiba léphet fel, amikor egy fájlon valamiféle műveletet szeretnénk végrehajtani. Lehet, hogy a fájl nem is létezik, vagy csak olvasható, esetleg a kódunk nem tartalmazza a megfelelő jogosultságokat, és így tovább.
- Hasonlóképpen, lehetnek jogosultsági problémák, ha egy adatbázis rekordjait szeretnénk változtatni, de az is lehet, hogy elveszítjük a kapcsolatot az adatbázissal, vagy egyszerűen csak nem megfelelő adatokat akarunk elmenteni, és így tovább.

Ezeket a szituációkat programozási körökben *kivételeknek (exception)* hívjuk. A kódunk automatikusan generál (dob) egy hibaüzenetet, ha kivételbe ütközik.



Használjuk `try/catch` utasításokat, amikor kivételekbe ütközünk, vagy csak el akarjuk kerülni az ilyen típusú hibaüzeneteket. A `try` blokkba azt a kódot írjuk, amit ellenőrizni szeretnénk, majd ehhez párosíthatunk egy vagy több `catch` parancsot, ahol különböző kivételeket kezelhetünk. Valójában annyi `catch` blokkot használhatunk, ahány hibára számítunk.

Megjegyzés: Nem ajánlatos a `Response.Redirect` metódust használni egy `try/catch` utasításban, mert kivételt okozhat az oldalunkban.

A következő példában létrehozunk egy oldalt, ami az első lekérdezéskor létrehoz egy szövegfájlt és megjelenít egy gombot, amivel a felhasználó megnyithatja azt. A példában szándékosan használunk rossz fájlnevet, hogy létrehozzunk egy kivételt. A kód két kivételt kezel le: Az egyik a `FileNotFoundException` vagyis a nem létező fájl kivétel, ami akkor történik ha a megadott fájlnév hibás, vagy nem létezik. A második a `DirectoryNotFoundException`, vagyis a nem létező mappa kivétel, ami akkor következik be, amikor az ASP.NET még a mappát sem tudja megtalálni. (Megszüntethetjük a megjegyzéseket az egyik utasításban, hogy lássuk, milyen, amikor minden jól működik.)

Ha a kódunk esetlegesen nem kezelne le egy kivételt, akkor egy olyan hiba oldalt jelenítene, meg amit az előző képen láthatunk. Azonban a `try/catch` részek segítenek megakadályozni az ilyen típusú hibaüzeneteket.

```
@{
    var dataFilePath = "/dataFile.txt";
    var fileContents = "";
    var physicalPath = Server.MapPath(dataFilePath);
    var userMessage = "Hello world, the time is " + DateTime.Now;
    var userErrMsg = "";
    var errMsg = "";

    if(IsPost)
    {
        // Amikor a felhasználó az „Open File” gombra kattint, és
elküldi
        // az oldalt, az megpróbálja megnyitni az adott fájlt
olvasásra.
        try {
            // Ez a kód hibát fog jelezni, mert a megadott fájl nem
létezik.
            fileContents = File.ReadAllText(@"c:\batafile.txt");

            // Ez a kód működik. Hogy elhárítsuk a hibát,
            // Helyezzünk dupla perjelt a fenti parancs elé, az
alábbi elől pedig töröljük.
            //fileContents = File.ReadAllText(physicalPath);
        }
        catch (FileNotFoundException ex) {
            // A kivétel objektum használható hibajavításra,
naplózásra
            errMsg = ex.Message;
            // Készítsünk felhasználóbarát hibaüzenetet.
            userErrMsg = "A file could not be opened, please contact
"
                + "your system administrator.";
        }
        catch (DirectoryNotFoundException ex) {
            // Similar to previous exception.
            errMsg = ex.Message;
            userErrMsg = "A directory was not found, please contact "
                + "your system administrator.";
        }
    }
    else
    {
        // Amikor először lekérik az oldalt, hozzuk létre a text
fájlt.
        File.WriteAllText(physicalPath, userMessage);
    }
}

<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8" />
        <title>Try-Catch Statements</title>
    </head>
    <body>
    <form method="POST" action="" >
      <input type="Submit" name="Submit" value="Open File"/>
```

```
        </form>

        <p>@fileContents</p>
        <p>@userErrMsg</p>

        </body>
    </html>
```

További források

Magyar nyelven:

C# jegyzet: http://devportal.hu/content/CSharpjegyzet.aspx

ASP.NET indulókészlet: http://devportal.hu/groups/tanulk/pages/mvc.aspx

Függelék: A Visual Basic programozási nyelv és szintaxisa

Angol nyelven:

Referencia dokumentumok

ASP.NET

C# nyelv

4. fejezet – Egységes megjelenés kialakítása

Egy hatékonyabb weboldal kialakításához létrehozhatunk a webhelyünknek többször felhasználható tartalomblokkokat (például fejlécet és láblécet), így egységes megjelenést biztosítva számára.

A fejezetből megtudhatjuk:

- Hogyan hozhatunk létre többször felhasználható tartalomblokkokat, mint például fejlécet és láblécet?
- Hogyan alakíthatunk ki egy layout oldal segítségével egységes kinézetet az összes oldalnak?
- Hogyan küldhetünk át adatot futás közben a layout oldalra?
- Hogyan hozzunk létre és használjunk egy saját helpert?

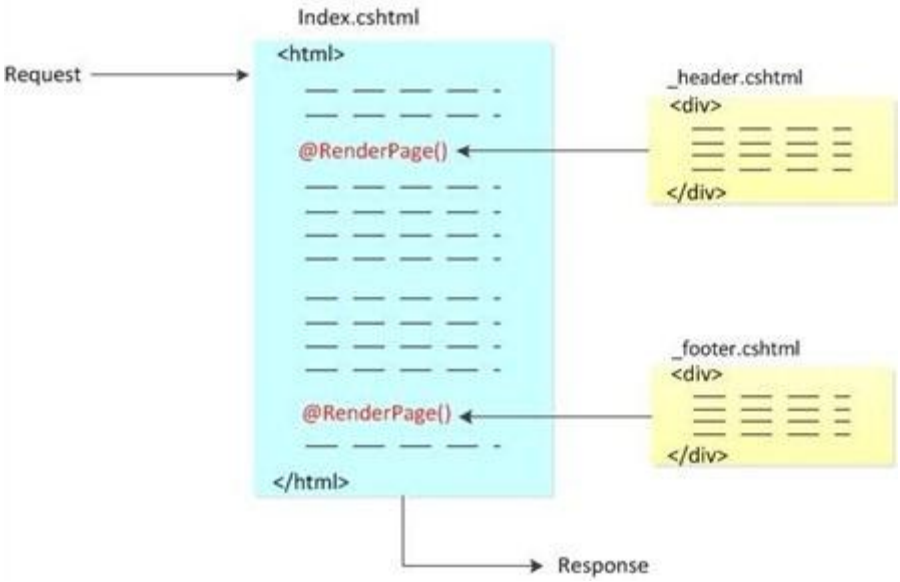
A fejezetben a következő ASP.NET funkciókról olvashatunk:

- Tartalomblokkok, amelyek HTML kódot tartalmazó fájlok, és több oldalra is beszúrhatók.
- Layout oldalak, amelyek HTML tartalommal rendelkeznek, és több oldalon is felhasználhatjuk webhelyünkön.
- A `RenderPage`, `RenderBody` és `RenderSection` metódusok, amelyek jelzik az ASP.NET-nek, hogy hova kell az oldal elemeit beilleszteni.
- A `PageData` szótár, amely lehetővé teszi, hogy adatot osszunk meg tartalomblokkok és layout oldalak között.

Többször használható tartalomblokkok létrehozása

Több weboldalon is találunk olyan tartalmat, amely minden oldalon megjelenik, mint például a fejléc és lábléc, vagy egy doboz, amely mutatja, hogy a felhasználó be van-e jelentkezve. Az ASP.NET-tel egy különálló fájlt hozhatunk létre egy olyan tartalomblokkal, amelyben van szöveg és kód, mint egy átlagos weboldalon. Ezután beilleszthetjük ezt a tartalomblokkot olyan oldalakra a weblapon, ahol szeretnénk, hogy megjelenjen a rajta lévő információ. Ezáltal nem kell külön bemásolni ugyanazt a szöveget minden oldalra. Egy ilyen egyszerű tartalom létrehozása a frissítést is egyszerűbbé teszi. Ha meg szeretnénk változtatni a tartalmat, csak egy fájlt kell frissítenünk ahhoz, hogy ez mindenhol frissüljön, ahova ezt a tartalmat beszúrtuk.

A következő diagram bemutatja, hogyan működnek a tartalomblokkok. Amikor egy böngésző kér egy oldalt a webszervertől, az ASP.NET beszúr egy tartalomblokkot, ott, ahol a `RenderPage` metódust a főoldalra hívják. A kész (összefésült) oldalt ezután küldi el a böngészőnek.



Ebben az eljárásban olyan oldalt hozunk létre, amely két tartalomblokkra utal (egy fejlécre és egy láblécre), melyek külön fájlokban találhatók. Ezeket a tartalomblokkokat használhatjuk bármely oldalnál a weblapunkon. Ha készen vagyunk, egy hasonló oldalt kapunk:

1. A weblap gyökérmappájában hozzunk létre egy fájlt: *Index.cshtml!*
2. Cseréljük ki a már létező kódot a következőre:

```
<html>
  <head>
    <title>Main Page</title>
  </head>
  <body>

    <h1>Index Page Content</h1>
    <p>This is the content of the main page.</p>

  </body>
</html>
```

3. A gyökérmappában hozzunk létre egy mappát: *Shared!*

Megjegyzés: Általában az olyan fájlokat, amelyeket megosztunk weboldalak között, *Shared* nevű mappába helyezzük.

4. Hozzunk létre a *Shared* mappában egy fájlt: *_Header.cshtml!*

5. Cseréljünk ki bármilyen már létező tartalmat a következőre:

```
<div class="header">
   This is header text.
</div>
```

Figyeljünk arra, hogy a fájl neve *_Header.cshtml*, egy alulvonással (_) az elején. Az ASP.NET nem küld el egy oldalt a böngészőnek, ha annak a neve alulvonással kezdődik. Ezzel megelőzhetjük, hogy ezeket az oldalakat (figyelmetlenségből vagy másképp) lekérdezzék. Hasznos, ha alulvonással jelöljük azokat az oldalakat, amelyekben van tartalomblokk, mert nem szeretnénk, hogy ezekhez a felhasználók hozzáférhessenek – ezek kifejezetten arra készültek, hogy más oldalakra is beszúrjuk őket.

6. Hozzunk létre a *Shared* mappában egy fájlt: *_Footer.cshtml,* majd cseréljük ki a tartalmat a következőre:

```
<div class="footer">
   © 2010 Contoso Pharmaceuticals. All rights reserved.
</div>
```

7. Az Index.cshtml oldalon adjuk hozzá a következő, kiemelt kódot, amely két hívást intéz a `RenderPage` metódushoz:

```
<html>
  <head>
    <title>Main Page</title>
  </head>
  <body>

    @RenderPage("/Shared/_Header.cshtml")

    <h1>Index Page Content</h1>
    <p>This is the content of the main page.</p>

    @RenderPage("/Shared/_Footer.cshtml")

  </body>
</html>
```

Ez megmutatja, hogyan szúrjunk be egy tartalomblokkot a weboldalra. Előhívjuk a `RenderPage` metódust, és átadjuk neki a fájl nevét, amelyet ennél a pontnál be akarunk illeszteni. Itt a *_Heather.cshtml* és a *_Footer.cshtml* fájlok tartalmait illesztjük be az *Index.cshtml* fájlba.

8. Futtassuk le a böngészőben az *Index.cshtml* oldalt! (Futtatás előtt győződjünk meg róla, hogy az oldalt a *Files* névhelyről választottuk!)

9. Jelenítsük meg a forrást a böngészőben. (Például az Internet Explorerben jobb egérgomb lenyomása után kattintsunk a *View Source*-ra!)

   Ez elérhetővé teszi a weboldal által a böngészőnek küldött kódot, amely egyesíti az index oldal kódját a tartalomblokkokkal. A következő példa megmutatja az oldal forrását, amelyet az *Index.cshtml*-nek ad át. A `RenderPage`-hez indított hívások, amelyeket az *Index.cshtml*-be illesztettünk be felcserélődtek a fejléc és lábléc fájlok aktuális tartalmával.

```
<html>
  <head>
    <title>Main Page</title>
  </head>
  <body>
```

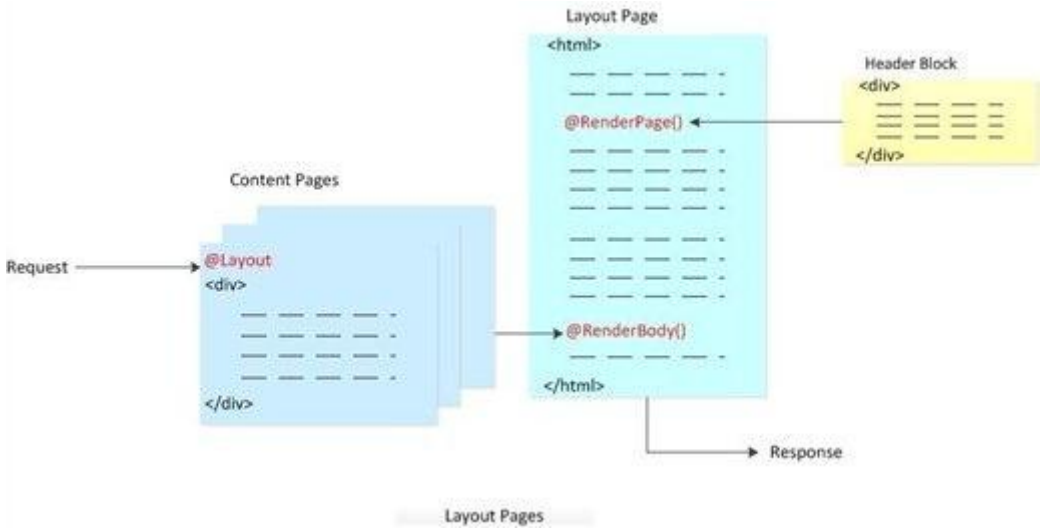
```
    <div class="header">
      This is header text.
    </div>

      <h1>Index Page Content</h1>
      <p>This is the content of the main page.</p>

    <div class="footer">
      © 2010 Contoso Pharmaceuticals. All rights reserved.
    </div>

    </body>
</html>
```

Egységes látvány kialakítása layout oldalakkal

Már láthattuk, mennyire egyszerű tartalmat létrehozni egyszerre több oldalon. Egy ennél is strukturáltabb módja az áttekinthető megjelenítésnek a layout oldalak használata. A layout oldal meghatározza az oldal struktúráját, de nem tartalmaz semmilyen tartalmat. Egy ilyen oldal elkészítése után létrehozhatunk tartalommal is rendelkező oldalakat és hozzákapcsolhatjuk őket a layout oldalakhoz. Amikor megjelenítjük ezeket a tartalmakat, a layout oldal szerint lesznek formázva. (Ebben az értelemben a layout oldal úgy viselkedik, mint egy sablon a tartalommal rendelkező oldalak számára.)

Egy megjelenés oldal annyiban tér el egy másik HTML oldaltól, hogy előhívhatja a `RenderBody` metódust. A `RenderBody` metódus pozíciója a megjelenés oldalon meghatározza, hogy hova kerüljön a megjelenítendő tartalom.

A következő diagram megmutatja, hogyan kapcsolódnak össze a tartalommal rendelkező oldalak a megjelenés oldalakkal, hogy egy kész weboldalt kapjunk. A böngésző lekéri a tartalommal rendelkező oldalt. Ez az oldal tartalmaz egy kódot, amely meghatározza, milyen megjelenést kell használni az oldal megjelenítéséhez. A megjelenítés oldalban a tartalom oda lesz beillesztve, ahol a `RenderBody` metódust előhívják. Tartalomblokkokat is beilleszthetünk a megjelenés oldalba a `RenderPage` metódus meghívásával, ahogyan azt az előző részben csináltuk. A kész weboldal ezután jelenik meg a böngészőben.



A következő eljárás megmutatja, hogyan készíthetünk layout oldalt, majd hogyan csatolhatunk ahhoz tartalmat.

1. A weboldalunk *Shared* mappájába hozzunk létre egy fájlt: *_Layout1.cshtml!*

2. Cseréljünk ki minden, már létező tartalmat a következőre:

```
<head>
  <title> Structured Content </title>
  <link href="@Href("/Styles/Site.css")" rel="stylesheet"
type="text/css" />
</head>
<body>
  @RenderPage("/Shared/_Header2.cshtml")
  <div id="main">
    @RenderBody()
  </div>
  <div id="footer">
    © 2010 Contoso Pharmaceuticals. All rights reserved.
  </div>
</body>
</html>
```

A `RenderPage` metódust használjuk tartalomblokkok beillesztéséhez a layout oldalakon. A layout oldal csak egy hívást kérhet le a `RenderBody` metódustól.

Megjegyzés: A webszerverek nem mind ugyanúgy kezelik a hiperhivatkozásokat (a linkek `href` attribútumát). Ezért az ASP.NET rendelkezik egy `@href` helperrel, amely értelmezi az elérési utat, és olyan formára alakítja, hogy azt a webszerver is elfogadja.

3. A *Shared* mappában hozzunk létre egy fájlt: *_Header2.cshtml*, majd cseréljünk ki minden, már létező tartalmat a következőre:

```
<div id="header">
  Chapter 3: Creating a Consistent Look
</div>
```

4. A gyökérkönyvtárban hozzunk létre egy mappát: *Styles!*

5. A *Styles* mappában hozzunk létre egy fájlt: *Styles.css* majd adjuk meg a következő stílus definíciókat:

```
h1 {
    border-bottom: 3px solid #cc9900;
    font: 2.75em/1.75em Georgia, serif;
    color: #996600;
}


ul {
    list-style-type: none;
}


body {
    margin: 0;
    padding: 1em;
    background-color: #ffffff;
    font: 75%/1.75em "Trebuchet MS", Verdana, sans-serif;
```

```
    color: #006600;
}

#list {
    margin: 1em 0 7em -3em;
    padding: 1em 0 0 0;
    background-color: #ffffff;
    color: #996600;
    width: 25%;
    float: left;
}

#header, #footer {
    margin: 0;
    padding: 0;
    color: #996600;
}
```

Ezek a stílus definíciók csak bemutatják, hogyan használhatjuk a stílus lapokat a layout oldalaknál. Ha szeretnénk, akár saját stílust is összeállíthatunk ezekre az elemekre.

6. A gyökérmappában hozzunk létre egy fájlt: *Content1.cshtml,* majd cseréljünk ki minden, már létező tartalmat a következőre:

```
@{
    Layout = "/Shared/_Layout1.cshtml";
}

<h1> Structured Content </h1>
<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna
aliqua.
Ut enim ad minim veniam, quis nostrud exercitation ullamco
laboris
nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor
in
reprehenderit in voluptate velit esse cillum dolore eu fugiat
nulla
pariatur. Excepteur sint occaecat cupidatat non proident, sunt
in
culpa qui officia deserunt mollit anim id est laborum.</p>
```

Ez az oldal használni fogja a layout oldalt. A tetején található kódblokk határozza meg, hogy mely layout oldalt fogja használni a tartalom formázásához.

7. Futtassuk le a böngészőben a *Content1.cshtml*-t! A lekért oldal a *_Layout1.cshtml*-ben lévő formázást és stílust fogja használni a *Content1.cshtml*-ben összefoglalt szövegnek.

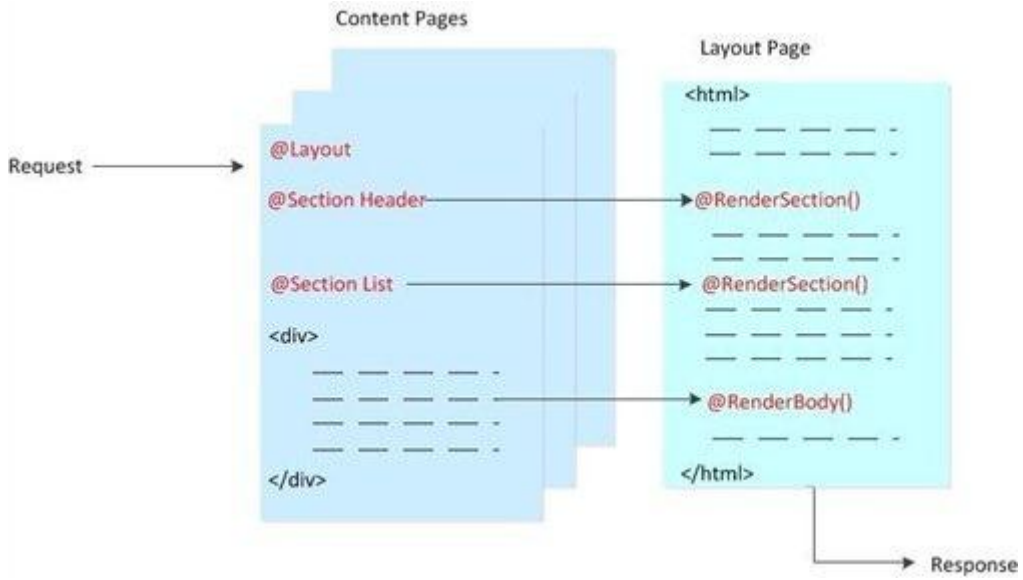
A 6. lépést megismételhetjük, ha további tartalommal rendelkező oldalt szeretnénk hozzáadni a weblaphoz ugyanezzel a megjelenéssel.

Megjegyzés: Beállíthatjuk az oldalunkat úgy, hogy automatikusan ugyanazt a layout oldalt használja a mappában lévő összes tartalomra. További információt találunk a 19.fejezetben - A Site-Wide viselkedés testreszabása.

Többféle tartalommal rendelkező layout oldalak tervezése

Egy tartalommal rendelkező oldal többféle szakaszból állhat. A tartalmi oldalon minden szakasznak külön nevet adunk. (Az alapértelmezett szakasz névtelen marad.) A layout oldalon a `RenderBody` metódussal meghatározzuk, hogy hol jelenjen meg a névtelen (alapértelmezett) szakasz. Ezután külön `RenderSection` metódusokat adunk hozzá, hogy külön elnevezzük a hozzáadott szakaszokat.

A következő diagramban láthatjuk, hogy az ASP.NET hogyan kezeli az olyan tartalmakat, amelyek több részre vannak osztva. Minden elnevezett szakasz megtalálható a tartalom oldal szakasz blokkjában. (A példában Header és Listnek nevezzük őket.) A keretrendszer azon a ponton illeszti be a tartalom szakaszt a layout oldalba, ahol a `RenderSection` metódust lehívják. Mint már korábban láttuk, a névtelen (alapértelmezett) szakaszt arra a pontra illeszti be, ahol a `RenderBody` metódust lehívták.



Ebből a folyamatból láthatjuk, hogyan tudunk olyan tartalommal rendelkező oldalt szerkeszteni, amelyben több tartalomszakasz is van, és hogyan lehet ezeket hozzáadni a layout oldalhoz, amely támogatja a több tartalommal bíró szakaszokat.

1. A weboldalunk *Shared* mappájában hozzunk létre egy fájlt: *_Layout2.cshtml.*
2. Cseréljünk ki minden, már létező tartalmat a következőre:

```
<html>
    <head>
      <title>Multisection Content</title>
      <link     href="@Href("/Styles/Site.css")"     rel="stylesheet"
type="text/css" />
    </head>
    <body>
      <div id="header">
        @RenderSection("header")
      </div>
      <div id="list">
        @RenderSection("list")
      </div>
      <div id="main">
        @RenderBody()
      </div>
      <div id="footer">
        © 2010 Contoso Pharmaceuticals. All rights reserved.
      </div>
    </body>
</html>
```

Mind a fejléc, mind a lista szakaszok hozzáadásához a `RenderSection` metódust használjuk.

3. A gyökérmappában hozzunk létre egy fájlt: *Content1.cshtml,* majd cseréljünk ki minden, már létező tartalmat a következőre:

```
@{
    Layout = "/Shared/_Layout2.cshtml";
}

@section header {
    <div id="header">
        Chapter 3: Creating a Consistent Look
    </div>
}

@section list {
    <ul>
        <li>Lorem</li>
        <li>Ipsum</li>
        <li>Dolor</li>
        <li>Consecte</li>
        <li>Eiusmod</li>
        <li>Tempor</li>
        <li>Incididu</li>
    </ul>
}

<h1>Multisection Content</h1>
<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna
aliqua.
Ut enim ad minim veniam, quis nostrud exercitation ullamco
laboris
nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor
in
reprehenderit in voluptate velit esse cillum dolore eu fugiat
nulla
pariatur. Excepteur sint occaecat cupidatat non proident, sunt
in
culpa qui officia deserunt mollit anim id est laborum.</p>
```

Ez a tartalommal rendelkező oldal tartalmaz egy kódblokkot a lap tetején. Mindegyik elnevezett szakasz egy szakaszblokkban található. Az oldal többi része az alapértelmezett (névtelen) tartalom szakaszt tartalmazza.

4. Futtassuk az oldalt a böngészőben!

Opcionális tartalomszakaszok létrehozása

Általában a tartalommal rendelkező oldalon létrehozott szakaszok ugyanolyanok, mint a layout oldalakon lévők. Különféle hibaüzeneteket kaphatunk, ha a következők merülnek fel:

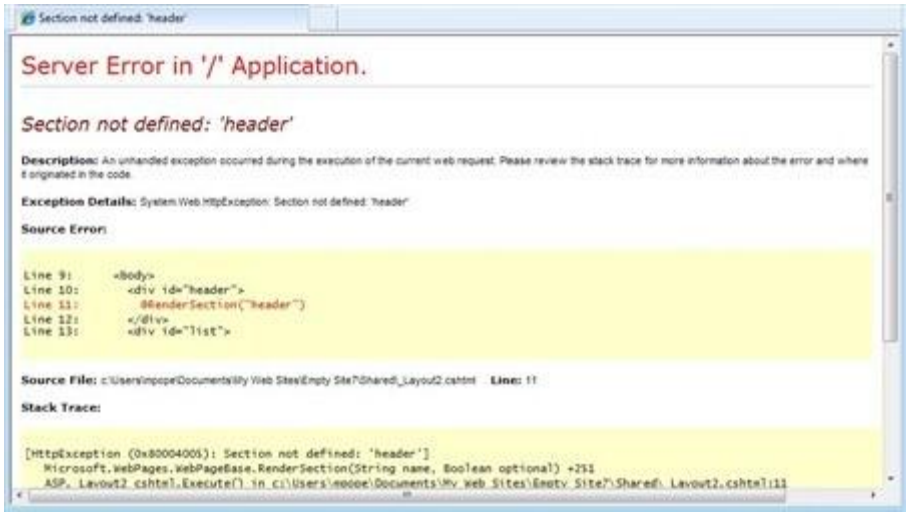
- A tartalmi részben van olyan szakasz, amely nem tartozik egy szakaszhoz sem a layout oldalon.
- A layout oldal tartalmaz olyan szakaszt, amihez nincs tartalom.
- A layout oldal több olyan metódust is tartalmaz, amely ugyanazt a szakaszt kéri le.

Azonban ezt a fajta hibalehetőséget kiküszöbölhetjük, ha egy megnevezett szakaszt opcionálisnak választunk ki a layout oldalon. Ezzel definiálhatunk többszörös tartalommal bíró oldalakat, amelyek egyszerre használhatják a layout oldalt, így akár az is előfordulhat, hogy nem lesz tartalom egy kifejezett szakaszhoz.

1. Nyissuk meg a *Content2.cshtml*-t és töröljük ki a következő részt:

```
@section header {
  <div id="header">
    Chapter 3: Creating a Consistent Look
  </div>
}
```

2. Mentsük el az oldalt, majd futtassuk le egy böngészőben! Egy hibaüzenet jelenik meg, mert a tartalommal rendelkező oldal nem tud szolgáltatni tartalmat a layout oldal egyik szakaszához, nevezetesen a fejléc szakaszhoz.



3. *Shared* mappában nyissuk meg a *_Layout2.cshtml* oldalt, és helyettesítsük ezt a sort:

```
@RenderSection("header")
```

a következő kóddal:

```
@RenderSection("header", required: false)
```

Alternatívaként az előző kódsort kicserélhetjük a következő kódblokkra, ami ugyanazt eredményezi:

```
@if (IsSectionDefined("header")) {
    @RenderSection("header")
}
```

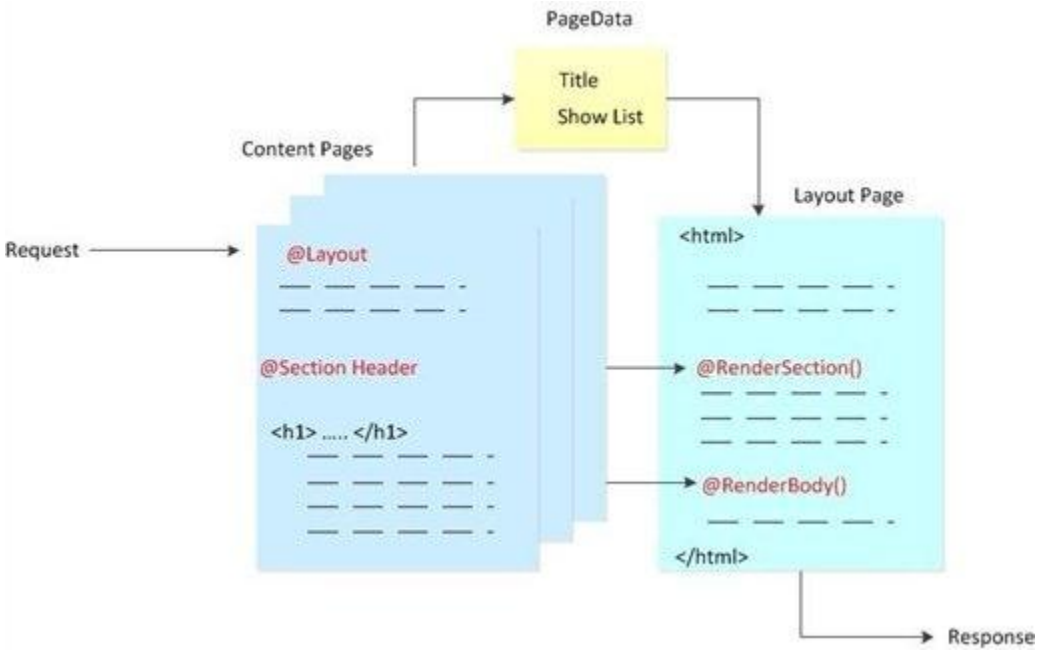
4. Futtassuk újra a *Content2.cshtml* oldalt a böngészőben! (Ha még mindig ez az oldal fut a böngészőben, egyszerűen frissíthetjük azt.) Ez alkalommal nem hoz fel hibaüzenetet, bár hiányzik a fejléc is.

Adat küldése layout oldalakra

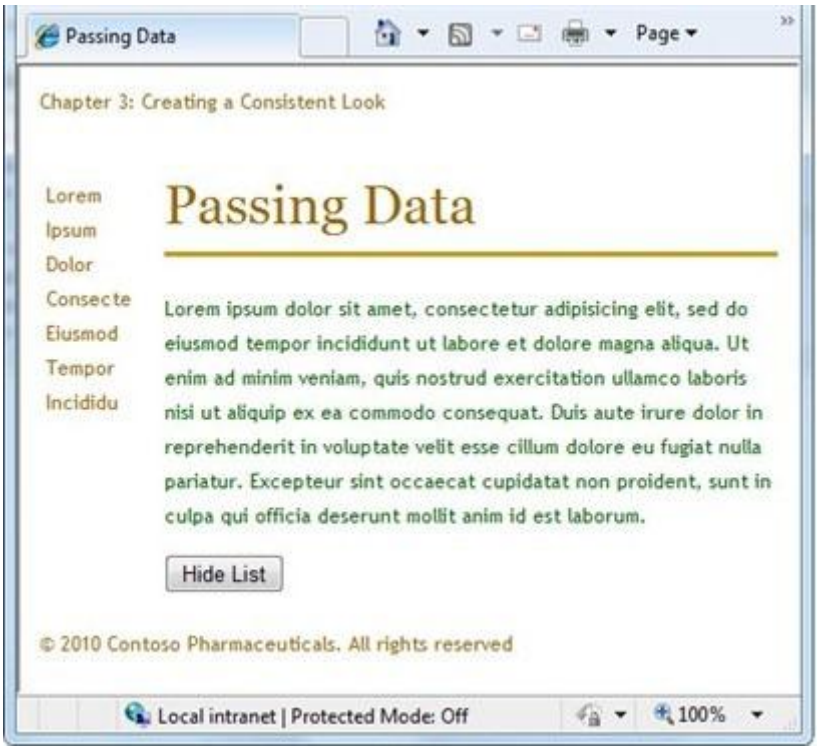
A tartalmi részben előfordulhat olyan adat, amelyre a layout oldalon utalni kell. Ha találkozunk ilyennel, az információt át kell küldenünk a tartalmi oldalról a layout oldalra. Például ha fel akarjuk tüntetni a felhasználó állapotát, vagy megjeleníteni, illetve elrejteni részeket a felhasználók típusa szerint.

Ahhoz, hogy adatot küldjünk át a tartalmi oldalról a layout oldalra, értékeket adhatunk hozzá a tartalmi oldal `PageData` tulajdonságához. A `PageData` tulajdonság egy név/érték pár, amely az általunk átküldeni kívánt adatokat tartalmazza. A layout oldalon később ezeket az értékeket olvashatjuk a `PageData` tulajdonságainál.

Itt egy újabb diagram. Ez bemutatja, hogy az ASP.NET hogyan használja a `PageData` tulajdonságot ahhoz, hogy a tartalmi oldalról értékeket küldjön át a layout oldalra. Amikor az ASP.NET elkezdi építeni a weboldalt, elkészíti a `PageData` kollekciót. A tartalmi oldalon kódot kell írnunk ahhoz, hogy adatot írjunk be a `PageData` kollekcióhoz. A `PageData` kollekcióban levő értékeket a tartalmi oldal más szakaszaiból vagy külön tartalmi blokkból is elérhetjük.



A következő folyamat bemutatja, hogyan küldhetünk át adatot a tartalmi oldalról a layout oldalra. Amikor az oldal fut, egy gombbal lehet eltüntetni illetve megjeleníteni egy listát a layout oldalról. Ha a felhasználó a gombra kattint, igaz/hamis (Boole-i) értéket állít be a `PageData` tulajdonságban. A layout oldal kiolvassa az értéket, és ha hamis, eltünteti a listát. Az érték a tartalmi oldalon is szerepel, hogy megjelenítse-e a *Hide List* (Lista eltüntetése) vagy a *Show List* (Lista megjelenítése) gombokat.



1. A gyökérmappában hozzunk létre egy fájlt: *Content3.cshtml,* majd cseréljünk ki minden, már létező tartalmat a következőre:

```
@{
    Layout = "/Shared/_Layout3.cshtml";

    PageData["Title"] = "Passing Data";
```

```
    PageData["ShowList"] = true;

    if (IsPost) {
        if (Request["list"] == "off") {
            PageData["ShowList"] = false;
        }
    }
}

@section header {
    <div id="header">
        Chapter 3: Creating a Consistent Look
    </div>
}

<h1>@PageData["Title"]</h1>
<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna
aliqua.
Ut enim ad minim veniam, quis nostrud exercitation ullamco
laboris
nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor
in
reprehenderit in voluptate velit esse cillum dolore eu fugiat
nulla
pariatur. Excepteur sint occaecat cupidatat non proident, sunt
in
culpa qui officia deserunt mollit anim id est laborum.</p>

@if (PageData["ShowList"] == true) {
    <form method="post" action="">
      <input type="hidden" name="list" value="off" />
      <input type="submit" value="Hide List" />
    </form>
}
else {
    <form method="post" action="">
      <input type="hidden" name="list" value="on" />
      <input type="submit" value="Show List" />
    </form>
}
```

A kód két adatot tárol a `PageData` tulajdonságon - a weboldal címét és a lista megjelenítéséhez szükséges igaz vagy hamis értéket.

Jegyezzük meg, hogy az ASP.NET feltételesen engedi a HTML kódot használni az oldalon egy kódblokkal! Például az `if/else` blokk a szövegtörzsben meghatározza, melyik formát használja, attól függően, hogy a `PageData["Show List"]` igaz értékre van-e állítva.

2. A *Shared* mappában hozzunk létre egy fájlt: *_Layout3.cshtml*, majd cseréljünk ki minden, már létező tartalmat a következőre:

```
<!DOCTYPE html>

<html>
    <head>
      <title>@PageData["Title"]</title>
      <link href="@Href("/Styles/Site.css")" rel="stylesheet"
type="text/css" />
    </head>
    <body>
```

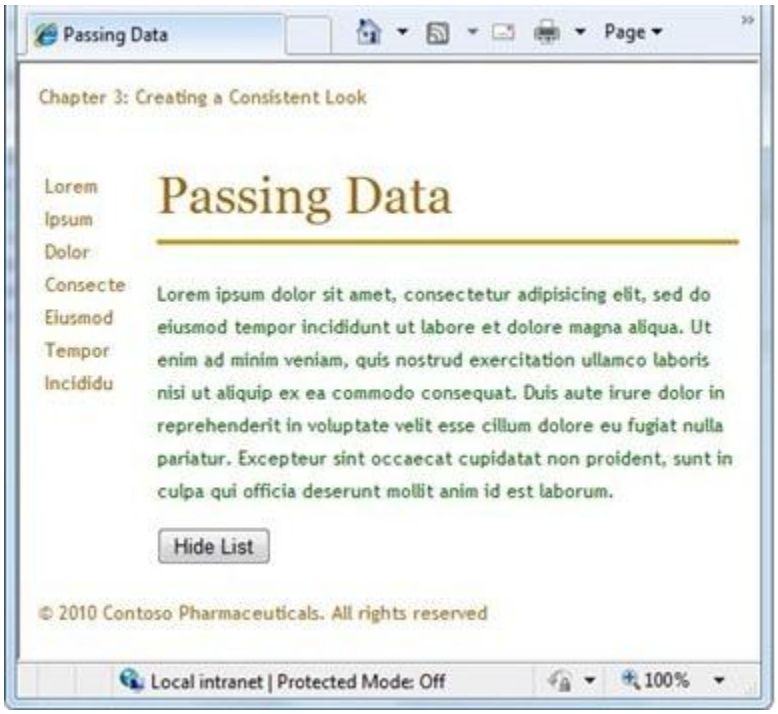
```
        <div id="header">
          @RenderSection("header")
        </div>
        @if (PageData["ShowList"] == true) {
            <div id="list">
              @RenderPage("/Shared/_List.cshtml")
            </div>
        }
        <div id="main">
          @RenderBody()
        </div>
        <div id="footer">
          © 2010 Contoso Pharmaceuticals. All rights reserved.
        </div>
    </body>
</html>
```

A layout oldal tartalmaz `<title>` elemet, ami lekéri a címértéket a `PageData` tulajdonságtól. Ez szintén használja a `PageData` tulajdonság `ShowList` értékét, hogy megjelenítse-e a lista tartalmát.

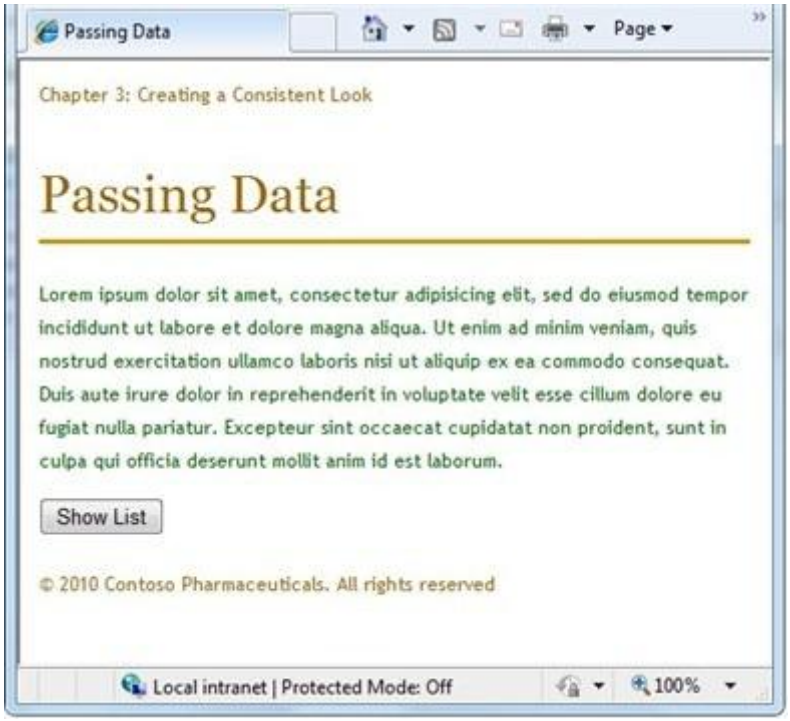
3. A *Shared* mappában hozzunk létre egy fájlt: *_List.cshtml*, majd cseréljünk ki minden, már létező tartalmat a következőre:

```
<ul>
  <li>Lorem</li>
  <li>Ipsum</li>
  <li>Dolor</li>
  <li>Consecte</li>
  <li>Eiusmod</li>
  <li>Tempor</li>
  <li>Incididu</li>
</ul>
```

4. Futtassuk a *Content3.cshtml* oldalt egy böngészőben! Az oldalon látható a lista a bal oldalon és egy *Hide List* gomb az oldal alján.



5. Kattintsunk a *Hide List*re! A lista eltűnik és a gomb átvált *Show List*re.



6. Kattintsunk a *Show List* gombra, és a lista ismét megjelenik!

Egyszerű helper létrehozása és használata

További lehetőség egy egységes nézet kialakításához, ha létrehozunk egy egyéni helpert. Mint azt már láthattuk a 2. fejezetben, a helper olyan elem, amellyel egy összetett feladatot vihetünk véghez egyetlen kódsor segítségével. Az ASP.NET több helperrel is rendelkezik, melyekkel a későbbi fejezetekben jobban megismerkedünk. A helperek teljes listáját a Függelék – ASP.NET API referencia részben találjuk.

Az egységes megjelenítés kialakításában egy helper úgy tud segíteni, hogy lehetővé teszi a gyakran használt kódblokkok egyszerű elérését a webhely összes oldalán. Tegyük fel, hogy oldalunkon olyan megjegyzéseket szeretnénk gyakran használni, amelyek különállnak a normál paragrafusoktól. Ezeket a `<div>` elemmel tehetjük meg, amely egy körülhatárolt dobozt formál. Ahelyett, hogy minden oldalra ugyanazt a jelölést vinnénk fel, készíthetünk belőle egy kisegítőt, amelyet megjegyzéseinkhez mindenhova beilleszthetünk egyetlen kódsorral. Ezzel az oldalon lévő kódok könnyen felismerhetőek, és egyszerűbben átláthatóbbak. Az oldal fejlesztését is könnyebbé teszi, hiszen ha csak jegyzetek kinézetét szeretnénk megváltoztatni, elég csak egy helyen átírni a kódot.

Ez a folyamat bemutatja, hogyan készítsünk helpert, amely a fent említett példánkban a megjegyzéseket formázza. Ez egy egyszerű példa, az egyéni helper bármely, számunkra hasznos jelölést vagy ASP.NET kódot tartalmazhatja.

1. A weboldal gyökérmappájában hozzunk létre egy új mappát: *App_Code!*
2. Az *App_Code* mappában hozzunk létre egy új .cshtml fájlt: *MyHelpers.cshtml!*
3. Cseréljünk ki minden már meglévő tartalmat a következőre:

```
@helper MakeNote(string content) {
    <div class="note" style="border: 1px solid black; width:
90%; padding: 5px; margin-left: 15px;">
      <p>
          <strong>Note</strong>   @content
       </p>
    </div>
}
```

   A kód a `@helper` szintaxist használja, hogy létrehozza az új `MakeNote` kisegítőt. Ez a kisegítő segít a `content` paraméter átküldésében, amely tartalmazhat szöveget és formázást is egyaránt. A kisegítő a `@content` változó segítségével illeszt be egy stringet a jegyzet törzsébe.

   Jegyezzük meg, hogy a fájl neve *MyHelpers.cshtml* , de a kisegítőé `MakeNote`! Több kisegítőt is elhelyezhetünk egy fájlban.

4. Mentsük el és zárjuk be a fájlt!

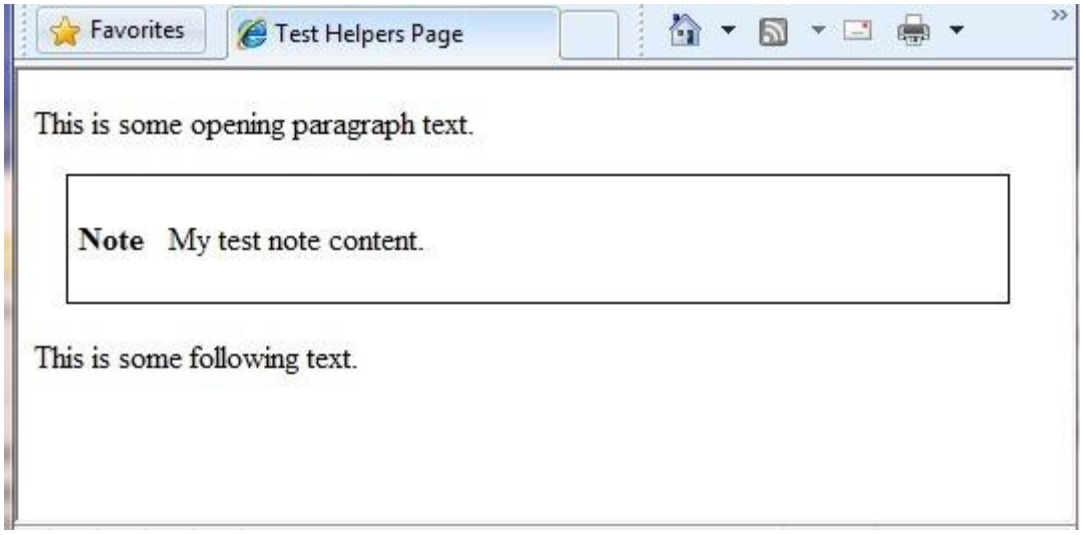
A következő folyamat azt mutatja be, hogyan illesszünk be a weboldalba megjegyzést az általunk elkészített kisegítővel.

1. A gyökérmappában hozzunk létre egy új fájlt: *TestHelper.cshtml!*
2. Adjuk a fájlhoz a következő kódot:

```
  <head>
    <title>Test Helpers Page</title>
  </head>
  <body>
    <p>This is some opening paragraph text.</p>

    <!-- Itt helyezzük el a megjegyzés helperünkre szóló
hívást. -->
    @MyHelpers.MakeNote("My test note content.")

    <p>This is some following text.</p>
  </body>
</html>
```

Ahhoz, hogy előhívjuk az általunk elkészített kisegítőt, használjuk a @-ot a kisegítő fájlnév előtt, egy pontot, majd a kisegítő nevét! (Ha több mappánk is volt az *App_Code* mappában, akkor használhatjuk a `@FolderName.FileName.HelperName` szintaxist is, hogy előhívjuk a kisegítőt bármelyik beágyazott mappaszintről.) A zárójelesen, idézőjelek között megadott szöveget fogja a helper úgy megjeleníteni az oldalon, mint egy megjegyzést.

3. Mentsük el az oldalt, majd futtassuk egy böngészőben! A kisegítő oda fogja helyezni a jegyzetet, ahol azt előhívtuk: a két paragrafus közé.



5. fejezet – Munka az űrlapokkal

Egy űrlap a HTML dokumentum azon része, ahol elhelyezhetünk felhasználói beviteli vezérlőket, például szövegdobozokat, jelölőnégyzeteket, választókapcsolókat vagy legördülő listákat. Akkor használunk űrlapokat, amikor a felhasználó által bevitt adatokat szeretnénk összegyűjteni és feldolgozni.

A fejezetből megtudhatjuk:

- Hogyan hozzunk létre egy HTML űrlapot?
- Hogy olvassunk ki felhasználó által bevitt adatokat az űrlapból?
- Hogyan ellenőrizhetjük a bevitt adatokat?
- Hogyan állítsuk vissza az űrlap értékeit az oldal elküldése után?

A következő ASP.NET programozási fogalmakat mutatjuk be a fejezetben:

- A `Request` objektum
- Bevitel ellenőrzése
- HTML kódolás

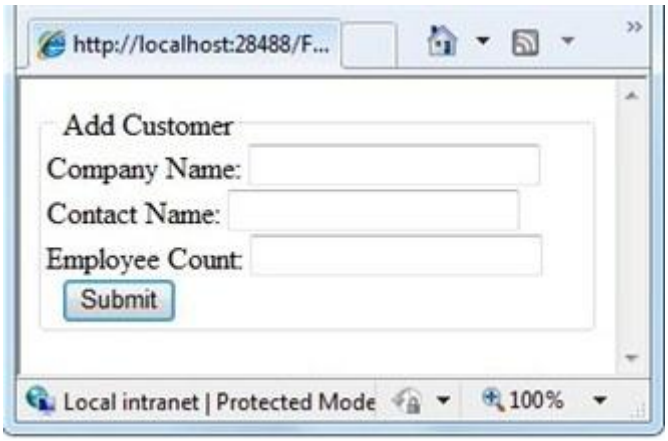
Egy egyszerű HTML űrlap létrehozása

1. Hozzunk létre egy új webhelyet.
2. A gyökérmappájába hozzuk létre a *Form.cshtml* nevű weboldalt, és írjuk be a következő kódot:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Customer Form</title>
    </head>
    <body>
      <form method="post" action="">
        <fieldset>
          <legend>Add Customer</legend>
          <div>
            <label for="CompanyName">Company Name:</label>
            <input type="text" name="CompanyName" value="" />
          </div>
          <div>
            <label for="ContactName">Contact Name:</label>
            <input type="text" name="ContactName" value="" />
          </div>
          <div>
            <label for="Employees">Employee Count:</label>
            <input type="text" name="Employees" value="" />
          </div>
          <div>
            <label> </label>
            <input type="submit" value="Submit" class="submit" />
          </div>
        </fieldset>
      </form>
```

```
    </body>
</html>
```

3. Futtassuk az oldalt a böngészőben.(Bizonyosodjunk meg, hogy az oldal ki van választva a *Files* névtérben, mielőtt futtatjuk.) Egy egyszerű űrlap jelenik meg három beviteli mezővel és egy *Submit* gombbal.



Ezen a ponton, ha a *Submit* gombra kattintunk, semmi sem történik. Hogy az űrlapot használhatóvá tegyük, hozzá kell adnunk néhány sor kódot, ami a szerveren fog lefutni.

Felhasználó által bevitt érték olvasása az űrlapból

Hogy feldolgozzuk az űrlapot, hozzáadunk egy kódot, ami kiolvassa a benyújtott mezőértékeket és valamilyen feladatot végrehajt. A most következő feladat bemutatja, hogyan lehet kiolvasni a mezőket és megjeleníteni a bevitt értékeket az oldalon. (A való életben rengeteg izgalmas módon használhatjuk fel a felhasználói bevitelt. Ezekkel később az adatbázisokkal kapcsolatos fejezetben találkozhatunk.)

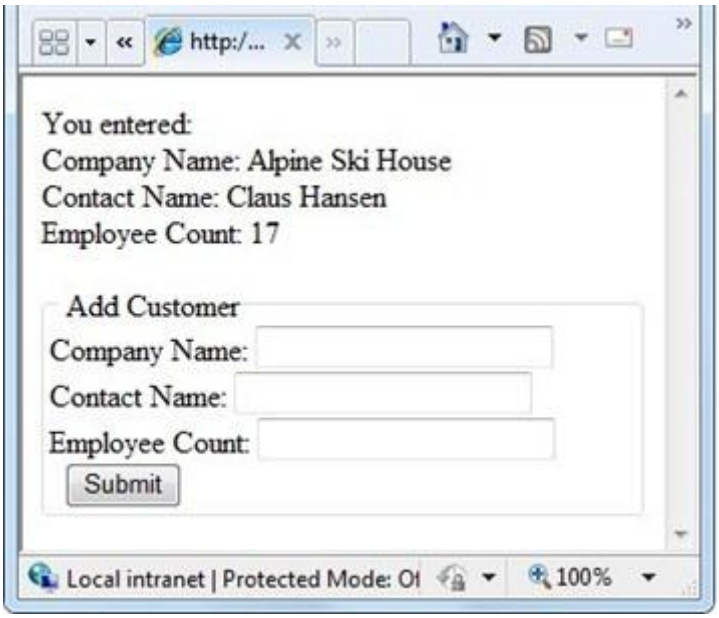
1. A *Form.cshtml* file tetejére szúrja be a következő kódot:

```
@{
    if (IsPost) {
        string companyname = Request["companyname"];
        string contactname = Request["contactname"];
        int employeecount = Request["employees"].AsInt();

        <text>
          You entered: <br />
          Company Name: @companyname <br />
          Contact Name: @contactname <br />
          Employee Count: @employeecount <br />
        </text>
    }
}
```

Amikor a felhasználó először kéri az oldalt, csak az üres űrlap fog megjelenni. A felhasználó (jelen esetben Ön) kitölti az űrlapot és rákattint a *Submit* gombra. Ezek a bejegyzések a felhasználó által bevitt adatok a szerverre. A kérelem eljut ugyanerre a weboldalra (névszerint a *Form.cshtml*-re), mert amikor elkészítettük az űrlapot az előző feladatrészben, a `form` elem `action` attribútumát üresen hagytuk:

```
<form method="post" action="">
```

Amikor elküldjük az oldalt, a megadott adatok megjelennek az űrlap fölött:

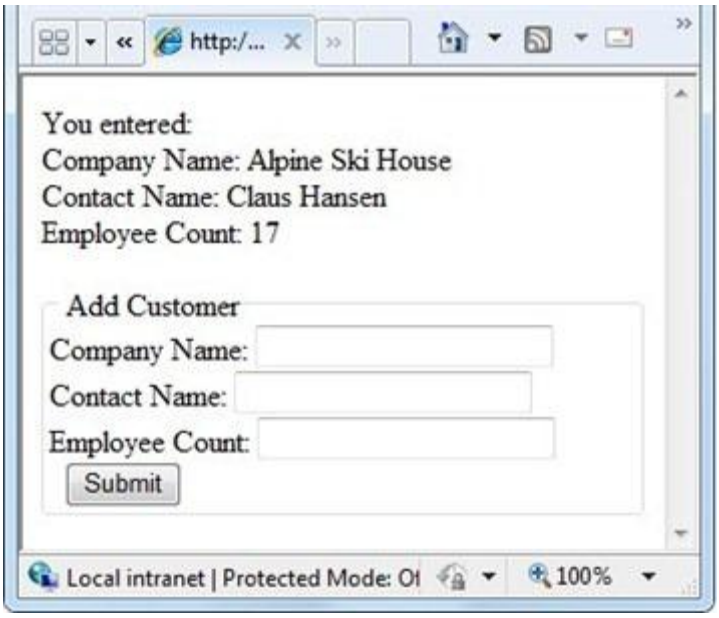


Nézzük meg a lap kódját! Legelőször használjuk az `IsPost` metódust, hogy meghatározzuk, a lap el lett-e küldve, vagyis rákattintott-e a felhasználó a *Submit* gombra. Ha elküldte, az `ISPost` igazat fog mutatni. Ez az általános eljárás az ASP.NET-ben annak kiderítésére, hogy belső lekéréssel (GET request) vagy felhasználói bevitellel (POST request) dolgozunk. (A GET-ről és a POST-ról részletesebben a 3. fejezetben, a HTTP GET és POST metódusok és az IsPost tulajdonság részben olvashatunk.

Ezután megkapjuk a felhasználó által megadott értékeket a `Request` objektumból, és betesszük őket a változókba, későbbi felhasználásra. A `Request` objektum tartalmazza az oldal által benyújtott értékeket, mindegyiket egy kulccsal azonosítva. A kulcsok megegyeznek az űrlapmezők `name` attribútumaival. A `companyname` mező (szövegdoboz) kiolvasásához például használjuk a `Request["companyname"]` parancsot!

Az űrlapértékek a `Request` objektumban stringként vannak tárolva. Ezért ha szám, dátum vagy egyéb típusú értékkel szeretnénk dolgozni, át kell konvertálni azt stringből a nekünk megfelelő típusra. A példában, a `Request` parancs `AsInt` metódusa átkonvertálja az *employees* mező értékeit (ami a foglalkoztatottak számát mutatja) egy integerre (egész számra).

2. Futtassuk az oldalt a böngészőben, töltsük ki az űrlap mezőit, és kattintsunk a *Submit* gombra! Az oldal megjeleníti a megadott értékeket.



HTML kódolás a megjelenésért és biztonságért

A HTML-nek vannak speciális felhasználású karakterei, mint a <,> és &. Ha ezek a speciális karakterek megjelennek ott, ahol nem várható, ronthatják a weboldal megjelenését és funkcionalitását. Például a böngésző egy HTML elem kezdeteként értelmezi a <karaktert (hacsaknem szóköz követi), mint a <b> vagy <input…> elemeket. Ha a böngésző nem ismeri fel az elemet, egyszerűen eldobja a < karakterrel kezdődő karakterláncot, amíg el nem ér valamihez, amit újra felismer. Nyilvánvalóan ez néhány furcsa jelenséget okozhat a weboldalon.

A HTML kódolás kicseréli ezeket a lefoglalt karaktereket egy kóddal, amit a böngésző a helyes szimbólummal helyettesít. Például a <karaktert kicseréli a < kóddal, és a > karaktert a > kóddal. A böngésző ezeket a karakterláncokat helyettesíti azokkal a karakterekkel, amiket látni szeretnénk.

Érdemes mindig HTML kódolást használni a felhasználó által bevitt stringek (adatok) megjelenítésére. Ha nem tesszük, a felhasználó megpróbálhat rosszindulatú scripteket vagy valami mást futtatni, ami veszélyeztetheti webhelyünk biztonságát vagy olyat tehet, amit mi nem szeretnénk. (Ez különösen fontos lehet, ha a felhasználótól kérünk be adatokat, tároljuk valahol, és csak később jelenítjük meg azokat – ahogyan azt például blogos hozzászólásokkal, felhasználói értékelésekkel tesszük.

Adatbevitel ellenőrzése

A felhasználók hibázhatnak. Elfelejtenek kitölteni mezőket, vagy alkalmazottjaik száma helyett egy nevet írnak be. Hogy megbizonyosodjunk arról, hogy az űrlap helyesen lett kitöltve, feldolgozás előtt érvényesítjük az adatbevitelt.

Ez a művelet mutatja, hogyan ellenőrizzük mindhárom űrlapmezőt, hogy biztosak legyünk, a felhasználó nem hagyta üresen. Azt is ellenőrizzük, hogy az alkalmazottak száma érték egy szám-e. Ha ezek között valamelyik hibás, kiírunk egy hibaüzenetet, ami jelzi, hogy a felhasználó melyik értékeket adta meg rosszul.

1. A *Form.cshtml* fájlban cseréljük ki a kód első blokkját a következővel:

```
@{
    if (IsPost)  {
        var errors = false;
        var companyname = Request["companyname"];
        if (companyname.IsEmpty()) {
            errors = true;
```

```
            @:Company name is required.<br />
        }
        var contactname = Request["contactname"];
        if (contactname.IsEmpty()) {
            errors = true;
            @:Contact name is required.<br />
        }
        var employeecount = 0;
        if (Request["employees"].IsInt()) {
            employeecount = Request["employees"].AsInt();
        } else {
            errors = true;
            @:Employee count must be a number.<br />
        }
        if (errors == false) {
            <text>
              You entered: <br />
              Company Name: @companyname <br />
              Contact Name: @contactname <br />
              Employee Count: @employeecount <br />
            </text>
        }    }
}
```

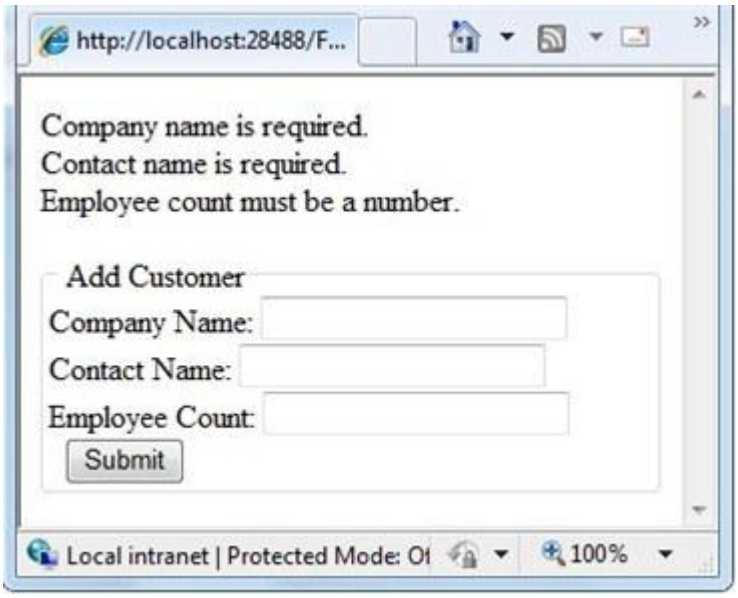
Ez a kód hasonlít a lecseréltre, néhány dologban azonban különbözik. Az első különbség az, hogy megadja az `errors` nevű változó kezdőértékét hamisra. Akkor fogjuk megváltoztatni igazra, ha bármelyik érvényesítési teszt hibát jelez.

Minden alkalommal, amikor a kód kiolvassa az űrlapmezők értékeit, lefuttat egy hitelesítési tesztet. A `companyname` és `contactname` mezőket ellenőrizzük az `IsEmpty` funkció meghívásával. Ha a teszt nem sikerül (tehát az `IsEmpty` igaz értéket ad), a kód az `errors` változót igazra állítja és megjeleníti a megfelelő hibaüzenetet.

A következő lépés, hogy megbizonyosodunk arról, hogy a felhasználó számértéket (egy integert) adott-e meg a beosztottak számának. Ehhez hívjuk meg az `IsInt` funkciót. Ez a funkció igaz értéket ad vissza, ha az általunk tesztelt érték átalakítható stringből integerbe (és természetesen hamisat, hogyha nem). Ne feledjük, hogy a request objektumban az összes érték string! Bár ebben a példában ez nem igazán számít, gyakran végzünk matematikai műveleteket a megadott számokkal, ehhez pedig számmá kell alakítanunk a szövegeket.

Ha az `IsInt` szerint az érték egy integer, állítsuk az `employeecount` változót arra az értékre. Mindazonáltal, mielőtt ezt megtesszük, ténylegesen át kell alakítanunk egy integerré, mert amikor `employeecount`-nak értéket adtunk, szövegként gépeltük be. Nézzük meg a mintát: az IsInt funkció megállapítja, hogy a szöveg egy egész számot tartalmaz-e. Az `AsInt` funkció a következő sorban ténylegesen végrehajtja az átalakítást. Ha az `IsInt` nem ad igaz értéket, az else blokkban lévő parancsok beállítják az `errors` változót igazra.

Végül, miután minden teszt lefutott, a kód megállapítja, hogy az `errors` változó még mindig hamis-e. Ha hamis, a kód megjeleníti a szövegdobozt, ami tartalmazza a felhasználó által megadott értékeket. Futtassuk az oldalt a böngészőben, hagyjuk az űrlapmezőket üresen, és kattintsunk a *Submit* gombra. A weblap hibát fog jelezni.



2. Adjuk meg az értékeket az űrlap mezőkben, és kattintsunk a *Submit* gombra! Az oldal ugyanúgy mutatja a megadott értékeket, mint ahogy korábban láttuk.

Az űrlap értékeinek visszaállítása az elküldés után

Amikor az előző részben az oldalt teszteltük, észrevettük, hogy ha az érvényesítés nem sikerült, minden, amit megadtunk (nemcsak a hibás adatok) eltűntek, és az összes mezőt újra ki kellett tölteni. Ez egy fontos dolgot mutat: amikor beküldjük az oldalt, feldolgozzuk, és aztán az oldalt újra megjelenítjük, a semmiből. És mint látjuk, ez azt jelenti, hogy minden érték, ami az oldalon volt, amikor elküldtük, elveszett.

Ezt könnyen orvosolhatjuk. Hozzáférésünk van a beküldött értékekhez (a `Request` objektumban), tehát azokat az értékeket vissza tudjuk írni az űrlapmezőkbe, amikor az oldalt újra megjelenítjük.

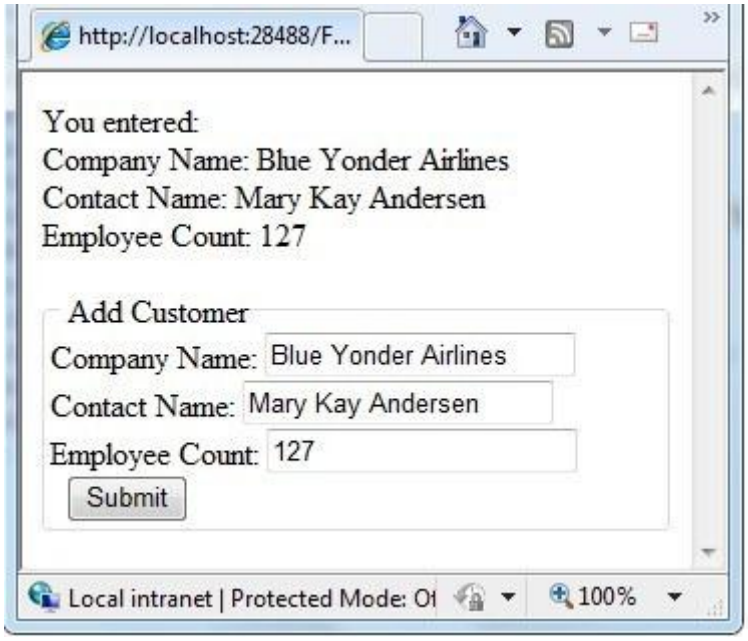
1. A *Form.cshtml* fájlban cseréljük ki az alapértelmezett oldalt a következő kóddal:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Customer Form</title>
    </head>
    <body>
      <form method="post" action="">
        <fieldset>
          <legend>Add Customer</legend>
          <div>
            <label for="CompanyName">Company Name:</label>
            <input type="text" name="CompanyName"
                   value="@Request["companyname"]" />
          </div>
          <div>
            <label for="ContactName">Contact Name:</label>
            <input type="text" name="ContactName"
                   value="@Request["contactname"]" />
          </div>
          <div>
            <label for="Employees">Employee Count:</label>
            <input type="text" name="Employees"
value="@Request["employees"]" />
          </div>
```

```
            <div>
              <label> </label>
              <input type="submit" value="Submit" class="submit" />
            </div>
          </fieldset>
        </form>
      </body>
</html>
```

Az `<input>` elem `value` attribútumát úgy állítottuk be, hogy a `Request` objektum értékmezőit dinamikusan olvassa. Amikor az első alkalommal hívjuk le az oldalt, a Requset objektum értékei mind üresek. Ez rendben van, hiszen az űrlap üres.

2. Futtassuk az oldalt a böngészőben, töltsük ki a mezőket vagy hagyjuk őket üresen, és kattintsunk a *Submit*-re! A megjelenő oldal tartalmazni fogja a megadott értékeket.



További források angolul

- 1,001 Ways to Get Input from Web Users
- Using Forms and Processing User Input
- Using AutoComplete in HTML Forms
- Gathering Information With HTML Forms
- Go Beyond HTML Forms With AJAX

6. fejezet – Műveletek adatokkal

Ez a fejezet ismerteti, hogyan érhetők el az adatok egy adatbázisban, és hogy hogyan jelenítsük meg ezeket az ASP.NET weboldalakkal.

Az útmutatóból megtudhatjuk:

- Hogyan készítsünk adatbázist?
- Hogyan csatlakozzunk egy adatbázishoz?
- Hogyan jeleníthetünk meg adatokat egy weboldalon?
- Hogyan szúrjunk be, frissítsünk és töröljünk adatbázis rekordokat?

A fejezetben bemutatott szolgáltatások:

- Műveletek egy Microsoft SQL Server Compact Edition adatbázissal.
- Műveletek SQL lekérdezésekkel.
- A `Database` csoport.

Bevezetés az adatbázisokba

Képzeljünk el egy tipikus címjegyzéket! Minden egyes bejegyzéshez a címjegyzékben (azaz minden személyhez) tartoznak információk (adatok), mint a vezetéknév, keresztnév, cím, e-mail cím és telefonszám.

Egyik jellegzetes módja a szemléltetésnek egy oszlopokra és sorokra tagolt táblázat. Adatbázis értelmében gyakran minden sort egy-egy rekordként említünk. Minden oszlop (néha mezőként említjük) tartalmaz egy értéket az összes adattípushoz: vezetéknév, keresztnév, stb… .

ID	Keresztnév	Vezetéknév	Cím	E-mail	Telefon
1	Jim	Abrus	210 100th St SE Orcas WA 98031	jim@contoso.com	555 0100
2	Terry	Adams	1234 Main St. Seattle WA 99011	terry@cohowinery.com	555 0101

A legtöbb adatbázistáblához kell, hogy tartozzon oszlop, ami egy egyedi azonosítót tartalmaz, mint például egy ügyfélszám, számlaszám, stb... . Ez a tábla elsődleges kulcsaként ismert, és az egyes oszlopok azonosításához használjuk. Példánkban az ID nevű oszlop az elsődleges kulcs a címjegyzékünkhöz.

Az adatbázisokról való alapvető ismereteinkkel készen állunk megtanulni, hogyan csináljunk egy egyszerű adatbázist, és hajtsunk végre műveleteket, mint adatok hozzáadása, módosítása és törlése.

Relációs adatbázisok

Sokféleképpen tudunk adatokat tárolni, beleértve a szöveges fájlokat és táblázatokat. Az üzleti életben leggyakrabban relációs adatbázisokkal találkozhatunk.

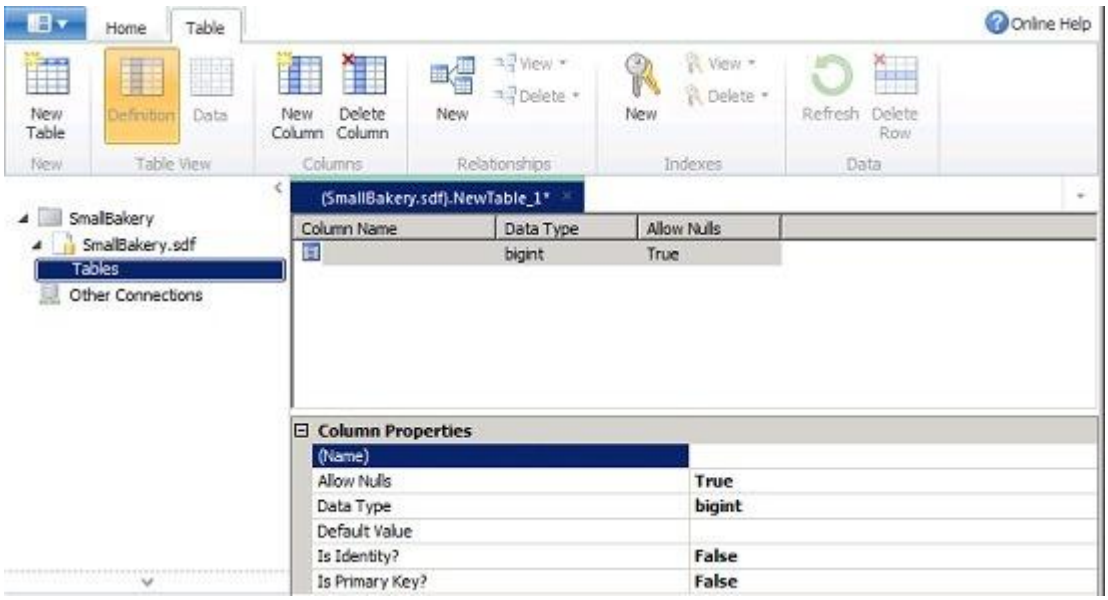
Ez a fejezet nem foglalkozik túl részletesen az adatbázisokkal, azonban érdemes néhány alapfogalmat megismerni. Egy relációs adatbázisban az információ logikusan szét van osztva táblázatokba. Például egy iskolai adatbázis tartalmazhat külön táblákat a diákokról és tanárokról. Az adatbázis-kezelő szoftverek (mint például az SQL Server) támogatják azokat a hasznos parancsokat, amelyek segítségével a táblák között dinamikus kapcsolatokat hozhatunk létre. Például összekapcsolhatjuk a diákok és osztályok tábláit, hogy elkészítsük az órarendeket. A különböző adatok különböző táblákon

való elhelyezése egyszerűbbé, átláthatóbbá teszi az adatbázis struktúráját, és csökkenti a redundáns adatok szükségességét.

Adatbázis létrehozása

Ez az eljárás megmutatja, hogyan hozzuk létre a SmallBakery nevű adatbázist az SQL Server Compact Database tervező eszközével, melyet a WebMatrixban találhatunk meg. Habár tudunk adatbázist készíteni kódok használatával is, sokkal tipikusabb, hogy adatbázist és adatbázistáblákat tervező eszközökkel készítjük el, például a WebMatrixszal.

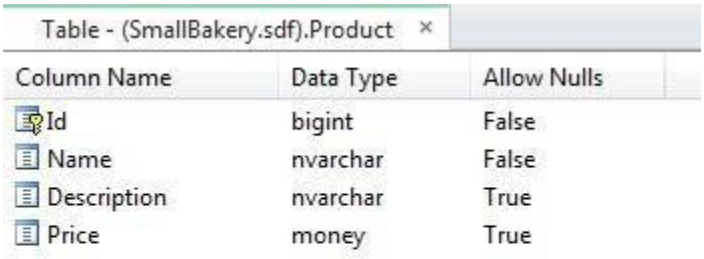
1. Indítsuk el a WebMatrixot, és a gyorsindítás lapon kattintsunk a *Site From Tempalate*-re!
2. Válasszuk az *Empty Site*-ot, aztán a *Site Name* mezőbe írjuk be, hogy SmallBakery, és kattintsunk az OK-r!. A lap elkészült és megjelent a WebMatrixban.
3. A bal oldali panelon kattintsunk a *Databases* munkafelületre!
4. A szalagon kattintsunk a *New Database*-re. Így egy üres adatbázist készítettünk, ugyanazzal a névvel, mint az oldalunk.
5. A bal oldali panelon bontsuk ki a *SmallBaker.sdf* csomópontot, aztán kattintsuk a *Tables*-re.
6. A szalagon kattintsunk a *NewTable*-re. A WebMatrix megnyitja a táblatervezőt.



7. A *Column Properties*-ben a *(Name)* mezőbe írjuk be, hogy „Id”!
8. Az új Id oszlop miatt állítsuk az *Is Identity*t és az *Is Primary Key*t True-ra (igazra)!
   Ahogy a neve is mutatja, az *Is Primary Key* utasítja az adatbázist, hogy ez legyen a tábla elsődleges kulcsa. Az *Is Identity* utasítja az adatbázist, hogy rendeljen automatikusan hozzá egy azonosító számot minden új rekordhoz, és hogy ez a következő sorszám legyen (kezdve az 1-gyel).
9. A szalagon kattintsunk a *New Colum*ra!
10. A Column Properties-ben a (Name) mezőbe írjuk be, hogy „Name”!
11. Állítsuk az *Allow Nulls*-t hamisra! Ez biztosítja, hogy a Name oszlop nem maradhat üresen.
12. Állítsuk a *Data Type*-t „nvarchar”-ra! A *var* része a *nvarchar*-nak azt jelzi, hogy az oszlop adatai egy karakterláncot fognak alkotni, amelynek a mérete változhat rekordról rekordra. (Az *n* prefixum fejezi ki a *national*-t, jelezve azt, hogy a mező karakteres adatokat tartalmazhat, amely bármely ábécét vagy írásrendszert képviseli—vagyis azt, hogy a mező Unicode adatokat tartalmaz.)

13. Ugynezzel a leírással készítsünk egy *Description* nevű oszlopot! Állítsuk az *Allow Nulls*-t hamisra és a *Data Type*-ot „nvarchar”-ra!
14. Készítsünk egy oszlopot *Price* névvel! Állítsuk az *Allow Nulls*-t hamisra és a *Data Type*-t „money”-ra!
15. Nyomjuk meg a Ctrl+S-t a tábla mentéséhez, és nevezzük el „Product”-nak!

Ha elkészültünk, a tábla meghatározása így fog kinézni:



Adatok hozzáadása az adatbázishoz

Most töltsük fel néhány mintaadattal az adatbázisunkat, amikkel a későbbiekben dolgozni fogunk a fejezetben.

1. A bal oldali panelon bontsuk ki a SmallBaker.sdf csomópontot, aztán kattintsuk a Tables-re!
2. Jobb gombbal kattintsunk a Product táblára, utána kattintsunk a *Data*-ra!
3. A szerkesztés ablakba vigyük be a következő adatsorokat:

Name	Description	Price
Bread	Baked frash every day.	2.99
Starwberry Shortcake	Made with organic strawberries from our garden.	9.99
Apple Pie	Second only to your mom’s pie.	12.99
Pecan Pia	If you like pecans, this is for you.	10.99
Lemon Pia	Made with the best lemons in the world.	11.99
Cupcakes	Your kids and the kid in you will love these.	7.99

Ne feledjük, hogy semmit sem kell írnunk az ID oszlopba! Mikor elkészítettük az Id oszlopot, az *Is Identity* tulajdonságát igazra állítottuk, melynek következtében a rekord automatikusan kap egy azonosítót.
Ha befejeztük az adatok bevitelét, a táblatervezőnk így fog kinézni:

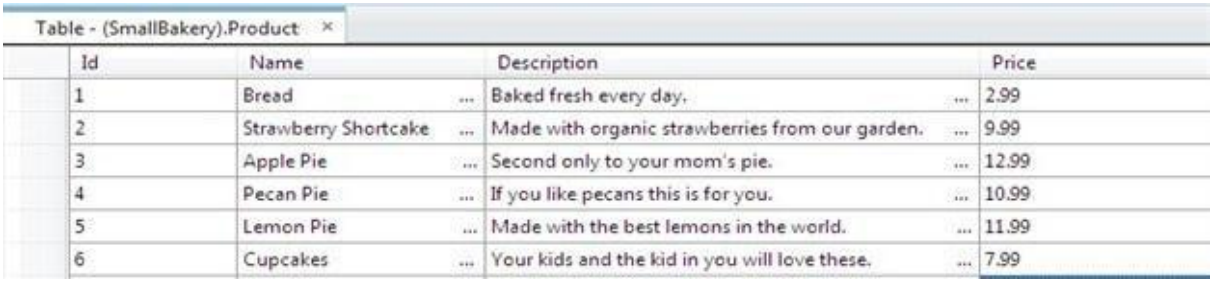


Table - (SmallBakery).Product

Id	Name	Description	Price
1	Bread	Baked fresh every day.	2.99
2	Strawberry Shortcake	Made with organic strawberries from our garden.	9.99
3	Apple Pie	Second only to your mom's pie.	12.99
4	Pecan Pie	If you like pecans this is for you.	10.99
5	Lemon Pie	Made with the best lemons in the world.	11.99
6	Cupcakes	Your kids and the kid in you will love these.	7.99

4. Zárjuk be az adatbázis adatokat tartalmazó lapot!

Adatok megjelentítése az Adatbázisból

Ha az adatbázisunk már tartalmaz adatokat, akkor meg tudjuk azokat jeleníteni egy ASP.NET weboldalon.

1. A bal oldali panelon kattintsunk a *Files* munkaterületre!
2. A weboldal gyökerében készítsünk egy új CSHTML oldalt *ListProducts.html* névvel!
3. Cseréljük ki a meglévő parancsokat a következőre:

```
@{
    var db = Database.Open("SmallBakery");
    var selectQueryString = "SELECT * FROM Product ORDER BY Name";
 }
<!DOCTYPE html>
<html>
 <head>
   <title>Small Bakery Products</title>
   <style>
       table, th, td {
         border: solid 1px #bbbbbb;
         border-collapse: collapse;
         padding: 2px;
       }
    </style>
 </head>
 <body>
   <h1>Small Bakery Products</h1>
   <table>
       <thead>
           <tr>
               <th>Id</th>
               <th>Product</th>
               <th>Description</th>
       <th>Price</th>
           </tr>
       </thead>
       <tbody>
           @foreach(var row in db.Query(selectQueryString)){
            <tr>
               <td>@row.Id</td>
                   <td>@row.Name</td>
                   <td>@row.Description</td>
                   <td>@row.Price</td>
            </tr>
           }
       </tbody>
   </table>
```

```
 </body>
</html>
```

Az első kódtömbben megnyitjuk a *SmallBakery.sdf* fájlt (adatbázis), amit korábban készítettünk el. A `Database.Open` eljárás feltételezi, hogy a *.sdf* fájl a weboldalunk *App_Data* mappájában van. (Megfigyelhetjük, hogy nem kell megadni az *.sdf* kiterjesztést – sőt, ha megadjuk, az `Open` eljárás nem is fog működni.

> **Megjegyzés**: Az *App_Data* egy speciális mappa az ASP.NET-ben, amit az adat fájlok tárolására használ. További információkért tekintsük meg a fejezet végén a Csatlakozás egy adatbázishoz részt

Ezek után készítünk egy kérelmet, amely lekérdezi a következő SQL `Select` utasítást:

```
SELECT * FROM Product ORDER BY Name
```

Az utasításban a Product azonosítja a táblát a lekérdezéshez. A * karakter határozza meg, hogy a lekérdezésnek minden oszlopot vissza kell adnia.(Az oszlopokat egyenként, vesszőkkel elválasztva is ki tudtuk volna választani, ha csak néhány oszlopot akarunk látni.) Az `Order By` kikötés azt jelzi, hogyan lesznek az adatok válogatva – ebben az esetben a *Name* oszlop szerint. Ez azt jelenti, hogy az adatok betűrend szerint lesznek rendezve minden sorban a *Name* oszlop értékei alapján.

Az oldal törzsén a jelölés egy HTML táblát hoz létre, amelyet az adatok megjelenítésére fog használni. A `<tbody>` elem belsejében használjuk a `foreach` ciklust annak érdekében, hogy egyenként megkapjunk minden adatsort, amit a lekérdezés visszaadott. Minden egyes adatsorhoz készítünk egy HTML tábla sort (`<tr>` elem). Ezután létrehozunk HTML tábla cellákat (`<td>` elem) minden oszlop számára. Minden alkalommal, mikor átmegyünk a cikluson, a következő rendelkezésünkre álló sor az adatbázisból a `row` változóban van (ezt állítottuk be a `foreach` segítségével). Ahhoz, hogy egy egyéni oszlopot kapjunk egy sorból, használhatjuk a `row.Name`-et, vagy `row.Description`-t, vagy bármilyen nevet, amit a sornak szeretnénk adni.

4. Futtassuk az oldalt a böngészőben! (Futtatás előtt győződjünk meg róla, hogy az oldalt kiválasztottuk a *Files* névmezőben.) Az oldalon megjelenik egy lista, ami a következőképpen fog kinézni:

Id	Product	Description	Price
3	Apple Pie	Second only to your mom's pie.	12.99
1	Bread	Baked fresh every day	2.99
6	Cupcakes	Your kids and the kid in you will love these.	7.99
5	Lemon Pie	Made with the best lemons in the world.	11.99
4	Pecan Pie	If you like pecans, this is for you.	10.99
2	Strawberry Shortcake	Made with organic strawberries from our garden.	9.99

Strukturált lekérdező nyelv (SQL)

A legtöbb kapcsolatos adatbázis az SQL nyelvet használja az adatok kezelésére egy adatbázisban. Ez magában foglalja a parancsokat, melyek segítségével lehet az adatokat lekérdezni és frissíteni, ezen felül adatbázistáblákat lehet létrehozni, módosítani és kezelni. Az SQL különbözik a többi

programozási nyelvtől (például attól is, amit a WebMatrixban használunk), mivel az SQL alapelve az, hogy megmondhatjuk az adatbázisnak, amit szeretnénk, és az adatbázis feladata kitalálni azt, hogy hogyan teljesítsítse a feladatot. Íme néhány példa az SQL parancsok közül, és hogy mit csinálnak:

```
SELECT Id, Name, Price FROM Product WHERE Price > 10.00 ORDER BY Name
```

Ha a Price értéke több, mint 10, akkkor kiválasztja az Id, Name és Price oszlopokat a rekordok közül a Product táblában, ezek után betűrendben visszaadja az eredményeket, ami a Name oszlop értékein alapul. Eme parancs vissza fog adni egy eredményhalmazt, amely tartalmazza a kritériumnak megfelelő rekordokat, vagy egy üres halmazt, ha nincs megfelelő rekord.

```
INSERT INTO Product (Name, Description, Price) VALUES ("Croissant", "A
flaky delight", 1.99)
```

Ezeket az új rekordokat illeszti be a Product táblába, a Name oszlop értékét „Croissant”-ra, a Description oszlopét „A flaky delight”-ra és a Price oszlopét pedig 1.99-re állítja.

```
DELETE FROM Product WHERE ExpirationDate < "01/01/2008"
```

Ez a parancs rekordokat töröl a Product táblából,melyek lejárati dátum oszlopa korábbi, mint 2008. január 01.. (Természetesen ez azt feltételezteti, hogy a Product táblának létezik ilyen oszlopa. A dátum HH/NN/ÉÉÉÉ formátumban kerül ide, de beírhatjuk a helyileg használt formátumban is.

Az `Insert Into` és a `Delete` parancsok nem dobnak ki eredményhalmazokat. Ehelyett egy számot ad vissza, ami kifejezi, hogy hány rekordot érintett ez a parancs.

Néhány ilyen művelethez (mint rekordok beillesztése és törlése) a folyamatnak szüksége van a megfelelő engedélyekre a művelet végrehajtásához. Ezért kell gyakran felhasználónévvel és jelszóval hitelesítenünk magunkat, amikor fontos adatbázisokhoz csatlakozunk.

Több tucat SQL utasítás létezik, de mindegyik ugyanazt a mintát követi, mint ez. Használhatunk SQL parancsokat adatbázistáblák készítésére, rekordok megszámlálására, árak kiszámítására, valamint számos további művelet végrehajtásához.

Adatok beillesztése egy adatbázisba

Ez a rész bemutatja, hogyan lehet létrehozni egy oldalt, amely lehetővé teszi, hogy a felhasználók új termékeket adhassanak hozzá a Product adatbázis táblához. Miután beillesztettünk egy új terméket, az előző részben létrehozott *ListProducts.html* oldal használatával az oldal megjeleníti a frissített táblát.

Az oldal tartalmaz egy hitelesítőt, ami ellenőrzi, hogy a felhasználó által beillesztett adatok érvényesek-e az adatbázisban. Például az oldalban levő kód biztosítja, hogy minden szükséges helyre kerüljön valamilyen érték.

Megjegyzés: Néhány ilyen művelethez (mint rekordok beillesztése és törlése) a folyamatnak szüksége van a megfelelő engedélyekre a művelet végrehajtásához. Ezért kell gyakran felhasználónévvel és jelszóval hitelesítenünk magunkat, amikor fontos adatbázisokhoz csatlakozunk (a WebMatrixban létrehozott tesztadatbázisunk nem ilyen).

1. A weboldalon készítsünk egy új CSHTML fájlt *InsertProducts.html* névvel!
2. Cseréljük ki a meglévő kódot a következőre:

```
@{
    var db = Database.Open("SmallBakery");
    var Name = Request["Name"];
    var Description = Request["Description"];
    var Price = Request["Price"];

    if (IsPost) {
```

```
        // Read product name.
        Name = Request["Name"];
        if (Name.IsEmpty()) {
           ModelState.AddError("Name", "Product name is required.");
        }

        // Read product description.
        Description = Request["Description"];
        if (Description.IsEmpty()) {
          ModelState.AddError("Description",
                "Product description is required.");
         }

        // Read product price
        Price = Request["Price"];
        if (Price.IsEmpty()) {
          ModelState.AddError("Price", "Product price is required.");
        }

        // Define the insert query. The values to assign to the
        // columns in the Product table are defined as parameters
        // with the VALUES keyword.
        if(ModelState.IsValid) {
            var insertQuery = "INSERT INTO Product (Name,
Description, Price) " +
                "VALUES (@0, @1, @2)";
            db.Execute(insertQuery, Name, Description, Price);
            // Display the page that lists products.
            Response.Redirect(@Href("/ListProducts"));
        }
    }
}

<!DOCTYPE html>
<html>
<head>
   <title>Add Products</title>
   <style type="text/css">
      label {float:left; width: 8em; text-align: right;
               margin-right: 0.5em;}
      fieldset {padding: 1em; border: 1px solid; width: 35em;}
      legend {padding: 2px 4px; border: 1px solid; font-weight:bold;}
      .validation-summary-errors {font-weight:bold; color:red; font-
size: 11pt;}
   </style>
</head>
<body>
   <h1>Add New Product</h1>

   @Html.ValidationSummary("Errors with your submission:")

   <form method="post" action="">
       <fieldset>
           <legend>Add Product</legend>
           <div>
               <label>Name:</label>
               <input name="Name" type="text" size="50" value="@Name"
/>
           </div>
           <div>
               <label>Description:</label>
```

```
                <input name="Description" type="text" size="50"
                    value="@Description" />
            </div>
            <div>
                <label>Price:</label>
                <input name="Price" type="text" size="50"
value="@Price" />
            </div>
            <div>
                <label> </label>
                <input type="submit" value="Insert" class="submit" />
            </div>
        </fieldset>

    </form>
</body>
</html>
```

Az oldal törzse tartalmaz egy HTML űrlapot három szövegmezővel, melyek segítségével a felhasználók neveket, leírásokat és árakat vihetnek be. Mikor a felhasználó az *Insert* gombra kattint, az oldal tetején lévő kód egy kapcsolatot fog nyitni a *SmallBakery.sdf* adatbázissal. Ezek után megkapjuk azokat az értékeket, amelyeket a felhasználó hagyott jóvá a `Request` objektummal, és hozzárendelte azok értékét a helyi változókhoz.
Annak ellenőrzésére, hogy a felhasználó minden szükséges oszlopba bevitt adatokat, tegyük ezt:

```
Name = Request["Name"];
if (Name.IsEmpty()) {
    ModelState.AddError("Name",
        "Product name is required.");
}
```

Ha a `Name` oszlop értéke üres, akkor használjuk a `ModelState.AddError` eljárást, ami kiír egy hibaüzenetet! Megismételjük ezt minden oszloppal, ha ellenőrizni szeretnénk őket. Miután minden oszlopot ellenőriztünk, végezzük el ezt a vizsgálatot:

```
if(ModelState.IsValid) { // ... }
```

Mikor minden oszlop érvényesítve lett (nem volt üres), menjünk tovább, és hozzunk létre egy SQL utasítást az adatok beillesztéséhez, majd hajtsuk ezt végre a következő módon:

```
var insertQuery =
    "INSERT INTO Product (Name, Description, Price) VALUES (@0, @1,
@2)";
```

Értékek beillesztéséhez hozzunk létre paraméteres helyőrzőket (`@0, @1, @2`).

> **Megjegyzés**: Mint biztonsági óvintézkedés, mindig fogadtassunk el értékeket egy paramétereket tartalmazó SQL utasítással, mint ahogy azt láthatjuk az előző példában. Így a bejövő adatok ellenőrzésen esnek át, és megelőzhetjük az úgynevezett SQL injekciós támadást, amikor a rosszindulatú felhasználó kártékony kódot küld a lekérdezésben.

A lekérdezés végrehajtásához használjuk ezt az állítást, ezzel átadva neki a változókat, amelyek tartalmazzák a helyőrzőket helyettesítő értékeket:

```
db.Execute(insertQuery, Name, Description, Price);
```

Miután az `Insert Into` utasítás végrehajtódott, a felhasználót a termékeket felsoroló oldalra küldjük ezzel a sorral:

```
Response.Redirect("~/ListProducts");
```

Ha a hitelesítés nem sikerült, akkor hagyjuk ki a beillesztést. Ehelyett van egy helperünk az oldalban, amely képes megjeleníteni a felgyülemlett hibaüzeneteket (ha van ilyen).

```
@Html.ValidationSummary("Errors with your submission:")
```

Figyeljük meg, hogy a stílusblokk tartalmaz egy `.validation-summary-errors` nevű CSS osztálydefiníciót! Ez annak a CSS osztálynak a neve, amelyet alapértelmezettként használunk a `<div>` elemhez, ami tartalmazza az ellenőrzési hibákat. Ebben az esetben a CSS osztály meghatározza, hogy a hitelesítési összefoglaló hibák piros színnel és félkövér betűtípussal jelenjenek meg, de meg tudjuk határozni, hogy olyan formátumban mutassa a `.validation-summary-errors` osztályt, amilyenben szeretnénk.

3. Nézzük meg az oldalt a böngészőben! Az oldalon megjelenik egy űrlap, ami hasonlít a következő illusztráción lévőhöz:

4. Írjuk be az értékeket minden oszlopba, de a *Price* oszlopot hagyjuk üresen!
5. Kattintsunk az *Insert* gombra! Az oldal egy hibaüzenetet fog megjeleníteni, mint ahogy a következő kép mutatja. (Nem készült új rekord.)

6. Töltsük ki teljesen az űrlapot, majd kattintsunk az *Insert*-re! Most a *ListProducts.cshtml* oldal fog megjelenni, ami meg is mutatja az új rekordot.

Adatok frissítése egy Adatbázisban

Miután egy adat bekerült a táblába, szükség lehet rá, hogy frissítsük. A most következő eljárás megmutatja, hogyan készítsünk két oldalt, melyek hasonlítanak az előző példában az adatok feltöltéséhez használt oldalhoz. Az első oldal a termékeket mutatja és lehetőséget ad arra, hogy a felhasználó kiválasszon egyet a változtatáshoz. A második oldal lehetővé teszi, hogy a felhasználó valóban szerkeszteni tudja azt, és el is menthesse.

Fontos: Egy élesben futó weboldalon általában korlátozzák, hogy kik szerkeszthetnek adatokat. A tagok hozzáadásáról és az egyes tagok jogainak beállításairól részletesebben a 17. fejezetben (Biztonsági elemek és felhasználói fiókok hozzáadása) olvashatunk.

1. Készítsünk egy *EditProduct.cshtml* nevű CSHTML fájlt a webhelyen!
2. Cseréljük ki a meglévő kódot a következőre:

```
@{
    var db = Database.Open("SmallBakery");
    var selectQueryString = "SELECT * FROM Product ORDER BY Name";

}
<!DOCTYPE html>
<html>
<head>
    <title>Edit Products</title>
    <style type="text/css">
        table, th, td {
          border: solid 1px #bbbbbb;
          border-collapse: collapse;
          padding: 2px;
        }
    </style>
</head>
<body>
    <h1>Edit Small Bakery Products</h1>
    <table>
      <thead>
        <tr>
          <th> </th>
          <th>Name</th>
          <th>Description</th>
          <th>Price</th>
        </tr>
      </thead>
      <tbody>
        @foreach (var row in db.Query(selectQueryString)) {
          <tr>
            <td><a href="@Href("/UpdateProducts",
row.Id)">Edit</a></td>
            <td>@row.Name</td>
            <td>@row.Description</td>
            <td>@row.Price</td>
          </tr>
        }
      </tbody>
    </table>
</body>
</html>
```

Az egyetlen különbség a mostani és a korábbi *ListProducts.cshtml* oldal között, hogy a HTML oldal táblája tartalmaz egy extra oszlopot egy *Edit* linkkel. Mikor erre a hivatkozásra kattintunk, átirányít minket az *UpdateProducts.cshtml* oldalra (amit ezt követően fogunk elkészíteni), ahol módosíthatjuk a kiválasztott rekordot.
Nézzük meg a kódot, ami elkészíti az *Edit* linket:

```
<a href="@Href("/UpdateProducts", row.Id)">Edit</a></td>
```

A kód elkészít egy HTML rögzítést (egy `<a>` elemet), aminek a `href` tulajdonsága dinamikusra van állítva. A `href` attribútum köti ki, hogy az oldal megjelenjen, mikor a felhasználó rákattint a hivatkozásra. Ugyancsak ez adja át az aktuális sor `Id` értékét a linknek. Mikor fut az oldal, az oldal forrása ezekhez hasonló linkeket tartalmazhat:

```
<a href="UpdateProducts/1">Edit</a></td>
<a href="UpdateProducts/2">Edit</a></td>
<a href="UpdateProducts/3">Edit</a></td>
```

Figyeljük meg, hogy a `href` tulajdonság `UpdateProducts/n`-re lett állítva, ahol *n* a termékszám. Mikor a felhasználó valamelyik hivatkozásra kattint ezek közül, a megjelenő URL ehhez hasonlóan fog kinézni:

http://localhost:18816/UpdateProducts/6

Más szavakkal, a szerkesztés alatt álló termékszám átadásra kerül az URL-be.

3. Nézzük meg az oldalt egy böngészőben! Az adatok a következő képhez hasonlóan fognak megjelenni:

	Name	Description	Price
Edit	Apple Pie	Second only to your mom's pie.	12.99
Edit	Bread	Baked fresh every day.	2.99
Edit	Cherry Pie	Made with organic cherries from our garden.	8.99
Edit	Cupcakes	Your kids and the kid in you will love these.	7.99
Edit	Lemon Pie	Made with the best lemmons in the world.	11.99
Edit	Pecan Pie	If you like pecans this is fro you.	10.99
Edit	Strawberry Shortcake	Made with organic strawberries from our garden.	9.99

Ezek után létrehozzuk az oldalt, amin a felhasználó ténylegesen frissíteni tudja az adatokat. A frissítési oldal tartalmaz egy érvényesítést, amely ellenőrzi a felhasználó által bevitt adatokat. Például a kód az oldalon gondoskodik arról, hogy minden szükséges oszlopba kerüljön érték.

4. Készítsünk egy *UpdateProducts.cshtml* nevű CSHTML fájlt a weboldalon!
5. Cseréljük ki a meglévő kódokat a következőre:

```
@{
    var db = Database.Open("SmallBakery");
    var selectQueryString = "SELECT * FROM Product WHERE Id=@0";

    var ProductId  = UrlData[0];

    if (ProductId.IsEmpty()) {
         Response.Redirect(@Href("/EditProducts"));
     }

    var row = db.QuerySingle(selectQueryString, ProductId);

    var Name = row.Name;
    var Description = row.Description;
    var Price = row.Price;

    if (IsPost) {
         Name = Request["Name"];
         if (String.IsNullOrEmpty(Name)) {
           ModelState.AddError("Name", "Product name is required.");
         }

        Description = Request["Description"];
        if (String.IsNullOrEmpty(Description)) {
          ModelState.AddError("Description",
              "Product description is required.");
        }

        Price = Request["Price"];
        if (String.IsNullOrEmpty(Price)) {
```

```
          ModelState.AddError("Price", "Product price is required.");
        }

        if(ModelState.IsValid) {
            var updateQueryString =
              "UPDATE Product SET Name=@0, Description=@1, Price=@2
WHERE Id=@3" ;
            db.Execute(updateQueryString, Name, Description, Price,
ProductId);
            Response.Redirect(@Href("/EditProducts"));
        }
    }
}

<!DOCTYPE html>
<html>
<head>
    <title>Add Products</title>
    <style type="text/css">
       label { float: left; width: 8em; text-align: right;
                margin-right: 0.5em;}
       fieldset { padding: 1em; border: 1px solid; width: 35em;}
       legend { padding: 2px 4px;  border: 1px solid; font-weight:
bold;}
       .validation-summary-errors {font-weight:bold; color:red; font-
size:11pt;}
    </style>
</head>
<body>
    <h1>Update Product</h1>

    @Html.ValidationSummary("Errors with your submission:")

    <form method="post" action="">
        <fieldset>
            <legend>Update Product</legend>
            <div>
                <label>Name:</label>
                <input name="Name" type="text" size="50"
value="@Name" />
            </div>
            <div>
                <label>Description:</label>
                <input name="Description" type="text" size="50"
                   value="@Description" />
            </div>
            <div>
                <label>Price:</label>
                <input name="Price" type="text" size="50"
value="@Price" />
            </div>
            <div>
                <label> </label>
                <input type="submit" value="Update" class="submit" />
            </div>
        </fieldset>
    </form>
</body>
</html>
```

Az oldal törzse tartalmaz egy HTML űrlapot, ahol egy termék van kijelezve, és ahol a felhasználó szerkeszteni tudja azt. Ahhoz, hogy megjelenjen a termék, használjuk ezt az SQL utasítást:

```
SELECT * FROM Product WHERE Id=@0
```

Ez ki fogja választani azt a terméket, amelyiknek az Id-je megegyezik azzal az értékkel, amelyik a `@0` paraméter helyére került. (Mivel az Id az elsődleges kulcs, és ezen felül egyéni is, egyszerre csak egy termék rekordot lehet kiválasztani ezzel az eljárással.) Ahhoz, hogy megkapjuk azt az Id értéket, amelyet ennek a `Select` utasításnak adtunk meg, ki tudjuk olvasni az értéket, ami át lett adva az oldalnak, mint az URL része, használjuk a következő szintaxist:

```
var ProductId = UrlData[0];
```

Hogy megérkezzen a termék rekordja, használjuk a `QuerySingle` metódust, amely csak egy rekordot fog adni:

```
var row = db.QuerySingle(selectQueryString, ProductId);
```

Az egyetlen sor visszakerül a `row` változóba. Kivehetünk adatokat minden egyes oszlopból, és azt átadja a helyi változóknak, mint ez:

```
var Name = row.Name;
var Description = row.Description;
var Price = row.Price;
```

Az űrlap utasításában ezek az értékek jelennek meg automatikusan az egyes szövegmezőkben, a következő beágyazott kódot használva:

```
<input name="Name" type="text" size="50" value="@Name" />
```

A kód ezen része jelzi ki azt a rekordot, amelyik frissítésre kerül. Miután a rekord megjelent, a felhasználó szerkeszteni tudja az egyes oszlopokat.

Mikor a felhasználó elküldi az űrlapot az *Update* gombra kattintva, az `if(IsPost)` tömbben lévő kód lefut. Ez megkapja az értékeket a `Request` objektumtól, majd elraktározza azokat a változókban és ellenőrzi, hogy minden oszlop ki van-e töltve. Ha a hitelesítés sikerrel járt, a kód a következő SQL Update utasítást hozza létre:

```
UPDATE Product SET Name=@0, Description=@1, Price=@2, WHERE ID=@3
```

Egy SQL Update utasításban meghatározzuk minden oszlopnak, hogyan frissítsen, és állítsa be annak értékét. Ebben a kódban az értékek meg vannak határozva a paraméteres helyőrzőkkel: `@0`, `@1`,`@2` és így tovább. (Mint korábban említettük, a biztonság kedvéért mindig fogadtassuk el az értékeket egy paramétereket tartalmazó SQL utasítással!)

Amikor a `db.Execute` módszert használjuk, átadjuk a változókat, amelyek tartalmazzák az értékeket abban a sorrendben, mely megfelel az SQL utasításban lévő paraméterek sorrendjének:

```
db.Execute(updateQueryString, Name, Description, Price, ProductId);
```

Miután az Update utasítás végrehajtásra került, a következő módszert használjuk annak érdekében, hogy a felhasználót visszairányítsuk a szerkesztési oldalra.

```
Response.Redirect(@Href("~/EditProducts"));
```

Amit a felhasználó látni fog, az a frissített adatok listája az adatbázisban, és egy másik termék szerkesztésére is lehetősége nyílik.

6. Mentsük el az oldalt!
7. Futtassuk az *EditProducts.cshtml* oldalt (ne az update oldalt), majd kattintsunk az *Edit*-re, hogy egy terméket szerkeszthessünk! Megjelenik az *UpdateProducts.cshtml* és a kiválasztott terméket fogja mutatni.

8. Végezzük el a változtatást, majd kattintsunk az *Update*-re! A termékek listáját fogja mutatni, de most már a frissített adatokkal.

Adatok törlése egy Adatbázisból

Ez a rész bemutatja, hogyan tud a felhasználó egy terméket törölni a *Product* adatbázistáblából. A példa két oldalból áll. Az első oldal az, ahol a felhasználó kiválasztja a törölni kívánt rekordot. A törölendő rekord ezután egy második oldalon fog megjelenni, ami a törlés megerősítésére kéri fel a felhasználót.

Fontos: Egy élesben futó weboldalon általában korlátozzák, hogy kik szerkeszthetnek adatokat. A tagok hozzáadásáról és az egyes tagok jogainak beállításairól részletesebben a 17. fejezetben (Biztonsági elemek és felhasználói fiókok hozzáadása) olvashatunk.

1. Készítsünk egy *ListProductsForDelete.cshtml* nevű CSHTML fájlt a weboldalon!
2. Cseréljük ki a meglévő kódot a következőre:

```
@{
  var db = Database.Open("SmallBakery");
  var selectQueryString = "SELECT * FROM Product ORDER BY Name";
}
<!DOCTYPE html>
<html>
<head>
    <title>Delete a Product</title>
    <style>
        table, th, td {
          border: solid 1px #bbbbbb;
          border-collapse: collapse;
          padding: 2px;
        }
     </style>
</head>
<body>
  <h1>Delete a Product</h1>
  <form method="post" action="" name="form">
    <table border="1">
      <thead>
        <tr>
          <th> </th>
          <th>Name</th>
          <th>Description</th>
          <th>Price</th>
```

```
        </tr>
      </thead>
      <tbody>
        @foreach (var row in db.Query(selectQueryString)) {
          <tr>
            <td><a href="@Href("/DeleteProduct",
row.Id)">Delete</a></td>
            <td>@row.Name</td>
            <td>@row.Description</td>
            <td>@row.Price</td>
          </tr>
        }
      </tbody>
    </table>
  </form>
</body>
</html>
```

Az oldalunk hasonló az EditProducts.cshtml oldalhoz, amit korábban készítettünk. Azonban ahelyett, hogy egy Edit gomb jelenne meg minden terméknél, most egy Delete linket találunk. A Delete link a következő beágyazott kód az utasításban:

```
<a href="@Href("~/DeleteProduct", row.Id)">Delete</a>
```

Ez kiad egy URL-t, ami hasonlóan néz ki, ha egy felhasználó a hivatkozásra kattint:

```
http://<server>/DeleteProduct/4
```

Az URL előhívja a DeleteProducts.cshtml oldalt (amit a következőkben fogunk elkészíteni) és továbbadja a termék Id-jét a törléshez (itt a négyest).

3. Mentsük el a fájlt, de hagyjuk nyitva!
4. Készítsünk egy másik CSHTML fájlt *DeleteProducts.cshtml* névvel, és cseréljük ki a meglévő tartalmat a következőre:

```
@{
  var db = Database.Open("SmallBakery");
  var ProductId = UrlData[0];
  if (ProductId.IsEmpty()) {
    Response.Redirect(@Href("/ListProductsForDelete"));
  }
  var prod = db.QuerySingle("SELECT * FROM PRODUCT WHERE ID = @0",
ProductId);
  if( IsPost && !ProductId.IsEmpty()) {
    var deleteQueryString = "DELETE FROM Product WHERE Id=@0";
    db.Execute(deleteQueryString, ProductId);
    Response.Redirect("/ListProductsForDelete");
  }
}

<!DOCTYPE html>
<html
<head>
    <title>Delete Product</title>
</head>
<body>
  <h1>Delete Product - Confirmation</h1>
  <form method="post" action="" name="form">
    <p>Are you sure you want to delete the following product?</p>

    <p>Name: @prod.Name <br />
       Description: @prod.Description <br />
```

```
        Price: @prod.Price</p>
    <p><input type="submit" value="Delete" /></p>
  </form>
</body>
</html>
```

Az oldal ez után megkéri a felhasználót, hogy kattintson egy gombra a törlés megerősítéséhez. Ez egy fontos biztonsági lépés. Amikor kritikus műveleteket végzünk, például rekordok módosítása vagy törlése, mindig a POST művelethez kössük, ne a GET művelethez. Ha a webhelyünkben a GET művelettel elvégezhető ilyen művelet, bárki egy http://<server>/DeleteProduct/4 Url beírásával törölhetne adatokat az adatbázisból. A jóváhagyó oldallal beépített plusz lépcső a POST módszerrel hagyja jóvá ezeket a kritikus műveleteket, így megelőzve a véletlen adatvesztést.

A tényleges törlést a következő kód hajtja végre. A kód ellenőrzi azt is, hogy ez egy POST művelet, és hogy az adott ID nem üres-e:

```
if( IsPost && !ProductId.IsEmpty()) {
    var deleteQueryString = „DELETE FROM Product WHERE Id=@0”;
    db.Execute(deleteQueryString, ProductId);
    Response.Redirect(„/ListProductsForDelete”);
}
```

A kód egy SQL utasítást futtat, ami törli a megadott rekordot, majd visszairányítja a felhasználót a lista oldalára.

5. Futtassuk a *ListProductsForDelete.cshtml* oldalt a böngészőben!

	Name	Description	Price
Delete	Apple Pie	Second only to your mom's pie.	12.99
Delete	Bread	Baked fresh every day.	2.99
Delete	Cherry Pie	Made with organic cherries from our garden.	8.99
Delete	Cupcakes	Your kids and the kid in you will love these.	7.99
Delete	Lemon Pie	Made with the best lemmons in the world.	11.99
Delete	Pecan Pie	If you like pecans this is fro you.	10.99
Delete	Strawberry Shortcake	Made with organic strawberries from our garden.	9.99

6. Kattintsunk a *Delete* linkre valamelyik termék mellett! Ekkor megjelenik a *DeleteProducts.cshtml* oldal, hogy megerősítsük a rekord törlését.
7. Kattintsunk a *Delete* gombra! A termék rekordja törlődik, majd az oldal frissül az új terméklistával.

Csatlakozás egy adatbázishoz

Kétféle módon csatlakozhatunk egy adatbázishoz. Az első, hogy használjuk a `Database.Open` eljárást és meghatározzuk az adatbázis nevét (lehagyva a *.sdf* kiterjesztést):

```
var db = Database.Open("SmallBakery");
```

Az `Open` eljárás azt feltételezi, hogy a *.sdf* fájl a weboldal *App_Data* mappájában van. Ennek a mappának számos jellemzője van, kimondottan az adatok tárolására tervezve. Például rendelkezik a megfelelő jogosultságokkal, hogy engedje a weboldalnak az adatok olvasását és írását, és mint biztonsági intézkedés, a WebMatrix nem teszi lehetővé a hozzáférést a fájloknak ehhez a mappához.

A második út, hogy használjunk egy úgynevezett connection stringet. A connection string információkat tartalmaz az adatbázishoz való csatlakozáshoz. Ez magában foglalhatja a fájl elérési útját, vagy tartalmazhatja az SQL Server adatbázis nevét egy helyi vagy távoli szerveren, és a

szerverhez való csatlakozáshoz szükséges felhasználónevet és jelszót. (Hogyha az adatokat egy központilag kezelt SQL Serveren tároljuk, például egy tárhelyet szolgáltató oldalon, akkor mindig a connection stringet használjuk, hogy megadjuk az adatbázis-kapcsolat információit.

Gyakran a WebMatrixban a connection string egy *Web.config* nevű XML fájlba van elraktározva. Ahogy a neve is mutatja, használhatjuk a Web-config fájlt a weboldal gyökerében konfigurációs információk tárolására, beleértve bármilyen connection stringet, amire az webhelyünknek szüksége lehet. Egy példa a Web.config fájlban lévő connection stringre:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <connectionStrings>
   <add
     name="SQLServerConnectionString"
     connectionString=
"server=myServer;database=myDatabase;uid=username;pwd=password"
     providerName="System.Data.SqlClient" />
  </connectionStrings>
</configuration>
```

A példában a connection string rámutat az SQL Server egyik példányában lévő adatbázisra, amelyik valahol a szerveren fut (nem pedig egy helyi *.sdf* fájl). Szükségünk lehet arra, hogy behelyettesítsük a megfelelő neveket a `myServer` és `myDatabase` helyére, és megadjuk az `SQL` Server belépési értékeit a felhasználónévnek és jelszónak. (A felhasználónév és jelszó értékeinek nem feltétlenül kell megegyeznie a saját Windows-os hitelesítő adatainkkal vagy azzal az értékkel, amelyet a tárhely szolgáltató adott nekünk, hogy csatlakozhassunk a szervereikre. Ellenőriztessük a rendszergazdával a szükséges értékeket!)

A `Database.Open` eljárás rugalmas, mivel lehetővé teszi, hogy elfogadjon mind *.sdf* adatbázis fájl nevet, vagy a *Web.config* fájlban tárolt connection string nevet. A következő példa bemutatja, hogyan csatlakozhatunk az adatbázishoz az előző példában szemléltetett connection stringet használva.

```
@{
    var db = Database.Open("SQLServerConnectionString");
}
```

Ahogy megfigyelhettük, a `Database.Open` eljárás lehetővé teszi, hogy elfogadjon egy adatbázis nevet vagy egy connection stringet, és majd kitalálja, melyiket használja. Ez igen hasznos, mikor felvonultatjuk (közzétesszük) a weboldalunkat. Használhatunk egy *App_Data* mappában lévő *.sdf* fájlt, mikor közzétesszük, és teszteljük az oldalunkat. Ezután, mikor az oldalunkat egy gyártási szerverre visszük fel, használhatjuk a connection stringet a *Web.config* fájlból, aminek ugyanaz a neve, mint a *.sdf* fájlunknak, de az a tárhely szolgáltató adatbázisára mutat – mind anélkül, hogy megváltoztattuk volna a kódot.

Végezetül, ha közvetlen a connection stringgel szeretnénk dolgozni, használhatjuk a `Database.OpenConnectionString` eljárást és megadhatjuk az aktuális connection stringet a *Web.config* fájlban tárolt neve helyett. Ez hasznosnak bizonyulhat olyan szituációkban, mikor nincsen hozzáférésünk a connection stringhez (vagy a benne levő értékekhez, mint a *.sdf* fájl neve), amíg az oldal fut. A legtöbb esetben azonban használhatjuk a `Database.Open`-t, mint ahogy ebben a fejezetben le volt írva.

További forrás angolul

SQL Server Compact

7. fejezet – Adatok megjelenítése gridekben

Az előző fejezetben megtanultuk, hogyan tároljunk adatokat egy adatbázisban, és hogyan dolgozzunk velük. Ebben a fejezetben megtudhatjuk, hogyan jelenítsünk meg adatokat egy HTML táblázatban, úgynevezett griden.

Az útmutatóból megtudhatjuk:

- Hogyan jelenítsünk meg adatokat egy weboldalon a `WebGrid` helper használatával?
- A griden megjelenített adatok formázása.
- Lapozás hozzáadása a gridhez.

A fejezetben bemutatott az ASP.NET weboldal funkció:

- A `WebGrid` helper

A WebGrid Helper

Az előző fejezetben saját kezűleg jelenítettünk meg adatokat az oldalon. Azonban van egy könnyebb mód is az adatok megjelenítésének – használjuk a `WebGrid` helpert! A helper le tud renderelni egy HTML táblát az adatok kijelzésére. A helper támogatja a formázási beállításokat, és lehetővé teszi a felhasználónak, hogy lapozzon a hosszú adatsorok között, és hogy rendezze az adatokat az oszlopok fejléceire kattintva.

Adatok megjelenítése a WebGrid Helper használatával

Ez az feladat megmutatja nekünk, hogyan jelenítsünk meg adatokat a `WebGrid` helperben annak legegyszerűbb beállításait használva.

1. Nyissuk meg az 6. fejezet – Műveletek adatokkal című fejezetben készített weboldalt.
   Ha nem csináltuk végig az előző fejezetben leírtakat, nem kell mindet pótolnunk, csupán arra van szükség, hogy létrehozzuk a *SmallBakery.sdf* adatbázis fájlt, amit az 5. fejezet elején készítettünk el. Ennek a fájlnak a weboldal *App_Data mappában* kell lennie.
2. Készítsünk egy új, *WebGridBasic.cshtml* nevű CSHTML fájlt a weboldalon!
3. Helyettesítsük a meglévő kódokat a következővel:

```
@{
    var db = Database.Open("SmallBakery") ;
    var selectQueryString = "SELECT * FROM Product ORDER BY Id";
    var data = db.Query(selectQueryString);
    var grid = new WebGrid(data);
}
<!DOCTYPE html>
<html>
    <head>
        <title>Displaying Data Using the WebGrid Helper</title>
    </head>
    <body>
        <h1>Small Bakery Products</h1>
        <div id="grid">
            @grid.GetHtml()
        </div>
    </body>
</html>
```

A kód először megnyitja a *SmallBakery.sdf* adatbázist, majd egy SQL `Select` utasítást hoz létre:

```
SELECT * FROM Product ORDER BY Id
```

A `data` nevű változót az SQL `Select` utasítás által szerzett értékek töltik fel. A `WebGrid` helpert ezek után új gridek készítésére használjuk:

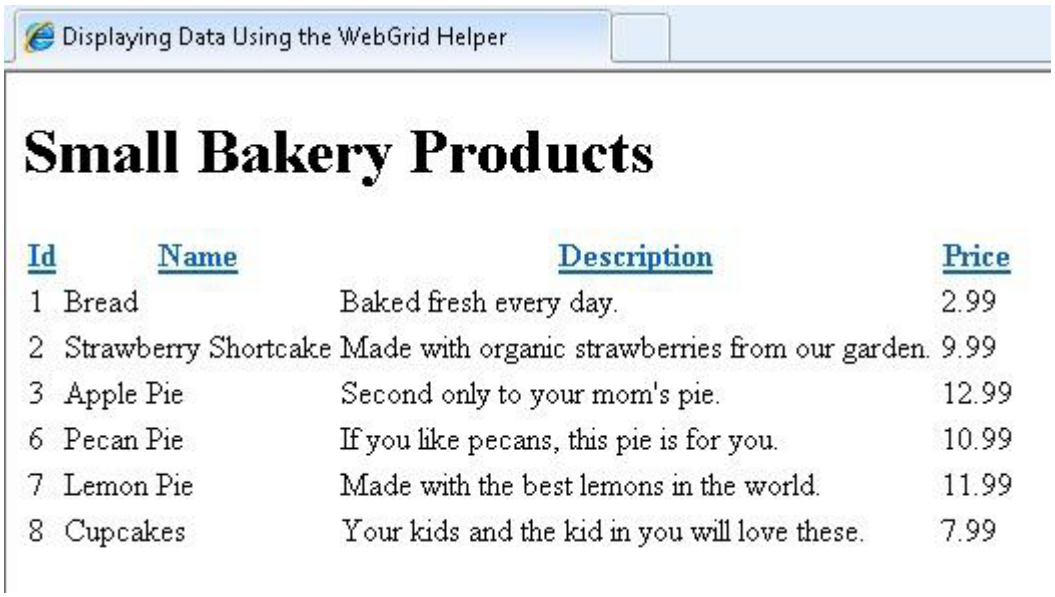
```
var data = db.Query(selectQueryString);
var grid = new WebGrid(data);
```

A kód egy új `WebGrid` objektumot hoz létre és hozzárendeli a `grid` változóhoz. Az oldal törzsében a `WebGrid`-et használva rendereljük az adatokat, ezt a kódot alkalmazva:

```
@grid.GetHtml()
```

A `grid` változó az az érték, ami a `WebGrid` objektum létrehozásakor készült.

4. Futtassuk az oldalt! (Futtatás előtt győződjünk meg arról, hogy az oldal ki van választva a Files névmezőben!) A `WebGrid` helper elkészít egy HTML táblát, ami az SQL `Select` utasítás által kiválasztott adatokat tartalmazza:



Figyeljük meg, lehetőségünk van az oszlopnevekre kattintani, hogy rendezzük a tábla adatait a megfelelő oszlop szerint.

Mint láthatjuk, ha a lehető legegyszerűbb kódot is használjuk, a `WebGrid` helper rengeteg műveletet végrehajt, mikor kijelzi (és szétválogatja) az adatokat. A helper emellett még számos dologra képes. A fejezet hátralévő részéből megtudhatjuk, hogyan állítsuk be a `WebGrid` helpert, hogy végrehajtsa a következőket:

- Meghatározza, hogy melyik adatoszlopok jelenjenek meg és milyen formázás mellett.
- Egységes stílust ad az egész gridnek.
- Lapozzon adatokon keresztül.

Oszlopok kiválasztása és formázása megjelenítéshez

Alapértelmezetten a `WebGrid` helper minden oszlopot megjelenít, amit az SQL lekérdezése kiadott. Az adatok kijelzését a következő úton szabhatjuk testre:

- Meghatározhatjuk, melyik oszlopokat jelenítse meg a helper és milyen sorrendben. Ezt akkor használhatjuk, ha a lekérdezett adatok csak egy részét szeretnénk megjeleníteni.
- Meghatározhatjuk a formázási utasításokat, hogy hogyan jelenhetnek meg az adatok – írjuk ki például a pénznem jelét (például a $ jelet) a pénzmennyiségeket jelző oszlopokban!

Ebben az eljárásban az oszlopok különálló formázására fogjuk használni a `WebGrid` helpert.

1. Hozzunk létre egy új, *WebGridColumnFormat.cshtml* nevű oldalt a webhelyen!
2. Helyettesítsük a meglévő kódokat a következővel:

```
@{
    var db = Database.Open("SmallBakery") ;
    var selectQueryString = "SELECT * FROM Product ORDER BY Id";
    var data = db.Query(selectQueryString);
    var grid = new WebGrid(data);
}
<!DOCTYPE html>
<html>
    <head>
        <title>Displaying Data Using the WebGrid Helper (Custom
Formatting)</title>
        <style type="text/css">
            .product { width: 200px; font-weight:bold;}
        </style>
    </head>
    <body>
        <h1>Small Bakery Products</h1>
        <div id="grid">
            @grid.GetHtml(
                columns: grid.Columns(
                    grid.Column("Name", "Product", style: "product"),
                    grid.Column("Description",
format:@<i>@item.Description</i>),
                    grid.Column("Price",
format:@<text>$@item.Price</text>)
                )
            )
        </div>
    </body>
</html>
```

Ez a példa hasonlít az előzőhöz, annyi különbséggel, hogy amikor a grid-et lerendereljük a oldal törzsében a grid.`GetHtml` segítségével, akkor meghatározzuk, hogy melyik oszlop jelenjen meg és hogyan. A következő kód mutatja meg, hogyan határozzuk meg, melyik oszlopok jelenjenek meg és milyen sorrendben:

```
@grid.GetHtml(
    columns: grid.Columns(
        grid.Column("Name", "Product", style: "product"),
        grid.Column("Description", format:@<i>@item.Description</i>),
        grid.Column("Price", format:@<text>$@item.Price</text>)
)
```

Hogy megmondjuk a helpernek, melyik oszlopok jelenjenek meg, be kell foglalnunk a `columns` paramétert a `WebGrid` helper `GetHtml` metódusában. Ebben a gyűjteményben meghatározhatjuk, melyik oszlopot foglalja magában. Itt meghatározhatunk minden egyes megjeleníteni kívánt oszlopot a `grid.Column` objektummal, az oszlop nevének beírásával. Ebben a példában a `WebGrid` objektum három oszlopot jelenít meg: *Name*, *Description* és

Price. (Ezeknek az oszlopoknak jelen kell lenniük az SQL lekérdezés eredményei között – a helper nem tud megjeleníteni olyan oszlopokat, amelyek nem szerepeltek a lekérdezésben.)

Vegyük észre, hogy amellett, hogy megjelenítünk egy oszlopot, továbbíthatunk további formázási utasításokat is. A példában a kód a *Name* oszlopot jeleníti meg a következő kódot használva:

```
grid.Column("Name", "Product", style: "product")
```

Ez a következőket határozza meg a `WebGrid` számára:

- Mutassa a *Name* adatoszlop értékeit!
- Írja ki a „Product” stringet az oszlop fejléceként az eredeti helyett (ami ebben az esetben „Name” lenne)!
- Alkalmazza a „product” nevű CSS stílusosztályt! A példaként vett oldal utasításaiban a CSS osztály az oszlopok szélességét (200 pixel) és a betűtípusát (félkövér) állítja be.
  A *Description* oszlophoz a példa a következő kódot használja:

```
grid.Column("Description", format:@<i>@item.Description</i>)
```

Ez határozza meg a helpernek, hogy a *Description* oszlopot jelenítse meg. Itt olyan formátumot állít be, ami sortöréssel több sorba rendezi a szöveget, ha az túl hosszú lenne:

```
@<i>@item.Description</i>
```

A *Price* oszlop példája további változatait mutatja meg annak, hogyan határozhatjuk meg a `format` tulajdonságait:

```
grid.Column("Price", format:@<text>$@item.Price</text>)
```

Ez ismét meghatároz néhány HTML utasítást, hogy rendereljen és hozzáadjon egy dollárjelet ($) az oszlop értéke elé.

3. Futtassuk az oldalt egy böngészőben!

Ebben az esetben csak három oszlopot láthatunk. A *Name* oszlop testreszabja az oszlop fejlécét, méretét, valamint betűtípusát. A *Description* oszlop dőlt és a *Price* oszlop most már tartalmaz egy dollár jelet.

A teljes grid formázása

Amellett, hogy meg tudjuk határozni a különálló oszlopoknak, hogyan jelenjenek meg, formázhatjuk az egész gridet is. Ehhez meg kell határoznunk CSS osztályokat, amik kikötik, hogyan nézzen ki a lerenderelt HTML tábla.

1. Készítsünk egy *WebGridTableFormat.cshtml* nevű lapot a webhelyen!
2. Helyettesítsük a meglévő kódokat a következővel:

```
@{
    var db = Database.Open("SmallBakery");
    var selectQueryString = "SELECT * FROM Product ORDER BY Id";
    var data = db.Query(selectQueryString);
    var grid = new WebGrid(source: data, defaultSort: "Name");
}
<!DOCTYPE html>
<html>
    <head>
        <title>Displaying Data Using the WebGrid Helper (Custom Table
Formatting)</title>
        <style type="text/css">
            .grid { margin: 4px; border-collapse: collapse; width:
600px; }
            .head { background-color: #E8E8E8; font-weight: bold;
color: #FFF; }
            .grid th, .grid td { border: 1px solid #C0C0C0; padding:
5px; }
            .alt { background-color: #E8E8E8; color: #000; }
            .product { width: 200px; font-weight:bold;}
        </style>
    </head>
    <body>
        <h1>Small Bakery Products</h1>
        <div id="grid">
            @grid.GetHtml(
                tableStyle: "grid",
                headerStyle: "head",
                alternatingRowStyle: "alt",
                columns: grid.Columns(
                    grid.Column("Name", "Product", style: "product"),
                    grid.Column("Description",
format:@<i>@item.Description</i>),
                    grid.Column("Price",
format:@<text>$@item.Price</text>)
                )
            )
        </div>
    </body>
</html>
```

   Ez a kód az előző példára épül, megmutatva, hogyan készítsünk új stílus osztályokat (`grid`, `head`, `grid th`, `grid td`, `alt`). Ezek után a `grid.GetHtml` hozzárendeli a stílusokat a grid különböző elemeihez a `tableStyle`, `headerStyle`,valamint `alternatingRowStyle` paramétereket használva.
3. Tekintsük meg az oldalt egy böngészőben! Ebben az esetben a grid különböző stílusok használatával jelenik meg, melyek az egész táblára érvényesek, például a páratlan sorok színezése:

Adatok lapozása

Ahelyett, hogy egyszerre kijeleztetnénk minden adatot a grid-en, lehetővé tesszük a felhasználó számára, hogy lapozhasson az adatok között. Kevés adat mellett a lapozás nem olyan fontos, több száz vagy ezer adatsoros gridek esetén azonban jól jöhet.

Hogy hozzáadjuk a lapozás lehetőségét a lerenderelt grid-hez, meg kell határoznunk további paramétereket a `WebGrid` helpernek.

1. Készítsünk egy *WebGridPaging.cshtml* nevű oldalt!
2. Helyettesítsük a meglévő kódokat a következővel:

```
@{
    var db = Database.Open("SmallBakery");
    var selectQueryString = "SELECT * FROM Product ORDER BY Id";
    var data = db.Query(selectQueryString);
    var grid = new WebGrid(source: data,
                           defaultSort: "Name",
                           rowsPerPage: 3);
}
<!DOCTYPE html>
<html>
    <head>
        <title>Displaying Data Using the WebGrid Helper (with
Paging)</title>
        <style type="text/css">
            .grid { margin: 4px; border-collapse: collapse; width:
600px; }
            .head { background-color: #E8E8E8; font-weight: bold;
color: #FFF; }
            .grid th, .grid td { border: 1px solid #C0C0C0; padding:
5px; }
            .alt { background-color: #E8E8E8; color: #000; }
            .product { width: 200px; font-weight:bold;}
        </style>
    </head>
    <body>
        <h1>Small Bakery Products</h1>
```

```
        <div id="grid">
            @grid.GetHtml(
                tableStyle: "grid",
                headerStyle: "head",
                alternatingRowStyle: "alt",
                columns: grid.Columns(
                    grid.Column("Name", "Product", style: "product"),
                    grid.Column("Description",
format:@<i>@item.Description</i>),
                    grid.Column("Price",
format:@<text>$@item.Price</text>)
                )
            )
        </div>
    </body>
</html>
```

A kód kibővíti az előző példát egy `rowsPerPage` paraméter hozzáadásával, amikor a `WebGrid` objektum készül. Ez a paraméter lehetővé teszi számunkra, hogy beállítsuk a megjelenítendő sorok számát. Automatikusan engedélyezve lesz a lapozás onnantól, hogy beillesztettük ezt a paramétert.

3. Nézzük meg az oldalt egy böngészőben! Figyeljük meg, hogy csak három sor jelenik meg. A grid alján láthatjuk a kezelőfelületet, aminek segítségével átlapozhatunk a többi adatsorhoz.

További források

5. fejezet – Műveletek adatokkal

8. fejezet – Adatok megjelenítése diagramokon

Angolul:

ASP.NET Web Pages with Razor Syntax Reference

8. fejezet – Adatok megjelenítése diagramokon

Az előző fejezetekben megtanulhattuk, hogy kell adatokat megjeleníteni manuálisan és táblázatként. Ebben a fejezetben megtanuljuk, hogyan jeleníthetjük meg az adatokat diagramokon.

Az útmutatóból megtudhatjuk:

- Hogyan ábrázoljunk adatokat diagramokon?
- Hogyan formázzuk a diagramokat a beépített témák segítségével?
- Hogyan mentsük el a diagramokat ,és hogyan rakjuk őket a gyorsítótárba a jobb teljesítmény elérése érdekében?

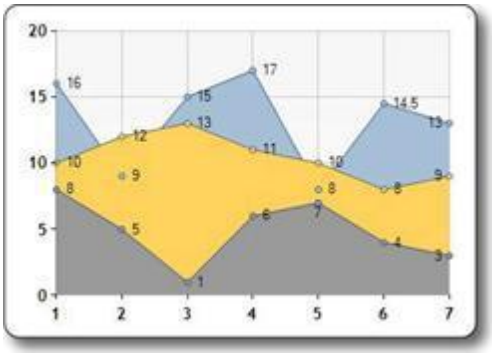
Az alábbi ASP.NET programozási funkciókat ismerhetjük meg a fejezetben:

- A `Chart` helper.

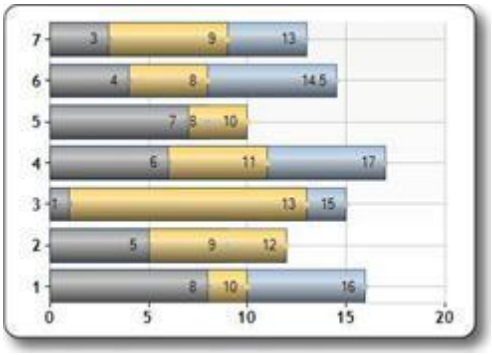
A Chart helper

Ha grafikus formában akarunk adatokat megjeleníteni, akkor erre használhatjuk a `Chart` helpert. A `Chart` helper létrehoz egy képet, ami különféle grafikonokon jeleníti meg az adatokat. Számos különféle lehetőséget kínál a formázásra és feliratozásra. A `Chart` helper több mint 30 féle grafikont tud készíteni, többek között azokat a típusokat is, melyeket már a Microsoft Excelből vagy egyéb alkalmazásokból ismerhetünk, például a területdiagram, sávdiagram, oszlopdiagram, grafikon, kördiagram és több speciális diagram, mint például az árfolyamdiagram.

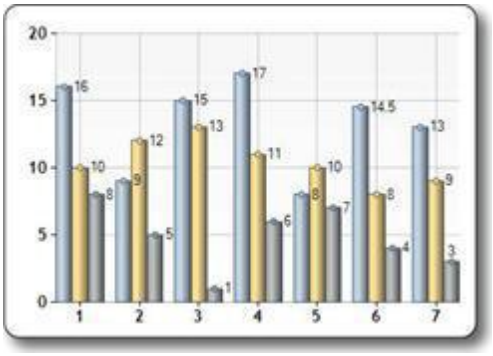
Területdiagram



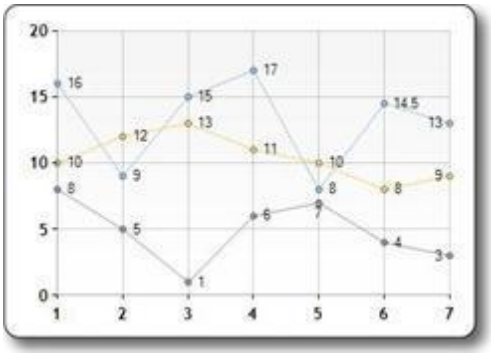
Sávdiagram



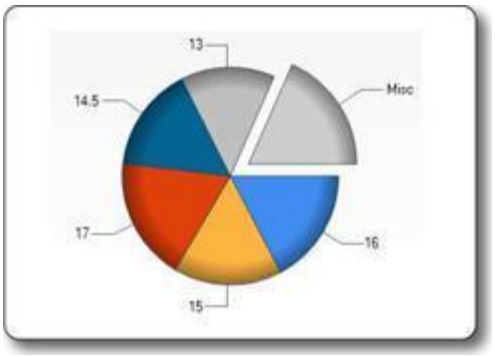
Oszlopdiagram



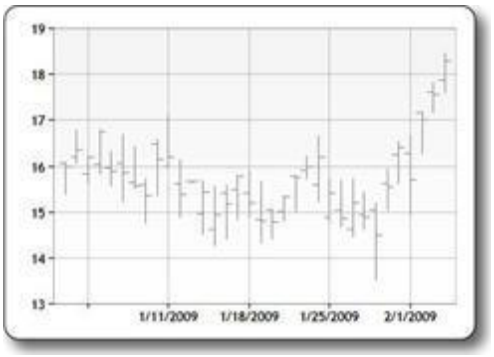
Grafikon



Kördiagram



Árfolyamdiagram



A diagram elemei

A diagramok adatokat és számos egyéb elemet tartalmaznak, például jelmagyarázatokat, tengelyeket, adatsorokat, stb. A következő képen láthatjuk ezeket az elemeket, melyek a `Chart` helper segítségével testreszabhatóak. Ez a fejezet bemutatja, hogyan állíthatjuk be ezeket az elemeket.

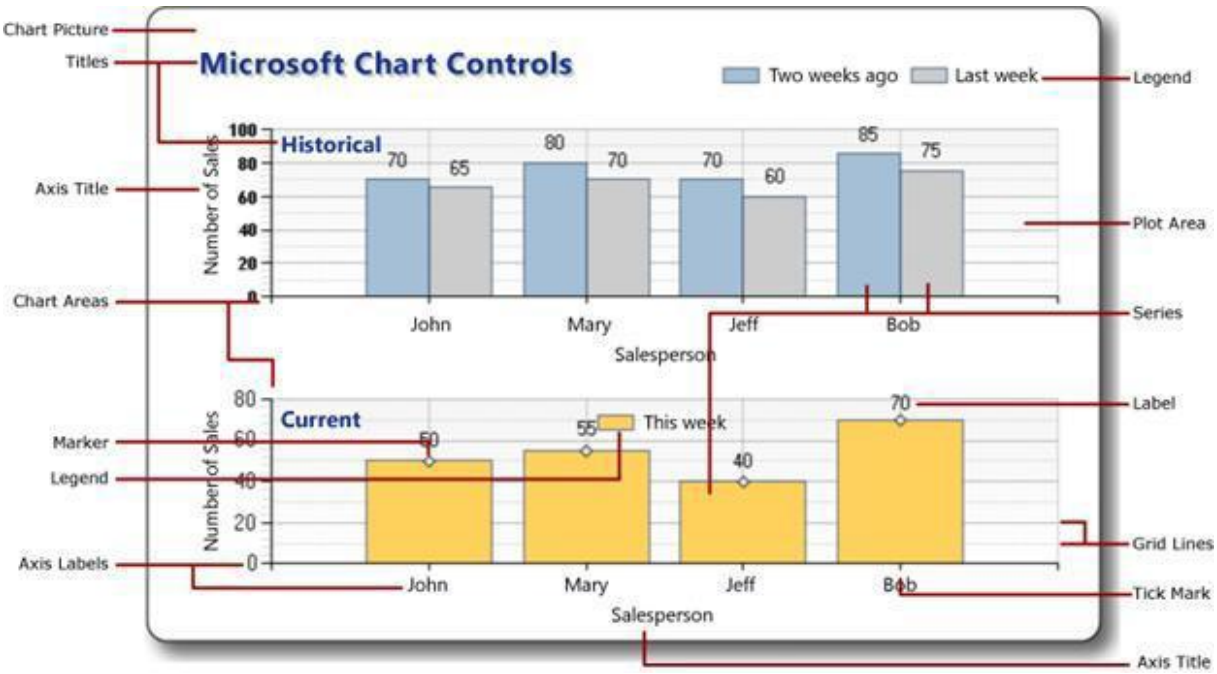


Diagram készítése az adatokból

Az adatok, melyekből a diagramot készítjük, származhatnak egy tömbből, egy lekérdezés eredményeiből, vagy egy XML fájl adataiból is.

Tömb használata

Ahogyan a 2.fejezetben (Bevezetés a WebMatrixba és az ASP.NET weboldalakba) olvashattuk, egy tömbel több hasonló elemet tárolhatunk egy egyszerű változóban. Ezeket a tömböket használhatjuk arra, hogy a diagramhoz szükséges adatokat tárolják.

Ez az eljárás megmutatja, hogyan csinálhatunk diagramot a tömbökben tárolt adatokból az alap diagramtípus használatával. Azt is megmutatja, hogyan jeleníthetjük meg a diagramot az oldalon.

1. Hozzunk létre egy új fájlt *ChartArrayBasic.cshtml* névvel!

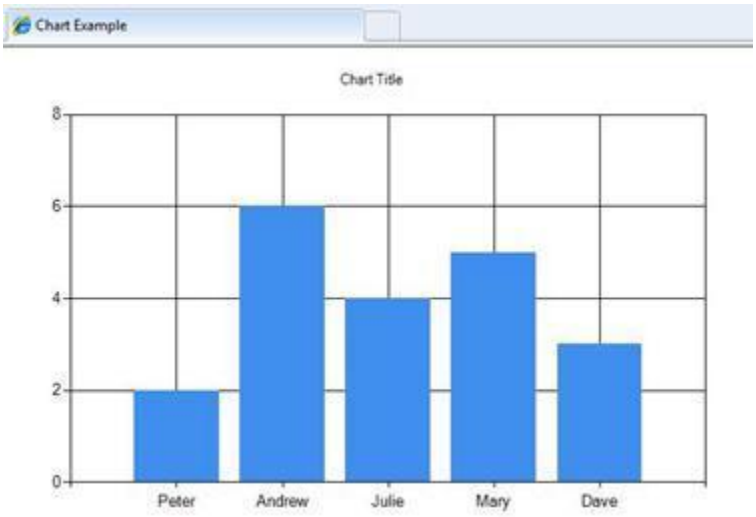
2. Cseréljük ki a meglévő kódot a következőre:

```
@{
    var myChart = new Chart(width: 600, height: 400)
        .AddTitle("Chart Title")
        .AddSeries(
            name: "Employee",
            xValue: new[] {  "Peter", "Andrew", "Julie", "Mary",
"Dave" },
            yValues: new[] { "2", "6", "4", "5", "3" })
        .Write();
}
```

A kód először létrehoz egy új diagramot, és beállítja annak szélességét és magasságát. Az `AddTitle` metódussal megadhatjuk a diagram címét. Adatok hozzáadásához az `AddSeries` metódust alkalmazzuk. Ebben a példában az `AddSeries` metódus `name`, `xValue`, és `yValues` paramétereit használjuk. A `name` paraméter a diagram jelmagyarázatában jelenik meg. Az `xValue` paraméter egy olyan sor adatot tartalmaz, amelyek a diagram vízszintes tengelyén láthatók. Az `yValues` pedig egy olyan sor adatot tartalmaz, melyek a diagram függőleges irányú pontjait adják meg.

A `Write` metódus hozza létre a diagramot. Ebben a példában, mivel nem adtuk meg a diagram típusát, ezért a `Chart` helper oszlopdiagramot készít alapbeállításként.

3. Futtassuk az oldalt a böngészőben! (Mielőtt futtatjuk, ellenőrizzük, hogy az oldal a *Files* munkaterületen ki van-e választva!) Ekkor a böngésző megjeleníti a diagramot.



Adatbázislekérdezés használata a diagram adataihoz

Ha a diagramhoz szükséges adatok egy adatbázisban vannak, futtathatunk egy adatbázis lekérdezést, és a lekérdezés eredményéből származó kiválasztott adatokat felhasználhatjuk. Ez az eljárás megmutatja, hogyan olvassuk le és hogyan jelenítsük meg az előző példában létrehozott adatokat.

1. Hozzunk létre egy *App_Data* mappát a weboldal gyökérkönyvtárában, ha még nincs ilyen mappa!
2. A mappába másoljuk be az 4. fejezetben létrehozott *SmallBakery.sdf* nevű adatbázisfájlt!
3. Hozzunk létre egy új fájlt *ChartDataQuery.cshtml* névvel!
4. Cseréljük ki a meglévő kódot a következőre:

```
@{
    var db = Database.Open("SmallBakery");
    var data = db.Query("SELECT Name, Price FROM Product");
    var myChart = new Chart(width: 600, height: 400)
        .AddTitle("Product Sales")
        .DataBindTable(dataSource: data, xField: "Name")
        .Write();
}
```

A kód először megnyitja a *SmallBakery* adatbázist, és hozzárendeli egy `db` változóhoz. Ez a váltózó egy olyan `Database` objektumot képvisel, ami az adatbázis írására és olvasására használható. Ez után a kód egy SQL lekérdezést futtat, hogy megtalálja a termékek nevét és árát. A kód létrehoz egy új diagramot, és a `DataBindTable` metódus használatával betölti a lekérdezés adatait a diagramba. Ehhez a metódushoz két paraméter szükséges. A `DataSource` paraméter a lekérdezésből származó adatokhoz van, az `xField` paraméterrel pedig beállíthatjuk, hogy melyik adatoszlop legyen a diagram x tengelyén.

A `DataBindTable` metódus alternatívájaként használhatjuk a `Chart` helper `AddSeries` metódusát is. Az `AddSeries` metódussal megadhatjuk az `xValue` és az `yValues` paramétereket. Például a `DataBindTable` metódus ilyen kódja helyett:

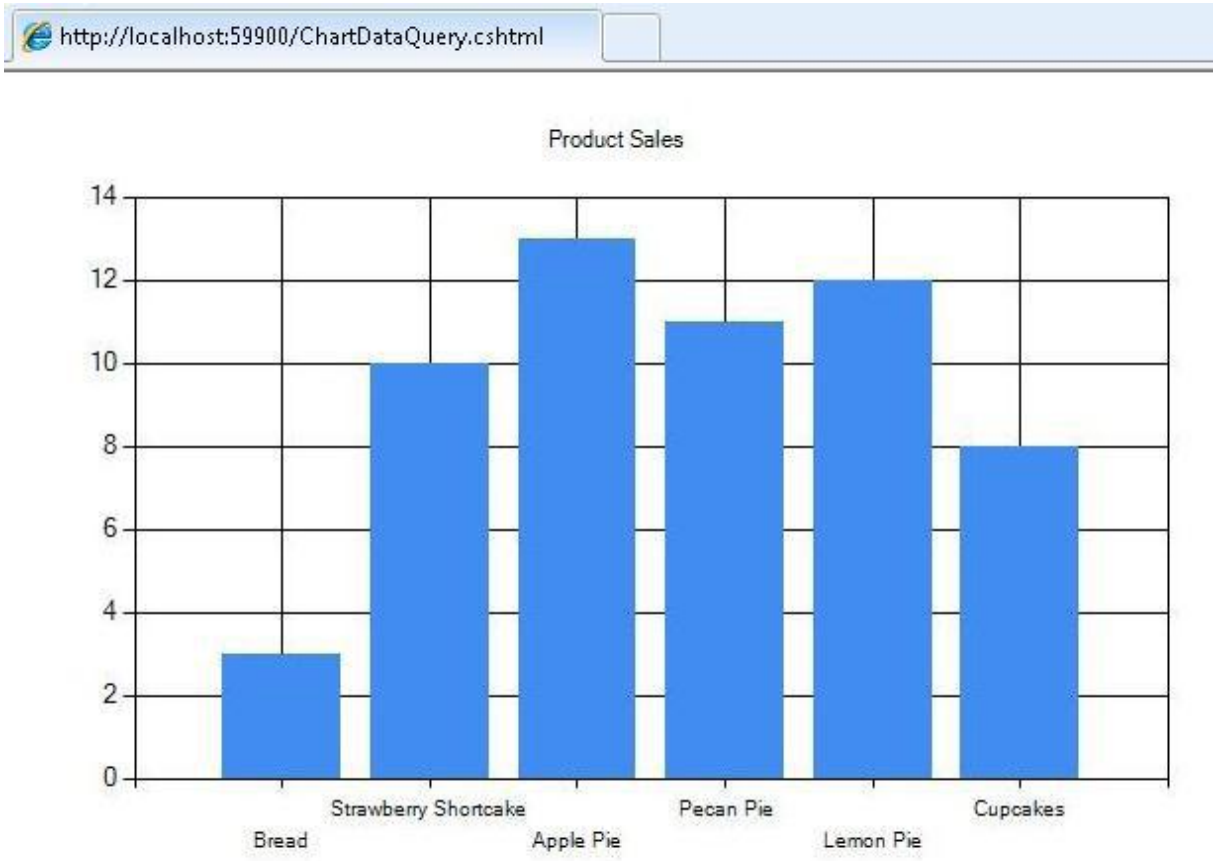
```
.DataBindTable(data,"Name)
```

Használhatjuk az `Addseries` metódust így:

```
.AddSeries("Default",
    xValue: data, xField: "Name",
    yValues: data, yFields: "Price")
```

Mindkét módszerrel ugyanazt az eredményt kapjuk. Az `Addseries` metódus rugalmasabb, mivel pontosabban megadhatjuk az adatokat és a diagramtípust, de a `DataBindTable` metódus egyszerűbb, amennyiben nincs szükségünk nagyobb rugalmasságra.

5. Futtassuk az oldalt a böngészőben!



XML adatok használata

A harmadik lehetőség diagramok létrehozására egy XML fájl használata a diagram adataihoz. Ehhez az szükséges, hogy az XML fájlnak legyen egy séma fájlja (*.xsd*) is, ami leírja az XML fájl struktúráját. Ez az eljárás megmutatja, hogyan nyerjünk ki adatokat egy XML fájlból.

1. Az *App_Data* mappában hozzunk létre egy új XML fájlt *data.xml* névvel!
2. Cseréljük ki a meglévő XML-t a következővel, ami egy elképzelt cég munkavállalóinak adatait tartalmazó XML adat:

```
<?xml version="1.0" standalone="yes" ?>
<NewDataSet xmlns="http://tempuri.org/data.xsd">
    <Employee>
        <Name>Erin</Name>
        <Sales>10440</Sales>
    </Employee>
    <Employee>
        <Name>Kim</Name>
        <Sales>17772</Sales>
    </Employee>
    <Employee>
        <Name>Dean</Name>
        <Sales>23880</Sales>
    </Employee>
    <Employee>
        <Name>David</Name>
        <Sales>7663</Sales>
    </Employee>
    <Employee>
        <Name>Sanjay</Name>
```

```
        <Sales>21773</Sales>
    </Employee>
    <Employee>
        <Name>Michelle</Name>
        <Sales>32294</Sales>
    </Employee>
</NewDataSet>
```

3. Az *App_Data* mappában hozzunk létre egy új XML fájlt *data.xsd* névvel! (Figyeljünk, hogy most a kiterjesztés *.xsd!*)
4. Cseréljük ki a meglévő XML-t a következőre:

```
<?xml version="1.0" ?>
<xs:schema
    id="NewDataSet"
    targetNamespace="http://tempuri.org/data.xsd"
    xmlns:mstns="http://tempuri.org/data.xsd"
    xmlns="http://tempuri.org/data.xsd"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
    attributeFormDefault="qualified"
    elementFormDefault="qualified">
    <xs:element name="NewDataSet"
        msdata:IsDataSet="true"
        msdata:EnforceConstraints="False">
        <xs:complexType>
            <xs:choice maxOccurs="unbounded">
                <xs:element name="Employee">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element
                                name="Name"
                                type="xs:string"
                                minOccurs="0" />
                            <xs:element
                                name="Sales"
                                    type="xs:double"
                                    minOccurs="0" />
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:choice>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

5. A weboldal gyökérkönyvtárában hozzunk létre egy új fájlt *ChartDataXML.cshtml* névvel!
6. Cseréljük ki a meglévő kódot a következőre:

```
@using System.Data;
@{
    var dataSet = new DataSet();
    dataSet.ReadXmlSchema(Server.MapPath("/App_Data/data.xsd"));
    dataSet.ReadXml(Server.MapPath("/App_Data/data.xml"));
    var dataView = new DataView(dataSet.Tables[0]);

    var myChart = new Chart(width: 600, height: 400)
        .AddTitle("Sales Per Employee")
        .AddSeries("Default", chartType: "Pie",
            xValue: dataView, xField: "Name",
            yValues: dataView, yFields: "Sales")
```

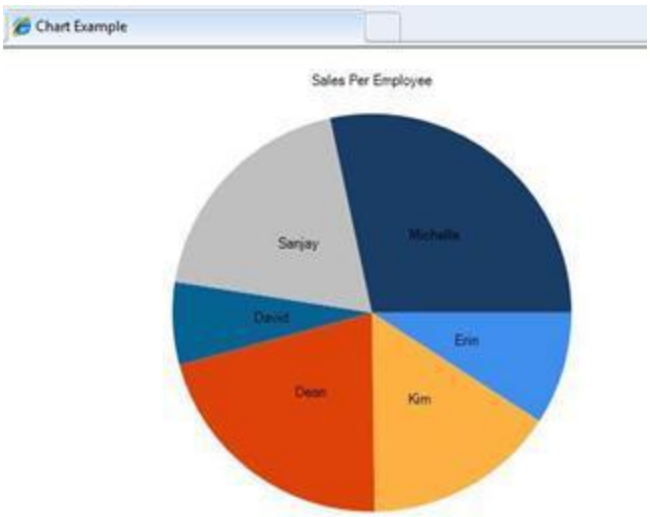
```
            .Write();
}
```

A kód először létrehoz egy `DataSet` objektumot. Ez az objektum kezeli az XML-ből kiolvasott adatokat, és rendszerezi őket a séma fájlban található információk szerint. (Vegyük észre, hogy a kód tetején látható a `using SystemData` állítás! Ez azért szükséges, hogy együtt tudjon működni a `DataSet` objektummal. További információk az Állítások és teljesen kvalifikált nevek használata című szövegdobozban találhatók.)

Ezután a kód létrehoz egy `DataView` objektumot az adatsor alapján. Az adat nézet ad egy objektumot, amihez a diagram kapcsolódni tud, kiolvassa és megjelenítse az adatokat. A diagram az `AddSeries` metódus használatával kapcsolódik az adatokhoz, ahogy korábban láttuk a soradatokból készített diagramoknál, kivéve, hogy itt az `xValue` és az `yValues` paramétereket állítjuk a `DataView` objektumhoz.

Ez a példa azt is megmutatja, hogy hogyan határozhatunk meg egy bizonyos diagramtípust. Amikor az adatokat az `AddSeries` metódussal adjuk meg, a `chartType` paraméter kördiagrammá változik.

7. Futtassuk az oldalt a böngészőben!



Állítások és teljesen kvalifikált nevek használata

Néha olyan komponenssel (osztállyal) kell dolgoznunk, mely a .NET Framework könyvtárban található, de ez nem érhető el automatikusan az ASP.NET Razor oldalakra. Ilyenkor két lehetőség van. Az egyik az úgynevezett *fully qualified names* használata a használni kívánt komponensre. A teljesen kvalifikált név egy komplett, félreérthetetlen név, ami nemcsak az osztály nevét tartalmazza, hanem azt az úgynevezett namespace-t is, amiben az osztály található. (A *namespace* egy gyűjtemény – egy könyvtárhoz hasonlót képzeljünk el –, ami a kapcsolódó osztályok gyűjteménye, és ez teszi praktikussá, hogy rendezze a több ezer osztályt a .NET Frameworkben.) A teljesen kvalifikált név megadásával bizonyosak lehetünk abban, hogy az ASP.NET megtalálja az osztályt, amivel dolgozunk, amikor az oldal épp fut.

Például a `DataSet` osztály a `System.Data` namespace-ben található. A `System.Data` namespace nem érhető el automatikusan az ASP.NET Razor oldalakról. Ezért a `Dataset` osztállyal való munkához a teljesen kvalifikált név használatával alkalmazzuk a következő kódot:

```
var dataSet = new System.Data.DataSet();
```

Ha a `DataSet` osztályt ismételten használjuk (vagy egyéb osztályt a `System.Data` namespace-ből), nehézkessé válhat mindig a teljesen kvalifikált nevet használni. Ezért alternatívaként *importálhatunk* is namespace-t. Ehhez az `using` állítást használjuk (`import` a Visual Basicben), ahogy a fejezet egy korábbi példájában láthattuk. Ha importáltuk a namespace-t, nem kell többé teljesen kvalifikált neveket használnunk az ebben a namespace-ben lévő osztályokhoz. A fejezet korábbi részében láthattunk egy példát:

```
@using System.Data;
@{
    var dataSet = new DataSet();
    // etc.
}
```

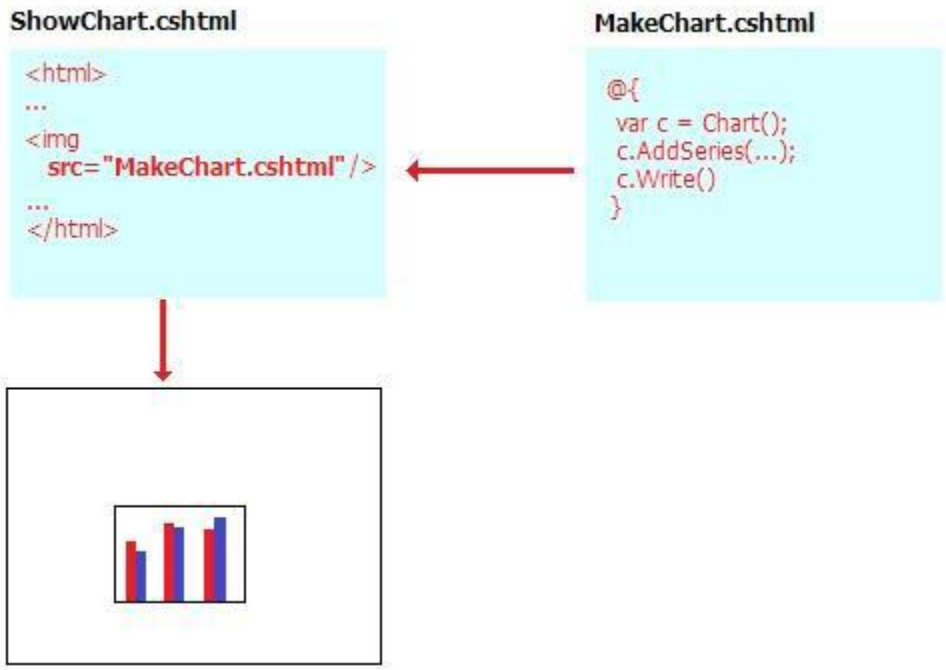
Vegyük észre, hogy mivel a `System.Data` namespace-t a `using` állítással importáltuk, a kód nem használ teljesen kvalifikált nevet a `DataSet`-hez – az ASP.NET tudja, melyik namespace-t kell keresni, hogy megtalálja a `DataSet` osztályt.

Hozzáadhatjuk a `using` állítást bármely egyéb .NET Framework namespace-hez amiket referenciaként szeretnénk. Azonban ezt nem kell túl gyakran megcsinálnunk, mivel a legtöbb osztály, amivel dolgozunk, olyan namespace-ekben van, amiket az ASP.NET automatikusan importál a *.cshtml* és a *.vbhtml* oldalak használatához.

Diagramok megjelenítése weboldalakon

Az eddigi példákban készítettünk egy diagramot, és utána ezt képként illesztettük a böngészőbe. Sok esetben azonban szeretnénk, ha a diagram az oldal része lenne, nemcsak ott lenne a böngészőben. Ehhez egy kétlépcsős folyamatot kell elvégezni. Az első lépés egy oldal létrehozása, ami generál egy diagramot, ahogy azt már láttuk korábban.

A második lépés, hogy megjelenítsük a létrejövő képet egy másik oldalon. A kép megjelenítéséhez egy HTML `<img>` elemet használunk, ugyanúgy, mintha bármilyen egyéb képet akarnánk megjeleníteni. Azonban egy *.jpg* vagy *.png* fájlra való hivatkozás helyett az `<img>` elem arra a *.cshtml* fájlra hivatkozik, amit az a `Chart` helper tartalmaz, ami létrehozza a diagramot. Amikor a megjelenítő lap fut, az *<img>* elem a `Chart` helper által készített diagramot jeleníti meg.



1. Készítsünk egy *ShowChart.cshtml* nevű fájlt!
2. Cseréljük ki a meglévő kódot a következőre:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Chart Example</title>
  </head>
  <body>
    <h1>Chart Example</h1>
    <p>The following chart is generated by the
<em>ChartArrayBasic.cshtml</em> file, but is shown
       in this page.</p>
    <p><img src="ChartArrayBasic.cshtml" /> </p>
  </body>
</html>
```

   A kód az *<img>* elem használatával megjeleníti azt a diagramot, amit korábban a *ChartArrayBasic.cshtml* fájlban hoztunk létre.

3. Futtassuk a weboldalt a böngészőben! A *ShowChart.cshtml* fájl megjeleníti azt a diagramot, ami a korábbi *ChartArrayBasic.cshtml* fájl kódján alapul.

Diagram formázása

A `Chart` helper számos olyan opcióval rendelkezik, melyekkel testreszabhatjuk a diagramok kinézetét. Beállíthatjuk a színeket, betűtípusokat, szegélyeket, stb. Egy egyszerű módszer a testreszabáshoz egy *theme* használata. A témák információk gyűjteményei, melyek meghatározzák, hogyan készüljön el a diagram, milyen betűtípussal, színekkel, feliratokkal, palettákkal, szegélyekkel és effektekkel. (Megjegyzendő azonban, hogy a diagram stílusa nem határozza meg a diagram típusát.)

A következő táblázatban láthatóak a beépített témák:

Téma	Leírás
`Vanilla`	Fehér háttéren piros oszlopokat jelenít meg.

`Blue`	Kék színátmenetes háttéren kék oszlopokat jelenít meg.
`Green`	Zöld színátmenetes háttéren kék oszlopokat jelenít meg.
`Yellow`	Sárga színátmenetes háttéren narancssárga oszlopokat jelenít meg.
`Vanilla3D`	Fehér háttéren piros 3D-s oszlopokat jelenít meg.

Új diagram készítésekor választhatunk a témák közül.

1. Hozzunk létre egy új fájlt *ChartStyleGreen.cshtml* néven!
2. Cseréljük ki a meglévő kódot a következőre:

```
@{
    var db = Database.Open("SmallBakery");
    var data = db.Query("SELECT Name, Price FROM Product");
    var myChart = new Chart(width: 600,
                        height: 400,
                        theme: ChartTheme.Green)
        .AddTitle("Product Sales")
        .DataBindTable(data, "Name")
        .Write();
}
```

Ez a kód megegyezik azzal a korábbi kóddal, ami az adatbázist használja adatként, de hozzáteszi a `theme` paramétert, amikor létrehozza a `Chart` objektumot. A következő példa mutatja a megváltoztatott kódot:

```
var myChart = new Chart(width: 600,
                    height: 400,
                    theme: ChartTheme.Green)
```

Futtassuk az oldalt a böngészőben! Ugyanazokat az adatokat láthatjuk, de a diagram jobban mutat.

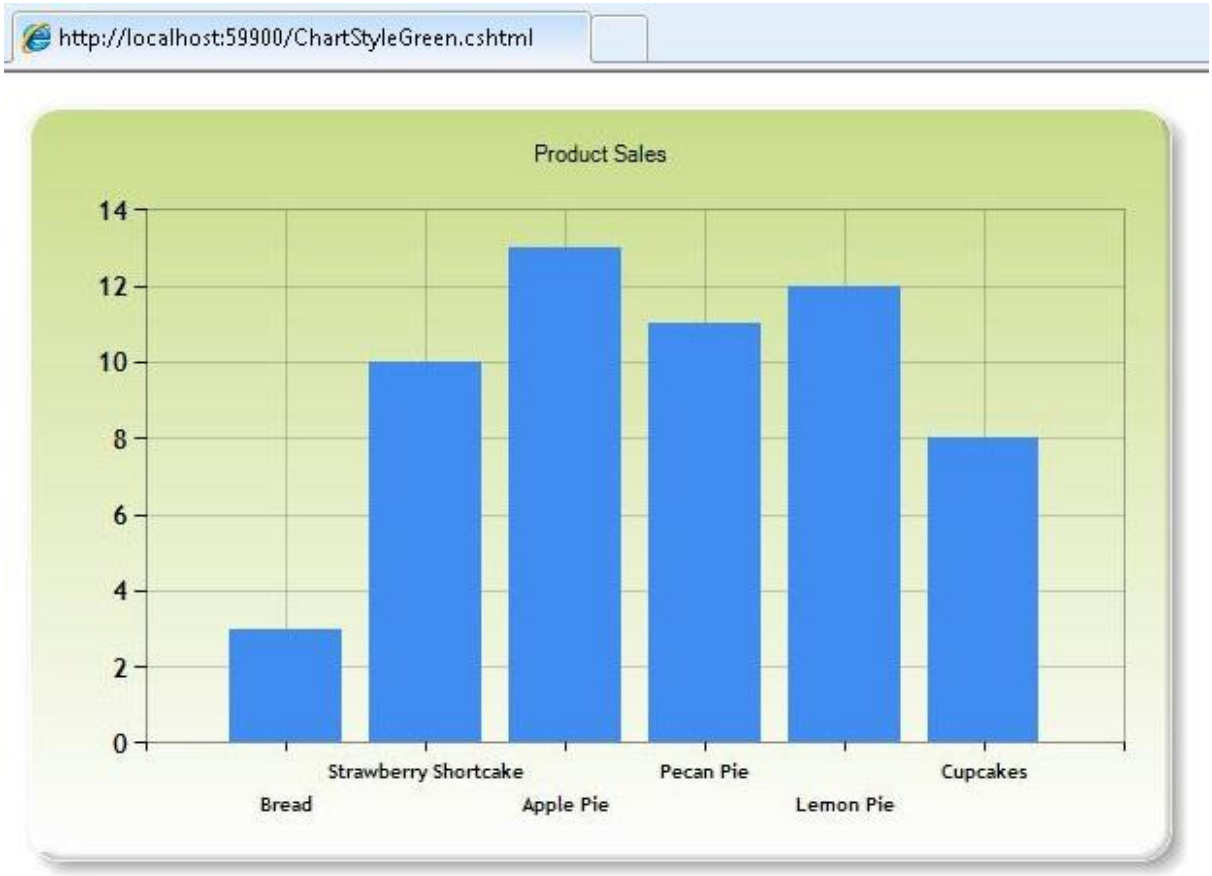


Diagram mentése

Ahogy a korábbiakban láttuk a fejezetben, amikor a `Chart` helpert használjuk, a helper minden alkalommal újra készíti a diagramot a vázlataiból. Ha szükséges, az adatok megszerzéséhez a diagram kódja újra lekérdezi az adatbázist, vagy újra kiolvassa az XML fájlt. Némely esetben ez elég bonyolult művelet, például ha a lekérdezett adatbázis túl nagy vagy az XML fájlban túl sok adat van. Még ha a diagram nem is tartalmaz sok adatot, a kép dinamikus elkészítése sok szerver erőforrást használ fel, és ha sokan kérdezik le a táblázatot tartalmazó oldalt vagy oldalakat, az befolyásolhatja a weboldalunk sebességét.

Ezért, hogy csökkentsük a diagram elkészítésének potenciális negatív hatásait az erőforrásainkra, először elkészíthetjük a diagramot, majd elmenthetjük azt. Ezután ha szükségünk van a diagramra, az újragenerálás helyett csak elővesszük az elmentett változatot, és azt jelenítjük meg.

A következő módon tudjuk elmenteni a diagramokat:

- Diagram elmentése a gyorsítótár bejegyzéseként. Ez a számítógép memóriájába (a szerveren) menti a diagramot, így gyorsan elérhető.
- Diagram elmentése képfájlként.
- Diagram elmentése XML fájlként. Ezzel az opcióval módosíthatjuk a diagramot mentés előtt.

Diagram mentése a gyorsítótárba

Miután készítettünk egy diagramot, berakhatjuk a gyorsítótárba. Ez azt jelenti, hogy a diagramot nem kell újra létrehozni, ha ismét meg szeretnénk jeleníteni. Amikor elmentjük a gyorsítótárba, a diagram kap egy egyedi kulcsot.

A gyorsítótárba mentett diagramok törlődhetnek, ha a szerver számára csak kevés memória érhető el. Továbbá a gyorsítótár törlődik, ha valamilyen okból újraindul az alkalmazás. Ezért a szokásos módszer a diagramok gyorsítótárban tárolásához az, hogy először mindig ellenőrizzük, megtalálható-e a gyorsítótárban, és ha nem, akkor létrehozzuk, vagy újra létrehozzuk.

1. A weboldalunk gyökérkönyvtárában hozzunk létre egy új fájlt *ShowCachedChart.cshtml* néven!
2. Cseréljük ki a meglévő kódot a következőre:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Chart Example</title>
    </head>
<body>
    <h1>Chart Example</h1>
    <img src="ChartSaveToCache.cshtml?key=myChartKey" />
</body>
</html>
```

   Az `<img>` jelző egy olyan `src` attribútumot tartalmaz, ami a *ChartSaveToCache.cshtml* fájlra mutat és egy kulcsot ad az oldalnak egy lekérdezés szálként. A kulcs értéke „myChartKey”. A *ChartSaveToCache.cshtml* fájl tartalmazza a `Chart` helpert, ami elkészíti a diagramot. Ezt az oldalt fogjuk a következőkben elkészíteni.

3. A weboldalunk gyökérkönyvtárában hozzunk létre egy új fájlt *ChartSaveToCache.cshtml* névvel!
4. Cseréljük ki a meglévő kódot a következőre:

```
@{
    var chartKey = Request["key"];
    if (chartKey != null) {
        var cachedChart = Chart.GetFromCache(key: chartKey);
        if (cachedChart == null) {
            cachedChart = new Chart(600, 400);
            cachedChart.AddTitle("Cached Chart -- Cached at " +
DateTime.Now);
            cachedChart.AddSeries(
               name: "Employee",
               axisLabel: "Name",
               xValue: new[] { "Peter", "Andrew", "Julie", "Mary",
"Dave" },
               yValues: new[] { "2", "6", "4", "5", "3" });
            cachedChart.SaveToCache(key: chartKey,
               minutesToCache: 2,
               slidingExpiration: false);
        }
        Chart.WriteFromCache(chartKey);
    }
}
```

   A kód először ellenőrzi, hogy a lekérdezés szálban valami megfelelt-e a kulcsértéknek. Ha igen, a kód megpróbálja kiolvasni a diagramot a gyorsítótárból a `GetFromCache` metódussal és a kulcs biztosításával. Ha nincs semmi a gyorsítótárban ilyen kulccsal (ami az első lekérdezés alkalmával mindig így lesz), akkor a kód elkészíti a diagramot a szokásos módon. Ha a diagram elkészült, a kód a `SaveToCache` hívásával elmenti a gyorsítótárba. Ehhez a metódushoz egy kulcs kell (hogy a diagramot később le lehessen kérdezni), és meg kell

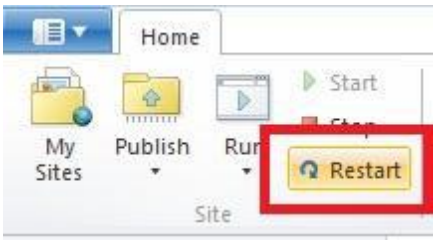
határozni, hogy mennyi ideig maradjon a gyorsítótárban. (A pontos időtartam attól függ, milyen gyakran változhatnak azok az adatok, melyeket ábrázol.) A `SaveToCache` metódus egy `SlidingExpiration` paramétert is tartalmaz– ha ez igaz, akkor a visszaszámláló minden hozzáférés alkalmával újraindul. Ebben az esetben ez azt jelenti, hogy a diagram gyorsítótár bejegyzése egy megnyitás után 2 perccel lejár. (Erre egy alternatíva az abszolút lejárat, amikor a diagram gyorsítótár bejegyzése a bekerülése után pontosan 2 perccel jár le, attól függetlenül, hogy közben hányszor lett megnyitva.)

Végül a kód a `WriteFromCache` metódust használja a diagram gyorsítótárból való elérésére és megjelenítésére. Fontos megjegyezni, hogy ez a metódus az `if` blokkon kívül van, ami a gyorsítótárat ellenőrzi, mert ez attól függetlenül lehívja a diagramot, hogy az benne van-e már a gyorsítótárban, vagy generálni kell és a gyorsítótárba menteni azt.

Vegyük észre, hogy a példában az `AddTitle` metódus egy időbélyegzőt tartalmaz. (Az aktuális dátumot és időt – `DateTime.Now` - adja a címhez.)

5. Futtassuk a *ShowCachedChart.cshtml* weboldalt a böngészőben! Az oldal megjeleníti a *ChartSaveToCache.cshtml* fájl kódján alapuló diagramot. Figyeljük meg, mit mutat az időbélyegző a diagram címében.

6. Zárjuk be a böngészőt!
7. Futtassuk újra a *ShowCachedChart.cshtml* fájlt! Vegyük észre, hogy az időbélyegző ugyanaz, mint előző alkalommal, ami azt jelzi, hogy a diagram nem lett újra generálva, hanem a gyorsítótárból került kiolvasásra!
8. A WebMatrixban a *Home* fül *Site* csoportjában kattintsunk a *Restart* gombra! Ez leállítja és újraindítja az ISS Expresst, ami újraindítja a weboldal alkalmazásunkat.



Másik megoldás, hogy várunk 2 percet a gyorsítótár kiürülésére.

9. Futtassuk a *ShowCachedChart.cshtml* fájlt ismét! Vegyük észre, hogy az időbélyegző megváltozott, mivel az alkalmazás újraindítása kiüríti a gyorsítótárat! Ezért a kód újragenerálta a diagramot és ismét berakta a gyorsítótárba.

Diagram mentése képfájlként

A diagramokat képfájlként is elmenthetjük (például *.jpg* fájlként) a szerverre. Ilyenkor úgy használhatjuk a diagramot, mint bármely egyéb képfájt. Az előnye az, hogy a fájl tárolva van az ideiglenes gyorsítótárba mentés helyett. Különböző időközönként (például óránként) elmenthetjük az új diagramképet, és így a változásokat folyamatosan nyomon követhetjük. Fontos, hogy a webalkalmazásnak legyen engedélyezve a fájlok mentése abban a szerveren lévő mappában, ahova a képeket szeretnénk menteni.

1. A weboldalunk gyökérkönyvtárában hozzunk létre egy új mappát *_ChartFiles* névvel, ha még nincsen ilyen mappa!
2. A weboldalunk gyökérkönyvtárában hozzunk létre egy új fájlt *ChartSave.cshtml* névvel!
3. Cseréljük ki a meglévő kódot a következőre:

```
@{
    var filePathName = "_ChartFiles/chart01.jpg";
    if (!File.Exists(Server.MapPath(filePathName))) {
        var chartImage = new Chart(600, 400);
        chartImage.AddTitle("Chart Title");
        chartImage.AddSeries(
                name: "Employee",
                axisLabel: "Name",
                xValue: new[] {  "Peter", "Andrew", "Julie", "Mary",
"Dave" },
                yValues: new[] { "2", "6", "4", "5", "3" });
        chartImage.Save(path: filePathName);
    }
}
<!DOCTYPE html>
<html>
    <head>
        <title>Chart Example</title>
    </head>
    <body>
        <img src="@filePathName" />
    </body>
</html>
```

   A kód először a `File.Exists` metódussal ellenőrzi, hogy a *.jpg* fájl létezik-e. Ha a fájl nem létezik, a kód elkészíti az adatsorból az új diagramot. Ekkor a kód előhívja a `Save` metódust, és hozzáadja a `path` paramétert, hogy meghatározza a fájl elérési útját és a fájl nevét, illetve hogy hova mentse a fájlt. Az oldal testében az *<img>* elem felhasználja az elérési utat a *.jpg* fájl megjelenítéséhez.

4. Futtassuk a *ChartSave.cshtml* fájlt!

Diagram mentése XML fájlként

Végül XML fájlként is elmenthetjük a diagramot a szerveren. Ennek az előnye a gyorsítótárba mentéssel és a fájlként elmentéssel szemben az, hogy ha akarjuk, a megjelenítés előtt módosíthatjuk az XML-t. Az alkalmazásunknak engedélyezve kell legyen az írás/olvasás a szerver azon könyvtárára, ahová a képet menteni szeretnénk.

1. A weboldalunk gyökérkönyvtárában hozzunk létre egy új fájlt *ChartSaveXML.cshtml* névvel!
2. Cseréljük ki a meglévő kódot a következőre:

```
@{
    Chart chartXml;
    var filePathName = "_ChartFiles/XmlChart.xml";
    if (File.Exists(Server.MapPath(filePathName))) {
        chartXml = new Chart(width: 600,
                             height: 400,
                             themePath: filePathName);
    }
    else {
        chartXml = new Chart(width: 600,
                             height: 400);
        chartXml.AddTitle("Chart Title -- Saved at " + DateTime.Now);
        chartXml.AddSeries(
            name: "Employee",
            axisLabel: "Name",
            xValue: new[] { "Peter", "Andrew", "Julie", "Mary",
"Dave" },
            yValues: new[] { "2", "6", "4", "5", "3" });
        chartXml.SaveXml(path: filePathName);
    }
    chartXml.Write();
}
```

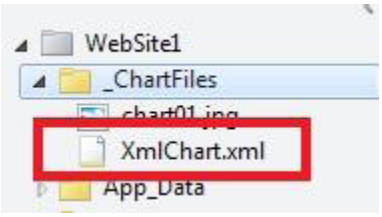
   Ez a kód hasonló ahhoz, amit korábban a diagram gyorsítótárba mentésénél láttunk, kivéve azt, hogy ez egy XML fájlt használ. A kód először a `File.Exists` metódussal leellenőrzi, hogy létezik-e az XML fájl. Ha a fájl létezik, a kód létrehoz egy új diagram objektumot, és a `themePath` paraméterként megadja a fájlnevet. Ez az XML fájl tartalma alapján létrehozza a diagramot. Ha az XML fájl még nem létezik, akkor a kód egyszerűen létrehoz egy diagramot, és a `SaveXml`-lel elmenti azt. A diagramot a `Write` metódussal hozza létre, mint ahogy azt korábban láthattuk

   Ahogy a gyorsítótáras oldalnál, ez a kód is tartalmaz egy időbélyegzőt.

3. Hozzunk létre egy új oldalt *ChartDisplayXMLChart.cshtml* névvel, és adjuk hozzá a következőt:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Display chart from XML</title>
  </head>
  <body>
    <img src="ChartSaveXML.cshtml" />
  </body>
</html>
```

4. Futtassuk a *ChartDisplayXMLChart.cshtml* oldalt! A diagram megjelenik. Figyeljük meg a diagram címében az időbélyegzőt!
5. Zárjuk be a böngészőt!
6. A WebMatrixban kattintsunk a *_ChartFiles* mappára jobb gombbal, majd kattintsunk a *Refresh* gombra, és nyissuk meg a mappát! A mappában lévő *XMLChart.xml* fájlt a `Chart` helper hozta létre.



7. Futtassuk a *ChartDisplayXMLChart.cshtml* fájlt ismét! A diagram ugyanazt az időbélyegzőt jeleníti meg, amit az előző megnyitásnál is. Ez azért van, mert a diagramot a korábban elmentett XML-ből generálja.
8. A WebMatrixban nyissuk meg a *_ChartFiles* mappát, és töröljük a *XMLChart.xml* fájlt!
9. Futtassuk megint a *ChartDisplayXMLChart.cshtml* fájlt! Ezúttal az időbélyegzőt frissítette, mert a `Chart` helpernek újra létre kellett hoznia az XML fájlt. Ha akarjuk, megnézhetjük a *_ChartFiles* mappát, amiben ismét ott lesz az XML fájl.

További forrás angolul

- Diagramkezelés

9. fejezet – Munka fájlokkal

Az előző fejezetekben megtanultuk, hogyan tároljunk adatokat adatbázisban. Azonban akár szövegfájlokkal is dolgozhatunk a weboldalunkban. Például a weboldal adatainak egyszerű tárolásához használhatunk szövegfájlokat is. (Az adatok tárolására használt szövegfájlt néha *flat file*-nak hívjuk.) A szövegfájlok többféle kiterjesztésűek lehetnek, mint például *.txt, .xml,* vagy *.csv* (vesszővel elválasztott értékek).

Az útmutatóból megtudhatjuk:

- Hogyan készítsünk szövegfájlt, és hogyan írjunk bele adatokat?
- Hogyan adjunk adatokat egy meglévő fájlhoz?
- Hogyan olvassunk egy fájlt, és hogyan jelenítsük meg?
- Hogyan töröljünk fájlokat egy weboldalról?
- Hogyan tölthetnek fel a felhasználók egy vagy több fájlt?

Az alábbi ASP.NET programozási funkciókat ismerhetjük meg a fejezetben:

- A `File` objektumot, amivel fájlokat kezelhetünk.
- A `FileUpload` helpert.
- A `Path` objektumot, amivel elérési utakat és fájlneveket módosíthatunk.

Megjegyzés: ha képeket akarunk feltölteni és szerkeszteni (például forgatni és átméretezni), akkor nézzük meg a 10. fejezet – Munka képekkel című részt!

Szövegfájl létrehozása és adatok beleírása

Amikor adatot szeretnénk tárolni szövegfájlban, használhatjuk a `File.WriteAllText` metódust a létrehozandó fájl meghatározásához, és az adatok beírásához. Ezzel az eljárással létrehozunk egy oldalt, ami egy egyszerű űrlapot tartalmaz 3 `input` mezővel (vezetéknév, keresztnév, e-mail cím) és egy *Submit* gombbal. Amikor a felhasználó elküldi az űrlapot, eltároljuk az általa bevitt adatokat egy szövegfájlban.

1. Ha még nincs, hozzunk létre egy új mappát *App_Data* néven.
2. A weboldalunk gyökérkönyvtárában hozzunk létre egy új fájlt *UserData.cshtml* néven.
3. Cseréljük ki a meglévő kódot a következőre:

```
@{
    var result = "";
    if (IsPost)
    {
        var firstName = Request["FirstName"];
        var lastName = Request["LastName"];
        var email = Request["Email"];

        var userData = firstName + "," + lastName +
            "," + email + Environment.NewLine;

        var dataFile = Server.MapPath("/App_Data/data.txt");
        File.WriteAllText(@dataFile, userData);
        result = "Information saved.";
    }
}
```

```
<!DOCTYPE html>
<html>
<head>
    <title>Write Data to a File</title>
</head>
<body>
    <form id="form1" method="post">
    <div>
        <table>
            <tr>
                <td>First Name:</td>
                <td><input id="FirstName" name="FirstName"
type="text" /></td>

            </tr>
            <tr>
                <td>Last Name:</td>
                <td><input id="LastName" name="LastName" type="text"
/></td>
            </tr>
            <tr>
                <td>Email:</td>
                <td><input id="Email" name="Email" type="text"
/></td>
            </tr>
            <tr>
                <td></td>
                <td><input type="submit" value="Submit"/></td>
            </tr>
        </table>
    </div>
    <div>
    @if(result != ""){
        <p>Result: @result</p>
    }
    </div>
    </form>
</body>
</html>
```

A HTML markup elkészíti az űrlapot 3 szövegdobozzal. A kódban az `IsPost` tulajdonságot használjuk annak meghatározásához, hogy az oldalt - mielőtt elkezdjük feldolgozni - már elküldték-e. Az első feladat az, hogy megszerezzük a felhasználó bevitt adatait, és hozzárendeljük változókhoz. A kód ekkor a különböző változók értékeit egy vesszőkkel elválasztott szálba összefűzi, amit egy másik változó alatt eltárol. Vegyük észre, hogy a vessző elválasztó egy idézőjelek közötti szál, mert a vesszőt szó szerint beillesztjük egy nagyobb szálba, amit készítünk. Az összekapcsolt adatok végén hozzáadjuk az `Environment.NewLine`-t. Ez egy sortörést ad hozzá. Így néz ki, amit ezzel az összekapcsolással létrehozunk:

```
David,Jones,davidj@contoso.com
```

(A végén egy láthatatlan sortöréssel)

Ezután létrehozunk egy változót (`dataFile`), ami az adat tárolásához szükséges fájl nevét és helyét tartalmazza. A hely beállítása speciális módszert igényel. A webhelyek esetén rossz gyakorlat a kódban abszolút elérési útra hivatkozni, mint például *C:\Folder\File.txt*. Ha a weboldal elköltözik, az abszolút elérési út rossz lesz. Ráadásul egy hostolt oldalnál (a saját számítógépünkkel szemben) általában nem is tudjuk az elérési utat, amikor írjuk a kódot.

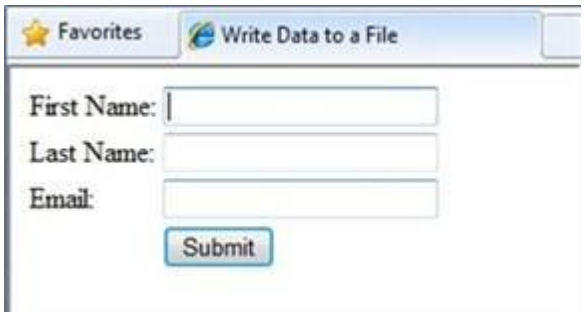
De néha (mint most, fájl írásánál) komplett elérési útra van szükség. A megoldás a `Server` objektum `MapPath` metódusának használata. Ez megmutatja a weboldal komplett elérési útját. Hogy megtaláljuk az utat a weboldal gyökérkönyvtárába, a „~” jelet adjuk a `MapPath`-hoz. (Alkönyvtár nevet is hozzáadhatunk, például *~/App_Data/,* hogy megkapjuk az alkönyvtár elérési útját.) Ekkor összekapcsolhatunk egyéb információt azzal, amit a metódus létrehoz, hogy egy komplett elérési utat készítsen. Ebben a példában egy fájlnevet adunk hozzá. (A 3. fejezetben többet is olvashatunk a fájlok és mappák elérési útjairól.)

A fájl az *App_Data* mappában kerül mentésre. Ez egy speciális mappa az ASP.NET-ben, amit adatfájlok tárolásához használunk, ahogy a 6. fejezetben láthattuk.

A `File` objektum `WriteAllText` metódusa írja az adatot a fájlba. Ehhez két paramétert használ, a fájl nevét (és elérési útját), amibe ír, és az aktuális adatokat, amiket beleír. Vegyük észre, hogy az első paraméter neve előtt egy @ karakter előtag található. Ez jelzi az ASP.NET számára, hogy szó szerinti szálat adunk meg, és például a „/” karaktereket ne értelmezze speciálisan. (További információért nézzük meg a 3. fejezetet!)

Megjegyzés: Ahhoz, hogy a kód az *App_Data* mappába mentse a fájlokat, az alkalmazásnak írás-olvasási felhatalmazásra van szüksége erre a mappára. A fejlesztő számítógépen általában ez nem okoz problémát. Azonban amikor egy hostolt webszerveren publikáljuk a weboldalunkat, kifejezetten szükség lehet az engedélyek beállítására. Ha ezt a kódot futtatjuk a webhosting szolgáltató szerverén, és hibaüzenetet kapunk, vegyük fel a kapcsolatot a szolgáltatóval, hogy hogyan állíthatók be ezek az engedélyek!

4. Futtassuk az oldalt a böngészőben! (Mielőtt futtatjuk, győződjünk meg róla, hogy a *Files* munkaterület alatt ki van választva az oldal!)



5. Írjunk be a mezőkbe értékeket, majd kattintsunk a *Submit* gombra!
6. Zárjuk be a böngészőt!
7. Térjünk vissza a projekthez, és frissítsük a nézetet!
8. Nyissuk meg a *Data.txt* fájlt! Az adatok, amiket az űrlapon beírtunk, a fájlban vannak.

9. Zárjuk be a *Data.txt* fájlt!

Adat hozzáadása meglévő fájlhoz

Az előző példában a `WriteAllText`-et használtuk olyan szövegfájlok létrehozására, amik egyetlen adatot tartalmaznak. Ha újra lehívjuk a metódust, és ugyanazt a nevet rendeljük hozzá, akkor a meglévő fájlt felülírja. Azonban egy fájl létrehozása után gyakran szeretnénk hozzáadni további adatokat is. Ezt megtehetjük a `File` objektum `AppendAllText` metódusának használatával.

1. A weboldalban készítsünk egy másolatot a *UserData.cshtml* fájlról, és a másolatot nevezzük át *UserDataMultiple.cshtml*-re!
2. A nyitó `<!DOCTYPE html>` tag előtti kódblokkot cseréljük ki a következőre:

```
@{
    var result = "";
    if (IsPost)
    {
        var firstName = Request["FirstName"];
        var lastName = Request["LastName"];
        var email = Request["Email"];

        var userData = firstName + "," + lastName +
            "," + email + Environment.NewLine;

        var dataFile = Server.MapPath("/App_Data/data.txt");
        File.AppendAllText (@dataFile, userData);
        result = "Information saved.";
    }
}
```

   Ez a kód egy dologban különbözik az előző kódtól. A `WriteAllText` metódus helyett az `AppendAllText` metódust használja. A két metódus hasonló, annyi különbséggel, hogy az `AppendAllText` metódus a fájl végéhez adja hozzá az adatokat. Csakúgy mint a `WriteAllText`, az `AppendAllText` is létrehozza a fájlt, ha az még nem létezik.

3. Futtassuk az oldalt a böngészőben!
4. Írjunk be értékeket a mezőkbe, és kattintsunk a *Submit* gombra!
5. Írjunk be további adatokat, és ismét küldjük el!
6. Térjünk vissza a projekthez, kattintsunk jobb gombbal a projektmappára, majd a *Refresh* gombra!
7. Nyissuk meg a *Data.txt* fájlt! Az imént beírt adatokat is tartalmazza.

Adatok olvasása és megjelenítése fájlból

Még ha nem is akarunk adatokat szövegfájlba írni, valószínűleg néha ki kell olvasnunk adatokat belőle. Ehhez újra a `File` objektumot használjuk. A `File` objektum használatával a sorokat egyenként is kiolvashatjuk (sortörésekkel elválasztva), vagy egy külön elemet is kiolvashatunk függetlenül az elválasztás módjától.

Ez az eljárás megmutatja, hogyan olvassuk ki és jelenítsük meg az adatokat, amiket az előző példában hoztunk létre.

1. A weboldalunk gyökérkönyvtárában hozzunk létre egy új fájlt *DisplayData.cshtml* névvel!
2. Cseréljük ki a meglévő kódot a következőre:

```
@{
    var result = "";
    Array userData = null;
    char[] delimiterChar = {','};

    var dataFile = Server.MapPath("/App_Data/data.txt");

    if (File.Exists(dataFile)) {
        userData = File.ReadAllLines(dataFile);
        if (userData == null) {
            // Empty file.
            result = "The file is empty.";
        }
    }
    else {
        // File does not exist.
        result = "The file does not exist.";
    }
}
<!DOCTYPE html>

<html>
<head>
    <title>Reading Data from a File</title>
</head>
<body>
    <div>
        <h1>Reading Data from a File</h1>
        @result
        @if (result == "") {
            <ol>
            @foreach (string dataLine in userData) {
            <li>
                User
```

```
                <ul>
                @foreach (string dataItem in
dataLine.Split(delimiterChar)) {
                    <li>@dataItem</li >
                }
                </ul>
            </li>
            }
            </ol>
        }
    </div>
</body>
</html>
```

A kód az előző példában készített fájl olvasásával kezd egy `UserData` változóra, a következő metódus hívásával:

```
File.ReadAllLines(dataFile)
```

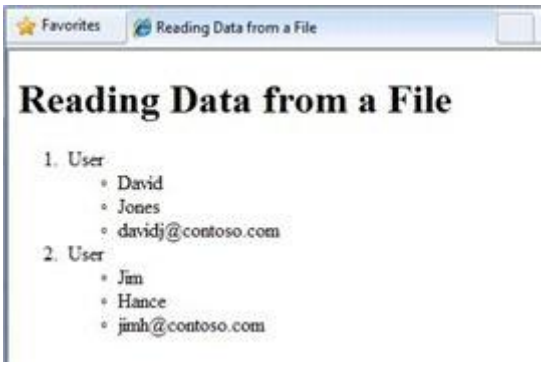
A kód ehhez egy `if` állításban van. Ha egy fájlt szeretnénk olvasni, célszerű először a `File.Exists` metódust használni, hogy meghatározzuk, elérhető-e a fájl. A kód ellenőrzi azt is, hogy üres-e a fájl.

A kód szövegtörzse két `foreach` hurkot tartalmaz, az egyiket beágyazva a másikba. A külső `foreach` hurok egyszerre egy sort vesz ki az adatfájlból. Ebben az esetben a sorokat sortörések definiálják, azaz minden adat egy külön sor a fájlban. A külső hurok egy új elemet hoz létre (`<li>` elem) egy rendezett listában (`<ol>` elem).

A belső hurok az adatsorokat elemekre (mezőkre) osztja a vesszőt használva elválasztóként. (Az előző példa alapján ez azt jelenti, hogy minden sor 3 mezőt tartalmaz – vezetéknév, keresztnév, e-mail cím, mindegyik vesszőkkel elválasztva.) A belső hurok létrehoz egy `<ul>` listát is, és az adatsorban minden mezőre egy listaelemet jelenít meg.

A kód a két adattípus használatát illusztrálja, a sor és a `char` adattípust. A sor azért szükséges, mert a `File.ReadAllLines` metódus sorként adja az adatokat. A `char` adattípus pedig azért kell, mert a `Split` metódus egy `array`-t ad, amiben minden elem `char` típusú. (A sorokról való további információkért nézzük meg a 3. fejezetet!)

3. Futtassuk az oldalt a böngészőben! Az adatok, amiket az előző példában bevittünk, megjelennek.



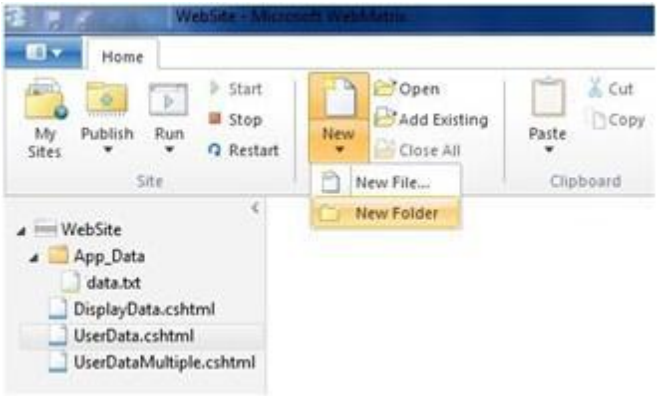
Adatok megjelenítése Microsoft Excel vesszővel elválasztott fájlból

A táblázatban tárolt adatok vesszővel elválaszott fájlként történő mentéséhez használhatjuk a Microsoft Excelt (*.csv* fájl). Ha ezt tesszük, az adatokat sima szövegként menti el, nem Excel formátumban. A táblázat minden sora sortöréssel lesz elválasztva a szövegfájlban, és minden adat elemet vessző választ el egymástól. Az előző példában bemutatott kódot használhatjuk Excel vesszővel elválasztott fájl olvasásához, csak az adatfájl nevét kell megváltoztatni a kódban.

Fájlok törlése

A weboldalról fájlok törléséhez a `File.Delete` metódust használjuk. Ez az eljárás megmutatja, hogyan törölhetnek a felhasználók egy képet (*.jpg* fájlt) egy *images* mappából, ha tudják a fájl nevét.

1. A weboldalon hozzunk létre egy almappát *images* névvel!



2. Másoljunk egy vagy több *.jpg* fájlt az *images* mappába!
3. A weboldal gyökérkönyvtárában hozzunk létre egy fájlt *FileDelete.cshtml* névvel!
4. Cseréljük ki a meglévő kódot a következőre:

```
@{
    bool deleteSuccess = false;
    var photoName = "";
    if (IsPost) {
        photoName = Request["photoFileName"] + ".jpg";
        var fullPath = Server.MapPath("/images/" + photoName);

        if (File.Exists(fullPath))
        {
                File.Delete(fullPath);
                deleteSuccess = true;
        }
    }
}
```

```
<!DOCTYPE html>
<html>
  <head>
    <title>Delete a Photo</title>
  </head>
  <body>
    <h1>Delete a Photo from the Site</h1>
    <form name="deletePhoto" action="" method="post">
      <p>File name of image to delete (without .jpg extension):
      <input name="photoFileName" type="text" value="" />
      </p>
      <p><input type="submit" value="Submit" </p>
    </form>

    @if(deleteSuccess) {
        <p>
        @photoName deleted!
        </p>
        }
  </body>
</html>
```

Ez az oldal egy űrlapot tartalmaz, ahol a felhasználók megadhatják egy képfájl nevét. Ha nem írják be a *.jpg* fájlnév kiterjesztést, a fájlnév ezen korlátozásával megakadályozhatjuk, hogy a felhasználók tetszőleges fájlokat törölhessenek az oldalról.

A kód elolvassa a beírt fájlnevet, majd egy komplett elérési utat hoz létre. Ehhez a kód a weboldal aktuális útját (ahogy a `Server.MapPath` metódus megadja), az *images* mappát, a felhasználó által megadott nevet és a „.jpg” szó szerinti szálat használja.

A fájl törléséhez a kód a `File.Delete` metódust hívja, és átadja a teljes elérési utat, amit az előbb készített. A kulcs végén a kód egy visszaigazoló üzenetet jelenít meg a fájl törléséről.

5. Futtassuk az oldalt a böngészőben!

6. Írjuk be a törölni kívánt fájl nevét, majd kattintsunk a *Submit* gombra! Ha a fájlt törölte, a fájlnév megjelenik az oldal alján.

Fájlok feltöltése a felhasználók által

A `FileUpload` helper segítségével a felhasználók fájlokat tölthetnek fel a weboldalra. A következő folyamat megmutatja, hogyan tölthetnek fel egy fájlt a felhasználók.

1. Adjuk hozzá az ASP.NET Web Helpers Libraryt a webhelyünkhöz a 2. fejezetben leírtak szerint, ha még nem tettük meg korábban!
2. Az *App_Data* mappában hozzunk létre egy új mappát *UploadedFiles* névvel!
3. A gyökérkönyvtárban hozzunk létre egy új fájlt *FileUpload.cshtml* névvel!
4. Cseréljük ki a meglévő kódot a következőre:

```
@{
    var fileName = "";
    if (IsPost) {
        var fileSavePath = "";
        var uploadedFile = Request.Files[0];
        fileName = Path.GetFileName(uploadedFile.FileName);
        fileSavePath = Server.MapPath("/App_Data/UploadedFiles/" +
          fileName);
        uploadedFile.SaveAs(fileSavePath);
    }
}
<!DOCTYPE html>
<html>
    <head>
    <title>FileUpload - Single-File Example</title>
    </head>
    <body>
    <h1>FileUpload - Single-File Example</h1>
    @FileUpload.GetHtml(
        initialNumberOfFiles:1,
        allowMoreFilesToBeAdded:false,
        includeFormTag:true,
        uploadText:"Upload")
    @if (IsPost) {
        <span>File uploaded!</span><br/>
    }
    </body>
</html>
```

Az oldal szövegtörzse a `FileUpload` helpert használja a feltöltő doboz és a már valószínűleg ismerős gombok létrehozásához.

A `FileUpload` helpernek megadott beállítások meghatározzák, hogy egy egyszerű dobozt akarunk a feltöltéshez, és a küldés gomb *Upload* feliratú legyen. (A későbbiekben több dobozt is hozzáadunk.)

Amikor a felhasználó az *Upload* gombra kattint, az oldal tetején lévő kód megszerzi a fájlt és elmenti. A `Request` objektum, amit általában arra használunk, hogy űrlapokból szerezzen értékeket, tartalmaz egy `Files` sort is, ami a feltöltött fájlt (vagy fájlokat) tartalmazza. Különböző fájlokat szedhetünk ki a sor meghatározott pozícióiból, például az első feltöltött fájl megszerzéséhez a `Request.Files[0]` szükséges, a másodikhoz a `Request.Files[1]` és így tovább. (Emlékezzünk, hogy a programozásban a számozás általában 0-val kezdődik!)

Amikor megszereztünk egy feltöltött fájlt, egy változóba rakjuk (itt `uploadedFile`), hogy így módosíthassuk. A feltöltött fájl nevének meghatározásához csak a `FileName` tulajdonsága szükséges. Azonban amikor a felhasználó feltölt egy fájlt, a `FileName` az eredeti nevét tartalmazza, amiben az egész elérési út megtalálható. Például így nézhet ki:

C:\Users\Public\Sample.txt

Nincs szükségünk erre az összes információra, mivel ez az elérési út a felhasználó számítógépén van, nem a szerveren. Csak az aktuális fájlnévre van szükségünk (*Sample.txt*). Kivehetjük csak a fájlt a `Path.GetFileName` metódus használatával, így:

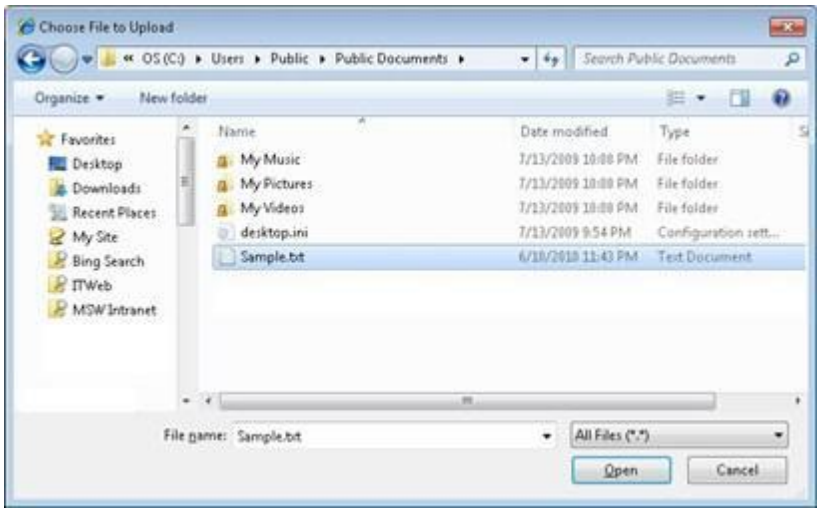
```
Path.GetFileName(uploadedFile.FileName)
```

A `Path` elem egy számos ilyen metódussal rendelkező eszköz, amikkel elérési utakat vehetünk ki, kombinálhatunk, stb.

Ha megszereztük a feltöltött fájl nevét, készíthetünk egy új elérési utat, ahol tárolni kívánjuk a fájlt a weboldalon. Ebben az esetben kombináljuk a `Server.MapPath`-t, a mappaneveket (*App_Data/UploadedFiles*) és az újonnan szerzett fájlnevet az új elérési út elkészítéséhez. Ezután a `SaveAs` metódust hívjuk a fájl mentéséhez.

5. Futtassuk az oldalt a böngészőben!

6. Kattintsunk a *Browse* gombra, és válasszunk ki egy feltölteni kívánt fájlt!



A *Browse* gomb melletti szövegdoboz tartalmazza a fájl elérési útját és helyét.

7. Kattintsunk az *Upload* gombra!

8. A weboldalon jobb gombbal kattintsunk a project mappára, és kattintsunk a *Refresh* gombra!
9. Nyissuk meg az *UploadedFiles* mappát! A fájl, amit feltöltöttünk, a mappában van.

Több fájl feltöltése a felhasználók által

Az előző példában a felhasználóknak egy fájl feltöltését engedtük. De a `FileUpload` helpert egyszerre több fájl feltöltésére is használhatjuk. Ez jól jön például fényképek feltöltésénél, ahol a fájlok egyenkénti feltöltése hosszadalmas. (Fényképek feltöltéséről bővebben a 10. fejezetben olvshatunk.) Ez a példa azt mutatja be, hogyan tölthetnek fel a felhasználók egyszerre két fájlt, azonban ugyanezzel a módszerrel több fájl is feltölthető.

1. Adjuk hozzá az ASP.NET Web Helpers Libraryt a weboldalunkhoz a 2. fejezetben leírtak szerint, ha még nem tettük meg korábban!
2. Hozzunk létre egy új fájlt *FileUploadMultiple.cshtml* névvel!
3. Cseréljük ki a meglévő kódot a következőre:

```
@{
  var message = "";
  if (IsPost) {
      var fileName = "";
      var fileSavePath = "";
      int numFiles = Request.Files.Count;
      int uploadedCount = 0;
      for(int i =0; i < numFiles; i++) {
          var uploadedFile = Request.Files[i];
          if (uploadedFile.ContentLength > 0) {
              fileName = Path.GetFileName(uploadedFile.FileName);
              fileSavePath =
Server.MapPath("/App_Data/UploadedFiles/" +
                fileName);
              uploadedFile.SaveAs(fileSavePath);
              uploadedCount++;
          }
       }
       message = "File upload complete. Total files uploaded: " +
         uploadedCount.ToString();
   }
}
<!DOCTYPE html>
<html>
    <head><title>FileUpload - Multiple File Example</title></head>
<body>
    <form id="myForm" method="post"
       enctype="multipart/form-data"
```

```
       action="">
    <div>
    <h1>File Upload - Multiple-File Example</h1>
    @if (!IsPost) {
        @FileUpload.GetHtml(
            initialNumberOfFiles:2,
            allowMoreFilesToBeAdded:true,
            includeFormTag:true,
            addText:"Add another file",
            uploadText:"Upload")
        }
    <span>@message</span>
    </div>
    </form>
</body>
</html>
```

Ebben a példában a `FileUpload` helpert az oldal szövegtörzsében arra konfiguráltuk, hogy a felhasználók két fájlt tölthessenek fel egyszerre alapértelmezésként. Mivel az `AllowMoreFilesToBeAdded` `true`-ra van állítva, a helper készít egy linket, amivel a felhasználók több feltöltési dobozt adhatnak hozzá.

A felhasználók által feltöltött fájlok feldolgozásához a kód ugyanazt az alapmódszert használja, mint a korábbi példában – a fájlok megszerzése a `Request.Files`-ból, majd mentése. (A különböző dolgokkal, például a fájlnév megszerzésével és az elérési úttal együtt.) Az újítás ebben az esetben az, hogy a felhasználó több fájlt is feltölthet, de nem tudjuk, hányat. Ezt megtudhatjuk a `Request.Files.Count` segítségével.

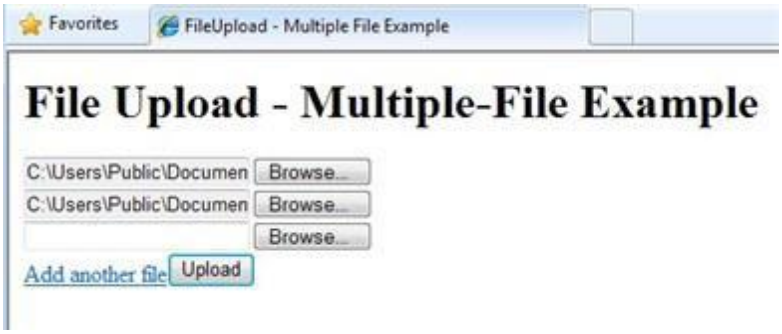
A kapott számmal már a `Request.Files`-on keresztüli hurkot létrehozhatjuk, minden fájlt megszerezhetünk és elmenthetünk. Amikor egy ismert számú gyűjteményen akarunk hurkot létrehozni, használhatjuk a `for` hurkot, így:

```
for(int i =0; i < numFiles; i++) {
    var uploadedFile = Request.Files[i];
    if (uploadedFile.ContentLength > 0) {
        fileName = Path.GetFileName(uploadedFile.FileName);

    // etc.
}
```

Az `i` változó csak egy ideiglenes számláló, ami nullától az általunk beállított felső határig megy. Ebben az esetben a felső határ a fájlok száma. De mivel a számláló 0-val kezd, a felső határ igazából eggyel kevesebb, mint a fájlok száma. (Ha három fájlt töltenek fel, a számláló nullától kettőig megy.)

Az `uploadedCount` változó összesíti a sikeresen feltöltött és elmentett fájlokat. Ez a kód számba veszi azt a lehetőséget, hogy a kívánt fájl nem mindig feltölthető.

4. Futtassuk az oldalt a böngészőben. A böngésző megjeleníti az oldalt és a két feltöltő dobozt!
5. Válasszunk ki két feltöltendő fájlt!
6. Kattintsunk az *Add another file*-ra! Az oldalon megjelenik egy új feltöltő doboz.



7. Kattintsunk az *Upload* gombra!
8. A weboldalon kattintsunk jobb gombbal a projektmappára, majd a *Refresh* gombra!
9. Nyissuk meg az *UploadedFiles* mappát, hogy lássuk a sikeresen feltöltött fájlokat!

További forrás angolul

- Exportálás CSV fájlba

10.fejezet – Munka képekkel

Ez a fejezet bemutatja, hogyan adjunk képeket a weboldalunkhoz, hogyan jeleníthetjük meg, és hogyan szerkeszthetjük –tükrözzük, méretezzük át és adjunk vízjelet hozzájuk –, mielőtt elmentenénk a képeket.

A fejezetből megtudhatjuk:

- Hogyan adjunk dinamikusan egy oldalhoz képet?
- Hogyan engedélyezhetjük a felhasználóknak képek feltöltését?
- Hogyan méretezzünk át egy képet?
- Hogyan forgassunk vagy tükrözzünk egy képet?
- Hogyan adjunk vízjelet egy képhez?
- Hogyan használjunk egy képet vízjelként?

Az alábbi ASP.NET programozási funkciókat ismerhetjük meg a fejezetben:

- A `WebImage` helpert
- A `Path` objektumot, amivel fájlneveket és elérési utakat módosíthatunk.

Kép dinamikus hozzáadása egy weboldalhoz

Webhelyünkhöz és különböző oldalakhoz hozzáadhatunk képeket, miközben szerkesztjük a weboldalt. A felhasználók számára is lehetővé tehetjük képek feltöltését, ami hasznos lehet például olyan feladatok esetében, mint egy profilkép feltöltése.

Ha egy kép már elérhető az oldalunkon és csak meg szeretnénk jeleníteni, akkor egy HTML `<img>` elemet használunk így:

```
<img src="images/Photo1.jpg"  alt="Sample Photo" />
```

Néha azonban a képeket dinamikusan kell megjeleníteni – azaz az oldal futtatásáig nem is tudjuk, milyen képet kell megjeleníteni.

Ebben a részben az eljárás megmutatja, hogyan jelenítsünk meg egy képet menet közben, ahol a felhasználók határozzák meg a képfájl nevét egy képfájl listából. Kiválasztják egy kép nevét a legördülő listából, és amikor elküldik az oldalt, a kiválasztott kép megjelenik.

1. A WebMatrix-ban hozzunk létre egy új weboldalt!
2. Az új oldalt nevezzük el *DynamicImage.cshtml*-nek!
3. A weboldal gyökérkönyvtárában hozzunk létre egy új mappát és nevezzük el *images*-nek!
4. Rakjunk a most létrehozott *images* mappába négy képet! (Bármilyen kéznél lévő kép megteszi, de az oldalba bele kell, hogy férjenek.) Nevezzük át a képeket *Photo1.jpg, Photo2.jpg, Photo3.jpg, Photo4.jpg*-re! (Ebben a folyamatban nem használjuk a *Photo4.jpg*-t, de majd a későbbiekben igen.)
5. Győződjünk meg róla, hogy a képek nem írásvédettek!
6. Cseréljük ki a meglévő kódot a következőre:

```
@{  var imagePath= "";
    if( Request["photoChoice"] != null){
        imagePath = @"images\" + Request["photoChoice"];
   }
}
<!DOCTYPE html>
<html>
<head>
  <title>Display Image on the Fly</title>
</head>
<body>
<h1>Displaying an Image On the Fly</h1>
<form method="post" action="">
    <div>
        I want to see:
        <select name="photoChoice">
            <option value="Photo1.jpg">Photo 1</option>
            <option value="Photo2.jpg">Photo 2</option>
            <option value="Photo3.jpg">Photo 3</option>
        </select>
         
        <input type="submit" value="Submit" />
    </div>
    <div style="padding:10px;">
        @if(imagePath != ""){
            <img src="@imagePath" alt="Sample Image" width="300px" />
```

```
        }
    </div>
</form>
</body>
</html>
```

A weboldal szövegtörzse egy legördülő listát tartalmaz (egy `<select>` elemet), ezt `photoChoice`-nak nevezzük. A lista 3 opciót tartalmaz, és minden opció `value` attribútuma az *images* mappában található képek neveit viseli. Lényegében a lista a felhasználó számára egy barátságos választható nevet biztosít, mint „Photo1”, majd hozzáadja a *.jpg* fájlnevet, amikor az oldalt elküldik.

A kódban megkaphatjuk a felhasználó választását (más szóval a képfájl nevét) a listából a `Request["photoChoice"]` olvasásával. Először láthatjuk, hogy egyáltalán volt-e kiválasztva valami. Ha volt, létrehozunk egy elérési utat a képnek, ami a képeket tartalmazó mappának a nevéből és a felhasználó képének fájlnevéből áll. (Ha megpróbálnánk elérési utat készíteni, de semmi nincs a `Request["photoChoice"]`-ban, akkor hibaüzenetet kapnánk.) Ez egy relatív elérési utat ad, így:

images/Photo1.jpg

Az elérési út az `imagePath` változó alatt van tárolva, amire később szükségünk lesz.

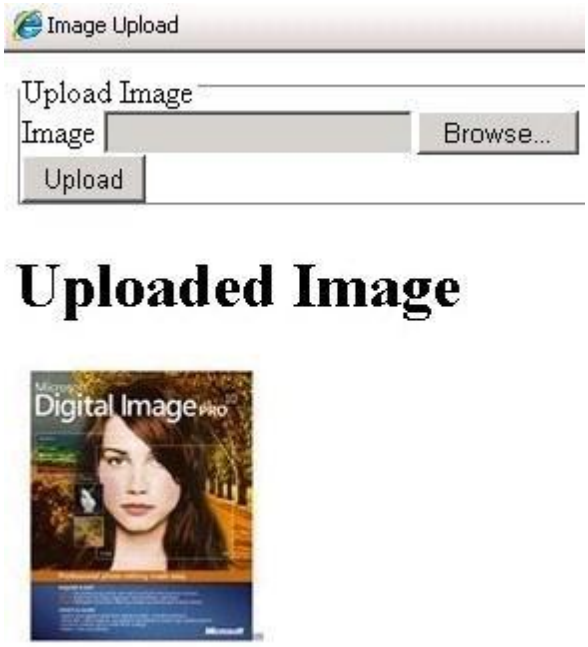
A szövegtörzsben található egy `<img>` elem is, amit a felhasználó által kiválasztott kép megjelenítésére használunk. Az `src` attribútum nincs fájlnévre vagy URL-re állítva, mintha egy statikus elemet szeretnénk megjeleníteni. Ehelyett a `@imagePath`-ra van állítva, azaz az értékét a kódban beállított elérési útból kapja meg.

Azonban amikor az oldal először fut, nincs megjelenítendő kép, mert a felhasználó nem választott ki semmit. Ez azt jelenti, hogy az `src` attribútum üres lesz, és a kép helyén egy piros X jelenik meg (vagy bármi más, amit a böngésző akkor jelenít meg, ha nem találja meg a képet). Ennek megakadályozására az `<img>` elemet egy `if` blokkba rakjuk, ami megnézi, hogy az `imagePath` változó tartalmaz-e valamit. Ha a felhasználó kiválaszt valamit, az `imagePath` tartalmazza az elérési utat. Ha a felhasználó nem választott ki képet, vagy az oldal először fut, akkor az `<img>` elem nem lesz megjelenítve.

7. Mentsük el a fájlt, és futtassuk az oldalt a böngészőben! (Mielőtt futtatjuk, győződjünk meg róla, hogy az oldal ki van választva a *Files* munkaterületen.)

Kép feltöltése

Az előző példában láthattuk, hogyan jelenítsünk meg egy képet dinamikusan, de csak olyan képekkel működött, melyek már a weboldalunkon voltak. Ez az eljárás megmutatja, hogyan engedélyezhetjük a felhasználók számára egy kép feltöltését, ami utána megjelenik a weboldalon. Az ASP.NET-ben menet közben szerkeszthetünk képeket a `WebImage` helperrel, ami olyan metódusokat tartalmaz, amivel képeket készíthetünk, módosíthatunk és menthetünk. A `WebImage` helper az összes gyakori webes képfájl típust támogatja, többek között a *.jpg, .png és .bmp* fájlokat. Ebben a fejezetben *.jpg* fájlokat fogunk használni, de bármilyen képfájl típust használhatunk.



1. Hozzunk létre egy új oldalt, és nevezzük el *UploadImage.cshtml*-nek!
2. Cseréljük ki a meglévő kódot a következőre:

```
@{  WebImage photo = null;
    var newFileName = "";
    var imagePath = "";

    if(IsPost){
        photo = WebImage.GetImageFromRequest();
        if(photo != null){
            newFileName = Guid.NewGuid().ToString() + "_" +
                Path.GetFileName(photo.FileName);
            imagePath = @"images\" + newFileName;

            photo.Save(@"~\" + imagePath);
        }
    }
}
<!DOCTYPE html>
<html>
<head>
  <title>Image Upload</title>
</head>
<body>
  <form action="" method="post" enctype="multipart/form-data">
    <fieldset>
      <legend> Upload Image </legend>
      <label for="Image">Image</label>
      <input type="file" name="Image" />
      <br/>
      <input type="submit" value="Upload" />
    </fieldset>
  </form>
  <h1>Uploaded Image</h1>
    @if(imagePath != ""){
    <div class="result">
        <img src="@imagePath" alt="image" />
```

```
    </div>
    }
</body>
</html>
```

A szöveg törzse tartalmaz egy `<input type="file">` elemet, amivel a felhasználók kiválaszthatnak egy feltöltendő fájlt. Amikor a *Submit*-ra kattintanak, az űrlappal együtt a fájl, amit kiválasztottak is elküldésre kerül.

A feltöltött kép megszerzéséhez a `WebImage` helpert használjuk, ami sok hasznos metódust tartalmaz a képekkel való munkához. Most a `WebImage.GetImageFromRequest`-et használjuk a feltöltött kép megszerzéséhez és a `photo` változó alatti tárolásához.

Ebben a fejezetben sok munkánk lesz az elérési utak és fájlnevek megszerzésével és beállításával. A probléma az, hogy szeretnénk a felhasználó által feltöltött kép nevét (és csak a nevét) megszerezni, majd ezután egy új elérési utat készíteni neki oda, ahol tárolni fogjuk a képet. Mivel a felhasználók feltölthetnek esetleg több képet is ugyanolyan névvel, ezért egy kis extra kódot használunk az egyedi nevek létrehozásához, így biztosítva, hogy a felhasználók nem írnak felül meglévő képeket.

Ha egy képet feltöltöttek (a teszt `if (photo != null)`), akkor a kép nevét a `FileName` tulajdonságából megkapjuk. Amikor a felhasználó feltölti a képet, a `FileName` a felhasználó eredeti nevét tartalmazza, ami magában foglalja a felhasználó gépén lévő elérési utat. Így nézhet ki:

C:\Users\Joe\Pictures\SamplePhoto1.jpg

Azonban nincs szükségünk erre az elérési útra – csak az aktuális fájlnév (*SamplePhoto1.jpg*) kell nekünk. A fájlnevet kiszedhetjük ez elérési útból a `Path.GetFileName` metódussal, így:

```
Path.GetFileName(photo.FileName)
```

Ekkor létrehozunk egy egyedi fájlnevet az eredeti fájlnévből egy GUID hozzáadásával. (A GUID-okról több információt a fejezet későbbi részében olvashatunk.) Ezután készítünk egy komplett elérési utat, amit a kép mentéséhez használunk. A mentéshez szükséges elérési út az új fájlnévből, a mappából (*images*) és az aktuális weboldal helyéből áll.

Megjegyzés: Ahhoz, hogy a kód az *images* mappába mentse a fájlokat, az alkalmazásnak írás-olvasási jogosultságra van szüksége erre a mappára. A fejlesztő számítógépen általában ez nem probléma. Azonban amikor egy hostolt webszerveren publikáljuk a weboldalunkat, kifejezetten szükség lehet az engedélyek beállítására. Ha ezt a kódot futtatjuk a webhosting szolgáltató szerverén, és hibaüzenetet kapunk, vegyük fel a kapcsolatot a szolgáltatóval, hogy hogyan állíthatók be ezek az engedélyek!

Végül a mentéshez szükséges elérési utat *a* `WebImage` helper `Save` metódusához adjuk. Ez elmenti a feltöltött képet az új neve alatt. A `Save` metódus így néz ki: `photo.Save(@"~\" + imagePath)`. A komplett elérési utat hozzáfűzi a `@"~\"`-hoz, ami az aktuális weboldal helye. (A ~ operátorról bővebbi információ a 3. fejezetben található.)

Ahogy az előző példában, az oldal szövegtörzse egy `<img>` elemet tartalmaz a kép megjelenítéséhez. Ha az `imagePath` be lett állítva, az `<img>` elem megjelenik és az `src` attribútuma az `imagePath` értékre állítódik.

3. Futtassuk az oldalt a böngészőben!

A GUID-okról

A GUID (globally-unique ID) egy azonosító, ami hasonlóan néz ki: *936DA01F-9ABD-4d9d-80C7-02AF85C822A8*. (Technikailag egy 16 byte/128-bites szám.) Amikor egy GUID-ra van szükségünk, egy speciális kódot hívhatunk meg, ami generál egy GUID-ot nekünk. A GUID lényege, hogy a szám hatalmas mérete ($3{,}4 \times 10^{38}$) és a generáló algoritmusa egy gyakorlatilag garantáltan egyedi számot hoz létre. Ezért a GUID-ok hasznosak olyan dolgok nevének generálására, ahol garantálni kell, hogy nem használjuk kétszer ugyanazt a nevet. A másik oldalról persze a GUID-ok nem éppen felhasználóbarátok, ezért csak akkor használjuk őket, amikor a neveket csak a kódban használjuk.

Kép átméretezése

Ha a weboldalunk képeket fogad egy felhasználótól, a mentése vagy a megjelenítése előtt szükségünk lehet a kép átméretezésére. Ehhez újra a `WebImage` helpert használhatjuk.

Ez az eljárás megmutatja, hogyan méretezzük át a feltöltött képet miniatűr létrehozásához, majd hogyan mentsük a miniatűrt és az eredeti képet a weboldalon. A miniatűrt megjelenítjük a weboldalon, és hiperhivatkozással átirányítjuk a felhasználókat a teljes méretű képhez.

1. Hozzunk létre egy új oldalt *Thumbnail.cshtml* névvel!
2. Az *images* mappában készítsünk egy almappát *thumbs* névvel!
3. Cseréljük ki a meglévő kódot a következőre:

```
@{  WebImage photo = null;
    var newFileName = "";
    var imagePath = "";
    var imageThumbPath  = "";

    if(IsPost){
        photo = WebImage.GetImageFromRequest();
        if(photo != null){
             newFileName = Guid.NewGuid().ToString() + "_" +
                 Path.GetFileName(photo.FileName);
             imagePath = @"images\" + newFileName;
             photo.Save(@"~\" + imagePath);

            imageThumbPath = @"images\thumbs\" + newFileName;
            photo.Resize(width: 60, height: 60, preserveAspectRatio:
true,
```

```
                preventEnlarge: true);
            photo.Save(@"~\" + imageThumbPath);         }
    }
}
<!DOCTYPE html>
<html>
<head>
  <title>Resizing Image</title>
</head>
<body>
<h1>Thumbnail Image</h1>
  <form action="" method="post" enctype="multipart/form-data">
    <fieldset>
      <legend> Creating Thumbnail Image </legend>
      <label for="Image">Image</label>
      <input type="file" name="Image" />
      <br/>
      <input type="submit" value="Submit" />
    </fieldset>
  </form>
    @if(imagePath != ""){
    <div class="result">
        <img src="@imageThumbPath" alt="Thumbnail image" />
        <a href="@Html.AttributeEncode(imagePath)" target="_Self">
            View full size
        </a>
    </div>

    }
</body>
</html>
```

Ez a kód hasonló az előző példához. A különbség az, hogy ez a kód kétszer menti el a képet, egyszer normálisan és egyszer, miután egy bélyegképet készítettünk a képből. Először megkapjuk a feltöltött képet, és elmentjük az *images* mappába. Ezután létrehozunk egy új elérési utat a bélyegkép számára. A bélyegkép létrehozásához a `WebImage` helper `Resize` metódusát használjuk, hogy egy 60x60 pixeles képet készítsen. A példa megmutatja, hogyan tartsuk meg az oldalarányt, és hogyan akadályozzuk meg, hogy a képet kinagyítsa (abban az esetben, ha az új méret nagyobbá tenné a képet). Az átméretezett kép ezután a *thumbs* almappában kerül mentésre.

A kód végén az `<img>` elemet használjuk a dinamikus `src` attribútummal, amit a korábbi példákban a képek feltételes megjelenítésénél láthattunk. Ebben az esetben a bélyegképet jelenítjük meg. Egy `<a>` elemet is használunk egy hiperhivatkozás létrehozásához, ami a kép nagy változatához kapcsolódik. Ahogy az `src` attribútumot az `<img>` elemhez, a `href` attribútumot az `<a>` elemhez állítjuk dinamikusan, ami az `imagePath`-ban van. Hogy megbizonyosodjuk arról, hogy az elérési út URL-ként működik, az `imagePath`-t a `Html.AttributeEncode` metódushoz adjuk, ami az elérési út korlátozott karaktereit átalakítja az URL-hez használható karakterekké.

4. Futtassuk az oldalt a böngészőben!

Kép forgatása és tükrözése

A `WebImage` helper segítségével tükrözhetjük és elforgathatjuk a képeket. Ez az eljárás bemutatja, hogyan kapjuk meg a képet a szerverről, hogyan tükrözzük függőlegesen, hogyan mentsük el, majd

hogyan jelenítsük meg a tükrözött képet az oldalon. Ebben a példában csak egy olyan képet használunk, ami már megtalálható a szerveren (*Photo2.jpg*). Valós felhasználásban valószínűleg egy olyan képet tükröznénk, aminek a nevét dinamikusan kapjuk meg, mint ahogy az előző példákban láthattuk.

1. Hozzunk létre egy új oldalt *Flip.cshtml* névvel!
2. Cseréljük ki a meglévő kódot a következőre:

```
@{  var imagePath= "";
    WebImage photo = new WebImage(@"~\Images\Photo2.jpg");
    if(photo != null){
        imagePath = @"images\Photo2.jpg";
        photo.FlipVertical();
        photo.Save(@"~\" + imagePath);
     }
}
<!DOCTYPE html>
<html>
<head>
  <title>Get Image From File</title>
  <meta http-equiv="content-type" content="text/html;charset=utf-8"
/>
</head>
<body>
<h1>Flip Image Vertically</h1>
@if(imagePath != ""){
  <div class="result">
    <img src="@imagePath" alt="Image" />
  </div>
}
</body>
</html>
```

A kód a `WebImage` helpert használja, hogy megkapjuk a képet a szerverről. Mi készítjük el az elérési utat a képhez, ugyanazt a technikát alkalmazva, amit a korábbi példákban képek mentéséhez használtunk, és ezt az elérési utat a kép létrehozásánál a `WebImage` használatával átadjuk:

```
WebImage photo = Snew WebImage(@"~\Images\Photo2.jpg");
```

Ha talál egy képet, létrehozunk egy új elérési utat és fájlnevet, mint ahogy a korábbi példákban tettük. A kép tükrözéséhez a `FlipVertical` metódust hívjuk, majd újra elmentjük a képet.

A kép újra az `<img>` metódus használatával, az `src` attribútum `imagePath`-ra állításával jelenik meg az oldalon.

3. Futtassuk az oldalt a böngészőben! A *Photo2.jpg* képe fejjel lefele elenik meg. Ha újra lekérdezzük az oldalt, a kép újra felfelé tükrözve jelenik meg.

Kép elforgatásához ugyanezt a kódot használjuk, kivéve, hogy a `FlipVertical` vagy `FlipHorizontal` hívása helyett a `RotateLeft`-et vagy a `RotateRight`-ot hívjuk.

Vízjel hozzáadása képhez

Ha a weboldalunkhoz képeket töltünk fel, szükségünk lehet vízjel hozzáadására a képek mentése és megjelenítése előtt. Gyakran azért használnak vízjelet, hogy szerzői jogokról szóló információkat rakjanak a képre, vagy hogy reklámozzák a márkanevüket.

1. Hozzunk létre egy új oldalt *Watermark.cshtml* névvel!
2. Cseréljük ki a meglévő kódot a következőre:

```
@{  var imagePath= "";
    WebImage photo = new WebImage(@"~\Images\Photo3.jpg");
    if(photo != null){
        imagePath = @"images\Photo3.jpg";
        photo.AddTextWatermark("My Watermark", fontColor:"Yellow",
fontFamily:
            "Arial");
        photo.Save(@"~\" + imagePath);    }
}
<!DOCTYPE html>
<html>
<head>
  <title>Water Mark</title>
  <meta http-equiv="content-type" content="text/html;charset=utf-8"
/>
</head>
<body>
<h1>Adding a Watermark to an Image</h1>
@if(imagePath != ""){
  <div class="result">
    <img src="@imagePath" alt="Image" />
```

```
  </div>
}
</body>
</html>
```

Ez a kód olyan, mint a *Flip.cshtml* oldal kódja korábbról (bár most a *Photo3.jpg* fájlt használja). Vízjel hozzáadásához a `WebImage` helper `AddTextWatermark` metódusát használja mentés előtt. Az `AddTextWatermark` hívásához a „My Watermark” szöveget adjuk, a betűszínt sárgára állítjuk, és a betűtípust Arialra állítjuk.(Bár itt nem mutatjuk be, de a `WebImage` segítségével az átlátszóságot, betűtípust, betűméretet és a vízjel elhelyezkedését is beállíthatjuk.) Amikor mentjük a képet, nem lehet írásvédett.

Ahogy korábban láttuk, a kép az `<img>` elem `src` attribútumának `@imagePath`-ra állításával jelenik meg az oldalon.

3. Futtassuk az oldalt a böngészőben!

Kép használata vízjelként

Vízjelként használhatunk szöveg helyett másik képet is. Néha egy céglogót használnak vízjelként, vagy a szerzői jogokról szóló információknál is szöveg helyett képet használnak.

1. Hozzunk létre egy új oldalt *ImageWatermark.cshtml* névvel!
2. Rakjunk egy olyan képet az *images* mappába, amit logóként szeretnénk használni, és nevezzük át a képet *MyCompanyLogo.jpg*-re! Ez egy olyan kép kell, hogy legyen, ami tisztán látható akkor is, ha 80 pixel széles és 20 pixel magasra van állítva.
3. Cseréljük ki a meglévő kódot a következőre:

```
@{  var imagePath = "";
   WebImage WatermarkPhoto = new WebImage(@"~\" +
        @"\Images\MyCompanyLogo.jpg");
    WebImage photo = new WebImage(@"~\Images\Photo4.jpg");
    if(photo != null){
        imagePath = @"images\Photo4.jpg";
        photo.AddImageWatermark(WatermarkPhoto, width: 80, height:
20,
           horizontalAlign:"Center", verticalAlign:"Bottom",
           opacity:100,  padding:10);
      photo.Save(@"~\" + imagePath);
   }
}
<!DOCTYPE html>
```

```
<html>
<head>
  <title>Image Watermark</title>
  <meta http-equiv="content-type" content="text/html;charset=utf-8"
/>
</head>
<body>
  <h1>Using an Image as a Watermark</h1>
  @if(imagePath != ""){
    <div class="result">
      <img src="@imagePath" alt="Image" />
    </div>
  }
</body>
</html>
```

Ez egy variációja a korábbi példákban látott kódoknak. Ebben az esetben az `AddImageWatermark`-ot hívjuk a vízjel célképhez (*Photo3.jpg*) való hozzáadásához, mielőtt elmentenénk a képet. Amikor az `AddImageWatermark`-ot hívjuk, szélességét 80 pixelre, magasságát 20 pixelre állítjuk. A *MyCompanyLogo.jpg* a célkép aljára középre kerül elhelyezésre. Az átlátszóság 100%-ra van állítva, a padding pedig 10 pixelre. Ha a vízjel kép nagyobb, mint a célkép, semmi sem történik. Ha a vízjel kép nagyobb, mint a célkép, és a vízjel padding-ot nullára állítjuk, a vízjelet figyelmen kívül hagyja.

Mint korábban, a képet az `<img>` elemmel és egy dinamikus `src` attribútummal jelenítjük meg.

4. Futtassuk az oldalt a böngészőben!

11.fejezet – Műveletek videókkal

Van egy videónk? Ha igen, akkor könnyen megjeleníthetjük egy weboldalon. Az ASP.NET segítségével képesek vagyunk lejátszani Flash (*.swf*), Media Player (*.wmv*), valamint Silverlight (*.xap*) videókat.

Az útmutatóból megtudhatjuk:

- Hogyan válasszunk videólejátszót?
- Hogyan adjunk egy weblaphoz videót?
- Hogyan állítsuk be a lejátszó tulajdonságait?

Ezek a fejezetben bemutatott újdonságai az ASP.NET weboldalaknak:

- A `video` helper.

Videolejátszó kiválasztása

Rengeteg video fájlformátum létezik, és általában mindegyikhez különböző lejátszó szükségeltetik, különböző úton kell konfigurálni a lejátszót. Az ASP.NET weboldalakban le tudunk játszani egy videót a `video` helper használatával. A `video` helper leegyszerűsíti a beágyazott videók folyamatait a weboldalon, mivel automatikusan legenerálja az `object` és az `embed` HTML elemeket, amelyeket általában a videók hozzáadásához használunk.

A video helper a következő lejátszókat támogatja:

- Adobe Flash
- Windows Media Player
- Microsoft Silverlight

A Flash lejátszó

A `video` helper `flash` lejátszója lehetővé teszi nekünk a Flash videók (*.swf* fájlok) lejátszását a weboldalon. Legalább a fájl elérési útját biztosítanunk kell. Hogyha nem adunk meg semmit sem, csak az elérési utat, a lejátszó az alapértelmezett értékeket használja, melyet a Flash aktuális verziója alapján fog beállítani. Tipikus beállítások:

- A video az alapértelmezett szélességgel és magassággal fog megjelenni, háttérszín nélkül.
- A video automatikusan elindul, mikor az oldal betölt.
- A video folyamatosan ismétlődik, míg azt le nem állítjuk.
- A video úgy van méretezve, hogy a teljes videót megjelenítse, nem pedig úgy, hogy a szélek levágásával kitöltse a keretet.
- A video egy ablakban fog lejátszódni.

A MediaPlayer lejátszó

A `video` helper `MediaPlayer` lejátszója lehetővé teszi nekünk a Windows Media Video (*.wmv* fájlok), Windows Media Audio (*.wma* fájlok), valamint MP3 (*.mp3* fájlok) lejátszását. Lejátszáshoz szükségünk van a média fájl elérési útvonalára; a többi paramétert nem kötelező megadnunk. Hogyha nem adunk meg semmit sem, csak az elérési utat, a lejátszó az alapértelmezett értékeket használja, melyet a Media Player aktuális verziója alapján fog beállítani, mint ezek:

- A video az alapértelmezett szélességgel és magassággal fog megjelenni.
- A viedo automatikusan elindul, mikor az oldal betölt.
- A video csak egyszer fog lejátszódni (nem ismétlődik).

- A lejátszó kijelzi a teljes kezelőfelületet a felhasználói felületen.
- A video egy ablakban fog lejátszódni.

A Silverlight lejátszó

A `video` helper `Silverlight` lejátszója lehetővé teszi nekünk a Windows Media Video (.wmv fájlok), Windows Media Audio (.wma fájlok), valamint MP3 (.mp3 fájlok) lejátszását. Be kell állítanunk az elérési út paramétereit, hogy a Silverlight alapú alkalmazáscsomagra mutasson. A szélesség és magasság paramétereit is be kell állítanunk. A többi paramétert nem kötelező megadnunk. Mikor a Silverlight lejátszót úgy használjuk, hogy csak a szükséges paramétereket adtuk meg, a Silverlight lejátszó háttérszín nélkül fogja megjeleníteni a videót.

Megjegyzés: Ha még nem ismernénk a Silverlightot: a *.xap* fájl egy tömörített állomány, amely az *.xaml* fájlban szerkezeti utasításokat, a részegységekben kezelt kódokat és tetszőleges erőforrásokat tartalmaz. Mint Silverlight application project, készíthetünk *.xap* fájlokat a Visual Studióban.

A `Sliverlight` lejátszó használja az általunk megadott beállításokat és a *.xap* fájlban tárolt beállításokat is.

MIME típusok

Amikor a böngésző letölti a fájlt, a böngésző gondoskodik arról, hogy a fájl típusa megegyezzen a megjeleníteni kívánt weboldal MIME típusával. A MIME-típus egy fájl tartalmi vagy média típusa. A video helper a következő MIME-típusokat használja:

- `application/x-shockwave-flash`
- `application/x-mplayer2`
- `application/x-silverlight-2`

Flash videók (.swf) lejátszása

Az eljárás megmutatja nekünk, hogyan játsszunk le egy *sample.swf* nevű Flash videót. A feladathoz szükségünk van arra, hogy legyen egy *Media* nevű mappánk a webhelyen, ami tartalmazza a *.swf* fájlt.

1. Ha még nem tettük volna meg, akkor adjuk hozzá az ASP.NET Web Helpers Libraryt a webhelyünkhöz, ahogyan az a 2. fejezetben le van írva!
2. A hozzunk létre egy új lapot, és nevezzük el *FlashVideo.cshtml*-nek!
3. Adjuk hozzá a következő kódot a laphoz:

```
<!DOCTYPE html>
<html>
<head>
    <title>Flash Video</title>
</head>
<body>
    @Video.Flash(path: "Media/sample.swf",
                 width: "400",
                 height: "600",
                 play: true,
                 loop: true,
                 menu:  false,
                 bgColor: "red",
                 quality: "medium",
                 scale: "exactfit",
                 windowMode: "transparent")
</body>
</html>
```

4. Futtassuk az oldalt egy böngészőben! (Győződjünk meg arról, hogy az oldal ki van választva a *Files* munkaterületen, mielőtt futtatjuk.) Az oldal és a video automatikusan meg fog jelenni.

Beállíthatjuk a Flash video `quality` paraméterét `low`-ra, `autolow`-ra, `medium`-ra, `high`-ra és `best`-re.

```
// Set the Flash video quality
@Video.Flash(path: "Media/sample.swf", quality: "autohigh")
```

A következő módon beállíthatjuk, hogy a Flash video a meghatározott méretekkel játszódjon le:

- `showall`: Ez a teljes videót láthatóvá teszi, miközben megőrzi az eredeti képarányt. Előfordulhat azonban, hogy a széleken fekete sávok lesznek láthatók.
- `noorder`: Ez átméretezi a videót, megtartva az eredeti képarányt, de lehet, hogy levágja egy részét.
- `exactfit`: Ez a teljes videót láthatóvá teszi az eredeti képarány megtartása nélkül, ekkor fennáll a torzulás veszélye.

Hogyha nem adjuk meg a méretezési paramétereket, a teljes videó megjelenik az eredeti képaránnyal bármilyen vágás nélkül. A következő példa mutatja, hogyan használjuk a `scale` paramétereit:

```
// Set the Flash video to an exact size
@Video.Flash(path: "Media/sample.swf", width: "1000", height:
"100",scale: "exactfit")
```

A Flash lejátszó támogatja a `windowMode` nevű video módot. `window`-ra, `opaque`-ra, valamint `transparent`-re állíthatjuk. Alapértelmezés szerint a `windowMode` `window`-ra van állítva, ami a videót egy

külön ablakban jeleníti meg a weboldalon. Az `opaque` beállítás mindent a video mögé rejt a weboldalon. A `transparent` beállítás hagyja, hogy a weboldal háttere megjelenjen a videón keresztül, feltételezve azt, hogy van a videón átlátszó rész.

Media Player (.wmv) videók lejátszása

A következő feladat megmutatja nekünk, hogyan játsszunk le egy *sample.wmv* nevű Windows Media videót, amit a *Media* mappában találunk.

1. Ha még nem tettük volna meg, akkor adjuk hozzá az ASP.NET Web Helpers Libraryt az oldalunkhoz, ahogy az a 2. fejezetben le van írva!
2. Készítsünk egy új, *MediaPlayerVideo.cshtml* nevű lapot!
3. Adjuk hozzá a következő utasítást a laphoz:

```
<!DOCTYPE html>
<html>
<head>
  <title>MediaPlayer Video</title>
</head>
<body>
    @Video.MediaPlayer(
        path: "Media/sample.wmv",
        width: "400",
        height: "600",
        autoStart: true,
        playCount: 2,
        uiMode:  "full",
        stretchToFit: true,
        enableContextMenu: true,
        mute: false,
        volume: 75)
</body>
</html>
```

4. Futtassuk az oldalt egy böngészőben! A video automatikusan be fog tölteni és le fog játszódni.

Megadhatunk a `playCount`nak valamilyen egész számot, amely azt jelzi, hogy hányszor kell automatikusan lejátszania a videót:

```
// Set the MediaPlayer video playCount
@Video.Flash(path: "Media/sample.swf", playCount: 2)
```

Az `uiMode` paraméter lehetőséget ad arra, hogy megszabjuk, melyik irányítóeszközök jelenjenek meg a felhasználói felületen. Beállíthatjuk `invisible`-re, `none`-ra, `mini`-re, valamint `full`-ra. Amikor nem határozzuk meg az `uiMode` paramétert, a video az állapotablakkal, keresősávval, vezérlőgombokkal és hagerőszabályzóval fog megjelenni a video ablaka mellett. Ezek a kezelőszervek akkor is meg fognak jelenni, ha a lejátszót hangfájlok lejátszására használjuk. Íme egy példa, hogyan használjuk az `uiMode` paramétert:

```
// Set the MediaPlayer control UI
@Video.MediaPlayer(path: "Media/sample.wmv", uiMode: "mini")
```

Alapértelmezetten a hang be van kapcsolva, mikor a video lejátszás alatt van. Lenémíthatjuk a hangot, ha a `mute` paramétert igazra állítjuk:

```
// Play the MediaPlayer video without audio
@Video.MediaPlayer(path: "Media/sample.wmv", mute: true)
```

A Media Player video hangerejét irányíthatjuk, ha a `volume` paraméter értékét 0 és 100 között állítjuk. Alapértelmezetten ez az érték 50. Íme egy példa:

```
// Play the MediaPlayer video without audio
@Video.MediaPlayer(path: "Media/sample.wmv", volume: 75)
```

Silverlight videók lejátszása

A következő feladat megmutatja nekünk, hogyan játsszunk le videót, amelyet egy *Media* nevű mappában lévő Silverlight *.xap* oldal tartalmaz.

1. Ha még nem tettük volna meg, akkor adjuk hozzá az ASP.NET Web Helpers Libraryt az oldalunkhoz, ahogyan azt a 2. fejezetben olvashattuk!
2. Készítsünk egy új, *SilverlightVideo.cshtml* nevű oldalt!
3. Adjuk hozzá a következő utasítást a laphoz:

```
<!DOCTYPE html>
<html>
<head>
  <title>Silverlight Video</title>
</head>
<body>
    @Video.Silverlight(
        path: "Media/sample.xap",
        width: "400",
        height: "600",
        bgColor: "red",
        autoUpgrade: true)
</body>
</html>
```

4. Futtassuk az oldalt egy böngészőben!

További források angolul

- Silverlight Overview
- Flash OBJECT and EMBED tag attributes
- Windows Media Player 11 SDK PARAM Tags

12.fejezet – E-mail hozzáadása a webhelyünkhöz

Ebben a fejezetben megtanuljuk, hogyan küldhetünk automatikus e-mail üzenetet webhelyünkről.

Amit megtanulunk:

- Hogyan küldjünk egy e-mail üzenetet a weboldalról?
- Hogyan csatoljunk fájlt az e-mail üzenethez?

Ezt az ASP.Net szolgáltatást vezetjük be a fejezetben:

- A `WebMail` helper.

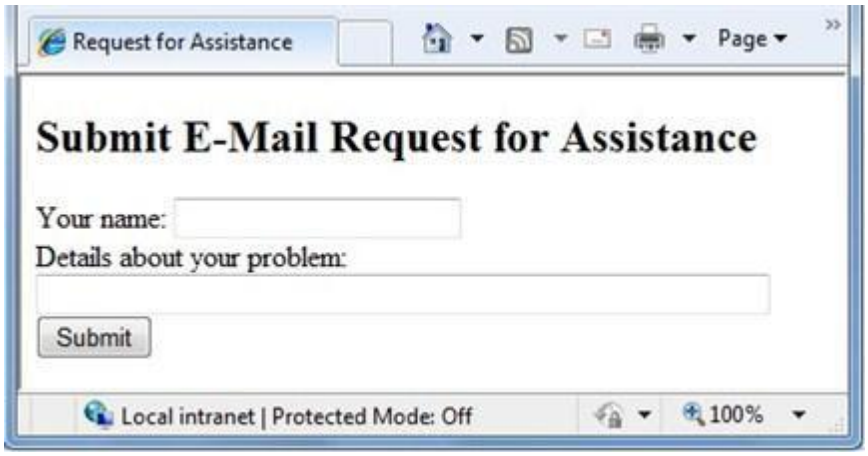
E-mail üzenet küldése a webhelyről

Sokféle okból kellhet a webhelyünkről e-mailt küldeni. Megerősítő üzenetet küldhetünk a felhasználónak vagy értesítéseket magunknak (például, hogy egy új felhasználó regisztrált.) A `WebMail` helper egyszerűvé teszi nekünk ezeket a műveleteket.

A `WebMail` helper használatához hozzáférésre van szükségünk egy SMTP szerverhez.(SMTP= Simple Mail Transfer Protocol) Az SMTP szerver egy e-mail szerver, ami továbbítja az üzeneteket a címzett szerverére. Ez a kimenő oldala az e-mailnek. Ha webhelyünk egy tárhely szolgáltatónál üzemel, akkor lehetséges, hogy rendelkezünk náluk e-mail fiókkal. Ez esetben kérjük a szolgáltató segítségét a beállítások elvégzéséhez. Ha saját vállalati rendszerünkön fut a webhely, a rendszergazda, vagy az IT részleg tud segíteni az SMTP szerver beállításaiban. Otthoni futtatás során akár saját e-mail fiókunkkal is tesztelhetjük a rendszert. Amire általában szükségünk van:

- Az SMTP szerver neve.
- A port száma. (Ez majdnem mindig 25. Azonban lehetséges, hogy a szolgáltatónk megköveteli az 587-es port használatát.)
- Azonosítók (felhasználónév, jelszó).

Ebben a feladatban két weboldalt készítünk. Az első oldal egy űrlap, amin a felhasználó egy leírást adhat, mintha egy technikai támogatást kérő űrlapot töltene ki. Az első oldal elküldi az információit a második lapnak. A második lapon a kód értelmezi a felhasználó információit, és küld egy e-mail üzenetet. Ez egy megerősítő üzenet a probléma bejelentésének fogadásáról.



Megjegyzés: Annak érdekében, hogy ez a példa egyszerű maradjon, a kód inicializálja a `WebMail` helpert közvetlenül azo az oldalon, ahol használni fogjuk. Ugyanakkor a valós webhelyen jobb ötlet az ehhez hasonló initializáló kódot beletenni egy globális fájlba ,hogy az inicializálja a `WebMail` helpert az

összes fájlnak a weboldalon. További információkért nézzük meg a 19. fejezetet (A Site-Wide viselkedés testreszabása)!

1. Készítsünk egy új weboldalt!
2. Adjuk hozzá a weboldalhoz az ASP.NET Web Helpers Library-t a 2. fejezetben leírtak szerint (ha még nem tettük volna meg)!
3. Adjunk hozzá egy új weboldalt *EmailRequest.cshtml* néven és írjuk be a következő kódot:

```
<!DOCTYPE html>
<html>
<head>
    <title>Request for Assistance</title>
</head>
<body>
  <h2>Submit Email Request for Assistance</h2>
  <form method="post" action="ProcessRequest.cshtml">
    <div>
        Your name:
        <input type="text" name="customerName" />
    </div>

    <div>
        Details about your problem: <br />
        <textarea name="customerRequest" cols="45"
rows="4"></textarea>
    </div>

    <div>
        <input type="submit" value="Submit" />
    </div>
  </form>
</body>
</html>
```

   Figyeljük meg, hogy az űrlap elem `action` attribútumának a *ProcessRequest.cshtml* van megadva. Ez azt jelenti, hogy az űrlapot a megadott helyre küldi az oldal, nem pedig az éppen megnyitottra.
4. Adjunk hozzá egy új lapot a website-hoz *ProcessRequest.cshtml* néven, és írjuk bele a következő kódot:

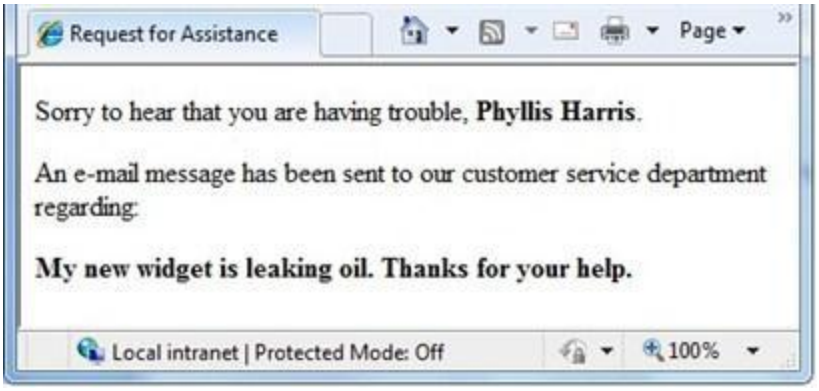
```
@{
    var customerName = Request["customerName"];
    var customerRequest = Request["customerRequest"];
    try {
        // Initialize WebMail helper
        WebMail.SmtpServer = "your-SMTP-host";
        WebMail.SmtpPort = 25;
        WebMail.EnableSsl = true;
        WebMail.UserName = "your-user-name-here";
        WebMail.From = "your-email-address-here";
        WebMail.Password = "your-account-password";

        // Send email
        WebMail.Send(to: "target-email-address-here",
            subject: "Help request from - " + customerName,
            body: customerRequest
        );
    }
    catch (Exception ex ) {
```

```
        <text>
            <b>The email was <em>not</em> sent.</b>
            The code in the ProcessRequest page must provide an
            SMTP server name, a user name, a password, and
            a "from" address.
        </text>
    }
}
<!DOCTYPE html>
<html>
<head>
  <title>Request for Assistance</title>
</head>
<body>
  <p>Sorry to hear that you are having trouble,
<b>@customerName</b>.</p>

  <p>An email message has been sent to our customer service
     department regarding the following problem:</p>

  <p><b>@customerRequest</b></p>
</body>
</html>
```

A kódban az űrlapmezők elküldött értékeivel dolgozunk. Ezután meghívjuk a `Webmail` helper `Send` metódusát az e-mail üzenet létrehozásához és elküldéséhez. Ebben az esetben a szöveget alkotó értékeket összekapcsoljuk az űrlapból nyert értékekkel.

Ennek az oldalnak a belsejében van egy `try/catch` block. Ha az e-mail elküldésére tett kísérlet sikertelen (például helytelenek a beállítások), az oldal feldob egy üzenetet. `<text>` címkével jelölhetünk több sornyi szöveget a kódon belül. (További információkért a `try/catch` blokkokkal vagy a `<text>` címkével kapcsolatban nézzük meg a 3. fejezet – Bevezetés az ASP.NET webszerkesztésbe Razor szintaxissal fejezetet.

5. Módosítsuk a következő e-mailhez kapcsolódó beállításokat a kódban:
   - Állítsuk be a `your-SMTP-host`-ot annak az SMTP szervernek a nevére, amihez hozzáférésünk van!
   - Állítsuk be a `your-user-name-here`-t az SMTP szerverünk felhasználónevére!
   - Állítsuk be a `your-email-address-here`-t a saját e-mail címünkre. Ez lesz az az e-mail cím, amiről az üzenetet elküldjük.
   - Állítsuk be a `your-account-password`-öt az SMTP szerver felhasználói fiókjának jelszavára.
   - Állítsuk be a `target-email-address-here`-t annak a személynek az e-mail címére, akinek el szeretnénk küldeni az üzenetet. Normálisan ez a címzett e-mail címe, de a tesztelés érdekében a saját e-mail címünket adjuk meg. Amikor futtatjuk az oldalt, meg fogjuk kapni az üzenetet.
6. Futtassuk az *EmailRequest.cshtml* oldalt a böngészőnkben! (Futtatás előtt bizonyosodjunk meg arról, hogy az oldal ki van választva a *Files* munaterületen!)
7. Adjuk meg a nevünket és a probléma leírást, és kattintsunk a *Submit* gombra! Átirányít minket a *ProcessRequest.cshtml* oldalra, ami megerősíti az üzenetet és elküldi az e-mail üzenetet.



Fájl küldése e-mail használatával

Küldhetünk fájlokat is az e-mail üzenethez csatolva. Ebben a feladatrészben készítünk egy szövegfájlt és két HTML oldalt. A szövegfájlt fogjuk használni e-mail csatolmánynak.

1. Hozzunk létre egy *MyFile.txt* nevű szöveges dokumentumot!
2. Másoljuk ki a lenti szöveget és illesszük be a fájlba:

```
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim
ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut
aliquip ex ea commodo consequat.
```

3. Készítsünk egy *SendFile.cshtml* nevű oldalt és írjuk be a következő kódot:

```
<!DOCTYPE html>
<html>
<head>
    <title>Attach File</title>
</head>
<body>
  <h2>Submit Email with Attachment</h2>
  <form method="post" action="ProcessFile.cshtml">
    <div>
        Your name:
        <input type="text" name="customerName" />
    </div>

    <div>
        Subject line: <br />
        <input type="text" size= 30 name="subjectLine" />
    </div>

    <div>
        File to attach: <br />
        <input type="text" size=60 name="fileAttachment" />
    </div>

    <div>
        <input type="submit" value="Submit" />
    </div>
  </form>
</body>
</html>
```

4. Készítsünk egy *ProcessFile.cshtml* nevű oldalt, és lássuk el a következő kóddal:

```
@{
    var customerName = Request["customerName"];
    var subjectLine = Request["subjectLine"];
    var fileAttachment = Request["fileAttachment"];

    try {
        // Initialize WebMail helper
        WebMail.SmtpServer = "your-SMTP-host";
        WebMail.SmtpPort = 25;
        WebMail.EnableSsl = true;
        WebMail.UserName = "your-user-name-here";
        WebMail.From = "your-email-address-here";
        WebMail.Password = "your-account-password";

        // Create array containing file name
        var filesList = new string [] { fileAttachment };

        // Attach file and send email
        WebMail.Send(to: "target-email-address-here",
            subject: subjectLine,
            body: "File attached. <br />From: " + customerName,
            filesToAttach: filesList);
    }
    catch (Exception ex) {
        <text>
            <b>The email was <em>not</em> sent.</b>
            The code in the ProcessFile page must provide an
            SMTP server name, a user name, a password, and
            a "from" address.
        </text>
    }
}
<!DOCTYPE html>
<html>
<head>
    <title>Request for Assistance </title>
</head>
<body>
  <p><b>@customerName</b>, thank you for your interest.</p>

  <p>An email message has been sent to our customer service
  department with the <b>@fileAttachment</b> file attached.</p>

</body>
</html>
```

5. Módosítsuk a következő e-mailhez kapcsolódó beállításokat a példakódban:
   - Állítsuk be a `your-SMTP-host`-ot annak az SMTP szervernek a nevére, amihez hozzáférésünk van!
   - Állítsuk be a `your-user-name-here`-t az SMTP szerverünk felhasználónevére!
   - Állítsuk be a `your-email-address-here`-t a saját e-mail címünkre! Ez lesz az az e-mail cím, amiről az üzenetet elküldjük.
   - Állítsuk be a `your-account-password`-öt az SMTP szerver felhasználói fiókjának jelszavára!
   - Állítsuk be a `target-email-address-here`-t a saját címünkre! (Mint az előbb, most is el tudnánk küldeni a levelet valaki másnak, de tesztelésre saját magunknak küldjük el!)
6. Futtassuk a *SendFile.cshtml* oldalt a böngészőben!

7. Adjuk meg a nevünket, a tárgyat és a szöveges dokumentum nevét (*MyFile.txt),* hogy csatoljuk az e-mailhez!
8. Kattintsunk a *Submit* gombra! Mint korábban, most is át leszünk irányítva a *ProcessFile.cshtml* oldalra, ami megerősíti az üzenetet, és elküldi az levelet a csatolt fájllal.

További forrás angolul

- Simple Mail Tranfer Protocol

13.fejezet – Kereső hozzáadása a webhelyünkhöz

Ebben a fejezetben megtudhatjuk, hogyan keressünk egy weboldalt a Bing keresőmotor használatával.

Mit fogunk tanulni:

- Hogyan tegyük lehetővé, hogy webhelyünkön kereshessenek a felhasználók?

Ezt az ASP.NET funkciót ismertetjük a fejezetben:

- A `Bing` helper

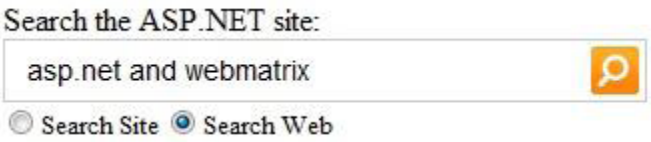
Keresés a webhelyünkön

Ha keresőt adunk a webhelyünkhöz, a találatokat megjeleníthetjük a webhely elhagyása nélkül is. A kereső hozzáadása a következő előnyökkel jár:

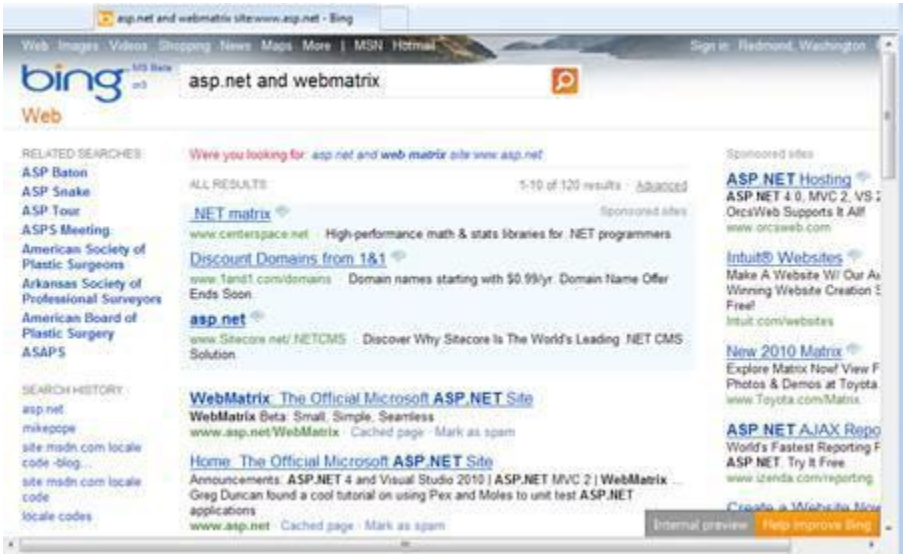
- Adjunk hozzá egy „Search this site” gombot, amivel a felhasználók az oldalon kereshetnek. Így biztosan megtalálják a webhelyünkön azt, amit szeretnének megnézni.
- Adjunk hozzá egy gombot, amivel a felhasználók egyszerűen kereshetnek kapcsolódó weboldalakon! Például ha a webhelyünk egy iskolai focicsapat oldala, akkor hozzáadhatunk egy, az iskola webhelyén kereső mezőt.
- Adjunk egy mezőt, amivel a felhasználó a weben kereshet az oldal elhagyása vagy új lap nyitása nélkül!

Hogy keresővel lássuk el wehelyünket, használjuk a `Bing` helpert és (opcionálisan) adjuk meg a kereséshez webhely URL-jét! A `Bing` helper elkészít egy szövegdobozt, amiben a felhasználó megadhatja a keresett kifejezést.

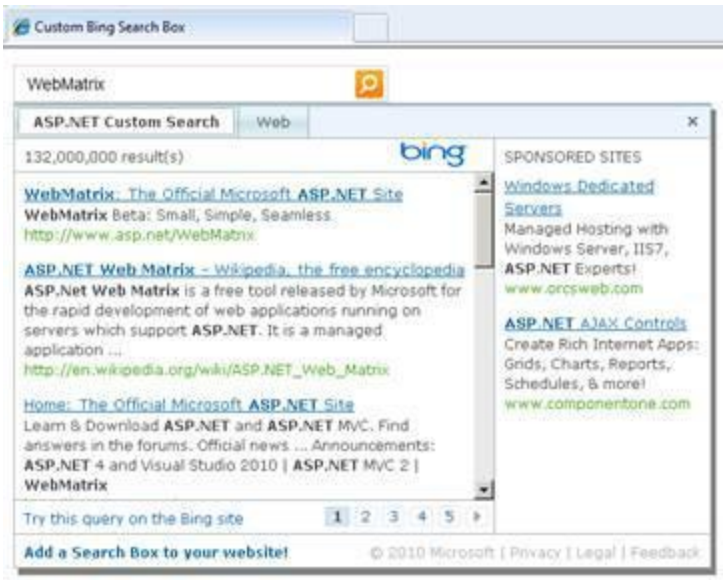
Két módja van a keresés beállításának: az „egyszerű” és a „haladó” lehetőség. Az egyszerű lehetőségben a helper készít egy szövegdobozt, ami tartalmazza a Bing kereső ikont, amire rákattintva kereshetünk:



Ha már megadtuk webhelyünk címét a kereséshez, a helper készít egy választógombot, amivel a felhasználó megmondhatja, hogy csak azon a lapon szeretne keresni, vagy az egész weben. Ha a felhasználó rákattint a keresésre, az egyszerű beállításnál a kereső csak átirányítja a keresést a Bing oldalára (http://www.bing.com/). A találatok egy új ablakban fognak megjelenni, mintha a felhasználó a Bing főoldalán keresett volna:



A haladó beállításoknál a kereződoboz rádiógombok nélkül kerül a lapra. Azonban ahelyett, hogy csak továbbirányítana a Bing oldalára, a helper a keresési találatokat egyből az oldalon jeleníti meg:



Részletesen beállíthatjuk a keresési találatok megjelenését. Ahogyan a példa is mutatja, ha már meghatároztunk egy oldalt a kereséshez, a találatok két fülön jelennek meg (találatok az oldalon, illetve a weben), így a felhasználó könnyedén tud váltani az oldalspecifikus és a webes keresés között.

Ebben a feladatban készítünk egy oldalt, ami bemutatja mindkét keresési opció használatát.

1. Készítsünk egy új weboldalt!
2. Adjunk hozzá egy *Search.cshtml* nevű oldalt, és írjuk meg a következő kódot:

```
@{
    Bing.SiteUrl = "www.asp.net";
    Bing.SiteTitle = "ASP.NET Custom Search";
}
<!DOCTYPE html>
<html>
    <head>
        <title>Custom Bing Search Box</title>
```

```
    </head>
    <body>

    <div>
       <h1>Simple Search</h1>
       <p>The simple option displays results by opening a new browser
window that shows the Bing home page.</p>
       Search the ASP.NET site: <br/>
       @Bing.SearchBox()
    </div>

    <div>
       <h1>Advanced Search Option</h1>
       <p>The advanced option shows search results directly in this
page. You can specify options to format the results.</p>
        Search the ASP.NET site: <br/>
        @Bing.AdvancedSearchBox(
            boxWidth: "300px",
            resultWidth: 600,
            resultHeight: 900,
            themeColor: "Green",
            locale: "en-US")
        </div>
    </body>
</html>
```

A kódban kétszer hívjuk meg a `Bing` helpert. Először a `SearchBox` metódust (az „egyszerű” beállítás), másodszorra az `AdvencedSearchBox` metódust (a „kiterjesztett” beállítást). Mindkettő metódusnál a `siteUrl` paramétert módosítsuk arra az oldalra, amin keresni szeretnénk! (Ha nem tesszük meg, a Bing a weben keres.) Ebben az esetben a `www.asp.net` weboldalon keresünk. Ha a saját oldalunkon szeretnénk keresni, helyettesítsük annak az URL-jével!

Az `AdvancedSearchBox` metódusban a `siteName` paramétert használjuk arra, hogy megmutassa a találatok panelen annak az oldalnak a nevét, amin keresünk. A kereső doboz átméretezéséhez a `boxWidth` paramétert használhatjuk. A találati panel méreteit a `resultWidth` és a `resultHeight` paraméterek tartalmazzák, a színének megadásához pedig állítsuk be a themeColor paramétert! A `locale` paraméter beállításával meg tudjuk határozni a nyelvet is, amit a helper a keresési találatokhoz és a grafikus felülethez használ.

> **Megjegyzés:** A hely utal az ország/régió és nyelvi beállításokra, amit a keresési felületen és a találatoknál használ. Ez nem jelenti azt, hogy csak a megadott nyelvű találatokat listázza a helper.

A helyek (locale) két részből állnak, mint például en-US (angol, amerikai), en-GB (angol, Nagy-Britannia), es-MX (spanyol, Mexikó), fr-Ca(francia,Kanada), és így tovább. A „local codes” kulcsszóra rákeresve találhatunk még több országkódot. A magyar területi beállítások paramétere hu.

3. Futtassuk a *Search.cshtml* oldalt a böngészőben! (Futtatás előtt bizonyosodjunk meg arról, hogy az oldal ki van választva a *Files* munkaterületen!)
4. Írjunk be egy keresendő kifejezést az egyszerű keresőmezőbe, és kattintsunk a  gombra! A találatok egy új böngészőablakban jelennek meg.
5. Most keressünk a haladó dobozzal! Egy panel fog megjelenni a keresés találataival.

Megjegyzés: Annak érdekében, hogy a `Bing` helper kereshessen, a keresett webhelyeknek nyilvánosan elérhetőnek kell lenniük, és annak tartalmát meg kell vizsgálnia („feltérképeznie”) a Bingnek. Ha „Keress ezen a lapon” gombot készítettünk, és beállítottuk a Bing helpert az oldalunkon való keresésre, addig nem tudjuk tesztelni, amíg az oldalunk elég idős nem lesz ahhoz, hogy a keresőmotorok megtalálják. Más szóval, nem tudjuk tesztelni a keresési képességet közvetlenül a WebMatrixból.

További források angolul

- Make your Website SEO Friendly
- Locale ID (LCID) Chart

14.fejezet – Közösségi hálózatok hozzáadása a weboldalunkhoz

Az egyik legjobb módja, hogy weboldalunkat még népszerűbbé és közvetlenebbé tegyük, a közösségi hálózatok szolgáltatásainak integrálása az oldalunkra. Ebben a fejezetben megtanuljuk, hogyan tudnak a webhelyünk látogatói hivatkozni oldalainkra a Facebook vagy a Digg hálózatában, hogyan adjunk Twitter hírfolyamokat az oldalhoz, és hogyan színesítsük webhelyünket Gravatar képekkel és Xbox Gamer kártyákkal.

A fejezetből megtudhatjuk:

- Hogyan hivatkozhatnak a látogatók a webhelyünkre?
- Hogyan adjunk hozzá Twitter hírfolyamot?
- Hogyan jelenítsünk meg Gravatar.com képeket?
- Hogyan rakjunk Xbox Gamer kártyákat az oldalakra?
- Hogyan adjunk az oldalhoz Facebook-os *Like* gombot?

A következő ASP.NET programozási koncepciókat ismertetjük a fejezetben:

- A `LinkShare` helper.
- A `Twitter` helper.
- A `Gravatar` helper.
- A `GamerCard` helper.
- A `Facebook` helper.

Weboldalunk linkelése közösségi oldalakon

Ha egy ember kedvel valamit az oldalunkon, gyakran meg szeretné osztani a barátaival. Ezt a műveletet egyszerűvé tehetjük azzal, hogy olyan ikonokat jelenítünk meg, amire rákattintva megoszthatja az oldalt a Diggen, Facebookon, Twitteren vagy hasonló oldalakon. Ezen ikonok megjelenítéséhez adjuk hozzá a `LinkShare` helpert az oldalunkhoz! A látogatók egyéni ikonokra kattinthatnak. Ha van regisztrált felhasználójuk a közösségi oldalhoz, akkor akkor közzétehetik oldalunk linkjét az adott hálózatban.

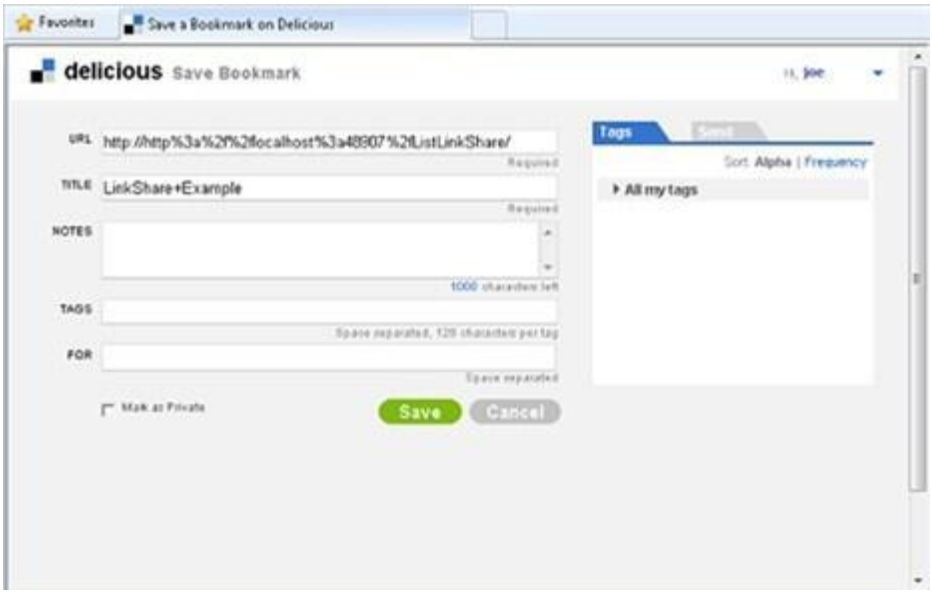
1. Készítsünk egy *ListLinkShare.cshtml* nevű lapot, és adjuk hozzá a következő kódot:

```
<!DOCTYPE html>
<html>
  <head>
    <title>LinkShare Example</title>
  </head>
  <body>
    <h1>LinkShare Example</h1>
```

```
    Share: @LinkShare.GetHtml("LinkShare Example")
  </body>
</html>
```

Ebben a példában, ha a `LinkShare` helper fut, az oldal címét beilleszti paraméterként, amit a közösségi oldalra ki fog tenni. Ezenkívül bármilyen szöveget beilleszthetünk mellé.

2. Futtassuk a *ListLinkShare.cshtml* oldalt a böngészőben! (Futtatás előtt bizonyosodjunk meg arról, hogy az oldal ki van jelölve a *Files* munkaterületen!)
3. Kattintsunk az ikonok valamelyikére, ahol van regisztrációnk! A link átirányít minket arra az oldalra, ahol megoszthatjuk a linket a kiválasztott közösségi oldalon. Például ha rákattintunk a del.icio.us linkre, akkor eljutunk a `Save Bookmark` oldalra a Delicious webhelyen.



Twitter hírfolyam hozzáadása

ASP.NET előre megírt helperje segít Twitter hírfolyam hozzáadásában. Ha a `Twitter.Profile` metódust használjuk a kódban, akkor megjeleníthetjük a weboldalunkon a konkrét Twitter felhasználó bejegyzéseit. Ha a `Twitter.Search` metódust használjuk, megadhatunk Twitter keresést (szavakra, kifejezésekre vagy bármilyen kereshető szövegre), és megjeleníthetjük a találatokat az oldalunkon. Mindkét helpernek magassága, szélessége, stílusa állítható.

A Twitter információk nyilvánosak, tehát nincs szükség Twitter felhasználóra a Twitter helperek oldalunkon történő használatához.

A következő feladat bemutatja, hogyan készítsünk olyan weboldalt, ami mindkét helpert bemutatja.

1. Adjuk hozzá az ASP.NET Web Helpers Libraryt a weboldalunkhoz a 2. fejezetben leírtak szerint, ha még nem tettük volna meg!
2. Hozzunk létre egy új *Twitter.cshtml* nevű lapot a webhelyünkön!
3. Írjuk be a következő kódot:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Twitter Example</title>
  </head>
  <body>
    <table>
      <tr>
        <td>Twitter profile helper</td>
        <td>Twitter search helper</td>
      </tr>
      <tr>
        <td>@Twitter.Profile("<Insert User Name>")</td>
        <td>@Twitter.Search("<Insert search criteria here>")</td>
      </tr>
    </table>
  </body>
</html>
```

4. A `Twitter.Profile` kódrészben cseréljük ki a `<Insert User Name>` elemet a megjelenítendő Twitter felhasználónevünkkel!
5. A `Twitter.Search` kódrészlet `<Insert search criteria here>` részét cseréljük ki a keresendő szövegre!
6. Futtassuk a weboldalt a böngészőben!

Gravatar kép megjelenítése

A Gravatar („globálisan elismert profilkép”) egy kép, amit több weboldalon is használhatunk avatarként – vagyis a teljesen publikus profilképünk. Gravatar segítségével könnyen felismerhetők a felhasználók fórum hozzászólásaikban, blog kommentjeikben és így tovább. (Regisztrálhatjuk a saját Gravatarunkat a gravatar weboldalán http://www.gravatar.com/.) Ha meg szeretnénk jeleníteni az emberek neve vagy e-mail címe mellett a képüket is, akkor használjuk a Gravatar helpert!

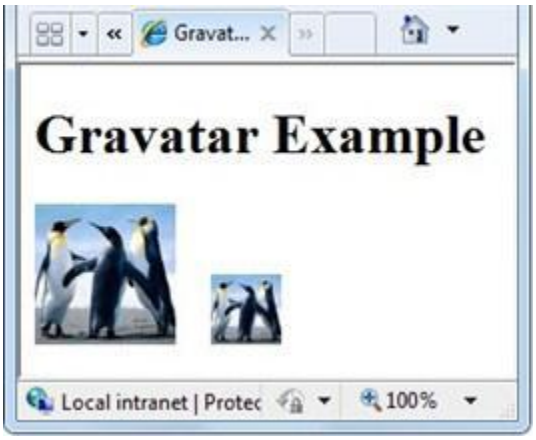
Ebben a példában egy egyszerű gravatart használunk, ami minket ábrázol. Egy másik módja a gravatar használatának az, hogy az emberek megadják gravatar címüket, amikor regisztrálnak webhelyünkön. (A 17. fejezetben megtudhatjuk, hogyan tudunk felhasználókat regisztrálni.) Amikor információkat jelenítünk meg a felhasználóról, egyszerűen megjeleníthetjük gravatarját a felhasználó neve mellett.

1. Készítsünk egy *Gravatar.cshtml* nevű weblapot!
2. Adjuk hozzá a következő kódot:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Gravatar Example</title>
    </head>
    <body>
        <h1>Gravatar Example</h1>
        @Gravatar.GetHtml("<Your Gravatar account here>")
           
        @Gravatar.GetHtml("<Your Gravatar account here>", 40)
    </body>
</html>
```

   A `Gravatar.GetHtml` metódus megjeleníti a Gravatar képet a lapon. A kép méretének megváltoztatásához meg tudunk adni egy számot második paraméterként. Az alapméret 80. A kisebb számok a kisebbre, nagyobb számok nagyobbra változtatják a képet.

3. A `Gravatar.GetHtml` metódusban cseréljük ki a `<Your Gravatar account here>` részt a Gravatar felhasználónk e-mail címére. (Ha nekünk nincs Gravatar accountunk, akkor használjuk valaki más e-mail címét, akinek van.)
4. Futtassuk a lapot a böngészőben! Két, az e-mail címünkhöz beállított Gravatar képet látunk az oldalon. A második kép az első egy kicsinyített változata.



Xbox Gamer kártya megjelenítése

Minden online Microsoft Xbox játékos saját azonosítóval játszik. Az egyes játékosok statisztikái, ezen kívül a játékeredményeik és a legutóbb játszott játékok is megjelennek személyes kártyájukon, az úgynevezett Gamer Cardokon. Ha Xbox játékosok vagyunk, meg tudjuk mutatni Gamer kártyánkat a weboldalunkon a `GamerCard` helper használatával.

1. Készítsünk egy új, *XboxGamer.cshtml* nevű oldalt és írjuk be a következő kódot!

```
<!DOCTYPE html>
<html>
  <head>
    <title>Xbox Gamer Card</title>
  </head>
  <body>
    <h1>Xbox Gamer Card</h1>
    @GamerCard.GetHtml("major nelson")
  </body>
</html>
```

   Használjuk a `GamerCard.GetHtml` tulajdonságot a játékos kártya tulajdonságainak beállításához!
2. Futtassuk az oldalt a böngészőben! Az oldalon megjelenik a beállított Gamer kártya.

Facebook „Like” gomb megjelenítése

Egyszerűen megoszthatunk tartalmakat a Facebook ismerőseinkkel a `Facebook` helper `LikeButton` metódusával.

Megjegyzés: A The Facebook helper a *Facebook.Helper* csomagban érhető el, amit külön kell telepítenünk a *Package Manager* eszköz használatával.

A Facebook helper készít egy *Like* gombot, ami számolja (a Facebookról olvassa), hogy mennyi ember kattintott már rá a *Like* (vagy *Tetszik)* gombra:

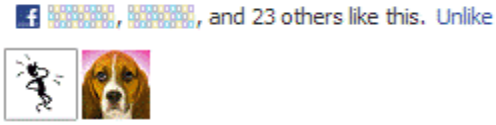
Amikor a felhasználó rákattint a *Like* gombra, egy link jelenik meg a Facebookos üzenőfalán, hogy tetszik neki az oldalunk.

Alapbeállításként a `Facebook` helper `LikeButton` metódusa létrehoz egy *Like* gombot, ami rámutat a jelenlegi oldalra. Ez a tipikus eset – a *Like* gomb megnyomásával arra az oldalra hivatkozunk, ahol éppen járunk. Lehetőségünk van azonban arra is, hogy a `Facebook` helper `LikeButton` metódusa egy általunk megadott linkre hivatkozzon. Ez akkor hasznos, ha több hivatkozás listáját szeretnénk megjeleníteni oldalunkon, és szeretnénk, hogy azokra közvetlenül hivatkozhassanak a *Like* gombbal

A `LikeButton` metódussal személyre szabható a *Like* gomb megjelenése, például:

- A link a *Like*, vagy *Recommend* linkre mutat.
- A *Like*-oló emberek számának megjelenítése az oldalon:

- A *Like* gombra kattintók profilképei:



- A *Like* gomb szélessége és színsémája (`light` vagy `dark`)

A következő példában készítünk két *Like* gombot! Egy a jelenlegi oldalra mutat, a másik pedig egy megadott URL-re (esetünkben az ASP.NET WebMatrix webhelyre). A példa teszteléséhez szükségünk van egy Facebook profilra.

1. Adjuk hozzá a *Facebook.Helper* könyvtárat a weboldalunkhoz a 2. fejezetben írtak szerint, ha még nem tettük meg!
2. Készítsünk egy új weblapot *FacebookLikeBtn.cshtml* néven, és adjuk meg a következő kódot:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Facebook 'Like' Button</title>
        <style>body {font-family:verdana;font-size:9pt;}</style>
    </head>
    <body>
    <p>Points to the current page, uses default settings:</p>
        @Facebook.LikeButton()

    <p>Points to the ASP.NET Web site:</p>
         @Facebook.LikeButton(
          href: "http://www.asp.net/webmatrix",
          action: "recommend",
          width: 250,
          buttonLayout: "button_count",
          showFaces: true,
          colorScheme: "dark")
    </body>
</html>
```

Az első `Facebook.LikeButton` metódus az alapértelmezett beállításokkal készült, tehát a jelenlegi oldalunkra mutat. A második példa tartalmaz néhány módosítást. Használjuk az `URL` paramétert a Like URL-jének módosításához! A *Like Recommend*-re módosításához állítsuk be az action paramétert „recommend”-re (az alapértelmezett „like” helyett). A gomb stílusának módosításához állítsuk a `layout` paramétert „button_count”-ra (a „standard” vagy „box_count” helyett)! A Facebook profilképek *Like* gomb alatti megjelenítéséhez állítsuk a showFaces paramétert igazra! Végül a színséma beállításához állítsuk a `colorScheme` paramétert „dark”-ra (alapértelmezetten „light”)!

3. Futtassuk a weblapot a böngészőnkben! Az oldalon láthatjuk a módosított Facebook *Like* gombot.

4. Kattintsunk a *Recommend* gombra, ami az ASP.NET weboldalára mutat! Ha nem vagyunk bejelentkezve, a Facebookra, kéri, hogy ezt pótoljuk. Ha beléptünk, láthatjuk a *Recommend* linket az üzenőfalunkon.
   Ha az oldal teszteléséhez a WebMatrixot használjuk, nem fogjuk tudni tesztelni az első linket (amelyik a jelenlegi oldalra mutat), mert a helyi számítógépen tárolt weboldalt a Facebook nem tudja elérni. Amint az oldalunk nyilvános lesz, a link működésbe lép.

15.fejezet – Forgalomanalízis

Miután üzembe helyeztük webhelyünket, érdemes lehet követni annak forgalmát.

A fejezetből megtudhatju:

- Hogyan küldjünk információkat a weboldalunkról egy analizáló szolgáltatóhoz?

A következő ASP.NET programozási funkciót ismerhetjük meg a fejezetben:

- Az `Analyitics` helper.

Látogatások információinak követése (Analízis)

Analízisnek nevezzük azt a folyamatot, melyben a weboldal forgalmát vizsgáljuk, így megállapítva, hogyan böngésznek a felhasználók webhelyünkön. Több szolgáltató kínál analíziseszközöket, ilyen a Google, a Yahoo, a StatCounter, és még sok más.

Oldalunk forgalmának vizsgálatához először regisztrálnunk kell valamelyik szolgáltatónál. A szolgáltató küld nekünk egy rövid JavaScript kódot, ami tartalmazza a felhasználói fiókunk azonosítóját. A követni kívánt weboldalhoz hozzáadjuk ezt a kódtöredéket. (Érdemes ezt a kódot a láblécbe, vagy a layout oldalba beilleszteni, vagy egy olyan HTML kódhoz adni, ami minden lapon megtalálható.) Amikor a felhasználó lekéri az oldalt, a szkript információkat küld a lapról az analizáló szolgáltatónak, ami különféle adatokat rögzít a lapról.

Amikor látni szeretnénk az oldalunk statisztikáit, bejelentkezünk az analizáló szolgáltató weboldalára. Többféle jelentést láthatunk az oldalról, például:

- Az egyes lapok nézettségét. Ez elárulja, (nagyjából) hány ember látogatta az oldalt, és melyik lapok a legnépszerűbbek az oldalon.
- Mennyi ideig maradtak az emberek az egyes lapokon? Ebből kiderül, mennyire kötöttük le a látogató figyelmét.
- Milyen oldalon voltak az emberek, mielőtt a miénkre látogattak? Ez segít kideríteni, a forgalom melyik linkekről, keresőkről jött.
- Mikor látogatták meg a honlapot, és meddig maradtak rajta?
- Melyik országokból látogattak a lapra?
- Milyen böngészőket és operációs rendszert használtak a látogatók?

Az ASP.NET több analizáló helpert tartalmaz (`Analytics.GetGoogleHtml`, `Analytics.GetYahooHtml` és `Analytics.GetStstCounterHtml`), amik egyszerűvé teszik az analizáláshoz használt JavaScript kódrészletek kezelését. Ahelyett, hogy kitaláljuk, hova és hogyan illesszük be a JavaScript kódot, mindössze annyit kell tennünk, hogy hozzáadjuk a helpert az oldalhoz. Az egyetlen szükséges információ a felhasználónevünk. (StatCounter-hez szükségünk lehet további értékek megadására is.)

Ebben az eljárásban készíteni fogunk egy layout oldalt, ami a `GetGoogleHtml` helpert használja. Ha már regisztráltunk egy másik szolgáltatónál, használhatjuk azt is.

Megjegyzés: Amikor elkészítjük az analízishez a felhasználónevet, regisztráljuk a webhelyünk Url-jét. Ha mindent a saját gépünkön tesztelünk, nem a valós forgalmat fogjuk analizálni, tehát nem tudunk adatokat rögzíteni és statisztikákat nézni. Ez az eljárás azonban megmutatja, hogyan adjunk analizáló helpert az oldalunkhoz. Amikor publikáljuk az oldalt, az élesben futó webhely már fog információkat küldeni az analizáló szolgáltatónak.

1. Készítsünk fiókot a Google Analyticsnél, és rögzítsük a felhasználónevet!
2. Készítsünk egy *Analytics.cshtml* layout lapot és írjuk be a következő kódot:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Analytics Test</title>
  </head>
  <body>
    <h1>Analytics Test Page</h1>
    <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit,
    sed do eiusmod tempor incididunt ut labore et dolore magna
aliqua. </p>
    <div id="footer">
      &copy; 2010 MySite
    </div>
    @Analytics.GetGoogleHtml("myaccount")
  </body>
</html>
```

Megjegyzés: Az Analytics helper meghívását muszáj a weblap body részében megtenni (a `</body>` jel előtt), különben a böngésző nem fogja futtatni a scriptet.

Ha más szolgáltatót használunk, a következő helpereket használjuk az előző helyett:

- (Yahoo) `@Analytics.GetYahooHtml("myaccount")`
- (StatCounter) `@Analytics.GetStatCounterHtml("project", "partition", "security")`

3. Cseréljük ki a `myaccount` szöveget az első lépésben elkészített fióknévvel.
4. Futtassuk a lapot a böngészőben! (A futtatás előtt bizonyosodjunk meg arról, hogy az oldal ki van választva a *Files* munkaterületen!)
5. A böngészőben nézzük meg az oldal forráskódját! Így láthatjuk az elkészített analizáló kódot:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Analytics Test</title>
  </head>
  <body>
    <h1>Analytics Test Page</h1>
    <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit,
    sed do eiusmod tempor incididunt ut labore et dolore magna
aliqua.</p>
    <div id="footer">
      &copy; 2010 MySite
    </div>
    <script type="text/javascript">
      var gaJsHost = (("https:" == document.location.protocol) ?
"https://ssl." : "http://www.");
      document.write(unescape("%3Cscript src='" + gaJsHost + "google-
analytics.com/ga.js' type='text/javascript'%3E%3C/script%3E"));
    </script>
    <script type="text/javascript">
      try{
      var pageTracker = _gat._getTracker("myaccount");
      pageTracker._trackPageview();
      } catch(err) {}
    </script>
  </body>
</html>
```

6. Lépjünk be a Google Analytics weboldalára, és vizsgáljuk meg az oldalunk statisztikáit! Ha a webhely már élesben fut, láthatjuk a bejegyzéseket a látogatásokról.

16.fejezet – Gyorsítótárazás a webhely teljesítményének növeléséhez

Fel tudjuk gyorsítani webhelyünket azzal, hogy eltároljuk azon adatok értékeit a gyorsítótárban (cacheben), melyek ritkán változnak, előállításuk pedig hosszú folyamat.

A fejezetből megtudhatjuk:

- Hogyan használjuk a cache-t az oldal sebességének növeléséhez?

Ezt az ASP.NET funkciókat ismerhetjük meg a fejezetben:

- A `WebCache` helper.

Cache-elés a weboldal sebességének növeléséhez

Minden alkalommal, amikor valaki lekér egy oldalt a webhelyünkről, a webszervernek van némi dolga a kérés teljesítéséhez. Néhány oldalhoz a szervernek (viszonylag) időigényes feladatokat kell elvégeznie, például adatbázisokból kell lekérdezéseket végrehajtania. Még ha egy-egy ilyen feladat nem is tart sokáig, nagy forgalmú webhely esetén a rengeteg egyedi kérés kihathat az egész webhely teljesítményére.

A webhely teljesítménye javítható a gyorsítótár használatával. Ha ugyanazt az adatot többen lekérik, és azt nem kell változtatni a felhasználók és az idő függvényében, elég csak egyszer legenerálni, majd eltárolni a cache-ben. Ha valaki újra lekéri az adott információt, elég csak a gyorsítótárból újraküldeni azt.

A ritkán változó információkat általában cache-eljük. A gyorsítótárazott információt eltároljuk a webszerver memóriájában. A tárolás ideje másodpercektől napokig állítható. Amikor az adott idő letelt, az információ automatikusan törlődik a cacheből.

Megjegyzés: A gyorsító tárazott bejegyzések nem csak az idő lejárta miatt törlődhetnek, hanem akkor is, ha például a szervernek kevés szabad memóriája marad, melyet a cache törlésével tud orvosolni.

Képzeljük el, hogy a weboldalunkon megjelenítjük a jelenlegi hőmérsékletet és időjárást. Ezen típusú információkért valószínűleg egy külső szolgáltató felé küldünk kérést. Mivel ezek az információk nem változnak gyakran (például két órás időtartamon belül) és a külső hívások sávszélességet és időt igényelnek, érdemes ezt az információt gyorsítótárazni.

ASP.Net tartalmazza a WebCache helpert, ami egyszerűvé teszi a gyorsító tár hozzáadását az oldalhoz, és az adatok cache-elését. Ebben az eljárásban készítünk egy oldalt, ami gyorsító tárazza a jelenlegi időt. Ez nem egy való világban működő példa, hiszen gyakran változik és nem bonyolult a kiszámítása. Azonban egy jó módja, hogy bemutassuk a gyorsító tárazást működés közben.

1. Hozzunk létre egy *WebCache.cshtml* nevű lapot a webhelyünkön.
2. Írjuk meg a következő kódot:

```
@{
    var cacheItemKey = "Time";
    var cacheHit = true;
    var time = WebCache.Get(cacheItemKey);

    if (time == null) {
        cacheHit = false;
    }

    if (cacheHit == false) {
```

```
        time = @DateTime.Now;
        WebCache.Set(cacheItemKey, time, 1, false);
    }
}
<!DOCTYPE html>
<html>
<head>
    <title>WebCache Helper Sample</title>
</head>
<body>
    <div>
        @if (cacheHit) {
            @:Found the time data in the cache.
        } else {
            @:Did not find the time data in the cache.
        }
    </div>
    <div>
        This page was cached at @time.
    </div>
</body>
</html>
```

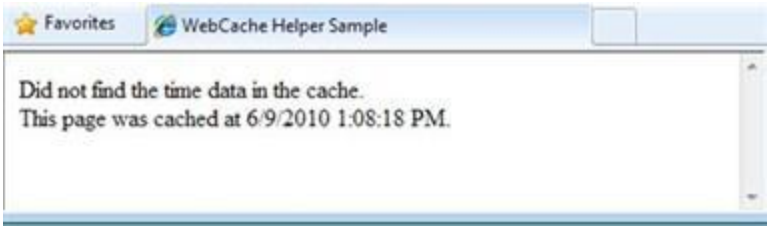
Amikor adatokat cache-elünk, egy, a webhelyen egyedülálló név használatával tesszük be a gyorsító tárba. Ebben az esetben mi a `Time` nevű cache bejegyzést használjuk. Ezt a `cacheItemKey` mutatja a példában.

A kód először a `Time` cache bejegyzést olvassa. Ha egy értéket ad vissza (tehát a cache értéke nem nulla), a kód csak beállítja a `time` változót a cache-ből kiolvasott értékre.

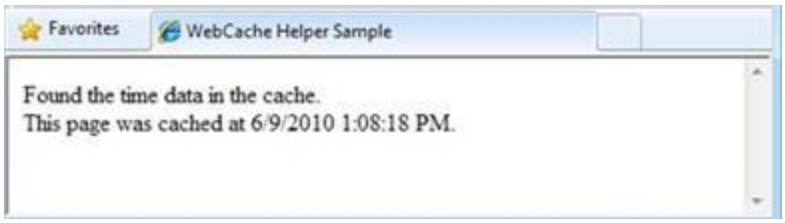
Azonban, ha a cache bejegyzés nem ad vissza értéket (tehát nulla), a kód beállítja az idő értéket, hozzáadja a gyorsítótárhoz és beállítja a lejárati időt egy percre. Ha az oldalt nem kérik egy percen belül, a cache bejegyzés törlődik. (A gyorsítótárazott elemek alapértelmezett lejárati ideje 20 perc.)

Ez a kód a mindig követendő példát mutatja, ha adatokat gyorsítótárazunk. Mielőtt valamit beteszünk a cache-be, mindig ellenőrizzük, hogy a WebCache.Get metódus nulla értéket ad-e. Tartsuk szem előtt, hogy a gyorsítótár bejegyzés lejárhat, vagy bármilyen más okból eltávolításra kerülhet, tehát nem garantálható, hogy minden bejegyzés a cacheben van.

3. Futtassuk a *WebCache.cshtml* oldalt a böngészőben. (A futtatás előtt bizonyosodjunk meg arról, hogy az oldal ki van jelölve a *Files* munkaterületen.) Az oldal első lekérésénél nincsenek adatok a gyorsítótárban, ezért a kód hozzáadja a `time` értéket a cache-hez.



4. Frissítsük a *WebCache.cshtml* oldalt a böngészőben. Ezúttal az idő értéke már gyorsítótárazva van. Vegyük észre, hogy az idő nem változott az előző látogatásunk óta.



5. Várjunk egy percet a cache kiürüléséig, és utána frissítsük az oldalt. Az oldal megint jelzi, hogy a `time` értékét nem találja a gyorsítótárban, és a frissített időt adja a cache-hez.

17.fejezet – Biztonsági elemek és felhasználói fiókok hozzáadása

Ebben a fejezetben megtanulhatjuk, hogyan korlátozhatjuk a hozzáférést webhelyünkhöz úgy, hogy egyes oldalak csak bejelentkezett felhasználók számára legyenek elérhetőek. (Azt is megtudhatjuk, hogyan készítsünk olyan oldalt, amihez akárki hozzáférhet.)

Az útmutatóból megtudhatjuk:

- Hogyan hozzunk létre olyan weboldalt, amelyen regisztrációs és bejelentkező oldal van, így korlátozhatjuk az egyes oldalakhoz való hozzáférést?
- Hogyan hozzunk létre nyilvános vagy privát oldalakat?
- Hogyan használjuk a CAPTCHA-t, hogy megakadályozzuk az automatizált programokat (botokat) a felhasználói fiókok létrehozásában?

Ebben a fejezetben bemutatott ASP.NET funkciók:

- A `WebSecurityk` helper.
- A `SimpleMembership` helper.
- A `ReCaptcha` helper.

Bevezetés a felhasználói fiókokba

A weboldalunkat beállíthatjuk úgy, hogy a felhasználók be tudnak jelentkezni – tehát az oldal támogatja a felhasználói fiókokat. Ez több okból is hasznos lehet, például az oldalunkon lehetnek olyan funkciók, amiket csak a regisztrált felhasználók érhetnek el . Egyes esetekben szükség van a felhasználók azonosítására, hogy visszajelzést küldjenek vagy hozzászólhassanak valamihez.

Annak ellenére, hogy a weboldal támogatja a felhasználókat, nem feltétlenül szükséges, hogy belépjenek, mielőtt használni tudják az oldal egyes részeit. A nem bejelentkezett felhasználók *anonymus* (ismeretlen) felhasználóként jelennek meg.

A felhasználók regisztrálhatnak, majd bejelentkezhetnek a weboldalra. A weboldalaknak szükségük van egy felhasználónévre (általában egy e-mail cím) és egy jelszóra, hogy megerősítést kapjunk arról, hogy a felhasználó az, akinek állítja magát. A bejelentkezés és a megerősítés folyamatát *azonosításnak* nevezzük.

A WebMatrixban használhatjuk a Starter Site sablont, hogy létrehozzunk egy weboldalt, ami a következőket tartalmazza:

- Egy adatbázist, ami a felhasználói fiókok felhasználónevét és jelszavait tárolja.
- Egy regisztrációs oldalt, ahol az új felhasználók regisztrálhatnak.
- Bejelentkező és kijelentkező oldalt.
- Egy jelszó visszanyerő és visszaállító oldalt.

Megjegyzés: Annak ellenére, hogy a Start Site minta automatikusan létrehozza önnek ezeket az oldalakat, ebben a fejezetben mi egy egyszerűsített változatot fogunk létrehozni annak érdekében, hogy az ASP.NET biztonsági és felhasználói beállításainak alapjait megismerhessük.

Weboldal létrehozása regisztrációs és bejelentkező oldalakkal

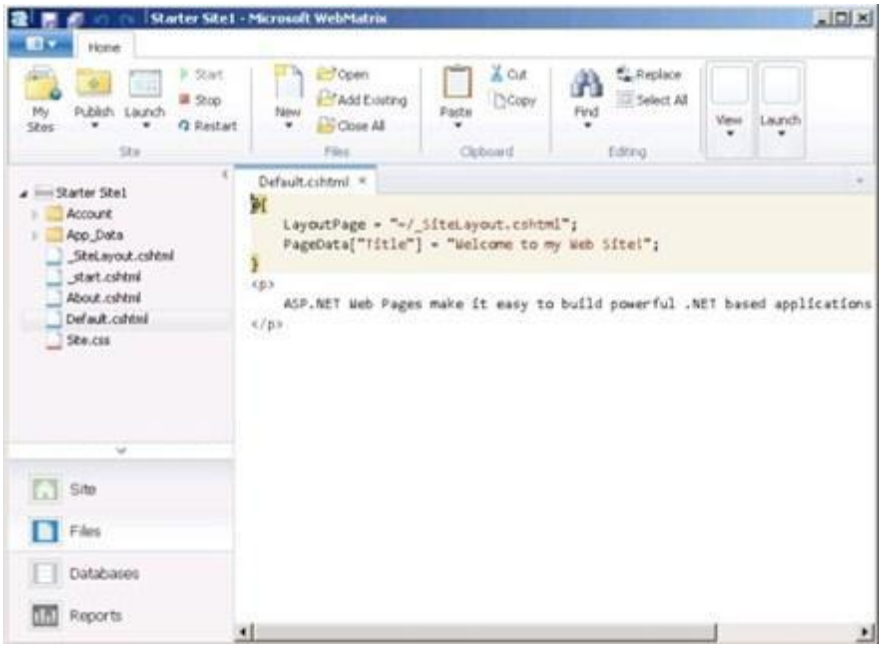
1. Indítsuk el a WebMatrixot!
2. A Quick Start oldalon válasszuk a *Site From Tamplate*-et!
3. Válasszuk a *Starter Site*-ot, majd kattintsunk a *OK* gombra!

4. A baloldali panelon kattintsunk a *Files workspace* választógombra!
5. A weboldal gyökérmappájában nyissuk meg az *_AppStart.cshtml* fájlt - ez egy különleges fájl, ami általában a teljes weboldalra vonatkozó beállításokat tartalmazza. Tartalmaz néhány parancsot, amik a // karakterekkel megjegyzésként viselkednek (így nem futnak le):

```
@{
   WebSecurity.InitializeDatabaseConnection("StarterSite",
"UserProfile", "UserId",
         "Email", true);
    // WebMail.SmtpServer = "mailserver.example.com";
    // WebMail.EnableSsl = true;
    // WebMail.UserName = "username@example.com";
    // WebMail.Password = "your-password";
    // WebMail.From = "your-name-here@example.com";
}
```

   Hogy e-mailt tudjunk küldeni, használhatjuk a `WebMail` helpert. Ehhez viszont szükség lesz egy SMTP szerverhez való hozzáférésre, ahogyan azt a 12. fejezetben megtanulhattuk. Abban a fejezetben megtudhattuk, hogyan állítsunk be számos SMTP beállítást egy oldalon. Ebben a fejezetben ugyanezeket fogjuk használni, de most egy központi fájlban fogjuk a beállításokat tárolni, így nem kell minden egyes oldalon külön elvégeznünk azokat. (Egy regisztrációs adatbázis felállításához nem kell feltétlenül SMTP szervert konfigurálnunk, erre csak akkor van szükség, ha megerősítő e-mailt szeretnénk küldeni a regisztrációhoz, illetve e-mailben szeretnénk elküldeni az elfelejtetett jelszavakat.

6. Távolítsuk el a // karaktereket mindegyik megjegyzés elejéről
7. Módosítjuk a következő e-mail-lel kapcsolatos beállításokat a kódban:
   - A `WebMail.SmtpServer` a rendelkezésre álló SMTP szerver nevére állítsuk be!
   - A `WebMail.EnableSsl`-t hagyjuk `true` beállításon! Ez a beállítás teszi biztonságosság az SMTP szervernek küldött azonosító adatokat azáltal, hogy kódolja őket.
   - A `WebMail.UserName`-et állítsuk be az SMTP szerverünkhöz tartozó felhasználónevünkre!
   - A `WebMail.Password`-t az SMTP szerverünkhöz tartozó jelszóra állítsuk be!
   - A `WebMail.From`-ot állítsuk a saját e-mail címünkre! Ez az e-mail cím az, amiről elküldjük az levelet!
8. Mentsük és zárjuk be az _AppStart.cshtml-t!
9. Nyissuk meg a Default.cshtml fájlt!



10. Futtassuk a *Default.cshtml* oldalt egy böngészöben

11. Az oldal jobb felső sarkában kattintsunk a *Register* linkre!
12. Írjunk be egy felhasználónevet és egy jelszót, majd kattintsunk a *Register* gombra!

Amikor létrehoztuk a weboldalt a *Starter Site* sablonból, egy *StarterSite.sdf* nevű adatbázis is létrejött az oldal *App_Data* mappájában. A regisztráció során a felhasználó adatai hozzáadódtak az adatbázishoz. A regisztráció befejezéséhez a rendszer elküld egy levelet a megadott e-mail címre.

13. Nyissuk meg a levezőprogramunkat és keressük meg a levelet, ami tartalmazza a megerősítő kódot, és egy hiperhivatkozást az oldalra!
14. Kattintsunk a hiperhivatkozásra, hogy aktiváljuk a felhasználói fiókunkat! A hivatkozás egy regisztrációt megerősítő oldalt nyit meg.

15. Kattintsunk a *Login* linkre, majd lépjünk be a regisztrált felhasználónévve!
    Miután beléptünk, a *Login* és *Register* linkek helyett a *Logout* link jelenik meg.

16. Kattintsunk az *About* linkre!
    Az *About.cshtml* oldal jelenik meg. Most, hogy bejelentkeztünk, az oldal tetején megjelent a Welcome Joe üzenet és a Logout hivatkozás.

Megjegyzés: Alapbeállításokkal az ASP.NET weboldalak egyszerű szövegként (emberek számára olvasható szöveg) küldik el a szervernek a hitelesítő adatokat. Egy működő oldalhoz biztonságos HTTP-t (https://, más néven secure sickets layer vagy SSL) érdemes használni azért, hogy a szerverrel közölt bizalmas információkat kódolja. A bizalmas információkat kódolhatjuk a `WebMail.EnableSsl=true` beállításával, ahogy az előző példában. Az SSL-lel kapcsolatban további információért olvassuk el a következő angol nyelvű oldalt: How To: Protect Forms Authentication in ASP.NET 2.0.

Csak felhasználók számára elérhető oldal létrehozása

Kezdetben akárki böngészheti a webhely akármelyik oldalát. Lehetséges, hogy szeretnénk olyan oldalakat létrehozni, melyeket csak a bejelentkezett falhasználók (tagok) érhetnek el. Az ASP.NET lehetővé teszi, hogy úgy konfiguráljuk a weboldalakat, hogy csak a bejelentkezett felhasználók érhetik el. Általában ha ismeretlen felhasználók próbálják elérni az oldalt, továbbítjuk őket a bejelentkező oldalra.

Ebben az eljárásban korlátozni fogjuk az About oldal (*About.cshtml*) hozzáférését, így csak bejelentkezett felhasználók érhetik el.

1. Nyissuk meg az *About.cshtml* fájlt! Ez egy tartalmi oldal, ami a *_SiteLayout.cshtml* layout oldal szerint jelenik meg. (Többet az elrendezési oldalakról a 4. fejezet – Egységes megjelenés kialakítása fejezetben tudhatunk meg)
2. Az About.cshtml oldalban lévő kódot cseréljük le a következővel:

```
@if (!WebSecurity.IsAuthenticated) {
      Response.Redirect("/Account/Login");
 }

@{
    Layout = "/_SiteLayout.cshtml";
    Page.Title = "About My Site";
}

<p>
This web page was built using ASP.NET Web Pages. For more
information,
visit the ASP.NET home page at <a href="http://www.asp.net"
target="_blank">http://www.asp.net</a>
</p>
```

   Ez a kód teszteli a `WebSecurity` objektum `IsAuthenticated` tulajdonságát, ami igaz választ ad vissza, ha a felhasználó be van jelentkezve. Ellenkező esetben a kód meghívja a `Respone.Redict`-et és a felhasználót az *Account* mappában lévő *Login.cshtml* oldalra küldi.

Megjegyzés: A példában szereplő URL-ek (mint az ~/Account/Login) nem tartalmazzák a *.cshtml* kiterjesztést. Az ASP.NET környezetben nem szükséges a kiterjesztés azokban az URL-ekben, amik *.cshtml* oldalra mutatnak. További információért lásd: 19. fejezet – A Site-Wide viselkedés testreszabása.

3. Futtassuk a *Default.cshtml*-t egy böngészőben! Ha be vagyunk jelentkezve, kattintsunk a *Logout* linkre!
4. Kattintsunk az *About* linkre! Most továbbított minket a *Login.cshtml* oldalra, mert nem vagyunk bejelentkezve.

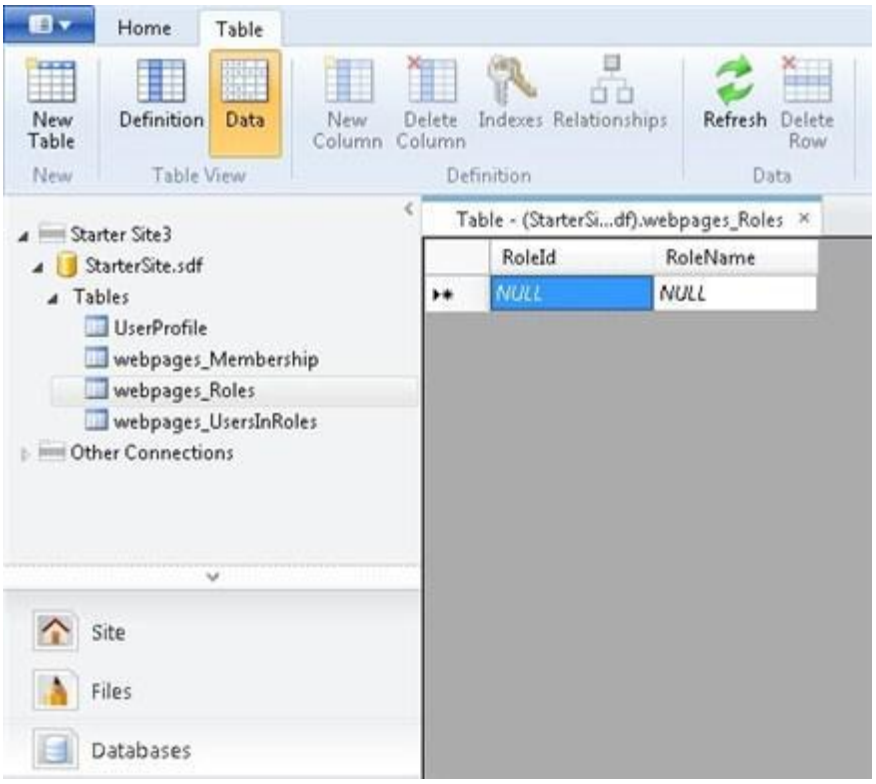
   Ahhoz, hogy több oldal hozzáférését is korlátozzuk, hozzáadhatjuk minden egyes oldalhoz a biztonsági ellenőrzést, vagy létrehozhatunk egy *_SiteLayout*-hoz hasonló minta oldalt, ami tartalmazza a biztonsági ellenőrzést. Így a biztonsági ellenőrzéssel ellátott layout oldalra hivatkoznánk a többi oldalról ugyanúgy, ahogy most a *Default.cshtml* hivatkozik a *_SiteLayout*-ra.

Csoportok biztonsági beállításai (szerepek)

Ha sok tagja van az oldalunknak, kényelmetlen minden egyes felhasználó engedélyeit külön ellenőrizni, mielőtt megnézhetnek egy oldalt. Helyette létrehozhatunk csoportokat vagy szerepeket (roles), amik az egyes felhasználókhoz tartoznak. Ezek után ellenőrizhetjük a jogosultságokat a szerepek alapján. Ebben a fejezetben meg fogjuk tanulni, hogyan készítsünk egy „rendszergazda” szerepet, és hogyan hozzunk létre olyan oldalt, amihez csak ebben a szerepben (csoportban) lévő felhasználók férnek hozzá.
Kezdésként hozzá kell adnunk a csoport információit a felhasználói fiókok adatbázisához.

1. A WebMatrixban kattintsunk a *Databases* workspace seletor-ra!

   A baloldali panelen nyissuk ki a *SarterSite.sdf*-ot, azon belül a *Tables*-t, majd dupla kattintással nyissuk meg a *webpages_Role* táblát!



2. Adjunk hozzá egy „admin” szerepet! A *RoleId* mező automatikusan kitöltődik. (Ez az elsődleges kulcs, és egy azonosító mezőnek lett beállítva, ahogya a 6. fejezetben tanultuk.) Figyeljük meg, hogy mi a *RoleId* mező értéke! (Ha ez az első, amit megadtunk, akkor 1 lesz.)

Table - (StarterSi...df).webpages_Roles

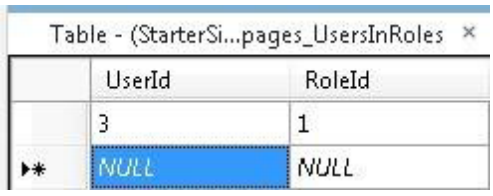
	RoleId	RoleName
	1	admin
▶*	NULL	NULL

3. Zárjuk be a *webpages_Roles* táblát!
4. Nyissuk meg a UserProfiles táblát!
5. Figyeljük meg, hogy egy vagy több felhasználó UserId értéke már a táblában van, majd zárjuk be a táblát!

Table - (StarterSite.sdf).UserProfile

	Bio	DisplayName	Email	UserId
▶	NULL	Jim	jim1@adventur...	2
	NULL	Keith	keith0@advent...	3
*				

6. Nyissuk meg a *webpages_UserInRoles* táblát, és írjunk be a táblába egy *UserID* és egy *RoleID* értéket! Például ahhoz, hogy a 3-as felhasználót („Keith” a fenti példában) „admin” szerepbe tegyük, a következő adatokat adjuk meg:



7. Zárjuk be a *webpages_UserInRoles* táblát!
   Most, hogy meghatároztunk egy szerepet, beállíthatunk egy oldalt, amihez csak azok a felhasználók férhetnek hozzá, akik a megfelelő szerepben vannak.
8. A webhely gyökérmappájában hozzunk létre egy *AdminError.cshtml* nevű oldalt és cseréljük ki a tartalmát a következőkre! (Azok a felhasználók, akiknek nincs joga az oldal megnyitásához, ide kerülnek).

```
@{
    Layout = "/_SiteLayout.cshtml";
    PageData["Title"] = "Admin-only Error";
}
<p>You must log in as an admin to access that page.</p>
```

9. A weboldal gyökérmappájában hozzunk létre egy AdminOnly.cshtml nevű oldalt, és írjuk be a következő kódot:

```
@{
    Layout = "/_SiteLayout.cshtml";
    PageData["Title"] = "Administrators only";
}

@if ( Roles.IsUserInRole("admin")) {
    <span> Welcome <b>@WebSecurity.CurrentUserName</b>! </span>
}
else {
     Response.Redirect("/AdminError");
}
```

   A Roles.IsUserInRole metódus igaz értéket ad vissza, ha a jelenlegi felhasználó az admin csoport tagja.

10. Nyissuk meg a Default.cshtml-t egy böngészőben, de ne lépjünk be! (Ha be vagyunk jelentkezve, lépjünk ki!)
11. A böngésző címsorában az URL-ben változtassuk meg a „*Default*”-ot „*AdminOnly*”-ra. (Más szóval töltsük be: az *AdminOnly.cshtml* fájlt!) Átirányít minket az *AdminError.cshtml* oldalra, mert jelenleg nem admin jogú felhasználóval vagyunk belépve.
12. Lépjünk vissza a *Default.cshtml* oldalra, és lépjünk be egy admin szerepű felhasználóval!
13. Nyissuk meg az *AdminOnly.cshtml* oldalt! Ebben az esetben látni fogjuk az oldalt.

Jelszót megváltoztató oldal létrehozása

Lehetővé tehetjük a felhasználók számára jelszavaik megváltoztatását egy jelszó-megváltoztató oldal létrehozásával. Ebben a példában csak alapszinten foglalkozunk a problémával. (A Starter Site sablonban található *ChangePassword.cshtml* fájl kifinomultabb hibaszűréssel rendelkezik.)

1. A weboldal *Account* mappájában hozzunk létre egy *ChangePassword2.cshtml-t!*
2. A tartalmát cseréljük le a következőkre:

```
@{
   Layout = "/_SiteLayout.cshtml";
   PageData["Title"] = "Change Password";

   var message = "";
   if(IsPost) {

        string username = Request["username"];
        string newPassword = Request["newPassword"];
        string oldPassword = Request["oldPassword"];

        if(WebSecurity.ChangePassword(username, oldPassword,
newPassword)) {
            message="Password changed successfully!";
        }
        else
       {
           message="Password could not be changed.";
       }
    }
}
<style>
        .message  {font-weight:bold; color:red; margin:10px;}
</style>
<form method="post" action="">
  Username: <input type="text" name="username"
    value="@WebSecurity.CurrentUserName" />
  <br/>
  Old Password: <input type="password" name="oldPassword" value="" />
  <br/>
  New Password: <input type="password" name="newPassword" value="" />
  <br/><br/>
  <input type="submit" value="Change Password" />
  <div class="message">@message</div>
  <div><a href="Default.cshtml">Return to home page</a></div>
</form>
```

Ez az oldal olyan szövegdobozokat tartalmaz, amik lehetővé teszik a felhasználónak, hogy beírja a felhasználónevét, jelszavát és az új jelszavát. Ebben a kódban meghívjuk a `WebSecurity` helper `ChangePassword` metódusát, és a felhasználótól kapott adatokat közvetítjük neki.

3. Nyissuk meg az oldalt egy böngészőben. Ha már be vagyunk jelentkezve, a felhasználónevünk megjelenik az oldalon.

   Próbáljuk ki, hogy rosszul adjuk meg a régi jelszavunkat! Ha rossz jelszót adunk meg, a `WebSecurity.ChangePassword` meghívása sikertelen, és megjelenik egy üzenet.

4. Adjuk meg az érvényes adatokat, és próbáljuk ismét megváltoztatni a jelszavunkat.

Új jelszó létrehozásának lehetősége

Ha a felhasználók elfelejtik a jelszavukat, lehetővé tehetjük, hogy egy újat készítsenek. (Itt nem kell megadni a régi jelszót.) Ahhoz, hogy a felhasználó egy új jelszót kapjon, a `WebSecurity` helper `GeneratePasswordResetToken` folyamatát használjuk, és ez hoz létre egy tokent. (A token egy kriptográfiailag biztonságos string, amit elküldünk a felhasználónak, és egyedi azonosítóként szolgál olyan célokra, mint a jelszó visszaállítása.) Ezt az eljárás gyakran használják a gyakorlatban – elkészítjük a kulcsot, elküldjük a felhasználónak, majd hivatkozunk arra az oldalra, amely kiolvassa a tokent, és lehetővé teszi az új jelszó megadását. A felhasználó postaládájába hasonló link érkezik:

http://localhost:36916/Account/PasswordReset2?PasswordResetToken=08HZGH0ALZ3CGz3

A link végén található véletlenszerű karakterek sorozata a token. (A Starter Site sablon *ForgotPassword.cshtml* fájlja ennél némileg komplexebb.)

1. A webhely *Account* mappájában készítsünk egy új *ForgotPassword2.cshtml* nevű oldalt!
2. A tartalmát cseréljük le a következőkre:

```
@{
  Layout = "/_SiteLayout.cshtml";
  PageData["Title"] = "Forgot your password?";

  var message = "";
  var username = "";

  if (WebMail.SmtpServer.IsEmpty() ){
    // The default SMTP configuration occurs in _start.cshtml
    message = "Please configure the SMTP server.";
   }
```

```
  if(IsPost) {
    username = Request["username"];
    var resetToken =
WebSecurity.GeneratePasswordResetToken(username);

    var portPart = ":" + Request.Url.Port;
    var confirmationUrl = Request.Url.Scheme
      + "://"
      + Request.Url.Host
      + portPart
      +
VirtualPathUtility.ToAbsolute("/Account/PasswordReset2?PasswordResetT
oken="
      + Server.UrlEncode(resetToken));

    WebMail.Send(
      to: username,
      subject: "Password Reset",
      body: @"Your reset token is:<br/><br/>"
        + resetToken
        + @"<br/><br/>Visit <a href="""
        + confirmationUrl
        + @""">"
        + confirmationUrl
        + @"</a> to activate the new password."
    );

    message = "An email has been sent to " + username
        + " with a password reset link.";
  }
}
<style>
    .message  {font-weight:bold; color:red; margin:10px;}
</style>
<form method="post" action="">

  @if(!message.IsEmpty()) {
    <div class="error">@message</div>
  } else{
  <div>
  Enter your email address: <input type="text" name="username" />
<br/>
  <br/><br/>
  <input type="submit" value="Get New Password" />
  </div>
  }
</form>
```

Az oldal tartalmazza a szövegdobozt, amibe a felhasználó e-mail címét kell beírni. Ha a felhasználó elküldi az űrlapot, először meg kell győződnünk, hogy az SMTP beállításokat megtettük, hiszen az oldal célja, hogy küldjön egy e-mail üzenetet.

Az oldal lényege a jelszó-visszaállító token, amit így készítünk, és a felhasználó által megadott e-mail címre (username) elküldjük.

```
string resetToken = WebSecurity.GeneratePasswordResetToken(username);
```

A kód további része az e-mail elküldésére szolgál. Nagy része a *Register.cshtml* fájlból származik, amely a sablon kiválasztásakor jött létre.

Az e-mailt a `WebMail` helper `Send` parancsával küldjük el. Az e-mail törzse szövegek és HTML elemeket tartalmazó változók és stringek összefűzésével jön létre. Amikor a felhasználó megkapja, az e-mail törzse körülbelül így fog kinézni:

3. Az *Account* mappában hozzunk létre még egy új oldalt *PasswordReset2.cshtml* néven, és a tartalmát cseréljük le a következőre:

```
@{
  Layout  = "/_SiteLayout.cshtml";
  PageData["Title"] = "Password Reset";

  var message = "";
  var passwordResetToken = "";

  if(IsPost) {
    var newPassword = Request["newPassword"];
    var confirmPassword = Request["confirmPassword"];
    passwordResetToken = Request["passwordResetToken"];

    if( !newPassword.IsEmpty() &&
       newPassword == confirmPassword &&
       WebSecurity.ResetPassword(passwordResetToken, newPassword)) {
          message = "Password changed!";
    }
    else {
      message = "Password could not be reset.";
    }
  }
}
<style>
    .message  {font-weight:bold; color:red; margin:10px;}
</style>
<div class="message">@message</div>
<form method="post" action="">
  Enter your new password: <input type="password" name="newPassword"
/> <br/>
  Confirm new password:    <input type="password"
name="confirmPassword" /><br/>
  <br/>
  <input type="submit" value="Submit"/>
</form>
```

Ez az oldal nyílik meg, ha a felhasználó az e-mailben lévő linkre kattint, hogy megváltoztassa a jelszavát. A törzs tartalmaz szövegdobozokat, hogy a felhasználó megadhassa a jelszavát és megerősítse azt.

A jelszó tokent az URL-ből tudhatjuk meg a [„PasswordResetToken”] olvasásával. Emlékezzünk, hogy az URL valahogy így fog kinézni!

http://localhost:36916/Account/PasswordReset2?PasswordResetToken=08HZGH0ALZ3CGz3

Amint megvan a token, meghívjuk a `WebSecurity` helper `ResetPassword` folyamatát az új jelszóval és a tokennel. Ha a token érvényes, akkor a segéd frissíti annak felhasználónak a

jelszavát, aki az e-mailt kapta. Ha a jelszó-visszaállítás sikeres, a `ResetPassword` metódus `true` értéket ad vissza.

Ebben a példában a `ResetPassword` meghívását kombináltuk néhány érvényességi vizsgálattal a `&&` (logikai ÉS) operátorral. A logika, hogy a jelszóváltoztatás sikeres, ha:

- a `newPassword` szövegdoboz nem üres (a ! operátor jelentése *not* – *nem*), és
- a `newPassword` és a `confirmPassword` mező értéke megegyezik, és
- a `ResetPassword` folyamat sikeres volt.

4. Nyissuk meg a *ForgotPassword2.cshtml*-t egy böngészőben!

5. Írjuk be az e-mail címünket és kattintsunk a *Get New Password* gombra! Az oldal elküld egy e-mailt. (Ez néhány pillanatot vehet igénybe.)

6. Ellenőrizzük a leveleinket, és keressünk egy „Password Reset” tárgyú levelet!
7. Az e-mailben kattintsunk a linkre! A *PasswordReset2.cshtml* oldalra jutunk.
   Adjunk meg egy új jelszót és kattintsunk a *Submit* gombra!

Automatizált programok csatlakozásának megakadályozása

A bejelentkező képernyő nem fogja megállítani az automatizált (néha *web robot*-okként vagy *bot*-okként említett) programokat abban, hogy regisztráljanak a weboldalra. (A botok csatlakozásának célja általában az, hogy az eladásra tett termékek linkjét megszerezzék.) A CAPTCHA teszt használatával kideríthetjük, hogy a felhasználó valódi személy-e, nem csak egy számítógépes program. (A CAPTCHA a Completely Automated Public Turing teszt rövidítése, aminek az a feladata, hogy megkülönböztesse a számítógépeket és az embereket.)

Az ASP.NET oldalakon használhatjuk a `ReCaptcha` helpert, hogy készítsen egy CAPTCHA tesztet a reCAPTCHA szolgáltatás (http://recaptcha.net) segítségével. A `ReCaptcha` helper megjelenít egy képet két elválasztott szóval, amiket a felhasználóknak helyesen le kell írniuk, mielőtt az oldal érvényes lesz. A felhasználó válaszát a `ReCaptcha.Net` szolgáltatás ellenőrzi.

1. Regisztráljuk a weboldalt a ReCaptcha.Net (http://recaptcha.net) oldalon! Amikor készen vagyunk a regisztrációval, kapunk egy nyilvános és egy privát kulcsot.
2. Az *Account* mappában nyissuk meg a *Register.cshtml* fájlt!
3. Töröljük a `//` karaktereket a `captchaMessage` változó elől!
4. Cseréljük le a `PRIVATE_KEY` stringet a saját privát kulcsunkkal!
5. Töröljük a `//` karaktereket abból a sorból, ami tartalmazza a `ReCaptcha.Validate` meghívását!

   A következő példa a kész kódot mutatja a kulcs helyével.

```
// Validate the user's response
if (!ReCaptcha.Validate("user-key-here")) {
   captchaMessage = "Response was not correct";
   isValid = false;
}
```

6. A *Register.cshtml* oldal alján cseréljük ki a `PUBLIC_KEY` stringet a nyilvános kulcsunkkal!
7. Töröljük a `//` karaktereket abból a sorból, ami tartalmazza a `ReCaptcha` meghívását! A alábbi példa a kész kódot mutatja a kulcs helyével.

```
@ReCaptcha.GetHtml("user-key-here", theme: "white")
```

   A következő illusztráció a kész regisztrációs oldalt mutatja.

8. Nyissuk meg a Default.cshtml oldalt egy böngészőben! Ha be vagyunk jelentkezve, kattintsunk a *Logout* llinkre!
9. Kattintsunk a Register linkre és teszteljük a regisztrációt a CAPTCHA teszt használatával!

Megjegyzés: Ha a számítógépünk egy olyan tartomány tagja, ami proxy szervert használ, lehet, hogy konfigurálnunk kell a *Web.config* fájl `defaultproxy` paraméterét. Az alábbi példa egy olyan *Web.config* fájlt mutat, ahol a `defaultproxy` úgy van konfigurálva, hogy a reCAPTCHA szolgáltatás működik.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.net>
      <defaultProxy>
         <proxy
            usesystemdefault = "false"
            proxyaddress="http://myProxy.MyDomain.com"
            bypassonlocal="true"
            autoDetect="False"
         />
      </defaultProxy>
   </system.net>
</configuration>
```

18.fejezet – Bevezetés a hibakeresésbe

A hibakeresés a hibák megállapításának és kijavításának folyamata a kódlapunkon. Ez a fejezet megmutat néhány eszközt és technikát, hogyan használjuk a hibakeresést és elemezzük webhelyünket.

A fejezetből megtudhatjuk:

- Hogyan jelenítsünk meg információkat, melyek elősegítik az oldal elemzését és a hibakeresést?
- Hogyan használjuk a hibakereső eszközöket, mint az Internet Explorer Developer Toolst és a Firebugot a webhely elemzéséhez?

A fejezetben bemutatott ASP.NET funkciók és WebMatrix (valamint egyéb) eszközök:

- A `ServerInfo` helper.
- Az `ObjectInfo` helper.
- Az Internet Explorer Developer Tools és a Firebug hibakereső eszközök.

Egy fontos szempont a hibák és a problémák elhárításában, hogy először el kell kerülni őket. Ezt úgy tehetjük meg, hogy azokat a részeit a kódnak, amelyek a leginkább hibát szoktak okozni, behelyezzük egy `try/catch` blokkba. További információkért olvassuk el 3. fejezetben a hibák kezeléséről szóló részt!

A ServerInfo helper használata a szerverinformációk megjelenítéséhez

A `ServerInfo` helper egy olyan diagnosztikai eszköz, amely áttekintést nyújt a webszerver környezetéről. Megmutatja emellett az oldal lehívásakor elküldött HTML kéréseket is. A `ServerInfo` helper kijelzi az aktuális felhasználó azonosítóját, a böngésző típusát, amiről benyújtotta a kérelmet, és így tovább. Ezen információk birtokában könnyebben kivédhetjük a gyakori problémákat.

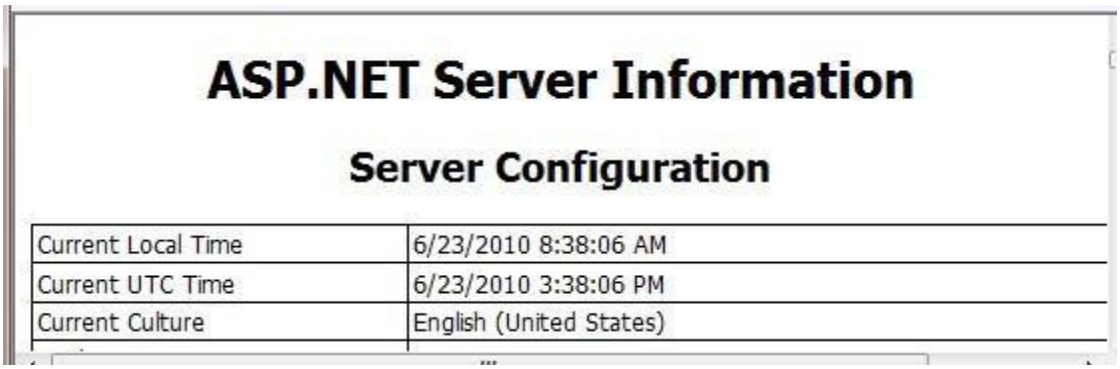
1. Készítsünk egy *ServerInfo.cshtml* nevű weblapot.
2. Az oldal végén, mielőtt bezárnánk a `</body>` címkével, adjuk hozzá az `@ServerInfo.GetHtml()` kódot:

```
<!DOCTYPE html>
<html>
    <head>
        <title></title>
    </head>
    <body>
   @ServerInfo.GetHtml()
    </body>
</html>
```

   Az oldalon bárhol elhelyezhetjük a `ServerInfo` kódot. Azonban ha a kód végéhez fűzzük, nem keveredik az oldal többi részével, így kódunk áttekinthető marad.

Megjegyzés: Mielőtt éles üzembe állítjuk a webhelyet, töröljük ki az összes diagnosztikával kapcsolatos kódot, beleértve a `ServerInfo` helpert. Nem szeretnénk, ha a weboldal látogatói bármilyen információt láthatnának a szerverünk nevéről, felhasználónevekről, szerveren lévő elérési utakról és hasonló részletekről, mivel ezek hasznosak lehetnek rossz szándékú emberek számára.

3. Mentsük el az oldalt, majd futtassuk a böngészőben! (Futtatás előtt ügyeljünk arra, hogy az oldal ki legyen választva a *Files* munkaterületen!)



A `ServerInfo` helper négy információs táblát jelez ki az oldalon:

- Server Configuration. Információkat ad a futtató webszerverről, és tartalmazza a számítógép nevét, az éppen futó ASP.NET verzióját, a domain nevét, valamint a szerveridőt.
- ASP.NET Server Variables. Információkat ad számos HTTP protokoll részleteiről (melyeket HTTP változóknak nevezünk) és azokról az értékekről, amelyek szerepelnek minden weboldal lekérésében.
- HTTP Runtime Information. Részleteket közöl a webhelyet futtató Microsoft .NET Framework verziójáról, , az elérési útat, részleteket a gyorsítótárról, és így tovább. (Ahogy az 3. fejezetben megtanultuk, az ASP.NET weboldalak által használt Razor szintaxis a Microsoft ASP.NET webszerver technológiájára épült, amely maga is egy .NET Framework nevű kiterjedt szoftverfejlesztő könyvtárra épül.)
- Environment Variables. Egy listát ad a webszerver összes környezeti változójáról és azok értékeikről.

A szerver teljes leírása és a kérelmi információ már túlmutatnak ezen a fejezeten, de láthatjuk, hogy a `ServerInfo` helper számos diagnosztikai információt ad nekünk. További információkért a `ServerInfo` által visszadobott értékekről látogassunk el a Recognized Enviromnent Variables oldalra a Microsoft TechNet weboldalon és az IIS Server Variables oldalra az MSDN weboldalon!

Oldal értékek kijelzése kifejezések beágyazásával

Kódunk működését egy másik eljárással is vizsgálhatjuk. Mint tudjuk, egy változó értékét közvetlenül is kijelezhetjük, ha hozzáadunk az oldalhoz egy ehhez hasonlót: `@myVariable or @(subTotal * 12)`. A hibakereséshez jelenítsük meg a változókat a kód kritikus részeinél! Ez lehetővé teszi, hogy lássuk a fontos változók értékét, illetve a számítások eredményeit, mikor fut az oldal. Amikor végeztünk a hibakereséssel, eltávolíthatjuk a kifejezéseket vagy megjegyzésbe tehetjük őket. Ez az eljárás bemutatja egy tipikus módját, hogy hogyan használjuk a beágyazást a hibakeresés elősegítéséhez.

1. Készítsünk egy új, *OutputExpression.cshtml* nevű WebMatrix lapot!
2. Cseréljük ki az oldal tartalmát a következőre:

```
<!DOCTYPE html>
<html>
    <head>
        <title></title>
    </head>
    <body>

    @{
        var weekday = DateTime.Now.DayOfWeek;
        // As a test, add 1 day to the current weekday.
        if(weekday.ToString() != "Saturday") {
```

```
            // If weekday is not Saturday, simply add one day.
            weekday = weekday + 1;
        }
        else {
            // If weekday is Saturday, reset the day to 0, or Sunday.
            weekday = 0;
        }
        // Convert weekday to a string value for the switch
statement.
        var weekdayText = weekday.ToString();

        var greeting = "";

        switch(weekdayText)
        {
            case "Monday":
                greeting = "Ok, it's a marvelous Monday.";
                break;
            case "Tuesday":
                greeting = "It's a tremendous Tuesday.";
                break;
            case "Wednesday":
                greeting = "Wild Wednesday is here!";
                break;
            case "Thursday":
                greeting = "All right, it's thrifty Thursday.";
                break;
            case "Friday":
                greeting = "It's finally Friday!";
                break;
            case "Saturday":
                greeting = "Another slow Saturday is here.";
                break;
            case "Sunday":
                greeting = "The best day of all: serene Sunday.";
                break;
            default:
                break;
        }
    }

    <h2>@greeting</h2>

    </body>
</html>
```

Ez a példa egy `switch` utasítást használ annak érdekében, hogy ellenőrizze a `weekday` értékét, és ezután megjelenítsen egy üzenetet, ami aszerint változik, hogy melyik napja az a hétnek. A példában az `if` tömb az első kód blokkban önkényesen megváltoztatja a hét napját, egyet hozzáadva a hét jelenlegi napjának értékéhez. Íme egy hiba, szemléltetés céljából.

3. Mentsük el az oldalt, és futtassuk egy böngészőben!

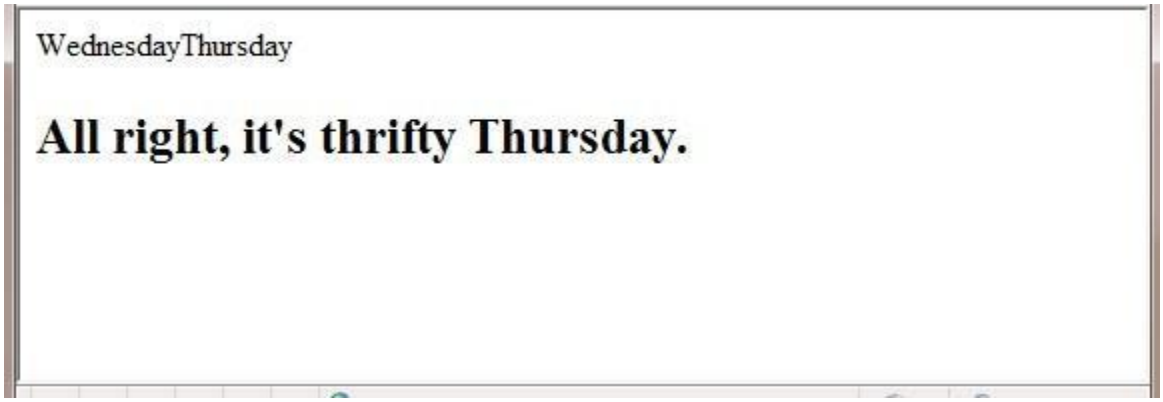
   Az oldalon az üzenet a hét nem megfelelő napjával fog megjelenni. A hét bármely napja is legyen aktuális, az egy nappal későbbi üzenetet fogjuk látni. Bár ebben az esetben tudjuk, hogy miért a hibás üzenet jelent meg (mivel a kód szándékosan a nem megfelelő nap értékét állítja be), a valóságban sokszor nehéz megállapítani, hogy hol van hiba a kódban. A hibakereséshez ki kell találnunk, mi is történik a kulcsobjektumok értékével és a változókkal, esetünkben a hét napjaival.

4. Adjunk hozzá egy kifejezést a `@weekday` beillesztésével két, megjegyzésekkel jelölt helyre a kódban. Ezek a kifejezések ki fogják jelezni a változók értékeit ezeken a pontokon.

```
var weekday = DateTime.Now.DayOfWeek;
    // Display the initial value of weekday.
    @weekday

    // As a test, add 1 day to the current weekday.
    if(weekday.ToString() != "Saturday") {
        // If weekday is not Saturday, simply add one day.
        weekday = weekday + 1;
    }
    else {
        // If weekday is Saturday, reset the day to 0, or Sunday.
        weekday = 0;
    }

    // Display the updated test value of weekday.
    @weekday

    // Convert weekday to a string value for the switch statement.
    var weekdayText = weekday.ToString();
```

5. Mentsük el az oldalt, majd futtassuk egy böngészőben!

   Az oldal először a hét valódi napját jelzi ki, majd a frissített napot - ami az egy nap hozzáadásának az eredménye -, majd a `switch` utasítástól kapott üzenetet. A két változó kifejezésből (`@weekday`) származó kimenet között nem lesz szóköz, mivel semmilyen HTML `<p>` címkét nem adtunk hozzá, a kifejezéseket most csak tesztelés céljából szúrtuk be.



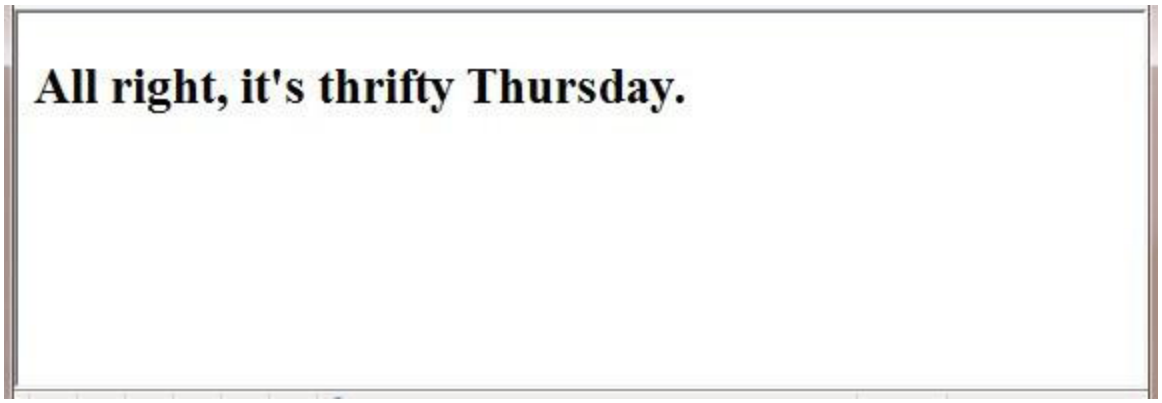
   Most már láthatjuk, hol a hiba. Mikor először jelenik meg a `weekday` változó a kódban, a megfelelő napot mutatja. Mikor második alkalommal jelenik meg az `if` tömb után, a nap eggyel későbbi a kelleténél, tehát beláthatjuk, hogy valami történt a weekday változó első és második megjelenése között a. Ha ez egy valódi hiba lenne, ezzel a megközelítéssel közelebb kerülünk a problémát okozó kódrészlethez.

6. Javítsuk ki a kódot az oldalon a két nemrég hozzáadott kifejezés törlésével, és a nap megváltozását okozó kód eltávolításával. A fennmaradó, teljes kódtömb ehhez így fog kinézni:

```
@{
    var weekday = DateTime.Now.DayOfWeek;
    var weekdayText = weekday.ToString();
```

```
    var greeting = "";

    switch(weekdayText)
    {
        case "Monday":
            greeting = "Ok, it's a marvelous Monday.";
            break;
        case "Tuesday":
            greeting = "It's a tremendous Tuesday.";
            break;
        case "Wednesday":
            greeting = "Wild Wednesday is here!";
            break;
        case "Thursday":
            greeting = "All right, it's thrifty Thursday.";
            break;
        case "Friday":
            greeting = "It's finally Friday!";
            break;
        case "Saturday":
            greeting = "Another slow Saturday is here.";
            break;
        case "Sunday":
            greeting = "The best day of all: serene Sunday.";
            break;
        default:
            break;
    }
}
```

7. Futtassuk az oldalt egy böngészőben! Mostantól a megfelelő üzenet fog megjelenni a hét aktuális napjával.



Objektumértékek kijelzése az ObjectInfo helper használatával

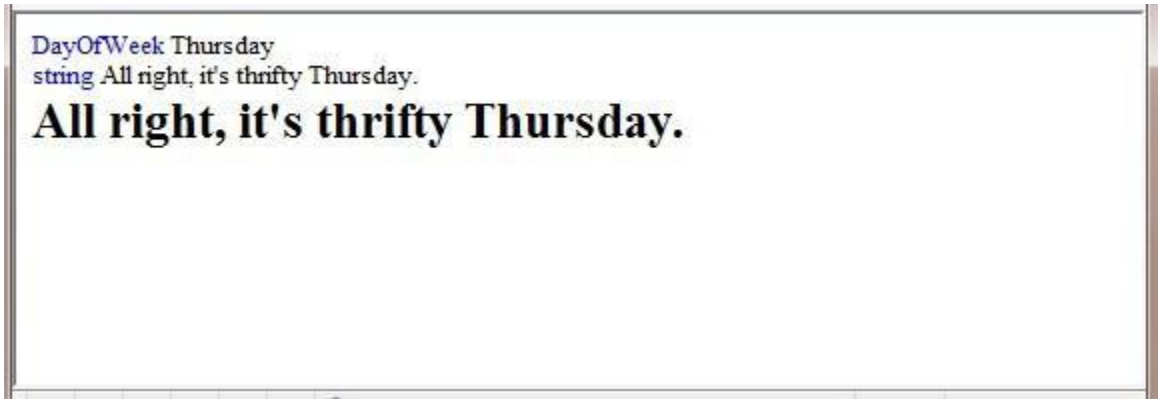
Az `ObjectInfo` helper az összes kívánt objektum típusát és értékét kijelzi. Használhatjuk változók és objektumok értékeinek megtekintéséhez a kódban (mint ahogy azt az előző példában tettük a kifejezések segítségével), valamint láthatunk a kiválasztott objektumok típusát is.

1. Nyissuk meg a korábban elkészített *OutputExpressions.cshtml* fájlt!
2. Helyettesítsük az összes kódot az oldalon a következő kódtömbbel:

```
<!DOCTYPE html>
<html>
    <head>
        <title></title>
```

```
    </head>
    <body>
    @{
      var weekday = DateTime.Now.DayOfWeek;
      @ObjectInfo.Print(weekday)
      var weekdayText = weekday.ToString();

      var greeting = "";

      switch(weekdayText)
      {
          case "Monday":
              greeting = "Ok, it's a marvelous Monday.";
              break;
          case "Tuesday":
              greeting = "It's a tremendous Tuesday.";
              break;
          case "Wednesday":
              greeting = "Wild Wednesday is here!";
              break;
          case "Thursday":
              greeting = "All right, it's thrifty Thursday.";
              break;
          case "Friday":
              greeting = "It's finally Friday!";
              break;
           case "Saturday":
              greeting = "Another slow Saturday is here.";
              break;
           case "Sunday":
              greeting = "The best day of all: serene Sunday.";
              break;
          default:
              break;
      }
    }
    @ObjectInfo.Print(greeting)
    <h2>@greeting</h2>

    </body>
</html>
```

3. Mentsük el, majd futtassuk az oldalt egy böngészőben!



Ebben a példában az `ObjectInfo` két adatot jelenít meg:

- A típust. Az első változóban a típus a `DayOfWeek`. A második változó egy `string`.

- Az érték. Ebben az esetben, mivel a kezdeti változó értéke már megjelent az oldalon, az érték ismét megjelenik, a változó eljut `ObjectInfo`hoz.

További, bonyolultabb objektumok esetén az `ObjectInfo` helper több információt is képes megjeleníteni – alapvetően minden típust és értéket ki tud jelezni az objektumok tulajdonságai közül.

Hibakereső eszközök használata

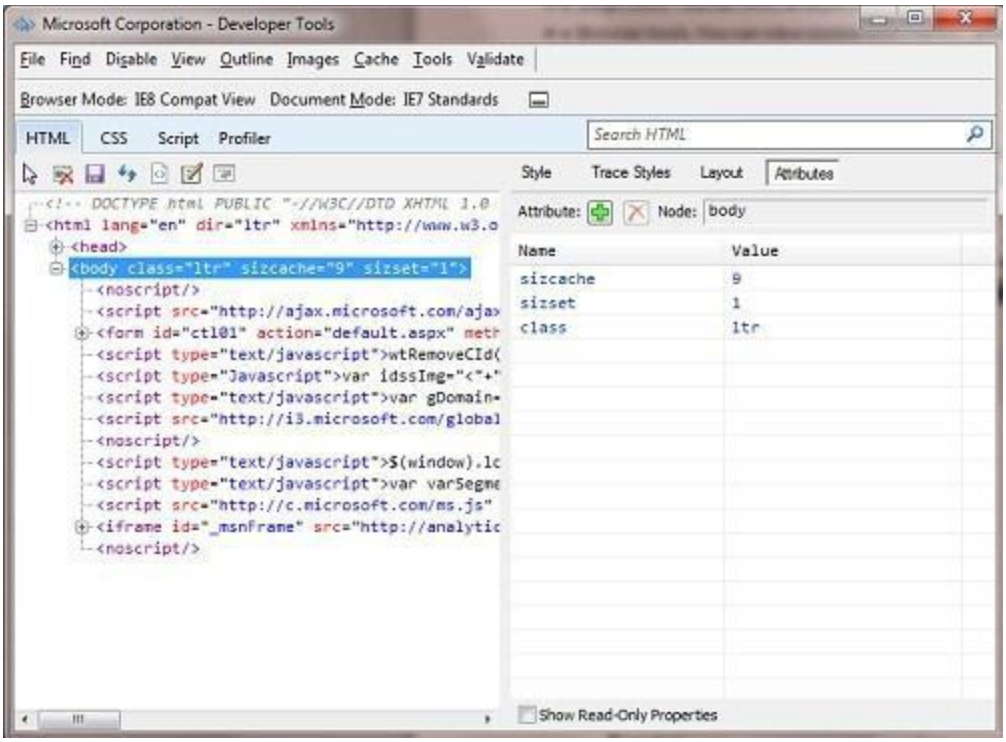
Amellett, hogy az oldal a hibakeresés megkönnyítéséhez információkat jelenítsen meg, használhatunk eszközöket is, amelyek információt nyújtanak arról, hogyan fut az oldal. Ez a rész bemutatja, hogyan használjuk a legnépszerűbb diagnosztikai eszközöket a weboldalon, és hogyan használjunk néhány eszközt a WebMatrixban, melyek ugyancsak segítik az oldalon való hibakeresést.

Internet Explorer Developer Tools

Internet Explorer Developer Tools egy Internet Explorer 8-ba beépülő web eszköz csomag. (Az Internet Explorer korábbi verzióihoz telepíthetjük az eszközt az Internet Explorer Developer Toolbar oldalról a Microsoft Download Centerben.) Ez az eszköz nem kifejezetten engedi az ASP.NET kódban való hibakeresést, de igazán hasznos lehet a hibakereséshez a HTML-ben, CSS-ben és szkriptekben, beleértve az ASP.NET weboldalakon dinamikusan létrejövő utasításokat és szkripteket.

Ez az eljárás szemlélteti, hogyan működik az Internet Explorer Developer Tools. Ez azt feltételezteti, hogy Internet Explorer 8-at használunk.

1. Az Internet Explorerben nyissunk meg egy publikus weboldalt, mint például a *www.microsoft.com*!
2. A *Tools* menüben kattintsunk a *Developer Tools*ra!
3. Kattintsunk a *HTML* fülre, nyissuk le a `<html>` elemet, majd a `<body>` elemet is! Az ablak az összes címkét a body elemben mutatja meg.
4. A jobb oldali panelen kattintsunk az *Attributes* fülre, hogy láthassuk a `<body>` címke tulajdonságait:



5. A jobb oldali panelon kattintsunk a *Style*-ra, hogy lássuk a CSS stílusokat, amik a body szakaszhoz tartoznak az oldalon.

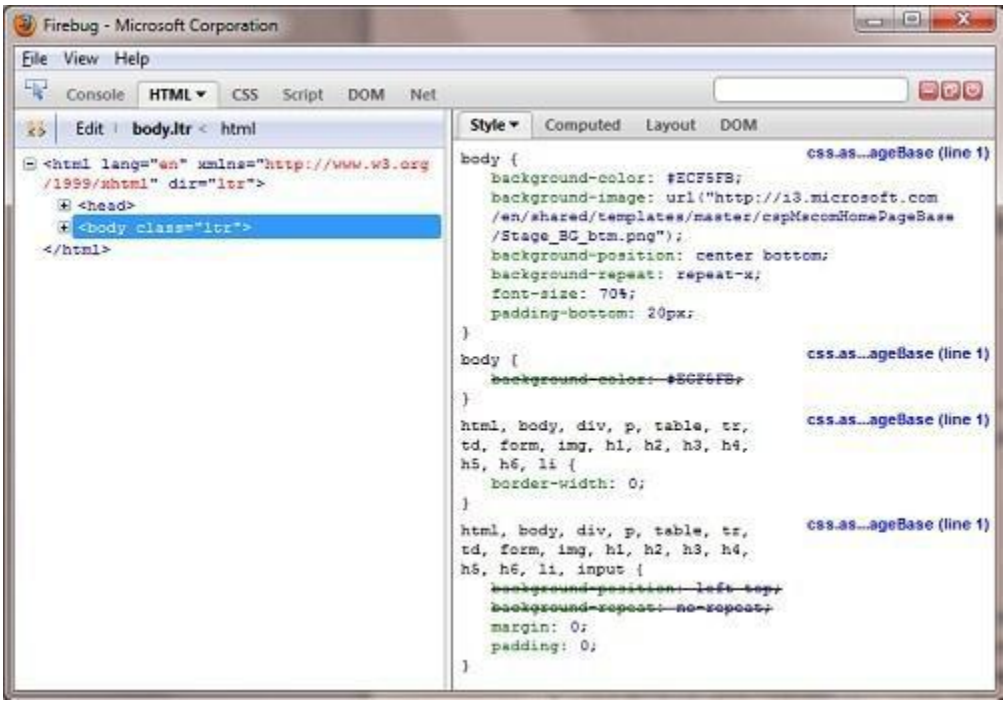
Az Internet Explorer Developer Tools-ról látogassunk el a Discovering the Internet Explorer Developer Tools-ra az MSDN weboldalán!

Firebug

Firebug egy add-on a Mozilla Firefoxhoz, amivel ellenőrizni lehet a HTML kódokat és a CSS-t, hibákat lehet kerestetni a kliens szkriptben és lehetővé teszi, hogy lássuk a cookie-kat és egyéb információkat az oldalról. A Firebugot telepíthetjük a Firebug weboldaláról. Ahogyan az Internet Explorer hibakereső eszközeiv, ez az eszköz sem kimondottan alkalmas az ASP.NET kódban való hibakeresésre, viszont hasznos lehet HTML és egyéb oldalelemek vizsgálatában, beleértve azokat, amelyeket az ASP.NET generált dinamikusan.

Ez a leírás megmutat nekünk néhányat a Firebug funkciói közül.

1. A Firefoxban nyissuk meg a *www.microsoft.com*-ot!
2. A *Tools* menüben kattintsunk a *Firebug*ra, majd kattintsunk az *Open Firebug in New Window*-ra!
3. A Firebug fő ablakában kattintsunk a *HTML* fülre, ezután nyissuk szét a `<html>` csomópontot a bal panelen!
4. Válasszuk a `<body>` címkét, majd kattintsunk a *Style* fülre a jobb oldali panelon! A Firebug megjeleníti nekünk az információkat a Microsoft oldaláról.



A Firebug számos beállítást tartalmaz a HTML és CSS stílusok szerkesztéséhez és érvényesítéséhez, valamint a szkriptekben lévő hibák kereséséhez és javításához. A *Net* fülön lehet elemezni a szerverek és weboldalak közti hálózati forgalmat. Például megvizsgálhatjuk hogy lássuk, meddig tart letöltenie az összes tartalmat a böngészőnek. Ha többet szeretnénk megtudni a Firebugról, látogassunk el a Firebug főoldalára és a Firebug Documentation Wiki-re!

További források angolul

IIS Server Variables

Recognized Enviromnent Variables

Internet Explorer Developer Tools:

- Discovering the Ineternet Explorer Developer Tools
- Download the IE Developer Tools (IE version prior to IE 8)
- Debugging HTML and CSS with the Developer Tools
- Debugging Script with the Developer Tools

Firebug Add-on for Web Developers:

- Firebug main site
- Firebug Documentation Wiki

19.fejezet – A Site-Wide viselkedés testreszabása

Ebben a fejezetben megtanuljuk, hogyan hozzunk létre olyan beállításokat, amelyek az egész weboldalra vagy egy mappára vannak hatással, nem csak egyetlen oldalra.

Az fejezetből megtudhatjuk:

- Hogyan futtassunk olyan kódot, amely egy weboldal összes oldalán tud értékeket beállítani?
- Hogyan futtassunk olyan kódot, amely egy mappában található összes oldalon tud értékeket beállítani?
- Hogyan futtassunk kódot, mielőtt és miután az oldal betöltődött?
- Hogyan használja az ASP.NET az útvonalkezelést, hogy könnyebben olvasható és kereshető URL-eket használhassunk?

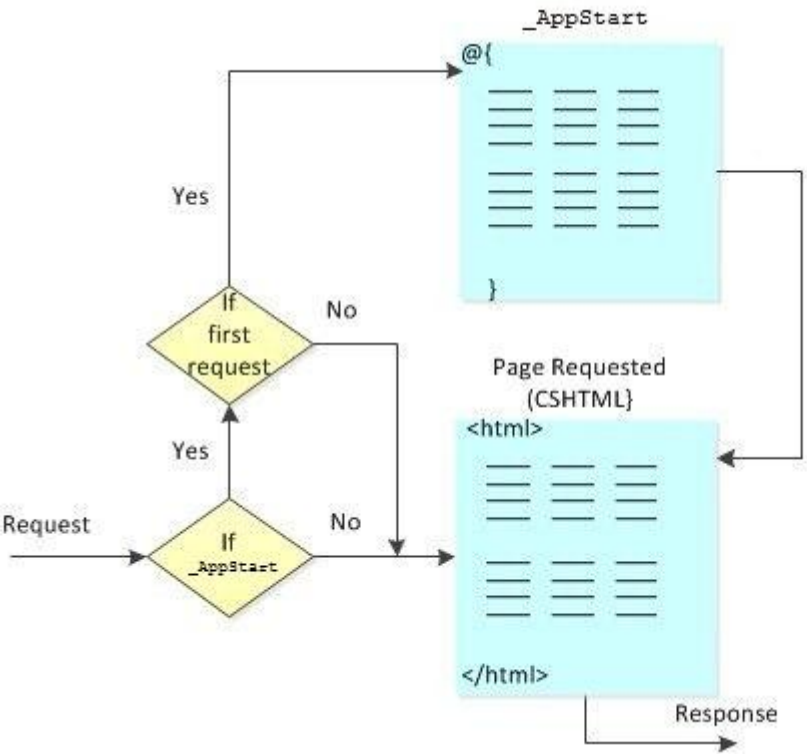
Weboldal indulásakor lefutó kód hozzáadása

Az általunk megírt kódok többsége és a beállítások, amiket használunk, csak egy-egy oldalra érvényesek. Például, ha egy oldal e-mailt küld, akkor általában az oldal tartalmazza az egész kódot, ami szükséges a beállítások inicializálásához (SMTP szerverhez) és az e-mail elküldéséhez.

Azonban néhány esetben már akkor szeretnénk kódot futtatni, amikor még egyetlen oldal sem töltődött be. Így beállíthatunk olyan értékeket, amiket az oldalunkon bárhol használhatunk (*Globális értékek*). Néhány helper kötelez különböző értékek/adatok megadására, mint például e-mail beállítások vagy account kulcsok, szóval hasznos lehet ezeket az adatokat globális változókban tárolni.

Ezt úgy tehetjük meg, hogy létrehozunk egy *_AppStart.cshtml* oldalt a honlap gyökerében. Ha ez az oldal létezik, akkor az ebben található kód lefut, mikor lekérnek egy oldalt a honlapról. Tehát ez jó hely olyan kód futtatására, ami globális változókat állít be.(Mivel az *_AppStart.cshtml* tartalmaz egy alulvonás előtagot, az ASP.NET nem fogja elküldeni az oldalt a böngészőnek még akkor sem, ha a felhasználó ezt szeretné.)

A következő diagram bemutatja, hogyan működik az *_AppStart.cshtml*. Amikor először kérelem érkezik egy oldalért, az ASP.NET először megvizsgálja, hogy létezik-e egy *_AppStart.cshtml* oldal, ha igen, akkor először az abban található kód fut le és csak utána a kért oldal.



Globális értékek beállítása a weboldalon

1. A WebMatrix weboldal gyökérkönyvtárában hozzunk létre egy *_AppStart.cshtml* fájlt! A fájlnak mindenképpen a honlap gyökerében kell lennie.
2. Cseréljük le az alapértelmezett kódot a következőre:
```
@{
  AppState["customAppName"] = "Application Name";
}
```
3. Ez a kód egy olyan értéket tárol az `AppState` szótárban, ami automatikusan elérhető az összes oldal számára a webhelyen.

Megjegyzés: Vigyázzunk, amikor az *_AppStart.cshtml*-t készítjük, ugyanis ha bármi hiba van a kódban, akkor nem fog elindulni a honlap!

4. Hozzunk létre egy *AppName.cshtml* oldalt a gyökérkönyvtárban!
5. Cseréljük le az alapértelmezett markupot és a kódot a következőre:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Show Application Name</title>
    </head>
    <body>
        <h1>@AppState["customAppName"]</h1>
    </body>
</html>
```

Ezzel a kóddal kiolvassuk azt az értéket az AppState-ből , amit az *_AppStart.cshtml*-ben állítottunk be.

3. Egy böngészővel nyissuk meg az *AppName.cshtml*-t.(Futtatás előtt bizonyosodjunk meg arról, hogy az oldal ki van választva a *Files* munkaterületen.) Az oldalon megjelenik a globális érték.

Értékadás helperek számára

Az *_AppStart.cshtml*-t jó ötlet arra felhasználni, hogy az oldalunkon felhasznált helpereknek értéket adjunk. Erre tökéletes példa a `ReCaptcha` helper, hiszen ennek a használatához egy publikus és egy privát kulcs megadására van szükség a reCAPTCHA felhasználói fiók számára. Ahelyett, hogy ezeket az értékeket minden oldalon beállítanánk, ahol a ReCaptchat használjuk, beállíthatjuk őket az *_AppStart.cshtml*-ben, így a weblapunk minden oldalán előre beállítjuk őket. A többi érték, amit beállíthatunk az *_AppStart.cshtml*-ben, e-mail küldésre szolgál SMTP szerver felhasználásával. Lásd: 17. fejezet – Biztonsági elemek és felhasználói fiókok hozzáadása

A következő példából megtudhatjuk, hogyan állítsuk be globálisan a `ReCaptcha` kulcsokat.(További információkért a ReCaptcha használatáról lásd: 17. fejezet – Biztonsági elemek és felhasználói fiókok hozzáadása)

1. Ha még nem tettük meg, adjuk hozzá a weboldalunkhoz az ASP.NET Web Helpers Libraryt a 2. fejezetben leírtak alapján!
2. Regisztráljuk weboldalunk a ReCaptcha.Net-en (http://recaptcha.net)! A regisztráció után megkapjuk publikus és privát kulcsunkat.
3. Ha még nem rendelkezünk *_AppStart.cshtml* fájllal a weboldal gyökérkönyvtárában, akkor most hozzunk létre egyet!
4. Töröljünk ki mindent az *_AppStart.cshtml*-ből, és adjuk hozzá a következő kódot:

```
@{
  // Adjunk hozzá PublicKey és PrivateKey stringeket,
  // a ReCaptcha.Net-ről szerezhetünk publikus és privát kulcsokat.
  // (http://recaptcha.net)
  ReCaptcha.PublicKey = "";
  ReCaptcha.PrivateKey = "";
```

5. Töltsük fel a `PublicKey` és `PrivateKey` tulajdonságainkat a saját publikus és privát kulcsainkkal!
6. Mentsük és zárjuk be az *_AppStart.cshtml* fájlunkat!
7. A weboldal gyökerében hozzunk létre egy *Recaptcha.cshtml* oldalt!
8. Cseréljük le az alapértelmezett kódot a következőre:

```
@{
  var showRecaptcha = true;
  if (IsPost) {
    if (ReCaptcha.Validate()) {
        @:Your response passed!
        showRecaptcha = false;
    }
    else{
      @:Your response didn't pass!
    }
  }
}
<!DOCTYPE html>
<html>
    <head>
        <title>Testing Global Recaptcha Keys</title>
    </head>
    <body>
    <form action="" method="post">
    @if(showRecaptcha == true){
        if(ReCaptcha.PrivateKey != ""){
            <p>@ReCaptcha.GetHtml()</p>
            <input type="submit" value="Submit" />
        }
        else {
            <p>You can get your public and private keys at
            the ReCaptcha.Net website (http://recaptcha.net).
            Then add the keys to the _AppStart.cshtml file.</p>
        }
    }
    </form>
    </body>
</html>
```

9. Egy böngészővel nyissuk meg a Recaptcha.*cshtml*-t! Ha a `PrivateKey` értékünk érvényes, akkor az oldalon meg fog jelenni a reCAPTCHA panel és egy gomb. Ha a kulcsokat nem állítottuk volna be globálisan az *_AppStart.cshtml*-ben, akkor az oldal hibát jelezne.

10. Tesztelés céljából írjuk be a szavakat! Ha megfelelünk a reCAPTCHA teszten, akkor megjelenik egy beállított üzenet, ha mégsem, akkor egy hibaüzenet jelenik meg, majd újra megjelenik a reCAPTCHA panel.

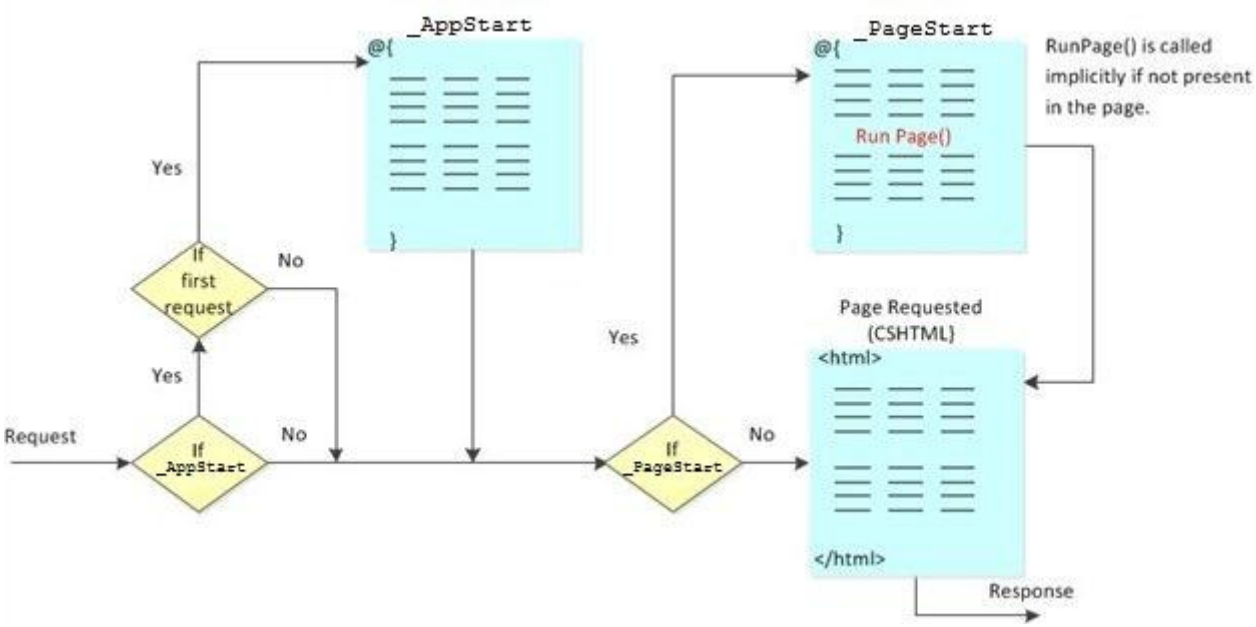
Kód futtatása egy mappa fájljainak elindulása előtt és lefutása után

Ahogy használhatjuk az *_AppStart.cshtml*-t arra, hogy kódot futtassunk, mielőtt a weboldal bármelyik oldala lefutna, ugyanúgy írhatunk olyan kódot, ami egy bizonyos könyvtárban található fájlok előtt és

után lefut. Ez azért hasznos, mert így ugyanazt a layout oldalt használhatjuk a mappa összes oldalára, vagy egy oldal megnyitása előtt ellenőrizhetjük, hogy a felhasználó be van-e jelentkezve.

Ilyen típusú kódok létrehozására és futtatására egy *_PageStart.csthml* fájlt használunk. A következő diagram bemutatja, hogyan működik a *_PageStart.cshtml* oldal. Ha kérelem érkezik egy oldalért, akkor az ASP.NET először megvizsgálja, hogy létezik-e egy *_AppStart.cshtml*, ha igen, akkor futtatja azt. Ezek után ellenőrzi, hogy létezik-e egy _PageStart.cshtml oldal, ha igen, azt is lefuttatja, és csak ezek után futtatja le a kért oldalt.

A *_PageStart.cshtml*-be beleírt `RunPage` metódussal megadhatjuk, hogy a feldolgozás melyik pillanatában jelenjen meg a lekért weboldal, ezzel lehetőségünk van kódot futtatni a lekért oldal előtt vagy akár utána. Ha nem teszünk `RunPage` metódust a *_PageStart.cshtml*-be, akkor előbb az fut le, majd utána a lekért weboldal.



Az ASP.NET segítségével a *_PageStart.cshtml* fájlokból egy egész hierarchiát hozhatunk létre. Tehetünk egy *_PageStart.cshtml*-t a weboldal gyökérkönyvtárába és minden alkönyvtárába. Ha lekérnek egy oldalt, akkor a legmagasabb szintű (legközelebbi a gyökérkönyvtárhoz) *_PageStart.cshtml* fut le, majd utána következő almappában található *_PageStart.cshtml*. Az ASP.NET így megy végig a könyvtár struktúrán, amíg el nem éri a kívánt oldalt tartalmazó mappát. A kért oldal csak akkor következik, miután minden megfelelő *_PageStart.cshtml* fájl végrehajtódott.

Például lehetséges, hogy a következő *_PageStart.cshtml* és *default.cshtml* kombinációval rendelkezünk:

```
@* /_PageStart.cshtml *@
@{
  PageData["Color1"] = "Red";
  PageData["Color2"] = "Blue";
}


@* /myfolder/_PageStart.cshtml *@
@{
  PageData["Color2"] = "Yellow";
```

```
  PageData["Color3"] = "Green";
}


@* /myfolder/default.cshtml *@
@PageData["Color1"]
<br/>
@PageData["Color2"]
<br/>
@PageData["Color3"]
```

Amikor a *default.cshtml*-t futtatjuk, a következőt látjuk:

```
Red
Yellow
Green
```

Inicializáló kód futtatása minden oldalra egy mappában

Jó ötlet a *_PageStart.cshtml*-t arra használni, hogy ugyanazt a szerkezeti oldalt inicializáljuk minden fájlra egy mappában.

1. A gyökérkönyvtárban hozzunk létre egy úgy mappát *InitPages* néven!
2. Hozzunk létre egy *_PageStart.cshtml* fájlt az *InitPages* könyvtárban, majd cseréljük le az alapértelmezett kódot a következőre:

```
@{
    // Beállítjuk a szerkezeti oldalt az összes oldalra a mappában.
    Layout = "/Shared/_Layout1.cshtml";

    // Beállítunk egy változót, ami a mappa minden oldala számára elérhető.
    PageData["MyBackground"] = "Yellow";
}
```

3. A weboldal gyökerében hozzunk létre egy *Shared* mappát!
4. A *Shared* mappába hozzunk létre egy *_Layout1.cshtml* fájlt, majd cseréljük le az alapértelmezett kódot a következőre:

```
@{
  var backgroundColor = PageData["MyBackground"];
}
<!DOCTYPE html>
<html>
<head>
  <title>Page Title</title>
  <link type="text/css" href="/Styles/Site.css" rel="stylesheet" />
</head>
<body>
  <div id="header">
    Using the _PageStart.cshtml file
  </div>
  <div id="main" style="background-color:@backgroundColor">
    @RenderBody()
  </div>
<div id="footer">
  &copy; 2010 Contoso. All rights reserved
</div>
</body>
</html>
```

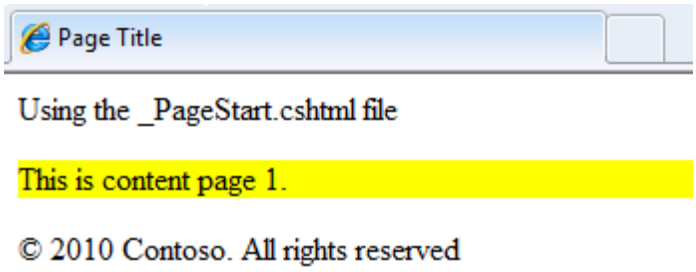
5. Az *InitPages* könyvtárban hozzunk létre egy *Content1.cshtml* fájlt, majd az alapértelmezett markupot cseréljük a következőre:

```
<p>This is content page 1.</p>
```

6. Az *InitPages* könyvtárban hozzunk létre még egy fájlt *Content2.cshtml* néven, és cseréljük le az alapértelmezett markupot a következőre:

```
<p>This is content page 2.</p>
```

7. Egy böngészővel nyissuk meg a *Content1.cshtml*-t!



   Mikor a *Content1.cshtml* elindul, akkor a *_PageStart.cshtml* beállítja a `Layout`ot és a `PageData["MyBackground"]`-t beállítja egy színre.
8. Egy böngészővel nyissuk meg a *Content2.cshtml*-t!
   A szerkezet és a színek ugyanazok lesznek a két oldalon, mivel mind a kettő ugyanabból a *_PageStart.cshtml*-ből veszi az adatokat.

A _PageStart.cshtml használata hibák kezelésére

Másik jó felhasználási módja a *_PageStart.cshtml*-nek, hogy kezeljük a különböző hibákat és kivételeket, amik felmerülhetnek bármilyen *cshtml* oldalon. A következő példa erre mutat be egy módszert.

1. A gyökérkönyvtárban hozzunk létre egy *InitCatch* könyvtárat!
2. Hozzunk létre egy *_PageStart.cshtml*-t az *InitCatch* könyvtárban, és cseréljük ki a létező kódot a következővel:

```
@{
    try
    {
        RunPage();
    }
    catch (Exception ex)
    {
        Response.Redirect("/Error.cshtml?source=" +
            HttpUtility.UrlEncode(Request.AppRelativeCurrentExecution
FilePath));
    }
}
```

   Ebben a kódban explicit módon hívjuk meg a kért oldalt a try blokkba zárt RunPage metódus segítségével. Ha bármi programozási hiba van a kért honlapban, akkor a catch blokk fog lefutni. A mi esetünkben a kód egy másik oldalra irányít (*Error.cshtml*), és az URL részeként elküldi az oldal nevét, amin a hiba történt. (Hamarosan elkészítjük az oldalt.)

3. Az *InitCatch* mappában hozzunk létre egy *Exception.cshtml* fájlt, és a tartalmát cseréljük a következőre:

```
@{
var db = Database.Open("invalidDatabaseFile");
 }
```

A példa szemléltetése miatt most hibát generálunk úgy, hogy olyan adatbázis fájlt próbálunk megnyitni, ami nem létezik.

4. A gyökérkönyvtárban hozzunk létre egy Error.cshtml oldalt, és cseréljük le a tartalmát a következőre:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Error Page</title>
    </head>
    <body>
<h1>Error report</h1>
 <p>An error occurred while running the following file: @Request["source"]</p>
    </body>
</html>
```

Ezen az oldalon a `@Request["source"]` olvassa ki az adatokat az URL-ből és jeleníti meg őket.

5. Mentsük a fájlt!

6. Nyissuk meg az *Exception.cshtml*-t egy böngészőben!

Mivel hiba történik az *Exception.cshtml*-ben, ezért a *_PageStart.cshtml* átirányít minket az *Error.cshtml*-re, ami megjeleníti a hibaüzenetet.

További információért a kivételkezelésről lásd: 3.fejezet – Bevezetés az ASP.NET webszerkesztésbe Razor szintaxissal

A _PageStart.cshtml használata korlátozott mappaeléréshez

Arra is használhatjuk a *_PageStart.cshtml*-t, hogy korlátozzuk a mappában levő összes fájl elérését.

1. Hozzunk létre egy új weboldalt a *Site From Template* opcióval!
2. A lehetséges sablonokból válasszuk ki a *Starter Site*-ot!
3. A gyökérkönyvtárban hozzunk létre egy mappát *AuthenticatedContent* néven!
4. Ebben a mappában hozzunk létre egy *_PageStart.cshtml*-t, és tartalmát cseréljük a következőre:

```
@{
    Response.CacheControl = "no-cache";

    if (!WebSecurity.IsAuthenticated) {
        Response.Redirect("/Account/Login");
    }
}
```

A kód elején megakadályozzuk, hogy bármilyen fájl a mappából a gyorsítótárba mentődjön.(Ez például publikus gépeknél szükséges, hogy az egyik felhasználó gyorsítótárba mentett oldalait a következő felhasználó ne érje el.) Ezután meghatározzuk, hogy a felhasználó belépett-e, mielőtt a mappa bármilyen oldalát megnyithatná. Ha a felhasználó nincs bejelentkezve, akkor a kód visszairányít minket a login oldalra.

5. Hozzunk létre egy *Page.cshtml* oldalt az *AuthenticatedContent* mappába!
6. Cseréljük le az alapértelmezett kódot a következőre:

```
@{
    Layout = "/_SiteLayout.cshtml";
    Page.Title = "Authenticated Content";
}
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
  </head>
  <body>
    Thank you for authenticating!
  </body>
</html>
```

7. Nyissuk meg a *Page.cshtml*-t egy böngészőben! A kód visszairányít minket a login oldalra. Először mindenképpen regisztrálnunk kell, mielőtt beléphetnénk. Miután regisztráltunk és beléptünk, már hozzáférünk az oldal tartalmához.

Könnyebben olvasható és kereshető URL-ek készítése

Oldalaink működését az URL-jeikkel is befolyásolhatjuk. Egy URL, ami „barátságosabb”, könnyebbé teszi az oldal használatát, és persze segít a kereső optimalizálásban is. Az ASP.NET weboldalak képesek automatikusan „barátságosabb” URL-eket használni.

Az útvonalkezelésről

Az ASP.NET segítségével könnyebben értelmezhető URL-eket hozhatunk létre, amik leírják a felhasználó tevékenységét, ahelyett, hogy csak egy fájlra mutatnának a szerveren. Hasonlítsuk össze a következő URL-eket, ezek egy kitalált bloghoz tartoznak:

http://www.contoso.com/Blog/blog.cshtml?categories=hardware

http://www.contoso.com/Blog/blog.cshtml?startdate=2009-11-01&enddate=2009-11-30

http://www.contoso.com/Blog/categories/hardware/

http://www.contoso.com/Blog/2009/November

Az első két példában a felhasználónak tudnia kellene, hogy blog a blog.cshtml oldalon jelenik meg. És össze kell állítania hozzá egy lekérdező parancsot, ami megadja a megfelelő kategóriát, vagy dátumot. Ezzel szemben a másik két példa sokkal könnyebben létrehozható és értelmezhető.

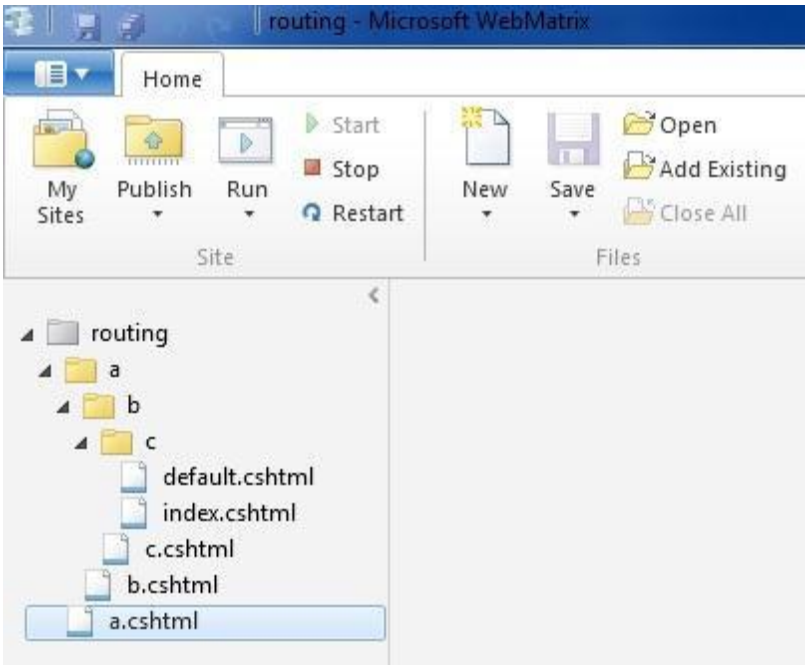
Az első két URL ráadásul egy fájlra mutat (blog.cshtml). Ha valamilyen okból a blogot eltávolítanánk vagy csak másik mappába helyeznénk át, vagy csak egyszerűen másik fájlt használnánk, akkor a linkek nem működnének. A másik két URL nem mutat semmilyen fájlra, így ha a blogon változtatnánk, a link akkor is érvényes maradna és működne.

Az ASP.NET segítségével létrehozhatunk „barátságosabb" URL-eket, olyanokat, mint az előző példákban látottak. Ez azért lehetséges, mert az ASP.NET útvonalkezelést (*routing*) használ. Az útvonal-kezelés segítségével egy olyan logikai térképet készíthetünk egy URL-ből, ami teljesíteni tud egy oldal kérelmet.

Mivel logikai térképet hoz létre és nem fizikait (nem egy fájlra mutat), az útvonalkezelés magas rugalmasságot biztosít abban, hogy hogyan definiáljuk az URL-jeinket.

Hogy működik az útvonalkezelés?

Amikor az ASP.NET feldolgoz egy kérelmet, akkor beolvassa az URL-t, hogy megállapítsa, milyen útvonalat használjon. Megpróbálja balról jobbra haladva párosítani az URL különböző szegmenseit a lemezen található fájlokhoz. Ha egyezést talál, akkor mindent, ami még megmaradt az Url-ben, továbbküld az oldalnak mint *útvonal információ*. Erre jó példa a következő könyvtárstruktúra:



Képzeljük el, hogy valaki a következő URL-t használja:

http://www.contoso.com/a/b/c

A keresés így megy végbe:

1. Létezik-e egy fájl az */a/b/c.cshtml* elérési úttal és névvel? Ha igen, akkor megjelenik, és semmilyen információt nem küld tovább. Ellenkező esetben:

2. Létezik-e egy fájl az */a/b.cshtml* elérési úttal és névvel? Ha igen, akkor megjelenik és továbbküldi a „c”-t mint információt. Ellenkező esetben:
3. Létezik-e egy fájl az */a.cshtml* elérési úttal és névvel? Ha igen, akkor megjelenik és tovább küldi a „b/c”-t mint információt.

Ha a keresés eredménytelen és nem talál pontos egyezést *.cshtml* fájlok között a meghatározott mappájukban, az ASP.NET sorban folytatja e fájlok keresését.

4. */a/b/c/default.cshtml* (Nincs útvonal információ).
5. */a/b/c/index.cshtml* (Nincs útvonal információ).

Megjegyzés: Természetesen azok az URL-ek is hibátlanul működni fognak, amik megadott oldalakra mutatnak (tartalmazzák a *.cshtml* kiterjesztést). Tehát például a *http://www.contoso.com/a/b.cshtml* is működni fog.

Egy oldalon belül az oldal útvonal információját az `UrlData` tulajdonságán keresztül érhetjük el (ez egy szótárérték). Képzeljük el, hogy van egy *ViewCustomers.cshtml* fájlunk, és az oldalunk ezt a kérelmet kapja meg:

http://mysite.com/myWebSite/ViewCustomers/1000

Az előbbiekben leírt szabályok alapján a kérés eléri az oldalunkat. Az oldalunkon a következőhöz hasonló kód segítségével megjeleníthetjük az útvonal információkat.(Ebben az esetben 1000):

```
<!DOCTYPE html>
<html>
    <head>
        <title>URLData</title>
    </head>
    <body>
      Customer ID: @UrlData[0].ToString()
    </body>
</html>
```

Megjegyzés: Mivel az útvonalkezelés nem használ teljes fájlneveket, néha kétértelműség léphet fel, például ha az oldalainknak ugyanaz a neve, de más a kiterjesztésük (*MyPage.cshtml* és *MyPage.html*). Ennek elkerülése érdekében legjobb, ha nem használunk olyan oldalakat, amik csak a kiterjesztésükben térnek el.

Függelék – ASP.NET API referencia

A könyv ezen oldalain megtalálhatjuk azokat az objektumokat, tulajdonságokat, és metódusokat, amiket az ASP.NET Razor szintaxissal legtöbbször használunk. Ez nem egy teljes referencia dokumentum. A teljes referencia majd a hivatalos megjelenéskor lesz elérhető.

Ez a függelék a következő témákban nyújt információkat:

Osztályok

`AsBool(),AsBool(true|false)`

String értékből Booleant (igaz/hamis) konvertál. Ha a megadott string nem felel meg az igaz/hamis (true/false) értékeknek, akkor visszatérési értéke hamis vagy a megadott érték lesz.

bool b = stringValue.AsBool();

`AsDateTime(),AsDateTime(value)`

String értékből Date/Time értéket konvertál. Ha a string nem felel meg a dátum/idő (date/time) értékeknek, akkor a visszatérési értéke a DateTime.MinValue vagy a megadott érték lesz.

```
DateTime dt = stringValue.AsDateTime();
```

`AsDecimal(),AsDecimal(value)`

String értékből decimal értéket konvertál. Ha a string nem felel meg a decimális (decimal) értékeknek, akkor a visszatérési értéke 0.0 vagy a megadott érték lesz.

```
decimal d = stringValue.AsDecimal();
```

`AsFloat(),AsFloat(value)`

String értékből floatot konvertál. Ha a string nem felel meg decimális értékeknek, akkor a visszatérési értéke 0.0 vagy a megadott érték lesz.

```
float d = stringValue.AsFloat();
```

`AsInt(),AsInt(value)`

String értékből integert konvertál. Ha a megadott érték nem felel meg egész számnak (integer), akkor visszatérési értéke 0 vagy a megadott érték lesz.

```
int i = stringValue.AsInt();
```

`Href(path[,param1[,param2]])`

Egy böngésző kompatibilis és opcionális plusz elérési útrészekkel rendelkező URL-t hoz létre egy helyi elérési útból.

```
<a href="@Href ("~/Folder/File")">Link to My File</a>
<a href="@Href ("~/Product", "Tea")">Link to Product</a>
```

`Html.Raw(value)`

Az értéket HTML kódolt kimenet helyett HTML markupként rendereli.

```
@* A string értékét behelyezi a markupba. *@
@Html.Raw("<div>Hello <em>world</em>!</div>")
```

`IsBool(), IsDateTime(), IsDecimal(), IsFloat(), IsInt()`

Visszatérési értéke igaz(true), ha a megadott érték stringből valamelyik kiválasztott típussá

konvertálható.

```
var isint = stringValue.IsInt();
```

`IsEmpty()`

Visszatérési értéke igaz(true) ha a változó vagy objektum nem tartalmaz értéket.

```
if (Request["companyname"].IsEmpty()) {
   @:Company name is required.<br />
}
```

`IsPost`

Ha a lekérdezés POST típusú, akkor visszatérési értéke igaz (true) lesz. (A kezdeti lekérdezések általában GET-ek.)

```
if (IsPost) { Response.Redirect("Posted"); }
```

`Layout`

Az oldalhoz használt layout oldal elérési útját adja meg.

```
Layout = "_MyLayout.cshtml";
```

`PageData[`*`key`*`],PageData[`*`index`*`],Page`

Az aktuális lekérdezés oldalak, szerkezeti oldalak és részleges oldalak között megosztott adatait tárolja. Használhatjuk a dinamikus Page tulajdonságot, hogy ugyanezeket az adatokat elérjük.

```
PageData["FavoriteColor"] = "red"; PageData[1] = "apples";
Page.MyGreeting = "Good morning";
// Megjeleníti a PageData[1]-hez tartozó értéket.
@Page[1]
// Megjeleníti a Page.MyGreetinghez tartozó értéket.
@Page.MyGreeting
```

`RenderBody()`

(Szerkezeti oldalak) Lerendereli az oldal olyan tartalmait, amik nincsenek megnevezett szekciók között.

```
@RenderBody()
```

`RenderPage(`*`path`*`,`*`values`*`)`
`RenderPage(path[,param1[,param2]])`

Lerenderel egy tartalmi oldalt a megadott elérési utat és az opcionális extra adatparamétereket használva. Az adat paramétereket pozíció (példa 1) vagy kulcs (példa2) alapján a PageDatából olvashatjuk ki.

```
RenderPage("_MySubPage.cshtml", "red", 123, "apples")
RenderPage("_MySubPage.cshtml", new { color = "red", number = 123,
food = "apples" })
```

`RenderSection(`*`sectionName[,required`*`=true|false`*`]`*`)`

(Szerkezeti oldalak) Lerenderel egy névvel ellátott tartalmi szekciót. A required részt hamisra (false) kell állítanunk, ha a renderelést opcionálisnak akarjuk hagyni.

```
@RenderSection("header")
```

`Request.Cookies[`*`key`*`]`

Beolvassa vagy beállítja egy http cookie értékét.

```
var cookieValue = Request.Cookies["myCookie"].Value;
```

`Request.Files[`*`key`*`]`

Az aktuális lekérdezésben feltöltött fájlokat olvassa be.

```
Request.Files["postedFile"].SaveAs(@"MyPostedFile");
```

```
Request.Form[key]
```

Egy űrlapon beküldött adatokat olvassa be (stringként). A rövidebb Request[key] megvizsgálja a Request.Form valamint a Request.QueryString gyűjteményt is.

```
var formValue = Request.Form["myTextBox"];
// Ugyanazt az eredményt kapjuk.
var formValue = Request["myTextBox"];
```

```
Request.QueryString[key]
```

Beolvassa az adatot, amit az URL query stringben adtunk meg. A rövidebb Request[key] megvizsgálja a Request.Form, valamint a Request.QueryString gyűjteményt is.

```
var queryValue = Request.QueryString["myTextBox"];
// Ugyanazt az eredményt kapjuk.
var queryValue = Request["myTextBox"];
```

```
Request.Unvalidated(key)
Request.Unvalidated().QueryString|Form|Cookies|Headers[key]
```

Egy kiválasztott űrlap elemnek, query-string értéknek, cookienak, vagy fejléc értéknek kikapcsolja a lekérdezés során történő validációját.

```
// Közvetlenül hívjuk meg a metódust, hogy kikapcsoljuk a validációt
valamelyik Request elemen.
Request.Unvalidated("userText");
// Megadhatjuk, hogy az érték melyik gyűjteményből származik.
var prodID = Request.Unvalidated().QueryString["productID"];
var richtextValue = Request.Unvalidated().Form["richTextBox1"];
var cookie = Request.Unvalidated().Cookies["mostRecentVisit"];
```

```
Response.AddHeader(name,value)
```

http szerver fejlécet ad a válaszba.

```
// A fejlécben elküld egy egyszerű hitelesítést a böngészőnek.
Response.AddHeader("WWW-Authenticate", "BASIC");
```

```
Response.OutputCache(seconds[,sliding][,varyByParams])
```

Egy megadott ideig elraktározza az oldal kimenetét. Lehetőség van beállítani minden oldal időtúllépését és a varyByParams értéket, hogy a lekérdezésekben minden egyes query string után más verziójú oldalt tároljunk el.

```
Response.OutputCache(60);
esponse.OutputCache(3600, true);
Response.OutputCache(10, varyByParams : new[]
{"category","sortOrder"});
```

```
Response.Redirect(path)
```

Átirányítja a böngészőben tett lekérdezést egy másik helyre.

```
Response.Redirect("~/Folder/File");
```

```
Response.SetStatus(httpStatusCode)
```

Beállítja a böngészőnek küldendő http státusz kódot.

```
Response.SetStatus(HttpStatusCode.Unauthorized);
Response.SetStatus(401);
```

```
Response.WriteBinary(data[,mimetype])
```

A válaszba beleírja az adatokat opcionális MIMI típussal.

```
Response.WriteBinary(image, "image/jpeg");
```

```
Response.WriteFile(file)
```

Egy fájl tartalmát beleteszi a válaszba.

```
Response.WriteFile("file.ext");
```

```
@section(sectionName){content}
```

(Szerkezeti oldalak) Meghatároz egy névvel ellátott tartalmi részt.

```
@section header { <div>Header text</div> }
```

```
Server.HtmlDecode(htmlText)
```

Egy HTML kódolt stringet dekódol.

```
var htmlDecoded = Server.HtmlDecode("&lt;html&gt;");
```

```
Server.HtmlEncode(text)
```

HTML markup rendereléséhez lekódol egy stringet.

```
var htmlEncoded = Server.HtmlEncode("<html>");
```

```
Server.MapPath(virtualPath)
```

A virtuális elérési utat fizikai elérési úttá alakítja.

```
var dataFile = Server.MapPath("~/App_Data/data.txt");
```

```
Server.UrlDecode(urlText)
```

Egy URL-ből dekódol szöveget.

```
var urlDecoded = Server.UrlDecode("url%20data");
```

```
Server.UrlEncode(text)
```

Lekódol egy szöveget, hogy beletehessük egy URL-be.

```
var urlEncoded = Server.UrlEncode("url data");
```

```
Session[key]
```

Beállít vagy beolvas egy olyan értéket, ami egészen addig létezik amíg be nem zárjuk a böngészőt.

```
Session["FavoriteColor"] = "red";
```

```
ToString()
```

Egy objektum értékét stringként jeleníti meg.

```
<p>It is now @DateTime.Now.ToString()</p>
```

```
UrlData[index]
```

Többletadatot olvas ki egy URL-ből(pl: /MyPage/ExtraData).

```
var pathInfo = UrlData[0];
```

```
WebSecurity.ChangePassword(userName,currentPassword,newPassword)
```

Egy megadott felhasználónak megváltoztatja a jelszavát.

```
var success = WebSecurity.ChangePassword("my-username","current-
password", "new- password");
```

```
WebSecurity.ConfirmAccount(accountConfirmationToken)
```

Az account confirmation token(megerősítő token) használatával megerősít egy felhasználói fiókot.

```
var confirmationToken = Request.QueryString["ConfirmationToken"];
```

```
if(WebSecurity.ConfirmAccount(confirmationToken)) {
      //...
}
```

```
WebSecurity.CreateAccount(userName, password[,
requireConfirmationToken=true|false])
```

A megadott felhasználónévvel és jelszóval létrehoz egy új felhasználói fiókot. Ha a requireConfirmationToken paraméterét igazra(true) állítjuk, akkor szükség van még egy megerősítő tokenre (confirmation token) is.

```
WebSecurity.CreateAccount("my-username", "secretpassword");
```

```
WebSecurity.CurrentUserId
```

Az éppen bejelentkezett felhasználó interger azonosítóját olvassa be.

```
var userId = WebSecurity.CurrentUserId;
```

```
WebSecurity.CurrentUserName
```

Az éppen bejelentkezett felhasználó nevét olvassa be.

```
var welcome = "Hello " + WebSecurity.CurrentUserName;
```

```
WebSecurity.GeneratePasswordResetToken(username[,tokenExpirationInMin
utesFromNow])
```

Létrehoz egy olyan jelszó-visszaállító tokent, amit el lehet küldeni a felhasználónak, hogy átállíthassa a jelszavát.

```
var resetToken = WebSecurity.GeneratePasswordResetToken("my-
username");
var message = "Visit http://example.com/reset-password/" + resetToken
+
    " to reset your password";
WebMail.Send(..., message);
```

```
WebSecurity.GetUserId(userName)
```

A megadott felhasználói névből visszaadja a felhasználó ID-jét.

```
var userId = WebSecurity.GetUserId(userName);
```

```
WebSecurity.IsAuthenticated
```

Ha az aktuális felhasználó be van jelentkezve, akkor visszatérési értéke igaz (true) lesz.

```
if(WebSecurity.IsAuthenticated) {...}
```

```
WebSecurity.IsConfirmed(userName)
```

Ha a felhasználó meg van erősítve, akkor visszatérési értéke igaz (true) lesz.(pl. megerősítő e-mail)

```
if(WebSecurity.IsConfirmed("joe@contoso.com")) { ... }
```

```
WebSecurity.IsCurrentUser(userName)
```

Igazat ad vissza, ha az aktuális felhasználó felhasználóneve megegyezik a megadott felhasználónévvel.

```
if(WebSecurity.IsCurrentUser("joe@contoso.com")) { ... }
```

```
WebSecurity.Login(userName,password[,persistCookie])
```

Egy cookieba elhelyez egy hitelesítő token-t, és belépteti a felhasználót.

```
if(WebSecurity.Login("username", "password")) { ... }
```

```
WebSecurity.Logout()
```

Eltávolítja a hitelesítő token sütit, és ezzel együtt kilépteti a felhasználót.

```
WebSecurity.Logout();
```

```
WebSecurity.RequireAuthenticatedUser()
```

A http státuszt 401-re (Jogosulatlan) állítja, ha a felhasználó nincs hitelesítve.

```
WebSecurity.RequireAuthenticatedUser();
```

```
WebSecurity.RequireRoles(roles)
```

Ha a felhasználó nem tagja valamelyik megadott csoportnak, akkor beállítja a http státuszt 401-re(Jogosulatlan).

```
WebSecurity.RequireRoles("Admin", "Power Users");
```

```
WebSecurity.RequireUser(userId) WebSecurity.RequireUser(userName)
```

Ha az aktuális felhasználó nem egyezik meg a felhasználó által megadott felhasználónévvel, akkor a HTML státuszkódot 401(Jogosulatlan)-re állítja.

```
WebSecurity.RequireUser("joe@contoso.com");
```

```
WebSecurity.ResetPassword(passwordResetToken, newPassword)
```

Ha a jelszó-visszaállító token(password reset token) érvényes,megváltoztatja a felhasználó jelszavát a megadottra.

```
WebSecurity.ResetPassword( "A0F36BFD9313", "new-password")
```

Adatok

```
Database.Execute(SQLstatement[,parameters]
```

Lefuttat egy SQL utasítást opcionális paraméterekkel együtt, majd az érintett rekordok számát visszaadja. Az utasítások lehetnek INSERT, DELETE vagy UPDATE.

```
db.Execute("INSERT INTO Data (Name) VALUES ('Smith')");
db.Execute("INSERT INTO Data (Name) VALUES (@0)", "Smith");
```

```
Database.GetLastInsertId()
```

Visszaadja a legutoljára beillesztett sorból az ID oszlopot.

```
db.Execute("INSERT INTO Data (Name) VALUES ('Smith')");
var id = db.GetLastInsertId();
```

```
Database.Open(filename) Database.Open(connectionStringName)
```

Megnyit egy megadott adatbázis fájlt, vagy azt, amit a Web.config fájlból vett elnevezett connection string ad meg.

```
// Figyeljük, meg hogy nincs kiterjesztés megadva.
var db = Database.Open("SmallBakery");
// Opens SmallBakery.sdf in App_Data
// Megnyitunk egy adatbázist egy elnevezett connection stringgel.
var db = Database.Open("SmallBakeryConnectionString");
```

```
Database.OpenConnectionString(connectionString)
```

Megnyit egy connection stringgel meghatározott adatbázist. Ez abban tér el a DataBase.Opentől, hogy az egy elnevezett connection stringet használ.

```
var db =
Database.OpenConnectionString("Data Source=|DataDirectory|\SmallBakery.sdf");
```

```
Database.Query(SQLstatement[,parameters])
```

Lekérdezi az adatbázist SQL utasítások használatával, majd a result változóba beletölti az eredményt.

```
foreach (var result in db.Query("SELECT * FROM PRODUCT"))
{<p>@result.Name</p>}
foreach (var result = db.Query("SELECT * FROM PRODUCT WHERE Price >
@0", 20)
{ <p>@result.Name</p> }
```

```
Database.QuerySingle(SQLstatement[,parameters])
```

Lefuttat egy SQL utasítást (paraméterek átadásával), majd visszaad egyetlen rekordot.

```
var product = db.QuerySingle("SELECT * FROM Product WHERE Id = 1");
var product = db.QuerySingle("SELECT * FROM Product WHERE Id = @0",
1);
```

```
Database.QueryValue(SQLstatement[,parameters])
```

Lefuttat egy SQL utasítást(paraméterek átadásával), és visszaad egyetlen értéket.

```
var count = db.QueryValue("SELECT COUNT(*) FROM Product");
var count = db.QueryValue("SELECT COUNT(*) FROM Product WHERE Price
> @0", 20);
```

Helperek

```
Analytics.GetGoogleHtml(webPropertyId)
```

Lerendereli a Google Analytics JavaScriptet a megadott ID-nek.

```
@Analytics.GetGoogleHtml("MyWebPropertyId")
```

```
Analytics.GetStatCounterHtml(project,security)
```

Lerendereli a StatCounter Analytics JavaScriptet a megadott projektnek.

```
@Analytics.GetStatCounterHtml(89, "security")
```

```
Analytics.GetYahooHtml(account)
```

Lerendereli a Yahoo Analytics JavaScriptet a megadott felhasználói fióknak.

```
@Analytics.GetYahooHtml("myaccount")
```

```
Bing.SearchBox([siteUrl])
```

Elküld egy akár URL-t is tartalmazó keresést a Bingnek.

```
@Bing.SearchBox() @* Searches the web.*@
@Bing.SearchBox(siteUrl: "www.asp.net") @* Searches the www.asp.net
site. *@
```

```
Bing.AdvancedSearchBox([siteUrl][,siteName][,
boxWidth][,resultWidth][,resultHeight][, themeColor][,locale])
```

Megjeleníti a Bing keresés eredményét a megadott formázással és URL-lel együtt.

```
@Bing.AdvancedSearchBox(
siteUrl: "www.asp.net",
siteName: "ASP.NET Custom Search",
boxWidth: 300,
resultWidth: 600,
resultHeight: 900,
themeColor: "Green",
locale: "en-US")
```

```
Chart(width,height[,template][,templatePath])
```

Inicializál egy grafikont.

```
@{
var myChart = new Chart(width: 600, height: 400);
}
```

```
Chart.AddLegend([title][,name])
```

Magyarázó feliratot ad a grafikonhoz.

```
@{
var myChart = new Chart(width: 600, height: 400)
    .AddLegend("Basic Chart")
    .AddSeries(
        name: "Employee",
        xValue: new[] {  "Peter", "Andrew", "Julie", "Mary", "Dave"
},   yValues: new[] { "2", "6", "4", "5", "3" })
    .Write();
}
```

```
Chart.AddSeries([name][,chartType][,chartArea][,axisLabel][, legend]
[, markerStep][,xValue] [,xField][,yValues][,yFields][,options])
```

Értékeket ad a grafikonhoz.

```
@{
var myChart = new Chart(width: 600, height: 400)
    .AddSeries(
        name: "Employee",  xValue: new[] {  "Peter", "Andrew",
"Julie", "Mary", "Dave" },
        yValues: new[] { "2", "6", "4", "5", "3" })
    .Write();
}
```

```
Crypto.Hash(string[,algorithm]) Crypto.Hash(bytes[,algorithm])
```

Visszaadja a megadott adat hash-ét. Az "sha256".az alapértelmezett algoritmus.

```
@Crypto.Hash("data")
```

```
Facebook.LikeButton(url[,action][, width][,layout][, showFaces][,
colorScheme])
```

A Facebook felhasználóknak enged kapcsolatot kiépíteni oldalakkal.

```
@Facebook.LikeButton("www.ASP.net")
```

```
FileUpload.GetHtml([initialNumberOfFiles][,allowMoreFilesToBeAdded][,
includeFormTag] [,addText][,uploadText])
```

Lerenderel egy felhasználói felületet fájlok feltöltéséhez.

```
@FileUpload.GetHtml(initialNumberOfFiles:1,
allowMoreFilesToBeAdded:false, includeFormTag:true,
uploadText:"Upload")
```

```
GamerCard.GetHtml(gamerTag)
```

Lerendereli a megadott Xbox játékos kártyát.

```
@GamerCard.GetHtml("joe")
```

```
Gravatar.GetHtml(email[,imageSize],[,defaultImage],[,
rating],[,imageExtension] [,attributes])
```

Lerenderel egy Gravatar képet a megadott e-mail címnek.

```
@Gravatar.GetHtml("joe@contoso.com")
```

```
Json.Encode(object)
```

Egy adat objektumból JavaScript Object Notation (JSON) formátumú stringet konvertál.

```
var myJsonString = Json.Encode(dataObject);
```

```
Json.Decode(string)
```

Egy JSON kódolt stringet adat objektummá konvertál.

```
var myJsonObj = Json.Decode(jsonString);
```

```
LinkShare.GetHtml(pageTitle[,
pageLinkBack][,twitterUserName][,additionalTweetText] [,linkSites])
```

Lerenderel egy közösségi portál linket a megadott címből és Url-ből.

```
@LinkShare.GetHtml("ASP.NET Web Pages Samples")
@LinkShare.GetHtml("ASP.NET Web Pages Samples", "http://www.asp.net")
```

```
ModelStateDictionary.AddError(key,errorMessage)
```

Társít egy hibaüzenetet egy űrlap mezővel. Használjuk a ModelState helpert, hogy elérjük ezt a tagot!

```
ModelState.AddError("email", "Enter an email address");
```

```
ModelStateDictionary.AddFormError(errorMessage)
```

Társít egy hibaüzenetet egy űrlappal. Használjuk a ModelState helpert hogy elérjük ezt a tagot.

```
ModelState.AddFormError("Password and confirmation password do not
match.");
```

```
ModelStateDictionary.IsValid
```

Ha nincs semmilyen validációs hiba, akkor igazat ad vissza. Használjuk a ModelState helpert, hogy elérjük ezt a tagot!

```
if (ModelState.IsValid) { // Save the form to the database }
```

```
ObjectInfo.Print(value[,depth][,enumerationLength])
```

Lerendereli egy objektum tulajdonságait és értékeit egy objektumnak

```
@ObjectInfo.Print(person)
```

```
Recaptcha.GetHtml([,publicKey][, theme][,language][, tabIndex])
```

Lerenderel egy reCAPTCHA verifikációs tesztet.

```
@ReCaptcha.GetHtml()
```

```
ReCaptcha.PublicKey
ReCaptcha.PrivateKey
```

Beállítja a publikus és privát kulcsokat a reCAPTCHA-hoz. Általában ezeket az _AppStart oldalon állítjuk be.

```
ReCaptcha.PublicKey = "your-public-recaptcha-key";
ReCaptcha.PrivateKey = "your-private-recaptcha-key";
```

```
ReCaptcha.Validate([,privateKey])
```

Visszaadja a reCAPTCHA teszt eredményét.

```
if (ReCaptcha.Validate()){
// Sikeres teszt.
}
```

```
ServerInfo.GetHtml()
```

Státusz információt renderel ASP.NET weboldalakról.

```
@ServerInfo.GetHtml()
```

```
Twitter.Profile(twitterUserName)
```

Egy Twitter csatornát renderel a megadott felhasználóhoz.

```
@Twitter.Profile("billgates")
```

```
Twitter.Search(searchQuery)
```

Egy Twitter csatornát renderel a megadott kereső szöveghez.

```
@Twitter.Search("asp.net")
```

```
Video.Flash(filename[,width,height])
```

Egy Flash lejátszót renderel a megadott fájlhoz. Méreteit megszabhatjuk.

```
@Video.Flash("test.swf", "100", "100")
```

```
Video.MediaPlayer(filename[, width,height])
```

Egy Windows Media lejátszót renderel a megadott fájlhoz. Méreteit megszabhatjuk.

```
@Video.MediaPlayer("test.wmv", "100", "100")
```

```
Video.Silverlight(filename,width,height)
```

Egy Silverlight lejátszót renderel a megadott .xap fájlhoz. Méreteit megszabhatjuk.

```
@Video.Silverlight("test.xap", "100", "100")
```

```
WebCache.Get(key)
```

Visszaadja a kulcs által meghatározott objektumot vagy null-t, ha az nem található.

```
var username = WebCache.Get("username")
```

```
WebCache.Remove(key)
```

A kulcs által meghatározott objektumot kiveszi a gyorsítótárból.

```
WebCache.Remove("username")
```

```
WebCache.Set(key,value[,minutesToCache][,slidingExpiration])
```

A kulcs által meghatározott névvel egy értéket tesz a gyosítótárba.

```
WebCache.Set("username", "joe@contoso.com ")
```

```
WebGrid(data)
```

A lekérdezésből vett adatokkal egy új WebGrid objektumot hoz létre.

```
var db = Database.Open("SmallBakery");
var grid = new WebGrid(db.Query("SELECT * FROM Product"));
```

```
WebGrid.GetHtml()
```

Egy HTML táblázatokban adatok megjelenítésére szolgáló markupot renderel.

```
@grid.GetHtml()// The 'grid' variable is set when WebGrid is created.
```

```
WebGrid.Pager()
```

Lerenderel egy pager-t a WebGrid objektumnak.

```
@ grid.Pager() // A grid változó akkor jön létre, amikor a WebGrid
létrejön.
```

```
WebImage(path)
```

Betölt egy képet a megadott útvonalról.

```
var image = new WebImage("test.png");
```

```
WebImage.AddImagesWatermark(image)
```

A megadott képet vízjelként adja hozzá egy képhez.

```
WebImage photo = new WebImage("test.png");
WebImage watermarkImage = new WebImage("logo.png");
photo.AddImageWatermark(watermarkImage);
```

```
WebImage.AddTextWatermark(text)
```

A megadott szöveget hozzáadja egy képhez.

```
image.AddTextWatermark("Copyright")
```

```
WebImage.FlipHorizontal() WebImage.FlipVertical()
```

Megfordítja a képet horizontálisan vagy vertikálisan.

```
image.FlipHorizontal();
image.FlipVertical();
```

```
WebImage.GetImageFromRequest()
```

Betölti a feltöltött képet.

```
var image = WebImage.GetImageFromRequest();
```

```
WebImage.Resize(width,height)
```

Átméretez egy képet.

```
image.Resize(100, 100);
```

```
WebImage.RotateLeft() WebImage.RotateRight()
```

Balra vagy jobbra forgatja a képet.

```
image.RotateLeft();
image.RotateRight();
```

```
WebImage.Save(path[,imageFormat])
```

A megadott elérési útra elmenti a képet.

```
image.Save("test.png");
```

```
WebMail.Password
```

Beállítja az SMTP szerver jelszavát. Ezt általában az _AppStart oldalon szoktuk megtenni.

```
Webmail.Password = „password”;
```

```
WebMail.Send(to,subject,body[, from][,
cc][,filesToAttach][,isBodyHtml] [,additionalHeaders])
```

Elküld egy e-mail-t.

```
WebMail.Send("touser@contoso.com", "subject", "body of message",
"fromuser@contoso.com");
```

```
WebMail.SmtpServer
```

Beállítja az SMTP szerver nevét. Ezt általában az _AppStart oldalon szoktuk megtenni.

```
WebMail.SmtpServer = "mailServer";
```

```
WebMail.UserName
```

Beállítja az SMTP szerver felhasználó nevét. Ezt általában az _AppStart oldalon szoktuk megtenni.

```
WebMail.UserName = "Joe";
```

Függelék – ASP.NET Weboldalak Visual Basickel

Ahogy láthattuk, az ASP.NET Razor syntax kódok ebben a könyvben C#-on alapulnak. De a Razor syntax Visual Basicet is támogat. Ahhoz hogy egy ASP.NET weblapot Visual Basicben hozzunk létre, egy olyan weboldalt készítünk aminek *.vbhtml* a kiterjesztése, és ebbe írjuk bele a Visual Basic kódot. Ebben a függelékben áttekintjük, hogyan használjuk a Visual Basic programozási nyelvet és annak szintaxisát, hogy ASP.NET weboldalakat hozzunk létre.

A fejezetből megismerhetjük:

- A 8 legjobb programozási tippet és trükköt.
- A Visual Basic programozási nyelvet és annak szintaxisát.

A 8 legjobb programozási tipp és trükk

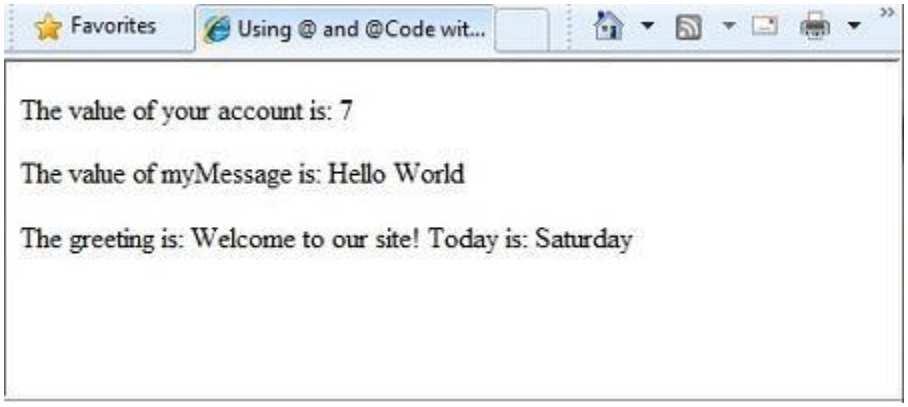
Ebből a részből megtudhatjuk azokat a dolgokat, amiket muszáj tudnunk, hogy elkezdhessünk ASP.NET Razor syntax szerver kódot írni.

1. Kódot a @ jel után adhatunk az oldalhoz.

A @ jel segítségével beágyazott kifejezéseket és egy vagy több soros parancsblokkok kezdetét jelezzük.

```
<!-- Egysoros parancsblokkok  -->
@Code  Dim total = 7  End Code
@Code  Dim myMessage = "Hello World" End Code

<!-- Beágyazott kifejezések -->
<p>The value of your account is: @total </p>
<p>The value of myMessage is: @myMessage</p>

<!-- Többsoros parancsblokk -->
@Code
    Dim greeting = "Welcome to our site!"
    Dim weekDay = DateTime.Now.DayOfWeek
    Dim greetingMessage = greeting & " Today is: " &
weekDay.ToString()
End Code
<p>The greeting is: @greetingMessage</p>
```

Az eredmény így jelenik meg a böngészőben:



HTML kódolás

Ahogy az előző példákban láthattuk, ha a @ jelet használjuk az oldalon, akkor az ASP.NET HTML kódolja a kimenetet. Ugyanis felcseréli a lefoglalt HTML karaktereket (mint például a <,> és az &)

olyan kódokkal, melyek engedélyezik, hogy a karakterek normálisan jelenjenek meg, nem pedig HTML címkeként vagy entitásként. A HTML kódolás nélkül a kódunk kimenete lehet, hogy nem fog jól megjelenni, és az oldalunkat biztonsági kockázatoknak tehetjük ki.

Ha esetlegesen HTML kódot szeretnénk megjeleníteni (pl `<p></p>` a bekezdésekhez vagy `<em></em>` hogy kiemeljünk egy szöveget), lásd: Szöveg, Markup és Kód a Kódblokkban.

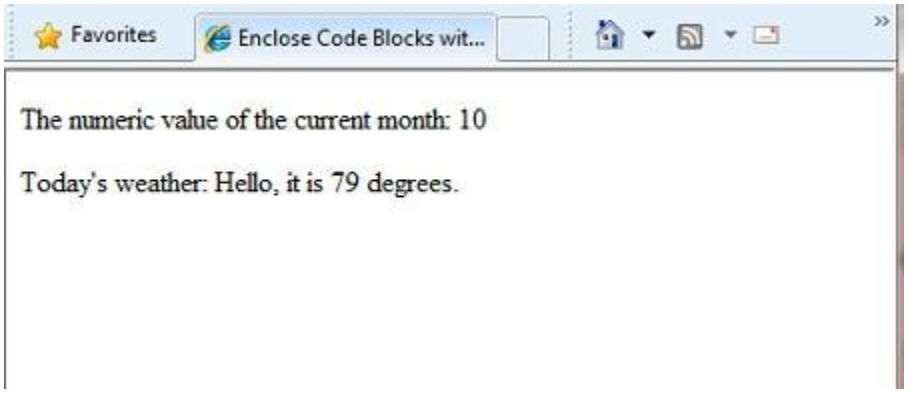
A HTML kódolásról bővebben lásd: 5 .Fejezet – Munka az űrlapokkal

2. A kódblokkokat Code és End Code közé tegyük

A kódblokkokban egy vagy több utasítást találunk és `Code`, illetve `End Code` kulcsszavakkal jelezzük őket. A kódblokk kezdetét jelölő `Code` kulcsszót használjuk közvetlenüla `@` jel után, ugyanis nem lehet üres hely közöttük!

```
<!-- Egysoros parancsblokk.  -->
@Code
    Dim theMonth = DateTime.Now.Month
End Code
<p>The numeric value of the current month: @theMonth</p>

<!-- Egysoros parancsblokk. -->
@Code
    Dim outsideTemp = 79
    Dim weatherMessage = "Hello, it is " & outsideTemp & " degrees."
End Code
<p>Today's weather: @weatherMessage</p>
```

Az eredmény így jelenik meg a böngészőben:



3. Egy kódblokkon belül az utasításokat egy sortöréssel zárjuk le

Egy Visual Basic kódblokkban minden utasítást egy sortöréssel zárunk le. (A későbbiekben megtanuljuk, hogy lehet egy hosszú utasítást több sorba írni, ha szükséges.)

```
<!-- Egysoros parancsblokk. -->
@Code
    Dim theMonth = DateTime.Now.Month
End Code

<!-- Többsoros parancsblokk. -->
@Code
    Dim outsideTemp = 79
    Dim weatherMessage = "Hello, it is " & outsideTemp & " degrees."
End Code

<!-- Beágyazott kifejezés, ezért nem kell sortörés. -->
<p>Today's weather: @weatherMessage</p>
```

4. Értékek tárolására használjunk változókat!

Értékeket változókban tárolhatunk. Ezek lehetnek stringek, számok, dátumok,…stb. Új változót a Dim kulcsszóval hozhatunk létre. Ahhoz, hogy a változókat közvetlenül beillesszük az oldalunkba, használjuk a @ jelet.

```
<!-- String tárolása -->
@Code
    Dim welcomeMessage = "Welcome, new members!"
End Code
<p>@welcomeMessage</p>

<!-- Dátum tárolása -->
@Code
    Dim year = DateTime.Now.Year
End Code

<!-- Egy változó megjelenítése -->
<p>Welcome to our new members who joined in @year!</p>
```

Az eredmény így jelenik meg a böngészőben:

5. Stringekhez a szöveget idézőjelek közé tegyük!

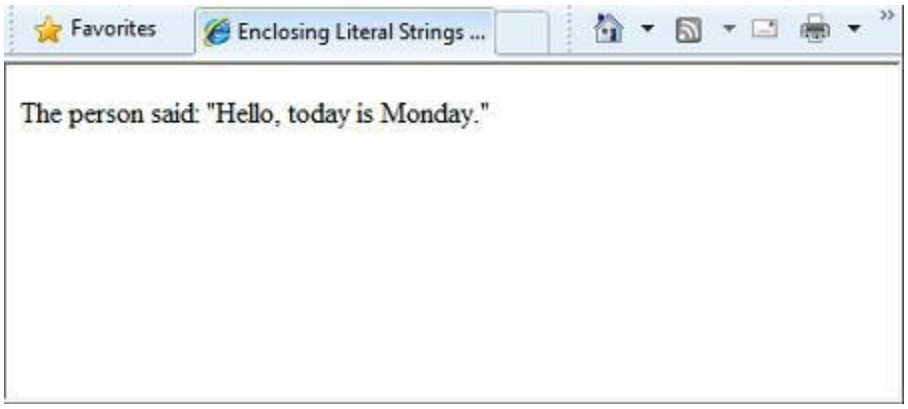
A string olyan karaktersorozat, amit szövegként kezelünk. Hogy meghatározzunk egy stringet, idézőjelek közé kell tegyük.

```
@Code
    Dim myString = "This is a string literal"
End Code
```

Hogy dupla idézőjelet tegyünk a stringünkbe, használjunk két dupla idézőjelet! Ha azt szeretnénk, hogy az idézőjelek megjelenjenek az oldalon, így használjuk őket: "" megjelenítendő szöveg"", ha esetleg dupla idézőjelet szeretnénk használni, akkor """"-t használjunk!

```
<!-- Dupla idézőjel használata egy stringben -->
@Code
    Dim myQuote = "The person said: ""Hello, today is Monday."""
End Code
<p>@myQuote</p>
```

Az eredmény így jelenik meg a böngészőben:



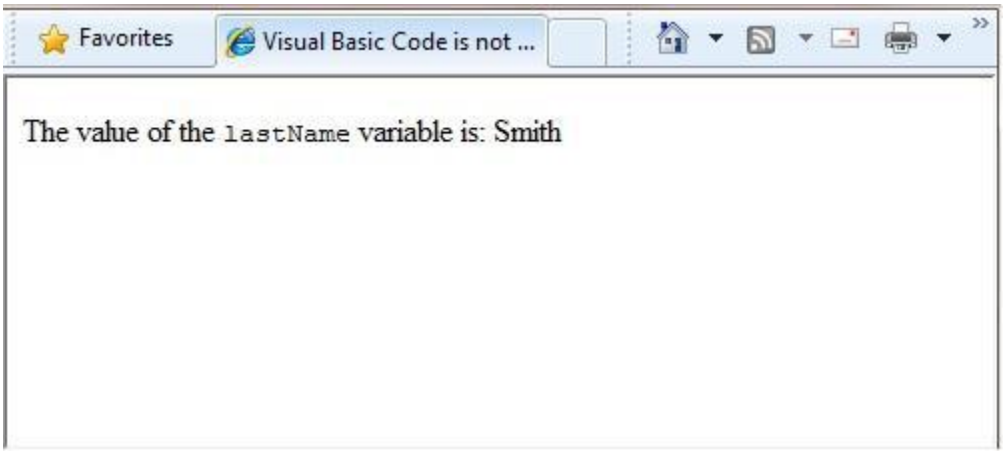
6. A Visual Basic kód nem case sensitive.

A Visual Basic programozási nyelv nem különbözteti meg a kis- és nagybetűket. A programozási kulcsszavak (`Dim`,`If`,`True`) és a változó nevek (`myString`) kis- és nagybetűvel is írhatóak.

A következő példában a `lastname` változót kisbetűsen deklaráljuk, de az oldalunkon nagybetűvel használjuk.

```
@Code
    Dim lastName = "Smith"
    ' A kulcsszavaknál, mint pl. a Dim, nem kell a kis- és
nagybetűkre figyelni.
    DIM someNumber = 7
End Code
<p>The value of the <code>lastName</code> variable is: @LASTNAME</p>
```

Az eredmény így jelenik meg a böngészőben.



7. Az oldalunk kódolása objektumokkal való munkát igényel

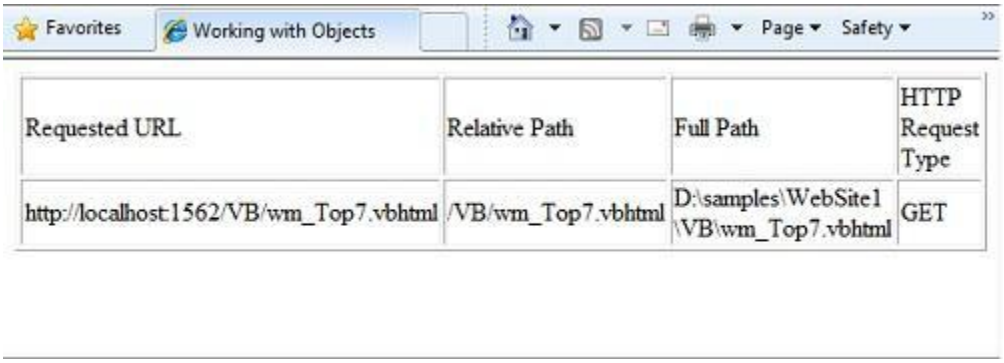
Objektumnak nevezünk olyan dolgokat, amiket a programozás során felhasználunk (oldal, szövegdoboz, fájl, kép, webes lekérés, e-mail, egy adatbázis sor… stb.) Egy objektum jellemzőit a tulajdonságain keresztül tudhatjuk meg. (Pl: egy szövegdoboznak van egy `Text` tulajdonsága, egy lekérésnek van egy `URL` tulajdonsága, egy e-mailnek van `From` tulajdonsága, az ügyfél objektumnak pedig `FirstName` tulajdonsága). Az objektumok természetesen rendelkeznek metódusokkal is, ezeket „igékként” láthatjuk. Erre jó példa a fájl objektum `Save` metódusa, egy kép objektum `Rotate` metódusa, vagy egy e-mail objektum `Send` metódusa.

Valószínűleg sűrűn fogunk dolgozni `Request` objektummal, ami egy űrlap mezőiben található értékeket adja vissza (szövegdoboz…stb.), vagy esetleg azt, hogy milyen típusú böngésző indította meg a lekérdezést, vagy a lekért oldal URL-jét,esetlegesen azonosítja a felhasználót, és még sok minden másra képes. A következő példa bemutatja, hogyan érjük el a `Request` objektum

tulajdonságait és hogyan hívjuk meg a `MapPath` nevű metódusát. Ezzel megismerhetjük egy oldal szerveroldali abszolút elérési útját.

```
<table border="1">
    <tr>
        <td>Requested URL</td>
        <td>Relative Path</td>
        <td>Full Path</td>
        <td>HTTP Request Type</td>
    </tr>
    <tr>
        <td>@Request.Url</td>
        <td>@Request.FilePath</td>
        <td>@Request.MapPath(Request.FilePath)</td>
        <td>@Request.RequestType</td>
    </tr>
</table>
```

Az eredmény így jelenik meg a böngészőben:

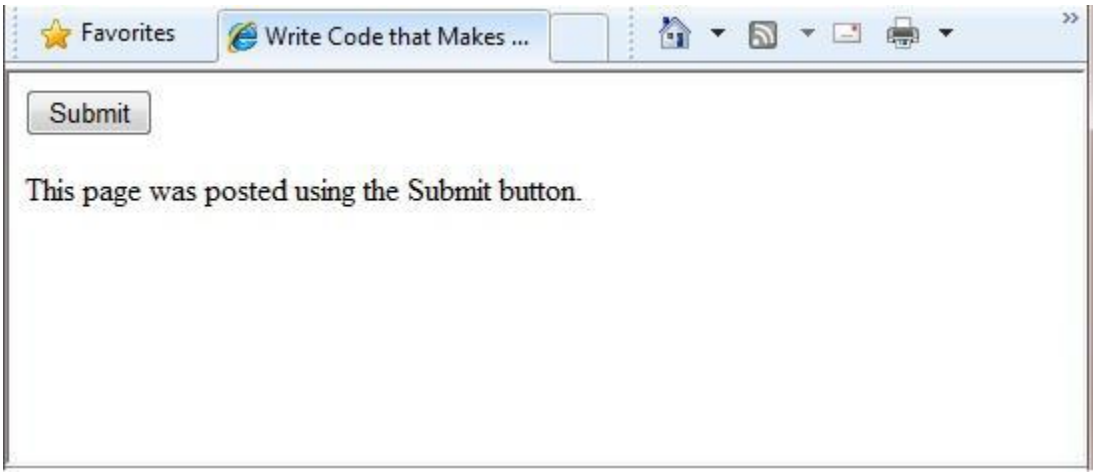


8. Írhatunk olyan kódot, ami helyettünk dönt.

A dinamikus weboldalak egyik kulcstulajdonsága, hogy képesek vagyunk eldönteni, mi történjen különböző helyzetekben. Ennek a legáltalánosabb módja a ha (`if`) parancs (és az opcionális vagy (`else`) parancs).

```
@Code
   Dim result = ""
   If IsPost Then
      result = "This page was posted using the Submit button."
   Else
      result = "This was the first request for this page."
   End If
End Code
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8" />
        <title>Write Code that Makes Decisions</title>
    </head>
<body>

    <form method="POST" action="" >
        <input type="Submit" name="Submit" value="Submit"/>
        <p>@result</p>
    </form>

</body>
</html>
```

Az `If IsPost` utasítás az `If IsPost = True` utasítás rövidítése. Az If utasítással sokféle módon tesztelhetünk különféle eseteket, ismételhetünk kódblokkokat. Ezekről a későbbiekben többet megtudhatunk:



HTTP GET és POST metódusok és az IsPost tulajdonság

A HTTP protokoll nagyon kevés olyan metódust támogat, amivel lekérdezünk a szerverről. A két legtöbbször használt a GET, amivel beolvashatunk egy oldalt, a másik pedig a POST, amivel pedig elküldhetünk egy oldalt. Általában amikor a felhasználó először lekér egy oldalt, akkor a lekérdezés a GET metódust használja. Ha a felhasználó kitölt egy űrlapot és ráklikkel az elküldre, akkor a böngésző egy POST lekérdezést küld a szervernek.

A web programozásban nem árt tudnunk, hogy egy oldalt éppen GET vagy POST metódussal kérnek le, ugyanis ebből tudhatjuk meg, hogyan dolgozzuk fel az oldalt. Az ASP.NET weboldalaknál használhatjuk az `IsPost` tulajdonságot, hogy megtudjuk a lekérdezés GET vagy POST típusú-e, ugyanis ha a lekérdezés POST típusú, akkor az `IsPost` tulajdonság visszatérési értéke igaz (true) lesz, ekkor megtehetünk olyan dolgokat, mint például egy szövegdoboz értékének kiolvasása. Ebben a könyvben sok olyan példát találunk, ami bemutatja, hogyan dolgozzunk fel egy weboldalt az `IsPost` tulajdonságtól függően.

Egyszerű példakódok

A következő példákból megismerhetjük az alapszintű programozási technikákat. Ebben a példában egy olyan weboldalt készítünk, amiben a felhasználó megad két számot, majd az oldal összeadja és megjeleníti az eredményt.

1. Hozzunk létre egy *AddNumbers.vbhtml* fájlt!
2. Másoljuk bele a következő kódot és markupot, felülírva bármit, amit az oldal tartalmazott!

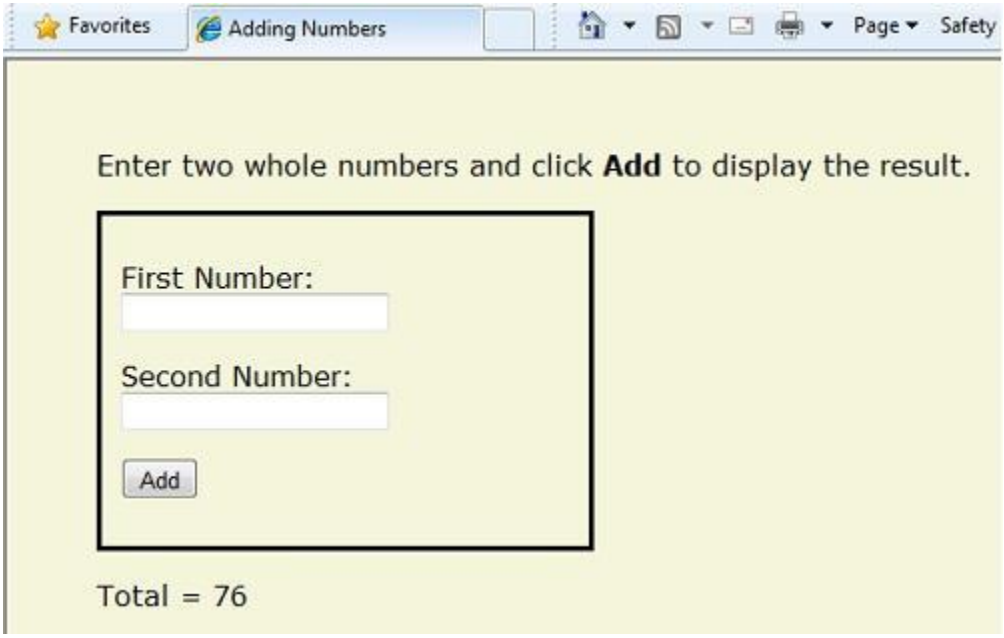
```
@Code
    Dim total = 0
    Dim totalMessage = ""
    if IsPost Then
        ' Retrieve the numbers that the user entered.
        Dim num1 = Request("text1")
        Dim num2 = Request("text2")
        ' Convert the entered strings into integers numbers and add.
        total = num1.AsInt() + num2.AsInt()
        totalMessage = "Total = " & total
    End If
End Code
```

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8" />
        <title>Adding Numbers</title>
        <style type="text/css">
            body {background-color: beige; font-family: Verdana,
Ariel;
                margin: 50px;
                }
            form {padding: 10px; border-style: solid; width: 250px;}
        </style>
    </head>
<body>
    <p>Enter two whole numbers and click <strong>Add</strong> to
display the result.</p>
    <p></p>
    <form action="" method="post">
    <p><label for="text1">First Number:</label>
    <input type="text" name="text1" />
    </p>
    <p><label for="text2">Second Number:</label>
    <input type="text" name="text2" />
    </p>
    <p><input type="submit" value="Add" /></p>
    </form>
    <p>@totalMessage</p>
</body>
</html>
```

Pár dolog, amire figyelni kell:

- A @ jel kezdi meg az oldalunk első kódblokkját, de megtalálható a `totalMessage` beágyazott változónk előtt is.
- Az oldal elején található kódblokk `Code` és `End Code` közé van zárva.
- A `total`, `num1`, és `num2` változók különféle számokat tárolnak, a `totalMessage` pedig szöveget tárol.
- A szövegérték, amit a `totalMessage` változóban tárolunk, idézőjelek közé van téve.
- Mivel a Visual Basic nem különböztet meg kis és nagy betűket, amikor az oldal vége felé használjuk a `totalMessage` változót, csak arra kell ügyelnünk, hogy a betűzése ugyanaz legyen, mint ahogy az oldal elején deklaráltuk.
- A `num1.AsInt()+num2.AsInt()` kifejezés bemutatja, hogyan dolgozzunk objektumokkal és metódusokkal. Az `AsInt` metódus átalakítja a változókban található szöveg értéket számmá (intergerré), amiket ezek után össze lehet adni.
- A `<form>` címke tartalmaz egy `method=”post”` tulajdonságot. Ez határozza meg, mikor klikkel a felhasználó az Add(összeadás) gombra,ekkor az oldalt elküldjük a szervernek a HTTP POST metódus segítségével. Ha elküldtük az oldalt, és az `If IsPost` visszatérési értéke igaz (true) lesz, akkor lefut az oldal kódja és megjeleníti az összeadott számokat.

3. Mentsük el az oldalt, és nyissuk meg egy böngészőben! (Bizonyosodjunk meg arról, hogy az oldal ki van választva a *Files* névtérben!) Írjunk be két egész számot, majd klikkeljünk az *Add* gombra!



A Visual Basic programozási nyelv és szintaxisa

A 2. fejezetben egyszerű példán keresztül láthattuk hogyan kell ASP.NET weboldalt készíteni, és hogyan adhatunk hozzá szerver kódot a HTML markuphoz. A könyvnek ebben a részében megtanuljuk, hogyan írjunk Razor szintaxist használó ASP.NET szerver kódot Visual Basicben.

Ha már van némi tapasztalatunk a programozás terén (főként, ha már programoztunk C-ben, C++-ban, C#-ban, Visual Basicben, vagy esetleg JavaScriptben), akkor az itt olvasható anyag nagyrészt ismerős lehet. Ha így van, akkor valószínűleg csak azt kell megszoknunk, hogy a *.vbhtml* fájloknál hogyan adjuk hozzá a markuphoz a WebMatrix kódot.

Alap szintaxis

Szöveg, markup és kód egyesítése kódblokkokban

A szerver kódblokkokban sűrűn szeretnénk kiíratni az oldalra a szövegeinket és a markupot. Ha a kódblokkunk tartalmaz szöveget, amit szövegként szeretnénk megjeleníteni, akkor az ASP.NET-nek képesnek kell lennie megkülönböztetni a szöveget a kódtól. Ezt különböző módokon érhetjük el.

- Tegyük a szöveget HTML elemek közé, mint például `<p></p>` vagy `<em></em>`:

```
@If IsPost Then
    ' Ebben a sorban minden tartalom <p> címkék között van.
    @<p>Hello, the time is @DateTime.Now and this page is a
postback!</p>
Else
    ' Itt a szövegek vannak címkék között, majd csak utána a szerver
kód.
    @<p>Hello, <em>Stranger!</em> today is: </p> @DateTime.Now
End If
```

  Egy HTML elem tartalmazhat szöveget, még több HTML elemet és kódot. Ha az ASP.NET érzékeli a kezdő HTML címkét, akkor lekódol mindent, amit az elem tartalmaz, és elküldi a böngészőnek.(Természetesen, ha még tartalmaz kódot, akkor azt is végrehajtja.)

- Használjuk a `@:` operátort vagy a `<text>` elemet. A `@:` segítségével egy sor szöveget vagy párosítatlan HTML címkéket írathatunk ki. A `<text>` elem segítségével több sort is

kiírathatunk. Ezek a módszerek akkor hasznosak, ha egy HTML elemet a kimeneten nem akarunk megjeleníteni.

```
@If IsPost Then
    ' Sima szöveg párosítatlan HTML címkével és kóddal.
    @:The time is: <br /> @DateTime.Now
    ' Kód aztán sima szöveg, párosított címkék,és még több szöveg.
    @DateTime.Now @:is the <em>current</em> time.
End If
```

A következő példa megegyezik az előzővel, csak itt `<text>` címkét használunk.

```
@If IsPost Then
    @<text>
    The time is: <br /> @DateTime.Now
    @DateTime.Now is the <em>current</em> time.
    </text>
End If
```

A következő példában használunk szöveget, párosítatlan HTML címkéket, programkódot és párosított HTML címkéket, amiket `<text>` és `</text>` közé zárunk. Természetesen a sorokat külön-külön írva is működne a kód, csak használni kellene a @ operátort minden sor előtt.

```
@Code
    dim minTemp = 75
    @<text>It is the month of @DateTime.Now.ToString("MMMM"), and
    it's a <em>great</em> day! <p>You can go swimming if it's at
    least @minTemp degrees.</p></text>
End Code
```

Megjegyzés: Ha a szöveget HTML elemmel, `@` operátorral vagy `<text>` elemmel jelenítjük meg, az ASP.NET nem HTML kódolja. (Természetesen ahogy már megjegyeztük, az ASP.NET a kód kifejezéseket és a `@` után írt kód blokkokat lekódolja, kivételek az itt felsorolt speciális esetek.)

Üres helyek

A plusz üres helyek a programkódban (stringen kívűl) nem változtatják annak lefutását.

```
@Code Dim personName=   "Smith"     End Code
```

Hosszú utasítások több sorba tördelése

Egy hosszú parancsot a sor végére elhelyezett alulvonás _ karakterrel tudunk több sorba tördelni (Visual Basicben *folytatás karakter*). Ahhoz, hogy használjuk a sortörést, az _ elé tegyünk egy szóközt, majd folytassuk a parancsot a következő sorban! Az olvashatóság érdekében a parancsokat annyi sorba tördelhetjük, amennyibe csak szeretnénk.

```
@Code
    Dim familyName _
    ="Smith"
End Code

@Code
    Dim _
    theName _
    = _
    "Smith"
End Code
```

Arra vigyázzunk, hogy a string szövegeket nem tördelhetjük több sorba! A következő példa ezt a hibát mutatja be.

```
@Code
    ' Doesn't work.
```

```
    Dim test = "This is a long _
      string"
End Code
```

Ha esetleg mégis szükség lenne arra, hogy egy hosszú szöveget több sorba tördeljünk, az „és” (&) jelet kell használnunk. Erről a fejezet egy későbbi részében olvashatunk.

Megjegyzések a kódban

A kódjainkban elhelyezhetünk megjegyzéseket magunknak vagy másoknak. A Razor syntax megjegyzések @*-gal kezdődnek és *@-cal zárulnak.

```
@* Ez egy egysoros megjegyzés. *@

@*
    Ez egy többsoros megjegyzés.
    A sorok száma nincs megszabva.
*@
```

A kódblokkokban használhatjuk a Razor syntax megjegyzéseket vagy közönséges Visual Basic jelölést, ami egy szimpla idézőjel (’) minden sor elején.

```
@Code
    ' Minden sor elejére tegyünk egy ’-t megjegyzés készítéséhez.
End Code

@Code
    ' A Visual Basicben nincs többsoros megjegyzés.
    ' Ezért használunk minden megjegyzés előtt egy ’ jelet.
End Code
```

Változók

A változó adatok/értékek tárolására alkalmas objektum. Egy változó neve akármi lehet, csak arra kell figyelni, hogy betűvel kezdődjön, és nem tartalmazhat szóközt vagy előre lefoglalt karaktereket. Ahogy már az előzőekben láthattuk, nem számít, hogy kis- vagy nagybetűsek.

Változók és adattípusok

A változóknak lehet egy megadott adattípusuk, amik megadják, hogy milyen adatokat tartalmazhatnak. A string változók szövegesek (pl ’’Helló világ”), az integer változókban egész számokat találhatunk (pl 3 vagy 79), a date változókban pedig dátum értékeket tárolhatunk különböző formátumokban (pl 4/12/2010 vagy március 2009). Természetesen még sokféle adattípust használhatunk. Azonban egy változónak nem kötelező típust megadni. A legtöbb esetben az ASP.NET kitalálja, hogy a változó milyen adatként van használva. (Esetenként meg kell adnunk egy típust, a könyvben található példákból kiderül, mikor.)

Hogy deklaráljunk egy típus nélküli változót, használjuk a `Dim` kulcsszót és írjuk utána a változó nevét (Pl `Dim myVar`). Ahhoz, hogy típussal rendelkező változót hozzunk létre használjuk a Dim kulcsszót és írjuk utána a változó nevét, majd használjuk az `As` kulcsszót és utána írjuk a típus nevét (Pl `Dim myVar As String`).

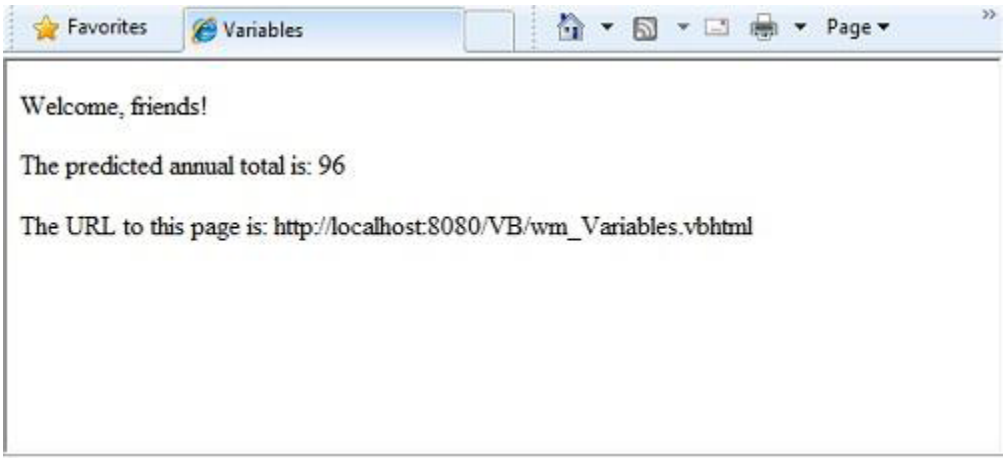
```
@Code
    ' Szöveget adunk meg a változónak.
    Dim greeting = "Welcome"

    ' Számot adunk meg a változónak.
    Dim theCount = 3

    ' Egy kifejezést adunk meg a változónak.
    Dim monthlyTotal = theCount + 5

    ' Dátumot adunk meg a változónak.
```

```
    Dim today = DateTime.Today

    ' Az aktuális oldal URL-jét adjuk meg a változónak.
    Dim myPath = Request.Url

    ' Változók deklarálása explicit adattípusokkal.
    Dim name as String = "Joe"
    Dim count as Integer = 5
    Dim tomorrow as DateTime = DateTime.Now.AddDays(1)
End Code
```

A következő példában olyan beágyazott kifejezéseket láthatunk, amik felhasználják a weboldal változóit.

```
@Code
    ' Változók értékeinek beágyazása a HTML markupba.
    ' A markup elé tegyünk @-ot, mert ez egy kódblokk.
    @<p>@greeting, friends!</p>
End Code

<!-- Változó használata beágyazott kifejezésben. -->
<p>The predicted annual total is: @( monthlyTotal * 12)</p>

<!-- Az oldal URL-jének megjelenítése változóval. -->
<p>The URL to this page is:  @myPath</p>
```

Az eredmény így jelenik meg a böngészőben:



Adattípusok konvertálása és tesztelése

Annak ellenére, hogy az ASP.NET általában automatikusan meg tudja határozni az adattípusokat, néha mégsem sikerül. Ilyenkor besegítünk egy explicit átkonvertálással. Még akkor is, ha nincs szükség átkonvertálásra, néha hasznos lehet, ha tudjuk, hogy milyen típusú adatokkal dolgozunk. A legáltalánosabb eset, hogy string-et konvertálunk át más típussá, mint például integerre, vagy datere. A következő példa egy tipikus esetet mutat be, amiben stringből számot csinálunk.

```
@Code
    Dim total = 0
    Dim totalMessage = ""
    if IsPost Then
        ' Retrieve the numbers that the user entered.
        Dim num1 = Request("text1")
        Dim num2 = Request("text2")
        ' Convert the entered strings into integers numbers and add.
        total = num1.AsInt() + num2.AsInt()
        totalMessage = "Total = " & total
```

```
    End If
End Code
```

A felhasználó minden adatot stringként ad meg. Még akkor is, ha a felhasználótól számot kérünk, és ők azt is írtak be, az elküldött adat string lesz. Ebből adódik, hogy azt számmá kell konvertálnunk. Mivel az ASP.NET nem tud két stringet összeadni, ha a példánkban össze akarnánk adni az értékeinket anélkül, hogy, átkonvertálnánk őket, a következő hibaüzenetet kapnánk.

```
Cannot implicitly convert type 'string' to 'int'.
```

Az értékek integerré konvertálásához az `AsInt` metódust használjuk. Ha a konvertálás sikeres, össze tudjuk adni a két számot.

A következő táblázat tartalmazza a változók gyakori konvertálásait és tesztelési metódusait.

Metódus	**Leírás**	**Példa**
`AsInt(), IsInt()`	Egy stringet, ami egy egész számnak felel meg átkonvertál integerré	`Dim myIntNumber = 0` `Dim myStringNum = "539"` `If myStringNum.IsInt() Then` `  myIntNumber = myStringNum.AsInt()` `End If`
`AsBool(), IsBool()`	Átkonvertál egy igaz vagy hamis stringet boolean típussá.	`Dim myStringBool = "True"` `Dim myVar = myStringBool.AsBool()`
`AsFloat(), IsFloat()`	Egy string-ből, aminek decimális értéke van (1,3) float-ot (lebegőpontos) konvertál.	`Dim myStringFloat = "41.432895"` `Dim myFloatNum = myStringFloat.AsFloat()`
`AsDecimal(), IsDecimal()`	Átkonvertál egy decimális értékű stringet (1,3) egy decimális számmá. (Az ASP.NET-ben a decimális szám pontosabb, mint a float.	`Dim myStringDec = "10317.425"` `Dim myDecNum = myStringDec.AsDecimal()`
`AsDateTime(), IsDateTime()`	Átkonvertál egy stringet ami dátum és idő értékű az ASP.NET `DateTime` típussá.	`Dim myDateString = "12/27/2010"` `Dim newDate = myDateString.AsDateTime()`
`ToString()`	Bármilyen adatot stringgé konvertál.	`Dim num1 As Integer = 17` `Dim num2 As Integer = 76` `' myString is set to 1776` `Dim myString as String = num1.ToString() & num2.ToString()`

Operátorok

Az operátor olyan kulcsszó vagy karakter, ami megmondja az ASP.NET-nek, hogy milyen parancsot hajtson végre egy-egy kifejezésben. A Visual Basic sokféle operátort támogat, de csak párat kell ismernünk ahhoz, hogy elkezdhessünk ASP.NET weboldalakat készíteni. A következő táblázatban megismerhetjük a leggyakoribb operátorokat.

Operátor	**Leírás**	**Példák**
.	Pont. Arra használjuk, hogy megkülönböztessük az objektumokat és azok tulajdonságait, illetve metódusait.	`Dim myUrl = Request.Url` `Dim count = Request("Count").AsInt()`
()	Zárójelek. Kifejezéseket csoportosíthatunk velük, paramétereket adhatunk át metódusoknak, és tömbök, listák, gyűjtemények elemeit érhetjük el.	`@(3 + 7)` `@Request.MapPath(Request.FilePath)`
=	Értékadás és egyenlőség. Kontextustól függően vagy a kifejezés jobb oldalán található értéket a kifejezés bal oldalán található objektumnak értékül adjuk, vagy egyenlőséget vizsgálunk közöttük.	`Dim age = 17` `Dim income = Request("AnnualIncome")`
Not	Tagadás. Az igaz (true) értéket megfordítja hamisra (false) és fordítva. Általában gyors módszerként használjuk hamisság vizsgálatára (vagyis tagadva igaz).	`Dim taskCompleted As Boolean = False` `' Processing.` `If Not taskCompleted Then` `  ' Continue processing` `End If`
<>	Egyenlőtlenség. Visszatérési értéke igaz(true), ha az értékek nem egyenlők.	`Dim theNum = 13` `If theNum <> 15 Then` `  ' Do something.` `End If`
<	Kisebb, mint	`If 2 < 3 Then`

`>` `<=` `>=`	Nagyobb mint… Kisebb vagy egyenlő, Nagyobb vagy egyenlő..	``` ' Do something. End If Dim currentCount = 12 If currentCount >= 12 Then ' Do something. End If```
`+` `–` `*` `/`	Matematikai operátorok, számolásoknál használjuk őket.	```@(5 + 13) Dim netWorth = 150000 Dim newTotal = netWorth * 2 @(newTotal / 2)```
`&`	Összeláncolás. Stringeket fűzünk össze vele.	```' The displayed result is "abcdef". @("abc" & "def")```
`AndAlso` `OrElse`	Logikai ÉS és VAGY művelet, amikkel feltételeket kötünk össze.	```Dim myTaskCompleted As Boolean = false Dim totalCount As Integer = 0 ' Processing. If (Not myTaskCompleted) AndAlso totalCount < 12 Then ' Continue processing. End If```
`+=` `-=`	A növelés és csökkentés operátorok. Hozzáadunk vagy kivonunk 1-et egy változó értékéből.	```Dim theCount As Integer = 0 theCount += 1 ' Adds 1 to count```

Fájlok és mappák használata a kódban

A kódjainkban sűrűn fogunk használni fájl és mappa elérési utakat. Ebben a példában egy weboldal fizikai mappastruktúráját láthatjuk, ahogy megjelenhet a weblapot fejlesztő számítógépen.

```
C:\WebSites\MyWebSite
    default.cshtml
    datafile.txt
    \images
        Logo.jpg
    \styles
        Styles.css
```

Egy webszerveren, minden weboldalnak van egy virtuális mappastruktúrája, ami megfelel az oldalunk fizikai mappastruktúrájának. Alapértelmezetten a virtuális és a fizikai mappanevek megegyeznek. A virtuális gyökérkönyvtárat egy perjel (/) jelöli, pont úgy, mint ahogy a számítógépünkön C: meghajtó gyökérkönyvtárát fordított perjel (\) jelöli. (A virtuális mappa elérési utakat mindig sima perjellel jelöljük.) A következő példában bemutatjuk a virtuális és fizikai elérési útját az előző példában látható *StyleSheet.css* fájlnak.

- Fizikai elérési út: *C:\WebSites\MyWebSiteFolder\styles\StyleSheet.css*
- Virtuális elérési út (a virtuális gyökérkönyvtártól /): */styles/StyleSheet.css*

Amikor fájlokkal és mappákkal dolgozunk a kódunkban, az alkalmazott objektumtól függően néha virtuális, néha fizikai elérési utakat használunk. Az ASP.NET-ben sokféle módszer áll rendelkezésünkre, hogy fájlokat és mappákat használjunk a kódunkban. Ezek lehetnek: a ~ operátor, a `Server.MapPath` metódus és a `Href` metódus.

A ~ operátor: a virtuális gyökérkönyvtár beolvasása

Hogy kódunkban megadjuk a fájljaink és mappáink virtuális gyökerét, használjuk a ~ operátort! Ez azért hasznos, mert így szabadon mozgathatjuk a weboldalunkat anélkül, hogy a kódban szereplő elérési utakat megzavarnánk.

```
@Code
    Dim myImagesFolder = "/images"
    Dim myStyleSheet = "/styles/StyleSheet.css"
End Code
```

A Server.MapPath metódus: A virtuális elérési út fizikaivá konvertálása

A Server.MapPath metódus egy virtuális elérési útból (pl */default.cshtm*) abszolút fizikai elérési utat (pl *C:\WebSites\MyWebSiteFolder\default.cshtml*) konvertál. Ezt a metódust olyan feladatok elvégzésére használhatjuk, amik teljesítéséhez fizikai elérési útra van szükség, mint például egy a webszerverünkön található szövegfájl olvasása vagy írása.(Egy kiszolgáló szerveren általában nem ismerjük weboldalunk fizikai elérési útját.) A metódus úgy működik, hogy egy fájl vagy mappa virtuális elérési útját átadjuk neki, és visszatérési értékként megadja a fizikai elérési utat.

```
@Code
    Dim dataFilePath = "/dataFile.txt"
End Code

<!-- Megjeleníti a fizikai elérési utat:
C:\Websites\MyWebSite\datafile.txt  -->
<p>@Server.MapPath(dataFilePath)</p>
```

A Href metódus: Elérési utak létrehozása az oldal forrásaihoz

A `WebPage` objektum `Href` metódusa átkonvertálja azokat az elérési utakat, amiket a szerverkódunkban hozunk létre (tartalmazhat ~ operátort), olyan elérési utakká, amiket a böngészőnk értelmezni tud. (A böngésző nem tudja értelmezni a ~ operátort mivel az szigorúan ASP.NET operátor. Arra is használhatjuk még a `Href` metódust, hogy elérési utakat készítsünk olyan forrásokhoz, mint például képfájlok, más weboldal, és CSS fájlok. Például használhatjuk ezt a metódust arra, hogy egy HTML markupban attribútumokat adjuk egy `<img>`,`<link>`, és egy `<a>` elemnek.

```
@Code
    Dim myImagesFolder = "/images"
    Dim myStyleSheet = "/styles/StyleSheet.css"
End Code

<!-- Ezt az elérési utat hozza létre"../images/Logo.jpg" az src
attribútumban. -->
<img src="@Href(myImagesFolder)/Logo.jpg" />

<!--  Ugyanaz, csak itt ~-t használunk -->
<img src="@Href("/images")/Logo.jpg" />

<!--  CSS fájlhoz hoz létre egy linket. -->
<link rel="stylesheet" type="text/css" href="@Href(myStyleSheet)" />
```

Logikai feltételek és ciklusok

ASP.NET kódban írhatunk olyan parancsokat, amik bizonyos feltételektől függenek, vagy akár olyan kódot, ami egy meghatározott alkalommal megismétli önmagát.

Feltételek tesztelése

Hogy egyszerű feltételeket teszteljünk, használjuk az If…Then parancsot! Ennek visszatérési értéke lehet igaz vagy hamis a feltételünktől függően.

```
@Code
    Dim showToday = True
    If showToday Then
        DateTime.Today
    End If
End Code
```

A teszt blokk az `If` kulcsszóval kezdődik. A valódi tesztelés (feltétel) az `If` kulcsszó után van, ennek lehet a visszatérési értéke igaz vagy hamis. Ezután következik a `Then` kulcsszó, ez lesz az a rész, ami lefut, ha a feltételünk (ami `If` és `End if` közé van zárva) visszatérési értéke igaz lesz. Az `If` parancs egy `Else` ágat is tartalmazhat, arra az esetre, ha feltételünk visszatérési értéke hamis lenne.

```
@Code
    Dim showToday = False
    If showToday Then
        DateTime.Today
    Else
        @<text>Sorry!</text>
    End If
End Code
```

Ha a kódblokkunkat az `If` utasítással kezdjük, nincs szükség a megszokott Code…End Code páros használatára, elég, ha a@ jelet használjuk. Ez a megoldás nemcsak az `If`-fel működik, hanem minden olyan Visual Basic programozási kulcsszóval, amit kódblokk követ.(`For, For each, Do While…stb.`)

```
@If showToday Then
    DateTime.Today
Else
    @<text>Sorry!</text>
End If
```

Egy vagy több `ElseIf` blokk használatával akár több feltételt is megadhatunk.

```
@Code
    Dim theBalance = 4.99
    If theBalance = 0 Then
        @<p>You have a zero balance.</p>
    ElseIf theBalance > 0 AndAlso theBalance <= 5 Then
        ' If the balance is above 0 but less than
        ' or equal to $5, display this message.
        @<p>Your balance of $@theBalance is very low.</p>
    Else
        ' For balances greater than $5, display balance.
        @<p>Your balance is: $@theBalance</p>
    End If
End Code
```

A következő példában láthatjuk, hogy ha az `If` águnk feltétele nem teljesül, akkor az `ElseIf` águnk feltételét vizsgáljuk meg. Ha ez a feltétel teljesül, az `ElseIf` ág fog lefutni, ha azonban egyik feltétel se teljesül, akkor az `Else` ág. Egy `If` ág mellé végtelen számú `ElseIf` ágat használhatunk, az a lényeg, hogy egy `Else` ággal zárjuk le, ugyanis ez felel meg a ''bármi más'' feltételnek.

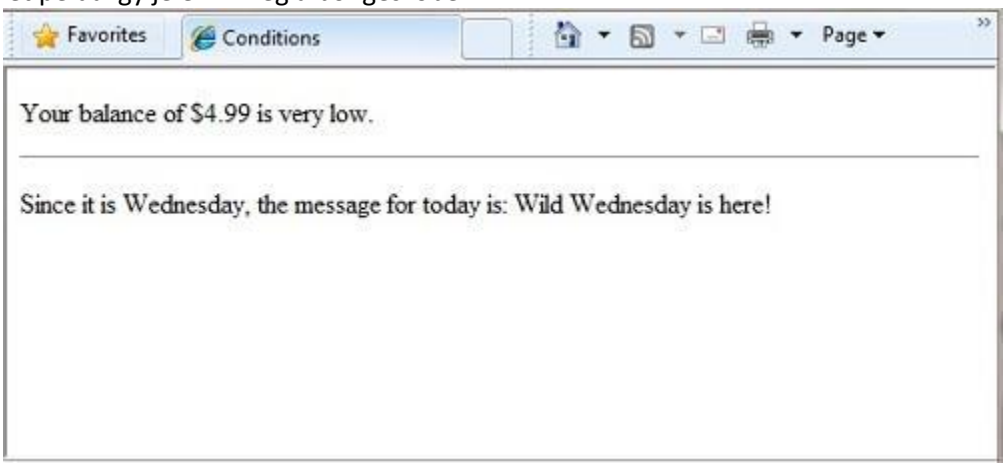
Sok feltétel esetén használjuk a `Select Case` blokkot:

```
@Code
    Dim weekday = "Wednesday"
```

```
    Dim greeting = ""

    Select Case weekday
        Case "Monday"
            greeting = "Ok, it's a marvelous Monday."
        Case "Tuesday"
            greeting = "It's a tremendous Tuesday."
        Case "Wednesday"
            greeting = "Wild Wednesday is here!"
        Case Else
            greeting = "It's some other day, oh well."
    End Select
End Code
<p>Since it is @weekday, the message for today is: @greeting</p>
```

Az ellenőrizendő változót időzőjelezni kell (A példánkban a weekday változó) Minden esetben egy Case parancsot használunk, amit a változó egy lehetséges értéke követ. Ha valamelyik érték megegyezik a letesztelt értékkel, akkor abban a `Case` ágban levő kód fut le.

Az utolsó két példa így jelenik meg a böngészőben:



Kód ismétlése (Ciklusok)

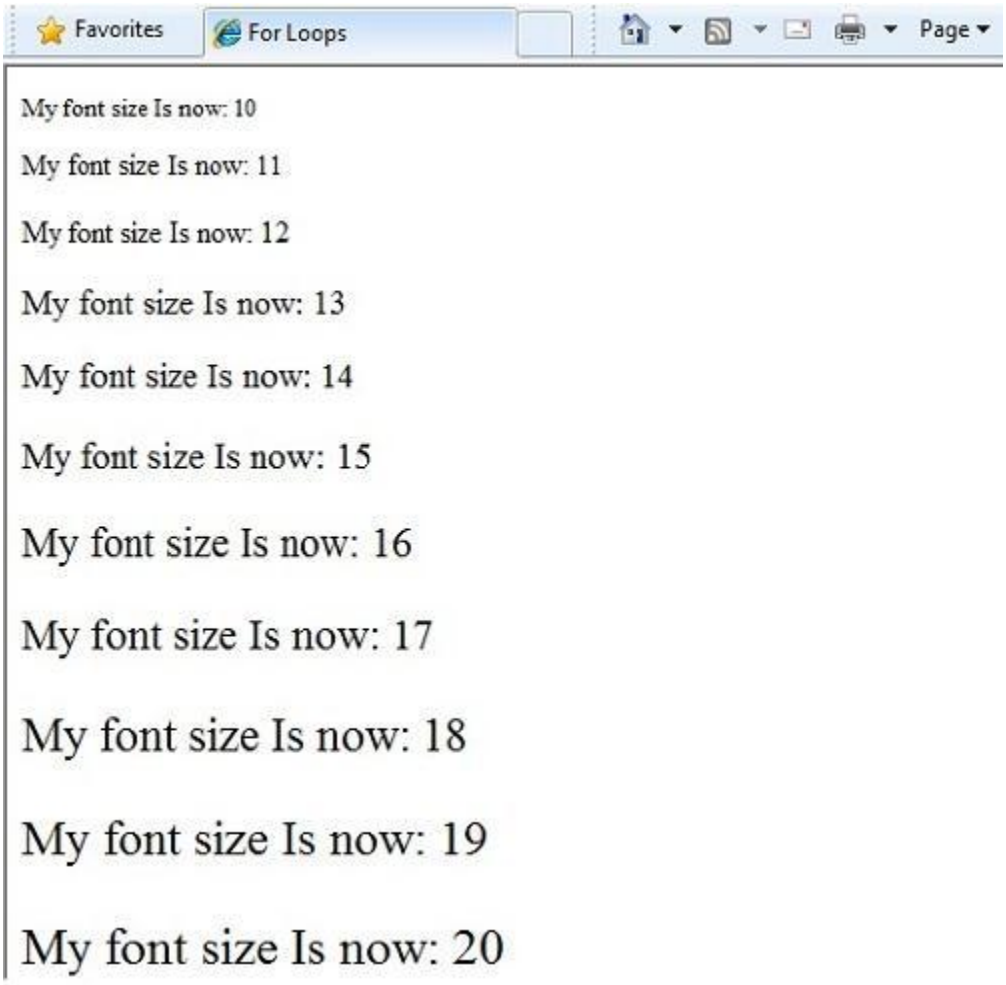
Sűrűn van arra szükség, hogy egy-egy parancsot többször is lefuttathassunk. Ezt ciklusok használatával érjük el. Például sokszor kell egy parancsot egy adattömb minden elemére lefuttatnunk. Ha tudjuk, hogy hányszor szeretnénk ismételni, akkor használjuk a `For` ciklust, mivel ez a ciklus nagyon hasznos ha előre vagy hátrafele szeretnénk számolni.

```
@For i = 10 To 20
    @<p>Item #: @i</p>
Next i
```

A ciklus a For kulcsszóval kezdődik, majd három fontos rész követi.

- `For` kulcsszó után deklarálunk egy számlálóként használatos változót (nincs szükség `Dim`-re), majd megadjuk a számlálás keretét, például `i=10 to 20`. Ez azt jelenti, hogy az `i` változónk 10-től 20-ig fog elszámolni.
- A `For` és a `Next` kulcsszavak között van a tartalmi rész. Ez egy vagy több parancsot tartalmazhat, ami lefut minden egyes ismétléskor.
- A `Next i` utasítás zárja le a ciklust. Megnöveli a számlálót és elindítja a ciklus következő iterációját.

A For cikluson belül a markup minden egyes iterációkor létre hoz egy új bekezdést (<p> elem) és beállítja a betűméretet az i (a számláló) értékére. Ha megnyitjuk ezt a példát, akkor 11 sornyi szöveget fogunk látni, amiben minden sor egy számmal nagyobb, mint az előző.



Ha tömbökkel vagy listákkal dolgozunk, akkor sűrűn fogunk For Each ciklust használni. A lista hasonló objektumok gyűjteménye, és a For Each segítségével feladatokat hajthatunk végre minden elemén. Ez a típusú ciklus azért hasznos a listáknál, mert nincs szükség számlálóra vagy beállított határra. Ehelyett a `For Each` végig megy a listán elejétől a végéig.

A következő példa visszaadja a `Request.ServerVariables` lista elemeit (információk a webszerverről). Arra használjuk a `For Each` ciklust, hogy a lista minden elemének a nevét megjelenítsük egy HTML listában egy új `<li>` elemként.

```
<ul>
@For Each myItem In Request.ServerVariables
    @<li>@myItem</li>
Next myItem
</ul>
```

A `For Each` kulcsszót egy változó követi, ami a lista egy elemének felel meg (pl `Myitem`), ezt követi az `In` kulcsszó, majd a lista, amin a ciklusunkat le szeretnénk futtatni. A cikluson belül a lista változóval mindig az éppen aktuális listaelemet érhetjük el.

- ALL_HTTP
- ALL_RAW
- APPL_MD_PATH
- APPL_PHYSICAL_PATH
- AUTH_TYPE
- AUTH_USER
- AUTH_PASSWORD
- LOGON_USER
- REMOTE_USER
- CERT_COOKIE
- CERT_FLAGS
- CERT_ISSUER
- CERT_KEYSIZE
- CERT_SECRETKEYSIZE
- CERT_SERIALNUMBER
- CERT_SERVER_ISSUER
- CERT_SERVER_SUBJECT
- CERT_SUBJECT
- CONTENT_LENGTH
- CONTENT_TYPE
- GATEWAY_INTERFACE
- HTTPS

A `Do While` utasítás segítségével sokkal általánosabb célú ciklusokat készíthetünk:

```
@Code
    Dim countNum = 0
    Do While countNum < 50
        countNum += 1
        @<p>Line #@countNum: </p>
    Loop
End Code
```

Ez a ciklus a `Do While`-lal kezdődik, majd megadjuk a feltételt, és ezt követi az ismételni akart blokk. A ciklusok egy számolásra használt változót vagy objektumot tipikusan vagy növelnek, vagy csökkentenek. A példánkban akárhányszor lefut a ciklus, a `+=` operátor mindig hozzáad 1-et a változóhoz. (Ha csökkenteni szeretnénk a változót, akkor a `-=` operátort kellene használni.)

Objektumok és gyűjtemények

Az ASP.NET weboldalak szinte minden része objektum, még maga a weboldal is. Ebből a részből megismerhetjük a fontosabb objektumokat.

Oldal objektumok

Az ASP.NET –ben a legalapvetőbb objektum az oldal. Egy oldal tulajdonságait közvetlenül elérhetjük, anélkül, hogy bármilyen más objektumot kellene használnunk, a következő kód a Request objektum segítségével beolvassa az oldal elérési útját:

```
@Code
    Dim path = Request.FilePath
End Code
```

A Page objektum tulajdonságaival sok információhoz juthatunk hozzá, mint például:

- `Request` (Kérelem). Ahogy már láthattuk, ez a gyűjtemény információkat tárol az éppen aktuális lekérdezésről, azt is beleértve, hogy milyen böngésző indította a lekérdezést, mi a megtekinteni kívánt oldal URL-je, ki a felhasználó… stb.
- `Response` (Válasz). Ez azoknak az információknak a gyűjteménye, amiket visszaküld az oldal a böngészőnknek, amikor lefutott a szerveroldali kód. Használhatjuk ezt a tulajdonságot például arra, hogy információkat helyezzünk el a válaszba.

```
@Code
    ' Access the page's Request object to retrieve the URL.
    Dim pageUrl = Request.Url
End Code
    <a href="@pageUrl">My page</a>
```

Gyűjtemény objektumok (Listák, tömbök)

Az ugyanolyan típusú objektumok csoportját nevezzük gyűjteménynek. Erre tökéletes példa a `Customer` (vásárlók) objektumok gyűjteménye egy adatbázisból. Az ASP.NET-ben sok beépített gyűjtemény találhatunk, mint például a Request.Files gyűjtemény.

Két megszokott gyűjtemény típus a tömb és a ''szótárérték''. A lista akkor hasznos, ha hasonló dolgokat akarunk együtt tárolni, de nem akarunk minden elemhez külön változót létrehozni.

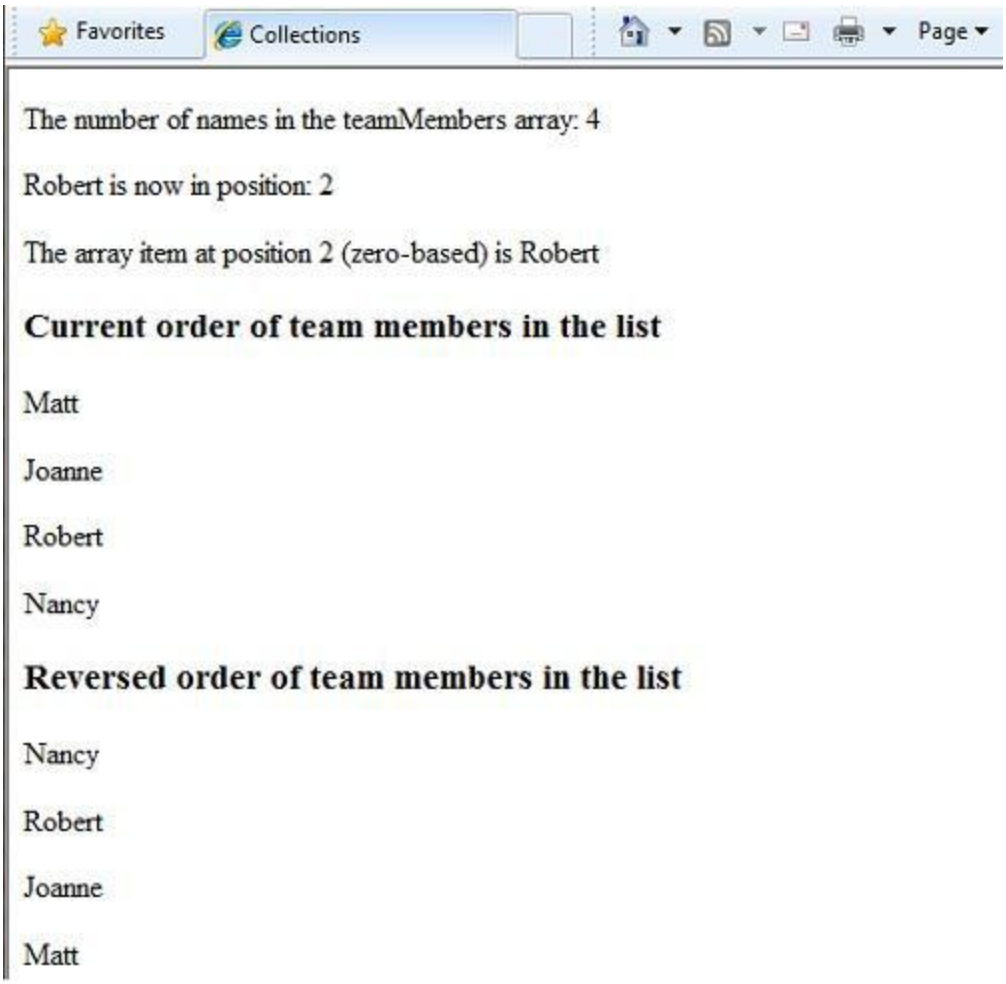
```
<h3>Team Members</h3>
@Code
    Dim teamMembers() As String = {"Matt", "Joanne", "Robert",
"Nancy"}
    For Each name In teamMembers
        @<p>@name</p>
    Next name
End Code
```

Minden tömbnek van egy megadott adattípusa, mint például `String`, `Integer` vagy `DateTime`. A változó neve után írt zárójelekkel jelezzük, hogy a változónk tömb (pl. `Dim myVar() as String`). A tömbben tárolt elemeket az indexüket használva érhetjük el, vagy egy `For Each` ciklussal. A tömbök indexei 0-val kezdődnek, vagyis a tömb első elemének az indexe 0, a másodiknak 1, és így tovább.

```
@Code
    Dim teamMembers() As String = {"Matt", "Joanne", "Robert",
"Nancy"}
    @<p>The number of names in the teamMembers array:
@teamMembers.Length </p>
    @<p>Robert is now in position: @Array.IndexOf(teamMembers,
"Robert")</p>
    @<p>The array item at position 2 (zero-based) is
@teamMembers(2)</p>
    @<h3>Current order of team members in the list</h3>
    For Each name In teamMembers
        @<p>@name</p>
    Next name
    @<h3>Reversed order of team members in the list</h3>
    Array.Reverse(teamMembers)
    For Each reversedItem In teamMembers
        @<p>@reversedItem</p>
    Next reversedItem
End Code
```

Egy tömb hosszát, vagyis hogy hány elemű, a `Length` tulajdonságából tudhatjuk meg. Ha szeretnénk egy megadott érték helyét meghatározni a tömbben, akkor az `Array.IndexOf` metódust használjuk. De ha úgy akarjuk, meg is fordíthatjuk a tömb tartalmát (`Array.Reverse` metódus) vagy akár szét is válogathatjuk (`Array.Sort` metódus).

A string tömbünk így jelenik meg a böngészőben:



A tár (szótárérték) nem más, mint kulcs/érték párok gyűjteménye, ezekben elég megadni a kulcsot (vagy nevet), hogy beállítsuk vagy beolvassuk a hozzátartozó értéket:

```
@Code
    Dim myScores = New Dictionary(Of String, Integer)()
    myScores.Add("test1", 71)
    myScores.Add("test2", 82)
    myScores.Add("test3", 100)
    myScores.Add("test4", 59)
End Code
<p>My score on test 3 is: @myScores("test3")%</p>
@Code
    myScores("test4") = 79
End Code
<p>My corrected score on test 4 is: @myScores("test4")%</p>
```

A `New` kulcsszóval jelezzük, hogy új Tár (szótárérték) objektumot akarunk létrehozni. A `Dim` kulcsszóval egy változóhoz fűzhetjük a tárunkat. A tárban használt adattípusokat zárójelek közé kell tennünk, és a deklarálás végére még egy zárójel párt kell tenni, mivel amit írtunk, igazából egy metódus, ami egy új tárat hoz létre.

Ahhoz, hogy új elemeket adjunk a tárunkhoz, meghívhatjuk a tár `Add` metódusát, és megadjuk hozzá a kulcs/érték párt. A következő példában láthatjuk az értékadás egy alternatíváját:

```
@Code
    myScores("test4") = 79
End Code
```

Hogy értéket olvassunk ki a tárunkból, zárójelezzük az elérni kívánt kulcsot:

```
@myScores("test4")
```

Paraméterekkel ellátott metódusok hívása

Ahogy a fejezet korábbi részeiben láthattuk, az objektumok metódusokkal rendelkezhetnek. Például a DataBase objektum tartalmazhat egy `DataBase.Connect` metódust. Léteznek olyan metódusok, amiknek egy vagy több paramétere is van. A paraméter olyan érték, amit azért adunk át egy metódusnak, hogy az sikeresen le tudjon futni. Erre jó példa a `Request.MapPath` metódus, ami három paraméterrel is rendelkezik.

```
Public Overridable Function MapPath (virtualPath As String, _
    baseVirtualDir As String, _
    allowCrossAppMapping As Boolean)
```

Ez a metódus visszaadja a megadott virtuális elérési útnak megfelelő fizikai elérési utat. A metódus három paramétere a `virtualPath`, `baseVirtualDir` és az `allowCrossAppMapping`. (Megjegyzés: a deklarálásnál a paraméterek azokkal az adattípusokkal jelennek meg, amiket elfogadnak.) Amikor meghívjuk ezt a metódust, mindhárom paraméterét meg kell adnunk.

Amikor Razor szintaxissal használjuk a Visual Basicet, kétféle módon adhatunk át paramétereket, *helyzeti és névszerinti* paraméterekként. Hogy helyzeti paraméterek használatával hívjunk meg egy metódust, a metódus deklarálásnál látható sorrendben kell megadni a paramétereket (a metódus dokumentációjából kiderül). Ehhez a sorrendhez mindenképp tartanunk kell magunkat, nem cserélhetjük fel a paramétereket, és nem hagyhatunk ki paramétereket (ha mégis szükséges, akkor null értéket vagy üres stringet ("") adjunk neki).

A következő példa feltételezi, hogy rendelkezünk a web oldalunkon egy *script* mappával. A kód meghívja a `Request.MapPath` metódust, és átadja a három paramétert a megfelelő sorrendben, majd megjeleníti a végeredményt.

```
@Code
    ' Pass parameters to a method using positional parameters.
    Dim myPathPositional = Request.MapPath("/scripts", "/", true)
End Code
<p>@myPathPositional</p>
```

Amikor egy metódus sok paraméterrel rendelkezik, név szerinti paraméterek használatával tisztábban és könnyen olvashatóbban tarthatjuk a kódunkat. Hogy név szerinti paraméterekkel hívjunk meg egy metódust, adjuk meg a paraméter nevét, majd egy := jel után adjuk meg az értékét. A névszerinti paraméterek előnye, hogy akármilyen sorrendben megadhatjuk őket.(Hátránya hogy a metódushívás sokkal hosszabb lesz (méretben).)

Az előző példában látható metódust most névszerinti paraméterekkel és eltérő sorrendben hívjuk meg:

```
@Code
    ' Pass parameters to a method using named parameters.
    Dim myPathNamed = Request.MapPath(baseVirtualDir:= "/",
allowCrossAppMapping:= true, virtualPath:= "/scripts")
End Code
<p>@myPathNamed</p>
```

Mint láthatjuk, a paramétereket más sorrendben adtuk át, ennek ellenére ha lefuttatjuk az előző két példát, ugyan azt az értéket fogják visszaadni.

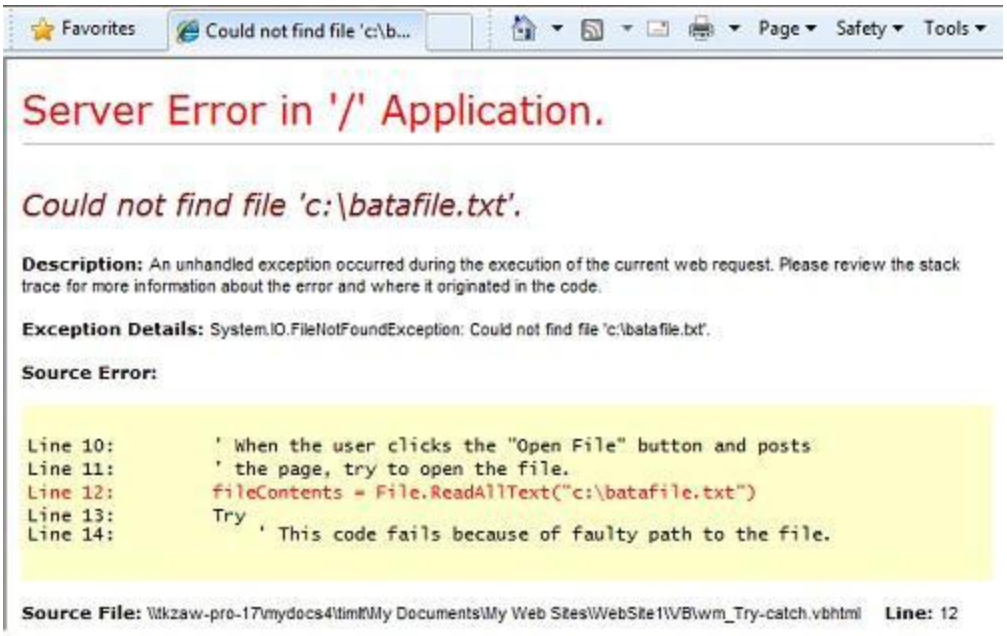
Hibák kezelése

Try-Catch utasítások

Lehetséges, hogy a kódunk tartalmaz olyan utasításokat, amik valamiféle hibát generálnak. Például:

- Sokféle hiba léphet fel, amikor egy fájlon valamiféle műveletet szeretnénk végrehajtani. Lehet, hogy a fájl nem is létezik vagy csak olvasható, esetleg a kódunk nem tartalmazza a megfelelő jogosultságokat, és így tovább.
- Hasonlóképpen lehetnek jogosultsági problémák, ha egy adatbázis rekordjain szeretnénk változtatni, de az is lehet, hogy elveszítjük a kapcsolatot az adatbázissal, vagy egyszerűen csak nem megfelelő adatokat akarunk elmenteni, és így tovább.

Ezeket a szituációkat programozási körökben *kivételeknek (exception)* hívjuk. A kódunk automatikusan generál (dob) egy hibaüzenetet, ha kivételbe ütközik.



Használjuk `Try/Catch` utasításokat, amikor kivételekbe ütközünk, vagy csak el akarjuk kerülni az ilyen típusú hibaüzeneteket. A `Try` blokkba azt a kódot írjuk, amit ellenőrizni szeretnénk, majd ehhez párosíthatunk egy vagy több `Catch`-t, ahol különböző kivételeket kezelhetünk le. Valójában annyi `Catch` blokkot használhatunk, ahány hibára számítunk.

Megjegyzés: Nem ajánlatos a `Response.Redirect` metódust használni egy `Try/Catch` utasításban, mert alapból létrehozhat egy kivételt az oldalunkban.

A következő példában megalkotunk egy oldalt, ami az első lekérdezéskor létrehoz egy szövegfájlt és megjelenít egy gombot, amivel a felhasználó megnyithatja azt. A példában szándékosan használunk rossz fájlnevet, hogy létrehozzunk egy kivételt. A kód két kivételt kezel: Az egyik a `FileNotFoundException` vagyis a nem létező fájl kivétel, ami akkor történik, ha a megadott fájlnév hibás vagy nem létezik. A második a `DirectoryNotFoundException` vagyis a nem létező mappa kivétel, ami akkor következik be, amikor az ASP.NET még a mappát sem tudja megtalálni. (Megszüntethetjük a megjegyzéseket az egyik utasításban, hogy lássuk, milyen, amikor minden jól működik.)

Ha a kódunk esetlegesen nem kezelne egy kivételt, akkor olyan hiba oldalt jelenítene meg, amit az előző képen láthattunk. Azonban a `Try/Catch` részek segítenek megakadályozni az ilyen típusú hibaüzeneteket.

```
@Code
    Dim dataFilePath = "/dataFile.txt"
    Dim fileContents = ""
    Dim physicalPath = Server.MapPath(dataFilePath)
    Dim userMessage = "Hello world, the time is " + DateTime.Now
    Dim userErrMsg = ""
```

```
    Dim errMsg = ""

    If IsPost Then
        ' A felhasználó ráklikkel a megnyitásra és elküldi az oldalt
        ' majd megpróbálja megnyitni a fájlt.
        Try
            ' Hibás kód, mivel rossz az elérési út.
            fileContents = File.ReadAllText("c:\batafile.txt")

            ' Ez a működő kód,
            ' Tegyük megjegyzéssé az előző sort, és ezt aktiváljuk:
            ' fileContents = File.ReadAllText(physicalPath)

        Catch ex As FileNotFoundException
            ' A kivétel objektumot használhatjuk debuggolásra,
logolásra,…stb.
            errMsg = ex.Message
            ' Létrehoz egy felhasználóbarát hibaüzenetet.
            userErrMsg = "The file could not be opened, please
contact " _
                & "your system administrator."

        Catch ex As DirectoryNotFoundException
            ' Hasonlít az előző kivételhez.
            errMsg = ex.Message
            userErrMsg = "The file could not be opened, please
contact " _
                & "your system administrator."
        End Try
    Else
        ' Létrehoz egy szöveg fájlt, amikor először lekérik az
oldalt.
        File.WriteAllText(physicalPath, userMessage)
    End If
End Code
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8" />
        <title>Try-Catch Statements</title>
    </head>
    <body>
    <form method="POST" action="" >
      <input type="Submit" name="Submit" value="Open File"/>
    </form>

    <p>@fileContents</p>
    <p>@userErrMsg</p>

    </body>
</html>
```

További források angolul

- ASP.NET
- Visual Basic programozási nyelv

Függelék – ASP.NET weboldalak programozása Visual Studióban

A függelékben megtudhatjuk, hogyan használhatjuk a Visual Studio 2010-et vagy a Visual Web Developer 2010 Expresst az ASP.NET weboldalak programozására a RazorSyntaxszal.

Megtudhatjuk:

- Hogyan telepítsük a Visual Web Developer 2010 Expresst és az ASP.NETRazorToolst (benne az ASP.NET MVC3 ReleaseCandidate-et is)?
- Hogyan használhatjuk a Visual StudioASP.NET-tel kapcsolatos funkcióit: az IntelliSense-t és a debuggert?

Miért válasszuk a Visual Studiót?

Az ASP.NET weboldalaknak a RazorSyntaxszal való programozására használhatjuk a WebMatrixot és sok más szerkesztőt. Használhatjuk a Microsoft Visual Studio 2010-et, mely egy teljes funkcionalitással bíró integrált fejlesztőkörnyezet (integrateddevelopmentenvironment – IDE), mely rengeteg hasznos eszközzel segít minket alkalmazásaink (nem csak weboldalak) fejlesztésében. Ha ASP.NETRazor weboldalakkal szeretnénk dolgozni, megvásárolhatjuk a Visual Studio teljes verzióját, vagy használhatjuk az ingyenes Visual Web Developer Express edition-t.

Két lényeges funkció, amit a Visual Studio nyújthat nekünk ASP.NET oldalak programozásakor:

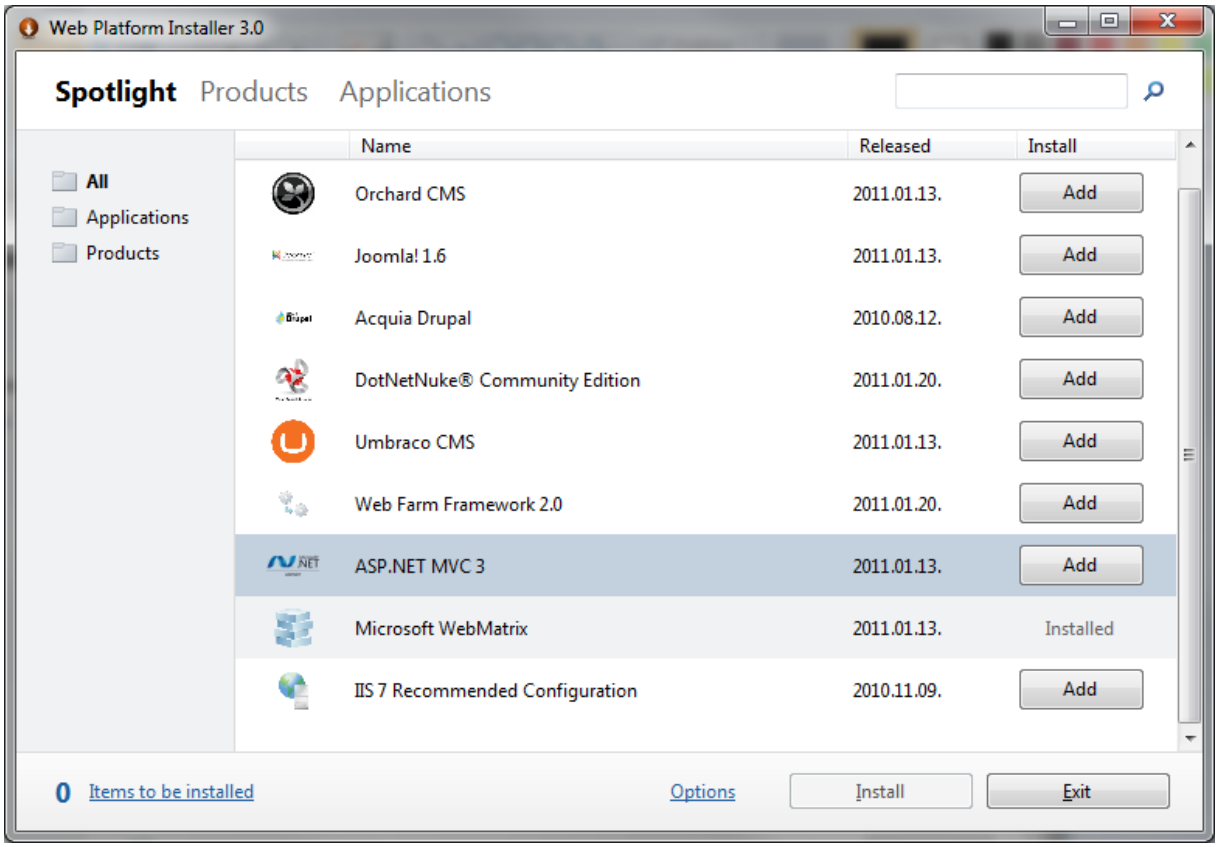
- *IntelliSense*. Gyorsabbá teszi a kódok írását azáltal, hogy egy-egy parancs begépelése közben automatikusan kilistázza azokat a lehetséges osztályokat és metódusokat, amelyeket használhatunk a szerkesztőben, és kiegészíti a parancsokat is.
- *Debugger*. A debugger lehetővé teszi a hibakeresést. A program futtatás közben bárhol megállítható, kilistázható az egyes változók értéke, és futtathatjuk a kódot sorról sorra, így egyszerűen megtaláljuk, hol hibás a kódunk.

Ezek a funkciók csak a Visual Studióban érhetők el.

Az ASP.NETRazorTools telepítése

Ebben a részben megtudhatjuk, hogyan telepítsük az ingyenes Visual Web Developer Express 2010-et és az ASP.NET Web PagesToolsfor Visual Studiót.

1. Ha még nem rendelkezünk a Web Platform Installerrel, töltsük le a következő helyről: http://go.microsoft.com/fwlink/?LinkID=205867
2. Indítsuk el a Web Platform Installert, válasszuk a *Products*ot, majd keressük meg az *ASP.NET MVC3 ReleaseCandidate*-et! Kattintsunk az *Add* gombra! Ez a termék tartalmazza a Visual Studio eszközöket, melyekkel ASP.NET Razor webhelyeket készíthetünk.

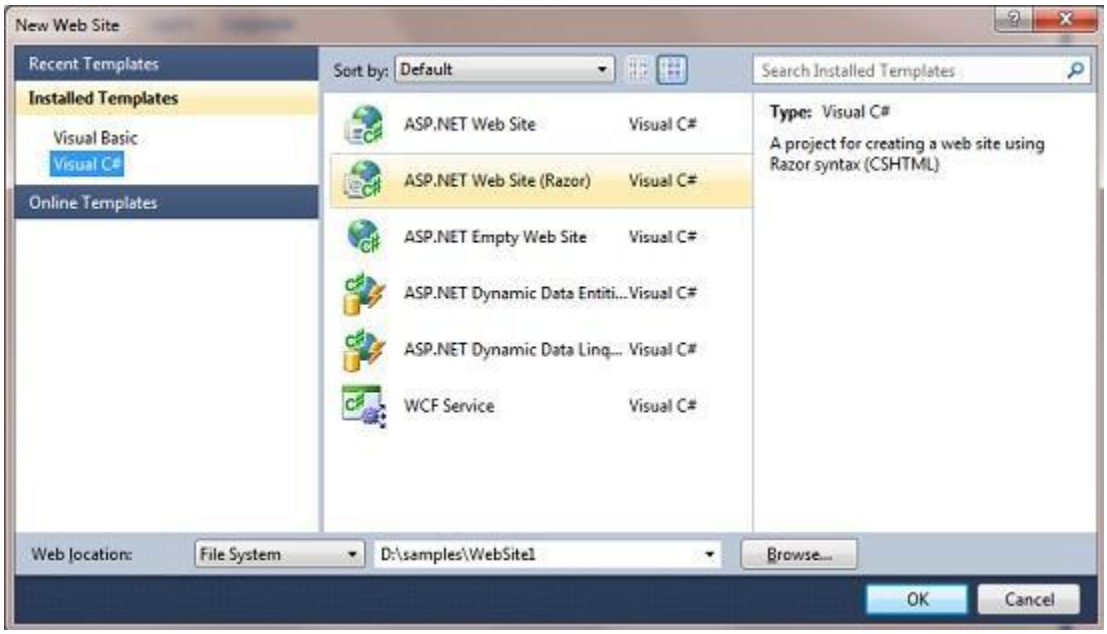


3. Ha még nincs feltelepítve a Visual Studio vagy a Visual Studio Express, keressük meg a *Visual Web Developer Express*t, majd kattintsunk az *Add* gombra!
4. Kattintsunk az *Install* gombra a telepítéshez!

Az ASP.NET Razor Tools használata Visual Studioval

Az IntelliSense és a debugger használatához készítsünk egy ASP.NETRazorwebhelyet a Visual Studióban.

1. Indítsuk el a Visual Studiót vagy a Visual Web Developert!
2. A *File* menüben válasszuk a *New Web Site* lehetőséget!
3. A megnyíló *New Web Site* ablakban válasszuk ki a használni kívánt nyelvet (C# vagy Visual Basic)!
4. Válasszuk az *ASP.NET Web Site (Razor)* sablont!
5. A *Web Locations* felirat melletti legördülő menüben válasszuk a *File System* lehetőséget, az elérési úthoz (path) pedig írjuk be egy mappa elérési útját a számítógépen!

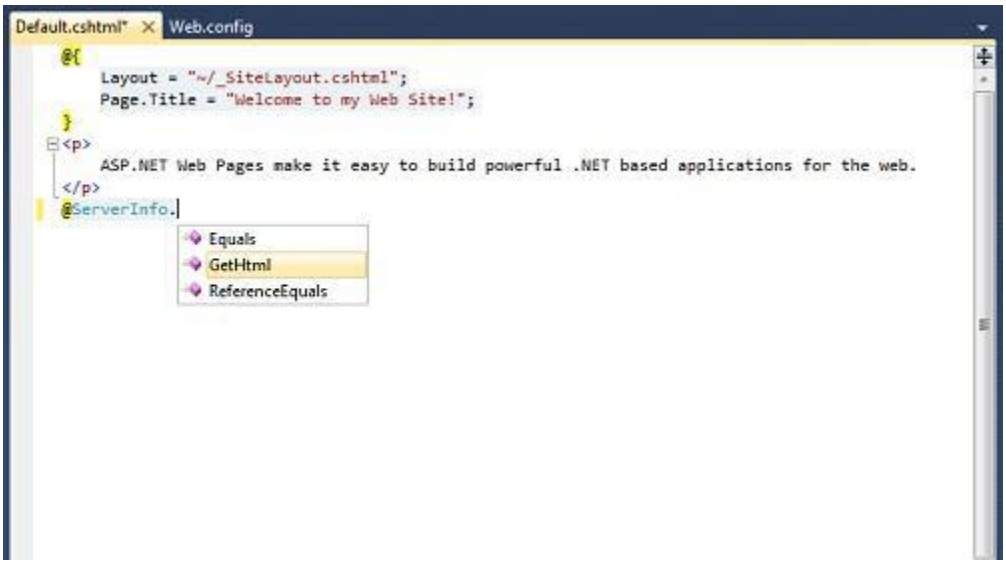


6. Kattintsunk az OK gombra!

Az IntelliSense használata

Miután létrehoztunk egy webhelyet, láthatjuk, hogyan működik az IntelliSense.

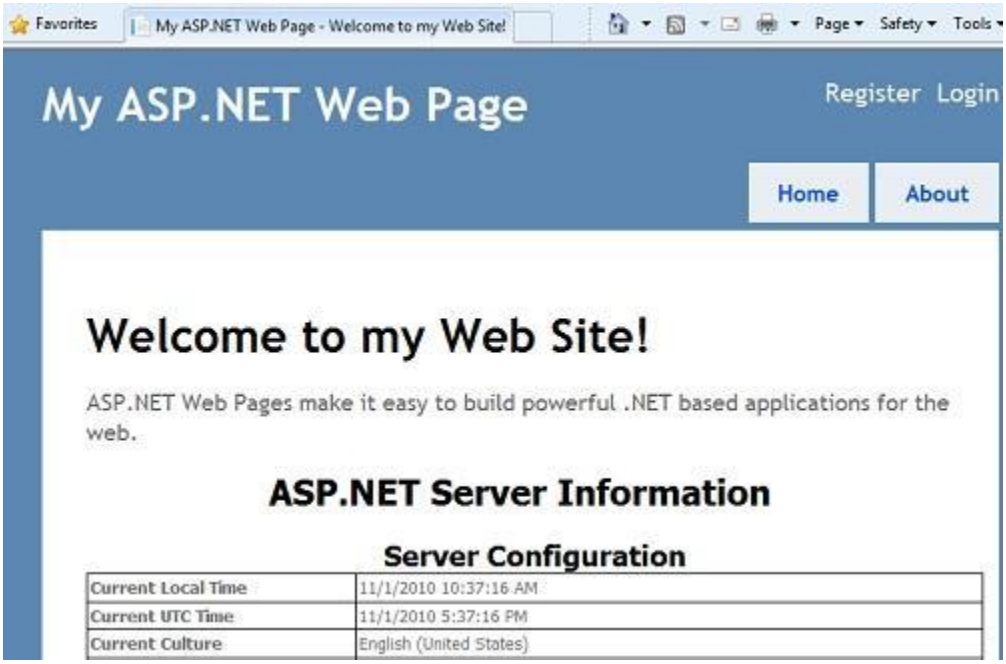
1. A frissen létrehozott webhelyen nyissuk meg a *Default.cshtml* oldalt! Győződjünk meg róla, hogy az ablak alján a *Source* fül van kiválasztva!
2. Amikor lezárjuk a </p>taget, írjuk be: @ServerInfo. (a pontot is a végére)! Az IntelliSense megjeleníti a lehetséges metódusokat, amiket használhatunk a ServerInfohelperhez.



3. Válasszuk a GetHtml metódust a listából, és nyomjuk le az Enter gombot! Az IntelliSense automatikusan kiegészíti a parancsot. (A C#-ban bármilyen metódus esetén oda kell írni a () karaktereket a metódus végére. A teljes kód a GetHtml parancshoz a következőképpen néz ki:

@Server.GetHtml()

4. Nyomjuk le a Ctrl+F5-öt az oldal futtatásához! Az oldal így jelenik meg a böngészőben:



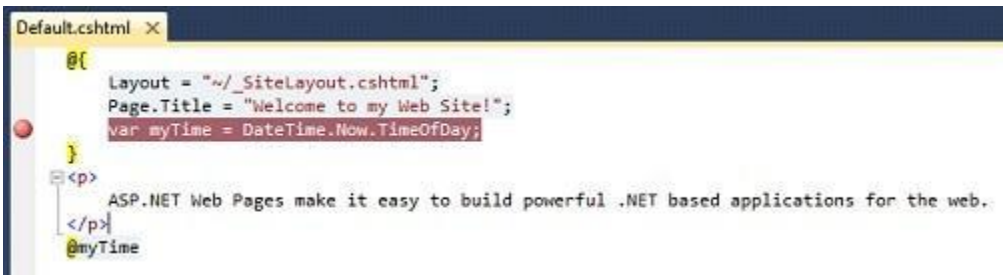
5. Zárjuk be a böngészőt, és mentsük a *Default.cshtml*oldal változásait!

A Debugger használata

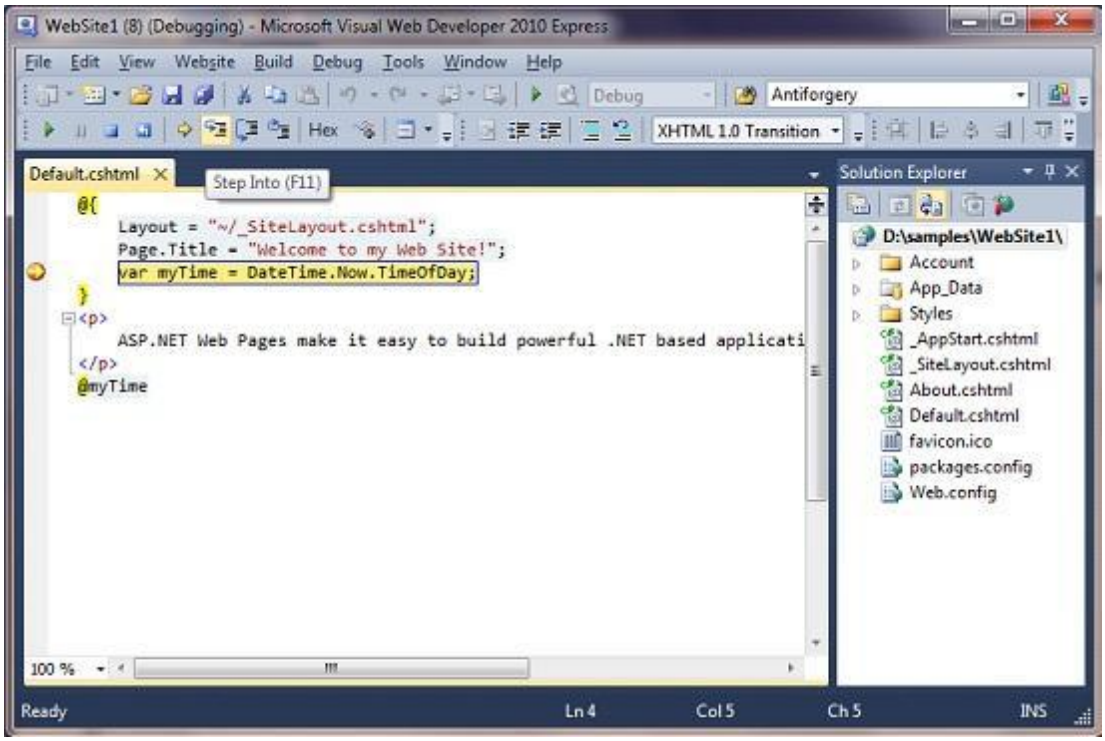
1. A *Default.cshtml* oldal tetején a Page.Title kezdetű sor után írjuk be a következő sort:

Var myTime = DateTime.Now.TimeOfDay;

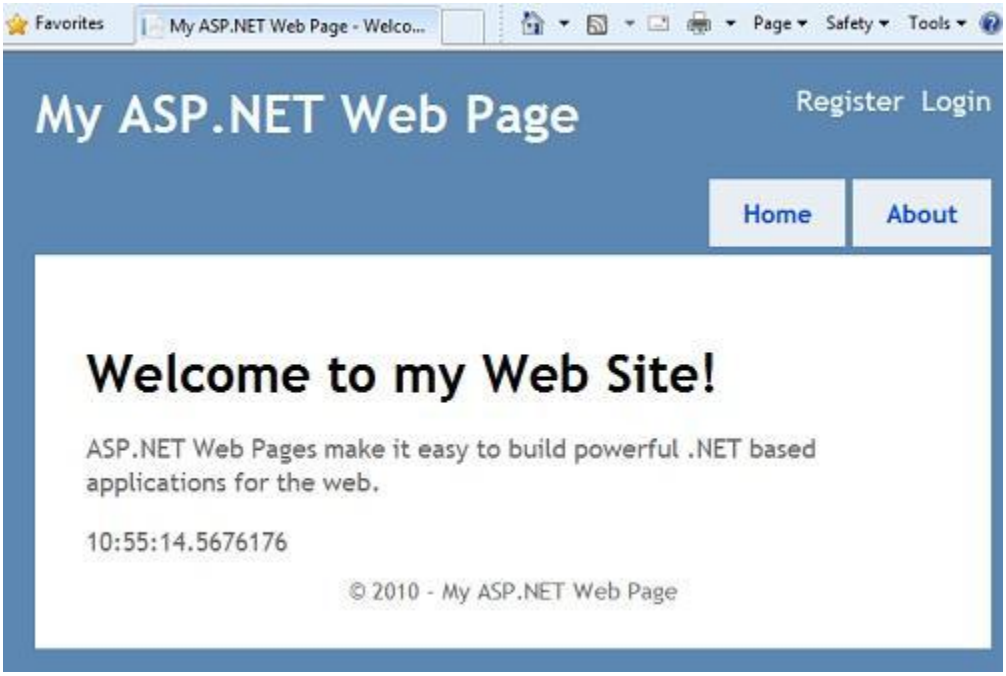
2. A szerkesztő bal oldalán lévő szürke területen kattintsunk a sor melletti részre, így létrehozva egy úgynevezett *breakpoint*ot! Ez egy olyan jelölő, mely jelzi a debuggernek, hogy állítsa meg a programot az adott ponton, így megvizsgálhatjuk a program adott részének működését.
3. Távolítsuk el a ServerInfo.GetHtml parancsot, és írjunk a helyére egy meghívást a @myTime változóra! Ez a meghívás megjeleníti az aktuális időt az új sorban.
   A megváltozott kód a két új sorral és a breakpointtal a következőképpen néz ki:



4. Nyomjuk le az F5-öt az oldal debuggerben való futtatásához! A futtatás megáll az általunk beállított breakpointon. A következő képen láthatjuk, hogyan jelenik meg az oldal a szerkesztőben a breakpointhoz (sárgával jelölt sor) érve. Láthatjuk továbbá a *Debug* eszköztárat, illetve a *Step Into* (sor futtatása) gombot.



5. Kattintsunk a *StepInto* gombra, vagy nyomjuk le az F11 gombot! Ez futtatja a kód következő sorát. Az F11 gomb következő lenyomásával a következő sort futtatjuk, így soronként haladhatunk a kódban.
6. Vizsgáljuk meg a MyTime változó értékét! Tartsuk az egérmutatót a felirat fölött, vagy nézzük meg a *Locals* és a *CallStack* ablakokban!
7. Amikor végeztünk a változók vizsgálatával, és lépésenként végighaladtunk a programon, nyomjuk meg az F5 gombot a kód megszakítás nélküli futtatásához! Az oldal így jelenik meg a böngészőben:



Ha szeretnénk részletesebben megismerkedni a debuggerrel, vagy szeretnénk megtudni, hogyan használjuk a debuggert a Visual Studióban, látogassunk el a következő címre, ahol angol nyelvű leírásokat találhatunk a debuggerről:

http://msdn.microsoft.com/en-us/library/z9e7w6cs.aspx

Nyilatkozat

Ez a dokumentum egy fejlesztés alatt álló termékről szól. Bizonyos információk és funkciók, beleértve a hivatkozásokat és egyéb internetes címeket, előzetes bejelentés nélkül megváltoztathatók. Vegye figyelembe ezt a kockázatot!

A dokumentumban szereplő példák csak illusztrációs példákat szolgálnak, és csak kitalált adatokat tartalmaznak. A valósággal való bármilyen egyezés csupán a véletlen műve, nem szándékos.

Jelen dokumentum nem ruházza fel Önt semmiféle joggal a Microsoft bármely szellemi termékével kapcsolatosan. A dokumentumot lemásolhatja és felhasználhatja belső, illetve referencia célokra.

Ez a dokumentum a Microsoft tulajdonát képezi. Nyilvános dokumentum, mely használható titoktartási szerződés értelmében is.

A Microsoft a Microsoft group védjegye. Minden más védjegyről a saját tulajdonosa rendelkezik.