DEBRECENI EGYETEM
INFORMATIKAI KAR

WEBPORTÁL SZERKESZTŐSÉGI RENDSZER LÉTREHOZÁSA AJAX TÁMOGATÁSSAL, SQL ALAPOKON.

Témavezetők:

Dr. Rutkovszky Edéné
egyetemi tanársegéd

Keczeli Csaba
CTS Informatika Kft.

Készítette:
Buszlai Krisztán
programozó matematikus

DEBRECEN
2008

Tartalom

Bevezető

Az internet és használata mindennapi életünk részévé vált. Már szinte észre sem vesszük, de a világhálón keresztül intézzük életünk ügyes-bajos dolgait. Így banki átutalásaink, vásárlásaink egy részét, munkánkhoz kacsolódó oldalak, árlisták böngészését stb. Az írott levelet teljes mértékben felváltotta annak digitális változata. Tehát egy számítógépet használó átlag ember naponta jelentős időt tölt a világhálón való „szörfözéssel“. Persze az internet felhasználása igen sokrétű, de egy valami közös a felhasználókban. Mindegyikünk napi rendszerességgel látogat olyan portált, ahol a világ híreit, vagy érdeklődésünknek megfelelő cikkeket találunk. Egy ilyen portál felépítése a látogató szemszögéből viszonylag egyszerűnek tűnik. A szakdolgozat egy ilyen hír portál belső felépítésének megvalósítását illetve működését mutatja be. A szakdolgozat célja egy olyan hír portál szerkesztőségi rendszerének megvalósítása, mely ötvözi egy desktop alkalmazás gyorsaságát az internet rugalmasságával. Tehát a létrehozandó portál, egy olyan technikát használ, mely segítségével kerüljük a weboldalak böngészőben történő újratöltéseit, ezáltal egy desktop alkalmazáshoz hasonló kezelőfelületet kaphat a felhasználó. Ez az eredmény az AJAX támogatással érhető el.

AZ ASP.NET bemutatása

ASP.NET

ASP.NET a Microsoft által megalkotott szerver oldali szkript technológia, mely lehetővé teszi a HTML kódhoz fűzött szkriptek futtatását. A szkriptek futtathatók mind lokálisan a felhasználó gépén, mind pedig távolról a webalkalmazást tároló szerveren. Az ASP.Net egy olyan program, mely az IIS –ben (Internet Information Service) fut. Az IIS egy kiszolgáló-szoftver, mely webszervert illetve levelező-kiszolgálót foglal magában. A Microsoft által megalkotott operációs rendszerek, például Windows 2000, Windows XP, Windows Vista magában foglalja az IIS -t.

Az ASP -t az Active Server Pages szavak rövidítéséből kapjuk. Egy ASP.Net oldal megalkotásakor egy ASP.NET fájlt hozunk létre.

Egy ASP.NET fájl éppen olyan, mint egy HTML fájl, mely az általános HTML kódon kívül tartalmazhat HTML, XML, és szkript elemeket. Az ASP.NET fájlba ágyazott szkriptek a szerveren futtathatók. Kiterjesztése „.aspx“.

ASP.NET működése

Mikor a böngészőben egy HTML fájlt hívunk meg, akkor egyszerűen visszakapjuk a kívánt HTML fájl kódját, melyet a böngésző értelmezni tud és betölti a weboldalt. Ezzel ellentétben egy ASP.NET fájlt meghívásakor, az IIS átirányítja a kérést a szerveren található ASP.NET motornak, mely a hivatkozott fájt sorról sorra beolvassa, a benne kódolt szkripteket futtatva egy a böngésző számára értelmezhető HTML kódot küld a böngészőnek. Az ASP.NET lapokban 3 féle módon lehet elhelyezni vezérlőkódot:

- A HTML kódot tartalmazó fileban elhelyezhetők szerver oldali script kódok, a .NET keretrendszer által támogatott nyelveken. Az ilyen kódokat a file beolvasása során interpreteres módon futtatja a keretrendszer.
- A HTML kódot tartalmazó fileban elhelyezhetők kliens oldali script kódok. Az ilyen kódok a HTML tartalommal együtt letöltődnek a kliens gépére, és végrehajtásukért a böngésző felel. Az ilyen scripteket Javaszkript nyelven lehet megírni.

- A szerver oldali kód a HTML kódot tartalmazó file mellett is elhelyezhető. Ennek előnye, hogy ilyenkor a komplett kódfilet a keretrendszer a lap elérésekor lefordítja köztes kódra, így nem interpreteresen futtat. A módszer további előnye, hogy a szerver a lefordított filet eltárolja, így az oldal későbbi elérésekor csak akkor fordítja újra, ha a forrás változik.

A Microsoft .NET Framework

A .NET Framework egy fejlesztői környezet Microsoft .NET platform -ra. Segítségével desktop, konzol, web alkalmazásokat és web szolgáltatásokat tudunk létrehozni, fejleszteni és futtatni.

.NET Framework tervezésnél kitűzött célok:

- Könnyebb gyorsabb programozás
- A kódok méretének jelentős csökkentése
- Deklaratív programozási modell felépítése
- Gazdagabb szerver kontrol hierarchia a hozzájuk kapcsolható eseményekkel
- Nagyobb class library
- A fejlesztői eszközök jobb támogatottsága
- OO programozás lehetősége
- Támogassa az ismert nyelvek többségét

Az ASP.NET a szkriptek kódolásához a következő programozási nyelveket támogatja:

- ASP.NET támogatja a teljes Visual Basic nyelvet, ide nem értve a VBScript -et.
- ASP.NET támogatja a C# (C sharp) és C++ nyelveket.
- ASP.NET támogatja JScript –et mint elődjét.

- Az ASP.NET támogatja a teljes Javaszkript nyelvet, a kliens oldali programozási feladatok megoldására.

ASP.NET kontrollok

Az ASP.NET nagyon sok HTML vezérlőt tartalmaz. Majdnem minden HTML elem definiálható, mint egy ASP.NET vezérlő objektum, melyek szkriptek segítségével könnyedén vezérelhetőek, manipulálhatóak.

Mindezen felül az ASP.NET számtalan objektum orientált input kontrolt is tartalmaz, mint például a programozható LISTBOX illetve validátorok.

Az új dataGrid kontrol minden olyan tulajdonsággal fel lett ruházva, melyekre egy dataSet kontrol használata során a programozónak szüksége lehet.

Felhasználó authentikáció

ASP.NET támogatja felhasználó autentikációt, ide értve a sütik kezelését és a jogosulatlan hozzáférés esetén bekövetkező automatikus átirányítást a bejelentkező oldalra. Ha egy ASP.NET oldal használata során felhasználókat szeretnénk autentikálni, úgy az oldal szerkesztésekor nincs más dolgunk, mint egy login kontrolt beszúrni a kódba. A login kontrol innentől kezdve létrehozza és kezeli a felhasználókat tároló adatbázist. A programozónak nincs más dolga, mint a login kontrol –on menedzselni a felhasználókat. A menedzselés során felhasználókat adhatunk hozzá illetve törölhetünk web alkalmazáshoz. A felhasználókhoz jogosultságokat rendelhetünk könyvtárszintekhez. A felhasználókat csoportokba rendezhetjük, és a csoportokhoz úgynevezett Role –okat rendelhetünk.

A role jelentése szerepkör. Adatbázisokhoz szerepköröket hozhatunk létre, mely szerepkörök esetében egyenként más-más jogosultságot állíthatunk be. A szerepkörökhöz felhasználókat rendelhetünk. Így egyszerűsíthető a felhasználónkénti jogosultság beállítás. Például, ha az egy szerepkörhöz tartozó felhasználók jogosultságát módosítani szeretnénk, úgy nem kell egyenként minden felhasználónál az adott módosítást elvégezni, elég a role tulajdonságát módosítani és a role- ban szereplő összes felhasználó jogosultsága a módosításnak

megfelelően változik. Az authentikációs rendszer nagyon rugalmas, így ha esetlegesen nem felel meg számunkra a beépített két azonosító vezérlő és adatmodell, akkor írhatunk saját vezérlőket, amelyet tetszőleges adatstruktúra fölött képesek működni. Ezek a vezérlők – köszönhetően az OO modellnek – teljesen csereszabatosak a gyári vezérlőkkel, így ezek előnyeiről nem kell lemondanunk.

SZERVER KONTROLLOK

A szerver kontrollok olyan vezérlők, melyeknek a szerver tulajdonít jelentőséget, a szerver értelmezi azt.
A szerver kontrolok 3 kategóriába sorolhatók:

- **HTML Szerver Kontrolok** – általános értelemben vett HTML vezérlők
- **Web szerver kontrolok** – új ASP.NET vezérlők
- **Érvényesítő szerver kontrolok** – Az input elemek érvényesítésére szolgálnak, a beviteli elemekhez kapcsolhatók

A kontrollok a szerver oldali futást követően a kliens oldalon HTML elemekké fordulnak. A programozott logika terén a rendszer rugalmas: a szerver oldal érzékeli, hogy a kliens oldali böngésző képes-e futtatni Javaszkript kódot, és ha igen, és nem tiltjuk le, akkor lehetőség szerint kliens oldali kódot generál. Ha a kliens oldali böngésző nem képes Javaszkriptet futtatni, akkor a mögöttes kód a szerveren fog futni. Jó példa erre az összes beépített érvényesítő vezérlő: a beviteli elemeket ha lehet, kliens oldalon ellenőrzi, hiszen ilyenkor nem kell az oldalt visszaküldeni, így gyorsabb az ellenőrzés. Ellenben, ha a kliens oldalon a script futtatása nem lehetséges, akkor automatikusan olyan HTML kódot ad vissza, ami a beviteli elemek ellenőrzését szerver oldalra küldi.

ASP.NET - HTML szerver kontrollok

HTML szerver kontrollok olyan HTML tageknek megfelelő vezérlők, melyeket a böngésző értelmezni tud, így egy működő weboldal ezekből építhető fel.

HTML elemek az ASP.NET fájlokban alapértelmezett módon szövegként kezeltek. A HTML elemek alapértelmezetten a kliens böngészőjében kerülnek értelmezésre, a webszerveren pedig nem. Ezek az elemek, úgy tehetők programozhatóvá, vagyis a szerver oldali kódból a lap megjelenítése előtt elérhetővé, hogy " runat="server" " tulajdonságot adunk nekik.

Ezt a tulajdonságot az elem címkéjében adhatjuk meg. Ezen a tulajdonság megadásával tudatjuk a szerverrel, hogy a megadott HTML elem szerver kontrolként használandó. A HTML elemnek ID tulajdonságot adva egyedi néven nevezhetjük el az adott szerver kontrolt, így hivatkozhatóvá tesszük és futási időben a ID tulajdonságra hivatkozva manipulálhatjuk azt a különböző szkriptekből. Az ID tulajdonság használata nem csak szerver oldali vezérlőknél hasznos, hanem kliens oldaliaknál is. Ezt a tulajdonságot is a vezérlő címkéjében adhatjuk meg.

Minden szerverkontrolnak a <form> </form> címkék között kell elhelyezkednie a runat="server" tulajdonság megadásával.

A következő szemléltető példában egy <a> anchor HTML szerver kontrollt helyezünk el egy .aspx fájlban. Attól lesz szerver kontroll, hogy felruházzuk a runat=”server” tulajdonsággal. A szintén a .aspx fájlban elhelyezett szkript segítségével href tulajdonságot módosítjuk egy esemény kezelőben (az eseménykezelő egy olyan alprogram, mely az adott esemény bekövetkeztekor hajtódik végre). Jelen példában a Page_Load eseményben manipuláljuk a tulajdonságot. A Page Load egy olyan ASP.NET esemény, melyet az oldal kódját futtató .NET motor az oldal betöltődésének pillanatában hajt végre:

```
<script runat="server">
      protecte void Page_Load()
      {
            link1.HRef="http://asp.net";
      }
</script>
<html>
  <body>
      <form runat="server">
            <a id="link1" runat="server">Kattints ide</a>
      </form>
  </body>
</html>
```

ASP.NET - Web szerver kontrolok

A web szerver kontrollok speciális ASP.NET vezérlők. Ezeket a kontrollokat a szerver használja fel.
Úgy, mint a HTML szerver kontrolok, a web szerver kontrolok is a szerveren kelnek életre és nekik is szükségük van a runat="server" tulajdonság beállítására.
Egy web szerver kontrol megadásának szintakszisa:

```
<asp:kontrol_nev id="azonosito" runat="server" />
```

ASP.NET –Validátor szerver kontrolok

A validátor szerver kontrolok a felhasználói beviteli mezők érvényesítésére használandók. Amennyiben a felhasználó egy beviteli mezőbe értéket visz be és arra a beviteli mezőre Validátor szerver kontrol van kapcsolva, abban az esetben a Validátor felülvizsgálja a bevitt értéket. Amennyiben a bevitt érték eltér a programozó által szabályosnak vélt, megengedett értékétől, úgy hibaüzenettel tér vissza. Egy beviteli mezőre úgy köthető Validátor kontrol, hogy a validátor kontrol *ControlToValidate* tulajdonságában a beviteli mező *ID* –ben megadott egyedi azonosítóját adjuk meg. Egy validátorhoz kizárólag egy beviteli mezőt adhatunk meg. A szakdolgozatban létrehozott szerkesztőségi rendszerben a következő Validátorok kerültek felhasználásra:

RequiedFieldValidator: Használatakor a hozzá csatolt beviteli mezőt vizsgálja, hogy tartalmaz e bevitt értéket, vagy nem. Amennyiben nem tartalmaz, úgy a programozó által módosítható hibaüzenettel tér vissza.

RangeFieldValidator: Használatakor a hozzá csatolt beviteli mezőt vizsgálja, hogy a beviteli mezőben megadott érték a programozó által megadott határok között van e. (pl: számok 1 –től 10 -ig)

RegularExpressionValidator: Használatakor a hozzá csatolt beviteli mezőről eldönti, hogy a beviteli mezőben megadott érték megfelel e a programozó által meghatározott reguláris kifejezésnek. (Például a létrehozott szerkesztőségi rendszer a következő reguláris kifejezést

használja a dátum bevitelére fenntartott beviteli mezőhöz kapcsolt RegularExpressionValidator –ban, mely a dátum YYYY/HH/NN formátumára tesz kikötést:

```
"^(((19)([29])(\d{1})|(20)([01])(\d{1})|([8901])(\d{1}))(-
|\/)(0?[13578]|10|12)(-|\/)(([1-9])|(0[1-9])|([12])([0-
9]?)|(3[01]?))|((19)([29])(\d{1})|(20)([01])(\d{1})|([8901])(\
d{1}))(-|\/)(0?[2469]|11)
(|\/)(([19])|(0[19])|([12])([0-9]?)|(3[0]?)))$")
```

Alapértelmezett esetben a validátor a vizsgálat megkezdését akkor kezdi meg, mikor az weboldalra helyezett Button, ImageButton vagy LinkButton kontrolok valamelyikére klikkel a felhasználó. PostBack esemény kizárólag akkor hajtódik végre, mikor a validátorok egyike sem tér vissza hibával.

A validátorok szerver kontrol létrehozásának szintakszisa:

```
<asp:kontrol_nev id="azonosito" runat="server"
controlToValidate=“beviteli_mezo_azonositoja“ />
```

ASP.NET Web Form

Mint az már említésre került, minden szerver kontrolnak a <form></form> címkéken belül kell elhelyezkednie, illetve a <form> címkének tartalmaznia kell a runat="server" tulajdonságot. A runat="server" tulajdonság jelzi, hogy a formot a szervernek kell feldolgoznia, illetve a benne megadott szerver kontrollok, a létrehozott kód szkriptjeiből hivatkozhatók, manipulálhatók:

```
<form runat="server">
...HTML + szerver kontrolok
</form>
```

Egy form postback esemény meghívásakor mindig önmagát hivatkozza. Ez az action tulajdonság megadásával átállítható. Amennyiben a method tulajdonság nem kerül megadásra, úgy alapértelmezettként a method tulajdonsághoz "post" érték rendelődik. Ha a

name és id tulajdonságok sem kerülnek megadásra, úgy hozzájuk automatikusan az ASP.NET érték rendelődik. A method tulajdonságokkal a visszaküldött adatoknál a visszaküldés helye állítható be. Get esetében a visszaküldött adatok az URL végéhez fűzve, a felhasználó által látható módon lesznek visszaküldve. Post esetében a visszaküldött értékeket a böngésző kérései között küldjük el.
Egy .aspx oldal kizárólag egy <form runat="server"> címkét tartalmazhat!

Egy Form elküldése

Egy form leggyakrabban egy gombra történő klikkeléssel küldhető el a szerver felé. A Button szerver kontrol ASP.NET környezetben a következő kóddal rendelkezik:

```
<asp:Button id="azonosito" text="szoveg" OnClick="sub" runat="server"/>
```

Az ID tulajdonság a Button szerver kontrol egyedi azonosítóját definiálja, a TEXT tulajdonság, pedig a gomb feliratát specifikálja. Az onClick eseménykezelő azt a szubrutint definiálja, melyet az esemény meghívásakor futtatni kell.
A következő példában egy Button szerver kontrolt demonstrál, melyre kattintással a submit szubrutin kerül végrehajtásra, mely megváltoztatja a kontrol feliratát:

```
<script  runat="server"> protected void submit(Source As Object, e As EventArgs)
   button1.Text="Megváltozott szöveg!"
</script>
<html>
<body>
      <form runat="server">
      <asp:Button id="button1" Text="Eredeti szöveg!" runat="server"
      OnClick="submit" />
      </form>
</body>
</html>
```

AJAX

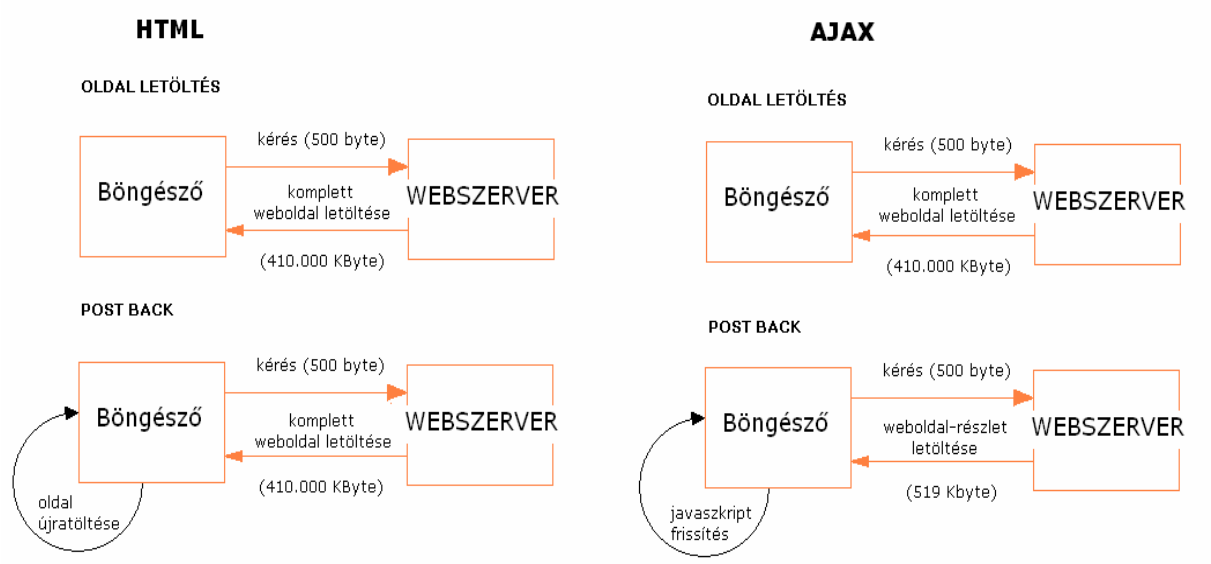
Az AJAX bemutatása

Az AJAX (asynchronous Javaszkript and XML) interaktív webalkalmazások létrehozására szolgáló webfejlesztési technika. A normál weboldalak olyan formában működnek, hogy ha a kliens oldalon megjelenített lapon olyan esemény történik, amelynek hatására a szerver oldallal kell kommunikálni (pl egy egyszerű oldalváltás, vagy akár egy lapozás), akkor a komplett adattartalom visszaküldésre kerül a szerverhez. Erre válaszul a szerver ismét a teljes weboldalt visszaküldi, amit a kliens oldali böngésző megjelenít. AJAX használata esetén lehetőség van az oldal teljes újratöltése nélkül kommunikálni a szerverrel. A weblap kis mennyiségű adatot cserél a szerverrel a háttérben, így a lapot a böngészőnek nem kell újra letöltenie minden egyes alkalommal, amikor a felhasználó módosít valamit. Ez növeli a weblap interaktivitását, sebességét és használhatóságát.
Az AJAX a következő technikák kombinációja:

- XHTML (vagy HTML) és CSS a tartalom leírására és formázására.
- DOM kliens oldali script nyelvekkel kezelve a dinamikus megjelenítés és a már megjelenített információ együttműködésének kialakítására.
- XMLHttpRequest objektum az adatok aszinkron kezelésére a kliens és webszerver között. Néhány AJAX keretrendszer esetén és bizonyos helyzetekben IFRAME –et használnak XMLHttpRequest objektum helyett.
- XML formátumot használnak legtöbbször az adatok továbbítására a kliens és a szerver között, bár más formátumok is megfelelnek erre a célra, mint a formázott HTML vagy sima szöveg.

Az AJAX használatának hátterében a felhasználói élmény fokozása áll. Az AJAX -ot használó webalkalmazások viselkedése sokkal inkább emlékeztet a desktop alkalmazásoknál megszokotthoz, mint a sima AJAX támogatás nélküli weblapokéhoz. Mikor egy postback esemény bekövetkeztekor a weboldal újra betöltődik a böngészőbe, az sokszor igen időigényes lehet, ráadásul még ha gyors is, az oldal villog. AZ AJAX használatával ez a kellemetlenség kiküszöbölhető. Segítségével, a teljes oldal újratöltése helyett kizárólag az oldal egy részének frissítésére kerül sor.

Mivel az AJAX -ot használó weboldalak a szervertől az adatokat HTML formázás nélkül kapják, ezért ez által a szerver terhelése csökken. Tehát az AJAX –szal nagymértékben csökkenthető a szerver terhelése, és nagy mértékben növelhető az oldalak letöltési sebessége. A szervertől kapott adatokból a HTML kód a böngészőben jöhet létre Javaszkript segítségével, ami jól optimalizált programkód esetén legtöbbször gyorsabb mintha az erősen leterhelt szerver hozná létre azt. Ennek oka, hogy ma már a kliens oldalon a felhasználók viszonylag gyors személyi számítógépekkel rendelkeznek, amelyek terhelése lényegesen alacsonyabb, mint a szerveré. Ráadásul, ha valamelyik kliens gép a lassúsága vagy leterheltsége miatt mégis lassabban hozza létre a HTML kódot, az nem érinti a párhuzamosan jelen lévő többi klienst, ami annál nagyobb előny, minél nagyobb a párhuzamosan jelen lévő kliensek száma. Tehát minél nagyobb egy oldal látogatottsága, annál nagyobb előnyt jelent a kliens oldali HTML generálás. Mindezeket tovább erősíti, hogy az AJAX segítségével sokszor jól megvalósítható, hogy mindig csak az éppen szükséges minimális mennyiségű adat töltődjön le a szerverről.



1. ábra HTML és AJAX támogatott oldalak letöltésének és postback -jének összehasonlítása

AJAX Control Toolkit

Az Ajax Control Toolkit egy nyílt forráskódú **community project**, vagyis Microsofton kívül fejlesztők együttműködésének eredménye, Így olyan eszközöket sikerült kifejleszteniük, melyek segítik a fejlesztőket mindennapi munkájukban, hogy AJAX - enabled weboldalakat

készíthessen egyszerűen. Tehát a Control Toolkit komponensek-gyűjteménye, melyek azt teszik lehetővé, hogy RIA –t (Rich Internet Application) fejleszthessünk, gazdag kliens oldali felülettel. A Control Toolkit egyik legnagyobb előnye, hogy esetében nem csak IE6 –ra optimalizált kódról van szó, hanem IE7, FireFox és Safari alatt is garantált működésről és helyes megjelenítésről gondoskodik.

Mint ahogy a nevéből is látszik, főleg AJAX Extension –re illetve AJAX Library Framework –re épül. Nem okoz nagy meglepetést számunkra a tény, hogy Control Toolkit szintén egy keretrendszer, ha tudjuk, hogy az AJAX Extension illetve az AJAX Library is tartalmaz vagy maga is egy framework. Ebből az az előnyünk származik, hogy akár saját komponenseket is fejleszthetünk gyorsan, hatékonyan, melyhez egy eléggé nagy SDK nyújt segítséget.

A teljesség kedvéért még annyit érdemes megemlíteni az AJAX Control Toolkit –ről, hogy régebben az „AJAX projekt“ része volt, amely ATLAS fedőnév alatt futott, viszont a megjelenés előtt átnevezték, szétszedték, így lett az ATLAS –ból:

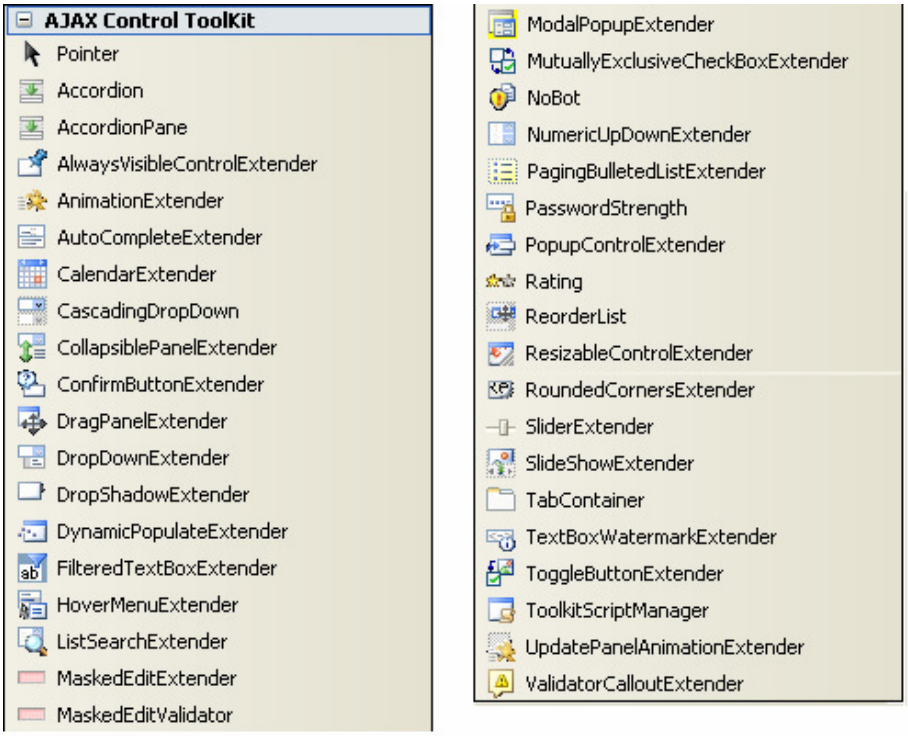
- AJAX Control Toolkit
- AJAX Extension
- AJAX Library

Az AJAX Control Toolkit három fő részből áll:

- Control
- Behavior
- Extender

A control –ok olyan vezérlők, melyek AJAX –os funkcionalitással rendelkeznek.

A behavior, mint ahogy nevéből következik, egy már meglévő ASP.NET 2.0 –ás vezérlő viselkedését, funkcionalitását változtatja meg, (általában kibővíti). Ez felelős a kliens oldali részért, így egyértelmű, hogy Javaszkriptben lett megírva. A behavior –öket általában együtt használjuk az Extender –ekkel. Ennek oka az, hogy az Extender a szerver oldali párja a dolognak. Ez a sok dolog egyetlen egy dll –ben kapott helyet, melyet nem kell a GAC –ban (global assembly cache) tárolnunk, hogy használhassuk.



2. ábra Ajax Control Toolkit elemei

FCK Editor

ASP.Net alapú oldalainkhoz is könnyedén integrálhatjuk a népszerű, formázott szövegek bevitelére alkalmas FCKEditor-t, mely ingyenesen letölthető a http://fckeditor.net oldalról. Az FCKEditor beillesztésével egy nagyon egyszerűen kezelhető szövegszerkesztő vezérlőt kapunk, mely segítségével a Microsoft Word szövegszerkesztőnél már megszokott eszköztár segítségével könnyedén formázhatjuk a bevitt szöveget az igényeknek megfelelően.

Az FCKEditor vezérlőt az alábbi minta szerint hozhatjuk létre:

Meg kell adnunk, hogy hova másoltuk be az fckeditor könyvtárát; ezt a BasePath tulajdonsággal is megadhatjuk minden egyes FCK vezérlőnek, de érdemesebb a Web.Config fájlban meghatározni az appSettings tag-en belül.
Így mindegyik FCK vezérlő BasePath tulajdonsága a fent beállított érték lesz.
Az elküldött formázott szöveget a Value tulajdonság fogja tartalmazni. Egyes formázások következtében előfordulhat, hogy "A potentially dangerous Request.Form value was detected from the client..." hibaüzenetet kapunk. A leggyorsabb javítási mód az, ha az oldal

ValidateRequest tulajdonságát hamisra állítjuk, azonban ilyenkor számolnunk kell a fellépő biztonsági kockázatokkal!

FCK Editor Konfiguráció - eszköztárak

Az FCKEditor gyökérkönyvtárában lévő fckconfig.js segítségével határozhatjuk meg a szerkesztőfelület működését. Néhány fontosabb beállítási lehetőség:

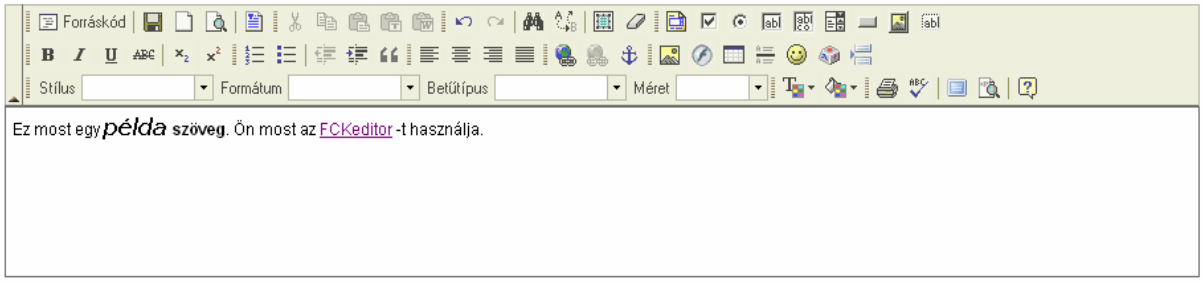
- FCKConfig.ToolbarSets - az eszköztár elemeinek megadásával itt határozhatjuk meg, hogy a felhasználók milyen formázási lehetőségeket alkalmazhatnak a szöveg szerkesztése folyamán. Több lehetőséget is meg lehet adni, és később a kódból ezeket könnyedén ki lehet választani. Jópár előre megadott eszköztár készlet is rendelkezésünkre áll, tehát nem is biztos, hogy hozzá kell nyúlnunk ehhez a részhez. Az egyes FCK vezérlőknek a ToolbarSet tulajdonságuk megváltoztatásával állíthatjuk be a megfelelő eszköztárat.
- FCKConfig.EnterMode és FCKConfig.ShiftEnterMode - azt adja meg, hogy az Enter vagy Shift+Enter gombok lenyomása milyen html elemet eredményezzen
- FCKConfig.FontColors - ezzel azt határozhatjuk meg, hogy milyen színeket választhat a felhasználó. Természetesen ennek csak akkor van értelme, ha az eszköztáron megjelenik a szövegszín választó gomb.
- FCKConfig.FontNames - a kiválasztható karakterkészleteket adhatjuk meg (csak akkor tudnak választani, ha van erre szolgáló gomb az eszköztáron)
- FCKConfig.FontFormats - azokat a szövegstílusokat adhatjuk meg, amelyeket a felhasználók kiválaszthatnak (csak akkor tudnak választani, ha van erre szolgáló gomb az eszköztáron)

Fájlfeltöltések beállítása

A felhasználók képekre is tudnak hivatkozni, illetve ha ezt beállítjuk, fel is tudnak tölteni képeket (vagy akár flash animációkat és egyéb fájlokat is) a szerverre.
Ha ezt lehetővé szeretnénk tenni, akkor meg kell adnunk, hogy melyik könyvtárba kerüljenek a fájlok. Ezt a Web.Config-ban kell beállítani, a BasePath-hoz hasonlóan:

Második lépésként az fckconfig.js fájlban keressük meg a _FileBrowserLanguage és a _QuickUploadLanguage változókat, és azok értéket állítsuk aspx-re.
Az fckeditor/editor/filemanager/connectors/aspx könyvtárban található config.ascx-ben még engedélyeznünk kell a fájlfeltöltését: a CheckAuthentication függvényt kell úgy átírni, hogy az igazzal térjen vissza, amennyiben engedélyezett a feltöltés. Ilyen eset az, ha pl. az oldal adminisztrátora bejelentkezett.

Az itt leírtaknál természetesen jóval több beállítási lehetőségünk van; az fckconfig.js szerencsére alaposan dokumentált, így nem lehet gondunk a lehetőségek megismerésével.



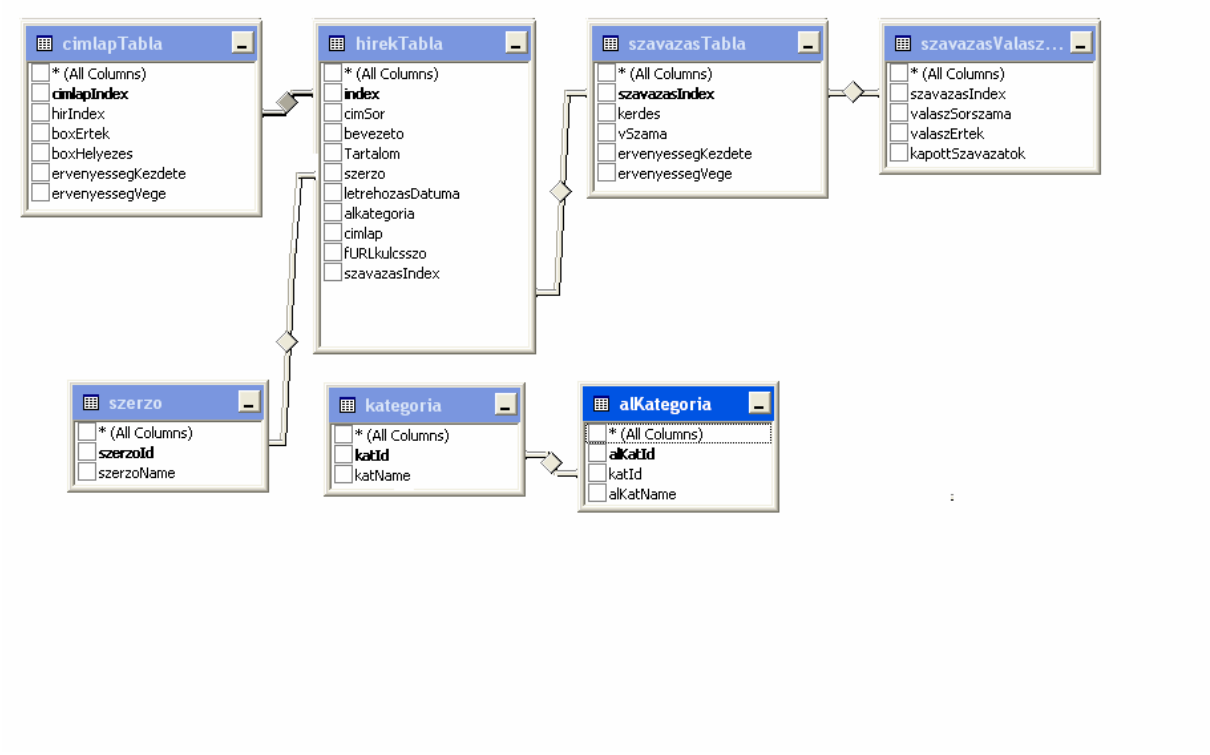
3. ábra FCKEditor

Webszerkesztőségi rendszer felépítése és működésének bemutatása

A szakdolgozat következő fejezetében az előzőekben ismertetett ASP.NET környezetben C SHARP nyelven kódolt webszerkesztőségi rendszer bemutatását, létrehozásának folyamatát ismerhetjük meg.

Az Alkalmazott adatbázis séma.

A portál rendszert felépítő adatok tárolása SQL adatbázisban történik. Az adatbázis 7 táblából épül fel. A következő ábra ezen táblák felépítését és a közöttük fennálló kapcsolatot mutatja be:



4. ábra adatbázis séma

A portál cikkek összességéből épül fel. A cikkek tárolására a hirekTabla-t használjuk. A cikkek az adatbázisban rendelkeznek egy index azonosítóval, mely a hirekTabla elsődleges kulcsa. Minden cikknek rendelkeznie kell egy cím sorral, mely a főcímet adja meg, egy bevezető szöveggel, mely a cikk rövid bevezető szövegét takarja, illetve egy tartalom

szöveggel, mely a cikk tartalmi részét jelenti. Mindhárom adatbázis elem típusa nvarChar(max), azaz ékezetes betűket is támogató UNICODE karakterlánc. Minimum ezen három összetevő alkot egy cikket.
A cikkeknél tároljuk még a szerző nevét, létrehozás dátumát, azt hogy szerepel e a címlapon (nyitóoldal), milyen Frendly URL azonosítóval (lásd: 50. oldal) rendelkezik, illetve melyik szavazás tartozik hozzá, milyen kategóriába és alkategóriába sorolta be a szerző.
Amennyiben a cikk megjelenik a címlapon, úgy a hirekTabla –ban a cikkhez tartozó címlap mező értéke 1 egyébként 0. Amennyiben 1, úgy a hir azonosítója, azaz a hirekTabla –ban a cikkhez tartozó index értéke felvitelre kerül a cimlapTabla soron következő sorába és a box illetve boxHelyezes értéke 0 –ra állítódik. Ezen kívül az érvényességre vonatkozó dátumok is a megfelelő input mezőbe kerülnek beírásra (ervenyessegKezdete, ervenyessegVege). A cimlapTabla box mezőjének egyes értékei az adott cikk címlapi megjelenésének pontos helyét határozza meg (címlap lásd később). A boxHelyezes mező értéke az adott cikk adott „box“ –on belüli sorszámát adja vissza, azaz a cikk boxon belül helyzetét határozza meg.
A webszerkesztőségi rendszer alkalmas szavazások létrehozására. Amennyiben egy felhasználó szavazást hoz létre, úgy azt a szavazasTabla SQL táblába mentheti el, ahol a szavazás kap egy szavazasIndex egyedi azonosítót. Ez lesz a szavazasTabla tábla elsődleges kulcsa, egyben a hirekTabla külső kulcsa.
Amennyiben egy cikkhez tartozik szavazás is, úgy a cikk szavazasIndex mezőjébe a létrehozott szavazások közül a hozzá csatolt szavazás azonosítóját tároljuk el, azaz a szavazasTabla táblában tárolt szavazáshoz tartozó szavazasIndex értéket.
Az elmentett szavazáshoz a szavazasTabla sql táblában tároljuk még a szavazáshoz tartozó kérdést, az arra adható válaszlehetőségek számát, az érvényességének kezdeti illetve lejárati dátumát. Külön táblában tárolódnak az egyes válaszlehetőségek adatai. Ez a tábla a szavazasValaszokTabla.

Külön táblában tárolódnak a kategóriára, alkategóriára illetve szerzőre vonatkozó információk. A közöttük lévő kapcsolatot és az egyes táblák oszlopainak nevét a 3. ábra szemlélteti.
A hírportál felépítése két fő szemszögből közelíthető meg:

- Látogatói nézet
- Hírszerkesztői nézet

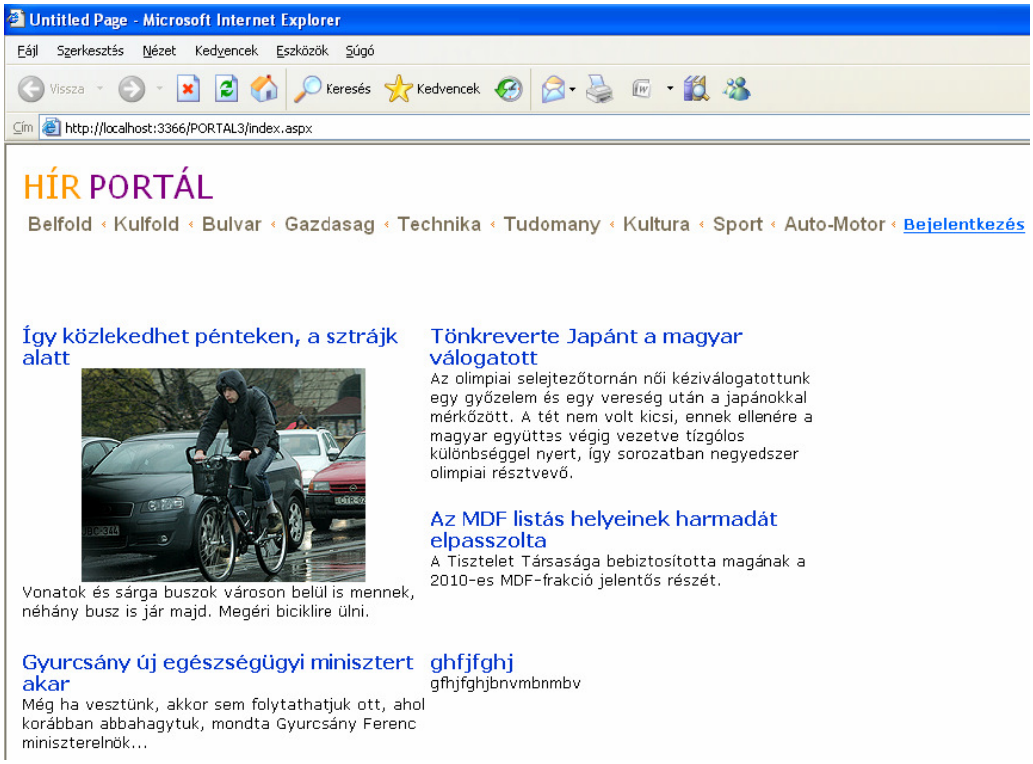
Látogatói nézet

Látogatói nézetben a látogató a már meglévő hírek között böngészgethet.
Mikor a látogató megérkezik a weboldalra, a főoldalon (index.aspx) találja magát, ahol a cikkszerkesztő által összeállított legfontosabb illetve legújabb cikkek közül böngészhet. Minden cikk tartalmaz egy főcímet. Ez egyben egy hiperhivatkozást tartalmaz, mely egy kattintással a cikk teljes tartalmának megjelenítését teszi lehetővé, illetve tartalmaz egy rövid leírást, mely a cikk teljes tartalmára utaló pár mondat hosszú szövegrészt jelent. (Ez a rövid leírás képet is tartalmazhat.)
Amennyiben a főoldalon található cikkek mennyisége nem elégíti ki a látogató kíváncsiságát, vagy nem tartalmaz a látogató érdeklődésének megfelelő cikkeket, úgy a látogató az oldal felső részén elhelyezkedő menüsorból navigálhat az adatbázisban tárolt összes cikk között úgy, hogy az általa kívánt kategóriát kiválasztva a meglévő hírek közül csak és kizárólag a választott kategóriába tartozó hírek jelennek meg, a létrehozás dátuma rendezésében.(előbb a legújabb, azt követve a későbbi hírek)
A kategória kiválasztásakor a menüsor alatt egy almenü jelenik meg, melyben a már kiválasztott kategóriához tartozó alkategóriák jelennek meg. Ezen alkategóriák segítségével a találati lista tovább szűrhető, így a találati lista a legjobban optimalizálható, azaz a látogató a lehető legközelebb kerülhet a számára legérdekesebb cikk kiválasztásához. (Kategória - Alkategória)

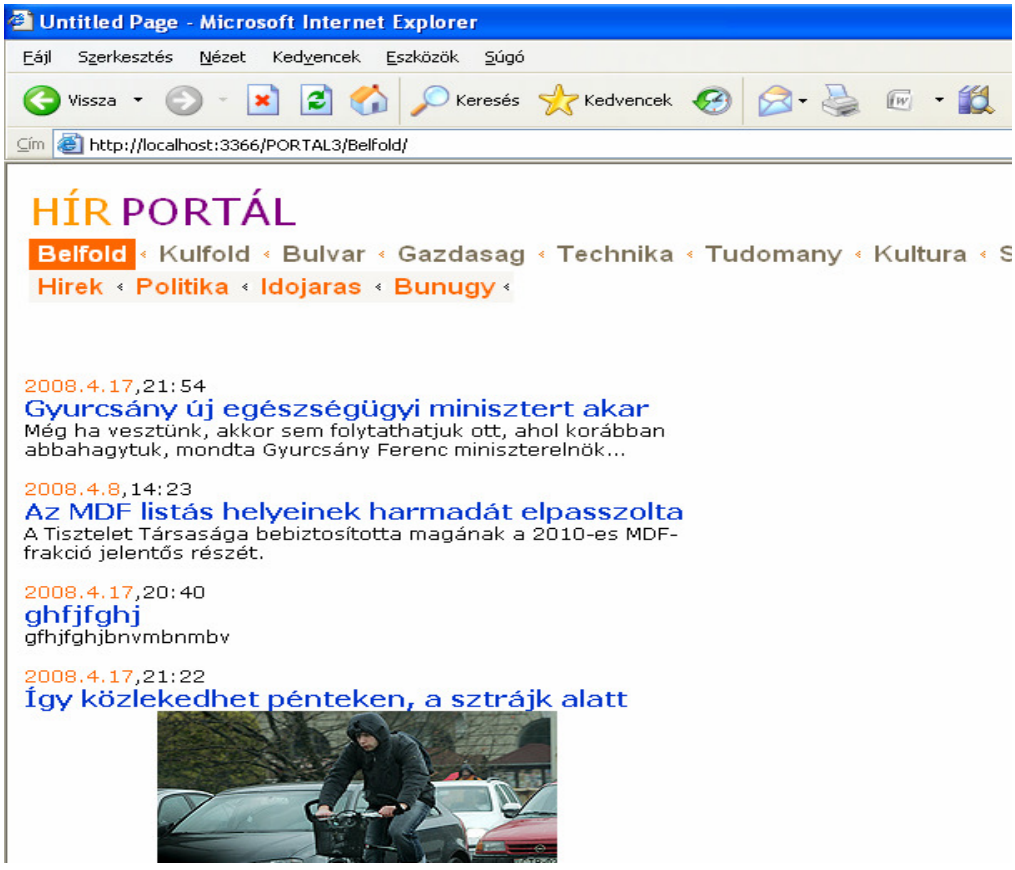
A találati lista cikkenként a következő információkat jeleníti meg:

- A létrehozás dátuma
- A létrehozás időpontja
- A cikk főcíme. Ez egyben egy hiperhivatkozást tartalmaz, mely egy kattintással a cikk teljes tartalmának megjelenítését teszi lehetővé.
- Cikk rövid leírása. Ez 1-2 mondat hosszú, a cikk teljes tartalmára utaló leírás lehet.

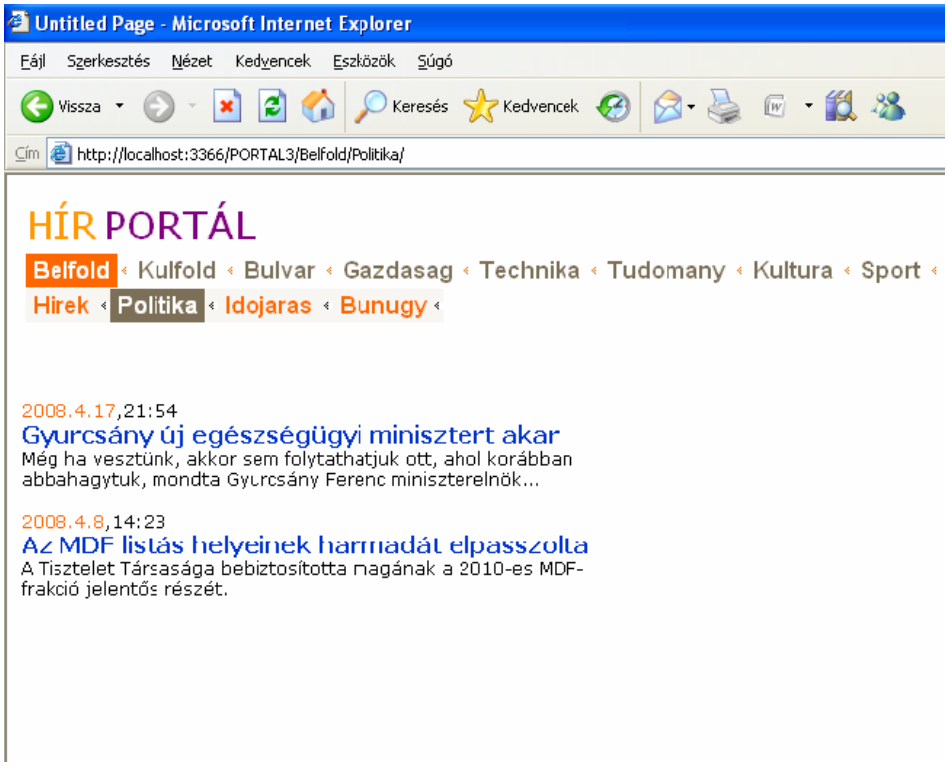
Néhány pillanatkép kiragadásával könnyen áttekinthető, hogy a látogató a portál használatakor milyen eshetőségekkel találkozhat:



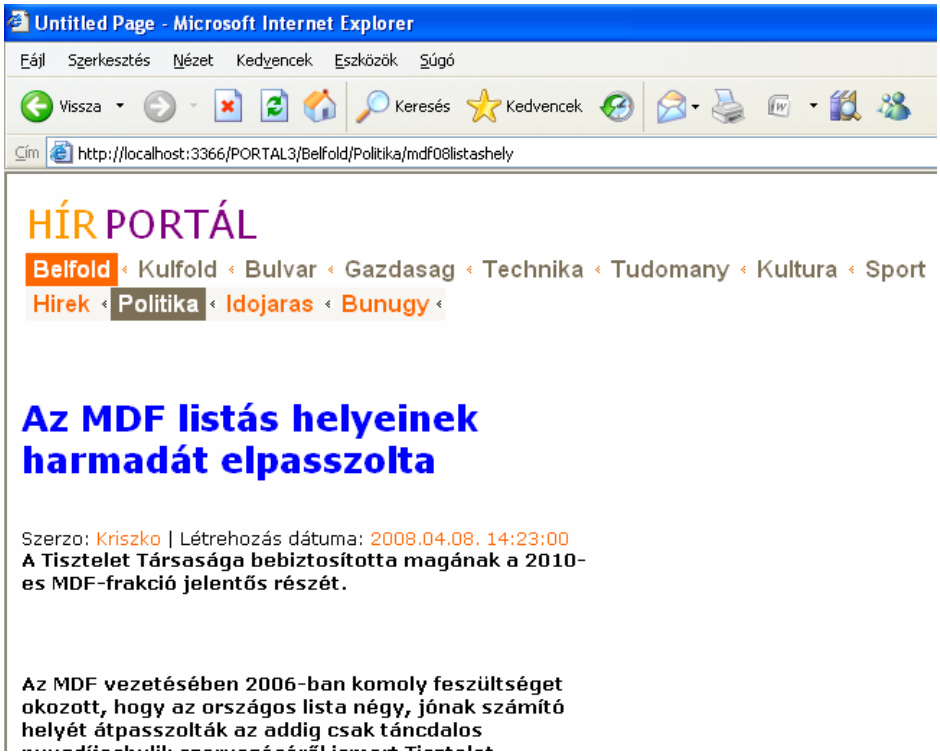
5. ábra címlap nézet



6. ábra kategória nézet



7. ábra alkategória nézet



8. ábra cikk nézet

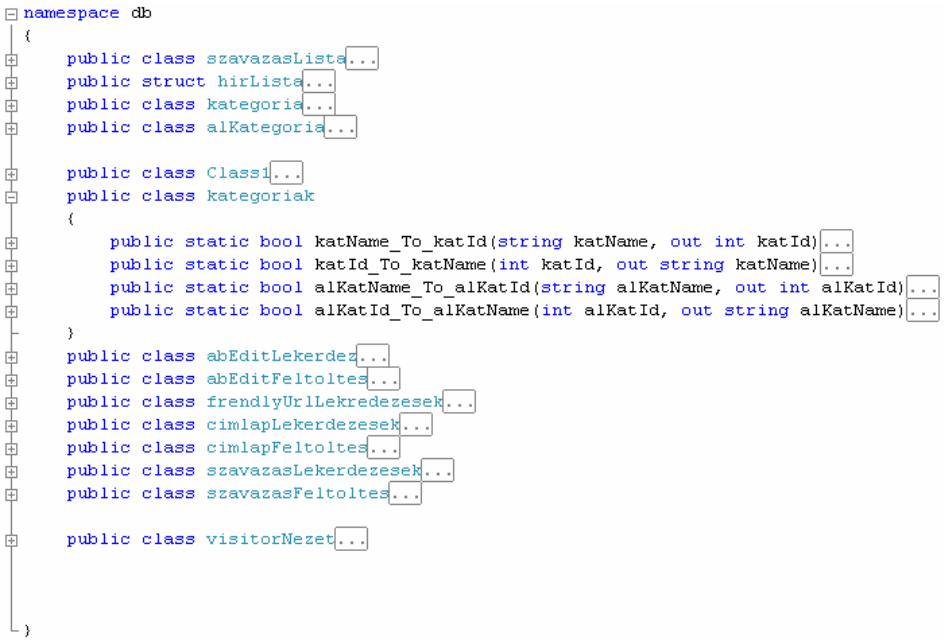
Hírszerkesztői nézet

A szerkesztőségi rendszer felépítése

A portál szerkezeti felépítése 2 részre osztható:

- adatbázis szint
- alkalmazás szint

A programozás során az adatbázis műveletekre vonatkozó metódusok teljes mértékben elkülönítésre kerültek az alkalmazás többi részétől. Az alkalmazás adatbázis lekérdezéseit, frissítéseit...stb megvalósító metódusok egy külön 'db' nevezetű névtéren belül, külön osztályokban lettek létrehozva, funkciójuktól függően, melyeket a következő ábra szemléltet:



A 'db' névtér egy külön osztálykönyvtárba került elhelyezésre, mely osztálykönyvtár egy külön .dll kiterjesztésű fájlba fordítódik le.

Minden egyes adatbázis műveletet végző metódus visszatérési értéke boolean típusú. Erre azért van szükség, mert minden adatbázis műveletet végző metódus egy try-catch kivételkezelőt tartalmaz. Ennek működése során, amennyiben a try blokkban kivétel keletkezik, abban a pillanatban a program futás a try –hoz tartozó catch ágban folytatódik, ahol a kivételt maszkolni lehet. Jelen esetben a kivétel bekövetkeztekor a visszatérési érték false értékű lesz, ellenkező esetben pedig true értékű. Így a webalkalmazás élete során nagyon jól elkülöníthető az adatbázis illetve általános hiba. Ezt a következő példa szemlélteti:

ADATBÁZIS MŰVELET KÓDJA:

```
...
public static bool cikkhezVanESzavazasCsatolva(int hirId, out int
szavazasId)
        {
            szavazasId = 0;
            try
            {
                SqlConnection kapcsolat = new SqlConnection();
                kapcsolat.ConnectionString =
ConfigurationManager.ConnectionStrings["adatBazisCS"].ToString();
                SqlCommand sqlConn = new SqlCommand();
                sqlConn.Connection = kapcsolat;
sqlConn.CommandText = "SELECT szavazasIndex FROM hirekTabla WHERE [index]="
+ hirId;
                kapcsolat.Open();
                szavazasId = (int)sqlConn.ExecuteScalar();
                kapcsolat.Close();
                return true;
            }
            catch (Exception ex)
            {
                return false;
            }
        }
...
```

WEBALKALMAZÁS KÓDJA:

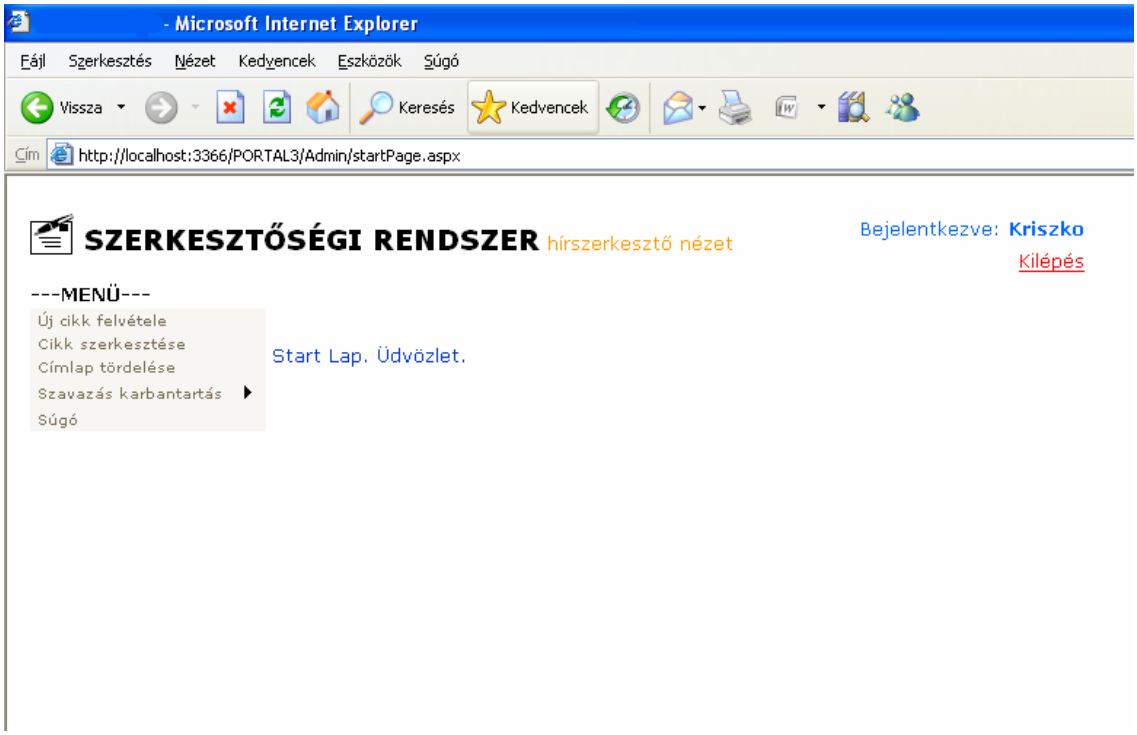
```
...
try
{
      if(!db.osztaly_nev.cikkhezVanESzavazasCsatolva(1,out xy))
      {
            Response.Redirect("~/Error.aspx?errorCode=2");
      }
      else
      {
            végrehajtandó_kód;
      }
}
catch(Exception ex)
{
      Response.Redirect("~/Error.aspx?errorCode=1");
}
...
```

A kód logikai működése illetve a felhasználó számára a hiba kijelzése:

A kód elérkezik egy adatbázis lekérdezéshez. Meghívódik a 'db' névtérben az adott osztályban az adott metódus. Amennyiben az sikeresen lefut, úgy a metódus visszatérési értéke true lesz és végrehajtandó_kód lefut. Ellenkező esetben, ha kivétel váltódik ki az adatbázis lekérdezés során, úgy nem az else ágban elhelyezett végrehajtandó kód fut le hanem az if után elhelyezett Response.Redirect utasítás, mely a paraméterben megadott URL –re irányít át. A paraméterben megadott URL egy paraméterezett URL és ebben az esetben az 'errorCode' paramétere 2 –es értéket kapja, így tudatja az Error.aspx –el, hogy adatbázis hibáról van szó, azaz ha a program futása során ebbe az állapotba kerül, akkor az csak úgy következhetett be, hogy az adatbázis lekérdezés során egy kivétel keletkezett. Ebben az esetben a felhasználó számára az „Adatbázis hiba“ üzenet jeleni meg a kijelzőn. Az ilyenfajta hibakezelés célja, hogy a felhasználó a pontos hibaüzenettel ne találkozzon, mert az abban található információk azon kívül, hogy az átlagos olvasó számára nem mondanak túl sokat, a hozzáértő és rossz indulatú látogató számára támadási információkat jelenthetnek (pl. táblanév esetleges SQL injectionhöz).
Amennyiben a webalkalmazás az Error.aspx –re úgy irányít át, hogy az 'errorCode' paraméter 1-es értékre van állítva, úgy nem adatbázis művelet végzésekor következett be a kivétel, hanem bármilyen más tevékenység végrehajtásakor (pl egy string típus int –re való konvertálásakor), azaz általános hibáról beszélünk, így a felhasználónak az „Általános hiba“ felirat jelenik meg a kijelzőn.

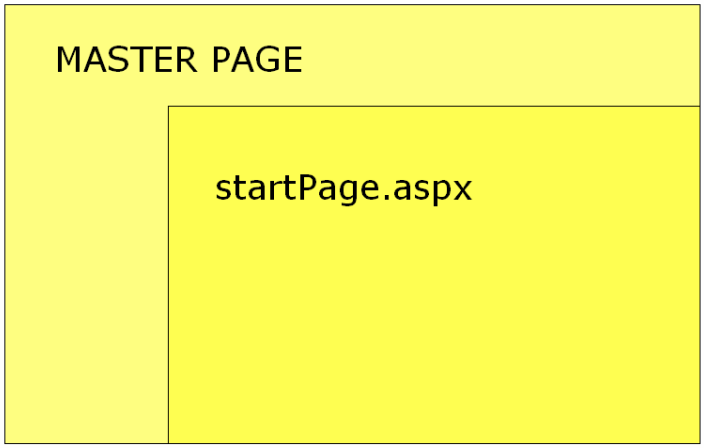
A portál funkcionális felépítése:

A főoldalon (persze ez tetszőleges helyre áthelyezhető) található bejelentkezés feliratú hiperlink a webszerkesztőségi rendszer beléptető oldalára navigál. A szerkesztőségi rendszerbe helyes jelszó és felhasználói név páros megadásával lehet belépni. Ennek hiányában a belépés sikertelen. A beléptető rendszerre azért van szükség, hogy illetéktelenek ne tudják szerkeszteni az oldalt, illetve a cikk kizárólag a felhasználói név ismeretével adható az adatbázishoz. Amennyiben sikeres bejelentkezésre kerül sor, úgy a következő üdvözlőkép fogadja a hírszerkesztőt (startPage.aspx):



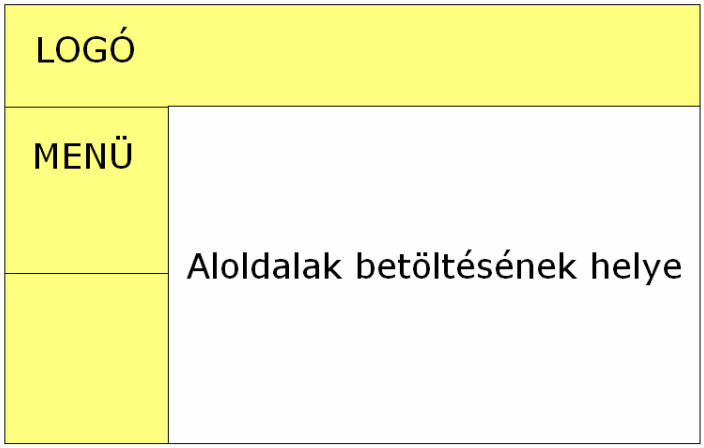
9. ábra Szerkesztőségi rendszer üdvözlő képernyő

A startPage.aspx felépítése a következő:



10. ábra StartPage felépítése

A MasetrPage felépítése:



11. ábra MasterPage felépítése

A MasterPage menüjében navigálhatunk a szerkesztőségi rendszer nyújtotta szolgáltatások között, melyek a következők:

- Új cikk hozzáadása
- Meglévő cikk szerkesztése
- Címlap tördelése
- Szavazás karbantartás
    - Szavazás létrehozása
    - Meglévő szavazás szerkesztése
- Súgó

A megadott szolgáltatások meghívásakor a szolgáltatásnak megfelelő aspx fájl töltődik be. Minden aspx fájl ugyanazt a MASTERPAGE –et tartalmazza, tehát a masterpage tartalma az egész szerkesztőségi rendszeren belül statikus és állandóan látható. A masterpage másik előnye, hogy ha a szerkesztőségi rendszer sablon szerkezetén változtatni kell, azt nem az egyes oldalakon külön-külön kell megtenni, hanem egyetlen helyen egyszeri módosítással végrehajtható.

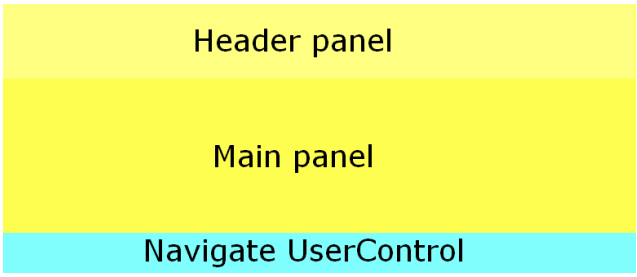
Szolgáltatások felépítése

Új cikk hozzáadása

Az új cikk hozzáadása menüpont meghívásakor az EngineAdd.aspx oldalhoz tartalmazó kód töltődik be.

Az oldalnak egy ASP MultiView kontroll az alapja. A MultiView kontroll tulajdonsága, hogy a kontrollon belül úgynevezett View –ek definiálhatok, és ezen Viewek közül egyszerre egy jelenik meg a képernyőn. Minden view 3 részből épül fel:

- HEADER panel
- MAIN panel
- FOOTER panel



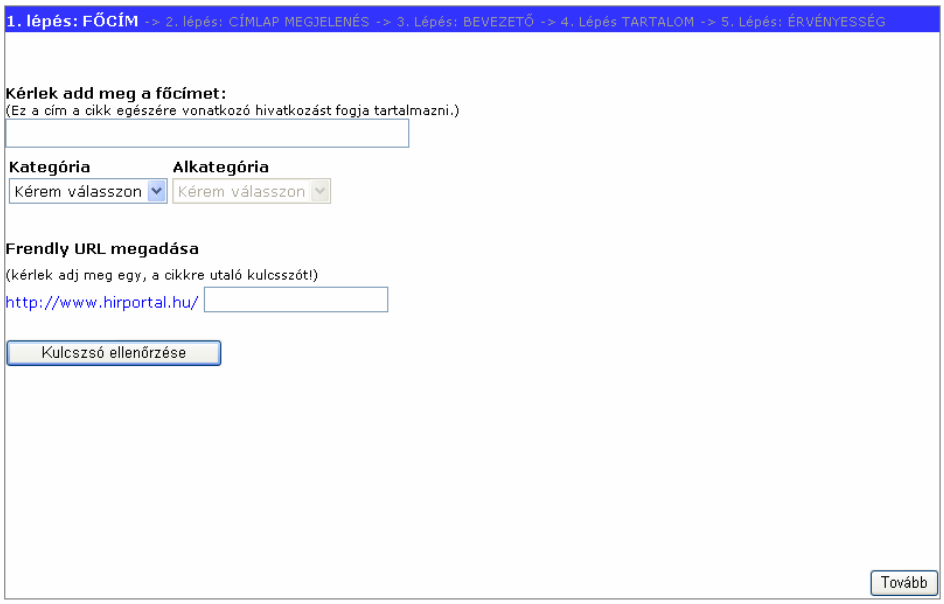
12. ábra View kontrol felépítése

A HEADER panel az aktuális *View* számát illetve az adott panelen végezhető művelet nevét tartalmazza. Az aktuális *View* számát illetve a hozzá kapcsolódó művelet leírását fehér és félkövér betűtípus , míg a nem látható *View* –ek jellemzőit szürke, normál betűtípus jelzi.

A FOOTER panelen egy *UserControl* ASP.NET eszköz objektum példánya került elhelyezésre. Ez a *UserControl*, mint neve is elárulja (navigateUC) a *View* –ek közötti navigációért felelős.

Az új cikk hozzáadása menüpontban a következő 5 db View került beépítésre:

1.VIEW:



13. ábra A MultiView kontrol első View kontrolja.

Header panel. Adekvát az előzőekben ismertetésre kerülttel.

Main Panel: Felhasznált kontrollok és tulajdonságaik: *TextBox* kontrolt, mely a létrehozandó cikk főcímének bevitelére szolgál. Ehhez tartozik egy *RequiredFieldValidator* validátor.

Két darab *DropDownList* kontrol egyik a kategóriák megjelenítését másik pedig az alkategóriák megjelenítését teszi lehetővé. Mindkét dropdownlist panel –hez tartozik egy-egy *RequiredFieldValidator* validátor, illetve egy-egy *CascadingDropDown* extender (továbbiakban *CCD*). A kategória legördülő lista feltöltését a hozzátartozó *CCD* extender végzi, mely kategória SQL táblából kapott információk alapján feltölti a legördülő listát a kategória táblában található elemekkel.

Mivel a legördülő listák egymástól függenek, vagyis a második értéke az elsőtől függ ezért meg kell oldani, hogy az kategória legördülő listában mikor kiválasztunk egy elemet, úgy az alkategória kiválasztására szolgáló legördülő listában kizárólag a kiválasztott kategóriához tartozó alkategória elemek jelenjenek meg. Ehhez az ASP.NET egy nagyon egyszerű megoldást kínál:

Létrehozunk egy *GetAlkat2*() metódust, mely rendelkezik egy *[WebMethod]* attribútummal, ami azt jelzi, hogy publikáljuk ezt a metódust a kliens felé. Ennek az eljárásnak a visszatérési értéke *CascadingDropDownNameValue[]* tömb lesz. Ennek a tömbnek az elemei *CascadingDropDownNameValue* típusúak lesznek. Az adatbázisból történő lekérdezést egy

DataTable táblában kapjuk vissza, melynek sorai DataRow formában a kívánt alkategória nevét illetve *ID* azonosítóját tárolja. Az adattáblát sorról sorra véve kiolvassuk a sor elemeit, melyekből egy új *CascadingDropDownNameValue* objektumot létrehozva egy Listához fűzünk hozzá.
Miután elértük az utolsó sort a létrehozott lista *toArray()* metódusával a listát tömb típussá konvertáljuk, így egy *CascadingDropDownNameValue[]* tömböt kapunk, mellyel a metódus visszatér:

```
[WebMethod]
public CascadingDropDownNameValue[] GetAlKat2(string knownCategoryValues)
{
StringDictionary kv =
CascadingDropDown.ParseKnowCategoryValuesString(knownCategoryValues);
             int AlKatId;
             if (!kv.ContainsKey("kategoria") ||
            !Int32.TryParse(kv["kategoria"], out AlKatId))
            {return null; }
      dsAlKategoriaTableAdapters.alKategoriaTableAdapter adapter = new
      dsAlKategoriaTableAdapters.alKategoriaTableAdapter();
      dsAlKategoria.alKategoriaDataTable alkategoria =
      adapter.GetAlKategoria(AlKatId);
      List<CascadingDropDownNameValue> values = new
      List<CascadingDropDownNameValue>();
       foreach (DataRow dr in alkategoria)
      {
            values.Add(new
            CascadingDropDownNameValue((string)dr["alKatName"],
            dr["alKatId"].ToString()));
      }
return values.ToArray();
}
```

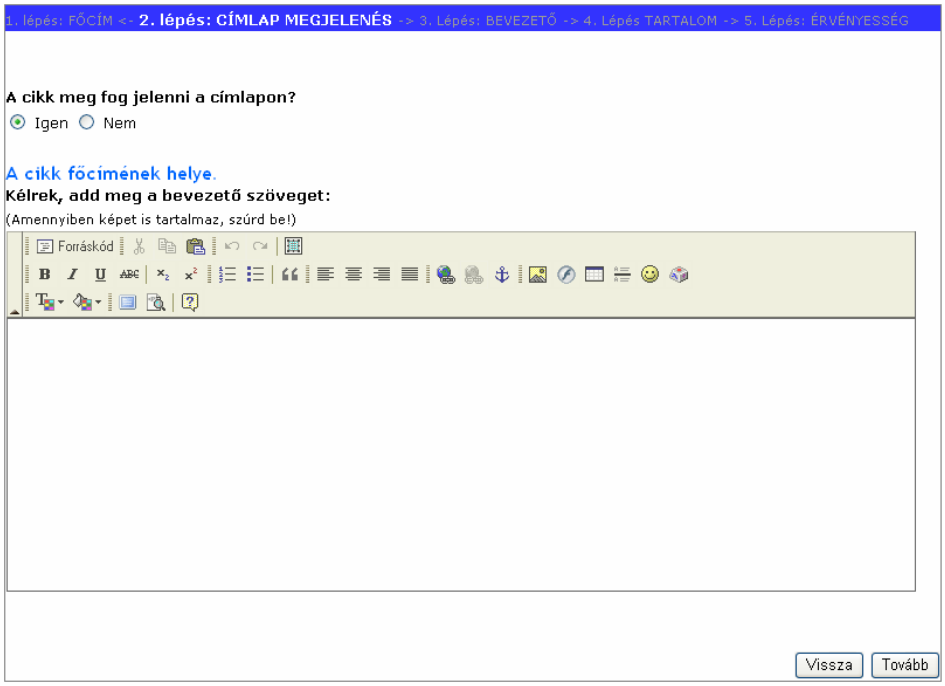
Ennek segítségével az alkategóriához tartozó *CCD* extender feltölti az alkategória legördülő listáját a megfelelő értékekkel.

Az első *View* utolsó eleme pedig a *Frendly URL* cikkazonosítójának megadására szolgál. A cikkazonosítónak kizárólag alkategórián belül kell egyedinek lennie, tehát két különböző alkategóriában szereplő cikk *FrendlyURL* cikkazonosítója megegyezhet, azonos kategórián szereplő cikk *FrendlyURL* cikkazonosítójának különböznie kell. Az azonosság kizárásáról egy *Button* kontroll Click eseménye gondoskodik, mely esemény bekövetkeztére definiált metódus hívódik meg. Ez a metódus egy egyszerű SQL lekérdezést tartalmaz, mely azt vizsgálja, hogy a hirekTabla táblánkban létezik e már azonos kategóriában illetve azonos alkategóriában az aktuálisan megadott *FrendlyURL* cikkazonító. Az első *View* –ről addig nem

engedélyezett a továbblépés, míg az ellenőrzés meg nem történt, vagy ha megtörtént, az ellenőrzés eredménye nem hozza az előzőekben ismertetett számunkra pozitív eredményt.

Amennyiben az első View –on minden input mező megfelelően kitöltésre került, úgy a tovább gombra kattintással a a 2. View jelenik meg.

2. VIEW:



14**. ábra A MultiView kontrol második View kontrolja.**

Header panel: Adekvát az előzőekben ismertetésre kerülttel.

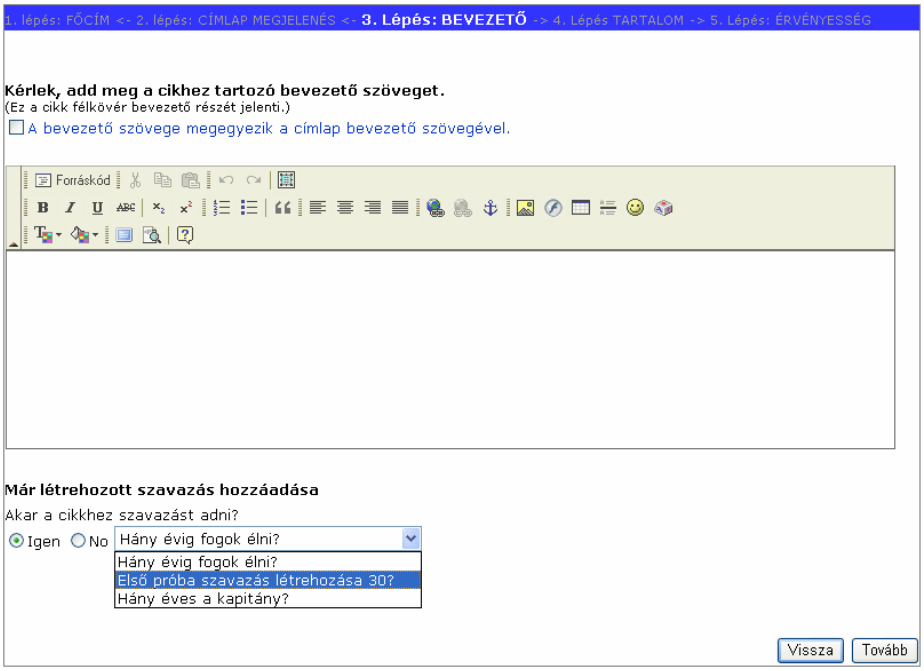
Main panel: A felhasználónak egy kérdés jelenik meg arra vonatkozóan, hogy a létrehozandó cikk szerepel e majd a címlapon. Amennyiben nem szerepelhet, úgy a tovább gombra kattintással a következő *View* –ra navigálhatunk. Ellenkező esetben a rendszer kéri a cikkhez tarozó rövid leírást, mely leírásba képet is ágyazhatunk.

Ez a következőképpen valósítottam meg: A felhasználó a kérdésre két válaszlehetőséget adhat. A válaszlehetőségeket *RadioButton* kontrol segítségével adhatja meg. Első radioButton az igen választ, míg a második a nemleges választ képviseli. Az első *radioButton -t* választva

egy *FCKEditor* tűnik fel. Az *FCKEditor -ba* az előző fejezetben ismertetett módon vihetünk be szöveget, tölthetünk be képet stb.

Amennyiben a cikk címlap szövegének bevitele befejeződött, úgy a *View* alján található tovább gombbal a 3. *View* –re navigálhatunk.

3. VIEW:



15. ábra A MultiView kontrol harmadik View kontrolja.

Header panel. Adekvát az előzőekben ismertetésre kerülttel.

Main panel: Ebben a *View* –ban a felhasználónak a cikk bevezető szövegét kell megadnia illetve választania, hogy a létrehozandó cikkhez kíván e csatolni szavazást.

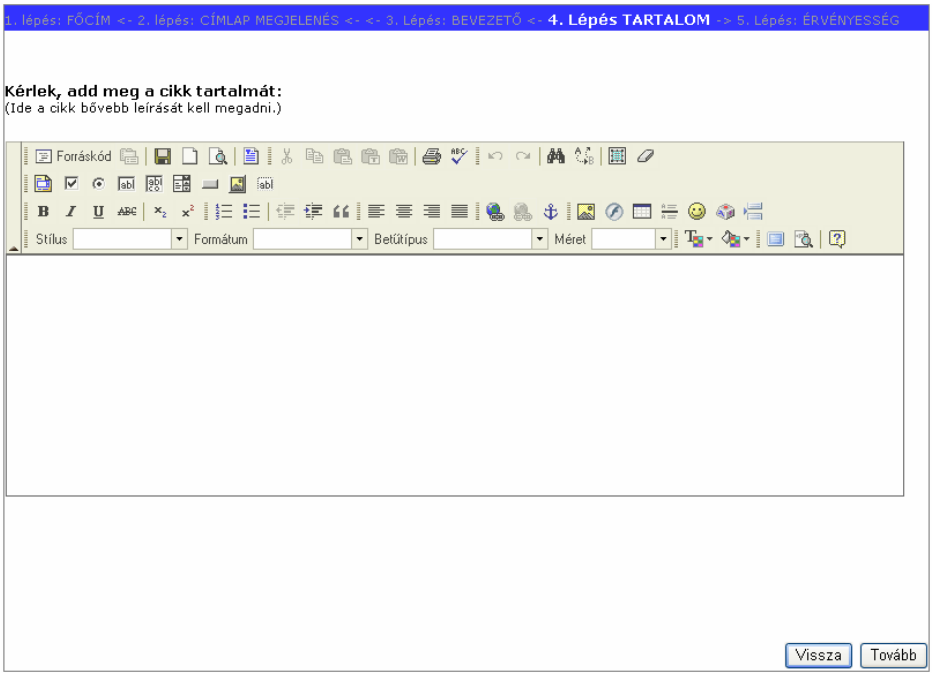
A cikk bevezető szövegének megadása a címlap szövegének megadásának módjával azonos annyi különbséggel, hogy a felhasználó a cikk bevezető szövegének a címlap szövegét is megadhatja. Ebben az esetben nincs szükség az azonos szöveg újbóli bevitelére. A címlap szövegének másolását egy *ChexBox* kontrol segíti. Amennyiben a létrehozandó cikkhez az előző *View* –ban adtunk meg címlap szöveget, úgy az *FCKEditor* felett elhelyezkedő *CheckBox* kontrol használhatóvá válik és bepipálásakor címlap szövegeként már előzőleg

bevitt tartalom automatikusan átmásolódik az aktuális *FCKEditor -ba*, így könnyítve a felhasználó dolgát.

Ha a felhasználó úgy dönt, hogy a létrehozandó cikkhez szavazást szeretne csatolni, úgy egy *radioButton* kontrol áll a rendelkezésére, mely segítségével kiválaszthatja, hogy szeretne e szavazást csatolni vagy sem. Az igen kiválasztása esetén a letiltott legördülő lista használhatóvá válik. A listába automatikusan betöltésre kerülnek az aktuálisan érvényes szavazások kérdései. Ezek alapján a a felhasználó választhat.

Mindezek után a *View* jobb alsó sarkában található tovább gombbal a 4. *View* –ra navigálhatunk.

4.VIEW:

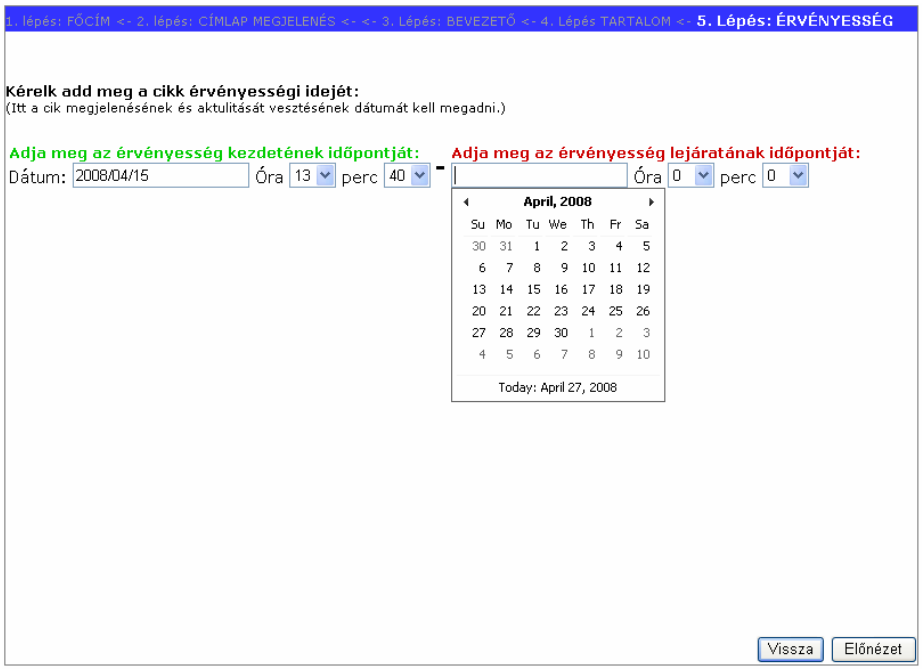


16**. ábra A MultiView kontrol negyedik View kontrolja.**

Header panel. Adekvát az előzőekben ismertetésre kerülttel.
Main panel: Ezen a View -on kizárólag egy darab FCKEditor –t található, mely segítségével a cikk tartalmának szövegét kötelezően lehet, illetve kell bevinni.

A bevitel végeztével a tovább gombbal az ötödik, egyben befejező **View** –ra navigálhatunk.

5. VIEW:



17**. ábra A MultiView kontrol ötödik View kontrolja.**

Header panel. Adekvát az előzőekben ismertetésre kerülttel.

Main panel: Itt két *TextBox* típusú beviteli mezőt találunk, melyek a cikkhez tartozó dátumok beállítására szolgálnak. Az első beviteli mezőbe a cikk érvényességének kezdetét adhatjuk meg, míg a második beviteli mező a cikk érvényesség lejártának megadására szolgál. Cikk érvényessége alatt az a dátum értendő, amikortól a cikk megjelenhet a látogató számára a portálon, érvényesség lejárata alatt pedig a cikk aktualitásának vége értendő. Az aktualitás vége dátumnak többek között a címlapon tulajdoníthatuk szerepet, például ha egy cikk aktualitását vesztette, már nem célszerű, hogy megjelenítsük azt a címlapon.

A beviteli mezők kitöltésére két lehetőséget kínál a rendszer:

1. ***Amennyiben a felhasználó manuálisan, a billentyűzet segítségével kívánja a mezőket kitölteni.***

Ebben az esetben fontos, hogy a beviteli mezőkbe vitt értékek kizárólag számjegyek legyenek, így ennek vizsgálata a rendszer feladata, melyek validátorok segítségével oldottam meg:

Mindkét beviteli mezőre különböző validátorok lettek kötve.

RequireFieldValidator validátor annak érdekében, hogy a felhasználó addig ne tudja befejezni a cikk felvitelét, míg ezeket a mezőket ki nem töltötte, tehát kitöltésük kötelező.

Illetve egy *RegularExpressionValidator* validátor, annak érdekében, hogy a dátumok megfelelő formátumban kerüljenek megadásra. Ez azért fontos, mert az adatbázis feltöltés ezen mezők értékével kerülnek feltöltésre és a helyes formátum megadására történő kényszerítéssel elkerülhetők a további konverziók illetve SQL hibák.

A megadható dátum formátumok:
ÉV/HONAP/NAP
ÉV-HONAP-NAP
Továbbá az elválaszto / illetve – karakterek tetszőlegesen variálhatoak a dátum megadásánál.

2. Amennyiben a felhasználó a rendszer nyújtotta kényelmesebb lehetőséget választja és a dátumok bevitelét egy felugró ablakban megjelenő Calendar kontrol segítségével végzi:
A bevitelt ebben az esetben egy, a beviteli mezőhöz rendelt *Calendar* kontrol végzi. A felhasználó csupán a felbukkanó *Calendar* kontrolból választja ki a kívánt dátumot. A *Calendar* kontrol a beviteli mezőre történő fokuszáláskor aktivizálódik, mely egy klasszikus értelemben vett naptárat bocsájt a felhasználó részére. A naptáron belül az egér segítségével navigálhatunk az évek, hónapok illetve napok között. A *Calendar* kontrol `Format="yyyy/MM/dd"` tulajdonságának beállításával a rendszer számára helyes formátumban adja vissza a kiválasztott dátumot a beviteli mező számára. Az előzőekben részletezett *RegularExpressionValidator* validátor itt is jelen van, de itt nem tulajdonítunk neki különösebb jelentőséget.

Az 5. *View* alsó részében elhelyezkedő *navigatieMultiView.ascx UserControl* kontrolnak a *MultiView* kontrolon belül talán itt van a legnagyobb szerepe.
Két lehetőség közül választhat a felhasználó:
1: Visszalép az előző View –ra
2: Megtekinti a mostanára felvitt cikk előnézeti képét.

Mivel a navigációért felelő *UserControl* alapértelmezettként első két gombja a View –ek közötti navigációért felelős, itt már a második gomb (a Tovább feliratú) számunkra feleslegessé válik, ezért azt el kell tüntetni, hogy a felhasználó ne tudja használni azt. Illetve *UserControl* –ban létrehozott harmadik, alapértelmezettként láthatatlan gombot a felhasználó

számára láthatóvá kell tennünk és azt tulajdonságokkal kell felruházni a használhatóság érdekében.

Ez a következőképpen került kivitelezésre:

```
nav5.hideBtn(2);
nav5.hideBtn(3);
nav5.renameBonusButton("Előnézet");
nav5.bonusEsemeny += new navigateMultiView.OnClickDelegate(bonus3_Click);
```

A kód sorról sorra a kivetkező műveleteket végzi el:

('*nav5*' az 5. *View* –on létrehozott *UserControl* példányának egyedi azonosítója.)

Meghívódik a *navigateMultiView* osztály *hideBtn()* metódusa mely a következő kódot tartalmazza:

```
public void hideBtn(int n)
{
	int k = n;
	switch (k)
	{
		case 1: btnPrev.Visible = false; break;
		case 2: btnNext.Visible = false; break;
		case 3: btnBonus.Visible = true; break;
		default: break;
	}
}
```

ez a metódus a UserControl –ban elhelyezkedő gombok Visible tulajdonságát hivatott állítani. Amennyiben a paraméterül kapott int típusú szám értéke 1, úgy a visszalépésért felelős gomb Visible tulajdonsága false értékre állítódik és az első gomb a felhasználó számára láthatatlanná válik. Amennyiben a paraméterül kapott int típusú szám értéke 2, úgy a továbblépésért felelős gomb Visible tulajdonsága false értékre állítódik és az második gomb a felhasználó számára láthatatlanná válik. Amennyiben a paraméterül kapott int típusú szám értéke 3, úgy a 'bonus' nevű általános tulajdonsággal felruházható gomb Visible tulajdonsága true értékre állítódik és ez a gomb a felhasználó számára láthatóvá válik.

A továbblépésért felelős gombot elrejtjük a felhasználó elől.

A '*bonus*' gombot a felhasználó számára láthatóvá tesszük. A program ezen pillanatában ez a gomb már látható, de hozzá semmilyen esemény nincs rendelve, illetve a gomb felirata is alapértelmezett módon *Button.*

A '*bonus* ' gomb feliratát "*Előnézetre*" állítjuk a *navigateMultiView* osztály *RenameBonusButton()* metódus segítségével. Ez a metódus annyi tesz, hogy a paraméterül kapott string típusú értéket adja a gomb *Text* tulajdonságának:

```
public void renameBonusButton(string s)
{
      string sz = s;
      btnBonus.Text = sz;
}
```

A program ezen pillanatában, a '*bonus*' gombunk már megjelenik a felhasználó számára és felirata is árulkodik a gombra történő kattintást következményéről, viszont a gombunkhoz még nem rendeltünk esemény így nem történik semmi a gombra történő klikkeléskor. A megadott kód 4. sora ezt az esemény hozzárendelést végzi a következő módon:

Mivel a *bonusGomb* létrehozásakor még nem tudtuk, hogy a gombunk milyen eseményt fog kiváltani ezért konkrét metódust sem tudtunk az eseményhez rendelni. Ezt fordítási vagy futási időben szeretnénk megoldani, viszont az gomb alapeseményei láthatóságuk révén kívülről nem hivatkozhatok. Ezt az úgynevezett delegált segítségével valósítja meg a rendszer, mely a a *navigateMultiView* osztályon belül került elhelyezésre. Kódja:

```
public delegate void OnClickDelegate(object sender, EventArgs e);
public event OnClickDelegate bonusEsemeny;
protected void btnBonus_Click(object sender, EventArgs e)
{
      if (bonusEsemeny != null) bonusEsemeny(sender, e);
}
```

Ezzel létrehoztunk a '*bonus*' gombra egy '*bonusEsemeny*' eseményt, mely a public hatáskör révén kívülről is látható, hivatkozható, módosítható.

A legutóbbi kódrészlet 3-6. sorában elhelyezett kód azt adja meg, hogy a 'bonus' gombra történő klikkeléskor hajtódjon végre a '*bonusEsemeny*' esemény. Viszont ez '*bonusEsemeny*'

esemény egyenlőre végrehajtandó kódot nem tartalmaz, egyenlőre nem történik semmi. Ehhez végrehajtódó kódot majd a használatának helyén (esetünkben az 5. *View* –ban) rendelünk.

Az 5. View ismertetése során megadott

```
navigateMultiView.OnClickDelegate(bonus3_Click);
```

sor a '*bonusEsemeny*' eseményhez végrehajtandó kód hozzárendelését végzi el. Tehát most már a '*bonus*' gombunkra történő klikkeléskor az EngineAdd osztály *bonus3_Click()* metódusa fog végrehajtódni, mely metódus következményéről már a '*bonus*' gomb felirata is utal („*előnézet*“).

Tehát az „*előnézet*“ feliratú gombra történő kattintáskor a következő metódus kerül végrehajtásra. Ez a metódus az előnézeti képet teremti meg számunkra. Az előnézeti kép egy panel kontrolon elhelyezett több *Label* kontrol biztosítja. Ehhez az előnézeti panelhez egy *ModalPopipExtendert* kapcsoltam, mely extender *show()* metódusa meghívásakor az előnézeti panel láthatóvá válik, így a felhasználó szembesülhet a felvitt cikk látványával. Innen két lehetőség adódik a felhasználó számára. Elmenti a létrehozott cikket az adatbázisba, vagy visszalép és módosítja azt. Elmentés esetén az EngineAdd osztály *addNewCikkToDataBase()* metódusa hívódik meg, mely beviteli mezők beolvasása után az onnan vett értékeket egy SQL *INSERT* parancs segítségével elmenti az adatbázisba.

Ezzel egy új cikket adtunk az adatbázisunkhoz.

Cikk szerkesztése

A Cikk szerkesztése menüpont kiválasztásakor az *EngineEdit.aspx* fájl töltődik be a böngészőbe.

Mint már említésre került az oldalhoz ugyanaz a *MasterPage* tartozik, tehát a *MasterPage* tartalmi része itt is látható. (pl.menü)

Az oldal betöltődésekor cél az volt, hogy a felhasználó számára lehetővé kell tenni nem csak az adatbázisból való cikkek listájának megjelenítését, hanem a portál használata során töménytelen mennyiségben feltöltött cikkek közül is a felhasználó könnyedén megtalálja a szerkeszteni kívánt cikket, azaz a találati lista szűkíthető legyen bizonyos feltételekkel megadásával.

Ezt a Szerző neve, kategória, alkategória illetve a létrehozás dátuma szerinti szűrés valósítja meg.

Az oldal betöltődésekor 3 legördülő lista áll a felhasználó szolgálatára.

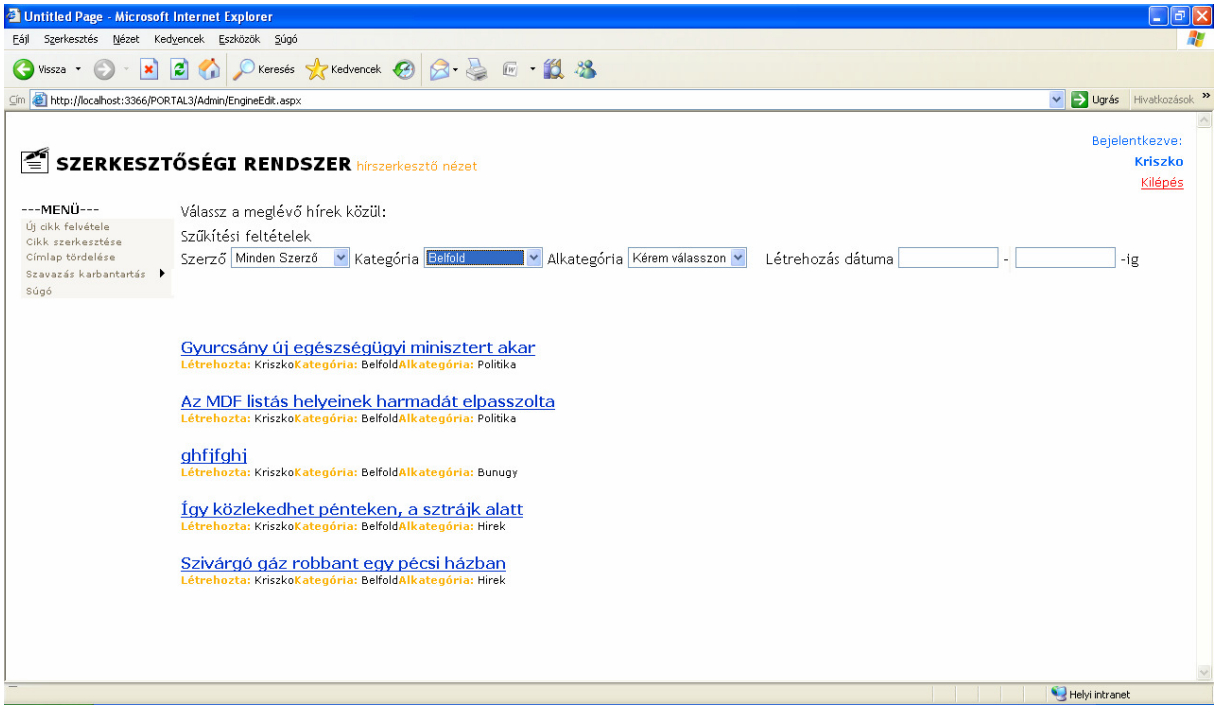
18. ábra Cikk szűrés 3 legördülő listája

Az első legördülő menü segítségével a szerző nevére tudunk szűrni, tehát amennyiben választunk egy értéket a legördülő listából, úgy azok a cikkek jelennek meg melyeket a kiválasztott szerző hozott létre. Amennyiben nem ismerjük a szerző nevét, úgy választható opcióként szerepel a *'bármely szerző'* érték. Továbbá a szerző kiválasztásakor a kategória kiválasztására szolgáló legördülő listában kizárólag azon kategória értékek jelennek meg, melyekben a kiválasztott szerző már hozott létre cikket. Így egy olyan találati listát kapunk, mely a *hirekTabla* adatbázis táblából kizárólag azokat a cikkeket jeleníti meg, melyekben a kiválasztott szerző szerepel szerzőként illetve a kategória is azonos a kiválasztott kategóriával. A harmadik legördülő lista is ugyanúgy függ a kategória legördülő listától, mint a kategória lista függ a szerző kiválasztására szolgáló legördülő listától. A kategória illetve az alkategória legördülő listák mindaddig inaktív állapotban vannak, míg az őt megelőző legördülő listában választás nem történt.

Tehát az alkategória kiválasztásával már egészen közel kerülhetünk a keresett cikk megtalálásához. Amennyiben még így is nagyméretű találati listát kapunk, úgy tovább

szűkíthetünk a létrehozás dátumának megadásával. Erre kettő *TextBox* típusú beviteli mező szolgál. Az első beviteli mezővel a dátumintervallum kezdetét, míg a második mezővel a dátumintervallum végét adhatjuk meg.

A találati listamegjelenítésére egy *UserControl* –t használtam. Ezt a *hirekUC.ascx* fájl implementálja. A *hirekUC UserControl* felépítése:
Az első sorban egy *LinkButton* kontrolt helyeztem el, mely linkbutton szövege a cikkhez tartozó címsort jeleníti meg.
Az adatbázis lekérdezés eredményeként visszakapott sorokat egyenként olvasva dinamikusan létrehozunk egy *hirekUC UserControl* objektum példányt és a megfelelő értékeket betöltjük, majd ezt az UserControl objektum példányt dinamikusan hozzáadjuk az oldalon elhelyezett eredmény panel kontrolhoz, és ennek eredményeképpen a a cikk Főcíme, szerzőjének neve, kategória besorolása, alkategória besorolása, létrehozás dátuma láthatová válik a képernyőn.
Amennyiben a felhasználó klikkel a címsort tartalmazó *UserControl* objektum példányában található *LinkButton*ra, úgy a klikkelés, az oldalra elhelyezett *UpdatePanel* –nek köszönhetően egy aszinkron postback eseményt okoz. Az aszinkron postback eseményt a *EngineEdit* osztály *Page_Load*() metódusában kezeltem le úgy, hogy az esemény bekövetkeztekor vizsgálom, hogy ki melyik *UserControl* –ra klikkelt a felhasználó, azaz melyik objektum volt a küldő. A küldő ismeretében már ismeretes, hogy melyik cikket választotta a felhasználó szerkesztésre.



19. ábra Cikk szűrése szerkesztéshez

A *Page_Load()* metódus utolsó lépése, hogy átirányítja a felhasználót az *EngineEditSelected.aspx* oldalra. Természetesen ebből az átirányítástól még nem derül ki, hogy mely cikket kell betölteni szerkesztésre. Tehát az átirányításnál a URL –t paraméterezni kell mégpedig úgy, hogy a megadom a küldő cikk azonosítóját:

```
Response.Redirect("EngineEditSelected.aspx?hirId="                    +
getKuldoId[2]);
```

Így például az első cikk kiválasztása esetében a következő linkre irányít át minket a *Page_Load*() metódus:

http://www.hirportal.hu/Admin/EngineEditSelected.aspx?hirId=1

Tehát a felhasználó választott, ekkor az *EngineEditSelected.aspx* fájl töltődik be a böngészőbe. Az oldal felépítése teljes mértékben megegyezik az előző pontban ismertetett *EngineAdd.aspx* felépítésével, működésében viszont annyi különbség van, hogy az oldal betöltődésekor végrehajtódik az *EngineEditSelected* osztály *adatokBetoltese()* metódusa. A metodus paraméterül kapja az URL hirId paraméterében kapott cikk azonosítot és a szerint egy adatbázis lekérdezéssel a metódus megkapja a cikkhez tartozó összes információt mely segítségével az összes beviteli mezőt feltölti a cikkre vonatkozó információval.(címsor,

kategória, alkategória, FrendlyURL kulcsszó, bevezetőszöveg…stb) Ezután a cikk az uj cikk hozzáadása részben ismertetett módon szerkeszthetővé válik a felhasználó számára.

Címlap tördelése

A címlap tördelése menüpont kiválasztásakor a EngineLayoutEditing.aspx töltődik be a böngészőbe.

Az oldal felépítése szerkezeti felépítése nagyon egyszerű. Sima ‹div› html elemek kerültek elhelyezésre az oldalban. Tartalmaz egy

‹div id=“dragAndDropMainContainer“› HTML elemet, mely azonosítóként megkapja az id tulajdonságban megadott sztringet. Ezen belül egy táblázat ‹div› került beszúrásra, mely az oldal tördelését szolgálja. A táblázaton belül egy sor és kettő oszlop lett definiálva. A táblázat második cellájában egy panel kontrol került beszúrásra. Ebben a panelben egy HTML elem (azonosítója "*elerhetoHirek*")és azon belül egy<ul>HTML lista (azonosítója "*box5*") található.

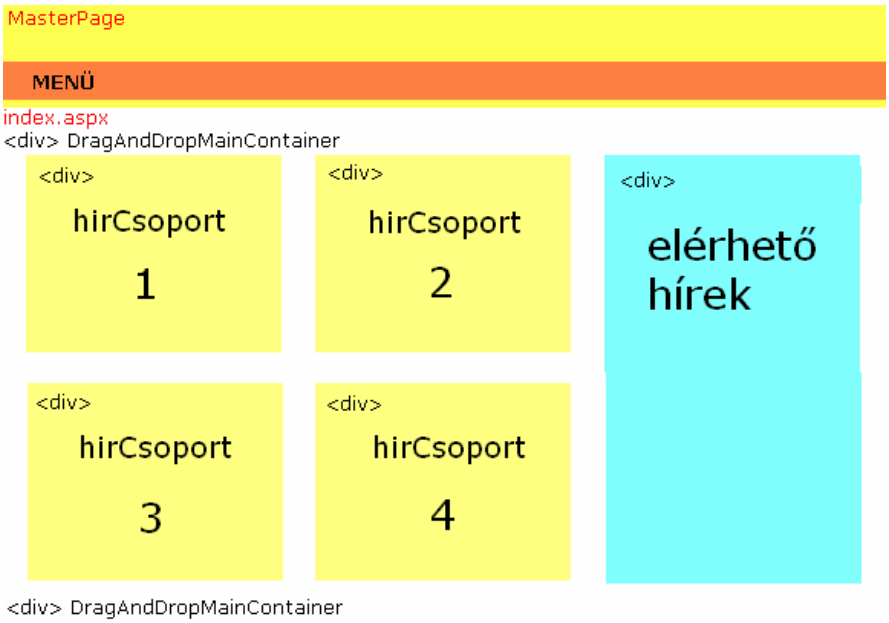
Ide kerülnek majd felsorolásra az adatbázisból azok a cikkek, melyeknél igaz, hogy az aktuális dátum az érvényesség kezdetének dátuma és érvényesség végének dátuma közé esik.

A táblázat első cellája egy újabb táblázatot tartalmaz, mely táblázatban sorok ‹tr› és sorokon belül oszlopok ‹td› lettek definiálva.

A táblázat adott celláiba ‹div› HTML elem került beszúrásra, melyek a következő id –ket kapják:

hirCsoport1, hirCsoport2, hirCsoport3, hirCsoport4

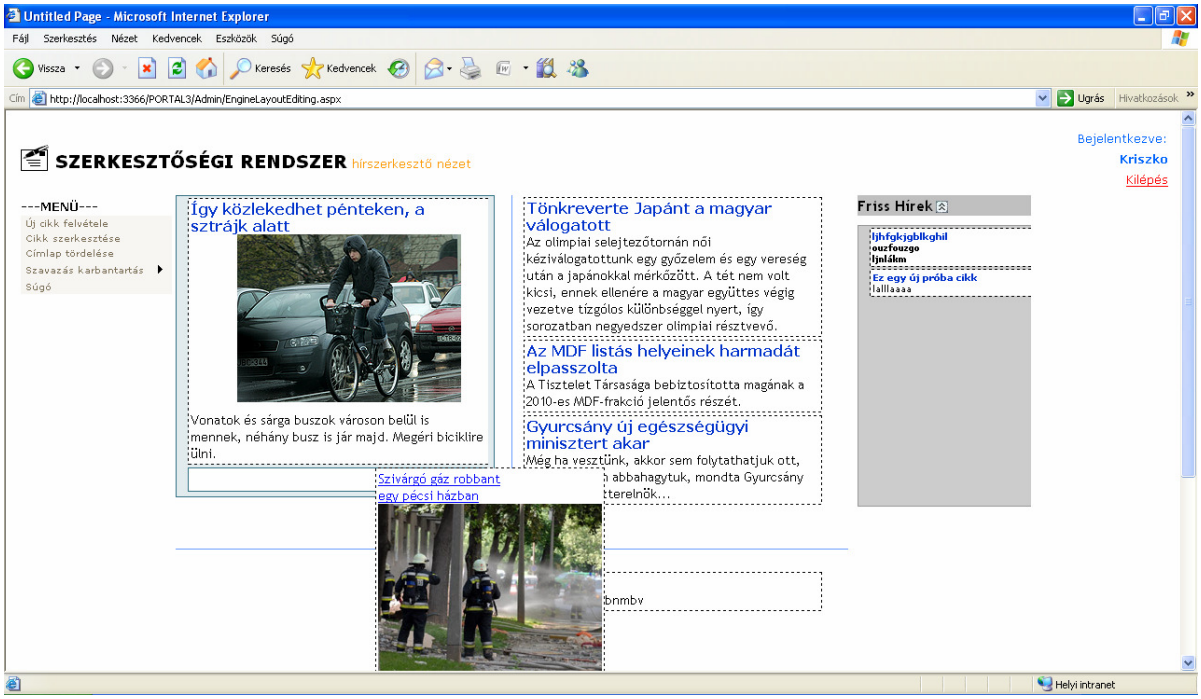
Így az oldalt vizuálisan a böngésző a következőképpen jeleníti meg:



20. ábra A címlaptördelés oldal felépítése

Ennek eredményeképpen a rendszer a portál látogatója által is kapott címlap megjelenést biztosít a szerkesztőségi rendszer felhasználója számára.
A hirCsoport HTML elemek mindegyikében egy<ul>HTML lista van definiálva, mely lista elemeit a<li>HTML címkék nyitó és záró tagjai közötti szöveg képviseli. Ezek a szövegek cikkek lehetnek. Minden szöveg tartalmaz egy címsor szövegű hiperlinket illetve egy hozzá tartozó címlap szöveget.

Az oldal betöltődésekor a *hirCsoport* elnevezésű elemekbe alaphelyzetben automatikusan nem töltődik be semmi. Ahhoz, hogy ezek közül bármelyik is cikket tartalmazzon a felhasználónak csupán annyi a dolga, hogy az weboldal jobb oldalán megjelenő Friss hírek oszlopból (elerhetoHirek azonosítójú HTML elemből) az egér segítségével *Drag and Drop* módszerrel egy kiválasztott cikket áthelyezzen a hirCsopot ‹div› HTML elemek bármelyikébe. Az egér gombjának lenyomásakor a kiválasztott cikk eltűnik arról a helyről ahonnan a felhasználó mozgatni akarja és egy kis lebegő *box*ba megjelenve folyamatosan követi az egér pozícióját mindaddig, míg a felhasználó az egér bal gombját fel nem engedi. Amennyiben az egeret a cikk mozgatása közben egy *hírCsoport* HTML elem fölé mozgatja a felhasználó, úgy az aktuális *hirCsoport* HTML elem –ben egy téglalap alakú doboz villan fel, jelezve, amennyiben a felhasználó befejezi a mozgatást, a mozgatott cikk azon belül melyik pozícióba kerül majd.



21. ábra Egy cikk mozgatása

Ezt a mozgatást a *LayoutEditing.js* javaszkriptet tartalmazó fájl valósítja meg. A javaszkript mindössze annyit csinál, hogy megkeresi és beolvassa a dragAndDropMainContainer azonosítójú HTML elemből az összes létrehozott ‹ul› listát illetve az egyes ‹ul› HTML elemen belüli ‹li› lista elemeket és ezeket egy kétdimenziós tömbbe képezi le a következőképpen:

Tomb[UL1[UL1_LI1,UL1_LI2,…],UL2[UL2_LI1,UL2_LI2…]…]

A mozgatás során a javaszkript a kétdimenziós tömb második dimenziójában lévő elemeket mozgatja egyik helyről a másikra és a szerint rendereli újra a listákat.

Ezzel bármely lista bármely eleme bármely másik lista bármely pozíciójába mozgathatóvá válik. Így a cikkek tetszőlegesen mozgathatok a hirCsoport ‹div› HTML elemek között vagy a hirCsoport ‹div› HTML elemek és az elérhetoHirek ‹div› HTML elem között.

Végül a változtatásokat menteni kell. Ehhez egy HTML gomb áll a felhasználó rendelkezésére, mely egy HTML form tagja. Ezen kívül a HTML form egy rejtett input mezőt tartalmaz:

```
<form name="myForm" method="post">
<input type="hidden" name="test1" value="">
<input type="submit" value="Változások Mentése"
name="saveButton">
</form>
```

A saveButton gombra történő kattintáskor a form postback eseményt idéz elő és az őt tartalmazó EngineLayoutEditing.aspx oldalt hívja meg, elküldve neki a a rejtett input mező értékét (value).

A rejtett input mező értékének beállítását a javaSzkript végzi. Minden mozgatási művelet végeztével frissíti annak értékét. a következő módon:

Definiál egy *saveString* változót, melynek értéke üres sztring lesz.

Sorra veszi a *dragAndDropMainContainer* HTML elem összes <ul> HTML lista elemét és annak összes <li> HTML lista elemét.

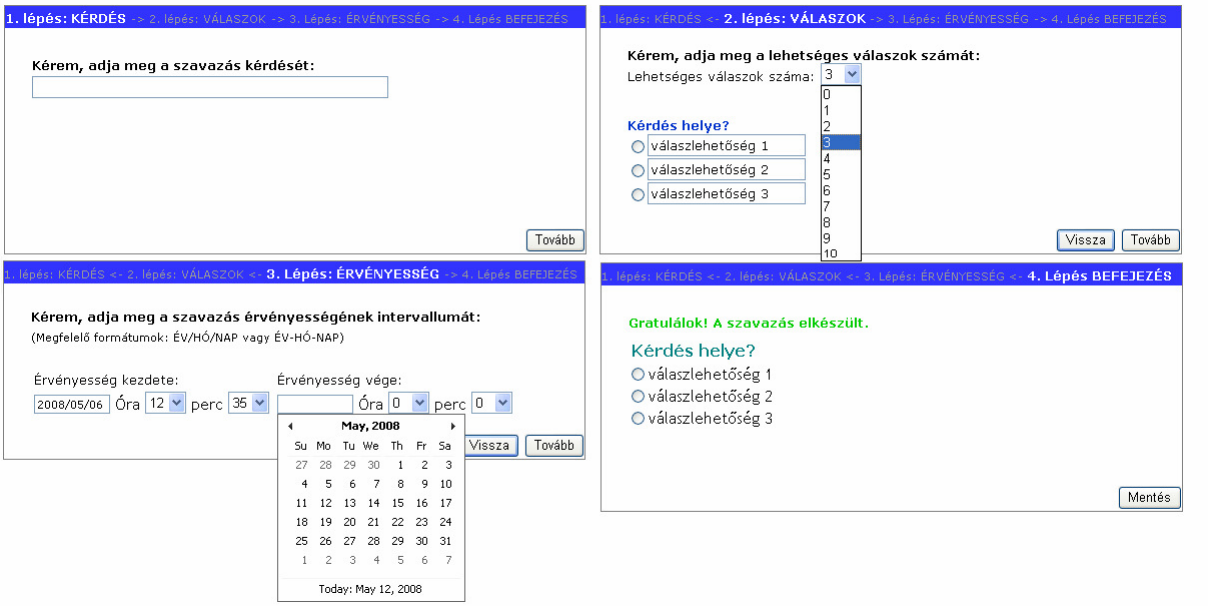
Minden egyes lista elem esetében bővíti a saveString nevű változót az adott listaelem tulajdonságaival. A végéhez fűzi a cikk azonosítóját, majd a végéhez fűzi a <ul> azonosítóját, majd a végéhez fűzi a <ul> -en belüli <li> azonosítóját. Ahol a cikk azonosítója a cikk egyedi azonosítója, a HTML lista azonosítója az előzőekben ismertetett *hirCsoport* sorszáma, a HTML listaelem azonosítója a listaelem a HTML listán belüli sorszáma.

Minden a saveStringben minden azonosítót vesszővel választ el, míg minden azonosító hármast pontosvesszővel.

Így a *PostBack* esemény bekövetkeztekor az oldal a *Page_Load()* eseményben beolvassa a paraméterül kapott a form által küldött rejtett mező értéket, s azt a pontosvesszők, majd a vesszők mentén darabolja a *string* –ekre alkalmazható *Split()* metódus segítségével. Az így kapott rész sztringek egy string típusú tömb elemei lesznek. Ennek a tömbnek véve minden egyes elemét ismét alkalmazzuk a *Split()* metódust, mely a vesszők mentén újabb rész sztringeket ad vissza, s így vissza kapva a cikk azonosítóját, a hírcsoport számát illetve a cikk hírcsoporton belüli pozícióját könnyedén frissíthetjük a *cimlapTabla* SQL táblák értékeit.

3. SZAVAZÁS KARBANTARTÁS

Szavazás létrehozása



22. ábra Szavazás létrehozásának fázisai

A menüsor szavazás karbantartás pontjának szavazás létrehozása almenüpontjában a webportálokon általánosan megjelenő szavazás készíthető. A szavazás lényege, hogy a portál szerkesztői által megadott kérdésre a portál látogatói a megadott válaszlehetőségek közül egyet kiválaszthatnak és „szavazok“ feliratú gomb megnyomásával eggyel növelhetik az általuk kijelölt válaszlehetőségre adott voksok számát.

A kiválasztásra a HTML –ben megszokott radiobutton –ok állnak a látogató rendelkezésére. Mivel a különböző radiobutton –ok ugyanabban a radiobutton csoportban (group) vannak, így azok közül kizárólag egyszerre egy jelölhető ki.

A szavazás létrehozásánál is az „új cikk hozzáadása“ –nál ismertetett MultiView - View megoldást alkalmaztam.

Itt egy MultiView kontrolon belül 3 db View kontrol került elhelyezésre.

Az első View kontrolban egy beviteli mező került elhelyezésre, mely segítségével a szerkesztő, a szavazás alapkérdését adhatja meg. A kérdés megadásakor bármilyen karakter használható, viszont egy regularExpressionValidator kontrol annyi kikötést tesz, hogy mivel kérdésről beszélünk, ezért az utolsó karakter kérdőjel kell, hogy legyen.

Ezen kívül a beviteli mező kitöltésének érdekében a beviteli mezőhöz egy requiedFieldValidator kontrol tartozik.

A 2. View felépítése már egy kicsit összetettebb. Itt a felhasználó a kérdésre adható válaszlehetőségeket viheti be. Alaphelyzetben a View megjelenésekor egy legördülő menüből választhatjuk ki, hogy hány válaszlehetőséget kíván a felhasználó megadni. A DropDownList legördülő menü kontrol onSelectedIndexChanged esemény bekövetkeztekor, a kiválasztott számnak megfelelő beviteli mező jelenik meg. Ezek a beviteli mezők dinamikusan, futási időben jönnek létre. Megjelenítésük pedig úgy történik, hogy a asp.net oldalba statikusan elhelyezésre került egy panel kontrol, melyhez futási időben hozzáadódik a dinamikusan létrehozott TextBox kontrolok mindegyike.
Megvalósítását a következő kód szemlélteti:

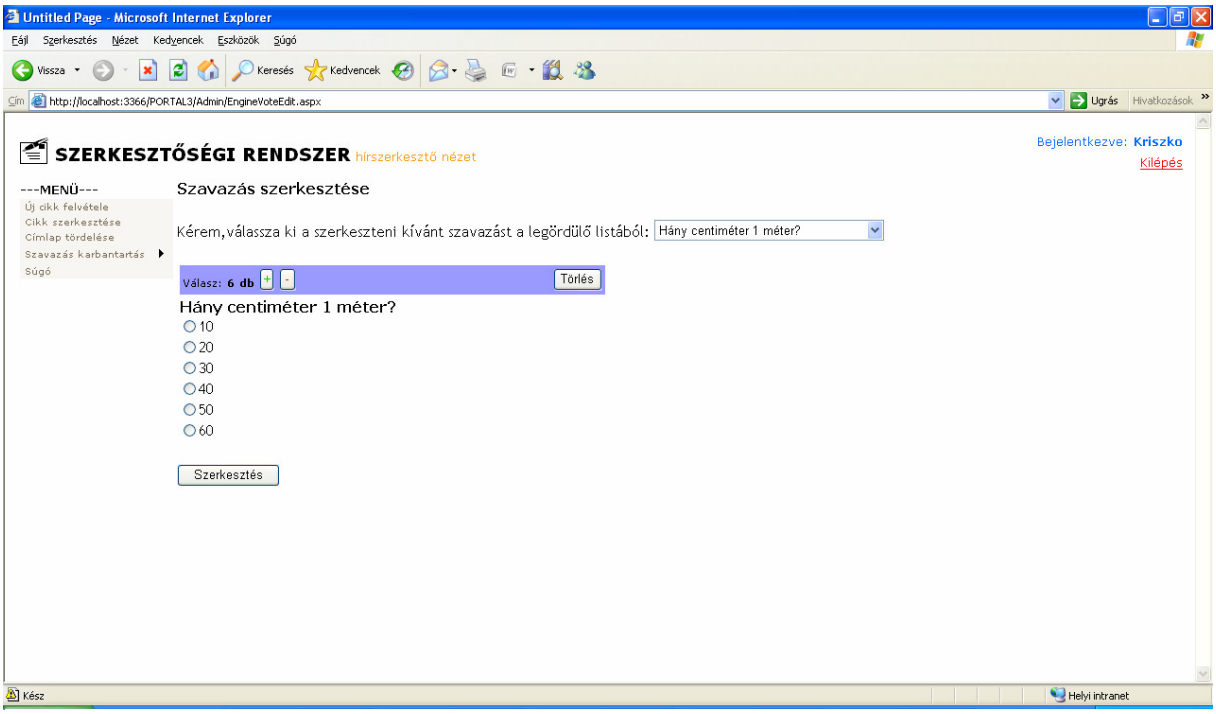
```
protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs
e)
    {
        //A legördülő menüből kiválasztott válaszok számának megfelelően
létrehozunk annyi db radiogombot és textboxot amennyit a legördülő listából
választott a felhasználó.
        DropDownList akt = (DropDownList)(sender);
       //A panel tartalmának ürítése
        pnlValaszok.Controls.Clear();
        if (akt.SelectedIndex > 0)
        {
            for (int i = 0; i < Convert.ToInt32(akt.SelectedItem.Value);
++i)
            {
                    //Radiobuttonok létrehozása az eredmény látszatának
                    szimulálására

                    RadioButton rd = new RadioButton();
                    rd.ID = "rdbtn" + i.ToString();
                    rd.GroupName = "szavazas";
                    //Beviteli mező létrehozása
                    TextBox tb = new TextBox();
                    tb.ID = "tb" + i.ToString();
                    tb.Font.Name = "Verdana";
                    t[i] = tb;
                    //Beviteli mezőhöz egy validátor kapcsolása a kötelező
                    kitöltés érdekében
                    RequiredFieldValidator rv = new
                    RequiredFieldValidator();
                    rv.ControlToValidate = "tb"+i.ToString();
                    rv.ErrorMessage = " <- A mező kitöltése   kötelező!";
                    rv.Font.Name = "Verdana";

                    //A létrehozott radiobutton kontrol, textbox kontrol
                    illetve validátor kontrol elhelyezése egy külön panebe.
                    Panel pl = new Panel();
                    pl.Controls.Add(rd);
```

```
                    pl.Controls.Add(tb);
                    pl.Controls.Add(rv);
                    //A kontrolokat tartalmazó panel elhelyezése az ASP.NET
                    oldalban statikusan elhelyezett pnlValaszok panelben.
                    pnlValaszok.Controls.Add(pl);
                }
            }
        }
```

A 3. View –ban a létrehozás alatt álló szavazáshoz érvényességére vonatkozó dátumokat adhatunk meg. Az érvényesség kezdete dátum megadásával azt az időpontot adhatjuk meg, amelytől a szavazás megjelenhet a portálon, és a szavazásra a látogatók voksokat adhatnak le. Az érvényesség vége dátum megadásával pedig azt az időpontot rögzíthetjük, ameddig a szavazás megjelenhet a portálon, és a szavazók voksolhatnak. Az időpont megadásának formája illetve validációja teljesen megegyezik az „új cikk hozzáadása” illetve „cikk szerkesztése” pontban ismertetett érvényességi dátumoknál ismertetettekkel.

A 4. View –ban a a létrehozott szavazás előnézete válik láthatóvá. A navigációs gombok helyett pedig egy mentés feliratú gomb jeleni meg, mely segítségével az elkészített szavazás hozzáadható az adatbázishoz.

Szavazás szerkesztése



23**. ábra Már meglévő szavazás kiválasztása szerkesztésre**

Ebben a menüpontban a már létrehozott és az adatbázisban elmentett szavazásokat módosíthatja a felhasználó, az adatbázisban szereplő szavazásokhoz tartozó kérdések alapján. A kérdések egy legördülő listában jelennek meg, melyek közül egyet kiválasztva, az ahhoz tartozó kérdés illetve válaszok a legördülő lista alatt elhelyezett panel kontrolban jelennek meg, a látogatói megjelenéshez hasonlóan.

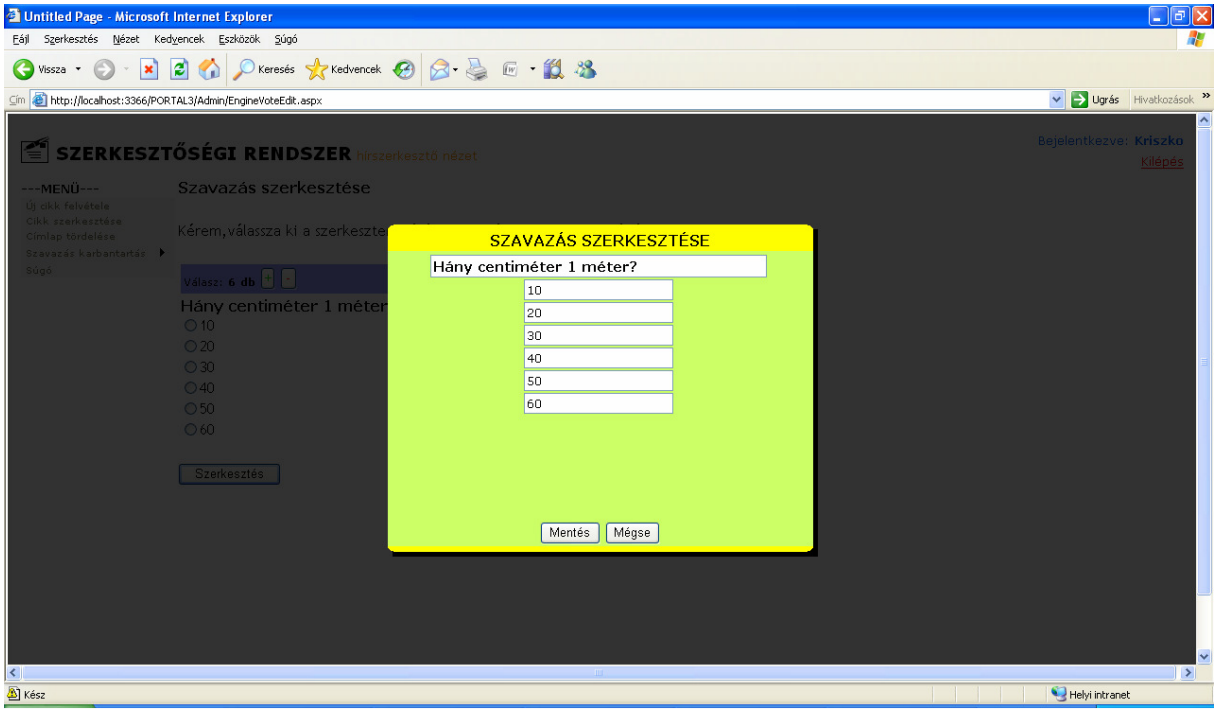
A megjelenítéshez alkalmazott panel kontrol 3 részből épül fel.

Felső részében 2 gomb jelenik meg, melyek segítségével újabb válaszlehetőségeket adhatunk a kiválasztott szavazáshoz, míg a másik gomb segítségével a válaszlehetőségek számát csökkenthetjük az aktuálisan utolsó törlésével.

A panel középső része a már említett szavazási megjelenítés helyeként szolgál.

A panel alsó részében egy „szerkesztés” feliratú gomb található. A gomb megnyomásával egy modalPopupExtender segítségével egy újabb panel válik láthatóvá, melyen a szavazáshoz tartozó kérdés illetve válaszlehetőségek már szerkeszthetők, azaz egy-egy beviteli mezőben jelennek meg. Itt még 2 gomb is szerepel. Az egyik a visszalépésre, míg a másik a változások elmentését, azaz az adatbázis módosítását végzi el.

A 3. pontban leírtak ismeretében szavazások készíthetők illetve szerkeszthetők. Az adatbázisban elhelyezett szavazások bármelyike bármely cikkhez hozzákapcsolhatók az „új cikk hozzáadása” illetve „cikk szerkesztése” menüpontokon belül. A cikkhez kapcsolt szavazások a látogató számára a portál cikk nézetében kerül sor.



24. ábra Kiválasztott szavazás szerkesztése

URL reWriting megvalósítása a Global.asax segítségével

Szakdolgozatom az ASP.NET alkalmazás fájl illetve a felhasználóbarát webcímek megvalósításának bemutatásával zárom.

A Global.asax egy olyan speciális fájl, melyre elhelyezhetünk különféle komponenseket és hozzárendelhetünk tetszőleges kódot is. Az itt létrehozott funkciók és objektumok globálisak lesznek az alkalmazásunkra nézve. Például találunk itt olyan eseménykezelő függvényeket, melyek az alkalmazásunkkal függnek össze, mint például az Application_Start függvény, mely akkor kerül meghívásra, mikor alkalmazásunk elindul, vagy az Application_BeginRequest, mely egy új kérés beérkezésekor fut le. Ezen a ponton tehát a programunkra globális hatással lévő tennivalókat láthatjuk el.

A szerkesztőségi rendszer felhasználóbarát URL címzéseket használ.
Ennek előnyei:

- A honlapunkon található URL –ek sokkal inkább felhasználó- és keresőbarátok.
- Megelőzhetőek a nem kívánt belső linkelések (inline linking / hot linking)
- A külső felhasználók elől elrejthető a honlap belső felépítése

A felhasználók a honlap linkjeit számtalan módon használhatják fel. Például elküldik e-mail –ben, kiírják őket nyilvános fórumokra vagy egyszerűen papírra jegyzik. Ezek nem csak egyes honlapok címeire, hanem azon belül lévő tartalomra is vonatkoznak.
Mivel ezzel a honlap látogatottsága növekedhet, ezért a fejlesztők támogatják is.
Ezért egy jól megtervezett weboldal esetén a felhasználók bármilyen URL –t megadhatnak, azok könnyen átláthatók, egyszerű felépítésűek.
Egy URL –t akkor lehet könnyen használni, ha azok rövidek, de mégis tájékoztatók. A keresőmotorok is sokkal egyszerűbben és gyorsabban indexelik be, és jelenítik meg találataik között az ehhez igazodó honlapokat.

A szakdolgozat során létrehozott hírportál a következő linkelési módszereket alkalmazza:
kategória szerinti szűrés esetén az URL: http://hirportal.hu/kategoria_nev
alkategória szerinti szűrés esetén az URL: http://hirportal.hu/kategoria_nev/alkategoria_nev
adott cikk URL -je: http://hirportal.hu/kategoria_nev/alkategoria_nev/cikk_fURL_azonosito

Egy cikk mindig egy konkrét frendly URL azonosítóval rendelkezik, melyet a cikk létrehozója ad meg a cikk létrehozása során (lásd: *"cikk létrehozása"* vagy *"cikk szerkesztése"*). Minden cikk az őt tartalmazó alkategórián belül egyedi frendly URL azonosítóval rendelkezik.

A látogatói nézetben például az egyes számú cikk kiválasztásakor a következő linket hivatkozza a rendszer:
http://www.portalcim.hu/index.aspx?kategoria=1&alkategoria=1&hirId=1
Mivel ez a link nem elég beszédes illetve nem egészen illeszkedik a fentebb leírt kereső és felhasználóbarát címzéshez, ezért azt a GLOBAL.ASAX fájl segítségével átírjuk, azaz a felhasználónak a böngészés során már csak az átírt cím jelenik meg. Illetve amennyiben az átírt címet hivatkozza, akkor a rendszer felismeri azt és az annak megfelelő adatokat tölti majd be.
Megvalósítás:

```
void Application_BeginRequest(object sender, EventArgs e)
    {
        // a böngészőbe beírt címet a / jelek mentén feldaraboljuk
        string[] split = Request.Url.AbsolutePath.Split(new Char[] { '/'
});
        int cimHossza = split.Length;
        //Vizsgáljuk, hogy a cím végén va e / jel
        if (split[split.Length - 1] == "")
        {
            // Ha van akko az split[] tömb utolsót elemét nem vizsgáljuk,
mert az "" üres string
            //akkor hajtódik végre, ha a cím formátum, például:
            //http://www.XXXX.hu/Belfold/
            //    ||
            //http://www.XXXX.hu/Belfold/Politika/
            //    ||
            //http://www.XXXX.hu/Belfold/Politika/1/
            cimHossza -= 1;
        }
        //Egyébként pedig nem csinálunk semmit, mert a split[] tömb utolsó
eleme a KATEGÓRIÁT || ALKATEGÓRIÁT || HIRINDEXET fogja tartalmazni
        if (cimHossza == 2 || cimHossza == 3 || cimHossza == 4 || cimHossza
== 5 && split[2]!="Error")
        {
        //öt értelmezhető eshetőség lehetséges:
        switch(cimHossza)
        {
            // amennyiben a beírt cím csak http://portal/ formátumban van
            case 2: HttpContext.Current.RewritePath("~/index.aspx");
                    break;
            // amennyiben a beírt cím a http://portal/kategoria/ formátumú
            case 3: if (!split[2].Contains(".aspx") &&
db.frendlyUrlLekredezesek.hivatkozasVizsgalatKat(split[2]))
```

```
                    {

HttpContext.Current.RewritePath("~/index.aspx?kategoria=" + @split[2]);
                    }
                    break;
// amennyiben a beírt cím a http://portal/kategoria/alkategoria/ formátumú
            case 4: if
(db.frendlyUrlLekredezesek.hivatkozasVizsgalatKatAlKat(split[2], split[3]))
                    {

HttpContext.Current.RewritePath("~/index.aspx?kategoria=" + @split[2] +
"&alkategoria=" + @split[3]);
                    }
                    break;
// amennyiben a beírt cím a http://portal/kategoria/alkategoria/cikk_Id/
formátumú
            case 5: if
(db.frendlyUrlLekredezesek.hivatkozasVizsgalatKatAlKatHirId(split[2],
split[3], split[4]))
                    {

HttpContext.Current.RewritePath("~/index.aspx?kategoria=" + @split[2] +
"&alkategoria=" + @split[3] + "&hirId=" + @split[4]);
                    }
                    break;

        }
        }
// amennyiben a beírt cím a következő formátumban van: http://portal/Error/
akkor a weblap használata során hiba lépett fel, tehát a látogatót az
Error.aspx oldalra irányítjuk
        if (cimHossza == 4 && split[2]=="Error")
        {
            HttpContext.Current.RewritePath("~/Error.aspx?errorCode=" +
@split[2]);
        }

    }
```

A cím átírását a HttpContext.Current.RewritePath() metódus végzi, mely az beírt címet a paraméterül megadott címre módosítja, azaz tudatja a szerverrel, hogy a megadott cím a paraméterben megadott címet jelenti, és mindezt teszti anélkül, hogy a böngésző címsorába átírná az eredetileg begépelt címet.

A kódban zöld színnel jelzett programozói megjegyzésből könnyen kivehető, hogy mikor mit reagál a global.asax.

Összefoglalás

A Microsoft a Visual Studio 2008 fejlesztői környezettel egy könnyen áttekinthető, egyszerűen használható és nagy támogatást nyújtó fejlesztői környezetet bocsát a programozó számára, melybe kiegészítő eszközök egyszerűen integrálhatók. (pl: FCKEditor, Ajax Control Toolkit) Mivel az ingyenes tárhelyet biztosító szolgáltatók többsége nem Microsoft Windows alapú web szervert biztosít felhasználói számára, így az ASP.NET technológia egyetlen hátrányának tekinthető, hogy az kizárólag Microsoft web szervereken futtatható.

Egy összetettebb web alkalmazás készítése során elengedhetetlen az adatok tárolására szolgáló adatbázis létrehozása és szerkesztése. A Microsoft Visual Studio 2008 erre a célra is tökéletesen alkalmasnak bizonyult. A fejlesztői környezet segítségével új táblák létrehozása vagy meglévő táblák módosítása könnyedén elvégezhető a megfelelő SQL parancsok ismerete nélkül is, igaz a web alkalmazás használata során azok ismerete már nélkülözhetetlenek.

A számítástechnika, informatika jövője leginkább a webes technológiák irányába fejlődik, így elengedhetetlen, hogy a web programozók számára egy könnyen használható, áttekinthető és teljes részletekbe menő támogatást nyújtó fejlesztői környezetet hozzanak létre a kisebb-nagyobb szoftverfejlesztő cégek. A hardver- és szoftvereszközök gyors fejlődése megnöveli a fogyasztók igényét az időben elérhető és pontos információkra, ezért az ismertetett AJAX technológia szerepe egyre inkább nélkülözhetetlenné válik, mivel a mai weboldalak számtalan nagyméretű multimédiás adatot tartalmaznak. Egy weblap működése során ezen média elemek megjelenítése után a weboldalakon történő böngészéskor a kiszolgálót tehermentesítő megoldások egyre nagyobb szerepet töltenek majd be.

Összességében, a szakdolgozat készítése során világossá vált, hogy az ASP.NET technológia a Visual Studio –val együtt a legalkalmasabb fejlesztői környezetek közé tartozik a weblapok illetve web alkalmazások létrehozására.

Irodalomjegyzék

A Wikipédia on-line lexikon

http://hu.wikipedia.org és http://wikipedia.org

ASP.NET dokumentációk és leírások

http://asp.net

http://forums.asp.net

http://msdn.microsoft.com

AJAX dokumentációk és leírások

http://www.asp.net/ajax

AJAX Control Toolkit dokumentáció és leírások

http://ajax.asp.net/ajaxcontroltoolkit/

http://codeplex.com/AtlasControlToolkit

Köszönetnyilvánítás

Ezúton szeretném megköszönni **Dr. Rutkovszky Edéné** illetve **Keczeli Csaba** témavezetőimnek a szakdolgozat készítése során nyújtott segítését illetve támogatását.
Külön szeretném megköszönni a CTS Informatika Kft. programozójának, **Dvorák Péternek**, hogy kiváló programozói tudásával és tanácsival segítette a szakdolgozat létrejöttét.