Debreceni Egyetem

Informatikai Kar

Web 2.0 alkalmazásfejlesztés CakePHP segítségével

Témavezető:
Dr. Kuki Attila
egyetemi adjunktus

Készítette:
Méhes Zoltán László
programozó matematikus szak

Debrecen
2008

1. Bevezetés

Napjainkra az Internet rohamos fejlődésének köszönhetően egyre több ember vált részesévé a világhálónak és nemcsak egy kiegészítő elfoglaltságként, hanem életünk részeként tekintünk rá. Interneten keresztül intézzük banki ügyeinket, adóbevallásunkat, megszervezzük nyaralásunkat, lefoglaljuk repülőjegyünket, a távoli barátainkkal, szeretteinkkel egyszerűen tudjuk tartani a kapcsolatot, valamint kijelenthetjük, hogy elsőszámú információforrásként is az Internetet vesszük alapul. Ezen megnövekedett információhalmaz kezelése viszont egyre nehezebbé vált. Be kell vonnunk a megnövekedett igényű felhasználókat, hogy segítségükkel szervezzük weboldalainkat, hisz jóval célravezetőbb, ha nem csak egy embernek kell létrehoznia a tartalmat. Az elmúlt egy-két évben gyökeres változások mentek végbe a fejlődésben, melyek óriási lehetőségeket kínálnak. Ez, a szakdolgozatom témájául szolgáló irányvonal a Web 2.0, mely egyre nagyobb teret hódít napjainkban és számos szolgáltatást veszünk igénybe. Dolgozatomban igyekszek egy átfogó képet mutatni erről a tendenciáról, az újdonságokról, a használatos szolgáltatásokról illetve lehetőségekről.

Gyakorló PHP fejlesztőként gyakran estem abba a hibába, hogy az egyre bonyolultabb, de ugyanakkor nagyon hasonló feladatokat olykor a nulláról kellett elkészítenem, valamint az alkalmazás üzleti logikája és a kliens oldali kód összekeveredése jóval megnövelte a fejlesztési illetve karbantartási időt. Ezen probléma megoldásaként találtam az MVC tervezési mintára és ezáltal a CakePHP keretrendszerre.

Szakdolgozatom célja, hogy bemutassam ezt a programozási mintát, valamint kiemeljem a CakePHP néhány erősségét egy Web 2.0 alkalmazás fejlesztésén keresztül.

2. A web 2.0 mint jelenség

Kezdetben a web célja egyszerű, statikus weblapok megjelenítése volt. Ezen korszak kezdete 1980-ig nyúlik vissza; 1990-re tehető a Tim Berners-Lee által készített első böngésző és weblap. A kiszolgáló webszerverek feladata nem volt más, mint elérhetővé tenni egy HTML fájlokat tartalmazó könyvtárszerkezetet, melyek közt linkek segítségével navigálhattunk. Nagyon ritka frissítési idő volt jellemző, gyakran az elkészítési időtől számított néhány évig hozzá sem nyúltak. A web ezen változatát szokás **Web 1.0** néven emlegetni.

Rövidesen az üzleti világ is fantáziát látott a kezdeményezésbe és ezzel elkezdett fejlődni a webes grafika – jellemzője a táblázat alapú struktúra, mely sajnos napjainkban is megtalálható. A statikus oldalakon kívül a dinamikus oldalak is napvilágot láttak, így különböző interaktív szolgáltatások jelentek meg (fórumok, webshopok, portálok). A szerveren lévő program hatására lehetőség volt akár minden oldalletöltésre más eredményt generálni, vagy adatbázisból vett információkat HTML oldalakon megjeleníteni. Ezeket az oldalakat szokás **Web 1.5**-ként emlegetni. A mai weboldalak nagyrészt ilyen tulajdonságokkal bírnak.

Az utóbbi években sok változás történt, melyeket már nem csak a szakmabeliek érezhettek, hanem a mindennapi internet felhasználók is. A változások eredményeként megjelent a böngészési-élmény következő változata, a **Web 2.0**. Elterjedtek a tartalomkezelő rendszerek, melyek közül legelterjedtebbek a blogok. Gyakorlatilag bárki számára elérhetővé vált, hogy egyszerűen közzétegye gondolatait. A blogok a cégek körében is terjedni kezdtek, s manapság már nem csak a személyes, hanem a vállalati naplók is terjedőben vannak, nem beszélve az életünkbe hirtelen berobbant politikai blogokról. Azonban ez csak egy a sok újdonság közül, megjelent többek közt számos közösségi szolgáltatás is.

Fontos, hogy a változás nem jelentős technikai újdonságnak tudható be, hanem annál inkább a fejlesztők megváltozott gondolkodásmódjának. A legnagyobb újdonság a webet használók bevonása. Ezáltal teljesen megváltozott a webhez való hozzáállás, új kommunikációs lehetőségek nyíltak és kezdenek egyre többen aktív részesévé válni ennek a rendszernek.

A web a változások hatására egyre inkább középpontba kerül. Már ma is vannak olyan területek, ahol úgy végezhetünk munkát, hogy egy böngészőn kívül más alkalmazás nem is fut gépünkön. Gondoljunk egy online újságíró szerepébe: egy webes programon keresztül megkapja az aktuális híreket (például Google Reader-en keresztül), webes levelezőben tartja a kapcsolatot a külvilággal (pl. Gmail), cseveg az ismerőseivel (pl. Gtalk), feltölt fényképeket az online galériájába (pl. Flickr), majd cikkeket publikál egy blogba (pl. WordPress-en keresztül).
De ugyanakkor a webfejlesztők körében elterjedtek a webes adatbázis-kezelő programok (pl. PhpMyAdmin), a szerver adminisztrátorok pedig webes szerverkarbantartó felületeken keresztül végezhetik munkájukat (pl. Cpanel).

A felsorolt lehetőségeket hívjuk összefoglalóan webalkalmazásoknak. Azaz a hagyományos alkalmazások a webre költöznek, ezáltal a világon bárhonnan, tetszőleges platformról elérhetőek. Sőt, léteznek olyan kezdeményezések is, melyek egy operációs rendszerhez hasonló felületet jelenítenek meg a böngészőben a rá jellemző tulajdonságokkal – átméretezhető ablakok, asztal, stb. Az egyik ilyen legnépszerűbb alkalmazás a Box.net (http://box.net), a legkecsegtetőbb magyar próbálkozás pedig a Rhitmo (http://rhitmo.hu).

1. ábra: A Web 2.0-val kapcsolatos fontosabb fogalmak címkefelhőben.

2.1 Jelentős Web 2.0 alkalmazások bemutatása

Mivel a Web 2.0 kifejezés csupán olyan második generációs internetes szolgáltatások gyűjtőneve, amelyek elsősorban a közösségre épülnek (azaz a felhasználók közösen készítik a tartalmat vagy megosztják egymás információit), azonban technológiai előírást nem tartalmaz, ezért elég egyszerűen rá lehet mondani egy weboldalra, hogy az webkettes. Az alábbiakban néhány tipikusan második generációs weboldal kerül bemutatásra.

Wikipédia

A szabad lexikon, mely a közösség által szerkesztett, nyílt tartalmú, nonprofit, online enciklopédia. A tartalmat a felhasználók alakítják ki, de nem ők birtokolják, ezáltal bárki szerkesztheti bármely bejegyzést. Jelenleg 260 féle nyelvi változatban érhető el és 11 millió szócikket tartalmaz, melyet 13 millió felhasználó alakított ki. Technológiai szempontból, egy kifinomult tartalomkezelő rendszer, a Mediawiki biztosítja a gördülékeny működést. Meglepő módon mellőzi a Web 2.0 alkalmazásokra jellemző Ajax technológiát. A Wikipédia az egyik legjellemzőbb megjelenése a közösségi erőnek.

Google Dokumentumok, Naptár, Gmail

A Google webes irodai alkalmazásai, angol nevén Google Docs, Calendar ill. Gmail. A Dokumentumok magába foglal egy táblázatkezelőt, szövegszerkesztőt, valamint prezentációkészítőt is. Lehetőség van, hogy egyszerre többen is szerkesszék valós időben a dokumentumokat. Másik hatalmas előnyként a legelterjedtebb fájlformátumok támogatottságát lehet felhozni.

A Calendar szolgáltatás rendelkezik a naptároknál megszokott funkciókkal, mint például a számos nézet közüli választás, az import-export stb. Egyik legfrissebb újítása, hogy az egyes eseményekről SMS értesítést is igényelhetünk, természetesen mindezeket ingyen.

A Google ingyenes levelezője webes, POP3 és IMAP szolgáltatásokkal rendelkezik. Az egyik legdinamikusabban fejlődő es legelterjedtebb e-mailszolgáltatás szerte a világon.

Ezen irodai csomagokra mind jellemző a Google által jellemző letisztult design, valamint az Ajax és XML-en alapuló működés, ezzel a Web 2.0 másik típuspéldájának tekinthetjük.

YouTube

Az internetes sávszélesség növekedésével megkezdődött a webes videoklipek térhódítása, melyek közül a legismertebb szolgáltató a YouTube. Ez az alkalmazás is a közösségre épít: a felhasználók videókat néznek, feltöltenek és osztanak meg. A képanyagok megjelenítését *Adobe Flash* végzi, valamint a felhasználói élmény fokozásáért egyre több AJAX is helyet kapott.

Flickr

Fényképmegosztó szolgáltatás és közösségi tér. A fényképek YouTube-ja. A képeket elláthatjuk kulcsszavakkal (keywords) és címkékkel (tag) is. A flickr elsők közt használta a címkézést, melyért az *Organizr* webalkalmazás a felelős. Segítségével a képeket több csoportba lehet szervezni, valamint földrajzilag is el lehet rendezni a *Yahoo! Maps* térképein.

Delicious

Könyvjelzőink számára biztosít webes tárolást, akár saját részre, akár mások számára is megoszthatjuk. A címkézés itt is segítségünkre van. Egyes böngészőkhöz kiegészítések is készültek, melynek köszönhetően jóval praktikusabb a böngésző beépített *Könyvjelzők* funkciójánál.

Last.fm

A zenehallgatás és internetes rádióállomás újgenerációs alkalmazása. A rendszer egy részletes statisztikát vezet a felhasználók zenei ízléséről (egy előre letöltött plugin segítségével). Ezáltal egy hatalmas nyilvántartás kerül elénk. A lejátszott dalokat a rendszer elnaplózza, és az ebben szereplő adatokból kalkulálja ki a toplistákat, valamint az ajánlott zenéket (*My Radio*). A már kötelezőnek mondható közösségi szolgáltatások itt is jelen vannak.

Ebay

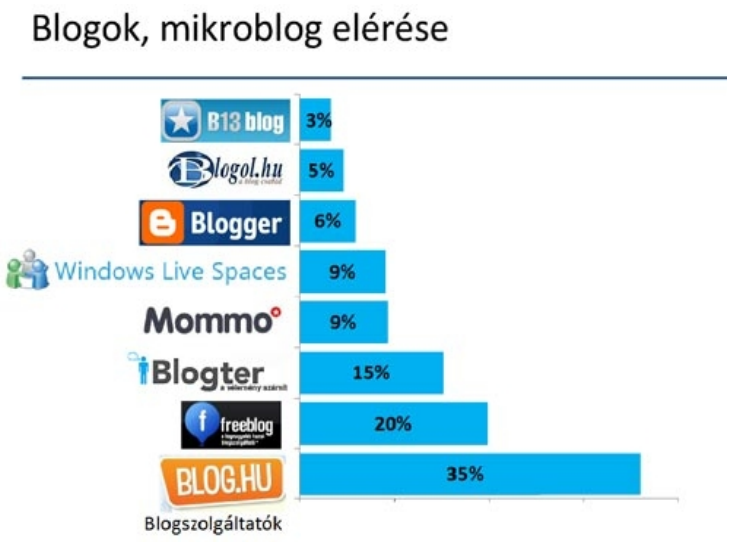
Internetes aukciós weboldal, ahol a felhasználók felkínálhatják eladásra termékük vagy szolgáltatásukat. Közösségi szintre, az egyes felhasználók értékelésének lehetősége emelte. Így könnyen megtudhatjuk, hogy a kiszemelt termékünket egy megbízható felhasználó árulja e.

iWiW

A legismertebb magyar Web 2.0 szolgáltatás, mely az angol „*international who is who*" rövidítése. 2006 óta a leglátogatottabb magyar weboldal. Sikerét annak köszönheti, hogy elsőként nyújtott olyan közösségépítő szolgáltatást, amelyet más addig nem, holott mára semmivel sem nyújt többet, mint bármely konkurenciája. Sőt, a növekedéssel az iWiW műszaki háttere nem tudott lépést tartani, így alapvető hibák keserítették meg a felhasználókat. Viszont napjainkban számos újítás van kilátásba: ilyen többek közt a nemzetközi *Facebook*-hoz hasonlóan, az API nyitása. Ezzel lehetőség lenne az iWiW adatbázisát alapul véve mini alkalmazások fejlesztésére. (A Facebook amerikai ismeretségi hálózat, melyet Mark Zuckerberg alapított 2004-ben, ki mára a sikeres üzletének köszönhetően a világ legfiatalabb milliárdosa.)

Blogok és mikroblogok

A blogok napjainkra több mint csak internetes naplókat jelentenek. Segítségükkel bárki könnyedén publikálhatja írásait, gondolatait. Ezáltal aktív részesévé válhatnak az átlagos internetes felhasználók. A blogok a cégek körében is közkedveltté váltak, nem beszélve az egyre inkább elterjedő politikai blogokról. Technikai újításként a hírcsatornák (feed) megjelenése a legkiemelkedőbb. Melyek segítségével hírolvasó programokon keresztül teszik lehetővé a felhasználóknak, hogy egy helyen kövessék kedvenc blogjaikat, híroldalaikat.



2. ábra: Blogok és mikroblogok elérésének megoszlása Magyarországon (2008.augusztus)

2.2 Technológiai változások

A Web 2.0 sikerét ugyan nem az óriási technikai újdonságok jelentették, azonban mégis számos olyan újítás alakult ki a második generációs weboldalak révén melyek nagyban befolyásolják a web jövőjét.

2.2.1 Címkék

A hagyományos kategorizálásnál egy dolgot csupán egy kategóriába lehetett elhelyezni. Tipikus példa erre a könyvtár: egy adott könyvet nem lehet egyszerre az informatika és matematika témakörbe helyezni, hisz mindkét csoporthoz másik könyvespolc tartozik. Ekkor a hozzáértő könyvtároson múlik, hogy mit választ. Ugyanez volt igaz a weboldalak terén is. Ha például egy *Apple Macbook 13.3"* laptopot szeretnénk internetes áruházunkba feltölteni, csak egy kategóriába tudjuk helyezni, például márka alapján az Apple csoportba.
A címkék (tag) segítségével viszont egy adott dologhoz több tulajdonságot, címkét rendelhetünk (melyek akár ellentmondóak is lehetnek), így az előbbi példánál maradva a termék megkaphatja az *apple, macbook, laptop, fehér, OSX stb.* címkéket melyek alapján jóval könnyebb rátalálni. Egyes szolgáltatásoknál lehetőségük van a felhasználóknak is a címkék szerkesztéséhez. Valamint keresésnél logikai kapcsolatokat teremthetünk a címkék közt. Hisz ha a *macbook* címkére keresünk, egyéb termékeket is találhatunk, amelyek számunkra érdektelenek.
A címkéket változó méretű szövegként sorolják fel, melyet címke felhőnek nevezünk (tag-cloud). Minél népszerűbb egy címke annál nagyobb a betűmérete.
A címkézésnek köszönhetően nagyban megkönnyíti a rendszerezést és a keresést is.

2.2.2 API

Web-re kidolgozott alkalmazási programfelületek (Application Programming Interface – API) fejlődésének eredményeként létrejött megoldásokkal az információ-megosztás és elérés új lehetőségei nyíltak meg. Az ilyen API-szolgáltatások alkotják napjaink és a közeljövő Web-ének alapelemét. Újszerűsége abban rejlik, hogy nyíltak és egyszerűen használhatóak. Így bárhonnan igénybe lehet venni az olyan szolgáltatásokat, mint például a Google keresője vagy akár az RSS hírcsatornákat is.

2.2.3 Hírcsatornák

A hírcsatornák valamilyen szabvány szerinti XML dokumentumok, amelyek egy weboldalhoz tartozó információkat tartalmazzák. Ezeket a dokumentumokat, valamilyen alkalmazás segítségével formázva jeleníthetjük meg – ezek a hírolvasók. Így nem kell minden weboldalt látogatnunk, hogy történt e frissítés, elég a hírolvasónkat figyelni és rögtön elolvashatjuk a legfrissebb híreket (egyes esetekben csak annak rövid változatát). Azonban a hírolvasókon kívül a csatornát felhasználhatja egy másik alkalmazás is. Ilyen alkalmazás lehet egy torrent kliens is akár, vagy éppen weboldalunkon közvetlenül megjeleníthetjük egy másik oldal híreit is. Ezeket az XML dokumentumokat a szerveren futó program automatikusan generálja le. A legelterjedtebb formátumok az RSS (*Really Simple Syndication*) és az Atom. Egy RSS hírcsatorna forrása:

```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0">
  <channel>
    <title>Példa hír címe</title>
    <link>http://pelda.hu/</link>
    <description>Ez egy példa RSS feed.</description>
    <item>
       <title>Aktualitások október hónapra</title>
       <link>http://pelda.hu/2008/10</link>
       <description>A hónap információinak összefoglalása.</description>
    </item>
    <item>
       <title>Aktualitások november hónapra </title>
       <link>http://pelda.hu/2008/11</link>
    </item>
  </channel>
</rss>
```

A *channel* elem tartalmazza az alapinformációkat. A *title* a csatorna nevét, a link az adott weboldal címét, míg a *description* egy rövid leírást. Emellett lehetőségünk van opcionális elemekkel ellátni, többek közt a frissítés gyakoriságára vonatkozóan, megadhatjuk milyen nyelven íródott a feed, vagy éppen a hírolvasó számára *category* elemmel is elláthatjuk a rendszerezés segítése érdekében.

A *channel* elemen belül tetszőleges *item* helyezhető el, melyek a konkrét hírre vonatkozó információkat tartalmazzák. Itt is számos elem megadása közül választhatunk, viszont meg kell adni a *title* (cím), *link* (hivatkozás), *description* (leírás) elemeket.

2.2.4 Felhasználóbarát hivatkozások

Gyakran előfordul, hogy egy honlap böngészése közben a böngészőnk címsorába egy megjegyezhetetlen, kusza címet kapunk. Például vegyük az alábbi összetett, bonyolult hivatkozást: *http://pelda.hu/modules.php?Name=hirek&Category=web2&mode=normal*. Melyet egy egyszerű technikával az alábbi formává tudunk alakítani: *http://pelda.hu/hirek/web2*. Mely egy jóval felhasználóbarátabb, megjegyezhetőbb cím. Mindezek mellett a mostanra elterjedt SEO (Search Engine Optimization, azaz keresőoptimalizálás) technikái közt is jelentős erővel bír. A megfelelő módon megírt PHP alkalmazásunk mellé egy az Apache URL Rewrite (IIS szerveren az ISAPI Rewrite) lehetővé teszi, hogy az Apache kérések kiszolgálásába ágyazott mintaillesztő kifejezéseket írjunk, melyek megváltoztathatják a szerver által kezelt webcímet. Ehhez egy .htaccess állományt kell létrehoznunk webszerverünkön. Íme, egy egyszerű példa a tartalmára:

```
<IfModule mod_rewrite.c>
  RewriteEngine on
  RewriteBase /cegunkwebhelye

  RewriteCond %{REQUEST_FILENAME} !-f
  RewriteCond %{REQUEST_FILENAME} !-d
  RewriteRule ^(.*)$ index.php?q=$1 [QSA]
</IfModule>
```

A *RewriteBase* megadja, hogy milyen alapkönyvtárat kell figyelembe vennünk.
A *RewriteCond* olyan feltételeket ad meg, amelyek teljesülésekor a *RewriteRule* végrehajtódik. Jelen példánkba a *RewriteRule* akkor hajtódik végre, ha a kért webcím nem létező állomány és nem létező könyvtár. A *RewriteRule-on* belül lévő mintaillesztés segítségével azt adjuk meg, hogy az *index.php q* paraméterébe kell áthelyeznünk a kért webcímet. A *[QSA]* opcionális kapcsolóval pedig megadjuk, hogy egy GET lekérdezési paraméter listát a magadott lista végéhez fűzze hozzá.

2.2.4 AJAX

Az AJAX webfejlesztési technikáról a következő fejezetben esik részletesen szó.

3. AJAX

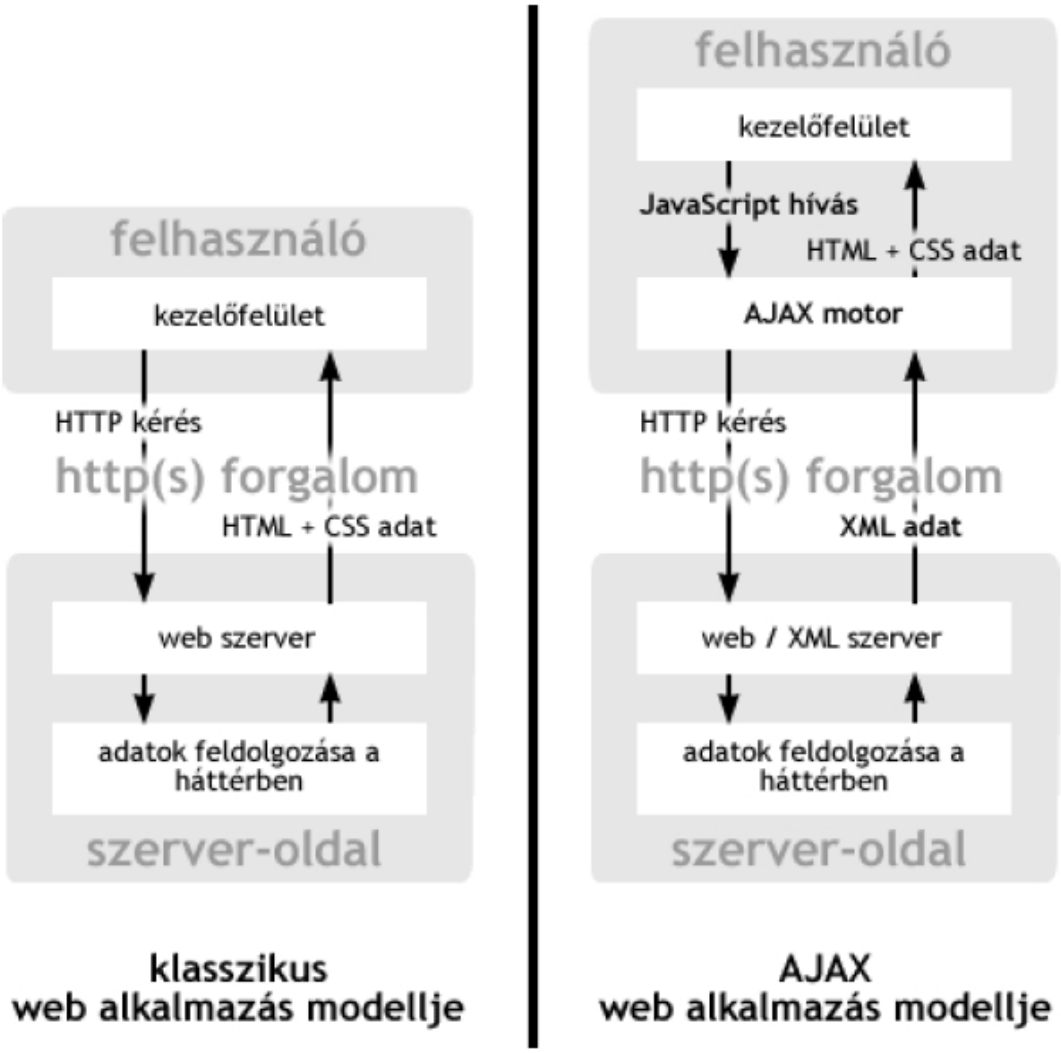
Az első generációs dinamikus weboldalak gyakorlatilag a felhasználó által bevitt adatok és a szerver kommunikációjaként működnek. A szerver feldolgozza a megadott kérést, majd a választ elküldi a böngészőnek és egy új oldal formájában a felhasználó visszakapja a feldolgozott adatokat, majd a szerver bontja a kapcsolatot. Ez a folyamat akár több másodperc is lehet, mialatt a felhasználó várakozásra kényszerül. Így ezzel a technikával képtelenség azt a felhasználói élményt elérni, amit egy asztali alkalmazás nyújt.
Viszont némi szemléletváltással könnyen megoldást találhatunk: vegyük a webet, mint információs közeg, majd az eredeti technológiára építve alkalmazzunk új megoldásokat, melyek segítségével jelentősen javítható a felhasználói élmény. Ilyen szemléletbeli változást képvisel az AJAX technika is.

Az AJAX az *Asynchronous JavaScript and XML*, azaz *Aszinkron JavaScript és XML* rövidítését jelenti. Olyan interaktív webalkalmazások létrehozására szolgáló webfejlesztési technika, mely az alábbi technikák kombinációjából áll:

- XHTML (vagy HTML) és CSS: tartalom leírására és formázása
- XMLHttpRequest: objektum az adatok aszinkron kezelésére
- JavaScript: az objektum létrehozásához
- DOM (Document Object Model): dinamikus megjelenítés

Tehát az AJAX nem értelmezhető egy önálló technológiaként, hanem a meglévő technológiák és szabványok egy halmazának újszerű felhasználása. Az AJAX alkalmazás az oldal letöltődése után további kéréseket küld a szerver felé a HTTP protokollon keresztül. A kérések küldése és fogadása a JavaScript felhasználásával aszinkron módon történik.
Gyakran előfordul, hogy csupán kis adatmennyiség cseréjére van szükség a kliens és a szerver között. Az AJAX-nak köszönhetően nem kell az egész oldal tartalmát újratölteni, csak a változásokat, így a válaszidő lényegesen kisebb lesz, ezzel közelítve az offline alkalmazások

felhasználói élményéhez. Az adatcsere aszinkron módon történik, általában XML (*Extensible Markup Language*) használatával. A kisebb adatforgalomnak köszönhetően csökken a szerver terhelése. A megnövelt interaktivitás mellett számos grafikai újítás is alkalmazható ezen új technológiának köszönhetően.



3. ábra: Hagyományos és egy AJAX alkalmazás összehasonlítása

Vegyük sorra az AJAX által használt technológiákat, hogy mi mire való:

3.1 (X)HTML és CSS

Az XHTML (*Extensible HyperText Markup Language, azaz kiterjesztett HTML*) a HTML-től annyiban különbözik, hogy *XML* alapokkal rendelkezik, azaz jóval szigorúbb szabályok vonatkoznak a szerkesztésére. Ennek köszönhetően jól formázott, valamint automatikusan feldolgozható. Lényeges, hogy elkülönítsük az adatot a megjelenítésért felelős stílusoktól.
A megjelenítésért a CSS (*Cascading Style Sheets*, azaz stílusleíró nyelv) a felelős. Tehát, ha új kinézetet szeretnénk az XHTML oldalunknak adni elegendő a CSS fájlunkat módosítani.
A stílusok egymásba ágyazhatóak: egy elem megjelenítését nem egy stílus határozza meg, hanem azok egymásba ágyazása (melyek tulajdonságai akár felül is bírálhatják egymást). Azonban szabványos kód írása mellett is előfordulhat, hogy a különböző böngészők másként jelenítik meg weboldalunkat (napjaink böngészői közül elsősorban az Internet Explorer 6, mely legkevésbé támogatja a szabványokat).

3.2 XMLHttpRequest

Az XHR (*XMLHttpRequest*) objektum az AJAX motor központi eleme. Ez az objektum teszi lehetővé, hogy az oldal háttérkérelemként kapjon adatokat a kiszolgálótól (a *GET* eljárás segítségével), illetve küldjön adatokat a kiszolgálónak (a *POST* eljárás segítségével), ami azt jelenti, hogy a folyamat során nem frissíti a böngészőt. Az XHR használatakor nem kell a kiszolgálóra várni, hogy minden kérelemre új oldallal válaszoljon, és lehetővé válik, hogy a felhasználók továbbra is kapcsolatban maradjanak az oldallal úgy, hogy a kérelmek küldése a háttérben folyik. Az XHR igényszolgáló (on-deamond) jellege az olyan webalkalmazásoknál rendkívül hasznos, ahol a felhasználók feladatokat próbálnak megoldani – a szabványos HTTP-kérelem inkább a bemutató típusú webhelyeknél alkalmazható.

3.3 JavaScript

A JavaScript egy osztályok nélküli, gyengén típusos, objektum-orientált script nyelv. Elsődleges célja, hogy segítségével manipuláljuk a DOM-ot, valamint AJAX esetén létrehozza az XHR objektumot, valamint feldolgozza ennek eredményét. Sajnos itt is jelen van a stíluslapoknál említett böngészők közti különbség, így kénytelenek vagyunk minden egyes böngészőben külön tesztelni.

3.4 DOM

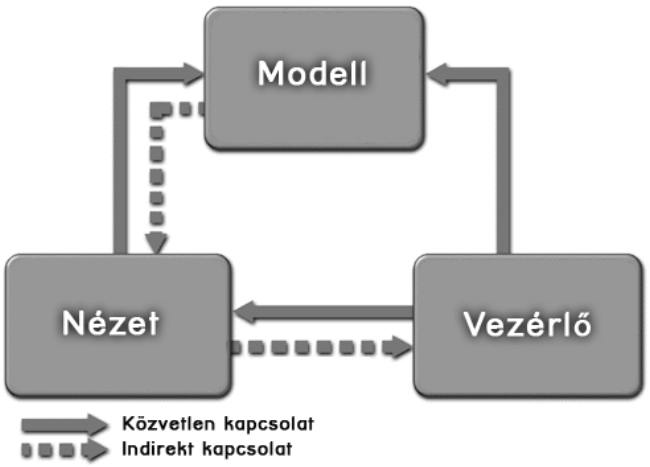
A DOM, az angol *Document Object Model*, azaz *Dokumentum Objektum Modell* szavak rövidítése, mely meghatározza az XML dokumentumok elérésének és módosításának szabványos módját. A DOM hozzáférést nyújt az XML, illetve az XHTML dokumentumok szerkezetét alkotó elemekhez, ezáltal teljes elérést nyújt a JavaScript számára. A hozzáférést a JavaScript DOM-kezeléssel foglalkozó belső objektumhalmaza teszi lehetővé. Elengedhetetlen az *XMLHttpRequest* (XHR) kérelmek létrehozásakor a kiszolgáló oldalról érkező válaszok feldolgozásához.

Fontos megjegyezni, hogy az AJAX terén is kialakultak kisebb-nagyobb keretrendszerek, meleyek használatával lényegesen rövidülhet a fejlesztési idő, ugyanis a legtöbb esetben böngészőfüggetlen futtatást biztosítanak. Ilyen keretrendszerek többek közt:

- Prototype: Sok keretrendszer alapjául szolgáló JavaScript gyűjtemény, mely használható önmagában is (honlap: http://prototypejs.org)
- Script.aculo.us: Prototype-ra épülő keretrendszer melyet többek közt a NASA, Apple, Digg oldalain is használnak. Részletes dokumentációval rendelkezik, ezáltal könnyen elsajátítható (honlap: http://script.aculo.us)
- jQuery: Egyre nagyobb népszerűségre tesz szert a gyors és tömör JavaScript könyvtár, mely az AJAX hívások megkönnyítése mellett hatásos esemény- és animáció kezeléssel rendelkezik. Használják: Google, NBC, Drupal stb. (honlap: http://jquery.com)

4. MVC tervezési minta

Az MVC (*model – view – controller*, azaz modell – nézet – vezérlő) egy olyan tervezési mintát ír le, mely az adatok elérését és az üzleti logikát (modell) elkülöníti a megjelenítéstől és a felhasználói interakciótól egy közbülső összetevő, a vezérlő bevezetésével.
Ezt a mintát 1979-ben Trygve Reenskaug írta le, mikor grafikus felületű SmallTalk alkalmazásokon dolgozott.



4. ábra: A Modell – Nézet – Vezérlő kapcsolatai.

A **modell** reprezentálja az üzleti logikát, ez a rész felelős az adatok kezeléséért. Leggyakrabban relációs adatbázisok, tárolt eljárások és függvények segítségével valósítják meg, ezáltal jelentést ad az adatoknak. Például kiszámolja egy számla áfáját, végösszegét, vagy éppen meghatározza, hogy a felhasználónak a mai napon van e a születésnapja.

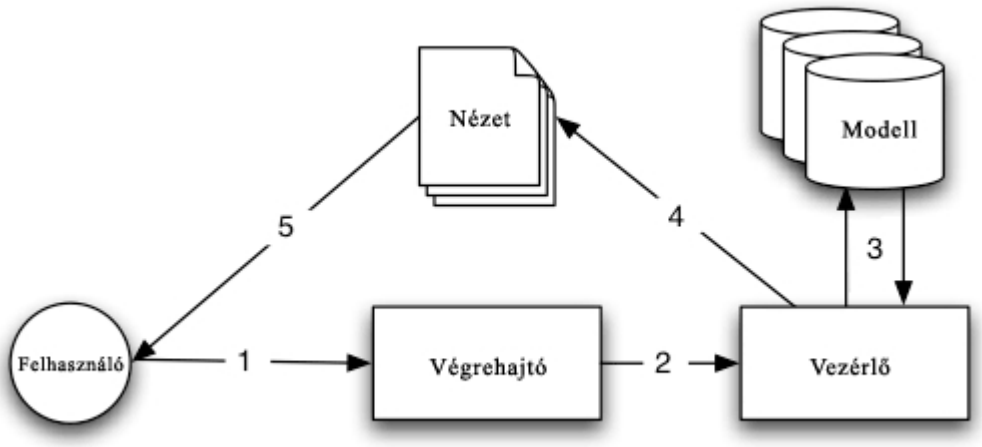
A **nézet** feladata a feldolgozott adatok megjelenítése. Ez általában egy XHTML oldal, melyen a szokásos elemek találhatóak: táblázatok, formok, szövegek stb. Összetett alkalmazásoknál a nézetek használatához szokás különböző sablonokat (template) használni, az egyik ilyen legelterjedtebb rendszer a *Smarty*.

A **vezérlő** dolgozza fel a felhasználó által megadott utasításokat, módosítja a modellt és a választ elküldi a nézetnek. Például egy hivatkozásra kattintva a vezérlő határozza meg, hogy ennek mi legyen a hatása (új adat hozzáadása, törlése, módosítása…).

Ez az elkülönített minta lehetővé teszi, hogy a programozók és a designerek egyszerre dolgozzanak, valamint egy módosítás során nem fogja a többi részt befolyásolni.

Tekintsünk meg egy konkrét MVC lekérést CakePHP alatt. Vegyünk egy alapvető szituációt: a felhasználó rákattint egy hivatkozásra, mely hatására az alábbiak történnek:

1. A hivatkozásra kattintva (ami a *http://pelda.hu/hirek/4* oldalra mutat) a böngésző elküld egy kérést a webszerver felé.
2. A végrehajtó ellenőrzi az URL-t (/hirek/4) és átadja a vezérlést a *hirek* controllernek. A vezérlő végrehajtja a feladatát – például ellenőrzi, hogy be van e jelentkezve a felhasználó.
3. A controller a modellhez fordul, hogy elérje az alkalmazás adatait. A modell többnyire egy adatbázis táblát képvisel.
4. A vezérlő átadja az adatokat a nézetnek. A view az adatokat átalakítja, hogy a felhasználó számára megjeleníthető legyen. Ez többnyire egy XHTML formátum.
5. Amint a view felépítette az oldalt a controller által kapott adatokból a tartalmat elküldi a felhasználó böngészőjének.



5. ábra: Egy alap MVC kérés CakePHP-ban.

4.1 PHP keretrendszerek

Az MVC sikerének köszönhetően számos keretrendszer alakult ki PHP nyelven is, melyek támogatják a modell-nézet-vezérlő központú fejlesztést. Ezek közül van, amelyik a Ruby on Rails, vagy éppen a Microsoft .NET környezetet vették alapul. Rengeteg keretrendszer alakult ki, ezek közül néhány:

Symfony

A PHP 5-ös verziójára alapuló keretrendszer 2005-ben jelent meg. Adatbázis-függetlenség, AJAX-implementáció, egyszerű telepítés, szép URL-ek, adminisztrációs felület kódgenerálása és közös feladatok automatizálása jellemzi.
Többek közt Symfony segítségével készült a *Yahoo Bookmarks és Answers* szolgáltatás.
(honlap: http://www.symfony-project.org)

CodeIgniter

Első verziója 2006 februárjában jelent meg. A Ruby on Rails környezetet vették alapul a fejlesztésnél, ennek köszönhetően számos közös tulajdonság megfigyelhető. Többféle adatbázis kezeléssel, form és adatellenőrzés, kifejlett cacheles és sablonkezelés jellemzi. Emellett rengeteg beépített osztállyal rendelkezik: e-mail küldése, képmanipulációs könyvtárak, FTP, titkosítás stb. (honlap: http://codeigniter.com)

Zend Framework

A Zend Framework egy nyílt forrású, objektumorientált webes keretrendszer, PHP 5-ben megvalósítva. Önálló összetevőket nyújt, az azonosításhoz és a jogosultságkezeléshez hozzáférés vezérlő listákon keresztül, adatok gyorsítótárazásához, a felhasználók által küldött adatok szűréséhez/ellenőrzéséhez a biztonság és az adatok sértetlensége érdekében, nemzetközisítéshez, felületeket AJAX funkciókhoz, e-mailek összeállításához/küldéséhez, illetve az összes Google Data API-hoz. (honlap: http://framework.zend.com)

5. CakePHP

A CakePHP egy szabad, nyílt forráskódú, gyors fejlesztői keretrendszer PHP-hoz. Az alábbi fejezetben bemutatom telepítését, felépítését, főbb funkcióit és tulajdonságait.

5.1 Telepítés

Telepítéshez az alábbi tulajdonságokkal kell rendelkeznie a fejlesztői környezetünknek:

- Webszerver (ideális választás lehet az Apache, a mod_rewrite funkció bekapcsolásával)
- PHP 4.3.2 vagy újabb változata
- Technikailag adatbázis hozzáférés nem szükséges, de valószínűleg úgyis ilyen formába szeretnénk adatainkat tárolni. Többek közt használható: MySQL (4 vagy újabb), PostgreSQL, Microsoft SQL Server, Oracle, SQLite, ADOdb stb.

A keretrendszer letöltése után (*http://cakeforge.org/frs/?group_id=23&release_id=428*) tartalmát másoljuk webszerverünkre. Ekkor az alábbi könyvtárszerkezet fogad minket:

/app
/cake
/docs
/vendors
.htaccess
/index.php

Az */app* könyvtárban találhatóak az alkalmazásunk fájljai, míg a */cake* könyvtárban a rendszerfájlok találhatóak, így ezekhez gyakorlatilag sosem kell nyúlnunk, csak az */app*-on belüli fájlokhoz. Ez a felépítés lehetővé teszi, hogy egy újabb verzió telepítésével ne veszélyeztessük saját munkánkat. A */docs* könyvtárban néhány szöveges dokumentum található, melyek felmásolása a webszerverre felesleges. A */vendors* könyvtárba másolhatjuk például azokat a külső osztályokat, amelyeket majd használni fogunk alkalmazásunkba. Lényeges, hogy az */app/tmp* könyvtárnak adjunk írási jogot, mert az alkalmazásunk az átmeneti fájlokat itt tárolja.

Értelemszerűen az */app/controllers*-ben találhatóak a vezérlők, az */app/models*-ben a modellek, az */app/views*-ben pedig a nézetek. Az */app/webroot* a webhelyünkhöz tartozó gyökérkönyvtár. A *.htaccess* fájlból pedig helyesen következtethetünk arra, hogy a CakePHP támogatja a felhasználóbarát URL-eket [2.2.4].

A könyvtárstruktúra ismeretében hozzákezdhetünk az alapvető beállításokhoz. Ilyen többek közt az adatbázis-kapcsolat, melyet az */app/config/database.php* fájl tartalmaz. Első telepítésre ez a fájl nem létezik, viszont a *database.php.default* fájl átnevezésével létrehozható. Ebben a fájlban található a DATABASE_CONFIG osztály, mely publikus változói tartalmazzák az adatbázis-kapcsolathoz szükséges paramétereket. Lehetőség van egyidejűleg több adatbázis-kapcsolat megadására is. Ezt a *var $test = array();* formájában adhatjuk meg, használata előnyös lehet, ha a fejlesztés során nem az éles adatbázist kívánjuk használni.
A core.php fájlban az egész rendszer működésére vonatkozó változókat és globális konstansokat állíthatunk be. A részletes dokumentációnak köszönhetően ezeket a beállításokat egyszerűen elvégezhetjük. Talán az egyik legfontosabb a DEBUG, mely a hibakövetéshez szükséges debug kimenetét állítja be. Négyféle értéket adhatunk:

- 0: éles üzemmód, nincsenek debug információk.
- 1: hibák és figyelmeztetések kiíratása.
- 2: hibák és figyelmeztetések, valamint az SQL-kérések és azok eredményeinek kiíratása.
- 3: az előző, valamint az adott modellhez tartozó objektum változói és függvényeinek listája.

A helyes és könnyed működés érdekében be kell tartanunk néhány konvenciós előírást, melyeket itt találhatunk: http://book.cakephp.org/view/22/CakePHP-Conventions

5.2 Gyorsváz

Egy webes alkalmazás első felében nagy segítséget nyújt a Scaffolding, azaz gyorsváz. A scaffolding folyamat megvizsgálja, analizálja az adatbázis tábláit, és egy listát készít szerkesztés céljából, hozzáad, töröl, és módosít gombokkal, illetve űrlapokat, valamint nézeteket, melyekkel az egyes adatbáziselemeket lehet vizsgálni. Azonban ez csak egy gyorsváz és nem egy rugalmasan módosítható eszköz, viszont a fejlesztés kezdeti stádiumába hatalmas segítség. Ehhez elegendő csupán a *$scaffold* változót hozzáadni a kívánt vezérlőhöz:

```
class CategoriesController extends AppController {
    var $scaffold;
}
```

5.3 Modellek

Mint ahogy már volt szó az előző fejezetben erről, a modell reprezentálja az üzleti logikát és a CakePHP alkalmazásokban az adatok elérésére használjuk. Egy modell általában egy adatbázis táblát képvisel, de használható bármi elérésére, ami adatok tárol. A modellek az */app/models* könyvtárban találhatóak. Konvenciós szabályok:

- A fájl nevének meg kell egyeznie a modell nevének kisbetűs változatával.
- Az osztály neve nagybetűvel kezdődik és a kapcsolódó tábla nevének egyesszámú alakja. Például az *articles* táblához tartozó modell neve *Article*.

A legegyszerűbb modell definíció:

```
class Article extends AppModel {
    var $name = 'Article';
}
```

A CakePHP egyik erőssége a modellek által biztosított kapcsolatok kezelésében van. Ez a CakePHP-ben a modell kapcsolatokon keresztül történik. Például az alkalmazásunk felhasználói adatait az *users* tábla tartalmazza, a létrehozott bejegyzéseket pedig az *articles*. A két tábla közt 1:N kapcsolat áll fenn. Az *articles* tábla *user_id* mezője megadja, hogy melyik bejegyzést melyik felhasználó hozta létre. A modellben ezt a kapcsolatot könnyen leírhatjuk, ezáltal az SQL lekérdezések automatikusan létrejönnek.

A CakePHP-ban négyféle kapcsolatot különböztetünk meg:

A *hasOne* az 1:1 kapcsolatokat reprezentálja (pl.: egy felhasználó egy bejegyzést hozhat létre).
A *hasMany* az 1:N kapcsolatokat írja le (pl.: egy felhasználó több bejegyzést is létrehozhat).
A *belongsTo*-t az 1:1 vagy 1:N kapcsolatok jobboldalán álló tábláknál állíthatjuk be (pl.: egy bejegyzés egy felhasználóhoz tartozik).
A *hasAndBelongsToMany* (HABTM) pedig az N:M kapcsolatokért felelős (pl.: egy bejegyzéshez több címke tartozhat, de ugyanakkor egy címke több bejegyzést is jellemez).

A fentebb említett felhasználók – bejegyzések kapcsolatot az alábbi módon írhatjuk le:

```
class User extends AppModel {
      var $name = 'Article';
      var $hasMany =
            array("Article" => array('className' => Article,
                                     'conditions' => '',
                                     'order' => 'name ASC',
                                     'foreignKey' => 'user_id',
                                     'dependent' => true,
                                     'exclusive' => false,
                                     'finderQuery' => ''));
}
```

Az összes modell az AppModel osztály leszármazottja. A szülőosztály számos beépített függvénnyel rendelkezik, ilyen többek közt:

findAll ([string $conditions, [array $fields, [string $order, [int $limit, [int $page, [int $recursive]]]]]])

Egy olyan SQL lekérdezést hajt végre, mely feltételeit a *$conditions* változó tartalmazza, visszatérési értéke egy asszociatív tömb. A *$fields* opcionális paraméter segítségével megadhatjuk, hogy mely mezőkre vagyunk kíváncsiak, melyek sorrendjére vonatkozó információkat az *$order* paraméter tartalmazza. A rekordok darabszámának korlátozásához a *$limit* paramétert állíthatjuk be, mely a *$page* oldaltól kezdve adja vissza a halmaz értékeit.

Amennyiben a modellünkben van kapcsolat és azt szeretnénk, hogy a lekérdezésben szerepeljen e ez a kapcsolat, valamint milyen mélységben azt a *$recursive* paraméterben adhatjuk meg.

A lekérdezéshez hasonló egyszerűséggel menthetjük adatainkat:

save (array $data,[boolean validate = true, [array $fieldlist]])

A *$data* tömbben adjuk meg a menteni kívánt adatokat, amelyeket a *$validate* alapján érvényesíthetünk, illetve a *$fieldset*-ben fel lehet sorolni a mentéskor engedélyezett mezőket.

5.4 Nézetek

A nézetek olyan kötelezően *.ctp* (CakePHP template) kiterjesztésű fájlok, melyek XHTML kódot tartalmaznak és az */app/views/controller_neve/* könyvtárban találhatóak. A legtöbb esetben egy controller eseményéhez egy nézetfájl tartozik, de persze előfordulhat akár az a szituáció is, amikor egy sem, vagy akár több is tartozik. A nézetekben csak a vezérlő *set()* által átadott változókat használhatjuk fel. A nézetek csupán a vezérlőtől kapott adatok megjelenítéséért felelősek, az egész honlap szerkezetét a *layout* fájlok adják meg (*/app/view/layouts/),* melyekből több is lehet. Az egyes nézeteknél a *setLayout()* megadásával választhatjuk ki a megfelelőt. Alkalmazásunk rendelkezhet olyan részekkel, melyek az összes funkciónál megjelennek, ilyen lehet például a menü, a be, ill. kiléptető űrlap. Ezek megjelenítésére használhatjuk az *elements*-eket (elemek), melyek az */app/views/elements/* könyvtárban találhatóak *.ctp* kiterjesztéssel. Az *elements*-eket a *layout* fájlokban tudjuk megadni az alábbi módon: *$this->element(pelda);*.

A nézeteknél használhatunk úgynevezett *Helper*eket (segítők), melyek olyan komponensszerű osztályok, amik megjelenítési logikát tartalmaznak, valamint megoszthatóak a views, elements valamint layouts között. Számos beépített helper található a CakePHP-ban, ilyenek például: Form, Html, JavaScript, RSS.

5.5 Vezérlők

A vezérlő az alkalmazás egy részének a logikájáért felelős. A controllerek az *AppController (/app/app_controller.php* fájl) osztály leszármazottjai. A főosztály metódusai elérhetőek az összes alosztály számára. A vezérlők tetszőleges számú metódust tartalmazhatnak, melyeket *action*-öknek (tevékenységeknek) nevezünk. A controllerek az /app/controllers/ mappában találhatóak. A CakePHP konvenciói alapján a vezérlő neve mindig többe számban álló angol főnév, valamint minden controllernek a *names_controller.php* alakot kell követnie.
Példa egy vezérlőre:

// file: articles_controller.php

```
class ArticlesController extends AppController {
        function index() {
            //az Articles controller index tevékenysége itt szerepel...
        }

        function show($id) {
            //a show metódus utasításai...
        }

        function delete($id = null){
            //pl. a http://pelda.hu/articles/delete/5/ hatasa
        }
    }
```

A vezérlők működését különböző változók segítségével tudjuk testre szabni:

- string $name: a vezérlő neve
- array $uses: a vezérlőben használatos modellek megadása
- array $helpers: a használni kívánt helperek megadása
- array $components: a használni kívánt komponensek listája
- string $layout: a használni kívánt layout neve
- string $pageTitle: az oldal címét tartalmazó változó megadása

Az AppController osztály rendelkezik számos metódussal, melyek közül néhány gyakran használt:

set(string $var, mixed $value)

A set metódus segítségével adatokat tudunk küldeni a vezérlőből a nézet számára. Például a *$this->set('szin', kék);* Ezután a nézetben a *$szin* változó értéke '*kék*' lesz. Lehetőség van egyetlen paraméterként egy asszociatív tömbként is megadni az átadni kívánt változókat.

render(string $action, string $layout, string $file)

A metódus automatikusan meghívódik minden egyes meghívott vezérlő tevékenysége után. A konvencióknak köszönhetően az alapértelmezett layout fájlt használja, de paraméterként megadható másik is.

A használható metódusok teljes listája az alábbi címen tekinthető meg: http://api.cakephp.org/1.2/class_controller.html

Lehetőség van komponensek létrehozására, melyek bizonyos feladatokat ellátó csomagok. A CakePHP rendelkezik számos beépített komponenssel, ilyenek például a security, sessions, ACL, emails, cookies, authentication. A vezérlőben használni kívánt komponenst az */app/controllers/components/* könyvtárban kell tárolnunk.

6. A webalkalmazás fejlesztése

Az alábbiakban bemutatom az alkalmazást illetve az egyes funkciók létrehozásának menetét, valamint a modellek, vezérlők, nézetek működését.

6.1 Az alkalmazás bemutatása

A szakdolgozatom témájául szolgáló web 2.0 alkalmazás elkészítésekor elsődleges céljaim közt az szerepelt, hogy egy olyan alkalmazást hozzak létre, mely Magyarországon még nem létezik, egyedülálló. Ebből kifolyólag a témaválasztás nem volt egyszerű, hisz hétről hétre új lehetőségek látnak napvilágot. A választás egy hozzám közelálló témára esett, mely alapján a búvárkodás iránt érdeklődő embereket helyezem a középpontba egy jól átlátható, könnyen használható funkciókkal rendelkező weboldalon. Éppen ezért a céljaim közt szerepel a szolgáltatás jövőbeli éles használata is. A látogatók tájékozódhatnak a legfrissebb hírekről, történésekről, mindezek mellett a regisztrált felhasználók kialakíthatják személyes profiljukat és a jól ismert közösségi funkciók révén az alkalmazás aktív részesévé válhatnak.

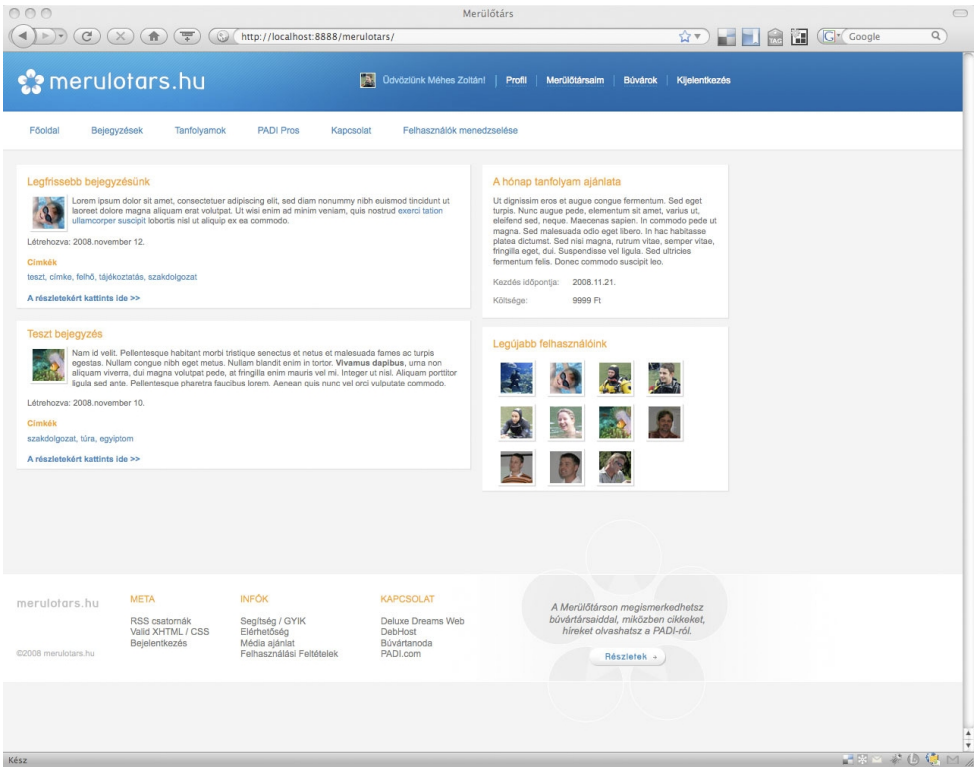
A technikai megvalósítás oldaláról az egyik legösszetettebb dolog a felhasználók különböző jogosultsági köreinek, illetve az alkalmazás alegységei láthatóságának a kialakítása volt. Ezt a CakePHP által támogatott ACL hozzáférési listák segítségével készítettem el, mely a legáltalánosabb módja, így egy teljesen újrafelhasználható kód lett az eredmény. Ezen lista alapján négyféle jogosultsággal rendelkező kör került kialakításra:

- Látogatók: Lehetősége van a bejegyzések, tanfolyamok valamint kapcsolat aloldalak megtekintésére, valamint a regisztrációra.
- Felhasználók: A regisztráció utáni állapot. A fentebbi lehetőségek mellett kialakításra kerül a saját fényképes profil, megtekinthető a többi felhasználó adatlapja, valamint lehetséges az ismerős tagok megjelölése, ezáltal kialakul egy ismeretségi háló. A többi felhasználó közt a keresés engedélyezett, akár szűrési feltételek segítségével is. Valamint az egyes bejegyzésekhez fűzhető megjegyzések is engedélyezettek számukra.
  A regisztráció folyamán a „Milyen szinten búvárkodsz?” kérdésre amennyiben a látogató a *hivatásos* opciót választotta, egy kiterjesztett jogosultsági körbe kerül. Ehhez szükséges

egy központilag kiadott, nemzetközileg elismert azonosítási kód megadására. Azonban az ezt tároló adatbázishoz automatikusan nem lehet hozzáférni. Mivel elméletileg bármilyen számjegyekből álló azonosító elfogadásra kerülhet, ezért csak az adminisztrátorok kézi ellenőrzése után lehet ebbe a jogkörbe kerülni.

- Pro felhasználók: A fentebb említetteken kívül egy ’PADI Pros’ menüpont is aktívvá válik, melyben állásajánlatok és munkaközvetítések valamint hirdetések segítik a felhasználókat.
- Adminisztrátorok: A legszélesebb jogkörrel rendelkező szekció. Az összes aloldal szerkesztése engedélyezett (CRUD – Create, Read, Update, Delete), mindezek mellett a felhasználók adminisztrálására is lehetőség van.

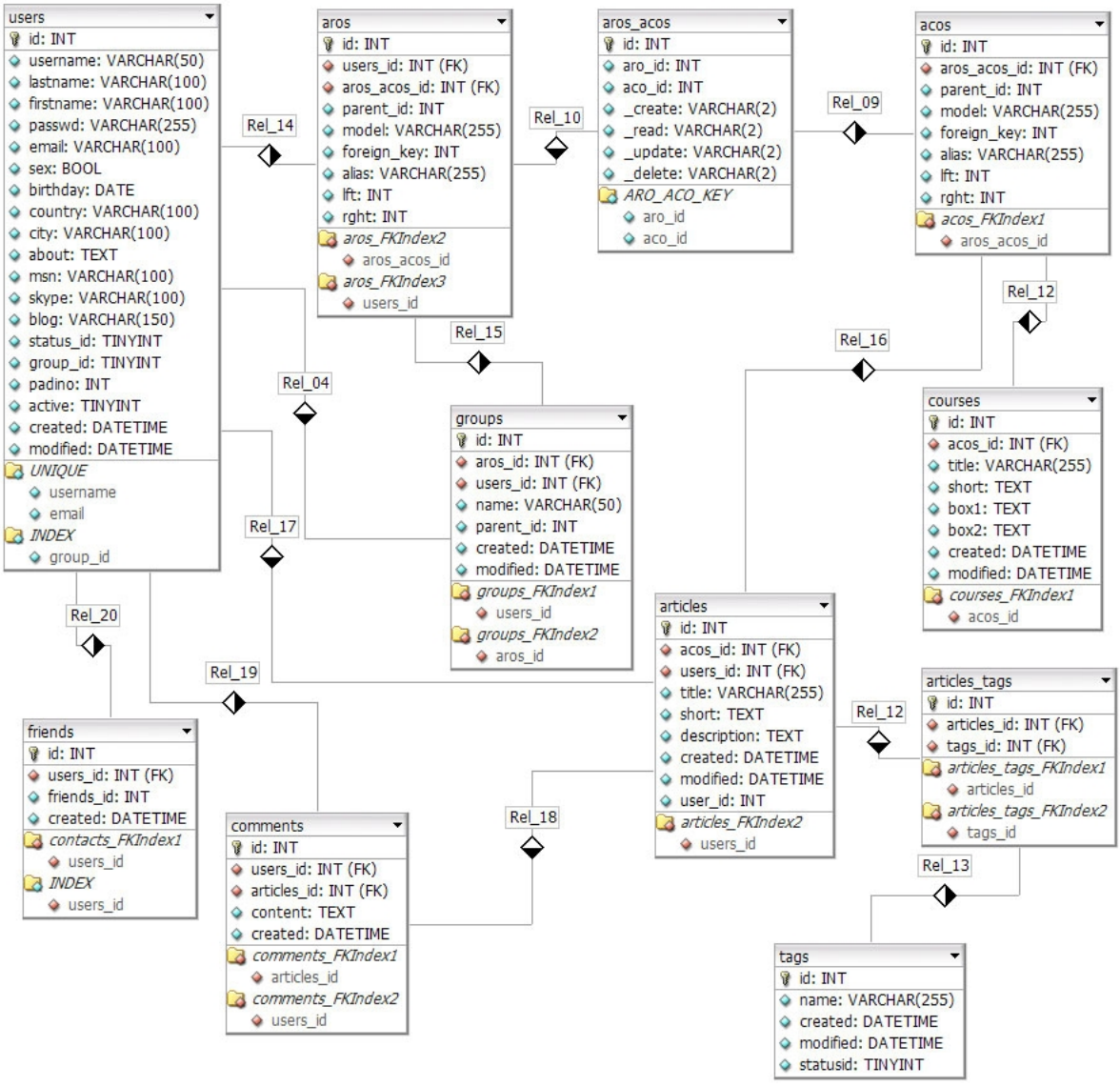
Az adminisztrátorok által szerkesztett bejegyzéseket el lehet látni címkékkel, a könnyebb rendszerezhetőség érdekében, valamint a másik megszokott web 2.0-s szolgáltatás az RSS hírlevél is jelen van. Az egész alkalmazásnál megjelennek az AJAX technológiának köszönhető felhasználói élményt növelő tulajdonságok.



6. ábra: Az alkalmazás főoldala

6.2 Az adatbázis kialakítása

A felhasználói funkcióknak és a CakePHP ide vonatkozó konvenciós irányelveinek eleget téve került kialakításra az adatbázis. Melyek szerint a táblák nevei angol többes számú főnevek. Mindezek mellett az egyes mezők nevei is angol szavak. Mindegyik tábla elsődleges kulcsa *id*, az egyes idegen kulcsok pedig *táblaneve_id* formában találhatóak.



7. ábra: Az alkalmazás alapjául szolgáló adatbázis séma

Az ábrán szereplő táblák funkciói:

- users: a felhasználók adatainak tárolása
- friends: a felhasználók közösségi kapcsolatait leíró tábla
- aros: az ACL-ben az ARO (Access Request Object) adatait tartalmazó tábla.
- acos: ACO ACL (Access Control Object) adatait tartalmazó tábla.
- aros_acos: az *aros* és az *acos* kapcsolótáblája
- groups: a felhasználók jogköreit tartalmazó tábla
- courses: a tanfolyamok aloldalhoz kapcsolódó tábla
- articles: a bejegyzések szekcióért felelős tábla
- comments: a bejegyzésekhez fűzött hozzászólások táblája
- tags: a bejegyzésekhez kapcsolódó címkék
- articles_tags: az *articles* és a *tags* kapcsolótáblája

6.3 Fejlesztői környezet

Az alkalmazás fejlesztése párhuzamosan történt PC valamint Mac platformon. Szerver oldalról, a PC Apache 2.2 webszerverrel, PHP 5.2.6 értelmezővel valamint MySQL 5 adatbázissal rendelkezett, Windows XP operációs rendszer alatt. A Mac OS X 10.5.5 (Leopard) operációs rendszer alatt pedig Apache 2 webszerver mellett PHP 5.2.5 valamint MySQL 5 állt rendelkezésemre. Kliens oldalról mindkét esetben a Firefox böngésző 3-as verzióját használtam.

A fájlok szerkesztéséhez Windows operációs rendszer alatt a PHP Designer 2007 (http://www.mpsoftware.dk) programot, míg OS X alatt a Coda 1.5 (http://www.panic.com/coda) használtam. Az adatbázis tervezéséhez a DB Designer 4 (http://www.fabforce.net/dbdesigner4), míg az egyéb ábrák készítéséhez az OmniGraffle 5 (http://www.omnigroup.com) programot használtam.

Az alkalmazás tárolásáért felelős szerveren Debian 4.0 operációs rendszer alatt PHP 5.2.5 értelmező és MySQL 5 adatbázis található.

6.4 A hozzáférést vezérlő listák

Az alkalmazás felhasználói jogköreit, valamint az egyes alegységek láthatóságának meghatározásához az ACLBehavior-t alkalmaztam, mely használatával kis lépésekben ugyan, de könnyen kezelhetően és karbantarthatóan engedélyezhetjük, vagy épp tilthatjuk le az egyes funkciókat. Az ACL-ek két dolgot kezelnek, egyrészről amik hozzáférést kérnek, jelen esetben a felhasználók. Ezt hívjuk ARO-nak (Access Request Object), azaz Hozzáférést Kérő Objektumnak. Viszont a felhasználók adatait nem itt tároljuk (az alkalmazásban az *users* tábla a felelős ezért)! Másrészről, amitől engedélyt kérünk, ezek lehetnek különböző események, vagy éppen adatok, ezt hívjuk ACO-nak (Access Control Object), azaz Hozzáférést Kontrolláló Objektumnak. Az ACL fa-struktúraként van megvalósítva, létezik egy ARO és egy ACO fa. Amennyiben valamilyen jogosultságot adunk az egyik csomópontnak az idetartozó levélelemekre is érvényes lesz.

Ez a jogosultsági rendszer adatbázis alapú, rendelkezik egy központi modellel mely elsősorban az ACL csomópontok adatbázisba mentésére és olvasásra használatos. Segítségünkre van egy parancssori script, mellyel viszonylag egyszerűen megtehetjük a kialakítás első lépéseit. Ehhez a `cake schema run create DbAcl` parancsot kell megadnunk a konzolba. Ekkor létrejönnek a megfelelő táblák, melyek a csomópontokat és ezek fában való helyét tárolják. Az *acos* és az *aros* táblák tárolják a hozzájuk tartozó információkat, míg az *aros_acos* tábla összekapcsolja őket.

Az alkalmazásban az ACL CRUD módban való alkalmazása kapott helyett, ekkor a négy művelethez – Create, Read, Update, Delete – kapcsolunk jogosultságokat. Melyek az adatbázistáblában egy-egy attribútumként vannak jelen és a megfelelő mezőben 1, 0 vagy -1 jelöli a hozzáférést, vagy éppen a jogosultság megtagadását. A -1-el pedig megvonhatjuk az öröklött jogosultságokat. (Emellett létezik egy másik mód, mely az *action*-ökhöz kötött jogosultságot veszi alapul.)

id	parent_id	model	foreign_key	alias	lft	rght
1	*NULL*	Group	1	users	1	10
2	1	Group	2	prousers	2	7
3	2	Group	3	admins	3	6
4	3	User	1	adminvagyok	4	5
5	1	User	2	tesztlajos	8	9

8. ábra: Az aros tábla

Az alábbi *aros* táblában szereplő *users* csoportra vonatkozó jogosultságokat mind örökli a többi csoport illetve felhasználó, ugyanis az összes rekord *lft* és *rght* értéke az itt szereplő 1 és 10 közé esik. Mindeközben az *adminvagyok* felhasználó rendelkezhet csak rá vonatkozó jogokkal és ugyanakkor örökli az *admins*, *prousers* valamint *users* csoportok jogosultságát is az *lft* és *rght* értékei alapján.

Az *acos* tábla felépítése teljesen megegyezik a fenti *aros* tábla struktúrájával.

id	aro_id	aco_id	_create	_read	_update	_delete
1	3	1	1	1	1	1
2	1	3	-1	-1	-1	-1
3	1	1	0	1	0	0
4	1	2	0	1	0	0

9. ábra: Az aros_acos tábla

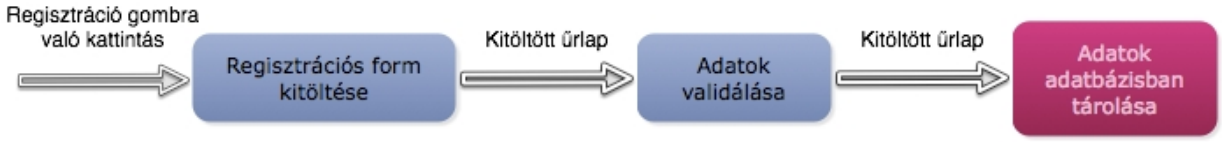
Az itt látható *aros_acos* adja meg a CRUD hozzáféréseket az *aros* és *acos* táblák alapján. A 3-mas *aro_id*-vel rendelkező (*adminvagyok* felhasználó) hozzáférést kérő objektum az 1-es jelzésű *aco_id*-re vonatkozóan (ami lehet például a felhasználók menedzselésére vonatkozó aloldal) rendelkezik minden jogosultsággal. De ugyanakkor azt is láthatjuk, hogy 3-mas *aco_id*-re vonatkozóan a *prousers*-ek nem örökölnek semmit sem az *users* csoporttól, viszont az 1-es és 2-es *aco_id* objektumokra kapnak olvasási jogot.

Az *aros* és az *acos* tábla megfelelő mezőinek feltöltéséhez a modellben lévő metódusok a felelősek, melyeket az */app/models/user.php* fájl tartalmaz. Ilyen többek közt a *parentNode()* amely a szülő – gyermek leszármazott viszony meghatározásáért felelős. Az alkalmazásban egy viszonylag összetettebb szituáció érvényes, ugyanis az *users* tábla a *groups* táblával *belongsTo* kapcsolatban áll:

```
var $name = 'User';
var $belongsTo = array('Group');
…
function parentNode(){
      if (!$this->id) {
            return null;
      }
      $data = $this->read();
      if (!$data['User']['group_id']){
            return null;
      } else {
            return array('model' => 'Group',
                        'foreign_key' => $data['User']['group_id']);
      }
}
```

6.5 A felhasználókkal kapcsolatos műveletek

Az */app/controllers/users_controller.php* fájl tartalmazza a felhasználókra vonatkozó összes függvényt. Elsőként nézzük meg a felhasználók regisztrációjáért felelős részt, melyért a *signup()* metódus a felelős. A folyamat általános módon történik, vagyis a kitöltött form elküldése után, amennyiben a megadott adatok eleget tesznek a modellben megadott validálási szabályoknak, az adatok tárolásra kerülnek az users táblában, valamint meghatározásra kerülnek a hozzáférési listához szükséges paraméterek, különben a helytelen adatok módosítása szükséges.



9. ábra: A regisztráció menete

```
function signup() {
      if (!empty($this->data)) {
            if(!empty( $this->data['User']['new_passwd'] ) ){
                  $this->data['User']['new_passwd_hash'] =
                  $this->Auth->password( $this->data['User']['new_passwd'] );
            }
            if ($this->User->save($this->data)) {
                  $lastId = $this->User->getLastInsertId();
                  $this->flash(Sikeres regisztráció','/users/view/'.$lastId );
            } else {
                  $this->data['User']['new_passwd'] = null;
                  $this->data['User']['confirm_passwd'] = null;
            }
      }
}
```

Első lépésben a felhasználó által megadott *new_passwd* jelszóból képzett hasht helyezzük el egy új, *new_passwd_hash* változóban, melyet a modellben található *beforeSave()* metódusban hagyunk jóvá. Majd amennyiben sikeres volt a validálás és ezáltal a regisztráció is, közöljük a

felhasználóval és egyből a saját adatlapjára irányítjuk, mely a */users/view/legutobbi_id* címen lesz elérhető. Sikertelen validálás esetén visszairányítjuk a regisztrációs formhoz, viszont biztonsági okokból a jelszóra vonatkozó adatokat *null*-ra állítjuk.

A regisztráció folyamán lehetőség van a már említett búvárkodási szint megadására is (kedvtelési, avagy professzionális). Azonban az ily módon szétválasztott két kategóriához különböző információk tartozhatnak. Ilyen például a hivatásosok számára az azonosítókód megadása. Azonban azt szeretnénk, hogy ez az *input* elem csak abban az esetben jelenjen mikor a látogató a „hivatásos" opciót választotta. Ezt egy AJAX technikával egyszerűen megoldhatjuk.

Ehhez a fentebb említett jQuery (http://jquery.com) JavaScript könyvtárat alkalmaztam. A letöltött JS fájlt az */app/webroot/js* könyvtárba helyezése után a használt *layout*-ban meg kell adnunk:

```
if(isset($javascript)) {
      echo $javascript->link(array('jquery-1.2.6.pack', 'common'), true);
}
```

Az alapvető könyvtáron kívül ezzel hozzáadtuk a *common.js* fájlt is a *helper*-hez, melyben helyet kap a számunkra szükséges JavaScript kód. Majd az users_controller.php fájlunknál az alábbi formába hivatkozunk rá: `var $helpers = array('Html', 'Form', 'Javascript');`

A nézetben (*/app/views/users/signup.ctp*) a manipulálni kívánt input elem:

```
echo $form->input('proId',array('label'=>Azonosítód:','class'=>'proId_hide'));
```

Az ezért felelős Javascript kód a *common.js* fájlból:

```
$(document).ready(function(){
	// A legördülő menü értékének kiválasztása
	var current_type = $('.pro_select option:selected').text();
	// amennyiben nem 'Hivatásos' a proId_hide elrejtése
	if(current_type != "Hivatásos") {
		$('.proId_hide').parent().hide();
	}
	// add az eseménykezelőnek a kiválasztott elemet
	$('. pro_select').change(function(){
		// a kiválasztott elem value eleme
		var type = $(this).find('option:selected').text();
		// ha ez „Hivatásos" akkor…
		if(type == "Hivatásos") {
			// a proId_hide elem megjelenítése
			$('.proId_hide').parent().fadeIn();
		} else {
			// a proId_hide elem eltüntése
			$('.proId_hide').parent().fadeOut();
		}
	});
});
```

A felhasználók beléptetése

Ahhoz, hogy a látogatók számára elérhető részeken kívüli szekciókat is el tudják érni a jogos felhasználók, be kell jelentkezni, amit az */app/controllers/users_controller.php* fájl *login()* metódusával tudunk megtenni. Amennyiben a bejelentkezés gombra kattintott a látogató, azaz elküldésre került POST adat a függvény megvizsgálja, hogy az „emlékezz rám" opciót kiválasztotta e, amennyiben nem, a felhasználó adatát törli a cookie-ból, különben a cookie engedélyezve lesz és a felhasználónevet és a jelszót eltárolja, majd átirányítja a kezdőoldalra, mint bejelentkezett felhasználót.

```
var $cookieTerm = '+4 weeks';
function login() {
      if ($this->Auth->user()) {
            if (!empty($this->data)) {
                  if (empty($this->data['User']['remember_me'])) {
                        $this->Cookie->del('User');
                  } elseif( $this->allowCookie ) {
                        $cookie = array();
                        $cookie['username'] = $this->data['User']['username'];
                        $cookie['passwd'] = $this->data['User']['passwd'];
                        $this->Cookie->write('User', $cookie, true,
                              $this->cookieTerm );
                  }
                  unset($this->data['User']['remember_me']);
            }
            $this->redirect('/', null, true);
      }
}
```

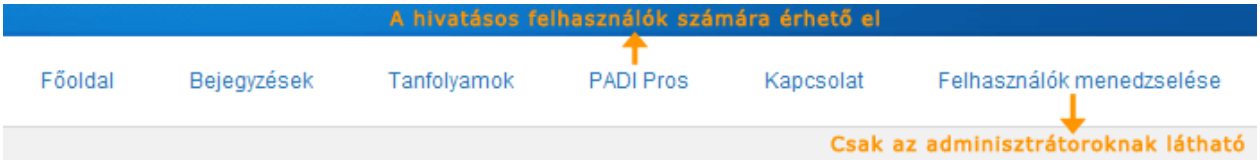
A bejelentkezett felhasználókat meg kell vizsgálni minden egyes szekció előtt, hogy jogosultak e annak használatára, ehhez az alkalmazás *app_controller.php* szülőosztályában elhelyezkedő *beforeFilter()* metóduson belüli ellenőrzéssel tudjuk megtenni, ugyanis ez minden egyes oldalletöltéskor meghívódik. Itt megadhatunk külön action-öket amelyeket mindig engedélyezhetünk `$this->Auth->allow( $allowActions );`, vagy éppen tilthatunk `$this->Auth->deny( $denyAction );`. Ezek után pedig az ACL által meghatározott jogosultság alapján kerül elbírálásra, hogy van e hozzáférése, vagy sem. Hiba esetén: `$this->Auth->authError = 'Nem vagy jogosult az oldal megtekintésére!';`.

A kilépést a *logout()* metódus végzi:

```
function logout() {
      $this->Auth->logout();
      $this->Cookie->del('User');
      $this->flash("Sikeresen kijelentkeztél az oldalról.", '/' );
}
```

6.6 A menü felépítése

Az alkalmazásban szereplő menüpontok megjelenítése dinamikusan történik, így mindenki csak az általa használható részeket látja. A megjelenítésért a *layout*-ba ágyazott element (/app/views/elements/menu.ctp) gondoskodik. A megfelelő menüpontok kiválasztása és tárolása az AppController osztályban van megadva.



10. ábra: A menü kialakítása

A menü kialakítását a beforeRender() metódusban található __buildMenu() függvény végzi:

```
function __buildMenu(){
	$this->menu = array();
	foreach( $this->menuItems as $menuLink ){
		if( $menuLink['protected'] ){
			if( $this->loggedUser && $this->Acl->check(
		$this->loggedUser, $menuLink['controller'], $menuLink['crud'] ) ){
				$this->menu[] = array(
					'label' => $menuLink['label'],
					'url' => $menuLink['url']);
			} else { continue; }
		} else {
			$this->menu[] = array('label' => $menuLink['label'],
			'url' => $menuLink['url']);
		}
	}
	$this->set('merulotars_menu', $this->menu);
}
```

A feldolgozandó tömb, mely tartalmazza, az egyes menüpontok információit az alábbi módon épül fel:

```
var $menuItems = array(
      array(
            'protected' => FALSE,
            'label' => 'Bejegyzések',
            'controller' => 'Articles',
            'crud' => 'read',
            'url' => '/articles/index'
      ),
      …
);
```

A megfelelően kialakított menüt a *menu* element jeleníti meg:

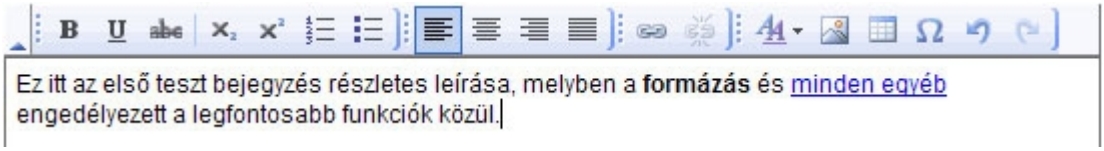
```
<ul class="navigation" id="navigation">
      <?php foreach($merulotars_menu as $link): ?>
      <li>
            <a href="<?php echo $html->url($link['url']); ?>">
            <span><?php echo $link['label']; ?></span></a>
      </li>
      <?php endforeach; ?>
</ul>
```

Hasonló módon történik az adminisztrációhoz kapcsolódó linkek kialakítása is. Az aktív vezérlő *action*-éhez vonatkozó funkciók vannak csak megjelenítve. Például az */articles/view/3/* oldal megtekintésénél az *edit* és *delete* tevékenységek az elérhetőek, viszont az */articles/index* esetén az *add*, azaz új bejegyzés létrehozása.

6.7 Bejegyzések, címkék és az RSS

Az adminisztrátorok által szerkeszthető bejegyzések és tanfolyamok szekciók bővítéséhez, módosításához a CRUD alapműveletek érhetőek el. Egy új bejegyzés létrehozásakor kötelezően meg kell adni a bejegyzés megnevezését, valamint lehetőségünk van egy rövid – figyelemfelkeltő – és egy részletes – leíró – rész megadására. Mindezek mellett egy, a cikkhez tartozó index kép is feltölthető, melyből automatikusan készül egy kisméretű kép is (thumbnail). A bejegyzés pontosításához pedig különböző címkékkel ruházhatjuk fel a leírásunkat. Az elkészült bejegyzéshez a regisztrált felhasználók hozzászólásokat fűzhetnek.

A rövid és részletes leírások szerkesztésének megkönnyítéséhez az FCKeditor (http://www.fckeditor.net) HTML alapú WYSIWYG szerkesztőt használtam, mely segítségével szinte az összes ismert funkciót elérhetjük, melyet egy asztali szövegszerkesztőnél is használunk.



11. ábra: Az FCKeditor formázási funkciói

Mivel az alkalmazásban több helyen is megjelenik ez a funkció, egy *helper* segítségével egyszerűbbé lehet tenni használatát, melyet az *app/views/helpers/fck.php* fájl tartalmaz:

```
App::import('Vendor', 'fckeditor');
class FckHelper extends AppHelper {
	function fckeditor($namepair, $content){
		$editor_name = 'data';
		foreach ($namepair as $name){ $editor_name .= "[" . $name . "]"; }
		$oFCKeditor = new FCKeditor($editor_name);
		$oFCKeditor->BasePath = '/js/fckeditor/';
		$oFCKeditor->Value = $content;
		$oFCKeditor->Create();
	}
}
```

Amely controllerben használni szeretnénk, elegendő csupán hozzáadni a *helpers* tömbhöz:

```
var $helpers = array('Html', 'Form', 'Fck');
```

A megjelenítéshez pedig:

```
echo $fck->fckeditor(array('Modell', 'mezőneve'), $FCKParaméterek);
```

Címkék

A bejegyzéseket elláthatjuk címkékkel (*tags*), így könnyebben rendszerezhetőbbé tehetjük. Az egyes címkékhez kapcsolódó információkat a *tags* tábla tartalmazza, az *articles* táblával való kapcsolatot pedig az *articles_tags* kapcsolótábla biztosítja. A bejegyzések és a címkék HasAndBelongsToMany (HABTM) kapcsolatban állnak, ami azt jelenti, hogy egy bejegyzés kaphat több címkét, viszont egy címke tartozhat több bejegyzéshez is.
Nézet szempontjából egy input mezőbe tudjuk megadni a tageket:

```
echo $form->input('tags', array('label'=>'Címkék:', 'type'=>'text'));
```

A vezérlőben pedig a *_parse_tags()* privát függvény gondoskodik, hogy az egyes címkék vesszővel legyenek elválasztva egymástól, ellenőrzi, hogy létezik e már ilyen címke az adatbázisban. Ha nem létezik, létrehozza és az *insert id*-vel tér vissza, különben a címke *id*-jével

```
function _parse_tags($tags = null) {
      $data = array();
      $explode = explode(',', $tags); // vesszővel való szétválasztás

      if(!empty($explode)) { // ha az explode tömb nem üres
            foreach($explode as $tag) {
                  $tag = trim($tag); // felesleges szóközök eltávolítása
                  if(!empty($tag)) { // ha a tag nem üres az Ab-ban keresése
                        $db_tag = $this->Article->Tag->find('first', array(
                              'conditions' => array(
                                    'Tag.name'  => $tag,
                                    'Tag.statusid'    => '1'
                              )
                        ));
```

```
                        if(!empty($db_tag)) { // ha megtalálta id mentése
                              $data[] = $db_tag['Tag']['id'];
                        } else { // különben új tag létrehozása
                              $save = array('Tag'=>array(
                                    'name'      => $tag
                                    'statusid'=> 1
                              ));
                              $this->Article->Tag->create(); // több elemnél
                              // az új tag mentése
                              $saveOK = $this->Article->Tag->save($save);
                              if($saveOK) { // ha a mentés sikeres volt
                                    $data[] = $this->Article->
                                          Tag->getLastInsertID();
                              }
                        }
                  }
            }
      }
return array('Tag' => $data);
}
```

Az *add()* actionnél, azaz a bejegyzések hozzáadásánál a címkék mentése:

```
      $this->data['Tag'] = $this->_parse_tags($this->data['Article']['tags']);
```

A megjelenítés megkönnyítéséhez az alábbi privát függvény egy olyan sztringet hoz létre melyben a címkék vesszővel vannak elválasztva (például: tag1, tag2, tag3,).

```
function _create_tag_string($tags) {
      $tag_string = '';
      if(!empty($tags)) { // ha a tags tömb nem létezik
            foreach($tags as $g) {
                  $tag_string .= $g['name'].", ";
            }
      }
return $tag_string;
}
```

RSS hírlevél készítése

A bejegyzésekhez kapcsolódó RSS hírlevél elkészítése a CakePHP 1.2-ben szereplő *RSSHelper* és a *RequestHandler* segítségével történik. Utóbbi egy olyan lekérés-kezelő komponens, mellyel kiegészítő információkat kaphatunk a HTTP lekérésekről. A *Router::parseExtension()* használatával pedig automatikusan átkapcsol arra a *layout*-ra amely illeszkedik az adott lekérés típusához. Jelen esetben gyakorlati haszna abban mutatkozik, hogy az *articles* controller egy action-évl ki tudjuk szolgálni az XHTML és az RSS kéréseket is.

Ehhez hozzá kell adni a használt komponensek tömbjéhez:

```
var $components = array('RequestHandler');
```

Valamint az *app/config/routes.php* fájlhoz: `Router::parseExtensions();`.
Ezáltal a rendszerünk tudni fogja, hogy ha a kérés az */articles* vezérlőre érkezik akkor az XHTML tartalmat jelenítse meg, viszont ha */articles.rss* a kérés akkor az RSS feed kerül előtérbe.
A */views/posts/rss* könyvtárba lévő *index.ctp* fájl:

```
items($posts, 'transformRSS');
function transformRSS($article) {
      return array(
            'title' => $post ['Article']['title'],
            'link' => array(
                  'action' => 'view', $article['Article']['id']),
                  'description' => $article['Article']['short'],
                  'author' => $article['Article']['users_id'],
                  'pubDate' => $article['Article']['modified']);
      );
}
```

Az index() action-ben pedig:

```
      $channel = array ('title' => 'Merülőtárs', 'link' =>
'http://www.merulotars.hu', 'description' => 'Leírás');
$this->set('channel', $channel);
```

Valamint a *views/feeds/index.ctp* nézethez az `items($items);` hozzáadásával el is érhető a bejegyzésekhez kapcsolódó hírcsatornánk.

6.9 A felhasználói kapcsolatok kialakítása

A felhasználók a regisztráció folyamán megadott információkat a későbbiek folyamán tudják módosítani. A különböző adatok és elérhetőségeken kívül fénykép feltöltésére és a közösségi oldalakra jellemző kapcsolatok kiépítésére is lehetőségük van. Az ismerősök bejelölése és annak visszaigazolása igen egyszerű módon van megvalósítva, ugyanis relatív kis létszámú felhasználóval fog működni a szolgáltatás. Így teljesen felesleges a nagy közösségi oldalakhoz hasonló bonyolult adatbázisok használata, - például a FaceBook, a legnagyobb social network szolgáltatás 1800 db MySQL 5 adatbázis szervert, 10000 db webszervert és 805 db cache szervert használ működéséhez.

A kapcsolatokat a *friends* tábla írja le, mely rekordjai többek közt három darab azonosítóból állnak. Az *id* a kapcsolatot azonosítja, míg az *user_id* a felhasználó azonosítója és a *friend_id* pedig a hozzá tartozó ismerősének az azonosítója.
Így lehetőség van csak azoknak a felhasználóknak a lekérdezésére, akik az adott *user_id*-hez tartoznak (*@ID*):

```
SELECT DISTINCT u.*
FROM Users u
      INNER JOIN Friends f ON u.id = f.friend_id
WHERE f.user_id = @ID
```

Azon felhasználók lekérdezése, akik az *@ID* felhasználót ismerősüknek jelölték:

```
SELECT DISTINCT u.*
FROM Users u
      INNER JOIN Friends f ON u.id = f.friend_id
WHERE f.friend_id = @ID
```

Azoknak a felhasználóknak a listája, akikkel kölcsönös a kapcsolati szándékunk, azaz mi is és ő is ismerősének tekint, a fenti két lekérdezés halmazát kell vennünk *UNION* képzéssel.

6.10 A felhasználói felület

A felhasználói felület kialakításakor szabványos, táblázatmentes XHTML kód került kialakításra, a stílusok megadásához pedig a CSS2 stílusleíró nyelv szabványos elemeit használtam. Az alkalmazás felépítését az */app/views/layouts/default.ctp* fájl tartalmazza.
Az alkalmazást az alábbi részekre lehet bontani:

Fejléc

A fejlécet (*header*) az */app/views/elements/layout_head.ctp* fájl alkotja, mely attól függően, hogy a felhasználó be van e jelentkezve, a saját adatai és profiljával kapcsolatos információkat jeleníti meg, vagy éppen a belépés és regisztráció található meg.

12. ábra: A fejléc látogatóként és belépett felhasználóként

Menü

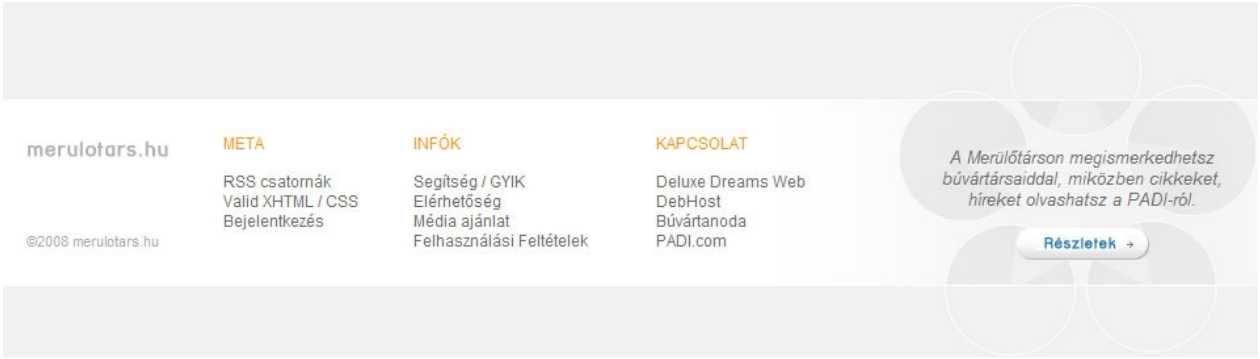
Itt is hasonlóan, dinamikusan épülnek fel a menüelemek a jogosultságtól függően, melyről bővebben a 6.6 fejezetben volt szó.

Tartalmi rész

A layoutban a *$content_for_layout* változó tartalmazza ezt a részt, vagyis az egyes nézet fájlok a felelősek e rész megjelenítéséért. Két hasábból épül fel ahol a tényleges tartalom a baloldalon, a jobboldali keskenyebb sávban pedig kiegészítő információk találhatóak. Például a főoldalon itt kap helyet a tanfolyamajánló és a legújabb regisztrált felhasználók kicsinyített képei is. Ugyanakkor a bejegyzéseknél az azonos címkével rendelkező bejegyzések ajánlása is itt található.

Lábléc

A lábléc (*footer*) kialakítása statikus, így mindig ugyanazokat az információkat közli, ilyenek többek közt a felhasználási feltételek, elérhetőségek, kapcsolat, valamint az oldal bemutatásáért felelős részt is innen lehet elérni.



13. ábra: A lábléc kialakítása

A tartalmi részben szereplő nézet fájlok az egyes vezérlőkhöz tartozó *action()*-ökkel megegyező névvel az */app/views/vezérlő/* könyvtárban találhatóak. A statikus oldalak, mint például a *Kapcsolat*, pedig az */app/views/pages/* könyvtárban kapott helyet.
A CSS fájlnak az */app/webroot/css/*, míg a JavaScript fájloknak az */app/webroot/js/* ad helyet.

Az alkalmazás néhány helyén az AJAX technikának köszönhetően megnövekedett felhasználói élményre tehetünk szert. Ilyen többek közt a már említett regisztráció, de ugyanakkor az adatok valós időben való validálása is ily módon lett megvalósítva.

7. Összefoglalás

A szakdolgozatom célja a web 2.0, valamint az új szemléletek bemutatása volt, egy CakePHP keretrendszer segítségével elkészített alkalmazás fejlesztése mellett. A dolgozat elméleti részében, az elsősorban AJAX technológiára épülő második generációs weboldalak ismertetéséről volt szó. Számos szolgáltatást próbáltam ki, amelyek közül néhányat be is mutattam, ennek köszönhetően konkrétan láthatóak az újdonságok és a felfogásbeli újítások.

Mindezek mellett a fejlesztés és annak menetével kapcsolatban az MVC tervezési mintára alapuló CakePHP keretrendszer használatát elemeztem. A fejlesztés előtt már rendelkeztem a szükséges webfejlesztési ismeretekkel (PHP, MySQL, XHTML, CSS), azonban a fejlesztés ilyen irányú változatával most ismerkedtem meg részletesebben. Megtapasztaltam a CakePHP rengeteg előnyét és néhány hátrányát is. Így ténylegesen megállapíthatom, hogy valóban megkönnyíti a fejlesztést, legyen szó egy egyszerű modul készítéséről, vagy a modellek összekapcsolásának köszönhetően egy komplex alkalmazásról.

A bemutatott alkalmazás jelenleg belső tesztelés alatt áll, a fejlesztése folyamatosan történik. Jelen állapotában még nem alkalmas az éles működésre, azonban néhány funkcióval kiegészítve remélem, elérem célomat, azaz ennek a szűk érdeklődési körrel bíró felhasználói rétegnek egy olyan alkalmazás áll majd rendelkezésére, amely köré szerveződve aktív részesévé válhatnak annak működésében. Így nem csak egy webes platformként, hanem személyes kapcsolatok kialakításához is vezethet.

Összességében úgy érzem sikerült bemutatnom ezt a nagyszerű irányba fejlődő ágazatot. Mindezek mellett én is számos új információra tettem szert, mind az újgenerációs alkalmazások, mind a CakePHP fejlesztési irányából véve, amelyek remek kiindulási alapot jelenthetnek a későbbi fejlesztéseim során.

8. Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani témavezetőmnek, Dr. Kuki Attilának, hogy lehetőséget biztosított munkám sikeres elvégzéséhez, valamint dolgozatom megírásához nyújtott útmutató tanácsaiért.

9. Irodalomjegyzék

[1] John Musser, Tim O'Reilly: Web 2.0 Principles and Best Practices

[2] George Schlossnagle: PHP fejlesztés felsőfokon. Kiskapu kiadó, 2005.

[3] Kris Handlock – Webalkalmazások fejlesztése AJAX segítségével

[4] What is Web 2.0:
http://oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html

[5] Model-view-controller: http://en.wikipedia.org/wiki/Model-view-controller

[6] CakePHP hivatalos honlap: http://cakephp.org

[7] CakePHP felhasználói dokumentáció: http://book.cakephp.org/

[8] CakePHP API dokumentáció: http://api.cakephp.org/