Debreceni Egyetem
Informatikai Kar

PHP alapú keretrendszer készítése Web 2.0 alapokon

Témavezető:
Dr. Adamkó Attila
egyetemi adjunktus

Készítette:
Gál József
programtervező informatikus (MSc)

Debrecen
2010

Tartalomjegyzék

Bevezetés

Az internet a mai világ mindennapjainak egyik szinte nélkülözhetetlen eszköze. Az ARPANET 1969-es megjelenésekor senki sem gondolta volna, hogy alig 40 év elteltével ez korábban katonai és tudományos célokat szolgáló hálózat a világot átszövő rendszerré növi ki magát. Végül 1991-ben megszületett a World Wide Web, első formájában. Mérete rohamosan nőtt, és ez a mai napig sem állt le, közben pedig különféle technológiák, módszerek és programok alakultak ki használatának megkönnyítése érdekében.

A kezdetekkor mindössze néhány weblap volt, melyek főleg pusztán információtartásra szolgáltak. Statikus lapok, amik az évek folyamán nem, vagy alig változtak, megtalálásuk pedig nehézkes volt, sokszor órákat vett igénybe egy-egy adatmorzsa felkutatása. A rendszer fejlődésével felmerült az igény arra, hogy ne pusztán adatokat tároljunk, tegyük is interaktívvá a weboldalakat: megjelentek a szerveroldali programozás alapjai. Ma három technológiát érdemes megemlíteni, amivel a különböző szerverek dinamizmust vihetnek a honlapokba: a Microsoft-tól származó ASP.NET (Active Server Pages), a Java alapú JSP (JavaServer Pages), és a nyílt forrású PHP (Hypertext Preprocessor). Természetesen vannak még más, különféle CGI (Common Gateway Interface, protokollszabvány HTTP szerverekhez) nyelvek, de ezeket ritkán használják, mivel a fejlesztés velük sokszor körülményes, nehézkes.

Ahogy nőtt az Internet és nőtt a célközönség is, úgy nőtt a dinamikus weboldalak és a webszolgáltatások száma is. Egyre népszerűbbek lettek az internetes vásárlást lehetővé tevő webáruházak, elektronikus kereskedelemmel foglalkozó honlapok. Mára ezt nevezzük a Web 1.0 korszakának, ami lényegében a csak olvasható információk kora is volt egyben. A honlapot meglátogató személy megnézhette a honlapot, kereshetett számára lényeges információk halmazában, de a tartalomhoz nem adhatott hozzá, nem szólhatott bele az adatfolyamba. Ezek még most is élnek, és valószínűleg ez még sokáig így is marad, hiszen az emberek szeretik a kényelmet, azt, hogy kiszolgálják az igényeiket, például a különféle webáruházaknál.

Azonban a fejlődés nem állt meg ezen a ponton. Egyre nagyobb népszerűségnek örvendenek azok a lapok, amik a kollektív tudás tárházát kívánják megalkotni, a szabad véleménynyilvánítást szolgálják, és mindezt elérhetővé teszik mások számára is, ingyen és bérmentve vagy akár némi plusz pénzért még többet is kínálnak. Ide tartoznak többek között a WikiPedia, a különféle blogok és hírportálok, vagy az online közösségi oldalak is, mint az iWiW vagy a myVIP. Elsődleges elv az olvasd, szólj hozzá, alakítsd és formáld. Ezt a korszakot nevezik többen a WEB 2.0 világának, ahol már az egyén az uralkodó, és ennek köszönhető a webes hirdetések megjelenése is.

Ezekhez már nem volt elegendő a szerveroldali támogatás, jelentős fejlesztés kellett a kliensoldalon is, főleg a böngészők terén. A nagy böngészőháború idején (ami 1995-től egészen 2000-ig tartott) egyre másra jelentek meg a legkülönfélébb technikai támogatások, hogy a programok maguk mellé édesgessék az egyszeri nethasználót. Ebből a harcból végül az Internet

Explorer került ki győztesen és ez a jelentős fejlesztések végét is jelentette néhány éven keresztül. Aztán 2002-ben és 2003-ban új harcos tűnt fel a színen: a Firefox és a Macintosh gépekre szánt Safari. Ezek a böngészők könnyű kezelhetőségüknek és gyors működésüknek hála gyorsan elterjedtek és egy új fejlesztési hullámot indítottak, aminek mára már megéreztük a hatását. Ennek köszönhetően váltak egyre népszerűbbé a napló egy új formáját képező blogok és az információpiac új szereplői is, az RSS hírforrások. Mára egy új korszak vette kezdetét és a harc újraéledt a böngészőpiacon, szinte havonta jelennek meg a legkülönfélébb frissítések a programokhoz, hibajavításokkal és új szolgáltatásokkal. A webfejlesztőknek pedig ezt folyamatosan szemmel kell tartani, nem elegendő csak Internet Explorer felhasználóira gondolni.

A WEB 2.0 azonban nem csak ennyiből áll. Míg a hagyományos honlapoknál megszokott volt a várakozás egy kattintás után, mára ezen a stratégián jelentősen változtattak, és a honlapok valóban interaktívvá és gyorssá váltak. Ennek hátterében a szerver- és kliens oldal szorosabb összekapcsolása áll, az ún. AJAX (Asynchronous JavaScript and XML) technológia.

És hogy hol a jövő? Már kezd formálódni a WEB 3.0 is, ami mögött egy interneten használható operációs rendszer és a gondolkodó web bújik meg. Jelszó a net mindenhol, mindenkinek. Gondoljunk a hazánkba is már beköszönő IPTV-re, a digitális TV adásokra és arra, hogy egyáltalán nem ritkák azok az okostelefonok sem, melyekről elérhetjük a világhálót. Ma még sokat kell keresnünk ahhoz, hogy megtaláljuk például a menetrendet, vagy szállást foglaljunk egy hotelban egy éjszakára. A következő generáció talán már megéli azt, mikor mindezt automatikusan elvégzi az internet maga, nekünk pedig elegendő lesz megadni pusztán azt, amit szeretnénk elérni. Semmi sincs kizárva.

Mostanság nap mint nap weblapok tucatjai készülnek és kerülnek az internetre. Legyen az egy blog-bejegyzés, egy új cikk vagy hír, vagy akár egy egyszerű fórumhozzászólás, ha nem is látszik mindig egyértelműen, az esetek többségében ezek egy erősen kontrollált rendszer segítségével látnak napvilágot. Sokszor fel sem tűnik, hogy egy ilyen rendszert használunk: szinte már természetesnek vesszük ezek meglétét, hogy egy regisztráció során nem adhatunk üres vagy már korábban bejegyzett felhasználónevet magunknak, egy terméknek akciós ára van, vagy éppen egy új hír létrehozásakor egy barátságos felhasználói felület fogad bennünket.

Ezek kialakítása sokszor napokat, heteket, hónapokat igényel, pedig gyakran csak ismétlődő feladatok végrehajtása a cél, legyen szó bármilyen weboldalról. A legtöbb esetben szükség van egy ún. adminisztrációs felület létrehozására is, melyen keresztül a megrendelő (vagy felhasználó) kezelni tudja az oldal tartalmát. Számos ilyen rendszer létezik, melyek ezt a kettős célt szolgálják: közös nevük Tartalomkezelő Rendszerek (CMS, Content Management System). PHP esetében ilyen a Joomla, a Drupal vagy a korábban népszerű PHPNuke. Nagy előnyük, hogy könnyen beállíthatóak, gyorsan használhatóak, viszont a bővítés gyakran sok fejtörést okoz a fejlesztőknek, a megbízható dokumentációk ellenére.

Ezen szakdolgozat célja egy olyan keretrendszer kialakításának leírása és azon alapelvek megfogalmazása, aminek a segítségével könnyen elvégezhetőek a fentebb említett feladatok, illetve egy jó alapot biztosít tetszőleges kiegészítő megírására, így a gyors bővítésre. A fejezetek

során végigvesszük azokat a legfontosabb szempontokat, melyek egy ilyen rendszer kialakítása során felmerülhetnek, illetve átbeszéljük azokat az elemeket, amik a tényleges keret megalkotásához szükségesek. Szó lesz a PHP eszközeinek felhasználásáról, az egységes szempontú fejlesztésről, a HTML és CSS közös használatáról, valamint a Javascript előnyeiről és megismerhetjük a Web 2.0 egy kis szeletét is.

A Web 2.0 és ami hozzá kapcsolódik

1

Alapozás

A PHP esetében – és persze az összes többi nyelv esetében – rendkívül fontos megérteni annak működését, a technológia előnyeit és hátrányait a hatékony programfejlesztéshez. A PHP egy gyengén típusos, szerveroldali szkriptnyelv, melyet dinamikus weboldalak előállítására terveztek és mára az egyik legelterjettebb ingyenes kiszolgáló-oldali nyelvek egyike lett. Gyakorlatilag minden platformra elérhető, könnyen kezelhető és karbantartható. Legfontosabb tulajdonsága, hogy nem igényel fordítást: a leírt kódot a webszerver egy interpreter segítségével értelmezi és futtatja. A válasz a legtöbb esetben egy HTML kimenet, amit a böngésző értelmez és jelenít meg a kliensoldalon, így gyakorlatilag a forráskód nem kerül ki a szerverről.

A PHP-ból ma két verziót használnak szélesebb körben, az egyik a régebbi Zend Engine-re épülő 4-es és 2004-ben kiadott, továbbfejlesztett Zend Engine II-re épülő 5-ös széria. A szakdolgozat az utóbbival fog foglalkozni, mivel a 4-es verzióban már meglévő objektum orientált eszközök meglehetősen gyenge lábakon állnak. Az 5-ös verzióban jelentős fejlesztések történtek ezen a téren és egy komplett OO rendszert tudunk a segítségével használni. Természetesen ebből is több változat létezik, a legnagyobb különbséget az 5.2 és az 5.3-as között tapasztalhatjuk: utóbbiba ugyanis bekerült a névterek fogalma. Mi az 5.2-es verzióval fogunk foglalkozni, ugyanis ez nagyrészt kompatibilis a korábbiakkal és jelenleg jobban elterjedt, mint az 5.3-as. Fejlesztés alatt áll a 6-os változat, de egyelőre erről nem sok információ van. A PHP által szolgáltatott eszközökről a következő fejezetben lesz részletesebben szó.

A legtöbb esetben a PHP önmagában nem elegendő egy komplett weboldal összeállításához (vannak persze kivételek, de ez ritka), szükség van más egyéb eszközök igénybevételére is. A kiszolgáló szoftver megléte alapvető, hiszen e nélkül a leendő oldalt sem lehetne elérni. Mivel a nyelv platformfüggetlen, értelemszerűen létezik az összes nagyobb kiszolgáló alá valamilyen verzió, így az Apache és az IIS alá egyaránt. PHP-hoz az Apache lényegesen jobb választás, gyorsabban és megbízhatóbban működik, mint a Microsoft terméke, illetve lényegesen könnyebb a használata is. A másik említendő szoftver, amire még szükség lehet, egy adatbáziskezelő alkalmazás. Bár a PHP valamilyen szinten támogatja a Microsoft Office-szal érkező Access-t is, nem érdemes erőt pazarolni annak beüzemelésére. Szerencsére beépített támogatást kapunk az összes nagyobb adatbázishoz, így az Oracle, az MSSQL, a PostgreSQL és a MySQL sem marad

ki a sorból. Leggyakrabban az utóbbi kettőt használják PHP-val, mi az utolsót fogjuk preferálni, annak is az 5-ös verzióját.

A MySQL 5 már elég jól kiforrott, megbízhatóan működik, bár korántsem olyan robosztus alkalmazás, mint az Oracle megoldása. Lehetőség van tranzakciók kezelésére az InnoDB táblaformátum használatával, tárolt eljárások és eseménykezelők (vagy más néven triggerek) létrehozására is. Sajnos az auditálási funkciók később kerülnek csak bele, egy új verzióban az Oracle-nek köszönhetően. Az adatbáziskapcsolat kialakításáról és annak objektum-orientált kezeléséről a 3-ik fejezetben lesz szó.

Ezen programok mellett több kiegészítő engedélyezésére lehet még szükség, például a képek kezeléséhez a gd2 könyvtárra, a különféle biztonsági funkciókhoz és titkosításhoz az mcrypt library, a webszolgáltatások használatához a soap vagy a mások weboldalainak korrektebb kezeléséhez a curl bővítmény. Ezeket szerencsére a PHP beállításai között engedélyezni-tiltani lehet, így az esetek többségében nem igényel újrafordítást. Amennyiben azt szeretnénk, hogy a leendő weblapunk keresőbarát legyen, és a hivatkozások „szépek” legyenek, úgy a webszerveren engedélyezni kell az Apache beállításai között a lokális .htaccess fájlok használatát is.

A kliens oldal

A szerveren szükséges alkalmazások már fent vannak és működnek, de még hiányoznak olyan eszközök, amik segítségével a kliens oldalon is egy jól használható weblap készül – hiszen a végén úgyis erre kell majd a legtöbbet odafigyelni. A jól használható felület mögött egy könnyen használható Javascript keretrendszert kell érteni, megfelelő HTML és CSS támogatással. A Javascript mára szerves részévé vált a weboldalaknak, segítségével kiegészítők nélkül tudunk használni a böngészőkben a Flash-hez hasonló funkciókat, például mozgás, Drag&Drop, áttűnés vagy dinamikus ellenőrzés. Ezen effektek egyszerű kezeléséhez azonban nélkülözhetetlen egy jó Javascript keretrendszer használat. Több ilyen könyvtár is létezik, ezek közül a két legszélesebb körben használt a PrototypeJS és a jQuery. Mi az utóbbit fogjuk jobban megvizsgálni részletesebben és a weblapokhoz felhasználni. Segítségével egyszerűbbé válik az eseménykezelés, az animáció és nem utolsó sorban az Ajax használata is. Rengeteg kiegészítő létezik hozzá, amivel az amúgy is széles funkcionalitást tovább bővíthetjük és legnagyobb előnye, hogy független a böngészőtől (értsd: az egyes funkciók elérése és használata nem függ a kliens oldalon alkalmazott internet-böngészőtől, azonban előfordulhatnak kisebb hibák). A HTML-ről, CSS-ről és a jQuery-ről az 5-ik fejezetben lesz részletesen szó.

Fejlesztőeszközök

Ezek után már csak egy egyszerű szövegszerkesztőre lenne csak szükség, mint a Windows-ban is megtalálható notepad (vagy magyar nevén jegyzettömb). Az egyszeri fejlesztő azonban jobban jár egy valódi keretprogram használatával, amiben a szükséges segédeszközök megtalálhatóak. A legegyszerűbb ilyen program a jegyzettömb egy továbbfejlesztett változata, ami notepad2 névre hallgat (ebből nőtt ki később a notepad++ alkalmazás). Ez többnyire csak a szintaxis-kiemelést segíti PHP-ban, Javascript-ben, HTML-ben és a CSS-ben, de ennél jóval fejlettebb keretprogramok is vannak. A NetBeans és az Eclipse neve ismerősen csenghet azoknak, akik korábban dolgoztak Java-val. Mindkét program jó támogatással bír PHP téren, de a jobb választás a kettő közül inkább a NetBeans – az Eclipse fejlesztése sajnos rendkívül lassú, és a Zend által kiadott Zend Studio for Eclipse sem elégíti ki maradéktalanul a gyors működés feltételeit. A NetBeans-ben lehetőségünk van használni a beépített függvényeket és függvénykönyvtárakat a code hinting funkció segítségével, és a projektek kezelése sem elhanyagolható szempont.

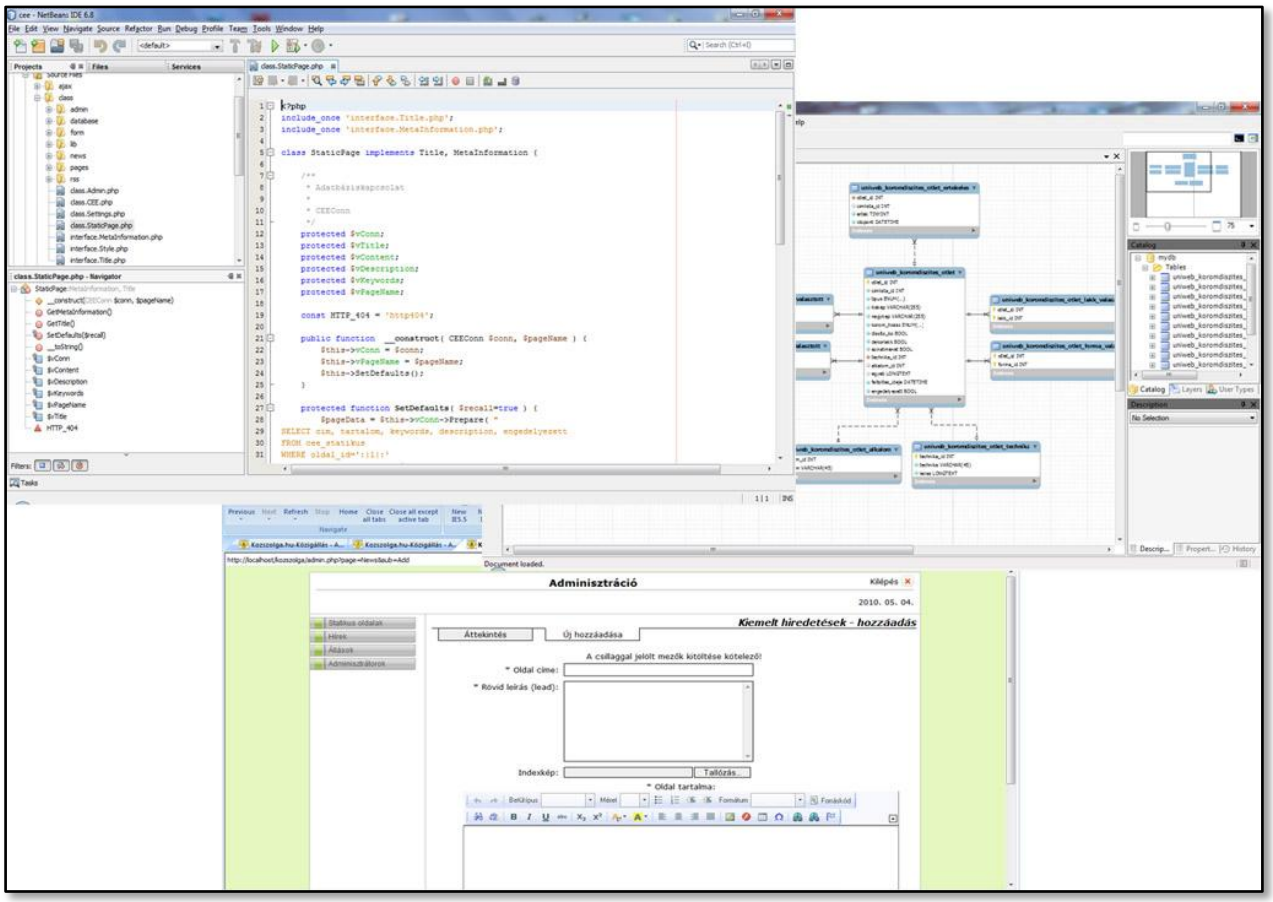
Több esetben szükség van arra is, hogy az adatbázist és az abban foglalt összefüggéseket láthassuk. Erre nyújt megoldást a MySQL fejlesztői által létrehozott ingyenesen használható segédeszköz, a MySQL Workbench. Bár létezik fizetős verziója is, amivel már azonnal változtathatjuk magát az adatbázist is, az ingyenes verzió legalább ekkora segítséget nyújt az adatbázis megtervezése során és a későbbi fejlesztések alatt egyaránt.

Természetesen szükséges a leendő weboldalunk legszélesebb körű tesztelése is, legfőbbképpen a böngészőket tekintve: érdemes tehát feltelepíteni a leendő fejlesztő gépére az Internet Explorert, a Firefoxot, a Google Chrome-ot, az Apple Safari-t és az Opera nevű böngészőt is. Mivel az Internet Explorer-ből ahány verzió létezik, annyiféle módon jeleníti meg a weblapot és sajnos a gépen egyszerre csak egyféle futhat belőle, érdemes feltenni egy ingyenes segédeszközt, amivel a jelenleg még forgalomban lévő verziókat egyszerre kipróbálhatjuk. A program neve: IETester. Segítségével megnézhetjük a honlap megjelenését, illetve kipróbálhatjuk a különféle Javascript funkciók viselkedését is.

További két hasznos segédeszköz kapcsolódik egy másik szélesebb körben elterjedt böngészőhöz, a Firefox-hoz. Ezek egyike a WebDeveloper Toolbar, amivel például azonnal törölhetjük az oldalhoz kapcsolódó Cookie-kat, letilthatjuk a Javascriptet, átállíthatjuk a formok elemeinek értékeit, és még számos hasznos további funkció. A másik igen hasznos szerszám a FireBug. Ez az egyik legteljesebb Javascript és HTML tesztelő programcsomag, amivel figyelemmel kísérhetjük a DOM-ban bekövetkező változásokat, a kliens és szerver között adatforgalmat, illetve az esetleges funkcionális hibákra is hamarabb fény derül a segítségével. Két további hasznos kiegészítő a HTMLValidator és a MeasureIt nevű plugin. Előbbivel a weboldal HTML kódját tudjuk könnyen a szabványnak megfelelő formára hozni, míg a második egy mérőszalag szerepét tölti be, így az egyes elemek méreteinek vagy pozíciójának meghatározása könnyebbé válik.

Összefoglalás

Ebben a fejezetben sorra vettük a fejlesztéshez szükséges alapelemeket és természetesen a fejlesztőeszközök sem maradtak ki a sorból. A következő fejezetben megismerkedünk a PHP 5 objektum-rendszerével, majd belekezdünk a rendszer egy részének kialakításába is.



1.1 ábra – Fejlesztőeszközök egy helyen

2

Web 2.0

A WEB 2.0 fogalmáról már korábban volt szó, így most annak történetét, kialakulását, a jelenlegi szolgáltatásokat és az ezt használó lehetőségeket fogjuk jobban megvizsgálni. Megnézzük, mik azok a technológiák, amik már valóban az új kor találmányai és melyek azok, melyek ránk maradtak, de mégis fontos szerepet játszanak egy-egy weblap történetében.

A Dotcom-válság

Az előző évtized végére úgy tűnt, az online cégek fellendülése megállíthatatlan. Egyre-másra készültek az egymásra nagyon hasonlító, különféle webáruházak és internetes vállalatok tömege lepte el a webet. Szinte aranybányának számított ezzel foglalkozni és egy jó ötlettel befektetőket keresni. A befektetők pedig évente több milliárd dollárt költöttek ebbe az iparágba csak Amerikában. A 21. század azonban egy komoly válsággal indított, amibe számos cég beleroppant – a részvények értéke semmivé foszlott, az alkalmazottak nem kaptak fizetést, magyarán csődbe mentek.

2000-ben tehát kidurrant a lufi, az új világ ifjú tőzsdései, akik elszórták a milliókat, rövid úton eltűntek a süllyesztőben. (Ironikus módon hasonló folyamat játszódik most le a közösségi oldalak felvásárlása terén is). De miért is történt mindez? Talán ennek magyarázata, hogy bár a weboldalak megalkotói ismerték a technológiát, a honlapok létrehozásának mikéntjét, az üzlethez már nem értettek és végül ez okozta a vesztüket. De közrejátszott az is, hogy nem kérték ki a felhasználók véleményét, nem voltak kíváncsiak a vásárlók reakcióira, csak a statisztikákat figyelték, hogy egyes termékek jobban fogynak.

Új korszak

2002-re aztán a legtöbb cég, mely túlélte a dotcom-lufi kidurranását, stratégiát váltott és más megoldásokat keresett a vásárlók elhódítása végett. Felismerték, hogy érdemes kikérni a vásárlók véleményét és azokat valamilyen formában közzé is tehetik – megjelentek a fórumok és a

különféle chat- és beszélgető alkalmazások, mint például az MSN. Ezek még mai is fontos részét képezik a webes alkalmazásoknak, gyakorlatilag minden honlapon megtalálhatóak erre a legkülönfélébb eszközök. Az egyik legérdekesebb internetes irányvonal azonban mindenképpen a keresés lett. Hogyan találjuk meg a számunkra szükséges információkat, anélkül, hogy minden egyes weblapot végig kellene néznünk, vagy csak éppen az aktuális oldalon szeretnénk megtalálni valamit, mert másvalaki már beszélt erről?

A választ talán legjobban egy kis kaliforniai cégnek sikerült megadnia, az akkor piacon lévő egyik legnagyobb keresőcégnek, a Yahoo-nak mintegy ellentmondva. Ez a cég a Google volt, és gondolom nem kell magyarázni, hogy a taktika bevált; az internetezők 70%-a ezt a weboldalt látogatja meg, ha valamit keres vagy kíváncsi valamire. Nem szabad azonban elfelejteni, hogy a piac is változik, és bár a Google lassan, de veszít egyeduralmából, (köszönhetően többek között a Microsoft-féle Bing-nek), erősen tartja magát.

És ha már a vélemény-nyilvánításnál tartunk, nem szabad elfelejteni, hogy az emberek szeretnek egymással beszélgetni, vagy információkat megosztani – erre mintegy válaszul megjelentek a legkülönfélébb közösségi oldalak, ahol ez a megosztás lett az elsődleges cél. Ilyen többek között a Facebook, a MySpace, az Iwiw vagy MyVIP, hogy csak az ismertebbeket említsük. Ezek a weboldalak manapság szintén nagyon gyorsan terjednek, gyakran már csak egymás funkcióit másolják, új ötletek nélkül. A reklámcégek viszont nagy fantáziát látnak bennük, talán nem is véletlenül: az ilyen oldalak látogatottsága hatalmas, az itt elhelyezett hirdetések szinte biztosan célba találnak.

A vélemény-kifejezés tehát a webre is kiterjeszkedett, és ennek egy még érdekesebb bizonyítéka a blogok megjelenése. A blogok lényegében a korábbi személyes naplók internetes megfelelői, amiben saját gondolatainkat, ötleteinket tehetjük közszemlére, véleményezésre, vagy akár saját magunknak is megtarthatjuk, ez mindenkinek a saját egyéni hozzáállásán és döntésén múlik. (Ne felejtsük el azonban, hogy a mai keresőmotorok szinte mindenre rátalálnak, így a blogunk sem maradhat sokáig érintetlen, hacsak nem gondoskodunk az ellenkezőjéről).

Szintén bekerültek a köztudatba az internet vásárláshoz kapcsolódó egyéb szolgáltatások, mint az árverési oldalak (pl. Vatera.hu), vagy a hazánkban még nem annyira széles körben elterjedt fizetési megoldások, pl. a PayPal. Ezen szolgáltatások viszonylag könnyen igénybe vehetőek, többnyire csak egy egyszerű regisztrációs lapot kell kitölteni. Nem szabad elfelejteni a weben fellelhető multimédiás tartalmat sem: ma már szinte mindennapos a videók interneten történő megtekintése, vagy az online vásárlás egy új fajtájáról, a mikrotranzakciókról. Legyen szó a Youtube-ról, vagy az iTunes-ról, mindenki tudja, mit mire használnak. És ezek már nem csak az internet részei: a multimédiás telefonok terjedésével a szolgáltatások száma egyre növekszik.

Ha pedig már szolgáltatásokról van szó, ne feledkezzünk meg az egyik szintén új keletű eszközről, ami globális szinten változtatott az informatika világán: ez nem más, mint a SOAP, a webszolgáltatások alapja. A webszolgáltatások segítségével a különféle rendszerek közti határok eltűnnek, legyen szó szoftveres, vagy akár hardveres különbségekről, az információcsere egy meghatározó és egységes formája lett. A távoli eljáráshívás most már mindössze néhány sor

megírását jelenti, miközben nem kell a tényleges művelet mikéntjével foglalkoznunk – az objektumorientáltság előnye.

Az információközlés másik formája, mellyel a weboldalak egymás között cserélhetnek híreket – és ez azért egy jóval elterjedtebb forma – az RSS-ek használata. Az RSS segítségével szintén egy egységes hírközlési felületet kapunk, amivel nem csak mi jeleníthetjük meg a fontosabb híreket, akár kategóriákra bontva, hanem mások csatornáit is olvashatjuk és persze az engedélyükkel meg is jeleníthetjük saját weboldalunkon. Jó példa erre a hírportálok (News Aggregator, hírgyűjtő) létezése, ahol a legkülönfélébb kategóriák sorolják be a híreket, különféle kulcsszavak alapján, teljesen automatikusan.

Az előző két módszer közös tulajdonsága, hogy alapjuk az XML (Extentensible Markup Language), aminek több szempontból is nagy jelentősége van. Felhasználási módjuk gyakorlatilag végtelen, bármire szabványos formában alkalmazhatóak, és épp ezért újrafelhasználhatóak, aminek az előnyeit gondolom nem kell sorolni.

A technológiák alkalmazása

Az eddig megismert módszereket mi magunk is tetszés szerint felhasználhatjuk saját weboldalunkon, és ezt a rendszerbe integrálhatjuk is. A legelső dolog, amit megnézünk, az RSS lesz, hiszen egy weboldalon a hírek publikálásának egy meglehetősen elterjedt módja ez. Egy mai weboldalon általánosan megjelennek a weblappal, céggel, programmal kapcsolatos friss információk, ezek mintegy általános megjelenítése módja az RSS. A portál-motorba tehát mindenképpen érdemes egy hír modult építeni az ilyen információk kezelésére. A modul legyen általános és bővíthető, hiszen bár a legtöbb hír mindössze egyetlen oldalt tesz ki, előfordulhatnak olyan cikkek, melyek terjedelme már lapozást tesz szükségessé, vagy olyanok, melyek több nyelven is megjelenhetnek (I18N).

A hírek, cikkek általánosan megjelenő tulajdonságai:

- Cím
- Bevezető szöveg, rövid leírás (lead, description)
- Tartalom
- Publikálás ideje
- Cikkhez kapcsolódó kép

Ezen információk nagy része a weboldalon belül jelenik meg, főleg a tartalmi részt és a képet tekintve. Az RSS-nek a lehető legtömörebben kell a figyelem felkeltését szolgálni, hiszen minél több hírt publikálunk, minél nagyobb tartalommal, a csatorna mérete is ezzel fog folyamatosan változni. Érdemes tehát limitálni a csatornába kerülő hírek számát valamilyen formában, aminek általában két módja van:

- Fixált mennyiség
- Adott napnál rövidebb időn belül publikálásra kerülő hírek

A fixált mennyiség nem mindig a legjobb eredményt hozza, de többnyire megfelel az igényeknek, mindenesetre a modulnak tartalmaznia kell a szükséges beállítási lehetőségek között ezt is.

Egy RSS csatornába a cikk tulajdonságai közül a cím, a leader szöveg, a publikálás ideje, valamint a hírhez kapcsolódó URL cím az, ami bekerül. Az RSS-nek több fajtája is elterjedt, a ma használt kettő ezek közül az Atom Feed és a Really Simple Syndication; mi az utóbbit fogjuk használni annak egyszerűbb kezelhetősége miatt, de a kettő lényegében némi eltéréssel bár, de kompatibilis. Egy RSS csatorna az alábbi mintát követi:

```
<rss version="2.0" xmlns:atom="http://www.w3.org/2005/Atom">
<channel>
<atom:link href="http://www.weblapom.com/rss" rel="self"
type="application/rss+xml"/>
    <title>Weboldalam legfrissebb hírei</title>
    <description>A legfrissebb hírek, újdonságok</description>
    <link>http://www.weblapom.com/</link>
    <item>
        <title><![CDATA[Első hír]]></title>
        <link>http://www.weblapom.com/news/1/Elso_hir.html</link>
        <description><![CDATA[Első hír rövid leírása, a leglényegesebb
információk összefoglalására]]></description>
        <guid>http://www.weblapom.com/news/1/Elso_hir.html</guid>
        <pubDate>Wed, 28 Apr 2010 12:00:00 +0100</pubDate>
        <category>Hírek</category>
    </item>
    <item>
        <title><![CDATA[Második hír]]></title>
        <link>http://www.weblapom.com/news/2/Masodik_hir.html</link>
        <description><![CDATA[Második hír rövid leírása, a leglényegesebb
információk összefoglalására]]></description>
        <guid>http://www.weblapom.com/news/2/Masodik_hir.html</guid>
        <pubDate>Wed, 28 Apr 2010 11:30:26 +0100</pubDate>
        <category>Hírek</category>
    </item>
…
</channel>
</rss>
```

2.1 kód – RSS Feed

A fenti mintából több dolog is feltűnhet. Az első, hogy az egyes címek és leírások nem közvetlenül az elemben találhatóak, hanem egy speciális, CDATA elemben foglalnak helyet. Ennek oka, hogy gyakran előfordul, hogy maga a leírás is tartalmaz formázási elemeket, vagy nem szabályos XML elemeket (pl. az „&” a HTML-ben teljesen szabályosan az „&” jelöli, az XML-ben azonban nem megengedett a használata).

A következő, ami szembeötlik a megvizsgálás után, az a publikálás dátumának formája, amire nagyon vigyázni kell. A formátum a szabvány szerint az RFC 822-nek kell megfeleljen, ennek

formázására szerencsére mind a PHP, mind a MySQL tartalmaz eszközöket. A felhasználásnál óvatosan kell bánni ezekkel az eszközökkel, mert könnyen átcsúszhat az érvénytelen feed kategóriába oldalunk.

Az egyes cikkek értelemszerűen az ITEM tagekbe csoportosulnak, azoknak a címe, elérhetősége (URL), leader szövege, valamint megjelenési ideje a mérvadó, ahogy azt már korábban mondtam, de feltüntethetjük a cikk szerzőjét (author), a kategóriát, vagy akár a kapcsolódó képet is (ezt az enclosure elemmel tehetjük meg, az esetek többségében azonban ennek használata nem szükséges).

PHP oldalon az RSS általános megjelenítését végző osztály:

```
<?php
abstract class RSSFeed {
    public function __construct(){
        $this->Conn = MySQLConnection::GetInstance();
    }
    public abstract function GetNewsList();
    public function __toString(){
        $tpl = new Smarty();
        $tpl->assign( "news", $this->GetNewsList() );
        return $tpl->fetch( " rss/rss.tpl" );
    }
?>
```

2.2 kód – RSS alaposztály

Értelemszerűen absztrakt osztályról van szó, hiszen más lekérdezés szükséges a hírekhez, illetve a termékekhez – persze van mód az összedolgozásukra is, de célszerűbb ezeket szétválasztani.

Közösségi oldalak

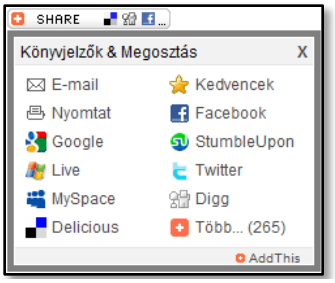
A különféle közösségi oldalak mára már fogalmakká váltak, mondhatni a mindennapi élet része, hogy felnézek a Facebook-ra és meglesem, mi történt az ismerősökkel a tegnapi bulin. Vagy megnézek egy vicces videót a Youtube-on, esetleg a kettőt összekötöm. De miért is ne közölhetném mindenkivel, hogy találtam egy jó cikket a weben? Szerencsére a legtöbb ilyen oldal ún. nyílt API-val rendelkezik, amik más oldalak számára is lehetővé teszi cikkek portolását a rendszerbe, de nem ritka, hogy teljes alkalmazást is hozzáadhatnak külsős fejlesztők. A weblapunkhoz is szükség lehet tehát egy osztályra, ami ezeket a megosztásokat automatikusan elvégzi, mindössze egy kattintásra rövidítve a felhasználók számára a korábban szükséges műveletsorokat.

```php
<?php
class SocialShare {
    public static $SHARE_URLS = array(
        'facebook'=>"http://www.facebook.com/share.php?u=<URL>",
        'twitter'=>"http://twitter.com/home?status=<TITLE> <URL>",
        'iwiw'=>"http://iwiw.hu/pages/share/share.jsp?u=<URL>&t=<TITLE>",
        'google'=>"http://www.google.com/bookmarks/mark?op=edit&bkmk=<URL>&titl
e=<TITLE>&annotation=<ANNOTATION>"
    );

    public function __construct( $url, $title="", $annotation="" ){
        $this->vUrl = $url;
        $this->vTitle = $title ? $title : self::DEAFAULT_TITLE;
        $this->vAnnotation = $annotation ? $annotation :
self::DEFAULT_ANNOTATION;
    }
    public function GetShareURLs() {
        $result = array();
        foreach ( self::$SHARE_URLS as $key=>$value ) {
            //helyettesítések: URL, TITLE, ANNOTATION
        }
        return $result;
    }
    public function __toString(){
        $tpl = new Smarty();
        $tpl->assign( "urls", $this->GetShareURLs() );
        return $tpl->fetch( " social/share.tpl" );
    }
?>
```

2.3 kód – Megosztás közösségi oldalakon

A fenti példa PHP alapú, de ennek Javascript megvalósítása azért jobb, mert az a munka, amit a rendszer az egyes helyettesítések során elvégez, a kliens oldalra is rábízható, ezzel is tehermentesítve a webszervert. Ennek egy ilyetén megoldása több helyen előfordul, ingyenesen felhasználható és könnyen beágyazható, ezen felül lényegesen több hálózatot is támogat (persze a fenti is bővíthető, csak a legnépszerűbb elemek szerepelnek rajta).



2.1 ábra – Megosztási ablak

Webszolgáltatások

A weboldalak együttműködése, egymás funkcióinak meghívása igen kényes művelet és nem egyszer problémát tudnak okozni a programozási nyelvek között tapasztalható, sokszor elég jelentős eltérések. Gondoljunk csak a Java és a PHP esetére; a Java erősen OO nyelv, míg a PHP-ban csak eszközök vannak az ilyen viselkedés programozására, ezen felül míg a Java típusos

nyelv, a PHP már csak gyengén típusos, de még sorolhatnánk. Mi lehet tehát a megoldás két weboldal, két szolgáltatás együttműködésében?

A kérdésre a SOAP (Simple Object Access Protocol) adhat választ. Segítségével XML alapokra helyezhetjük a kommunikációt, így mintegy egységbe kovácsolva a korábbi eltéréseket, áthidalva a problémákat. A webszolgáltatások azonban ennél jóval összetettebbek, a SOAP csak egy részét képezi azoknak, bár az egyik leglényegesebb építőelemről van szó. Egy webszolgáltatás létrehozására a legtöbb nyelvben létezik eszköz, amivel magát a szolgáltatást leíró WSDL (Web Service Description Language) fájlt nem kell külön megalkotnunk, azt a programozási környezet elvégzi helyettünk (például a Java és az ASP.NET esetén is, a PHP-ban azonban megfelelő felület hiányában ez sajnos nem elérhető). A webszolgáltatások működése és létrehozása nem célja a szakdolgozatnak, a téma ennél jóval több figyelmet érdemel és bőven áll megfelelő szakirodalom is rendelkezésre, akár a weben is.

A PHP esetén egy webszolgáltatás felhasználására beépített eszköz létezik, melynek neve PHP_SOAP. Szintén külső kiegészítő, és bár része az alapcsomagnak, általában nem aktív, engedélyezni-engedélyeztetni kell annak használatát. Egy szolgáltatás igénybevételéhez számunkra a SoapClient osztály jelenti a kulcsot, amely automatikusan értelmezi az adott szolgáltatást leíró WSDL fájlt. Egy adott szolgáltatás lekérésekor a válasz szintén XML formátumban érkezik, amit a SoapClient szintén egy előfeldolgozás után bocsájt a rendelkezésünkre, így lehetővé téve, hogy a választ objektumként kezelhessük.

A keretrendszerben általában ennek használatára nincs közvetlenül szükség, azonban előfordulhat, hogy egy másik weboldalról kell bizonyos időközönként információkat lekérdezni, ilyenkor pedig jól jön ennek az ismerete. Jó példa erre, ha a portál részét képező webáruházban a termékek árát csak forintban tároljuk (másképp nem is érdemes idehaza), azonban a nyilvános oldalon szükség van az Euróban, vagy más valutában meghatározott értékre is. A Magyar Nemzeti Bank webszolgáltatásként teszi elérhetővé a különféle valutainformációkat, amiket innen lekérdezve már meg tudjuk jeleníteni a korrekt árakat bármely tetszőleges pénznemben.

Összefoglalás

A fejezetben megismerkedtünk a Web 2.0 kialakulásának történetével, a jelenleg elérhető szolgáltatásokat és eszközöket is feltérképeztük. Láttunk példát arra, miként alkalmazhatjuk ezeket saját weboldalunkon, hogyan jelenítsünk meg egy RSS feed-et, illetve hogyan tegyük mások számára is elérhető egy-egy weboldal tartalmát könnyen és gyorsan a közösségi csatornákon. A Web 2.0 szolgáltatásainak felhasználása ma már egy weblap egyik lételeme, ezek nélkül valószínűleg jóval kevesebb látogató találnak rá az új honlapokra.

3

Objektumorientált PHP 5

Az objektumorientált programozás alapjait ismerjük – objektumok mindenek felett, mindezt az újrafelhasználhatóság jegyében. A legtöbb nyelvben számos eszköz áll rendelkezésünkre egy jól strukturált osztályhierarchia kialakításához: osztályok, interfészek, öröklés és számos más, az objektumok világából ismert fogalom és persze programnyelvbe épített osztályok.

A PHP egy egyszeres öröklési elvet valló, típus nélküli nyelv, az 5-ös változattal pedig már egy valóban komoly eszközrendszert használhatunk fel programjaink elkészítéséhez. Viszont ez a nyelv jelentősen különbözik ezen a téren a többi, valóban OO nyelvtől (mint például a Java vagy a C#). Elsőként talán a beépített osztályok hiánya tűnik fel legjobban (ami csak részben igaz), de elég sokszor hátrányt is jelent a PHP egyik előnye: a típusnélküliség. Óvatosan kell bánni a számokkal és szövegekkel, de a logikai kifejezések is rejthetnek csapdát. Erre persze a legjobb módszer a megszokás, a dokumentálás és az óvatosság persze, de a későbbiek folyamán láthatunk majd néhány olyan eszközt is, amely segítségünkre lehet a későbbi hibák előfordulásának csökkentésében.

Ebben a fejezetben megismerkedünk a PHP 5 OO részének alapjaival, majd sorra kerül a kivételkezelés és néhány apró trükk a könnyebben átlátható kódok készítésének érdekében.

Az OO alapjai PHP-ban

Hasonlóan más nyelvekhez, az objektumok alapját azok vázának elkészítése, vagyis az osztályok megírása jelenti itt is. Egy osztályt a CLASS kulcsszóval, majd az osztály nevével tudunk definiálni. A 4-es változatban alapvetően függvények jelentették a metódusokat, melyek közül egy kitüntetett szereppel bírt. Ez az osztály nevével egyező nevet viselte, és más nyelvekhez hasonlóan ez volt az osztály konstruktora. Az 5-ben azonban változott egy kicsit a helyzet ezen a téren. Bár a kompatibilitás miatt megmaradt a konstruktor támogatásának ezen formája, a fejlesztők beépítettek egy új függvényt, mellyel megkönnyíthetjük saját munkánkat: ez a

__construct() metódus. Ha mindkettő szerepel egy osztályban, úgy az 5 az utóbbit fogja választani.

A korábbi változatokhoz képest az 5-ös PHP már rendelkezik egy bezárási eszközrendszerrel is, amit a már más nyelvekben jól megszokott publikus (public), védett (protected) és privát (private) szavakkal vehetünk igénybe. Ezeket használhatjuk mind adattagok, mind pedig metódusok esetén. További fontos változás, hogy már létrehozhatunk osztályszintű adattagokat és metódusokat (aminek a lehetősége bár a PHP 4-ben is megvolt, az objektum és osztályszintű elemek megkülönböztetése nem volt egyértelmű). Utóbbi azért is lényeges, mert így lényegesen könnyebbé válik az egyke minta megvalósítása. Újabb kulcsszó a final, amivel nem felüldefiniálható függvényeket hozhatunk létre egy osztályon belül.

A PHP nem támogatja a polimorfizmust, azaz nincs lehetőségünk egy függvény több azonos nevű változatát elkészíteni egy osztályon belül. Ezt megkerülhetjük a paraméterek opcionálissá tételével a kezdőértékek megadásával, ezek azonban csak konkrét értékek lehetnek, változó vagy objektum nem. Lényeges újítás a type-hinting, azaz a metódusok paramétereit tipizálhatjuk, ezáltal kényszerítve a helyes paraméterezést. Sajnos itt is limitáltak az eszközök, típusként csak tömböt vagy osztály nevezhetünk meg, primitív adattagokat nem. Ha nem a megfelelő elemet adjuk paraméternek, egy kivétel keletkezik, amit try-catch blokkal kezelhetünk.

A Java-hoz hasonlóan PHP esetén is az extends kulcsszóval tudunk leszármazott osztályokat készíteni. Ilyenkor fontos megjegyezni, hogy a leszármazott osztályt tartalmazó fájlban el kell helyezni egy include utasítást (vagy require, de javasolt a require_once vagy az include_once használata) a szülő osztályra, ennek hiánya fatális hibát okoz, amennyiben a szülőosztály nem került korábban behívásra. A szülő osztály eszközeire a parent minősítéssel hivatkozhatunk, melyet két kettőspont, majd a szükséges eszköz neve követ. Saját eszközöket (vagy a szülő osztály eszközeit) a $this kulcsszót követő -> (nyíl)-lal hívhatunk meg. Ez vonatkozik mind a metódusokra, mind pedig az adattagokra. Ha statikus eszközt szeretnénk igénybe venni, úgy azt a self kulcsszóval és az azt követő két kettősponttal tehetjük meg, amennyiben az az objektum saját osztályában található. Ha másik osztály statikus eszközére van szükségünk, úgy a self-et helyettesítsük a kívánt osztály nevével. A 3.1-es kód egy egyszerű osztályszerkezetet mutat meg.

```
<?php
class Felhasznalo {
      protected $nev;
      protected $emailCim;
      protected $belepve;

      public function __construct( $nev, $emailCim, $belepve=false ) {
            $this->nev = $nev;
            $this->emailCim = $emailCim;
            $this->belepve = $belepve;
      }

      public function belep() {
```

```
            if ( $this->ellenorzes() )
                  $this->belepve = true;
            else $this->belepve = false;
      }

      protected function ellenorzes() {
            return preg_match( '/^\w+[\+\.\w-]*@([\w-]+\.)*\w+[\w-]*\.([a-
z]{2,4}|\d+)$/i', $this->emailCim );
      }
}
?>
```

3.1 kód – Osztály megvalósítása PHP-ben

Ezen rövid bevezető után lássuk azokat a speciális eszközöket, melyek jelentősen könnyítenek a fejlesztésben: az ún. mágikus függvényeket. Ide tartoznak azok a különleges metódusok, amik két aláhúzás jellel kezdődnek, mint az előbb említett konstruktor is. Segítségükkel lehetővé válik az objektum-szerializáció testre szabása, az automatikus osztály-importálás, adattagok egyszerű lekérdezése és beállítása vagy a Java-ban is nagyon hasznos string-gé alakítás. Az alábbi oldalakon ezeket fogjuk jobban szemügyre venni néhány példával tarkítva.

Mágikus függvények

__destruct

A konstruktorokról már volt szó, így essen most szó a kevésbé emlegetett, de legalább annyira hasznos destruktorról. A destruktor akkor jut szerephez, mikor az objektum jelentősége elvész, azaz a szemétgyűjtögető úgymond halálra ítéli. Ha nem történik több hivatkozás az objektumra, törlődik a memóriából a jobb helykihasználás érdekében, ilyenkor automatikus meghívódik a __destruct függvény. Ebben olyan dolgokat helyezhetünk el, amit korábban nem tudtunk megtenni, mert az adott elemre még szükség volt, de a gyorsabb futás érdekében jó lenne azt lezárni. Ilyen például egy fájl megnyitása, vagy éppen egy adatbázis-kapcsolat. Bár többnyire ezeket a PHP automatikusan elvégzi helyettünk, ha mi nem tettük meg, a gyorsabb futás miatt szükség lehet rá.

```
<?php
class Destructor {
     protected $nev;
     public  function __construct( $nev ) {
     $this->nev=$nev;
     echo "Új objektum: ".$this->nev."<br>";
     }
     public function __destruct() {
          echo "Objektum vége: ".$this->nev."<br>";
     }
```

```
}
$elso = new Destructor( "első" );
$masodik = new Destructor( "második" );
// kimenet:
// Új objektum: első
// Új objektum: második
// Objektum vége: első
// Objektum vége: második
?>
```

3.2 kód - Destruktor

__get, __set, __isset, __unset

Ezek a függvények a túlterhelést szolgálják. Ha egy objektum adattagját módosítjuk, úgy automatikusan meghívódik a __set metódus, vagy ha lekérdezünk egy adattagot, akkor a __get. Ezek akkor jöhetnek jól, ha egy nem nyilvános (publikus) adattaghoz szeretnénk hozzáférni, vagy olyan elemre hivatkozunk, ami az adott osztályban nincs deklarálva. Segítségükkel elvégezhető egy minimális ellenőrzés, például hogy az új érték megfelel-e egy adott mintának (regexp). Az __isset és __unset explicit módon meghívódik, ha az isset() és unset() függvényeket használjuk egy elem adattagján, így például ha egy adatbázis-kapcsolatot akarunk kilőni az unsettel, úgy az objektum __unset() metódusába beírhatjuk a megfelelő eljárást szabályosan lezárva a kapcsolatot.

__call

Ez a függvény különösen akkor hasznos, hogyha az objektum egy olyan függvényére hivatkozunk, amit nem implementáltunk semmilyen formában, magyarán nem létezik. Ilyenkor lehetőségünk van valamilyen értéket visszaadni, vagy éppen egy már kezelhető kivételt dobni egy végzetes hiba helyett.

__sleep, __wakeup

Az objektum-szerializációhoz szükséges függvények. A szerializáció akkor jön képbe, hogyha egy objektumot szeretnénk letárolni, például a munkamenetben. Alapesetben ha meghívjuk a beépített serialize függvényt (vagy implicit módon hivatkozunk rá), az az objektum minden adattagját elmenti és gyakran ez elég sok felesleges információ-átadással jár és nem is feltétlenül biztonságos. Például ha egy osztályban adattagként használtunk egy adatbázis-kapcsolatot (valamilyen formában), akkor az is elmentésre kerül, többnyire a kapcsolat-kiépítéshez szükséges felhasználónévvel és jelszóval együtt. Ez bár hasznos is lehet, itt elég nagy biztonsági kockázatot jelent. Ekkor siet a segítségünkre a sleep, amiben megmondhatjuk, hogy milyen adattagokat mentsen el a szerializáció, melyekre van feltétlen szükség az állapot visszanyeréséhez. A wakeup ennek a fordítottját végzi, itt lehetőségünk van az el nem mentett adattagok visszaállítására szabályos függvényhívásokon keresztül. Ez a függvény akkor kerül meghívásra, ha az unserialize függvényt hívjuk explicit vagy implicit módon.

__toString

Talán az egyik leghasznosabb mágikus függvény. Segítségével meghatározhatjuk az objektum kimenetét, az echo és/vagy print segítségével kiírathatjuk. Nagyon jól jön, mikor a weblapot

szeretnénk láthatóvá tenni, de addig még nem használtuk a kiíratást, például mert szükség volt egy fejrész-kommunikációra (header utasításokra). Ennek segítségével egyszerűsödik a felület és könnyebben használható osztályszerkezetet is kapunk. A későbbiek során sokat fogjuk használni.

```
<?php
session_start();
class Sleepy {
  protected $nev,$szamossag,$conn;
  public function __construct( $nev, $szamossag ) {
    $this->nev=$nev; $this->szamossag=$szamossag;
    $this->conn = new MySQLConnection( "username", "password", "localhost",
"database" );
  }
  public function __sleep() { return array( 'nev', 'szamossag' ); }
  public function __wakeup() { $this->conn = new MySQLConnection( "username",
"password", "localhost", "database" ); }
  public function __toString() { return $this->nev.": ".$this->szamossag."
db<br>"; }
}
$elem1 = new Sleepy( "termék1", "5" );
$elem2 = new Sleepy( "termek2", "10" );
$_SESSION["kosar"]=array( serialize($elem1), serialize($elem2) );
print_r( $_SESSION );
echo unserialize( $_SESSION["kosar"][0] );
echo unserialize( $_SESSION["kosar"][1] );
// kimenet:
// Array
// (
//     [kosar] => Array  (
//             [0] =>
O:6:"Sleepy":2:{s:6:"?*?nev";s:7:"termék1";s:12:"?*?szamossag";s:1:"5";}
//             [1] =>
// O:6:"Sleepy":2:{s:6:"?*?nev";s:7:"termek2";s:12:"?*?szamossag";s:2:"10";}
//         ))
// <br>termék1: 5 db<br><br>termek2: 10 db<br>
?>
```

3.3 kód – Mágikus függvények

Az itt ismertetett függvények nagy hasznunkra válhatnak a fejlesztés folyamán és feltétlen érdemes őket használni. Lényegesen megkönnyítik és jelentősen gyorsítják is a programozás folyamatát, mivel egységes eszközrendszert bocsátanak rendelkezésünkre, így a részrendszerek újrafelhasználhatóvá és áttekinthetővé válnak, ezek előnyei pedig nyilvánvalóak.

Kivételkezelés

További újítás a PHP 5-ben a kivételek bevezetése, segítségükkel könnyebben kezelhetővé válnak a különböző hibák. A weboldal elkészítése során két lényeges momentumnál használtam fel a kivételkezelés nyújtotta előnyöket. Ezek közül az egyik a felhasználói bejelentkezés. Ha a látogató véletlenül vagy szándékosan olyan oldalra érkezik, amihez eléréséhez előbb be kellene jelentkezzen, úgy azt egy kivétel kiváltásával jelezhetjük saját magunk és a program számára. A kezelés történhet úgy, hogy egy másik oldalra irányítjuk át, vagy egyszerűen hibaüzenetet adunk, hogy jelentkezzen be. Ezt láthatjuk is, hogyha bejelentkezés nélkül a hirdetésfeladásra kattintunk.

A másik, már nem annyira szembetűnő dolog a tranzakció-kezelés. Az adatbázist burkoló egyik osztály, amit az osztály-diagrammoknál láthattunk, kivételt dob, ha egy SQL-lekérdezést nem sikerült végrehajtani. Így például ha a felhasználói regisztrációnál vagy a hirdetések feladásánál nem tudtuk beszúrni az adattáblákba a szükséges sorokat, mivel a kérdéses elsődleges kulcs már létezett, vagy esetleg az adattábla zárolva volt, kivétel váltódik ki. Ha ezt lekezeljük, a tranzakciók segítségével visszagörgethetjük (rollback művelet) az adatbázist egy korábbi, valószínűleg biztonságos adatokat tartalmazó változatához anélkül, hogy komolyabb baj történt volna és a felhasználót is tájékoztathatjuk, hogy a műveletek végrehajtása problémamentesen sikerült-e vagy sem.

Ugyanakkor remek lehetőséget nyújt a rendszer naplózására is, hiszen ilyenkor láthatjuk, hogy mi okozott hibát a működés során és ezeket javíthatjuk is a fejlesztések folyamán. Természetesen nem univerzális eszközök ezek sem, mert egy-egy le nem kezelt kivétel bizony elég sok mindent elárulhat idegeneknek a rendszer működéséről. Szerencsére a PHP beépítve tartalmaz egy olyan függvényt, amivel globális kivételkezelőt állíthatunk be: ez a függvény pedig a set_exception_handler. További részletes információk erről a PHP kézikönyvében találhatók, a http://www.php.net weboldalon.

Összefoglalás

A fejlesztés folyamán az egyik legnagyobb előnyt az jelenti, ha kellőképpen járatosak vagyunk az adott nyelv szintaxisában és ismerjük a különleges eszközöket, jellemzőket. Ebben a fejezetben röviden megismerkedtünk a PHP objektum-orientált eszközrendszerével és megismerhettünk néhány igen hasznos trükköt egy újrafelhasználható program írásához. Ezek közé tartoztak a PHP mágikus függvényei és a hibakezelés egy új módja is.

4

Adatbázisok kezelése

A webes fejlesztés folyamán elengedhetetlen megfelelő adatbázisok használata a felhasználóktól származó vagy a tartalomkezelőhöz szükséges adatok tárolására, feldolgozására. Ahogy azt az első fejezetben is említettük, mi a MySQL adatbázis-kezelőt fogjuk előnyben részesíteni, így ebben a fejezetben erről lesz bővebben szó, főként annak az osztályrendszernek a használatáról, amivel PHP alatt azt kezelni tudjuk.

A PHP 5-ben egy új bővítmény áll rendelkezésünkre az adatbázisok gyors kezeléséhez. A bővítmény neve PDO (PHP Data Objects), a PHP alapértelmezés szerint tartalmazza a könyvtárat. Az egyes adatbázisokhoz viszont további bővítmények szükségesek a csatolófelület biztosítása érdekében. Használata viszonylag egyszerű, további előnye a gyorsaság; ez a lefordított függvénykönyvtáraknak köszönhető. Hátránya viszont, hogy nem ad megfelelő hibakezelést a fejlesztő kezébe, így a kivételek keletkezése nem az elvárt módon történik. Egy hibás lekérdezés mindössze egy üres elemet ad vissza, nem vált ki kivételt, így az üres objektum – amiről azt feltételezhetnénk, hogy egy objektum – függvényeinek hívása fatális hibát vált ki. Ezekre az apróságokra nagyon oda kell figyelni a fejlesztés során, egy nem ellenőrzött részlet komoly gondokat okozhat az alkalmazás működése folyamán.

```
<?php
try {
    $conn = new PDO( 'mysql:dbname='.Settings::DB_DATABASE.
                     ';host='.SETTINGS::DB_HOSTNAME,
                     Settings::DB_USERNAME, Settings::DB_PASSWORD );
    $sth = $conn->prepare( "SELECT * FROM users WHERE user_id=:user_id",
           array(PDO::ATTR_CURSOR => PDO::CURSOR_FWDONLY) );
    $sth->execute( array('user_id'=>15) );
    $user15 = $sth->fetchAll();
}
catch ( PDOException $e ) {
    echo "A lekérdezés végrehajtása során hiba történt: ".$e->getMessage();
}
?>
```

4.1 kód – Beépített adatbázis-kezelés PDO-val

A fenti példán látható, hogy az adatbázis beállítása viszonylag egyszerűen zajlik, és a beállítások között 4 változó definiálását igényli. Ezek rendre az adatbázis nevét, az adatbázis-kiszolgáló címét, valamint a csatlakozáshoz szükséges felhasználói nevet és jelszót tartalmazzák. A lekérdezés egy ún. előkészítésen megy keresztül, amivel a lekérdezéseket később fel tudjuk paraméterezni és így újrafelhasználhatóvá válnak. Az ilyen lekérdezések nagy előnnyel járnak, hiszen memóriát és erőforrásokat spórolunk meg, amennyiben egy lekérdezést többször kell egymás után lefuttatni, ahol csak a paraméterek különböznek.

A biztonság itt is megjelenik, hiszen az így átadott, ún. binding paramétereket a PDO bővítmény automatikusan ellátja a szükséges kiegészítésekkel, így az alkalmazás védve lesz a SQL befecskendezésnek (SQL Injection) nevezett támadás elől. Ha fenti lekérdezés nem paraméterezve lett volna végrehajtva, hanem a PDO execute függvényének egy natív stringet adtunk volna át, úgy, hogy a paramétert nem ellenőrizzük és szűrjük, bizony komoly problémával szembesülhetünk:

```
<?php
try {
    $conn = new PDO( 'mysql:dbname='.Settings::DB_DATABASE.
                     ';host='.SETTINGS::DB_HOSTNAME,
                     Settings::DB_USERNAME, Settings::DB_PASSWORD );
$user_id = "' OR true OR '15";
    $sth = $conn->exec( "SELECT * FROM users WHERE user_id='$user_id'" );
    $user15 = $sth->fetchAll();
}
catch ( PDOException $e ) {
    echo "A lekérdezés végrehajtása során hiba történt: ".$e->getMessage();
}
?>
```

4.2 kód – SQL-injection

A fenti lekérdezés az értelmezést követően az összes felhasználó adataival tér vissza, mivel a lekérdezésben szereplő WHERE záradék mindig teljesül. Ha a paraméter valamilyen request elemből származik ($_POST, $_GET, stb), akkor gyakorlatilag az alkalmazás támadhatóvá válik. A PHP automatikusan szűri ezeket az értékeket, amennyiben a beállítások között a magic_quotes értéke igaz. Ez gyakorlatilag az addslashes függvénnyel egyenértékű, azonban ennek bevezetése több problémát okozott, mint amennyit megoldott. A függvény alkalmazása tökéletesen megfelel, amennyiben MySQL-t használunk adatbázis gyanánt, viszont ha már PLSQL-t vagy Oracle-t használunk, úgy a függvény nem a várt eredményt fogja hozni. Helyette egyébként is javasolt a mysql_real_escape_string alkalmazása a mi esetünkben, ugyanis ez garantáltan úgy alakítja át a paramétert, ami már nem fog gondot okozni.

A PDO előnyei nyilvánvalóak ezek után, azonban nem szabad elfelejteni, hogy a PDO egy bővítmény, aminek betöltését engedélyezni kell. A legtöbb ingyenes szolgáltató azonban nem használja, vagy határozottan tiltja a használatát, így más megoldást vagyunk kénytelenek keresni, amennyiben egy ilyen helyen szeretnénk működtetni a weboldalt.

Követelmények

Az adatbázis kezeléséhez tehát szükségünk van egy osztályrendszerre, amivel hasonlóan jó funkcionalitást érhetünk el, mint a PDO-val. Azonban a bővítménnyel szemben itt le kell mondanunk a fordítás okozta előnyökről, tehát a sebesség egy kicsit kisebb lesz. A főbb funkciók, amik megvalósítást igényelnek a teljesség igénye nélkül:

- Tranzakció kezdése, lezárása, visszagörgetése
- Egyszerű lekérdezés
- Paraméterezett lekérdezés (binding, prepared statements)
- A lekérdezés eredményének visszaadása, egyet vagy akár az összest, az utolsó beszúrt elem azonosítójának lekérdezése (auto increment mező esetén)
- Az érintett és lekérdezett sorok számának lekérdezése
- Megfelelő hibakezelés lehetősége, kivételek dobása hibás lekérdezések vagy az adatbázis-kapcsolatban keletkező hiba esetére
- Központi beállítások
- Igény szerinti vagy perzisztens kapcsolat kialakítása

Az utóbb kérdés ami igazán fejtörést okozhat. Mi a különbség az igény szerinti és a perzisztens kapcsolat között? Az igény szerinti kapcsolódás minden esetben új szál indítását eredményezi, csak újrageneráljuk az oldalt. Jól megírt lekérdezések esetén az adatbázis ilyenkor van kitéve a legnagyobb terhelésnek. A perzisztens kapcsolat viszont a session időtartama alatt fennáll, azaz új kapcsolat kialakítására csak új session megkezdése esetén van szükség, ezáltal gyorsul a rendszer. Ilyenkor viszont a már befejezett, de még aktív session-ök (azaz a szemétgyűjtögető még törölte a session-t) memória-gondokat okozhatnak. Megfontolás kérdése, de a perzisztens kialakítás több előnnyel jár, mint hátránnyal.

A rendszer kialakítása követni fogja az egyke mintát, hiszen egy oldalon az esetek nagyon nagy többségében nincs szükség több adatbázis kapcsolatra egynél. Ebben segítségünkre lesz a PHP 5 OO rendszere, ahol már lehetséges privát konstruktorok létrehozása. Az egyke minta (singleton) olyan esetekben alkalmazható, ahol egy osztálynak csak egy példányára van szükség. Ezáltal könnyebben elérhetővé válik minden más osztály számára és nem foglal feleslegesen memóriát sem.

```
<?php
class MySQLConnection {

    protected $Hostname;
    protected $Username;
    protected $Password;
    protected $Database;
    private static $Instance;

    private function __construct( $host, $username, $password, $dbase ) {
```

```
        $this->Hostname = $host;
        $this->Username = $username;
        $this->Password = $password;
        $this->Database = $dbase;
    }
    public static final function GetInstance() {
        if ( !(self::$Instance) ) {
            self::$Instance = new MySQLConnection(
                 Settings::DB_HOSTNAME,
                 Settings::DB_USERNAME,
                 Settings::DB_PASSWORD,
                 Settings::DB_DATABASE );
        }
        return self::$Instance;
    }
}
?>
```

4.3 kód – Saját adatbázis-osztály az egyke mintával

A fenti kódban megtaláljuk ugyanazokat a paramétereket, amit a PDO esetén is át kellett adni, tehát a Settings osztály paramétereit. Ezzel sikerült elérni, hogy az osztály központosítva legyen, a beállításoktól függjön. Fontos azonban, hogy ez így nem feltétlenül a legjobb megoldás; előfordulhat, hogy az adminisztrációs felület más beállításokat igényel, mint a mindenki számára hozzáférhető rész, hiszen az adminisztrátornak általában lényegesen több joga van és ezt érdemes az adatbázisban is rögzíteni, a házirend beállításával. ilyenkor természetesen a GetInstance függvény változni fog, ugyanazokat a paramétereket kell átadni neki, mint a konstruktornak kellene. Azonban ilyenkor fel kell figyelni a kódburjánzás jelenségére is, felesleges kódsorok kerülhetnek a programba. Minél több ilyen van, annál nagyobb a hibázás lehetősége is. A megoldás erre a problémára viszonylag egyszerű: a függvénynek egyetlen paramétert adjunk át, ami egy logikai változó. Amennyiben igaz, adminisztrációs jogokkal kell csatlakozni, ellenkező esetben hagyományos módon. Ez két további mező felvételét igényli a Settings osztályba.

```
<?php
class MySQLConnection {
    public static final function GetInstance($isAdmin=false) {
        if ( !(self::$Instance) ) {
            self::$Instance = new MySQLConnection(
                 Settings::DB_HOSTNAME,
                 $isAdmin ? Settings::DB_ADMIN_USERNAME : Settings::DB_USERNAME,
                 $isAdmin ? Settings::DB_ADMIN_PASSWORD : Settings::DB_PASSWORD,
                 Settings::DB_DATABASE );
        }
        return self::$Instance;
    }
}
?>
```

4.4 kód – Adatbázis kapcsolat admin felhasználóval

Az adatbázis neve és a kiszolgáló megegyezik mindkét esetben, hiszen egyazon adatbázist kell kezelni, mindössze a felhasználói név és a jelszó különbözik. A megfelelő házirend kialakítása azért is hasznos, mert így elkerülhető, hogy a felhasználói oldalról olyan adatokhoz férjenek hozzá vagy módosítsanak véletlenül vagy szándékosan, amikhez csak az adminisztrátornak lehetne hozzáférése. Ilyen esemény például egy termék vagy felhasználó törlése.

Miután a hozzáférést biztosítottuk, vegyük sorra a többi funkciót. Ezek közül a legelső a tényleges kapcsolat kialakítása, valamint a lekérdezés végrehajtása. A kapcsolat felépítésére csak akkor van szükség, ha lekérdezést szeretnénk végrehajtani, egyébként nem érdemes ezzel terhelni a kiszolgálót. A kapcsolódás szükséges kód tehát az alábbi lehet:

```
<?php
class MySQLConnection {
…
    protected $Conn;
    protected function Connect() {
        $this->Conn = mysql_pconnect(
             $this->Hostname, $this->Username, $this->Password );
        if ( !is_resource( $this->Conn ) ) {
            throw new MySQLException( "Connection failed" );
        }
        else {
            mysql_query( "SET NAMES ".Settings::CHARSET );
            mysql_query( "SET CHARACTER SET ".Settings::CHARSET );
        }
        if ( !mysql_select_db( $this->Database, $this->Conn ) ) {
            throw new MySQLException( "Database cannot be reached" );
        }
    }
}
?>
```

4.5 kód - Kapcsolódás

Érdemes felfigyelni két tulajdonságra. Az egyik egy hivatkozás a karakterkódolás beállítására, a másik pedig egy új osztály, a MySQLException. Az elsőt nem érdemes tárgyalni, miért is szükséges, elegendő, ha az UTF-8 és Latin karakterkészletek közötti különbségekre gondolunk. A kivétel osztály megléte szintén szükséges; ezzel jelezhetjük a feldolgozó rutinoknak, ha az adatbázist nem sikerül elérni, egy lekérdezés végrehajtása nem sikerül, vagy a tranzakció műveletei során hiba keletkezik, és a műveleteket vissza kell görgetni. A hibaosztály:

```
<?php
class MySQLException extends Exception {
    public function __construct( $message ) {
        parent::__construct( $message.": ".mysql_error(), mysql_errno() );
    }
}
?>
```

4.6 kód – Adatbázis-kivétel osztálya

Érdemes megfigyelni, hogy a hibát magát nem adjuk át a konstruktornak, hanem az magának teszi hozzá, így nem történik szükségtelen memóriafoglalás és erőforrásokat takarítunk meg.

Egy alap lekérdezés végrehajtása az alábbi függvény segítségével történhet:

```
<?php
class MySQLConnection {
…
    protected function Exec($query) {
        if (!$this->Conn) {
            $this->Connect();
        }
        $result = mysql_query( $query, $this->Conn );
        if ( !$result ) {
            throw new MySQLException( "Query execution failed" );
        }
        $stmt = new MySQLStatement( $this->Conn, $query );
        $stmt->Result = $result;
        return $stmt;
    }
}
?>
```

4.7 kód – Egyszerű lekérdezést végrehajtó függvény

A függvényben a hibakezelésen túl (a lekérdezés nem sikerült, mivel vagy hibás, vagy a kapcsolat szakadt meg) egy új osztály is megjelenik: a MySQLStatement. Ezzel az osztállyal fogjuk kezelni a lekérdezés eredményét, azaz az érintett sorokat, a bejegyzéseket. Az előkészített lekérdezéseket szintén ez az osztály fogja végrehajtani. Az ehhez a függvényhez szükséges kód:

```
<?php
class MySQLConnection {
…
    public function Prepare($query) {
        if (!$this->Conn) {
            $this->Connect();
        }
        return new MySQLStatement( $this->Conn, $query );
    }
}
?>
```

4.8 kód – Lekérdezés előkészítése

A fenti függvény nem hajtja végre az utasítást, pusztán egy új objektumot hoz létre. Az eredmény az előbb említett MySQLStatement lesz, amiben a következő funkciók lesznek megtalálhatóak:

- Lekérdezések eredményének bejárása
- Lekérdezés eredményének összes sorának visszaadása tömbként
- Lekérdezett sorok száma, módosított sorok száma
- Előkészített lekérdezése paraméterezése, végrehajtása

```
<?php
class MySQLStatement {
    protected $Conn;
    protected $Query;
    protected $FetchMode = "mysql_fetch_assoc";
    protected $FetchObject;
    protected $FetchParams;
    public Bindings = array();

    public function __construct($conn,$query) {
        $this->Conn = $conn;
        $this->Query = $query;
        $this->Bindings = array();
    }
    public function SetFetchMode( $mode, $class="", array $params=array() ) {
        if ( $mode==MySQLConnection::FETCH_ASSOC ||
             $mode==MySQLConnection::FETCH_NUM ) {
            $this->FetchMode = $mode;
        }
        elseif ( $mode==MySQLConnection::FETCH_OBJECT ) {
            $this->FetchMode = $mode;
            $this->FetchParams = $params;
            if ( !$class_name || !class_exists( $class_name ) ) {
                throw new MySQLException( "Class $class_name not found" );
            }
        }
        else {
            throw new Exception( "Fetch mode must be a FETCH_* constant!" );
        }
    }
}
?>
```

4.9 kód – MySQLStatement osztály egy részlete

A konstruktor mellett található másik függvény segítségével azt lehet beállítani, hogy a lekérdezés feldolgozása milyen módon történjen. A PHP beépített MySQL funkciói között három mérvadó megoldással találkozunk, az egyszerű tömb, az asszociatív tömb és az objektum. Az egyszerű tömb a lekérdezés adott sorának oszlopait tömbszerűen adja vissza, az asszociatív mód ehhez hasonló, de ilyenkor az oszlopok nevei a tömb kulcsaival egyeznek, míg az objektum-mód értelemszerűen új objektumot hoz létre, melynek megfelelő adattagjai az egyes oszlopok. Ehhez természetesen a MySQLConnection osztályban definiálni kell a szükséges konstansokat.

A lekérdezés paraméterezése már nehezebb feladat. Háromféleképpen lehet ezt megtenni: külső függvényhívással (AddBind), a Bindings paraméter módosításával, illetve a végrehajtáshoz használt függvény paraméterezésével. A legjobb megoldást természetesen Az AddBind segítségével lehet elérni, melynek kódja:

```
<?php
class MySQLStatement {
    public Bindings = array();
    …
    public function AddBind($key,$value,$type="",$length=0) {
        if ( $type ) {
            //típus beállítása settype segítségével, vagy egyéb típus esetén
            //manuálisan, MySQLConnection TYPE_* konstansok szerint.
        }
        if ( $type==MySQLConnection::TYPE_STRING && $length>0 ) {
            $value = mb_substr( $value, 0, $length, Settings:CHARSET );
        }
        $this->Bindings[$key] = $value;
    }
}
?>
```

4.10 kód – Binding

A függvény alapértelmezés szerint két paramétert vár, a változó nevét ($key) és értékét ($value). A lekérdezésben így kell hivatkoznunk majd rá. Két további beállítást is meg lehet tenni, miszerint az értéket és a mező hosszát lehet állítani, szükség szerint. A hossz stringek esetén az adott string hosszát jelenti, egyébként nincs hatással a paraméterre. Az egyes típusokat érdemes konstansként deklarálni a MySQLConnection osztályban (lásd fentebb például TYPE_STRING).

A végrehajtást végző függvény szintén kaphat paramétereket, természetesen limitált lehetőségekkel (csak kulcs-érték párok adhatóak meg).

```
class MySQLStatement {
    …
    public function Execute(array $bindings=array()) {
        $this->Bindings = array_merge( $this->Bindings, $bindings );
        foreach ( $this->Bindings as $key=>$value ) {
            $query = str_replace( ":|".$key."|:",
                  mysql_real_escape_string($value), $query );
        }
        $this->Result = mysql_query( $query );
        if ( !$this->Result ) {
            throw new MySQLException( "Query execution failed" );
        }
        return $this;
    }
}
?>
```

4.11 kód – Előkészített lekérdezés végrehajtása

Az utolsó lényeges momentum, amire figyelni kell egy ilyen osztály kialakításánál, az az eredmények kezelése. A lekérdezések legtöbbször SELECT típusúak, de természetesen a DELETE, UPDATE, INSERT utasításokat sem szabad elfelejteni. A SELECT eredményeinek

feldolgozása szorosan kapcsolódik a korábban tárgyalt fetch mode-hoz, hiszen az osztály az egyes eredményeket eszerint fogja visszaadni. A kód a következő:

```
class MySQLStatement {
    …
    public function Fetch() {
        if ( !$this->Result ) {
            throw new MySQLException( "Query failed, not executed" );
        }
        if ( $this->FetchMode==MySQLConnection::FETCH_OBJECT ) {
            $row = call_user_function( $this->GetFetchFunction(), array(
                $this->Result,
                $this->FetchObject,
                $this->FetchParams
            ) );
        }
        else {
            $row = call_user_function( $this->GetFetchFunction(), array(
                $this->Result
            ) );
        }
        return $row;
    }
    public function FetchAll() {
        $result = array();
        while ( $row = $this->Fetch() ) {
            $result[] = $row;
        }
        return $result;
    }
    protected function GetFetchFunction(){
        switch ( $this->FetchMode ) {
            case MySQLConnection::FETCH_OBJECT: return "mysql_fetch_object";
            case MySQLConnection::FETCH_ASSOC: return "mysql_fetch_assoc";
            case MySQLConnection::FETCH_NUM: return "mysql_fetch_row";
            default: throw new MySQLException( "Function not found ".
                            $this->FetchMode );
        }
    }
}
?>
```

4.12 kód – Eredmény bejárása és összes eredmény

Az itt feltüntetett két függvény közül a Fetch végzi a tényleges munkát, a FetchAll pedig inkább egy hasznos kiegészítője az osztálynak, aminek segítségével az összes, a lekérdezésben szereplő sort visszakaphatjuk. Ez kifejezetten hasznos lehet, ha nem kell egyéb műveleteket elvégezni az adott sorokon és egy listát akarunk csak kiíratni. A GetFetchFunction értelemszerűen a FetchMode-tól függően adja vissza annak a függvénynek a nevét, amit meg kell hívni, nem létező eredmény esetén kivételt vált ki.

Az osztály további függvényeit nem kell sorra venni, hiszen azok javarészt a MySQL függvények elfedései, magyarázatot nem igényelnek. Egy példa az ilyen függvényekre:

```
class MySQLStatement {
    …
    public function RowCount() {
        return is_resource()?mysql_num_rows( $this->Result
):mysql_affected_rows( $this->Conn );
    }
    public function LastInsertId() {
        return mysql_insert_id();
    }
    public function GetVariable( $varname, $rownum=0 ) {
        return mysql_result( $this->Result, $varname, $rownum );
    }
    …
}
?>
```

4.13 kód – További függvények

Természetesen az egyes függvények elé egy ellenőrzés is berakható, hogy a lekérdezés valóban erőfforrás-e és végrehajthatóak-e a kérdéses függvények.

Összefoglalás

A fejezetben megismerkedtünk a PHP 5-ben megjelent PDO adatbáziskezelő osztállyal és megalkottunk egy saját rendszert annak helyettesítésére is. Átnéztük az adatbázisban tárolt adatatok védésének néhány módját, az ilyen irányú támadások elleni védekezéseket. Az általunk megalkotott osztályszerkezet a védekezést natívan támogatja, hathatós segítség az adatbázisok gyors és hatékony kezelésére, és ezen felül majdnem teljesen kompatibilis a PDO osztállyal is. Sajnos a futási hatékonyság a fordítás hiánya miatt nem olyan jó, viszont egységesebb rendszert követ a lekérdezések kezelését illetően. A következő fejezetben az adatbevitelről és az adatok kezeléséről lesz szó, melyhez felhasználjuk majd az itt megalkotott elemeket is.

5

A weblap tartalmának megjelenítése

A keretrendszer talán egyik legfontosabb és egyben legkényesebb része a weblap megjelenítése. Szét kell választani az adatok kezelését a weboldal tartalmától, ami egyáltalán nem könnyű feladat, mivel az adatokból függ megszületni a tényleges HTML. Az MVC (Model-View-Controller) minta pontosan ilyen feladatok miatt jött létre: válasszuk szét az adatkezelést, a megjelenítést és a vezérlést, azt ehhez kapcsolódó funkciót csoportosítsuk külön osztályokba, hiszen ezek lényeges, ugyanakkor eltérő feladatokat látnak el. Röviden összefoglalva:

- Modell - A rendszer azon belső része, amely a logika központi részét hajtja végre.
- Nézet - A rendszer valamennyi kimenetének formázásáról gondoskodó rész.
- Vezérlő - A bemenetet feldolgozó, és azt a modell felé közvetítő rész.

A PHP speciális ilyen tekintetben. A bemenet lényegében három, esetleg négy formában érkezhet meg a webszerver irányába, különféle request kérelmeken keresztül. Ezek ún. szuperglobális változókként jelennek meg, mindenhonnan elérhetőek. A feldolgozás, azaz a Controller szerepét tehát csak részben kell implementálni, a műveletek nagy részét elvégzi helyettünk a PHP. Az egyetlen kivételt itt a formok jelentik, ezeket később tárgyaljuk át egy külön fejezetben.

A feladat tehát már csak a megjelenítés és a központi logika elválasztása. A korai weblapok nem fordítottak erre nagy figyelmet, a PHP natívan íratta ki a generált HTML kódot, igencsak megnehezítve a fejlesztést és a HTTP fejlécek kezelését. Mivel gyakran szükség volt arra, hogy a fejlécet is módosítsuk, jelezve ezzel, hogy az oldal tartalma lejárt, frissítés szükséges, vagy akár csak a karakterkészletet akartuk közölni a böngészővel, az átláthatatlan echo és print utasítások miatt ez gyakran meghiúsult és hibákat okozott a működésben. Ugyanakkor elég sok fejtörést jelentett a fejlesztőknek, hogy egy design-tervből hogyan legyen tényleges weblap. Ez mára már nem a programozók dolga – az ilyen feladatok ellátására vannak szakemberek, akik ún. site-building-et végeznek, a tervekből HTML kódokat hoznak létre. A szétválasztás tehát ezen a ponton történhet meg, ennek az eszköze pedig a sablonrendszerek használata.

Sokan vallják azt, hogy felesleges egy sablonértelmező használata, hiszen a PHP maga a sablonrendszer. Ez a fentiek figyelembevételével kissé téves feltételezés. A rendszer működése ezek használatával ugyan lassulni fog, viszont a fejlesztési és a hibajavítási idő lényegesen

lecsökken, a programozási feladatok segítségükkel jól elkülöníthetőek lesznek a megjelenítési feladatoktól. Ebben a fejezetben ezekről a különálló rendszerekről lesz, hogyan tudjuk beépíteni azt a saját weboldalunkba könnyen és fájdalommentesen.

A sablonrendszer kiválasztása

A weben rengeteg sablonrendszer érhető el, vannak köztük egészen aprók, de nagyon összetett darabok is, mindenki megtalálhatja a saját szájízének megfelelőt. Az egyik legszélesebb körben elterjedt ezek közül a Smarty. A Smarty Template használata azért előnyösebb más, hasonló eszközöknél, mert a template-k értelmezése nem történik meg minden oldalhívásnál. Egy előfordítást használ, ami PHP parancsállományokat készít, és ezeket addig használja, amíg az adott fájlon nem módosítunk.

További nagy előnye a rengeteg funkció. Bár ezt sokan hátránynak is titulálják, hiszen egy új nyelv megtanulását igényli és segítségével új programokat is meg lehet alkotni – de mi maradjunk annál, amire használni szeretnénk. A csomag ingyenes letölthető a http://www.smarty.net weboldalról, a telepítés is egyszerűen zajlik. Sok mindent kapunk a program mellé, ezek nagy részére valószínűleg nem lesz szükségünk. Három fő könyvtárt feltétlenül ki kell emelni: a templates, a templates_c és configs. Ezek közül az első a saját sablonjainkat tartalmazza, a második a Smarty által lefordított elemeket, míg a configs a beállításokért felel. A kiszolgálónak olyan jogokkal kell rendelkeznie, hogy a templates_c mappába szabadon írhasson, hiszen a fordítást a Smarty végzi, és ide menti a programokat.

A template-knél egy nagyon fontos szabályt kell figyelembe venni: programkód ne kerüljön a sablonfájlokba, azok csak a megjelenítéshez szükséges információkat tartalmazzák. A Smarty esetén ennek betartása elég egyszerűnek tűnik, de könnyen kísértésbe lehet esni. További lényeges szempont, bár ez szintén csak formaság, a sablonok kiterjesztése legyen egységes, így könnyebb megkülönböztetni őket.

A Smarty használata

Szerencsére ezen eszköz megalkotóinak sikerült egy nagyon egyszerűen használható rendszer létrehozniuk, mind a programozók, mind a site-builderek részére. A Smarty objektumorientált elveket vall, így könnyen hozzá tudjuk illeszteni saját keretrendszerünkhöz. Az alábbi kód létrehoz egy új sablont és azokat megfelelően felparaméterezi.

```
<?php
$tpl = new Smarty();
$tpl->assign( 'nev', 'Mákos guba' );
$tpl->assign( 'hozzavalok', array( 'Liszt', 'Cukor', 'Só', … ) );
$tpl->display( 'index.tpl' );
?>
```

5.1 kód – Egyszerű Smarty példa

Ezek után az értelmező betölti a sablont, ha szükséges, újraértelmezi, és megjeleníti a kívánt oldalt. Ehhez azonban értelemszerűen szükség lesz a templates mappában az index.tpl fájlra, ami a HTML tartalmat hordozza.

```
<html>
<head>
  <title>{$nev} - hozzavalok</title>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-2">
</head>
<body>
<p>
  <h1>{$nev} - hozzávalók:</h1>
  <ul>
{foreach from=$hozzavalok item=hozzavalo}
    <li>{$hozzavalo}</li>
{/section}
  </ul>
</p>
</body>
</html>
```

5.2 kód – A template fájl

Ez csak egy nagyon egyszerű példa volt, a kimenet valami hasonló lesz:

Mákosguba – hozzávalók:

- Liszt
- Cukor
- Só

A Smarty azonban ennél jóval többre képes. A hasonlítási műveletek, az objektumkezelés és a gyorsítótár alkalmazása mind részét képezi a rendszernek. A nyelvezete kicsit nehézkes, azonban könnyen tanulható, többnyire csak megszokás kérdése. A fenti kód azonban nem teljes – a display helyett egy másik függvényt, a fetch függvényt kell használnunk majd a saját osztályainkban. Ennek magyarázata az osztályok felépítésében keresendő.

A rendszer osztályainak felépítése

Azt szeretnénk elérni, hogy a weblap kódja összefogott legyen, így a kérdéses részeket mindig ugyanott kell keressük. A megjelenítéshez szükséges függvény egységesen kell használni, hiszen innen válik igazán láthatóvá, hogy mit ír ki a weboldal. A függvény legyen a PHP által használt mágikus metódus, a __toString. Ennek két lényeges megkötése van használat közben: a visszatérési értéknek mindig string-nek kell lennie és nem dobhat kivételt. A kettő közül az utóbbi a súlyosabb; ha a metódusban kivétel keletkezik, az fatális hibát eredményez, míg ha a visszatérési érték nem string, úgy egy általunk kezelhető kivétel váltódik ki. Ezért ezt érdemes szem előtt tartani.

A weblap azon osztályai, melyek a kiíratást végzik, egységesen implementálják ezt a függvényt, így elkerülhetőek a későbbi félreértések. A függvény tehát stringként tér vissza, amit az echo függvénnyel jeleníthetünk meg. Erre azonban csak egy helyen lesz szükség, a kezdőlapon. Így a template-jeinket nem a display metódussal fogjuk megjelenítetni, hanem a fetch-el értelmeztetjük, és az így kapott eredményt adjuk vissza a __toString metódusban a saját osztályainkban. Tehát az előbbi példa OO környezetben:

```
<?php
class Index {
  public function __construct() {
  }
  public function __toString() {
    $tpl = new Smarty();
    $tpl->assign( 'nev', 'Mákos guba' );
    $tpl->assign( 'hozzavalok', array( 'Liszt', 'Cukor', 'Só', … ) );
    return $tpl->fetch( 'index.tpl' );
  }
}
?>
```

5.3 kód – Smarty a __toString metódussal

A tényleges megjelenítést végző php fájl tartalma pedig:

```
<?php
include_once 'class.Index.php';
$index = new Index();
echo $index->__toString();
?>
```

4.4 kód - Megjelenítés

Láthatóan az oldal mindössze néhány sorból áll, mégis megjeleníti a teljes weboldalt, anélkül, hogy a programozási oldalon HTML kód jelenne meg, vagyis elértük a célunkat, az alkalmazáslogika és megjelenítés szétválasztása a template-k használatával sikeresen megoldható és gyorsan használható.

Összefoglalás

A fejezetben megismerkedtünk az MVC mintával és annak előnyeivel, valamint lefektettük a keretrendszerünk talán egyik legfontosabb alappillérét is. Átbeszéltük a sablonrendszerek használatának szükségét és példát is láthattunk annak használatára a Smarty rendszer segítségével.

6

Megjelenés – HTML és Javascript

Egy weblap látogatói rendkívül érzékenyek annak megjelenésére és használhatóságára. Ha egy oldalt nem tudnak úgy használni, hogy az ne akadjon meg egy-egy ponton, vagy nem kapnak megfelelő tájékoztatást az egyes eseményekről, legyen szó bármilyen apróságról, az könnyen az oldal végét jelentheti. Ezért rendkívül fontos a jól átgondolt és szépen megtervezett felhasználói felület, de ugyanakkor arra is figyelni kell, hogy az oldal funkciói tökéletesen és gyorsan működjenek.

Forradalom

Korábban a weboldalak felépítése mögött egy olyan eszköz állt, ami garantáltan azt az eredményt hozta minden böngésző alatt, amit látni szerettünk volna. Mondhatnánk, az élet egyszerű volt. Ez az eszköz azonban nem felelt meg a Web 2.0-nek, mivel az oldalak felépítése nagyon összetett volt, a DOM (Document Object Model) manipulálása nagyon körülményes volt, és eredeti célja sem a weblapok szerkezeti felépítése lett volna. Ez az eszköz a táblázat volt. A fejlesztők aztán lassan ráeszméltek, hogy a táblázatos oldalépítés bár lényegesen gyorsabb, egy apró változtatás sokkal több módosítással jár, mint azt szerették volna. Ekkor fordultak a CSS eszközéhez. A CSS-ben egy HTML elem teljes mértékben testre szabható és pozícionálható, a szerkezethez tehát nincs szükség táblázatra.

Az oldalak szerkezete így lényegesen leegyszerűsödött, a böngészőkben sokkal gyorsabban jelent meg ilyen lap, kevesebb energiát vett el a felhasználó számítógépétől annak lerenderelése és kezelése. Ez azonban egy újabb végletet jelentett: immáron táblázatok helyett rétegek (DIV) szerepeltek az oldalon belül, mindenféle szemantikai jelentés nélkül, ami egy módosításnál azonnal mutatta, hogy az egyszerűbb felépítés nem biztos, hogy a legjobb. A fejlesztők gyakran elkövették ezt hibát, nem használták ki a HTML adta lehetőségeket. A HTML-ben nem csak egy elem létezik a megjelenítésre, hanem minden egyes elem saját jelentéssel is bírhat. Szükségessé vált tehát a CSS és HTML szoros együttműködése, a szintaxis és szemantika egységére a könnyebb fejlesztés érdekében.

Mára már sikerült eljutni oda, hogy az oldalak felépítésében a HTML-ben szereplő elemek nagy része megtalálható, és azok valóban a nekik szánt funkciókat látják el. Ezáltal egy jobban átlátható felülethez jutottak mind a felhasználók, mind a fejlesztők és a különféle platformokra történő fejlesztés is felgyorsult. Bár a weboldalak nagy többségét még mindig számítógépen keresztül látogatják, egyre terjednek azok a mobil eszközök is, amelyek képesek az internet elérésére egy teljes értékű böngészővel. A probléma azonban adott: hogyan jelenítsünk meg egy jóval kisebb felbontású képernyővel rendelkező eszközön egy weboldalt úgy, hogy az funkcionalitásából ne veszítsen. Az olyan jellegű funkciókat pedig szintén érdemes megemlíteni, amik például a nyomtatásból vagy a vakok és gyengénlátók számára optimalizált oldalakból fakadnak. A CSS és a HTML együttműködése tehát elengedhetetlen és szükséges, legyen szó bármiről is egy weboldalon.

A könnyebb oldalszerkezet manipulációja pedig lényegesen egyszerűbb, egy oldal Javascript funkcióit pedig végre kiteljesedhetnek. Korábban is megfigyelhető volt, egy-egy kisebb effekt megvalósításához javascriptet használtak, azonban ezek elég visszafogottak voltak és nem feltétlenül jártak a szerkezet változtatásával, köszönhetően a túl összetett felépítésnek. Nem egyszer fordult elő azonban, hogy például egy olyan egyszerű dologhoz, mint hogy legyen aláhúzott egy link, vagy változzon meg a háttérszíne, ha fölé viszem az egeret, a fejlesztők a Javascriptet hívták segítségül. Az oldalszerkezet CSS alapúvá tételével ez is megváltozott, mivel a CSS lehetőséget kínál ezen egyszerű effektek gyors megvalósítására, nem kell külön programot írni hozzá. Az út tehát nyitva állt az összetettebb funkciók felé – beköszöntött a valódi Ajax és Web 2.0 korszaka.

A Javascript ereje

Az oldalszerkezet könnyítésével tehát sikerült elérni, hogy a jóval összetettebb Javascript függvények is előtérbe kerüljenek, az olyan szolgáltatások pedig, mint az Ajax valóban mindennapi jelenséggé váljanak. De a fejlesztők hajlamosak túlzottan sok ilyen funkciót használni, és ezáltal elvész a honlap rugalmassága, a felépítmény ismét csak egy zagyva szöveg lesz. Fontos tehát a mértéktartás; a Javascriptet ott használjuk, ahol valóban szükség van rá, ne erre épüljön fel egy-egy weblap, a felhasználókért legyen, ne pedig ellenük.

A funkciók tehát egyre nagyobb szerephez jutottak, és ideje volt valamilyen egységet kovácsolni a rengeteg, hasonló jellegű eseménykezelőből és műveletből, azokat egy rendszerbe helyezni és ide is bevezetni a kódolási módszertanokat. Erre a legkülönfélébb eszközök kerültek forgalomba, melyek a különböző Javascript-effektek egységesítését szolgálták. A mai két leggyakrabban használt ilyen keretrendszer a PrototypeJS és a jQuery, de számos más keretrendszer is létezik, melyek ugyanilyen jól használhatóak, mindenki saját szájíze szerint dönthet ezek használatáról. A két megemlített közül talán a jQuery használata egyszerűbb, bár a nyelvezetét ennek is, mint sok minden másnak, szokni kell.

Mi az a jQuery?

A jQuery könyvtár egy általános célú réteget biztosít az egységes programozási környezethez, épp ezért hathatós segítséget nyújt szinte bármilyen script feladathoz. Minden egyes funkció és képesség leírása a könnyű bővíthetősége miatt jelentősen túlnő ezen szakdolgozat határain, főleg, hogy nap mint nap új kiegészítő lát napvilágot az eredeti könyvtárhoz. A legfőbb funkciók talán az alábbi kategóriákba sorolhatóak be:

- **A weblap egy részének elérése.** Egy megfelelő Javascript könyvtár nélkül rengeteg kódot igényelne a weblap egy részének elérése, a DOM bejárásához szükséges funkciók megírásával és a HTML kívánt részének behatárolásával. A jQuery egy robosztus és hatékony szelektálási mechanizmust biztosít a dokumentum egy adott részének eléréséhez és manipulálásához.
- **Egy oldal megjelenítésének módosítása.** A CSS hathatós eszköz a weboldalak megjelenítésére, de sajnos amíg a böngészők nem ugyanazt a szabványt támogatják, vagy nem úgy támogatják a szabványt, ahogy az meg lett fogalmazva, a fejlesztők különféle trükkök alkalmazására kényszerülnek az egységes megjelenítés érdekében. A jQuery böngészőfüggetlen módon kínál megoldást erre a problémára is.
- **Az oldal tartalmának módosítása.** Nem korlátozódik pusztán kozmetikai változásokra, a jQuery egymaga képes módosítani a teljes oldal tartalmát alig néhány sornyi kód megírásával. Segítségével a szöveget dinamikusan meg lehet változtatni, képeket lehet beszúrni vagy eltávolítani, a listák újrarendezhetőek, vagy akár a teljes oldal-struktúra újraírható ezen könnyen használható API segítségével.
- **Válasz a felhasználói beavatkozásra.** Még a legjobb funkciók is haszontalanok, ha azokat nem a kellő időben és helyen alkalmazzuk, ott és akkor, ahol azokra szükség lenne. A jQuery segítségével egy elegáns úton kezelhetjük le az oldalon történő eseményeket, interaktívvá tehetjük a teljes weblapot a legkülönfélébb eseménykezelők megírásával, mindezt a böngésző fajtájától függetlenül.
- **Egy oldal animálása.** Az interaktív műveleteket érzékeltetni is kell a felhasználóval. A jQuery számos vizuális visszacsatoláshoz kapcsolódó effektet támogat, ideértve az át- és eltűnéseket, valamint a legkülönfélébb mozgásokat. Lehetőséget kapunk saját, egészen új mozgásmechnizmus megalkotására is.
- **Információk lekérdezése a webszerverről az oldal frissítése nélkül.** Ez a szolgáltatás AJAX (Asynchronous JavaScript and XML) néven vált ismertté, és lehetővé tette a fejlesztők számára a funkciókban gazdag weboldalak létrehozását. A jQuery segítségével egységes módon, böngésző-függetlenül tudjuk az ilyen szolgáltatásokat biztosítani, lehetővé téve ezzel a fejlesztőknek, hogy valóban a végső kimenettel foglalkozzanak.

A jQuery segítségével egyszerűbbé válnak azok a feladatok, amik egy interaktív weblap fejlesztése során felmerülnek, tehát mindenképpen érdemes kihasználni az ebben rejlő erőt.

Szelektorok

A jQuery talán egyik legfontosabb újítása a selectorok használata, az adott elemek kiválasztására ID alapján, vagy akár a dokumentumban elfoglalt hely, osztály, vagy egyéb azonosító alapján. A használat megkezdéséhez természetesen szükségünk lesz a függvénykönyvtárra, amit a http://jquery.com weboldalról tudunk letölteni. De vigyázni kell, jelenleg két verzió van forgalomban: az 1.2 és az 1.3, amik között bár nagyon lényeges eltérés nincs használat közben, azonban sok kellemetlen pillanatot tud okozni, ha nem figyelünk ezekre.

A legelső, amit a használathoz meg kell jegyezni, az alapoperátor. Kétféleképpen használhatjuk, az egyik jQuery(), de a rövidebb talán egyszerűbb: $. Három alapszelektort különböztetünk meg:

- **HTML Tag kiválasztása**: például $('P'), jelentése az összes paragrafus elem az adott HTML fájlban
- **ID alapján**: $('#azon'), jelentése azon egyetlen elem, aminek az azonosítója „azon".
- **Osztály szerint**: $('.object'), jelentése az összes olyan elem kiválasztása, ami rendelkezik object osztállyal (class="object").

Ezek nagyon ismerősek lehetnek a CSS-ből, talán nem is véletlenül: A CSS szelektorok mindegyike használható a jQuery-ben. Ugyanakkor ezzel nem merül ki a dolog, az XML-ből ismert XPath szelektorok szintén használhatóak; ezekre néhány példa:

- **$('A[@rel]'):** az összes olyan link, aminek van rel attribútuma (csak az 1.2-es verzióban)
- **$('A[href^=mailto:]')**: az összes olyan link, aminek az értéke e-mail-küldésre utal
- **$('A[href$=.pdf]')**: az összes (vélhetően) PDF kiterjesztésű fájlra mutató link
- **$('A[href*=barmi.hu]')**: az összes olyan link, amiben szerepel a barmi.hu kifejezés.

A szelektorok természetesen bővíthetőek saját magunk által definiáltakkal, a munka megkönnyítése érdekében, de ezekre csak ritkán van szükség.

Adat- és attribútum-manipuláció

Mint korábban említettem, a jQuery segítségével lehetőségünk van az elemek tartalmának megváltoztatására is, de a dolog nem merül ki itt. Lehetőségünk van új attribútumok hozzáadására, meglévők eltávolítására, adat hozzáfűzésére. Az ezekhez szükséges néhány alapvető függvény:

- $('P').**html():** paraméter nélkül meghívva a dokumentum legelső paragrafusának tartalmával tér vissza, paraméteresen meghívva az összes paragrafus tartalma le lesz cserélve a paraméterre.
- $('P').**text()**: hasonló az előzőhöz, azonban a paraméter és a visszatérési érték tisztán szövegként lesz értelmezve

- $('P').**attr(attr[,value])**: lekérdezi az első paragrafus attr attribútumának értékét, ha a második paramétert is megadjuk, akkor az adott attribútumot arra állítja be.
- $('P').**addClass(classname)**,$('P').**removeClass(classname)**: hozzáadja/eltávolítja az összes paragrafusról a megadott osztályt.

De számos további is létezik, a például az append hozzáfűz az adott elemhez, a prepend a megadott elem tartalma elé helyezi a paramétert, a toggleClass pedig például hozzáadja vagy eltávolítja a megadott osztályt, attól függően, hogy volt-e az adott elemnek ilyen vagy sem.

Eseménykezelés

Talán az egyik leghasznosabb eszköz, segítségével a weboldalon bekövetkező különféle eseményekre tudunk reagálni, mint az adott elemen történő vagy globális kattintás, egy form elküldése, a különféle egér-események, billentyűzet-leütés, stb. Általában ezt az onmouseover, onkeyup, onkeydown, stb. attribútumokkal lokálisan is lehet kezelni, de a kód jóval átláthatóbb lesz, ha a javascript is egy helyre kerül. Néhány eseménykezelő:

- $('A').click(függvény)
- $('FORM').submit(függvény)
- $('INPUT').keypress(függvény)

Ezek a függvények azonban a bind függvény alternatívái. A bind segítségével rendelődnek hozzá a megadott eseményekhez, így a click() a bind('click',param) egy elfedése, a könnyebb használat érdekében, de az összes eseménykezelő ebből származik. Fontos ugyanakkor, hogy a paraméternek egy függvénynek kell lennie, nem pedig függvényhívásnak, hiszen csak az esemény bekövetkezésekor kell meghívni, nem pedig a hozzárendelésnél.

Ajax

Szintén igen hasznos eszköz a weboldalfejlesztés folyamán, leveszi a terhet a programozó válláról, ha a szerver és kliens közötti gyors információcseréről vagy adattovábbításról van szó. Nem kell a szokásos XmlHttpRequest objektumot lekérdezni, annak a módjait böngésző-fajtánként beállítani, ezt megteszi helyettünk a jQuery. Az Ajax lekérdezésekhez a legalapvetőbb funkció a $.ajax utasítás, ami egy objektumot vár paraméterül, a szükséges beállításokkal. Ezek közül a legfontosabbak:

- **url**: a szerveren található fájl neve, mellyel fel kívánjuk venni a kapcsolatot
- **data**: az elküldésre kerülő adatok, valtozo1=ertek1&valtozo2=ertek2&… formában; nem kötelező megadni
- **type**: az elküldés típusa, lehet POST vagy GET. Az alapértelmezett érték a GET.
- **dataType**: a válasz típusa, lehet null, xml, script vagy json. Az alapértelmezett érték null, ilyenkor a válasz előfeldolgozásra kerül, szétválasztva ezzel a javascriptet a html-től.

- **error**: függvény, mely akkor kerül meghívásra, ha valamilyen hiba történt a végrehajtás folyamán, vagy a visszaérkezett válasz nem felel meg a kívánalmaknak (pl. json-t vártunk, de szimpla HTML-t kaptunk)
- **success**: az ajax kérés sikeres befejezése után kerül meghívásra, az ebből származó DOM manipulációt itt tudjuk elvégezni

Az ajax függvénynek két elfedése a post és a get, értelemszerűen a kétféle lekérdezés felgyorsítása érdekében. Több esetben célszerű ezeket használni, összetett feladatok esetén viszont az eredetihez kell visszanyúlni. További ilyen feladatok ellátó függvények az ajaxForm és az ajaxSubmit, mindkettő a formok kezelésénél lehet segítségünkre.

Összefoglalás

A fejezetben átbeszéltük, hogy miért alakult ki mára a CSS alapú HTML DOM és miért van szükség a könnyűsúlyú HTML-re, valamint megismerkedtünk egy igen hasznos eszközrendszerrel is, melynek segítségével megkönnyíthetjük az oldal használhatóságát és a fejlesztés lehetőségeit egyaránt. Megnéztünk néhány, az eszközrendszer nyújtotta lehetőséget és átvettük a legfontosabb funkciókat is.

7

Formok és adatbevitel

Egy weblapon a szerver és a felhasználó között a kommunikációs felületet a HTML biztosítja, a látogató ezen keresztül értesül a portálra felkerült újdonságokról, termékekről, akciókról és egyebekről. A legtöbb esetben viszont ahhoz, hogy egy-egy terméket el tudjunk adni, vagy a felhasználót rávegyük arra, hogy iratkozzon fel egy hírlevélre, szükség van egy fordított irányú kommunikációra is. Itt jönnek képbe a HTML által biztosított beviteli elemek, a formok. Egy form segítségével POST vagy GET típusú kéréseket tudunk küldeni a szerver irányába a kliens oldalról és az így kapott adatokat feldolgozhatjuk a PHP-vel. A formokban különféle elemek segítségével különféle adatokat tudunk bekérni a felhasználótól:

- **Egyszerű szövegbevitel**: erre szolgál az INPUT elem text típusa, illetve a TEXTAREA elem
- **Választható érték**: erre több eszköz is van:
    - SELECT, mely segítségével egy legördülő listát tudunk kialakítani
    - rádiógomb (INPUT type=”radio”), amivel egymás mellett felsorolhatjuk a választható értékeket
    - checkbox (INPUT type=”checkbox”), amivel egy igen-nem kérdésre adhatunk például választ
- **Több választható érték:**
    - SELECT, ami rendelkezik egy multiple attribútummal; ilyenkor többnyire a shift gomb megnyomásával jelölhetünk ki több elemet
    - checkbox, aminek a név attribútum szögletes zárójelekkel végződik; egyszerűen jelölhetjük, hogy milyen opciókat kérünk valamihez.
- **Gombok**: bár ez nem adatbevitel, a későbbiek során mégis fontos szerephez fognak jutni. Az ilyen gombokra való kattintással küldhetjük el a bevitt elemeket, és rendelhetünk a gomb megnyomásához eseményt, ezáltal validálhatjuk és dolgozhatjuk fel a formot.

Mivel az ilyen adatok a kliens oldalról származnak, potenciális veszélyforrást jelentenek a szerverre nézve. Ha nem bánunk velük elég körültekintően, könnyen segíthetünk bejutni a támadóknak a webszerverre és ezáltal olyan információkhoz juthatnak, amikhez egyébként nem lenne jogosultságuk. Ezen okok miatt rendkívül fontos a felhasználó által bevitt adatok

ellenőrzése és az esetleges támadókódok automatikus hatástalanítása. Ez sajnos ahhoz is vezet, hogy a felhasználók számára nehézséget okozhat például egy regisztrációs form kitöltése. Éppen ezért a kliens oldali ellenőrzés legalább akkora szerepet kap, mint a szerver oldali validáció.

Validációs alapműveletek

Az egyes mezők ellenőrzéséhez szükség van néhány alapműveletre és néhány alaptulajdonságra, amit vizsgálni és érvényesíteni kell a hibátlan kitöltéshez. Ezek az alábbiak, a teljesség igénye nélkül:

- **Kötelező**: a leggyakoribb tulajdonság egy adatbeviteli elemnél, ha az itt szereplő érték üres, vagy csupán szóközöket tartalmaz, akkor a kitöltése hibás, a megadott értéket semmisnek tekintjük. Fontos megjegyezni, hogy a PHP akkor is üresnek tekint egy értéket, ha az adott elem nem létezik, 0 vagy NULL az értéke vagy valóban üres.
- **Minimum, maximum érték vagy hossz**: a mezőben szereplő értéknek bizonyos határok közé kell esnie (például ma még nem lehet valaki 200 éves). A hossz vizsgálat ugyanígy fontos, hiszen egy túl hosszú érték könnyen puffer-túlcsorduláshoz vezethet, ezáltal pedig rést nyit a szerveren.
- **Minta:** ha egy érték valamilyen reguláris kifejezéssel ellenőrizhető, például e-mail cím, telefonszám, vagy hogy pozitív egész vagy valós számról van szó, akkor használhatjuk a minta fogalmát.
- **Egyenértékűség:** ezt nem önmagában, hanem egy másik elemmel együtt érdemes alkalmazni, például ha jelszót kérünk be a felhasználótól és azt meg kell erősíteni.

Ezen műveletekre érdemes egy teljes értékű osztályrendszert megalkotni, ami automatikusan elvégzi helyettünk ezen tulajdonságok teljesülését. A rendszernek természetesen bővíthetőnek és könnyen kezelhetőnek kell lennie. Támogatnia kell az automatikus értékszűrést, vagyis a dupla és felesleges szóközök automatikusan kerüljenek eltávolításra, illetve a bevitt HTML kódokat is szűrni kell aszerint, hogy mi jelenhet meg (például egy felhasználói név nem tartalmazhat formázott szöveget).

Az egyes formok egymástól jól el kell különüljenek, hogy ne legyen névütközés és így hibás működés, vagy félreértelmezett információ. Ez utóbbi alatt azt kell érteni, hogy az egyes beviteli mezők neveinek nem szabad csak egyszerű neveket adni, mint email, jelszó, stb., hanem egyedi azonosítóval kell ellátni azokat, így a belépési „username” mező elkülöníthetővé válik a regisztrációnál megadott, hasonló nevű mező értékétől, amennyiben azok egy adatküldésben szerepelnének. Így a „username” mezőből „register_username” és „login_username” lesz.

Az alábbiakban egy egyszerű példát mutatunk egy regisztrációs formra, majd felírjuk, mik azok a követelmények, aminek eleget kell tegyen a felhasználó az egyes mezők kitöltése során.

7.1 ábra – Regisztrációs form

Az első, ami szembeötlik az a form felépítése, jól láthatóak, hogy melyik mezőbe milyen adat kell, hogy kerüljön. A fenti egyszerű formon nyilvánvalóan vannak kötelező mezők, illetve olyanok, amik bizonyos feltételeknek kell, hogy megfeleljenek, részletesebben:

- **Email cím:** kötelező mező, az email formátumának kell megfeleljen, egyedinek kell lennie
- **Jelszó:** kötelező mező legalább 6 karakter
- **Jelszó ismét:** szintén kötelező és minimum 6 karakter hosszú, és meg kell egyezzen a jelszóval
- **Név:** kötelező megadni, legalább két karakter, maximum 255 karakter hosszú lehet
- **Ország:** választható mező, nem kötelező megadni
- **Város, utca, házszám:** nem kötelező megadni
- **Irányítószám:** nem kötelező megadni, kitöltése esetén 4 számot tartalmazhat (maximum hossz, formátumellenőrzés szükséges).

Az egyes mezők ellenőrzése tehát ezen elvek alapján kell, hogy történjen, láthatóan a műveletek nagy részét a korábban definiált alapfunkciók alkotják. Az egyetlen plusz az e-mail cím ellenőrzésének egyedisége, ami általában nem általános meghatározható, így új művelet meghatározását is igényli.

Kliensoldali ellenőrzés

Az osztályrendszernek támogatást kell nyújtania a részleges vagy teljes kliensoldali ellenőrzés is, így elkerülve az érvénytelen validációt vagy azt, hogy egy-egy funkciót kétszer kelljen megírni. A kliensoldalon a HTML 4-ben még csak egy eszköz van a formok ellenőrzésére, ez pedig a

Javascript. A HTML legújabb változata azonban már vélhetően változást fog hozni ezen a téren is, hiszen a fentebb említett alapműveletek beépítetten támogatni fogja, bár az esetleges hibák visszajelzése egyelőre kétséges. A mai böngészők többsége azonban még csak részben, vagy egyáltalán nem támogatja az új szabványt, így meg kell maradnunk a Javascript nyújtotta lehetőségeknél.

Nagyon fontos megjegyezni ugyanakkor, hogy a kliens oldali ellenőrzés nem helyettesíti a szerver oldalon elvégzendő műveleteket, hiszen egy inaktivált javascript funkcionalitás máris védtelenné tenné a weblapot. A kliens oldali ellenőrzés így inkább egy kényelmi funkció, mellyel azonnal reagálhatunk az esetleges hibákra, a feldolgozás így felgyorsul és a felhasználó számára is egyszerűbbé válik a beviteli mezők kezelése.

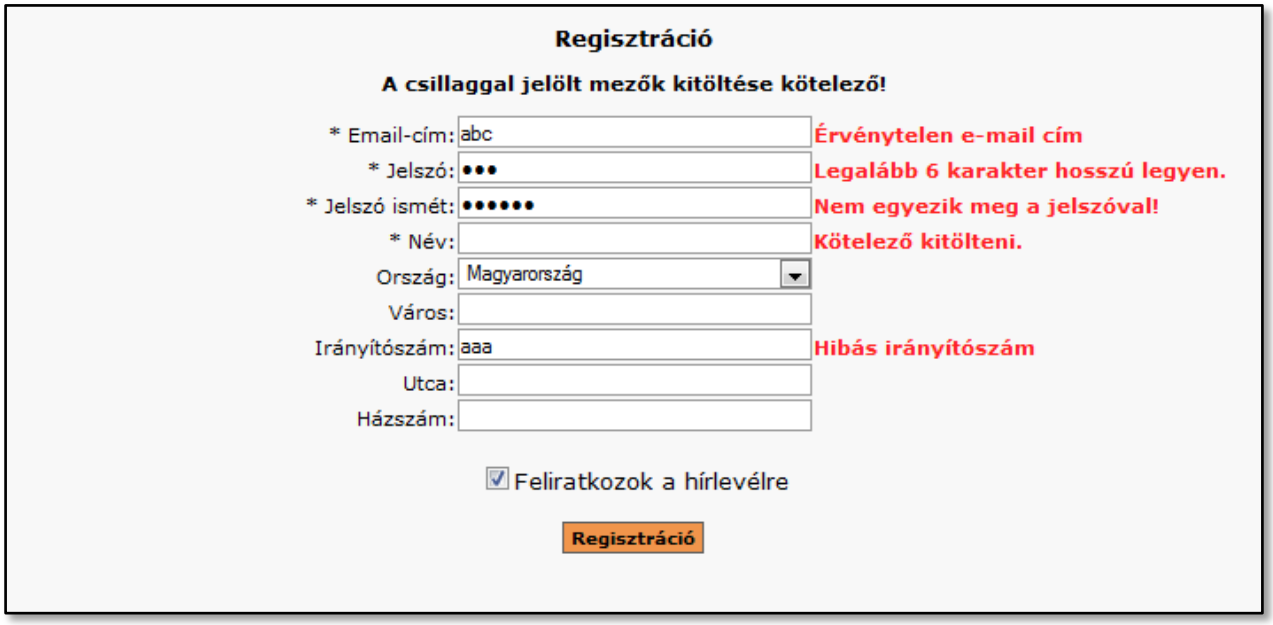
Szerencsére kiváló és szabadon felhasználható megoldások léteznek, melyek közül talán az egyik legjobb egy jQuery kiegészítő, melynek neve Validation. Segítségével könnyen és gyorsan elvégezhetjük egy form kliensoldali ellenőrzését és a felhasználók is látni fogják az esetlegesen általuk vétett hibákat. Szerencsére vagy nem, a kiegészítő a hibajelzést egy LABEL címkében illeszti be a DOM-ba, közvetlenül a hibát okozó input elem után. Ez annyit von magával, hogy a használata szinte megköveteli a CSS alapú oldalfelépítést és ez nagy figyelmet igényel, ez azonban szerencsére eltörpül az előnyök mellett. A fent említett alapműveleteket már beépítve tartalmazza, így nekünk elegendő egy gyors beállítás a teljes működéshez.

A példához tartva magunkat az alábbi kódot kell beilleszteni a dokumentum fejrészébe, vagy törzsébe (ízlés dolga, de inkább az előbbi legyen):

```
<script type=”text/javascript” src=”jquery.validate.js”></script>
<script type=”text/javascript” src=”jquery.validate.messages_hu.js”></script>
<script type=”text/javascript”>
$(document).ready(function(){
    $(’FORM[name=register]’).validate({
        rules: {
            ’register_email’: {required:true,email:true},
            ’register_jelszo’: {required:true,minlength:6},
            ’register_jelszo_ismet’: {required:true,minlength:6,
                        equalTo:’registration_jelszo’},
            ’register_nev’: {required:true,minlength:2,maxlength:255},
            ’register_irszam’: {regexp:/^([1-9][0-4]{3})?$/i}
        },
        messages: {
            ’register_irszam’: {regexp:’Hibás irányítószám’},
            ’register_jelszo_ismet’: {equalTo:’Nem egyezik meg a jelszóval!’}
        }
    });
});
</script>
```

7.1 kód – Validáció jQuery-vel

Mint látható, két fontos részre tagolódik a paraméterezés. Az egyik a szabályok meghatározására, a másik az egyes hibák egyedi üzeneteinek meghatározására szolgál. Utóbbira csak két esetben van szükség, hiszen a kiegészítővel érkezik magyar nyelvi támogatás is. Van egy kis csalás is a dologban, mivel a regexp, mint validációs elem, nem része a kiegészítőnek; az ehhez szükséges függvényt külön meg kell írni, bővíteni kell az osztályt, de ezt egyszerűen el lehet végezni. A hibajelzés először a form elküldésekor fog megtörténni, minden mező alatt vagy felett, de ez csak attól függ, hogy a CSS-ben hogyan definiáltuk a megfelelő HTML tag-et. Ilyenkor a létrehozott LABEL elem kap egy „error” osztályt, ahogy a hibát okozó input elem is, így még inkább vizualizálhatóvá válik a hiba oka. Ha a felhasználó megpróbálja elküldeni a formot, és így próbál érvénytelen adatokkal regisztrálni, a kiegészítő ezt megakadályozza és a következő eredményt látja viszont:



7.2 ábra – Hibás adatokkal kitöltött regisztrációs form

Láthatóan a várt eredményt hozta a dolog, így a felhasználónak már csak végig kell menjen az egyes mezőkön, megvizsgálva azokat, hogy mi lehet a probléma.

Szerver oldali ellenőrzés

A szerver oldalon szintén szükség van ugyanezen műveletek elvégzésére, hiszen mint azt korábban említettem, a kliensoldali ellenőrzés inkább kényelmi funkció, semmint valós adatellenőrzés, a meglétére és működésére pedig nem szabad hagyatkozni. A szerveroldalra tehát szükség van egy hasonlóan jó funkcionalitással bíró osztályrendszerre, amivel az is könnyen elvégezhető. A legtöbb PHP alapú keretrendszer rendelkezik beépített támogatással a formokhoz, azonban ezeknek sok esetben kényelmetlen a használatuk, alig, vagy egyáltalán nem bővíthetőek és ezért nem kínálnak jó alternatívát.

A megoldást egy saját osztályszerkezet kialakítása jelentette, ami támogatja a kliensoldali ellenőrzéshez szükséges kód kialakítását is. A rendszer teljesen objektumorientált, a validáció is objektumokon keresztül zajlik. Az egyes validációs objektumokat úgy kell elképzelni, mint a szűrőket a szennyvíznél: minél több szűrőt használunk, annál tisztább lesz a kapott adat.

```
<?php
class Register extends Form {
    public function __construct() {
        parent::Form('register');
        $email = new FTexT( 'email' );
        $email->addValidator( new RequiredValidator( $email ) );
        $email->addValidator( new Regexp( Regexp::EMAIL, $email ) );
        $this->addElement( $email );
        $this->addElement( new FText( 'varos' ) );
        $irszam = new FText( 'varos' );
        $irszam->addValidator( new Regexp( '/^([1-9][0-9]{3})?$/i', $irszam,
'Hibás irányítószám' ) );
        $this->addElement( $irszam );
        …
        $this->addElement( new FButton( 'regisztral', 'RegisterUser' ) );
        $this->Execute();
    }
    …
    public function IsValid() {
        $result = parent::IsValid();
        if ( $result ) {
            //ellenőrizzük, a fennmaradó információkat, például regisztrált-e
            //már valaki korábban hasonló e-mail címmel
        }
        return $result;
    }
    …
    protected function RegisterUser() {
        try {
            //regisztrációs műveletek
            $this->ClearForm();
            return new SuccessMessage( 'A regisztráció sikeresen megtörtént' );
        }
        catch ( Exception $e ) {
            Settings::LogException ( $e );
            return new ErrorMessage( 'A regisztráció során hiba történt, kérjük
próbálkozzon ismét!' );
        }
    }
    …
}
?>
```

7.2 kód – Szerver oldali validáció a form osztály segítségével

A form elemek nevei úgy válnak egyedivé, hogy azt mi közvetlenül nem látjuk: így az email mező a $_POST változóban a register_email nevet fogja kapni. Ez a szülőosztály konstruktorának meghívásával és az egyes Látható, hogy a fenti kódrészletben az ellenőrzést a korábban említett objektumok végzik, illetve tartalmaz egy kibővített ellenőrzést is, így olyan funkciót kap, amit a beépített validátor-objektumokkal nem lehetne elvégezni. Természetesen mivel a rendszer bővíthető, egy saját validáló elem megírása is megoldást jelenthet az alábbi interfész implementálásával:

```
<?php
interface Validator {
    public function IsValid();
    public function GetMessage();
}
?>
```

7.3 kód – Validátor interfész

Az e-mail cím egyediségének ellenőrzésére szolgáló validátor tehát így fog kinézni:

```
<?php
class UniqueEmailValidator implements Validator {
    protected $Message;
    protected $Conn;
    protected $Email;

    public function __construct( $email,
       $message='Már valaki regisztrált ezzel az e-mail címmel!' ) {
        $this->Conn = MySQLConnection::GetInstance();
        $this->Email = $email;
        $this->Message = Message;
    }

    public function IsValid() {
        return $this->Conn->Prepare("SELECT user_id FROM szakdolgozat_users
WHERE email=':|1|:' LIMIT 1;" )->Execute( $this->Email )
             ->NumRows()==0;
    }
    public function GetMessage() {
        return $this->Message;
    }
}
?>
```

7.4 kód – Saját validátor készítése

Mint látható, az objektum a korábban átbeszélt MySQLConnection segítségével ellenőrzi, hogy a megadott e-mail szerepel-e az adatbázisban. A hibajelzés szintén egyedi lehet és automatikusan az adott mezőhöz rendelődik a hibajelzés, amennyiben ez szükséges. Az adott elem aktuális értékét a CurrentValue adattag tárolja. Lehetőség van alapértelmezett érték megadására is, az input konstruktorában és magában az objektumban a DefaultValue érték megadásával.

Online validáció

Hogy a kép teljes legyen, létezik egy másik interfész is, amivel a feldolgozó tudtára adhatjuk, hogy a validátor a kliens oldalon is létezik, illetve a kliens oldal közvetlenül együtt tud működni a szerverrel. Ilyen esetben, amennyiben olyan függvényről van szó, ami nincs megírva (a kliens oldalon a Validator bővítményben nem szerepel), azt is jelezni illetve implementálni kell a hibák elkerülése végett.

```
<?php
interface ClientSideValidator extends Validator {
    public function GetScript();
    public function GetScriptMessage();
}
?>
```

7.5 kód – Kliens oldali validációhoz szükséges interfész

Ahhoz, hogy a kliens oldali Validate kiegészítővel együtt tudjon működni a rendszer, és az e-mail cím is validálásra kerüljön annak elküldése előtt, ennek implementálására van szükség. Az ellenőrzéshez ajax-ot fog használni, tehát szükség lesz egy szerver oldali részletre, amit szerencsére tisztán a validátorral meg lehet oldani. A szerver oldali validátor tehát a következőképpen változik:

```
<?php
class UniqueEmailValidator implements ClientSideValidator {

    const URL = 'ajax/registration/checkMail.php';

    public function getScriptMessage() {
        return 'remote: "'.$this->getMessage().'"';
   }
    public function getScript() {
        return array( 'rules'=>'remote: "'.self::URL.'"' );
    }
}
?>
```

7.6 kód – Kliens oldali validátor készítése

Ebben az esetben a szerver oldali Form objektumban automatikusan meghívásra kerül ezen validátor két függvénye, így legenerálva a szükséges scriptet. Ha a teljesítmény mindennél fontosabb, maradjunk az előzőleg megírt validátornál, így nem generálunk plusz többletet a szerver számára, a kliens oldali scriptet pedig állítsuk össze mi magunk, hiszen a fenti példából is látható, a kliens oldali jQuery plugin támogatja az ajax-ellenőrzést is az egyes elemek esetén a remote függvény segítségével. Paraméterül vagy egy, a $.ajax függvénynek is átadható objektumot, vagy egy egyszerű URL-t vár, amit az adott elem nevével és értékével paraméterez fel. A fenti kód a következőt url-re hivatkozik: ajax/registration/checkMail.php. Azaz a fenti mappában egy checkMail.php-t kell elhelyezni, melynek tartalma:

```
<?php
try {
    $email = 'registration_email';
    if ( isset( $_GET[$email] ) && $_GET[$email] ) {
        $validator = new UniqueEmailValidator( $_GET[$email] );
        $result = $validator->IsValid()?'1':'0';
    }
    else {
        $result = '0';
    }
}
catch( Exception $e ) {
    $result = '0';
}
echo $result;
?>
```

7.7 kód – Ajax validáció, a PHP irányából

A kliens oldal rövidzár-kiértékelést vall, azaz a távoli kapcsolatot igénylő ellenőrzéseket lehetőség szerint az ellenőrzési paraméterek végére kell tenni, így a szerver oldalra csak akkor kerül majd adat, ha az valóban (vagy legalábbis nagy valószínűséggel) érvényes információt hordoz. A szerver oldalon elegendő egy igen/nem válasz annak eldöntésére, hogy az adott elem nem tartalmaz-e hibát, a kliens oldal pedig ennek megfelelően dönt a form további sorsáról, ahogy a többi paraméter esetén is. Fontos tudni, hogy az így végzett ellenőrzés nem aszinkron, azaz a választ a további működéshez megvárja a szervertől (bizonyos időintervallumon belül persze), így nem fordulhat elő, hogy a form előbb kerül elküldésre, mint az ellenőrzéshez szükséges információ visszaérkezik.

Összefoglalás

A fejezetben átvettük a beviteli elemek ellenőrzésének fontosságát, illetve láthattuk az automatikus validáció egy lehetséges megvalósítását is. A rendszer megfelel az előzetesen tett kívánalmaknak, így megfelelően bővíthető és a kliens oldali támogatás is adott. A használathoz mindössze néhány dologra kell odafigyelni, ha a szerver oldalon megjegyzi az ember a szükséges dolgokat, úgy a kliens oldal sem fog problémát jelenteni és fordítva. A kliens oldalon ismét láthattuk, milyen fontos a megfelelő HTML kód összeállítása és a CSS kellő ismerete.

8

Ajax

A Web 2.0 egyik legérdekesebb jellemzője az újratöltés nélküli tartalom-megjelenítés. Ez a gyakorlatban annyit jelent, hogy a webszerverről anélkül kérünk le információt, hogy a teljes weboldalt újból meg kellene jeleníteni. Korábban ez kiegészítők és egyéb programok nélkül nem volt lehetséges, aztán a Javascript a fejlesztők kezébe adta a kulcsot. Ez nem volt más, mint az XmlHttpRequest objektum, amivel új kérést lehetett kezdeményezni a weboldal irányába, az eddig megszokott POST és GET kérelmeken keresztül. Beköszöntött az AJAX korszaka

Az AJAX lényegében egy betűszó, Asynchronous Javascript And XML, vagyis aszinkron továbbított adatra utal, mögötte pedig az előbb említett objektum áll. Ennek használata azonban meglehetősen körülményes, rengeteg plusz kód megírását igényli és nem megfelelően használva a korábban gyorsításnak szánt funkciók inkább lassulást eredményeznek. Ez persze már inkább a kliensen múlik, és ott is főleg a böngésző javascript-feldolgozó motorjának sebességén, de az átvitt adat mennyiségét és az internet sebességét sem szabad figyelmen kívül hagyni.

A felesleges kódolás persze megspórolható egy jól megválasztott keretrendszer használatával. A korábban átbeszélt jQuery kiválóan alkalmas az ilyen feladatok ellátására, ahogy az a fejezetből kiderült. Segítségével gyorsan tudunk formokat továbbítani, vagy csak egyszerű információkat lekérdezni, a feldolgozás pedig lehetőség szerint a kliens oldalon történjen. A választ háromféleképpen kaphatjuk meg, mindre van kezelő eszköz:

- Egyszerű HTML – a válasz ilyenkor szimpla HTML-ként érkezik, feldolgozása ennek megfelelően történik. Ezt akkor érdemes használni, ha új blokkokat kívánunk beilleszteni az oldalba, de a kliens oldalon történő HTML összeállítás túl körülményes, és már azt korábban megírtuk. Viszonylag kis mennyiségű információ közvetítésére ez a legmegfelelőbb forma. Szintén ebbe a kategóriába tartozik az egyszerű szöveg is.
- JSON – lényegében egy Javascript objektum, mely név-érték párokat tartalmaz, így kiválóan alkalmas egyszerű üzenetek közvetítésére, sikeres/sikertelen, igen/nem jellegű döntéshozatalra. Nem terheli a hálózatot, lényegi információt tartalmaz csak. Ez lesz a leggyakrabban használt módszer adatkommunikációra.

- XML – ilyen módszerre akkor lehet szükség, ha az adott elemet nem csak közvetlenül a weboldalon, hanem akár egy Flash-fájlból szeretnénk felhasználni, és bár hasonló bemenetre hasonló kimenet a válasz, a dolog így egységessé válik és nincs szükség egy funkció többszörös lekódolására.

A felhasználásnak azonban vannak korlátai. Ilyen lekérdezéseket böngészőből csak a saját névtéren belül indíthatunk, azaz másik domainre nem hivatkozhatunk a weboldalunkról, és ez sajnos az alterekre is vonatkozik. Ez bár a böngészők biztonsági korlátozása, érdemes odafigyelni erre; ha másik névtérre akarunk hivatkozni, PHP kódra lesz szükségünk a saját honlapunkon, ami ellátja a közvetítő szerepet. Egy másik igen súlyos korlátozás, hogy Javascript segítségével nem tudunk közvetlenül fájlt küldeni. Ez azért is okozhat problémát, mivel például egy kép feltöltése esetén szükségessé válik a weboldal frissítése.

Léteznek azonban áthidaló megoldások ez utóbbi probléma kezelésére, általában kétféle módszert alkalmaznak. Az egyik esetben egy ún. belső kerettel, egy iframe segítségével külön kis weboldalt ágyaznak a rendszerbe, majd a feltöltést javascript segítségével az iframe-ben található form elküldésével indítják. Ez az egyszerűbb eset, de ilyenkor a megjelenítési eszközök limitáltak, nincs lehetőségünk például a folyamat mutatására, hogy hol tart a feltöltés.

A második megoldás már barátságosabban bánik a felhasználóval, de ilyenkor szükség van egy plusz kiegészítő alkalmazására is a böngészőben. Ez a kiegészítő szerencsére a legtöbb esetben megtalálható, vagy könnyen telepíthető: a Flash plugin-ról van szó. Azonban vannak hátrányai is a dolognak. A Flash általában biztonsági problémát jelent a kliens oldalon, ugyanis bár az Adobe rendszeresen frissíti a lejátszó programot, néhány nagyobb rés mindig marad. A szerverre így feltöltött fájlok típusát pedig nem minden esetben közvetíti helyesen, sokszor előfordul, hogy hibás információt szolgáltat róla. A MIME típus megállapítása egyébként általános probléma, néhány böngésző például a kiterjesztés alapján mondja meg, hogy milyen fájlról van szó.

Formok és az AJAX

Az előbb átbeszélt fájlfeltöltésen túl másra is nagyon oda kell figyelni, ha valóban Ajax alapú form-kezelést akarunk kivitelezni. Ha egy formot javascript-tel szeretnénk elküldeni, annak egyetlen módja a *submit* metódus meghívása. Ez azonban hiányosságokat tartalmaz: a legtöbb böngészőben ezen metódus meghívása a olyan lekérdezést eredményez, mely nem tartalmazza az eseményt kiváltó gomb nevét. Ez számunkra azért lényeges, mert egy gombhoz hozzárendelhetünk egy szerveroldali függvényt, de amennyiben ez hiányzik, értelemszerűen a metódus sem kerül meghívásra.

Ugyanakkor a szerializáció során szintén kimarad a kérésből az azt kiváltó objektum neve. A helyzetet tovább nehezíti, hogy az enter billentyű lenyomása szintén kiváltja az elküldés eseményét, és a legtöbb böngésző ilyenkor a formban található legelső submit típusú elemet nevezi meg, mint a kiváltó eseményt, ami számunkra nem feltétlenül a legjobb eredményt jelenti. Van olyan is, mikor ugyanezen eseményre (enter billentyű leütése) a request-be nem kerül bele

egyetlen submit elem neve sem, vagy hasonlóan az előzőhöz, a legelső submit elem neve kerül bele, de azt a böngésző az erre való kattintás kiváltásával végzi.

Tehát a validálást követően valahogyan el kell döntenünk, hogy melyik gomb váltotta ki az elküldés eseményét. A legtöbb form esetében erre nem kell különösebb gondot fordítani, hiszen a legtöbb esetben mindössze egyetlen gomb szerepel a rendszerben. Ha viszont több, érdemes azokhoz eseménykezelőt rendelni valamilyen hasonló módszerrel:

```
$(document).ready(function(){
    var submitter = 'alapertelmezett_gomb_neve';
    $('FORM[name=form_neve] :submit').click(function(){
        submitter = $(this).attr('name');
    });
    $('FORM[name=form_neve]').validate…
});
```

8.1 kód – A küldést kiváltó gomb kiválasztása

A korábban ismertetett regisztrációs formon mindössze egyetlen gomb szerepel, így ezzel különösebb problémánk nem akadhat. Korábban, a formok ismertetésénél már megismertünk egy ajax metódust, amivel egy egyszerű elem validációját elvégezhetjük, név szerint az email cím egyediségét vizsgáltuk. Ebben a fejezetben a teljes formot fogjuk Ajaxon keresztül továbbítani és elvégezni a regisztrációt, feldolgozni a kapott választ és megfelelő értesítést adni a felhasználónak annak sikeréről, vagy hiba esetén, sikertelenségéről. Szerencsére a validator rendelkezésünkre bocsájtja a szükséges eszközök nagy részét, és mivel a form kialakítása miatt az újrafelhasználható ilyen módon is, lényegében csak a kapcsolatot kell létrehoznunk.

```
$(document).ready(function(){
    $('FORM[name=form_neve]').validate({
        //szabályok és üzenetek felsorolása
        submitHandler: function(form){
            var button = $(':submit',form).attr('disabled','disabled');
            var info = $('P.info',form).text('Kis türelmet…').
                removeAttr('class').addClass('info');
            $.ajax({
                url: 'ajax/forms/registration.php',
                data: $(form).serialize()+'&'+button.attr('name')+'=true',
                dataType: 'json', type: 'post',
                error: function(xhr){info.addClass('error').
                  text('A feldolgozás során hiba történt, próbálkozzon ismét!');
                  button.removeAttr('disabled');}
                }),
                success: function(data){
                   P.addClass((data && data.success)? 'success':'error').
                   text(data.message);
                   button.removeAttr('disabled');
                }
      }});});
```

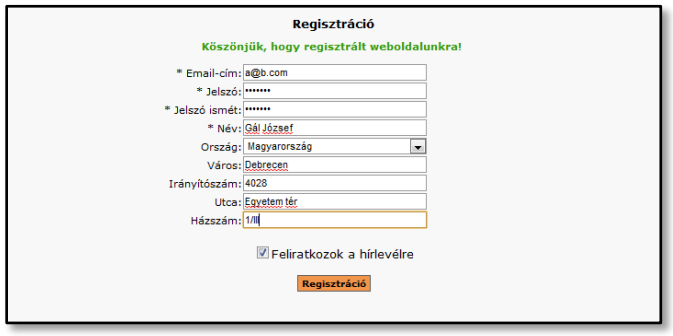
8.2 kód – Ajax-alapú form-küldés a Validate kiegészítővel

A fenti kódban a következő műveletsor játszódik le:

- A sikeres form validációt követően meghívásra kerül a submitHandler nevű függvény, amelynek paramétere a validált form.
- A függvényben az összes gomb, amivel elküldést lehetne kezdeményezni, kap egy attribútumot, így nem lehet azokra még egyszer rákkatintani
- Ajax kérés indul a szerver felé, az *ajax/forms/registration.php* irányába, POST fejléccel. Az adatok a form név-érték párjai, valamint a gomb neve, a visszaérkező válasz várhatóan JSON kódolású
- Ha valamilyen hiba következik be az elküldés, a válasz feldolgozása, vagy bármilyen nem várt esetben, meghívásra kerül az *error* függvény, amely opcionálisan három paramétert kaphat: az *XmlHttpRequest* objektumot, a válasz HTTP státuszát, valamint a hibát okozó elemet.

8.1 ábra – Hiba a feldolgozás során

- Ha nem történt hiba és a válasz is JSON-ban kódolt, akkor a *success* függvény meghívásra, a válaszként kapott *data* paraméterrel. A *data* tartalmazza a válaszobjektumot, amiből eldönthető, hogy történt-e hiba, ha igen, mi az ehhez kapcsolt üzenet, illetve siker esetén mi a válasz (*success*, *message* elemek).



8.2 ábra – Sikeres feldolgozás

- Bármilyen válasz esetén a gombról eltávolításra kerül a disabled attribútum és az újraküldésnél minden ellenőrzés elölről kezdődik, beleértve a validációt is (az új submit esemény miatt)

Mint látható a műveletsor meglehetősen összetett, de nem átláthatatlan, és ami fontos, nem függ a túlzottan a dokumentum-szerkezettől. Ezzel azonban még nincs vége, hiszen a szerver oldalon is meg kell írni a szükséges PHP kódot is. Viszont akad egy komoly gond: a PHP beépített json_encode nevű függvénye, mely arra hívatott, hogy előállítsa a választ, hibákat tartalmaz, az ékezeteket nem kezeli megfelelően, így erre egy saját függvényt kell megírni (a PHP manualjában megtalálható a kód, így ezt nem írom le).

```php
<?php
    $form = new Registration();
    echo $form->GetJSONResult();

    //a Registration fenti függvénye:
    public function GetJSONResult() {
        $result = array(
            'error'=>$this->Error,
            'success'=>!$this->Error&&$this->Success,
            'message'=>$this->GetMessage()
        );
        return Settings::JSONEncode($result);
    }
?>
```

8.3 kód – Form-feldolgozás szerveroldalon PHP-val

Láthatóan nem igényel sokat, köszönhetően az osztály egyszerű felépítésének, és a JSON kódoláshoz is mindössze egyetlen kóddal kell kiegészíteni az osztályt (sőt, ezt a függvény nem nekünk kell leírnunk; a könyvtár beépítve tartalmazza).

További segítség

A kód már így is működőképes, de előfordulhat, hogy további segítséget szeretnénk nyújtani a kitöltéshez. Erre a legjobb példa, ha a regisztrációs űrlapon látható város elemre gondolunk. Magyarországon mintegy 3400 település található, melyeket egy adatbázisból könnyen le lehet kérdezni. Amikor a felhasználó elkezdi beütni a város nevét, egy bizonyos hossz után (például 3 karakter), azt elküldjük a webszervernek, ami válaszképpen az ezzel a betűkkel kezdődő városneveket dobja vissza egy listában, melyek közül a felhasználó kiválaszthatja a számára megfelelőt. De ugyanígy a város beírása után egyértelművé válik az irányítószám is, amit a rendszer automatikusan ki tud tölteni, akár a felhasználó helyett.

Erre szintén létezik kiegészítő, így nekünk már csak fel kell használnunk azt. A kiegészítő természetesen a jQuery-hez kapcsolódik, a neve pedig autocomplete. A kliens oldali felhasználásához szükségünk lesz a listát formázó CSS-re is, valamint az alábbi kódra:

```
$('INPUT[name=registration_varos]').autocomplete('ajax/city.php',{
    minChars:3,
    scroll: true,
    matchContains: true,
    scrollHeight: 200
});
```

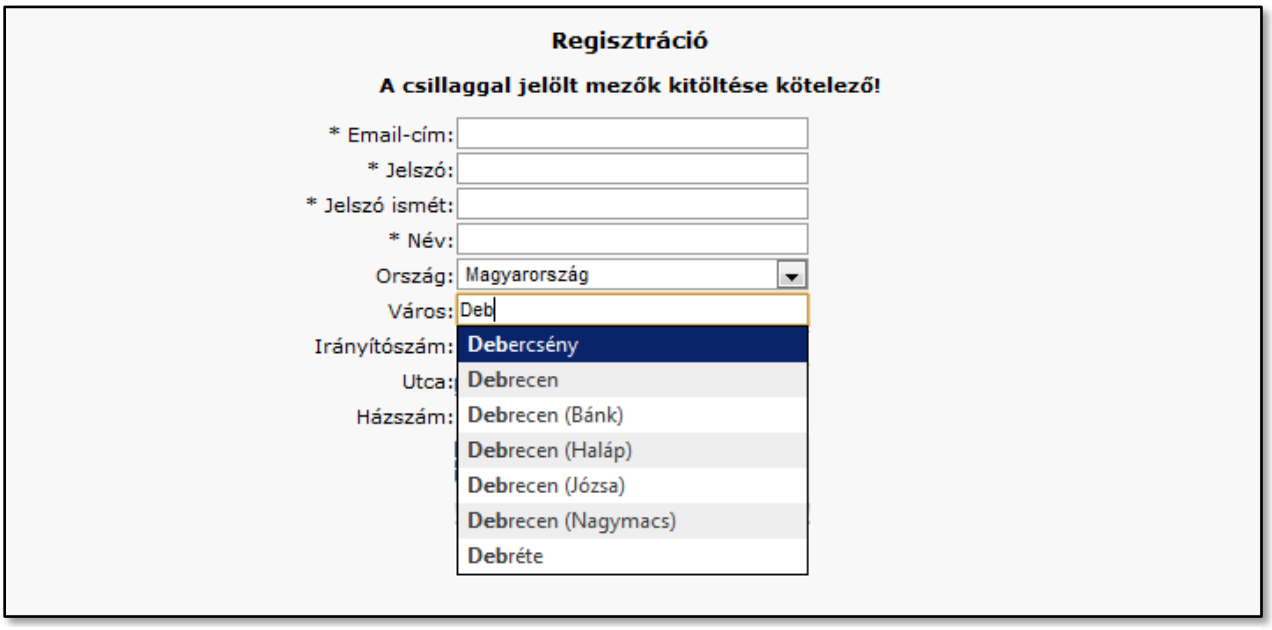
8.4 kód – Az autocomplete használata

A szerver oldalon az ajax mappa city.php fájljának tartalma:

```
<?php
$result = '';
if (isset($_GET['q'])&&$_GET['q']&&mb_strlen($_GET['q'],'UTF-8')>=3){
    try {
        $conn = MySQLConnection::GetInstance();
        $cities = $conn->Prepare("
SELECT DISTINCT(city) AS city
FROM szakdolgozat_city
WHERE city LIKE
ORDER BY city
LIMIT 20;")->Excute($_GET['q']);
        while ( $city = $cities->FetchAssoc() ) {
            $result.=$city['city']."\n";
        }
    }
    catch ( Exception $e ) {
        $result = 'Hiba történt!';
    }
}
echo $result;
?>
```

8.5 kód – Városok lekérdezése kereséssel szerveroldalon

A megfelelő feltételek teljesülése esetén valamilyen hasonló eredménnyel szembesülhetnek a felhasználók:



8.3 ábra – Városok listázása a kliens oldalon, Ajax alapú kereséssel

Mint látható, ez igen hasznos segítség a kitöltés során és segít elkerülni az esetleges félregépelési hibákat is. A kiegészítő szintén szabadon felhasználható és könnyen beépíthető, mindez pedig

illeszkedik a weblap rendszerébe is. Számos más, hasonló jellegű kiegészítő érhető el, melyek segítségével például egyfajta maszkot helyezhetünk el a mezőn, így a telefonszámok formátumát is egységessé lehet tenni. Mindenképpen érdemes körülnézni ezek után, sokszor igen jól jönnek a felhasználóbarát felület kialakítása során.

Összefoglalás

A fejezetben megismertük az Ajax technológia egy részletét, és az általa nyújtotta előnyöket. Megtanultuk felhasználni azt a saját céljainkra és barátságosabbá tettük a különféle formokat. A felhasználási lehetőségek száma gyakorlatilag végtelen, a kialakított rendszer pedig hathatós támogatást nyújt az ilyen lehetőségek kiaknázására anélkül, hogy túlzottan függene a Javascripttől.

Összefoglalás

A fejezetek folyamán megismerkedtünk a rendszertervezés néhány fontos részletével, a PHP fejlesztési eszközeivel, az AJAX technológia nyújtotta lehetőségek egy részével, illetve megnézhettünk néhány tervezési mintát is. Bár minden webalkalmazás egyedi a maga nemében, mégis vannak olyan részegységek, amik újrafelhasználhatóak más programoknál is – ezzel nem csak saját feladatunkon segítünk, hanem akik később átveszik a rendszert további fejlesztésre, vagy éppen apróbb javításokra, jobban át fogják látni az egészet, és így nem kell a korábbi
fejlesztőt felkeresni. Az általunk elkészített keretrendszer pedig biztosítja a folyamatos bővíthetőséget.

A rendszerhez tartozó, megjelenítést végző HTML fájlok még a 4.01-es verziónak felelnek meg, és nagymértékben alkalmazzák a CSS nyújtotta előnyöket és formázási eszközöket. A mai irányelvek ezt már megkövetelik, és a rendszer csak azért nem követi az XHTML szabványt, mivel a böngészők eltérően kezelik azt és a megrendelők elvárják, hogy a weboldal azoktól függetlenül, mindenhol és mindenkinél az elvárt módon jelenjen meg és működjön.

Az itt ismertetett eszközök csak egy részét képezik a teljes keretnek, az ismertető során nem került sor például a lapozás megvalósítására, amivel könnyebben limitálhatjuk az oldalon megjelenő, ismétlődő elemeket, illetve az url-kezelés ismertetője sem található meg itt. Utóbbi segítségével a különféle url-manipulációk végezhetőek el, paraméterek adhatóak hozzá, illetve vehetőek el, függetlenül attól, hogy permalinkes (azaz felhasználóbarát), vagy pedig hagyományos címekről van szó. Szintén nem került szóba a képkezelés megvalósításának mikéntje, hogy hogyan tudjuk azokat könnyen és gyorsan átméretezni, vagy vízjellel ellátni a keretrendszer segítségével, de ezek mind az alapeszközök részét képezik.

Végszóként szeretném megköszönni Adamkó Attila segítségét a szakdolgozat összeállításában, valamint a készítés moderálásában. Szintén szeretnék köszönet mondani azoknak a névtelen hozzászólóknak is, akik a különféle fórumokon másoknak segítettek problémájuk és így közvetve a fejlesztés folyamán felmerült gondok megoldásában is. Azokat sem szeretném elfelejteni, akik jelezték, hogy milyen változtatások szükségesek a rendszer könnyű kezelhetőségének érdekében.

Felhasznált irodalom

Ben Laurie, Peter Laurie: Apache, Kossuth Kiadó, 2001

Dan Livingston: CSS & DHTML webfejlesztőknek, Kossuth Kiadó, 1999

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Design Patterns - Elements of Reusable Object-Oriented Software, Addison-Wesley Professional Computing Series, 1998

George S. Schlossnagle: PHP fejlesztés felsőfokon, Kiskapu Kft., 2004

Joe Fawcett, Jeremy McPeak, Nicholas C. Zakas: Professzionális Ajax, Szak Kiadó, 2007

Jonathan Chaffer, Karl Swedberg: Learning jQuery – Better Interaction Design and Web Development with Simple Javascript Techniques, Packt Publishing, 2007

Kris Hadlock: Webalkalmazások fejlesztése Ajax segítségével, Kiskapu Kft., 2007

Martin Fowler: Refactoring, Kódjavítás újratervezéssel, Kiskapu Kft, 2006

Mike Andrews, James A. Whittaker: Hogyan törjünk fel webhelyeket – Webalkalmazások és webes szolgáltatások biztonsági vizsgálata, Kiskapu Kft., 2007

Neil Bradley: Az XML kézikönyv, Szak Kiadó, 2005

Síkos László: Stíluslapok a weben – CSS, BBS-Info Kft., 2005

Tervezési minták: http://www.fluffycat.com/PHP-Design-Patterns/, Larry Truet

Vikram Vaswani: XML and PHP, New Riders, 2002

Virginia DeBolt: HTML és CSS – Webszerkesztés stílusosan, Kiskapu Kft., 2005