Debreceni Egyetem
Informatikai Kar

Webalkalmazásfejlesztés PHP és XML alapokon

Témavezető:
Adamkó Attila
egyetemi tanársegéd

Készítette:
Gál József
III. programtervező informatikus

Debrecen
2008

Tartalomjegyzék

Bevezetés

Az internet a mai világ mindennapjainak egyik szinte nélkülözhetetlen eszköze. Amikor 1969-ben megjelent az ARPANET, amely tudományos és katonai hálózat is volt egyben, senki sem gondolta volna, hogy 40 év elteltével ez a világot behálózó rendszerré növi ki magát. Ennek megvolt a maga története, de 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 és módszerek alakultak ki használatának megkönnyítése érdekében.

A web kezdetén mindössze néhány honlap volt, és ezek 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. Aztán ahogy fejlődött a rendszer, 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 különféle 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 információt, de a tartalomhoz nem adhatott hozzá, ezen csak a fórumok és vendégkönyvek javítottak egy keveset. 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. 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, és már igen jelentős az XML alapú technológiák száma is.

Ezekhez már nem volt elegendő a szerveroldali támogatás, jelentős fejlesztés kellett a kliensoldalon is, főleg a böngészők terén. A 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 mintegy két 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.

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 hamarosan mindenhol elérhetővé válnak az ingyenes internetet elérhetővé tevő hotspotok. 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. Mindenesetre ez is egy lehetséges jövőkép.

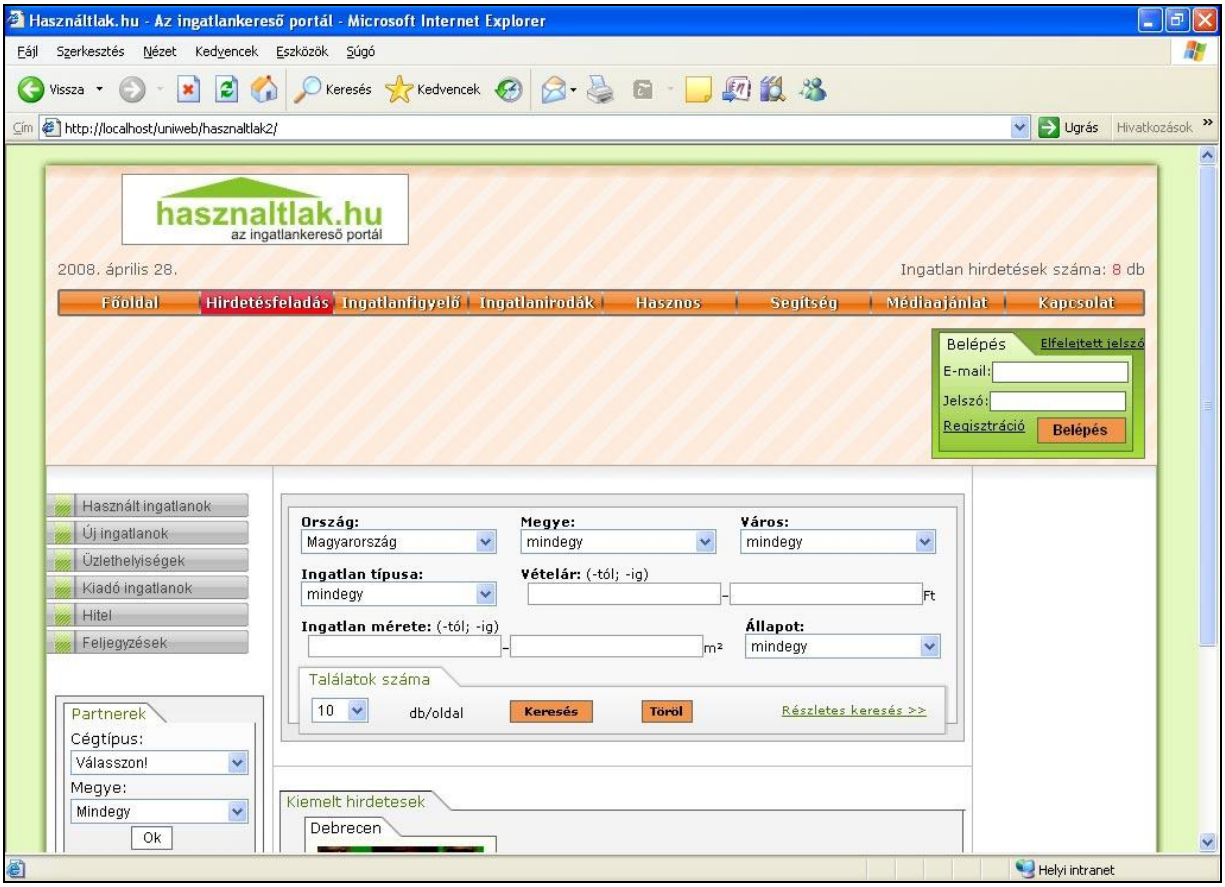
Ma gyakran találkozunk a videókat megosztó portálokkal, mint például a YouTube vagy a magyar Videa. Most még a minőség nem az igazi, gyakori a szemcsézettség, rossz minőség a nagy tömörítés miatt. A nagy felbontású videók kora azonban hamarosan elérkezik, köszönhetően az Adobe Flash fejlesztésének (korábban Macromedia Flash), valamint az új, Microsoft-féle SilverLight technológiáknak. Az élő HD média alapjai már megvannak, igazi forradalom várható hamarosan ezen a téren is.

Ezen szakdolgozat célja egy olyan honlap és webes alkalmazás elkészítése, amely felhasználja mind a PHP, mind pedig az XML és JavaScript nyújtotta lehetőségeket. Maga a honlap egy ingatlanhirdető- és kereső weblap lesz, melynek kialakítása során a fő szempont a gyors és biztonságos működés, valamint a könnyű kezelhetőség volt. A fejezetek során megismerkedünk a rendszerfejlesztés egyes lépéseivel, a PHP szerveroldali scriptnyelv objektum-orientált lehetőségeivel, az AJAX technológia alapjaival és betekintést kapunk a Google Maps világába is.

A dolgozat három fő részre tagolható. Az első 2 fejezetben egy alkalmazás tervezésének és fejlesztésének néhány lépésével ismerkedünk meg a követelmények összegyűjtésén és az UML diagramok használatán keresztül. Ezt követi a PHP bevezetője, ahol az 5-ös verzió által kínált eszközöket vesszük górcső alá és megismerünk néhány trükköt a hatékony rendszer

fejlesztése érdekében, valamint egy kis biztonságkezelést is tanulhatunk. Az utolsó fejezetekben arról lesz szó, hogy hogyan tehetjük interaktívvá a weblapot és hogyan segíthetjük elő a könnyű kezelhetőséget a Google Maps segítségével.

A szakdolgozat fejezeteiben nem ismertetem a teljes alkalmazást, viszont az említésre kerülő eszközök részletesen el lesznek magyarázva. A Google Maps támogatása remek lehetőséget nyújt egy későbbi továbbfejlesztésre és ezen keresztül az XML-t is érintjük majd. Az alkalmazás teljesen működőképes, kipróbálható változata jelenleg elérhető a http://gjtester.extra.hu/hasznaltlak/ weblapon.



1. ábra A leendő oldal

1

A rendszer tervezéséhez szükséges adatok összegyűjtése

Egy rendszer elkészítésének legelső és talán legfontosabb lépése a rendszerhez szükséges követelmények feltárása és elemzése. Sajnos ez korántsem olyan egyszerű, mint amilyennek hangzik, ugyanis egy felhasználó nem feltétlenül tudja megmondani, mit vár el a leendő programtól, vagy éppen hogyan is végzi napi teendőit. Ezért fontos, hogy lehetőleg tisztázzuk a megrendelővel, mire is készül a program, illetve mit várunk el majd tőle. Gyakran egy megvalósíthatósági tanulmány is megelőzi ezt a lépést, ami azt a célt szolgálja, hogy egy projekt mennyiben segíti elő egy cég fejlődését, illetve egyáltalán elvégezhető-e a jelenlegi technológiával adott pénzügyi kerettel adott időn belül.

A mi esetünkben a feladat viszonylag egyszerű: egy olyan honlap elkészítése, melyen felhasználók és különféle partnerek közzétehetik lakás- és ingatlanhirdetéseiket, ezek kereshető formában megjelennek és szerkeszthetőek is. Az ilyen jellegű követelmények minden esetben a megrendelőtől származnak, a megvalósításhoz viszont szükség lesz továbbiak meghatározására is. Így például egy hirdetés elhelyezéséhez tudnunk kell, ki tölti fel azt. Ez a ma elterjedt regisztrációval, felhasználónév-jelszó megadásával oldható meg a legjobban. Ezen felül a keresést és a feltöltést érdemes jól elválasztani, tehát bárki kereshet az oldalon, de feltölteni csak regisztrált személyek tudnak. Az adatbázis karbantartásához, például a feltöltött hirdetések moderálásához és bannerek elhelyezéséhez szükség lesz egy adminisztrációs felületre is, ami csak a helyi hálózatból lehet elérhető (erre persze a másik megoldás, ha ezt külső programmal oldjuk meg).

Az előbb említett dolgok az úgynevezett funkcionális követelményekhez tartoznak, viszont rendszerünknek más megkötéseknek is eleget kell tenniük. Ilyen például a nap 24 órájában való rendelkezésre állás, rövid válaszidő és hogy a program minél kevesebbszer hibázzon, de ide tartoznak még a biztonsági kérdések vagy a tanulhatóság is. Ezeket összességében nem

funkcionális követelményeknek nevezik, és mivel ezek a rendszer egészére vonatkoznak, gyakran nevezik őket rendszerkövetelményeknek is.

Egy harmadik kategória a szakterületi követelmények nevet viseli. Ide azok a kérdések tartoznak, amik az adott cég vagy a program szakterületét érintik. Ezek egy honlap esetében nem mindig vannak kihatással a programra, viszont érdemes figyelembe venni őket. Általában speciális előírások, igények és kezelési módok tartoznak ebbe a tárgykörbe. Ide vehetjük a rendszer architektúrájának egy részét, például a regisztrációt. Ez ugyan nem tartozik hozzá egy-egy cég profiljához, de az esetek nagy részében a weblapok alapját képezheti.

Követelmények feltárása

Ebben a szakaszban nagy szerep jut azoknak a személyeknek, akik valamilyen módon kapcsolatba kerülnek a rendszerrel, az úgynevezett kulcsfiguráknak. A követelmények feltárása az ezektől az emberektől beszerzett információk feltárását és elemzését jelenti.

Ezzel viszont elég sok probléma adódhat. A kulcsfigurák ugyanis általában nem tudják megmondani, mit várnak el egy rendszertől, vagy legalábbis nehezen, esetleg egyáltalán nem tudják megfogalmazni ezeket. Sőt az is elképzelhető, hogy olyan elvárások merülnek fel, amik ellentmondásosak. Ennek feloldása érdekében érdemes felállítani egy fontossági sorrendet, majd ebből azokat kiválasztani, amik bekerülhetnek a végső rendszerbe.

A legnehezebb probléma azonban még mindig a kommunikáció gondja a kulcsfigurák és a fejlesztők között. A gondot főleg az okozza, hogy minden szakterületnek saját szlengje van, amit egy külső szemlélő nem biztos, hogy megért vagy rossz esetben teljesen félreért. Ezt általában úgy szokás feloldani, hogy az informatikus tájékozódik az adott szakterületről, különös tekintettel az alapvető folyamatokra és terminológiára.

Gyakran felmerülnek külső körülmények is, például közbeszól egy politikai döntés. Politika alatt itt értendő a nagybetűs politika, de ugyanúgy az adott cég üzletpolitikája is. A gazdasági- és a törvényi környezet változhat egy rendszer készítése folyamán, és ez gyakran erőteljes kihatással van a rendszer egészének működésére.

Egy követelményrendszer elkészítésére több módszer is létezik, ezek egyike az utóbbi időben nagyon népszerű forgatókönyv-technika. Előnye, hogy viszonylag közel áll a felhasználóhoz, megrendelőhöz, így a kulcsfigurák könnyebben megértik annak tartalmát. Egy forgatókönyv a rendszerrel való interakciókat írja le, ideértve a kezdeti rendszer-állapotot, az eseményeket, amik normális használat közben történnek, a kivételeket (amik a megszokott használattól eltérő események), párhuzamos tevékenységeket, illetve a rendszer végállapotát az interakciók végén. A forgatókönyvnek egyfajta vizuális megjelenítést kölcsönöznek a használati eset (use case) diagramok, ezekről a következő fejezetben részletesebben is szó lesz. Sokszor szükség

van egy fogalomszótár összeállítására is, hogy az alapvető fogalmakat tisztázni lehessen, az adott szavakat és kifejezéseket hogyan kell értelmezni a program esetén.

A mi esetünkben a programnak a következő, általános követelményeknek kell eleget tennie:

- szép felület, a szemet zavaró színek és zavaró funkcióelrendezések kerülése
- érthető tájékoztatás a műveletek sikerességéről, hiba esetén a felhasználó minősítésétől függő értesítés nyújtása
- a műveletek elvégzéséről illetve a felhasználók be- és kilépéséről végzett rendszeres naplózás
- a biztonságot igénylő adatok biztonságos csatornán történő továbbítása

Ezek persze olyan követelmények, amik a legtöbb ma használt programban ugyanígy megtalálhatóak. Viszont ezekre nekünk is érdemes odafigyelni, bármennyire is csak egy honlapról van szó. A hangsúly a szép felületen van, ami egy weboldal alapvető sajátossága, a leendő vevőket, látogatókat egy könnyen átlátható, jól strukturált oldal jobban megragadja, mint egy tarka, összedobált lap.

Az előbb említett funkcionális követelmények a mi programunknál a következőek lesznek:

- lakáshirdetések elhelyezésének lehetősége
- felhasználók biztonságos kezelése, regisztrációja és nyilvántartása
- hirdetések karbantartása, ideértve a 2 hónapnál régebben frissítettek törlését
- partnerek felvitele, nyilvántartása és módosíthatósága
- hírlevél küldése

Ezek a követelmények egész biztosan a megrendelőtől jönnek, ha nem is pont ilyen formában megfogalmazva.

Erre a programra vonatkozó rendszer- vagy funkcionális követelmények a teljesség igénye nélkül a következőek:

- PHP 5 alkalmazása szerveroldalon
- HTML 4.01 szabvány használata
- CSS alkalmazása a formázáshoz
- rövid válaszidő, gyors megjelenítés
- kisméretű fájlok (képek, HTML oldalak, stb.) használata

A forgatókönyv

Az előbb felsorolt követelményeket valamilyen formában rendezni kell, ennek egy módja a korábban már említett forgatókönyv. A felhasználók már csak azért is szeretik, mert könnyen érthető és egyszerűen értelmezhető.

Persze egy forgatókönyv esetén lehetetlen leírni egy kezdeti állapotot teljes egészében, ezért érdemes címszavakba szedni az egyes interakciókat, majd leírni annak szabályos menetét, illetve a kivételeket és azok kezelését. A végállapotot általában egy adott művelet eredménye jelzi. Az alábbi sorokban megtaláljuk a rendszerrel végezhető műveleteket és azok leírását.

Regisztráció

Egy hirdetés elhelyezéséhez regisztráció szükséges. A regisztráció során a felhasználónak meg kell adnia az email-címét, kétszer a jelszavát, nevét, valamint opcionálisan lakhelyét. Ezek az azonosításhoz szükségesek. Amennyiben az adott email-cím már szerepel az adatbázisban, úgy a rendszer hibajelzést küld a foglaltságról. Az email-címnek megfelelő formájúnak kell lennie, ha ez nem teljesül, szintén hibajelzés történik. A jelszó hossza legalább 5 karakter legyen, ezt kétszer kell megadni a megerősítés végett. Ha nem egyeznek, szintén hibajelzést kell adnunk. Ezeken túl a regisztrálónak el kell fogadja az Általános Szerződési Feltételeket, ennek hiányában a regisztráció érvénytelen. Feliratkozhat a hírlevélre is, így rendszeres értesítéseket kaphat az esetleges rendszerben történő változásokról. Sikeres regisztráció esetén a felhasználó bekerül az adatbázisba, illetve egy megerősítő email-t küldünk a felhasználónak. Az email tartalmaz egy linket, amin jóváhagyhatja az adatokat és regisztrációt, ennek hiányában a felhasználó egy napon belül törlésre kerül. A jóváhagyás után a felhasználó bejelentkezhet a rendszerbe.

Cég felvitele

A rendszerben szerepelhetnek különböző cégek adatai. Céget csak a rendszeradminisztrátor tud felvinni a rendszerbe. A cégek két csoportba sorolhatóak: partnerek és viszonteladók. Partnerek nem tölthetnek fel hirdetést, nevük viszont kereshető a főoldalon és adataik is itt jeleníthetőek meg. Viszonteladók számára több hirdetés felvitele lehetséges, ez megegyezés szerint kerül bejegyzésre, valamint az általuk felvitt hirdetések csak bizonyos ideig szerepelhetnek a rendszerben, frissítéstől függetlenül, ennek hosszabbítása is lehetséges. Céget felvinni az adminisztrációs felületről a cég e-mail-címének, a cég és a cégvezető nevének, telefonszámának és e-mail-címének, valamint fellelhetőségi helyének megadásával lehet.

Bejelentkezés

Egy látogató regisztrált és megerősített email-címének és jelszavának megadása után bejelentkezhet a rendszerbe. Amennyiben az email-cím–jelszó páros nem szerepel az adatbázisban, úgy a felhasználó értesítést kap a sikertelenségről. Egy bejelentkezéshez maximum 3 próbálkozás lehetséges, a harmadik sikertelen kísérlet után a felhasználó negyed órára blokkolva lesz, ezzel is gátolva a feltörést. Sikeres belépés után lehetősége van hirdetést feltölteni, ami egyszerű felhasználó esetén 1 hirdetést jelent, míg viszonteladó esetén a megállapodás szerinti mennyiség felvitelét jelenti. A felhasználó mindaddig bejelentkezve marad, amíg rá nem kattint a kijelentkezésre, vagy be nem zárja a böngésző ablakát.

Hirdetés feltöltése

Egy regisztrált és bejelentkezett felhasználó tölthet fel hirdetést, viszonteladó esetén a megállapodástól függő mennyiséget, regisztrált felhasználó pedig egyet. A feltöltéshez szükség van a hirdetéshez szükséges adatok megadására, melyek: ország, város, utca, házszám, az ingatlan típusa, állapota, ára, mérete, szobák száma, építés éve, szükség esetén a felszereltség valamint a különféle extrák, például klíma, padlófűtés vagy kerti tó. Lehetőség van maximum 5 kép feltöltésére is, melyek formátuma jpg, gif vagy png lehet. A rendszer a feltöltés során ezeket szükségszerűen átméretezi és hozzárendeli a hirdetéshez. Sikeres feltöltés esetén a hirdetés megjelenik a rendszerben. Új hirdetés feltöltése, vagy meglévő esetén e-mailt kell küldeni a rendszeradminisztrátornak, aki azt szükség esetén letilthatja vagy engedélyezheti.

Jegyzetek készítése

A felhasználónak lehetősége van a neki megtetszett hirdetések elektronikus feljegyzésére is a munkamenet idejére. Jegyzetet a keresés után a találati listában szereplő elemek feljegyzés ikonjára való kattintással lehet készíteni. Egy jegyzet rövid leírást tartalmaz a hirdetésről, mely magában foglalja az ingatlan árát és helyét, részletesebb adatokhoz rá kell kattintani a feljegyzéshez tartozó képre.

Adminisztráció

Lehetőség van a hirdetések, bannerek, statikus oldalak és felhasználók adminisztrálására, ehhez külön jogosultság szükséges, melyet a rendszergazda tud kiosztani. Az adminisztrációs oldal csak a cégen belüli hálózatról érhető el, kívülről nem látható.

Hirdetésmoderálás

Egy felhasználó által elhelyezett hirdetésről az adminisztrátor e-mailes értesítést kap, valamint lehetősége van hirdetések moderálására (letiltására) és törlésére is az adminisztrációs felületen keresztül. Hirdetés letiltása vagy törlése esetén az érintett felhasználó értesítést kap az általa megadott e-mail címre. Törlés esetén visszavonási lehetőség nincs.

Felhasználók kezelése

Lehetőség van cégek felvitelére, regisztrált felhasználók letiltására/aktiválására, valamint törlésére is. Törlés esetén a felhasználó által felvitt hirdetések is eltávolításra kerülnek, visszavonásra nincs mód. A felhasználó minden esetben értesítést kell kapjon.

Keresés

Biztosítani kell a feltöltött hirdetések keresésének lehetőségét is. Az oldalon bárki kereshet a hirdetések között, ha megadja a számára fontos paramétereket, amik a típustól függően a következőek lehetnek alaphelyzetben: a hirdetés helye ország, megye és város szerint, az ingatlan típusa, a vételár minimuma és maximuma, az ingatlan mérete, valamint annak állapota. Részletes keresésnél a felhasználó megadhatja, hogy mikor épült ingatlanokra kíváncsi, valamint a szobák számát és a különféle extrákat (klíma, riasztó, elektromos kapu,

stb.) is. Ha nincs ezeknek megfelelő találat, úgy a program ezt is közli a felhasználóval, ellenkező esetben a találatokat listázza, oldalakra bontva.

Természetesen ez sokat segít majd a későbbiekben, segítségével feltárhatjuk az alrendszereket és az egyéb követelményeket is. Érdemes megvitatni a megrendelővel, hogy ezek valóban megfelelnek-e neki, mert ilyenkor még könnyű változtatni, az elkészült programot azonban már jóval nehezebb módosítani. Utóbbira azonban a korábban említett nehézségek miatt nagy valószínűséggel szükség lesz, erre is fel kell készülnünk.

Az itt megadott követelmények persze nem adnak teljes áttekintést a működésről, csak arról szolgálnak információval, hogy hogyan működjön a rendszer normális és esetleges extrém körülmények között. A forgatókönyv megrendelőnek és az informatikusnak egyaránt szóló leírás, egyszerű emberi nyelven megfogalmazva, informatikai leírásoktól mentesen. Erre azért van szükség, mert a megrendelő általában nem ért az informatikához és így lényegesen könnyebb a kommunikáció. A forgatókönyv tartalmazza azokat az információkat, amik alapján letehetjük a rendszer alapjait és nekikezdhetünk a tényleges tervezéshez is.

A forgatókönyvet természetesen később is elő kell venni, hiszen ez alapján ellenőrizhetjük, hogy teljesíti-e az ott meghatározott követelményeket a program. Ha a megrendelő valamiért elégedetlen a végeredménnyel, de elismeri, hogy az itt leírtaknak megfelel a program, akkor egyben azt is elismeri, hogy a felelősség őt terheli emiatt. Ha azonban mi hibáztunk, úgy feltétlen javítanunk kell a programon, vagy szükség szerint megegyezni a megrendelővel a módosításról. Egy jól strukturált rendszernek ezért mindenképp támaszkodnia kell erre a dokumentációra, valamint a szerkezetnek megfelelően bővíthetőnek kell maradnia a későbbi fejlesztések és az állandó elégedetlenségek miatt.

Összefoglalás

A fejlesztés talán egyik legnehezebb része a követelmények feltárása, jó megközelítéssel viszont megfelelő rendszert állíthatunk össze a megrendelő segítségével a leendő program számára. Ehhez azonban nélkülözhetetlenek a megfelelő minőségű egyeztetések, megbeszélések a specifikáció összeállításához.

Ebben a fejezetben megismerkedtünk a követelmények fogalmával és egy technikával, amivel rendszerezhetjük is azokat. Ha ismerjük a fejlesztendő rendszerhez szükséges kikötéseket, elkezdhetjük összeállítani a tényleges rendszer működését, legalábbis az architektúrát. Az architektúra a rendszer leglényegesebb része, ha a felvetés rossz, az alkalmazás sem lehet teljesen működőképes és így nem láthatja el feladatát. A következő fejezetben elkészítjük az itt leírt forgatókönyv alapján a rendszer használati eset diagramját, valamint megismerkedünk az osztálydiagramokkal is.

2

UML diagramok használata a tervezéshez

Az UML egy objektum-orientált szemléletre épülő elemző- és tervező eszköz. Segítségével elkészíthetjük az alkalmazás vázlatát, és ma már segítségünkre vannak olyan eszközök, amivel akár az implementációig is eljuthatunk. Természetesen ez nem alkalmas a teljes kód elkészítésére, de sok munkától megkíméli a programozót azzal, hogy az egyes metódusok, osztályok váza már rendelkezésre áll. Az UML-nek több különböző, más-más célra alkalmas diagram-típus van, segítségükkel a modellezett alkalmazásra különböző módon tekinthetünk. Vannak átfedések a diagramok között és lehet, hogy ugyanazt ábrázoljuk, pusztán más formában.

Ebben a fejezetben megismerünk néhány hasznos diagram-típust, melyek segítségével az előző fejezetben felvázolt forgatókönyvet jeleníthetjük meg, valamint dokumentálhatjuk a rendszert. Egy-egy diagram felrajzolásához számos eszköz áll rendelkezésre, ezek közül mi az ingyenes NetBeans plugint fogjuk használni. Az elsőként az úgynevezett use case, vagyis használati eset diagrammal ismerkedünk meg, majd a dokumentáláshoz is használható osztálydiagramot vesszük górcső alá.

Használati eset diagram

A használati eset, azaz a use case diagram segítségével a felhasználók rendszerrel szemben támasztott követelményeit jeleníthetjük meg vizuálisan, a részletekről, működésről, a folyamatokról viszont nem szolgál információval. Egy ilyen diagram elkészítéséhez szükségünk lesz az előző fejezetben elkészített forgatókönyvre, illetve néhány további lépésre:

- a rendszerhez köthető szereplők és külső eszközök, vagyis aktorok összegyűjtése
- különböző használati esetek összegyűjtése (forgatókönyv alapján)
- az elemek kapcsolata egymással és az aktorokkal

Az aktor jelentése legjobban a szerepkör kifejezéssel fejezhető ki. Lehet egyszerű felhasználó, hardver vagy akár egy adatbázis-kezelő rendszer is. Aktort általában egy pálcikaemberrel jelölünk, aláírva az elnevezését.

A mi rendszerünk öt aktort különböztet meg, ezeket a 2.1-es ábrán láthatjuk.

2.1 ábra Aktorok

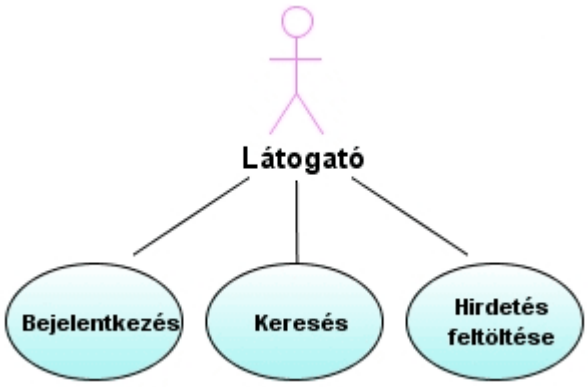
Ezek az aktorok más-más célból, más interfészen keresztül más eredményt várnak. Itt felmerül, hogy milyen funkciók kapcsolódnak az egyes szereplőkhöz. Látogatónak tekintünk minden embert, aki meglátogatja az adott honlapot, például azért, hogy keressen egy számára megfelelő ingatlant. A felhasználó jelenti a honlapon regisztrált látogatókat, a partner a korábban említett céges ügyfeleket, akiket az adminisztrátor tud felvinni. Az adminisztrátor a rendszerért felelős embereket szimbolizálja, azokat, akik a hirdetéseket moderálják, kezelik a felhasználókat. A MySQL aktor az adatbázis-kezelő rendszert jelenti, szerepe igen sokrétű, többek között ez biztosítja az adatbázishoz való hozzáférést is.

Az egyes használati esetek jelzik azokat a kapcsolódási pontokat, mely során egy aktor kapcsolatba kerül a rendszerrel, eredményt vár el, vagy szolgáltat a rendszer számára. A használati esetet a use case diagramokon ovális jelzi, ebbe kerül annak neve. Néhány példát láthatunk a 2.2-es ábrán.

2.2 ábra Használati esetek

Egy használati eset és egy aktor között az egyetlen kapcsolat, ami fennállhat, az az asszociáció, aminek a use case diagram esetén vett jelentése, hogy az aktor kommunikál az adott esettel. Jelölése a használati eset és az aktor közötti egyszerű vonallal történik, ezt a 2.3-as ábrán láthatjuk.

Az egyes funkcióknak lehetnek olyan részei, amik részként tartalmaznak egy másikat. Ilyen például a bejelentkezés és a regisztráció viszonya, aminél a bejelentkezéshez szükség van egy érvényes regisztrációra. Ezeket «include» típusú nyilakkal jelöljük, ami az esettől a tartalmazott eset irányába mutat. Az, hogy melyik megelőző eset mikor játszódik le, az adott eset dönti el.



2.3 ábra Asszociációs vonalak

Az egyes eseteknek lehetnek leszármazottai, amik az adott eset funkcionalitását bővítik valamilyen formában. A mi rendszerünkben ilyen az adminisztráció és a hirdetések karbantartásának viszonya. Érthető, hogy az adminisztráció elég tág fogalom, a hirdetéskezelés már viszont ennek jelentős szűkítése. Az ilyen viszonyt «extends» típusú nyilakkal jelöljük, amik a leszármazottaktól az őseset felé mutatnak.

A harmadik, és egyben utolsó általunk tárgyalt viszony az általánosítás/pontosítás fogalma, aminek a jelentése megegyezik az objektumorientált paradigma fogalmával. Az ilyet akkor használjuk, ha az actornak léteznek lényeges alváltozatai, például a felhasználó és az adminisztrátor esetén a felhasználó feljogosítást (authorizációt) követően válhat adminisztrátorrá. Jele egy egyszerű, az ős felé mutató nyíl.

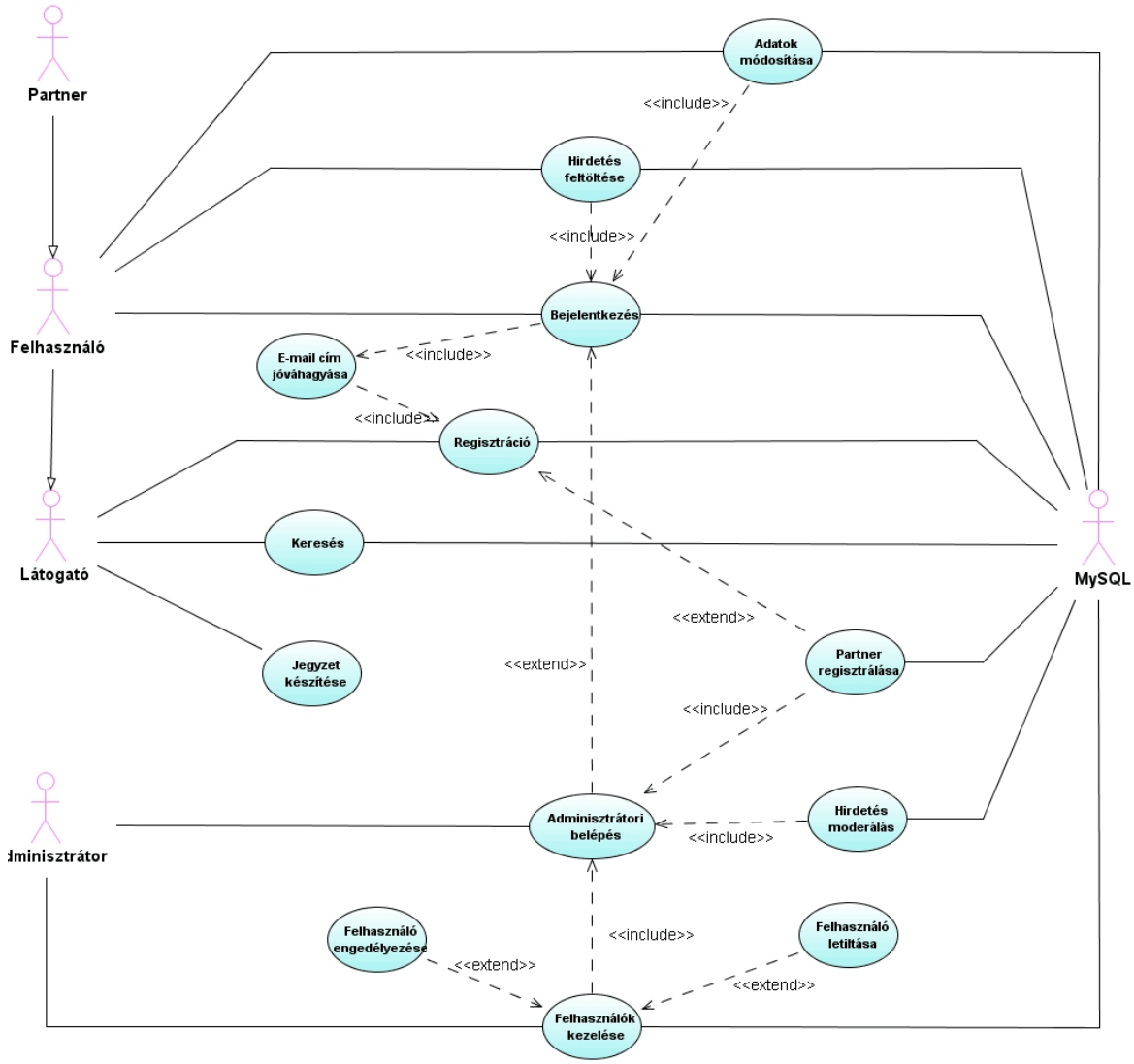
A honlaphoz tartozó használati eset diagramot ezek alapján már összeállíthatjuk, az eredmény a 2.4-es ábrán látható.

Az osztálydiagram

Az osztálydiagramok elsődleges célja a tervezés elősegítése, ugyanakkor a dokumentálásnál is hasznát vehetjük. Alapját az objektumorientált paradigma központi fogalma, vagyis az osztály képezi. Az UML nem foglalkozik implementációval, pusztán az osztály vázát modellezi, azt az interfészt, amin keresztül elérhetjük az osztályból példányosított objektumok tulajdonságait, metódusait.

Egy osztálydiagram esetén az osztály ábrázolása három részből áll: az osztály neve, az adattagok, végül pedig a metódusok. Ha az osztály neve dőlt betűs, akkor az azt jelenti, hogy absztrakt osztályról van szó, vagyis nem példányosítható. A metódusoknál hasonlóan absztrakciót fejez ki a dőlt betűvel szedés. Az adattagok alakja a következő mintával egyezik meg:

```
[láthatóság] adattagNév [: típus] [multiplicitás] [=alapÉrték]
```



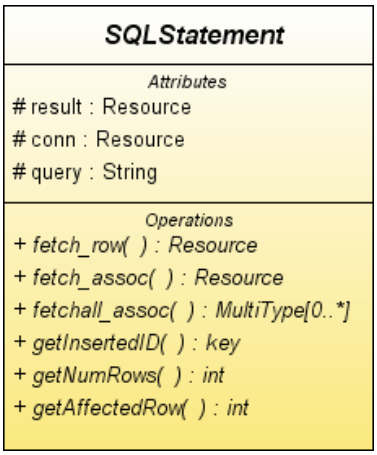
2.4 ábra Használati eset diagram

A láthatóság mind adattagok, mind metódusok esetében négyféle lehet: privát (-), csomagszintű (~), publikus (+), valamint védett (#). Ezek megfelelnek az OO paradigma bezárási szintjeinek, így nem érdemes részletezni őket. A PHP-ben a csomagszintű láthatóság nem értelmezett, így azzal a későbbiekben nem fogunk találkozni. A típus az adattag által felvehető értéktartományt jelenti, a multiplicitás pedig azt fejezi ki, hogy hány érték tartozhat az adattaghoz. Megadható egy alapérték is, ez például üres konstruktornál kaphat jelentőséget. Lehetőség van osztályszintű, azaz statikus tulajdonságok és tagfüggvények ábrázolására is, ezeket általában egy aláhúzás jellel különböztetjük meg a többi tulajdonságtól.

A metódusok definíciója osztálydiagram esetén a következő:

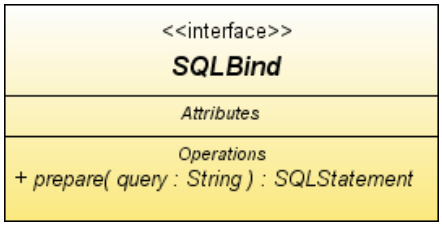
```
[láthatóság] név([paraméter : típus]*) : visszatérésiTípus
```

A paramétereknél megadhatjuk a paraméter nevét, illetve a típusát (értéktartományát). A paraméterlista hossza 0 vagy több lehet. A visszatérési típus a metódus által visszaadott adatok típusát definiálja, ez lehet void is, ha nincs visszatérési érték. A 2.5-ös ábrán az adatbázis-lekérdezések eredményét egységbe foglaló absztrakt osztályt láthatjuk.



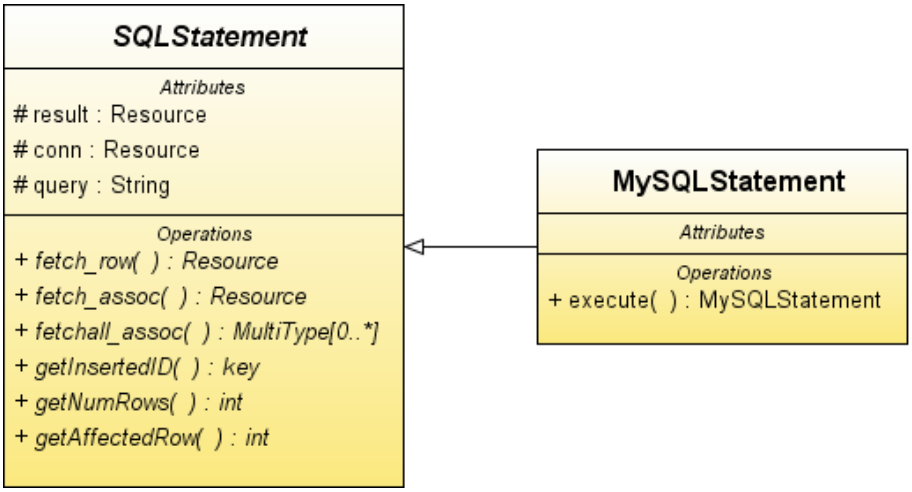
2.5 ábra Az SQLStatement osztály

Lehetőségünk van interfészek meghatározására is, ilyenkor annak neve felett egy «interface» jelző lesz látható, minden másban megegyezik az osztálydiagram alakjával. A különbség PHP esetén annyi, hogy interfészben nem lehetnek adattagok, míg Java-nál találhatunk osztályszintű, nem módosítható elemeket (állandókat). A 2.6-os ábrán a BindSQL-hez tartozó interfészt láthatjuk. Amelyik osztály ezt megvalósítja, annak a példánya ezáltal tud majd BindSQL utasításokat végrehajtani.



2.6 ábra Az SQLBind interfész

Ha egy osztályt származtatunk egy ősosztályból, azt az úgynevezett generalizációval ábrázolhatjuk az osztálydiagramban. Generalizációt a leszármazott osztályból az ősosztály felé mutató, háromszögben végződő nyíl jelenti. Interfészeknél az interfész felé mutató szaggatott nyíllal jelezhetjük, ha egy osztály megvalósítja az adott interfészt. A 2.7-es ábrán a rendszerhez tartozó MySQLStatement osztályt és annak ősosztályát, a korábban már látott SQLStatement-et láthatjuk, valamint az öröklési viszonyt ábrázoló generalizációs nyilat.



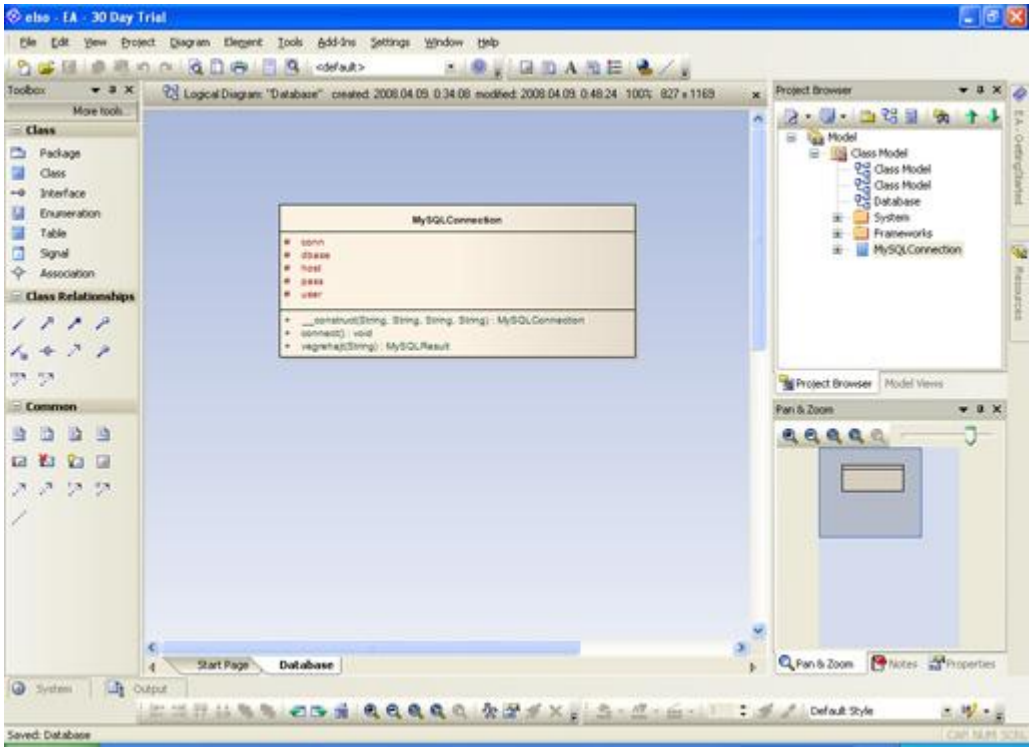
2.7 ábra Öröklés

Egy rendszerben az objektumok nem feltétlenül önállóan, hanem más osztályokkal együttműködve végeznek el feladatokat. A kommunikációs vonalakat, amin keresztül az egyes objektumok majd információt cserélnek, asszociációknak nevezzük, az osztálydiagramban pedig asszociációs vonalakkal jelezzük. Az asszociációknak általában több fajtája előfordul egy-egy rendszerben, ezek közül a leggyakoribbak a bináris asszociációk, aggregációk és kompozíciók. Az egyes összeköttetések általában kifejezik, hogy egy osztály kapcsolatban áll egy másikkal, része annak, vagy felel az adott osztály objektumainak élettartamáért. Egy harmadik kapcsolattípus is meg szokott jelenni, az úgynevezett függőség. Függőség nem csak osztályok, hanem más modellelemek között is értelmezhető. Mivel ezen szakdolgozatnak nem célja a teljes UML rendszer leírása, így részletes leírásukra most nem kerül sor.

Ha PHP-hez szeretnénk alkalmazni ezeket a diagramokat, komoly problémákba ütközünk, ugyanis a jelenlegi fejlesztéshez jól használható eszközök nagy része nem támogatja az együttműködést. Ez főleg annak köszönhető, hogy a PHP korábbi változataiban az OO eszközrendszer lényegében csak volt, de nem használták széles körben. Akad néhány program, amik segítenek a kódgenerálásban, azonban ezek vagy nem ingyenesek, vagy nem működnek megfelelően. Persze akad kivétel, ezek közül az Enterprise Architect nevű program érdemel nagyobb figyelmet, ugyanis ez képes együttműködni az Eclipse fejlesztői környezettel, főleg a kódgenerálásban. Sajnos a program nem ingyenes, de 30 napos határidővel kipróbálhatjuk szolgáltatásait. Kezelése szinte magától értetődik, de nagyon jó dokumentációt találunk a termék honlapján is PDF és HTML formátumban egyaránt. A szoftver próbaváltozata letölthető a http://www.sparxsystems.com/products/ea.html honlapról az Eclipse támogatással együtt. A kezelőfelületről a 2.8-as ábrán láthatunk egy részletet.

Bár Eclipse környezethez is vannak próbálkozások UML diagramok terén, ezek nem használhatóak olyan mértékben, mint a NetBeans plugin. A NetBeans 6.0-s változatában már megtalálható a kódgenerelás, ez azonban csak Java programokhoz használható és a jelenleg még béta stádiumban lévő PHP környezet is hagy kívánni valót maga mögött. Remélhetőleg

ez a közeljövőben változni fog és sikerül egy megfelelően támogatott rendszerhez hozzájutniuk a PHP fejlesztőknek is.



2.8 ábra Az Enterprise Architect kezelői felülete

Összefoglalás

Az objektumorientált szoftverfejlesztés ma már szerves részét képezi az UML diagramok használata, valamint a segítségükkel történő fejlesztés. Persze nem mentesít minket a kódolás alól, de jelentősen az előre megtervezett osztály-szerkezetből generált architektúra jelentősen megkönnyíti a munkánkat a későbbiekben.

A fejezetben röviden megismerkedtünk két UML diagram-típussal, illetve megbeszéltük, hogyan kell összeállítani egy-egy ilyen modellt. Az itt elkészített osztály- és use case diagramok később hasznunkra lesznek a rendszer fejlesztése folyamán.

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, hasonlóan a static kulcsszóhoz, mellyel osztályszintű eszközöket készíthetünk. Természetesen ezek keverése is megengedett, viszont a sorrend nem mindegy: előbb a láthatóság, majd szükség esetén az osztályszint beállítása.

Egy újabb kulcsszó a final, amivel az öröklés során találkozhatunk. Ennek segítségével olyan metódusokat írhatunk, melyek a leszármazott osztályban nem bírálhatóak felül. A későbbiek folyamán azonban ezt valószínűleg nem fogjuk alkalmazni.

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 az is, hogy megadhatjuk a paraméter osztályát, ha azt szeretnénk, hogy biztosan ilyen objektumot adjunk át a függvénynek. Ha nem így teszünk, és valamilyen más típusú elemet adunk át, úgy fatális hiba (fatal error) keletkezik és a futás leáll.

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 Egy egyszerű osztály PHP-ban

Ezen rövid bevezető után lássuk azokat a speciális eszközöket, melyek jelentősen könnyítenek a fejlesztésben: az ún. mágikus függvényeket. Ide tartoznak azok a különleges függvények, amik két aláhúzás jellel kezdődnek, mint az előbb említett konstruktor függvény 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 lényeges, a lezárást elősegítő eszközről, a destruktorról. A destruktor akkor jut szerephez, mikor az objektum jelentősége elvész. 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 A destruktor használata

__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 valamilyen úton. 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, azaz visszaállítja a szerializáció során

elveszett adattagok értékét. Ez a függvény akkor kerül meghívásra, ha az unserialize függvény hívjuk explicit vagy implicit módon.

```
<?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 Objektum-szerializáció és deszerializáció

__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.

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) 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.

Tervezési minták használata

Gyakran futunk olyan helyzetbe egy-egy program elkészítése során, amivel feltételezhetően mások már is találkoztak. Ezeknek a problémákra többnyire létezik egy általános elfogadott megoldási módszer, ún. minta, ami segít a megfelelő és természetesen újrafelhasználható

osztályszerkezet elkészítésében. A PHP esetén ezek a programnyelv adottságainak megfelelően módosultak, de az alapelvek megmaradtak.

Az egyik ilyen tervezési minta, az Illesztő (Adapter, Adaptor) célja, hogy egy objektumhoz adott felületen férjünk hozzá. Ennek az a haszna több más mellett, hogy egy adott eszközt lecserélhetünk teljesen másra anélkül, hogy a program többi részét módosítanunk kellene. Így például lehetővé válik, hogy míg a rendszert MySQL adatbázis-támogatás mellett tervezzük, addig a végleges változat PostgreSQL mellett fog működni (ennek persze vannak az SQL szintaxisra vonatkozó részletei is, de ezektől most tekintsünk el). Ezt a többalakúság biztosítja. A weboldal adatbázisának eléréséhez ezt a mintát is alkalmaztam, így egy olyan eszközhöz jutottam, ami más oldalak készítésénél is nagyon hasznos lehet.

Persze sokszor kényelmetlen egy-egy adatbázis-objektumhoz megadni a szükséges paramétereket (felhasználónév, jelszó, szerver címe és az adatbázis neve), ezért még egy további mintát is felhasználtam. Ez az ún. Sablon minta, ami egy olyan osztályt ír le, ami az öröklést felhasználva módosítja az ősosztályt úgy, hogy teljesebbé teszi, kiegészíti azt. Így elegendő egy központi helyen tárolni az oldalhoz szükséges információkat (például az adatbázis-elérés paramétereit), és ha az adatbázis-leíró osztály hozzáfér ezekhez, akkor máris rengeteg időt és vesződséget spóroltunk meg. A Sablon minta segítségével létrehozhatunk egy osztályt, ami az eredeti adatbázis osztály leszármazottja és egyetlen tagfüggvénnyel rendelkezik: egy paramétermentes konstruktorral, ami meghívja az ősosztály konstruktorát átadva a korábban definiált paramétereket. Így ha ezt a leszármazott osztályt használjuk a program készítése során mindig, a működő program éles szerverre való felmásolása után mindössze a központi állományban tárolt adatokat és esetleg az öröklésben megadott ősosztály (ha az ugyanazt az interfészt használja, mint a korábbi) nevét kell módosítani, hogy működőképessé váljon a program.

Egy honlap esetén a legelső szembetűnő dolog a megjelenés. Ez a fejlesztés folyamán azért jelent gondot, mivel ha a kódot a HTML-lel vegyítjük, egy meglehetősen nehezen módosítható rendszerhez jutunk. Szerencsére van megoldás, amit a Model-View-Controller (MVC) minta bocsájt rendelkezésünkre. Az MVC megköveteli, hogy az alkalmazást három fő alkotórésze bontsuk. A modell a logika központi részét hajtja végre, a nézet (view) a kimenet formázásáról gondoskodik, végül a vezérlő (controller) a bemenetet feldolgozó részegység. Webes környezetben ez annyival módosul, hogy mivel a rendszer bemenetet csak HTTP kérelmeken keresztül fogadhat (Get, Post vagy Cookie formájában), a vezérlő elhagyható. A maradék kettőt pedig úgy választhatjuk szét, ha a megjelenítést hordozó HTML fájlokat és a működésért felelős osztályokat külön fájlokba rendezzük. Ezt egy sablonrendszerrel (template) lehet legkönnyebben megvalósítani, ami olyan fájlokat jelent, ahol csak a HTML kód és a változó tartalomhoz szükséges speciális jelölések találhatóak. A mi rendszerünk egy saját fejlesztésű template-motorra épül, de számos népszerű rendszer létezik, amiket érdemes kipróbálni, pl. a Smarty. Ennek a használatáról a http://www.smarty.net/ weblapon kaphatunk

részletes információkat. És hogy mi az előnye, ha szétválasztjuk ezt a két szekciót? Nos, bár az alkalmazáshoz némileg több memóriára lesz szükség, egy könnyebben módosítható és jobban átlátható rendszerhez jutunk segítségével, tisztább lesz a kód és a megjelenítési kód újrahasznosítása is maximalizálható. Ezen felül, ha a megrendelőnek nem tetszik a design, sokkal könnyebb egy ilyen rendszert új formába önteni, nem kell feltétlenül az osztályokat módosítani. Nagyobb rendszerek, így a mi esetünkben is feltétlen érdemes megfontolni a template rendszerek használatát.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Használtlak.hu - Adminisztráció</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
<link rel="stylesheet" type="text/css" href="css/style.css">
<link rel="stylesheet" type="text/css" href="css/talalat.css">
<script language="javascript" type="text/javascript"
src="../js/ajax.js"></script>
<script language="javascript" type="text/javascript"
src="../js/popup.js"></script>
<script language="javascript" type="text/javascript"
src="../js/formCheck.js"></script>
<base href="{$BASE_URL}">
<!--[if IE]>
<style type="text/css">
#content { margin-left: 184px; }
</style>
<![endif]-->
</head>
<body>
<center>
<div id="container">
<div id="top">
<img src="images/site/logo.gif" width="242" height="60" align="left"
border="0">
<a href="site_admin/index.php?logout">Kilépés</a>
<h1>Adminisztráció</h1>
<p>{$datum}</p>
</div>
<div id="left">
<ul id="leftMenu">
      <li><a href="site_admin/index.php?page=Index">Statisztikák</a></li>
      <li><a href="site_admin/index.php?page=StaticPages">Statikus
oldalak</a></li>
      <li><a href="site_admin/index.php?page=Moderate">Hirdetés
moderálás</a></li>
      <li><a
href="site_admin/index.php?page=Register">Cégregisztráció</a></li>
```

```
      <li><a href="site_admin/index.php?page=Banners">Bannerek</a></li>
      <li><a href="site_admin/index.php?page=Users">Felhasználók</a></li>
      <li><a href="site_admin/index.php?page=Logins">Belépések</a></li>
</ul>
</div>
<div id="content">
{$content}
</div>
<div class="clear"></div>
<div id="bottom"><a href="{$BASE_URL}">www.hasznaltlak.hu</a></div>
</div>
</center>
</body>
</html>
```

3.4 kód Az adminisztrációs felület sablonja

Természetesen ezeken a mintákon kívül még számos más minta is a rendelkezésünkre áll, melyek esetleges egyedinek tűnő problémákra adhatnak olyan megoldást, amelyet mások már kifejlesztettek és kipróbáltak, míg mások igazolták ezek életképességét. Ha a PHP-hez tartozó tervezési mintákat szeretnénk áttekinteni, illetve szükség esetén használni is azokat, remek választás a http://www.fluffycat.com/PHP-Design-Patterns/ weboldal, ahol egy helyre gyűjtve megtalálhatjuk a szükséges anyagot, illetve nagy segítségünkre lehet a Négyek Bandájaként (Gang of Four, név szerint Erich Gamma, Richard Helm, Ralph Johnson és John Vlissides) emlegetett szakemberek által megalkotott könyv, a Tervezési minták (Design Patterns).

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, a hibakezelés egy új módja valamint a különféle tervezési minták felhasználásának jelentősége. A következő részben a rendszer alapvető működésével fogunk megismerkedni, és nekikezdünk annak interaktívvá formálásához is.

4

Adatbiztonság

A web egy állapotmentes környezet. Ez azt jelenti, hogy a weboldal megnyitásakor elindul egy program, lefut, majd a kapcsolat lezárásra kerül, azaz a futás is végetér. Tehát amikor a honlapon belül egy újabb linkre kattintunk, mert egy másik részletre vagyunk kíváncsiak, az egy újabb –vagy éppen a korábbi – program futtatását vonja magával.

Nem egy alkalommal azonban szükségünk van arra, hogy ismerjük a látogatónk múltját, például hogy sikeresen bejelentkezett-e már, melyik volt az utoljára meglátogatott oldal, vagy éppen elérhet-e egy megadott oldalt, esetleg ehhez hitelesítés, bejelentkezés szükséges. Erre problémára adtak egyfajta választ a böngészők a süti (cookie) formájában. Ez egy fájlt jelent, amit a felhasználó számítógépén tárolunk, és a böngésző az oldalra való látogatáskor automatikusan elküldi azt a honlap – így a mi számunkra. Ez viszont igen komoly biztonsági rést jelent, hiszen honnan tudhatjuk, hogy a süti tényleg jó adatokat tartalmaz, vagy csak egy rosszindulatú támadónk van, aki be szeretne jutni valaki nevében.

További biztonsági problémát jelent a felhasználó által a rendszerbe bevitt adatok kezelése. Mivel valamilyen formában tárolnunk kell a felhasználó email címét és jelszavát ahhoz, hogy beléphessen, biztosítanunk kell azt is, hogy rajtunk és a programon kívül más ne férhessen hozzá ezekhez az adatokhoz. Ismét probléma, ha a felhasználó nem valós, vagy éppen káros adatokat próbál megadni, ezzel hiba kiváltására késztetve a rendszert. Ez utóbbi során nagyon könnyen olyan adatok birtokába juthat, amihez nem lenne jogosultsága és ezzel kárt okoz nekünk és azoknak is, akik rendelkezésünkre bocsájtották személyes adataikat.

Ebben a fejezetben szó lesz az adatok védelméről és különféle olyan technikákról is, amivel tesztelhetjük honlapunk biztonságát. Megismerkedünk az alapvető titkosítási eljárásokkal és a naplózás módszerével, valamint a hibakezelés és naplózás egy új módjával is – ez utóbbiról már esett szó az előző fejezetben, de most részletesebben is megnézzük.

Felhasználók

Egy dinamikus honlapon, mint a miénk is, gyakran szükség van arra, hogy különböző felhasználókat különítsünk el és további jogokat biztosítsunk számukra a weblaphoz történő hozzáférés tekintetében. Ehhez tudnunk kell, hogy kivel állunk kapcsolatban és hogy az milyen lehetőségekkel rendelkezik a rendszerben.

Az első lépcső ebben a tekintetben, mint mindig, most is az alapos tervezés. Végig kell gondolnunk, hogy hogyan tároljuk a felhasználói adatait, hogyan azonosítsuk a felhasználót, és hogy milyen szintű hozzáférést engedélyezünk számára. Az adatokat tárolását a könnyebb keresés és kezelés érdekében érdemes adatbázisban tárolni, megfelelő védelemmel ellátva. Szerencsére a ma használatos adatbázis-kezelők biztosítanak olyan lehetőséget, hogy az adatokat kellőképp védeni tudjuk az illetéktelen hozzáférésektől. Az első ilyen az adatbázishoz való hozzáférés, amihez többnyire szükséges egy felhasználónév és jelszó a szerveren. Érdemes itt is többféle szintet létrehozni, hiszen egy látogató nem biztos, hogy rendelkezhet írási joggal az adatbázis egy részén, viszont egy regisztrált felhasználó már felvihet adatot, ugyanakkor nem biztos, hogy ezzel módosíthat is meglévő tartalmat. Tehát az adatbázis-kezelőben is érdemes beállítani ezeket és számukra más-más módot adni az elérésre más-más felhasználónév és jelszó páros segítségével.

Mikor a felhasználó a regisztráció során megadja felhasználónevét (email-címét) és jelszavát, valamint egyéb szükséges adatait, azt először nekünk is ellenőrizni kell, hogy megfelelnek-e a valóságnak valamilyen szinten. Naivan azt gondolhatjuk, hogy a felhasználó minden bizonnyal valós adatokat adott meg, és így ez teljesen felesleges. Ez azonban igencsak téves elképzelés, hiszen mi van, ha email-cím gyanánt megadott egy egyszerű szöveget, ami abszolút nem hasonlít egy email-címre. Ilyenkor lépnek képbe a reguláris kifejezések. Segítségükkel ellenőrizhetjük, hogy egy adott szövegrészlet megfelel-e egy mintának, így például érvényes email címet takarhat-e, vagy az irányítószám valóban 4 számjegyből áll, nem tartalmaz más karaktert. Az alábbi sor egy email-cím alakját írja le egy reguláris kifejéssel:

```
/^\w+[\+\.\w-]*@([\w-]+\.)*\w+[\w-]*\.([a-z]{2,4}|\d+)$/i
```

Ezeket a teszteket érdemes elvégeznünk mind a kliens, mind a szerver oldalon. Kliens oldalon azért érdemes elvégeznünk ezt, mert így tudathatjuk a felhasználóval még idejében, hogy valamit rosszul adott meg és így gyorsan javíthat, nem kell megvárnia, míg a szerver felöl megérkezik a válasz. Természetesen ez valamelyest gyorsítja is a rendszer működését, mivel így a szerver oldalon lényegesen gyorsabb lehet a feldolgozás, azonban nem szabad csak erre hagyatkoznunk, hiszen néhány kattintással kikapcsolható a JavaScript támogatás. Ezt a lépést hívják verifikációnak.

A következő lépcső a validáció, azaz az érvényesítés. Ekkor tudhatjuk meg, hogy valós adatokat adtak-e meg. Tudnunk kell, hogy a megadott e-mail cím valós-e, tényleg létezik. Erre a legegyszerűbb megoldás, hogy a regisztrációt követően küldünk egy értesítést a címre, hogy köszönjük a regisztrációt. Van, aki itt megáll, mivel ha sikerült elküldeni a levelet, a cím biztosan létezik, és azonnal engedélyezi a belépést. Viszont ezzel még mindig nem tudtuk meg, hogy a felhasználó érvényes címmel rendelkezik-e, hiszen nem mindegyik mail-szerver támogatja az automatikus ellenőrzést. Ahhoz, hogy meggyőződjünk a cím valódiságáról, a regisztrációt követő email-be rakjunk egy linket, ami egyértelműen azonosítja a felhasználót és ezzel érvényesíteni is tudja címét. Ameddig az érvényesítés nem történik meg, addig nem hagyhatjuk belépni, hiszen az egyetlen értesítési felület a változásokkal kapcsolatban a felhasználó email címe. Ha az érvényesítés egy napnál tovább tartana, érdemes kitörölni a felhasználót az adatbázisból, hiszen valószínűleg hibás adatok kerültek a rendszerbe, vagy nem sikerült elküldeni a levelet.

További kérdés az adatok tárolása, hiszen azokat későbbi visszakeresés céljából nekünk meg kell őriznünk. Itt lép képbe a titkosítás. A megadott e-mail cím és jelszó biztosítja a felhasználó számára a hozzáférést az oldalhoz, segítségükkel egyértelműen azonosíthatjuk, kivel van kapcsolatunk, így ezeket érdemes olyan formában tárolni, hogy más ne férhessen hozzá. Ez főként a jelszóra igaz, mivel azt a felhasználón kívül más számára nem szabad olvasható formában tárolni. Ennek oka, hogy egy felhasználó más oldalakon is ugyanazt a jelszót használhatja, és ha mi kiadjuk, akkor elég komoly gondot okozunk számára. Az MD5 függvény, ami valószínűleg mind az adatbázis-kezelőben, mind pedig a PHP-ban megtalálható, ehhez nyújt segítséget. Az így kódolt szöveg gyakorlatilag csak akkor értelmezhető, ha a megfelelő adatot – vagyis a szükséges jelszót – adtuk meg. Ezzel azonban meggátoltuk a felhasználót is abban, hogy később visszanyerjük számára a jelszót, ha netán elfelejtette volna. Erre megoldás, ha elfelejtett jelszó esetén újat generálunk és azt elküldjük a felhasználó email-címére.

Másik módszer, ha nem ezt a titkosítást alkalmazzuk, hanem olyan eljárásokat, amelyek minden felhasználó számára visszakereshető – de csak a saját adatuk. Ezek az eljárások bár kicsit időigényesebbek, mégis nagyobb fokú védelmet biztosítanak a védeni kívánt adatoknak. Többek között idetartozik az AES (Advenced Encryption Standard), a blowfish és a DES (Data Encryption Standard) is. Ezek segítségével egy megadott stringet titkosíthatunk egy másik szöveg alapján, ami akár a felhasználó egyedi azonosítója is lehet. Ezekhez a függvényekhez az adatbázis-kezelők biztosítanak saját eljárást, de a PHP is rendelkezik egy bővítménnyel, amivel elérhetővé válnak számunkra. Ennek a függvénycsomagnak a neve *mcrypt*; a PHP ezt beépítve tartalmazza, mindössze a beállítások között kell engedélyeznünk annak használatát.

A regisztrációs adatok érvényesítését követően sor kerülhet a belépésre. Itt már szükségünk lesz olyan dolgokra, amivel azonosíthatjuk a felhasználót, tárolhatjuk az ehhez szükséges

adatokat, legalábbis ideiglenesen. Ehhez egy munkamenet sütit fogunk használni, ami a böngészőablak bezárása után törlődik a felhasználó számítógépéről. A sütiben pedig a felhasználó azonosítóját, egy változó értéket a süti azonosítására és egy érvényességi időt fogunk tárolni, természetesen titkosítva. A változó érték azért szükséges, hogy biztosítsuk, hogy a süti tőlünk származik, az érvényességi idő pedig azért kell, hogy egy bizonyos idejű inaktivitás után (pl. 1 óra) biztosan kiléptessük a felhasználót. Az alábbi kódok a „PHP fejlesztés felsőfokon” című könyv 13. fejezete alapján készültek.

```
<?php
require_once( "database/class.HasznaltlakDB.php" );
class AuthException extends Exception {}
class UserLogin {
      private $createTime;
      private $userID;
      private $version;
      private $crypter;
      public static $cypher = "blowfish";
      public static $mode = "cfb";
      public static $key = "Hfgipy617";
      public static $cookieName = "HasznaltlakUser";
      public static $myVersion = 1;
      public static $expire = 3600;
      public static $warning = 1800;
      public static $glue = "|";
      public function __construct( $userID=false ) {
            $this->crypter = mcrypt_module_open( self::$cypher, '',
self::$mode, '' );
            if ( $userID ) $this->userID=$userID;
            else {
                  if ( array_key_exists( self::$cookieName, $_COOKIE ) )
                        $buffer = $this->_unpack(
$_COOKIE[self::$cookieName]);
                  else throw new AuthException( "Nincs Cookie" );
            }
      }
      public function beallit() {
            $_SESSION[self::$cookieName]=$this->_pack();
      }
      public function validacio() {
            if ( $this->version || $this->createTime || $this->userID )
                  throw new AuthException( "Hibas formatum" );
            if ( $this->version!=self::$myVersion )
                  throw new AuthException( "Hibas verzio" );
            if ( time() - $this->createTime > self::$expire )
                  throw new AuthException( "Lejart Cookie" );
            else if ( time() - $this->createTime > self::$warning )
                  $this->beallit();
```

```
	}
	public function logout() {
		unset( $_SESSION[self::$cookieName] );
		header( "Location: ".$_SERVER[REQUEST_URI] );
	}
	private function _pack() {
$cookie = implode( self::$glue, array( self::$myVersion, time(), $this-
>userID ) );
		return $this->_encrypt( $cookie );
	}
	private function _unpack( $cookie ) {
		list( $this->version, $this->createTime, $this->userID ) =
			explode( self::$glue, $this->_decrypt( $cookie ) ) ;
		if ( $this->version!=self::$myVersion || $this->created ||
$this->userID ) throw new AuthException();
	}
	private function _encrypt( $plainText ) {
		$iv = mcrypt_create_iv( mcrypt_enc_get_iv_size( $this->crypter
), MCRYPT_RAND );
		mcrypt_generic_init( $this->crypter, $this->key, $iv );
		$cryptedText = mcrypt_generic( $this->crypter, $plaintext );
		mcrypt_generic_deinit( $this->crypter );
		return $iv.$crypttext();
	}
	private function _decrypt( $cryptedText ) {
		$ivSize = mycrypt_get_iv_size( $this->crypter );
		$iv = substr( $cryptedText, 0, $ivSize );
		$cryptedText = substr( $cryptedText, $ivSize );
		mcrypt_generic_init( $this->crypter, $this->key, $iv );
		$plainText = mdecrypt_generic( $this->crypter, $cryptedText );
		mcrypt_generic_deinit( $this->crypter );
		return $plainText;
	}
	public static function checkUser( $userName, $password ) {
		$conn = new HasznaltlakDB();
		$felhasznalo = $conn->vegrehajt( FELHASZNALO_LEKERDEZES );
		if ( $felhasznalo->getNumRows()==1 )
			return $felhasznalo->getValtozo( "user_id" );
		else
			throw new AuthException(”Hibas felhasznalonev/jelszo”);
	}
}
?>
```

4.1 kód A bejelentkezést kezelő osztály

Mivel egy meglehetősen összetett osztályról van szó, elemezzük ki, hogyan is működik. Az osztály konstruktora készít egy kódolót az mcrypt könyvtár felhasználásával, valamint

ellenőrzi, hogy a felhasználó belépett-e már. Ezt abból tudjuk megmondani, hogy létezik-e a megadott nevű (HasznaltlakUser) munkamenet-változó. Ezt természetesen érvényesíteni kell, amit a validacio() nevű függvény végez el. Ha nincs, vagy érvénytelen a munkamenet-változó tartalma, úgy egy új kivétel keletkezik, ami az AuthException nevet kapta a keresztségben. Ezt az adott oldalnak megfelelő módon, például átirányítással kezelhetjük.

A titkosítást és visszafejtést az encrypt és decrypt privát metódusok végzik, a pack és unpack függvényeken keresztül. Ha az osztály konstruktorának nem adunk azonosítót, megnézi, hogy van-e bejegyzett munkamenet-változó, ami a belépett felhasználóra utal. Ha nincs, kivételt dob. Elég valószínű, hogy szükség lesz arra is, hogy a felhasználó be tudjon lépni, ehhez az osztály statikus *checkUser* függvényét vesszük igénybe. Mivel ez a függvény, az osztály konstruktora vagy az egyes metódusok dobhatnak kivételt, így szükség lesz azok lekezelésére is egy try-catch blokkban. Azokon az oldalakon, ahol tudnunk kell, hogy a felhasználó bejelentkezett-e, a 4.2-es kódnál található függvényt hívhatjuk meg.

```
function beleptetes() {
      try {
            $user_id=false;
            if ( $_POST[login_email] &&
                 $_POST[login_password] &&
                 preg_match( EMAIL_MINTA, $_POST[login_email] ) )
            $user_id = UserLogin::checkUser( $_POST[login_email],
$_POST[login_password] );
            $user = new UserLogin( $user_id );
            $user->validacio();
            return true;
      }
      catch ( AuthException $e ) {
            header( ”Location:
index.php?page=Login&fromPage=”.$_SERVER[REQUEST_URI] );
            return false;
      }
}
```

4.2 kód A beléptetéshez szükséges kód

Ha a belépés sikeres volt, a függvény igaz értékkel tér vissza, bármilyen hiba esetén pedig hamissal. A fentiekkel tehát létrehoztunk egy olyan eszközrendszert, ami némi módosítással újrafelhasználható máshol is, és kellő biztonságot nyújt rosszindulatú behatolók ellen.

Támadási módok

Tárolt adatainkat védeni kell az illetéktelen hozzáférésektől. Ahhoz, hogy tudjuk, hogyan védekezzünk, tudnunk kell, mi ellen kell védeni az adatokat. Többféle módszer létezik arra, hogyan törhetünk fel egy-egy webhelyet, ezek közül a legelterjedtebb az SQL-befecskendezés (SQL-injection) és a munkamenet-eltérítés.

A munkamenetek egyfajta hidat képeznek arra a problémára, amit a fejezet elején említettünk, hogy a web egy állapotmentes környezet. Felmerül a kérdés, hogyan adhatjuk át a munkamenet azonosítóját a szerver számára, hogy az meg tudja mondani, kivel áll kapcsolatban. Erre a válasz az, hogy egy munkamenet-sütiben, vagy az URL-ben. Utóbbira akkor van szükség, ha a felhasználó számítógépén a sütik tiltva vannak. Mindkét esetben egy következő PHPSESSID nevű változó kerül átadásra (hacsak ezt a php.ini-ben nem bíráltuk felül), ami tartalmazza munkamenet azonosítóját.

Ezek, mivel a felhasználó gépén megtalálhatók, módosíthatóak is. Ez sajnos azzal jár, hogy a támadó átveheti egy másik, éppen aktív felhasználó fölött az irányítást. Ennek megakadályozása érdekében megtehetjük, hogy a felhasználó IP címét is letároljuk, mikor belép, viszont tudnunk kell, hogy az annak a gépnek az IP címe lesz, ami az utolsó pont volt a szerverre érkezés előtt, így sajnos ez nem biztos, hogy megfelel a célra. Másik mód, ha a böngésző USER_AGENT állandóságát vizsgáljuk, viszont lehet, hogy a felhasználó és a támadó ugyanolyan böngészőt használ, vagy a támadó a böngészője által küldött információt módosította. A kettő együtt viszont már viszonylag jó védelmet biztosít. A felhasználó adatai (az azonosítója) a saját titkosítási kulcs miatt egy ideig védve vannak – egészen addig, amíg a támadó rá nem jön, hogy mi a kulcs és mi a titkosítási eljárás. A kérdés az, hogy ez mennyi ideig tart, hiszen ami lekódolható, az is vissza is fejthető, de nem biztos, hogy lesz rá elég idő.

A másik említett módszer az SQL-befecskendezés. Ezt a technikát akkor használják, mikor valamilyen adatbevitelre van lehetőség egy formon. Ha bejövő adattal nem bánunk kellő óvatossággal, könnyen áldozatul eshetünk ennek a módszernek. Szerencsére a PHP 5 automatikusan meghívja az addslashes() nevű beépített függvény a `$_REQUEST` elemeire (így a `$_POST`, `$_GET`, `$_SESSION` és `$_COOKIE` elemekre), ami az aposztrófokat és idézőjeleket egy \ (backslash) jellel egészíti ki, így amikor lekérdezést hajtunk végre, nem léphet ki a megadott környezetből a lekérdezés. Például ha a következő lekérdezést hajtjuk végre:

```
SELECT * FROM tablanev WHERE adat='".$_POST[adat]."';
```

ahol a $_POST[adat] értéke ”1’ OR TRUE ”, akkor ez elvileg azt hozza magával, hogy az adatbázis minden elemét lekérdeztük. Az addslashes() szerencsére megakadályozza ezt úgy, hogy a $_POST[adat] értékét a következő változtatja: \”1\’ OR TRUE \”, ami pedig várhatóan nem felel meg egyetlen feltételnek sem. További hathatós segítséget jelent az ilyen támadások

ellen a reguláris kifejezések használata, hiszen azzal megakadályozhatjuk, hogy érvénytelen adatok kerüljenek feldolgozásra. Gyakori hiba még az is, ha egyáltalán nem használunk aposztrófokat lekérdezéseinkben, ilyenkor egy nem segít az escapelés. Ugyanilyen hiba, ha lehetőséget nyújtunk linkek hozzáfűzésére, és a támadó egy javascript:alert(/xss/) kódszöveget ír be. Ez bár csak bosszantó, a támadónak mégis információval szolgál a szerveren zajló folyamatokról.

Nagyon hasonló a helyzet abban az esetben is, ha mi nem figyelünk a saját lekérdezéseinkre, és a hibajelzéseket is bekapcsolva hagyjuk az éles szerveren. Egy lekérdezési hiba bekövetkezésekor bár nem történik meg a hiba automatikus kiírása, ha ezt kivételekkel oldjuk meg, akkor viszont erre nagyon oda kell figyelni. Egy, a szerverről kikerülő lekérdezés nagyon sokat elárul a támadóknak az adatbázis szerkezetéről és ezzel együtt a kulcsokról is. Ez pedig elég érzékenyen érinthet bennünket, illetve azokat a felhasználókat, akik ránk bízták adataikat.

Az e-mail címek kiírása a hirdetésekbe ugyan segítséget jelenthet a keresőknek abban, hogy kapcsolatba lépjenek a hirdetővel, de egyben azt is jelenti, hogy egy nagyon egyszerű, reguláris kifejezésekkel dolgozó program nagyon könnyen kinyerheti ezeket, majd spam üzenetekkel áraszthatja el az ártatlan felhasználókat. Az ilyen automaták képesek arra is, hogy betörjenek a honlapra, megadva az e-mail címeket felhasználónév gyanánt, majd egy szótár segítségével a jelszót próbálhatják végig. Ezt a támadásfajtát nevezik szótártámadásnak és azért hatékony, mert a felhasználók általában egyszerű jelszavakat használnak. Védekezni lehet ez ellen úgy, ha valahány próbálkozás után a felhasználó még mindig nem tudott belépni, úgy 10 percre blokkoljuk, vagy az ún. CAPTCHA módszert használjuk. Ez egy kép, ami eltorzított betűket és számokat is tartalmaz még olvasható formában, és ha be akar lépni a felhasználó, ezt is meg kell adja. Ez azért hatékony, mert egy automata nem képes megfelelően gyorsan visszafejteni ezeket a jeleket – legalábbis még ma. Szintén további védelem, ha ötvözzük ezeket a módszereket.

Összefoglalás

A fejezetben megismerkedtünk az adatvédelem fontosságával és néhány módszerrel, amiket alkalmazva elkerülhetjük, vagy legalábbis nagymértékben akadályozhatjuk a sikeres támadásokat. Természetesen az itt ismertetett módszereken kívül számos, más technika is létezik, ezekről a netről szerezhetünk tudomást, vagy a Hogyan törjünk fel webhelyeket című könyv is segítséget nyújthat. Mindenképpen érdemes kicsit elmélyedni ebben a témakörben, saját érdekünkben is.

5

Interaktív weboldalak

A legtöbb mai honlap már olyan technikával készül, ami megfelel a modern kor követelményeinek és valódi felhasználói élményt nyújt. Jelenleg egy olyan korszakot élünk, amit a legtöbben Web 2.0-ként ismernek és legfőbb jellemzője, hogy szinte egyáltalán nem kell várnunk arra, hogy a böngésző egy kattintás után betöltse az oldalt és elérjük a kért információt. Ennek hátterében nem az internetelérések sebességének növekedése áll, hanem egy új technológia: az AJAX; ez valójában egy betűszó, ami az Aszinkron JavaScript és XML kifejezésből származik – persze sokan mondják, hogy nem így van, de mi azért maradjunk meg ennél.

Ez a módszer valójában nem annyira új eredetű, mint azt elsőre gondolnánk. Alapját a még 1997-ben debütáló Internet Explorer 5-ös változatának XMLHttpRequest JavaScript objektum adja, amivel egy böngésző képességeivel ruházhatjuk fel kliens-oldali scriptünket. A ’97 és ’99 között szabványosított JavaScript, valamint a ’98-as XML 1.0 definíciót követően rendelkezésre álltak az eszközök, csak sajnos a böngésző-programok megjelenítésével gyakran akadtak problémák.

Mára a helyzet szerencsére jelentős mértékben megváltozott, a népszerű böngészők nagy része (Mozilla Firefox, Internet Explorer, Opera és Safari) tudja alkalmazni ezeket, elegendő csak a GMail-re vagy YahooMail-re gondolnunk. Az AJAX-módszer valószínűleg akkor indult robbanásszerű növekedésnek, amikor megjelent a Google elektronikus levelező rendszere, az imént említett GMail, és egyre többen kaptak lehetőséget annak kipróbálására. A siker egyik kulcsfontosságú eleme az volt, hogy eltűnt a honlapoktól addig megszokott várakozás és helyébe a gyors működés lépett. Nagy előny, hogy ezzel a technikával jelentősen tehermentesíthetjük a szervert is, hiszen nem kell egyszerre több száz kilobájtnyi adatot átküldeni a kliensnek, hanem mindössze néhány 10 KB elegendő.

A továbbiakban megismerkedünk az AJAX technika alapjaival és hogy miként használhatjuk fel azt honlapunk gyorsítására és látványosabbá tételére. Ehhez szükségünk lesz az osztályszerkezet egységességére és „könnyű”, jól hasznosítható modulokra. Utóbbi titka a

korai tervezésben és a jól átgondolt adatstruktúrában rejlik – ennek módszereiről az előző fejezetekben volt szó.

Az AJAX alapjai

Mivel az AJAX technológia alapját a JavaScript képezi, nagyon óvatosan kell bánni vele a böngésző-kompatibilitás miatt. Általában a következő kódot szokták javasolni:

```
var request;
try{
  request = new XMLHttpRequest();
}
catch (e) {
  try {
    request = new ActiveXObject("Msxml2.XMLHTTP");
  }
  catch (e) {
      try{
        ajaxRequest = new ActiveXObject("Microsoft.XMLHTTP");
      }
      catch (e) {
        alert("Kérjük használjon másik böngészőt!");
        return false;
      }
    }
  }
}
```

5.1 kód Az AJAX objektum elkészítése

Ettől egy kicsit eltérünk és a következő megoldást alkalmazzuk, ami használatát tekintve egy az egyben megegyezik az előzővel:

```
var request;
function createRequest( type ) {
  result = 0;
  try {
    result = new ActiveXObject( type );
  }
  catch( e ) {}
  return result;
}
request = window.ActiveXObject ? createRequest('Msxml2.XMLHTTP') ||
createRequest('Microsoft.XMLHTTP') : new XMLHttpRequest();
```

5.2 kód AJAX objektum, egy kicsit másképp

Hogy mi ennek a jelentősége? Elegánsabb és rövidebb kódot kaptunk, miközben az eredmény nem változott, így ez szerintem további kommentre nem szorul. A fenti kód segítségével

elkészítettük az alapobjektumot, most már csak használnunk kell. Ehhez először el kell döntenünk, hogy hogyan szeretnénk az adatokat továbbítani a szerver felé. A HTTP protokoll alapvetően két módszert ad erre: POST és GET. Míg POST segítségével nagyobb méretű adatokat is továbbíthatunk, addig a GET rendelkezik némi megkötéssel, viszont sokszor egyszerűbb az utóbbi használata.

A dolog persze nem áll meg itt, gondolnunk kell a szerver által küldött válaszra is. A kérdés itt az, hogy milyen formában adunk választ a kérdésre. Három lehetőségünk van: az első a legkézenfekvőbb, használjuk az XML-t. Ennek egy jelentős hátránya, hogy lassabb a feldolgozás főleg Internet Explorerben. A másik variáció, hogy HTML-kódot adunk válasznak. Ez szinte azonnal, mindenféle feldolgozás nélkül megjelenik a böngészőben, viszont hátránya, hogy a Script tag-ek között érkező JavaScript utasítások nem kerülnek végrehajtásra, így arról külön gondoskodnunk kell. Többségében ezt fogjuk használni, pont a gyorsaság miatt. A harmadik, bár ritkán használt változat, az ún. JSON. Ez válaszul egy JavaScript tömböt ad, amit feldolgozhatunk és megjeleníthetünk. Ennek hátránya, hogy mind az adat generálása, mind a feldolgozás időigényes, viszont cserében egy jól alkalmazható JavaScript felületet kapunk. A GWT és más, AJAX-alapú generátor-felületek (pl. ExtJS) ez utóbbit preferálják.

A szerver válaszát egy státuszkód jelzi, ami siker esetén 200-as vagy 304-es, utóbbi akkor, ha az utolsó lekérdezés óta nem történt lényeges módosulás (cache-elt információ). A kliens oldalon az AJAX objektumunk jelzi, hogy áll a szerver a feldolgozással, ezt a readyState alapján tudjuk megnézni. A readyState lehetséges értékei:

- 0 (uninitialized)
- 1 (loading)
- 2 (loaded)
- 3 (interactive)
- 4 (complete)

Ami nekünk érdekes, az a 4-es állapot. Ez jelenti azt, hogy a válasz teljes egészében átért, megjeleníthető. Az 1-estől fölfelé 3-ig használhatjuk arra, hogy egy kis ikont jelenítsünk a „Kérem várjon” felirattal.

Ha elkészítettük az ehhez szükséges függvényt, úgy már csak annyi dolgunk van, hogy elküldjük a szerver felé az adatokat, amit az AJAX objektum open metódusával tehetünk meg. Ez három paramétert vár, az első a GET, POST vagy HEAD értékeket veheti fel (fontos, hogy nagy betűvel írjuk!). Ez gyakorlatilag megfelel a formok get és post metódusainak, illetve a szerveroldal különféle header utasításainak. A második paraméterben meg kell adjuk azt az url-t, ahova a kérést továbbítani szeretnénk, ez biztonsági okokból csak saját szerverre lehetséges. A harmadik paraméter azt mondja meg, hogy a kérésünk aszinkron-e, azaz értéke true vagy false lehet – a mi esetünkben ez általában az első lesz. Egy további metódus még

hátravan, ez pedig a send, ami paraméterként a szerver felé továbbítandó név=érték párokat várja stringként egy „&” jellel elválasztva.

Ha a szerver hagyományos HTML formában ad választ a kérésre, úgy azt a responseText változó fogja tartalmazni, XML válasz esetén a responseXML. Utóbbit, mint említettem fel kell dolgozni, de erről részletesebben a 6. fejezetben, a GoogleMaps-nél lesz szó. Viszont az első esetben is szükség van egy további lépésre: a szerver a választ url-ben kódolt formában adja vissza, így ezt vissza kell fejtenünk, ehhez az 5.3-as kódban látható függvény fogjuk használni.

```
function URLDecode( encoded ) {
  var HEXCHARS = "0123456789ABCDEFabcdef";
  var plainText = "";
  var i = 0;
  while ( i < encoded.length ) {
    var ch = encoded.charAt(i);
    if ( ch=="+" )
      plainText+=" ";
    else if ( ch=="%" ) {
      if ( i<( encoded.length-2 ) &&
           HEXCHARS.indexOf(encoded.charAt( i+1 )) != -1 &&
           HEXCHARS.indexOf(encoded.charAt( i+2 )) != -1 ) {
        plaintext+=unescape( encoded.substr( i,3 ) );
        i+=2;
      }
      else {
        alert( 'Hibás karakter-kombináció: '+encoded.substr( i ) );
        plainText+="%[ERROR]";
      }
    }
    else
      plainText+=ch;
    i++;
  }
  return plainText;
};
```

5.3 kód A szerveroldali üzenet visszaalakítása

Tehát egy nagyon egyszerű lekérdezés elkészítése, ami mindösszesen egy Hello világ szöveget fog kiírni egy linkre kattintás után a következőképpen néz ki, ha felhasználjuk az előzőekben összeállított függvényeket (lásd 5.4-es kód).

```
function demo( refreshID, url ) {
  request = window.ActiveXObject ? getRequest('Msxml2.XMLHTTP') ||
            getRequest('Microsoft.XMLHTTP') : new XMLHttpRequest();
  request.overrideMimeType && request.overrideMimeType('text/html');
  request.open( "GET", url, true );
```

```
  request.onreadystatechange = function() {
    if ( request.readyState<4 )
      document.getElementById( refreshID ).innerHTML="<b>Kérem
várjon...</b>";
    else if ( request.readyState==4 ) {
      if ( request.status==200 || request.status==304 )
        document.getElementById( refreshID ).innerHTML=URLDecode(
request.responseText );
      else alert("Hiba! Nem tudtam kapcsolatot létesíteni! : " +
request.statusText );
    } } request.send(null); }
```

5.4 kód A tartalom megváltoztatása

Ehhez jön még a HTML kód, ami az 5.5-ös kódban látható.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <title>Próba</title>
<script language=”JavaScript” type=”text/javascript”
src=”ajas.js”></script>
</head>
<body>
<a href=”#” onclick=”demo( ’demo1’, ’demo.php’ );”>Kattints ide!</a><br>
<div id=”demo1”>Cseréljük le a szöveget.</div>
</body>
</html>
```

5.5 kód A kerethez tartozó HTML fájl

A demo.php-ban pedig mindösszesen egyetlen sor szerepeljen a bemutatás kedvéért:

```
<?php echo ’Hello világ!’; ?>
```

Ezzel el is készítettük első AJAX-os programunkat.

A szerver és kliens oldal együttműködése

Ahhoz, hogy ezt hatékonyan fel tudjuk használni – főleg a navigáció során –, szükségünk lesz az objektumok azonos interfészére. Ennek egyik legfőbb eszköze a két fejezettel ezelőtt ismertetett __toString függvény, melynek segítségével lényegében kiírathatjuk objektumainkat. Egy másik nagyon fontos dolog a rendszer struktúrájára vonatkozik: a különböző osztályokat adott mappába kell csoportosítani feladatuknak megfelelően és egységes név-struktúrával is rendelkezniük kell. A mi esetünkben a felhasználó által elérhető oldalak a „pages” mappában találhatóak, mindegyik neve a megadott oldal nevéhez kapcsolódik. Általános eljárás a weboldalak készítése során, hogy az osztályokat class.OsztalyNeve.php alakú fájlokba mentjük. Ennek egyik legnagyobb előnye, hogy mások

számára is jelezhetjük, a fájl objektum-orientált eszközt tartalmaz, valamint jelentősen könnyebbé válik az osztályok behívása is.

Ezeket figyelembe véve a különböző oldalak megjelenítése rendkívül egyszerűvé válik. A Microsoft cég által preferált ASP.NET programozási nyelvhez szorosan kapcsolódik a .NET keretrendszer, ebben pedig található egy olyan eszköz, ami Mesteroldal néven vált ismertté. A Mesteroldal a honlap azon részét képezi, ami viszonylag statikus, ritkán vagy egyáltalán nem változik az oldalak elérése folyamán. Ezt természetesen PHP-ban is megvalósíthatjuk, így az oldalak egységes keretben jelenhetnek meg és a mi számunkra is egyszerűbbé válik ezek megjelenítése. Így ahelyett, hogy minden egyes oldalnak külön fájlt készítenénk (pl. a kezdőlapnak index.php, hirdetésfeladáshoz hirdetesfeladas.php), egyetlen fájlban letudhatjuk a keretet és a tartalmat is. Persze a mesteroldal számára jeleznünk kell valahogy, hogy milyen aktuális tartalmat szeretnénk megjeleníteni – ezt a linkbe kódolva tehetjük meg, egy GET típusú paramétert átadva az oldalnak.

További előny, hogy ezzel a lépéssel nagyon könnyen AJAX-ossá tehetjük a rendszert. Ehhez viszont szükség van még egy fájlra, amiben csak a változó részt generáljuk ki. Amikor a menüelemekre kattintunk, a linkekbe kódolt oldalak fognak megjelenni alapesetben. Ha egy onclick esemény-kezelővel figyeljük a menüre való kattintást, az oldal lekérdezést küldhet a szerver felé az előbbi fájlnak és az értelmezve azt legenerálhatja a tartalmat. Mivel OO rendszerünk van, a tartalom megfeleltethető egy objektumnak, az objektum pedig kiíratható – köszönhetően a __toString metódusoknak. A szerveroldalon a keresett oldal osztályának példányosítása után kiírva azt a válasz visszaérkezik a kliens oldalra az AJAX objektumnak, ahogy azt az előzőekben megfigyeltük. Ez pedig azonnal megjeleníthető a böngészőben.

Az oldal készítése során felmerült egy olyan dolog, hogy Magyarország minden városa szerepeljen a rendszerben. Ez bár nem tűnik nehéz dolognak, elég komoly fejtörést okozott, mivel azt az adatbázisban tárolni kell, de mindenekelőtt valahogy hozzá kellene jutni ahhoz. Az eredményt hosszas keresgélés után végül a Magyar Posta honlapján megtalálható irányítószám-kereső hozta meg, itt egy kis kutatás után megyékre bontva megtalálhatjuk az összes bejegyzett várost és falut. A lista több mint 3000 elemből áll, így ez az egyik legnagyobb kezdőérték-része a weblapunknak. Ez magával vonta azt is, hogy a hirdetések szorosan kapcsolódni fognak a városokat tartalmazó adattáblához, a gyors keresés érdekében. Mivel minden város besorolható valamilyen megyébe (feltéve, hogy magyarországi településről van szó), a kereső oldalon található ország-megye-város elemek is kapcsolódnak egymáshoz, a megye változásakor előkerülnek a megyéhez tartozó települések is. Ennek hátterében szintén AJAX áll, hiszen adatbázisból kell lekérdeznünk azokat és mivel a korlátoznunk kell az adatátvitelt, nem kérhettük le egyszerre az összes település nevét. Itt érdemes megjegyezni, hogy az Internet Explorer hajlamos arra, hogy túl nagy mennyiségű adat JavaScriptes feldolgozása során kiakadjon. Bár a megjelenítést követően problémamentesen tovább működik, elég bosszantó lehet, ha fél percre kifagy a böngésző

semmilyen beavatkozást sem engedve, így lehetőleg kerüljük a túl nagy mennyiségű adatok (ami körülbelül 50KB-tól kezdődik) ilyetén módon történő továbbítását.

További kérelem volt egy olyan eszköz elkészítése, ami egy másik népszerű oldalon is használnak, és segítségükkel jegyzeteket készíthetünk azokról hirdetésekről, amikre esetleg még kíváncsiak leszünk később; ezt itt a feljegyzések formájában valósítottuk meg. A keresés után az esetleges találatok egységes formát képviselnek, mindegyik esetben három lényeges linkre kell figyelni. Az első az előbbi feljegyzés, ami a találat jobb felső sarkában található. Erre kattintva a program egy AJAX kérésen keresztül utasítja a szervert, hogy készítsen egy session (munkamenet) bejegyzést az adott elemről és jelenítse azt meg az oldal jobb szélén, a feljegyzések között. Természetesen innen törölni is lehet, ezt szintén egy AJAX-kérés formájában tettük meg. A másik fontos link kapcsolódik a harmadikhoz, így ezekről együtt lesz szó: a részletes információk és a további képek. Mindkettő kibővíti a találatot néhány sorral, az első az ingatlanról szolgáltat részletesebb információt, a második a többi feltöltött kép miniatűrjét tárja a látogató szeme elé, mindezt AJAX kéréseken keresztül. Fontos megjegyezni, hogy sajnos elég sok AJAX-os oldal nem fordít figyelmet arra, hogy ha kikapcsoljuk a JavaScript támogatását az oldalon, a honlapnak akkor is működnie kell. A fejlesztés folyamán ezt is figyelembe vettük, a hagyományos linkek megmaradtak, míg az AJAX-felület az onclick eseménykezelők biztosítják.

Összefoglalás

Ebben a fejezetben megismerkedtünk az AJAX technológia egy részével és az interaktív felületek mögött álló rendszerrel. Mindamellett, hogy egy igen kényelmes felületet adhatunk honlapunknak ezen eszközök használatával, könnyen beláthatjuk, hogy a szervert is jelentős mértékben mentesíthetjük a plusz feladatoktól, például a statikus tartalom átküldésétől és a mögöttük álló scriptek végrehajtását is mellőzhettük, jelentős erőforrásokat felszabadítva.

A következő fejezetben megismerkedünk a Google Maps-sel és megnézzük, hogyan készíthetünk saját térképet oldalunkhoz, valamint hogy hogyan tehetjük láthatóvá látogatóink számára térképen is a feltöltött ingatlanokat.

6

Google Maps

A Google Maps-ről, vagy ahogy korábban nevezték, a Google Earth-ről elég sok szó esik mostanában a webes fórumokon. Nem véletlenül, hiszen ez egy mindenki számára hozzáférhető, ingyenes térképészeti alkalmazásról van szó. Segítségével megtalálhatunk olyan helyeket, amikről addig nem tudtuk, merre is van, vagy útvonalat tervezhetünk, de akár érdekességképpen tehetünk egy sétát valamelyik amerikai város utcáján is. Külön érdekesség, hogy rendkívül gyorsan reagál a navigációs mozgásra annak ellenére, hogy mindez az interneten keresztül történik.

Honlapunk elkészítéséhez mi is igénybe vehetjük ezt a térképet. Segítségével meg fogjuk jeleníteni azokat a helyeket, ahol az adott ingatlan található, feltéve, hogy a felhasználó felvitte az ahhoz szükséges adatokat is. Ebben a fejezetben megismerjük egy egyszerű térkép megjelenítését, azt, hogy hogyan vigyünk fel adatokat és jelenítsük meg azokat, valamint hogy hogyan hasznosítsuk mindezt egy honlap számára. Fontos megjegyezni, hogy ez a rész foglalkozik az XML-lel is egy keveset, illetve azt, hogy ez nem képezi a tényleges alkalmazás részét, viszont remek lehetőség a továbbfejlesztésre.

Térképészeti alapismeretek

Mikor sétálunk az utcán, többnyire tudjuk, merre megyünk, hol járunk, hiszen különféle támpontok alapján megjegyezzük a helyet, ha csak minimális szinten is. Segítséget nyújthat egy idősebb fa, egy jellegzetes épület, vagy egy-egy feltűnőbb objektum is. Ha valahova el szeretnénk jutni, azt valószínűleg a hely címe alapján tesszük meg, esetleg megkérdezve az arra járó embereket, akik le tudják írni a környéket is.

Mostanában egyre elterjedtebbek azok az eszközök, amik segítenek nekünk megtalálni a kérdéses helyet: ezek a GPS készülékek. Ezek az eszközök azonban nem képesek arra, hogy egy helyszínt ilyen információk alapján megtaláljanak, hanem egészen más módon dolgoznak:

a földrajzi koordinátákat hívják segítségül. Ezek két komponensből állnak, melyek egyértelműen azonosítanak egy megadott helyet a földön. A két összetevő egy-egy valós szám, az egyik a földrajzi szélességet (latitude), a másik a földrajzi hosszúságot jelenti. Az előbbi a függőleges távolságot jelenti az egyenlítőtől, míg utóbbi a greenwichi kezdőkörtől vett távolság keleti, illetve nyugati irányban. Számunkra is ezek az adatok fogják jelenteni azt az információt, ami ahhoz szükséges, hogy az adott ingatlant le tudjuk vetíteni a térképre.

Mint korábban említettem – más okokból ugyan, de – rendelkezésünkre áll a magyarországi településlista, azonban a Google Maps-en hiába mondjuk meg a hely nevét, azt csak igen körülményesen fogjuk tudni, a Geocoder segítségével megjeleníteni. Szerencsére van megoldás, egy PHP osztály képében. Ez az osztály nyilvánosan elérhető bárki számára a http://mooder.org/googlegeo/ honlapon. Az ott található leírás alapján könnyen működésre tudjuk bírni, hogy hozzáférjünk a magyar települések földrajzi koordinátáihoz. Szerencsére ezt már korábban elvégeztem, így az adatbázis most már rendelkezik ezekkel az adatokkal. Miért van erre szükség? Mikor a felhasználó felvitte az ingatlanhoz tartozó várost, egyből tudhatjuk, hol érdemes keresni, legalábbis megközelítőleg, mivel az utcanevek még nem teljesek magyar vonatkozásban a Google adatbázisában. Így a felhasználónak mindössze pontosítania kell a helyszínt, és elmenteni azt az ingatlanhoz tartozó adattáblába. A megjelenítésnél figyelni kell arra is, hogy az adott városban a többi ingatlant is megjelenítsük, ez nemcsak a hirdető számára, hanem a látogatók számára is fontos. Így tájékoztathatjuk őket arról, milyen házak találhatók a környéken, milyen áron, milyen felszereltséggel, valamint hogy elhelyeztük-e azt már korábban a honlapon.

Ingatlanok pozicionálása

Ahhoz, hogy a hirdetésekhez tartozó épületeket el tudjuk helyezni a Google Maps-en, mindenekelőtt otthonosan kell mozognunk az általa kínált osztályok között. Ahhoz, hogy a szolgáltatást igénybe tudjuk venni, szükségünk lesz egy kulcsra, ami csak az adott DNS nevű webszerveren lesz használható. Ha a honlap költözik, a kulcsot is újra kell igényelni. Új kulcsot a http://code.google.com/apis/maps/signup.html oldalon kaphatunk.

Egy egyszerű térkép megjelenítéséhez a következő kód szükséges, természetesen a már meglévő kulcs birtokában:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=ISO-885-
2"/>
    <title>Saját próba - saját térkép</title>
  <style type="text/css">
```

```
body { font-family: Arial, sans-serif; margin: 0; }
#map { height: 500px; width: 500px; border: solid 1px #000; text-align:
center; padding: 4px; }
  </style>
        <script
src="http://maps.google.com/maps?file=api&amp;v=2&amp;key=KULCS"
            type="text/javascript"></script>
    <script type="text/javascript">
function load() {
      if (GBrowserIsCompatible()) {
var map = new GMap2( document.getElementById("map") );
var centerPoint =new GLatLng(47.532669, 21.625091);
map.setCenter( centerPoint, 14);
map.addControl(new GSmallMapControl());
      }
    </script>
  </head>
  <body onunload="GUnload" onload="load()">
<div id="map">Betöltés...</div>
  </body>
</html>
```

6.1 kód Egyszerű térkép megjelenítése

A fenti kód mindösszesen megjeleníti a térképet, méghozzá a fenti két koordinátával megadott (`47.532669, 21.625091`) középpontra, ami Debrecenben a Kálvin teret jelenti. Új térképet a GMap2 osztály segítségével hozhatunk létre, ami paraméterként azt az elemet várja, ahol meg kell majd azt jeleníteni. Mivel meglehetősen nagy térképről van szó, így érdemes azt nagyítva megjeleníteni: a 14-es érték már nagyjából megfelelő méretet jelent, de természetesen ez még nem elegendő ahhoz, hogy az utcaneveket is ki lehessen venni. Ezért szükség van egy nagyító eszközre is, amit a térkép addControl metódusával tudunk hozzáadni, az eszköz neve pedig GSmallMapControl, vagy egy részletesebb a GLargeMapControl; mi az előbbit fogjuk használni.

Ahhoz, hogy a térképen legyen mit megjeleníteni, elsőként biztosítanunk kell a hirdetésekhez tartozó geodéta adatok felvitelét az adatbázisba. Természetesen megtehetnénk, hogy a felhasználó manuálisan felvigye a koordinátákat, de ez nem egy túl jó lépés a könnyű kezelhetőség felé, ezen felül valószínűleg a térképen sem egészen ott fognak megjelenni a hirdetések, ahogy azt a felhasználó szeretné. Ezért ehhez is a térképet fogjuk igénybe venni.

Lehetőségünk van létrehozni saját pontokat, földrajzi koordináták segítségével és természetesen meg is jeleníthetjük azokat. Ezeket a pontokat a Google Maps markereknek hívja. A marker valójában egy ikon, amihez rendelhetünk saját képet is, és ha rájuk kattintunk, egyéb információhoz is hozzájuthatunk az adott helyhez kapcsolódóan, egy buborékablak formájában. Ebbe az ablakba fogjuk majd elhelyezni azokat az információkat, amik

érdekelhetik a felhasználót, illetve ezen keresztül vehetjük rá majd, hogy lementse az adott helyszínt.

Egy marker létrehozásának első lépése az ikon létrehozása. Ezt a GIcon osztály segítségével fogjuk elvégezni. A GIcon-nak be kell állítsuk a hozzá tartozó képet, a méretét, valamint két pontot a képen belül, amik a későbbi marker kötési pontjai lesznek. A kötési pontok közül az egyik azt fogja jelenteni, hogy a kép mely pontja adja a tényleges helyszínt a térképen, míg a másik az információs ablakhoz tartozó azon pont lesz, ahova a buborékablak kötődik. Egy ikon elkészítéséhez tehát a következő kód szükséges:

```
var houseIcon = new GIcon();
houseIcon.image = ”images/site/house.gif”;
houseIcon.iconSize = new GSize( 32, 32 );
houseIcon.iconAnchor = new GPoint( 16, 27 );
houseIcon.infoWindowAnchor = new GPoint( 16, 5 );
```

6.2 kód Ikon

6.1 kép A házak ikonja

Persze további beállítási lehetőségek is rendelkezésünkre állnak, például árnyalhatjuk is az ikont, de ezzel a kiegészítéssel most nem élünk. Az ikonhoz rendelt képet megtekinthetjük az oldalsó ábrán. A kép méretei 32-szer 32 képpont, a tényleges pozíciót az ikon által jelképezett ház aljának közepe, míg az információs ablak horganypontját a ház tetejének közepe jelenti.

Mivel az ikon mindössze a külcsínt adja, ezért tovább kell haladnunk a megjelenítés felé: az ikonból egy markert kell készítenünk, ami már megjeleníthető a térképen. Egy markerhez szintén különféle beállítási lehetőségek vannak, így például mikor pozícionálni akarjuk, tudnunk kell a házat mozgatni a térképen, azonban egy már elhelyezett ingatlant abszolút nem szabad elmozgatnia, csak az ingatlanhirdetést elhelyezőnek. Az elkészült markert ezután elhelyezhetjük a térképen egy rétegen (overlay-en). Egy mozgatható marker elkészítéséhez szükséges kódok megtalálhatóak a 6.3-as kódban.

```
var markerSets = { icon: houseIcon, draggable: true };
var houseMarker = new GMarker( centerPoint, markerSets );
map.addOverlay( houseMarker  );
```

6.3 kód Mozgatható marker készítése

Persze gyakran előfordul, hogy mikor mozgatjuk a térképet a helyszínt keresve házunk számára, az ikont hátrahagyjuk és bizony elég nehéz később előkeríteni, hol is felejthettük. Így mikor az ikon a látható részen kívülre kerül – vagy akár a mozgás közben is –, érdemes azt újrapozícionálni a térkép közepére. Ehhez egy eseménykezelőre van szükségünk, ami a térkép mozgására (moveend) reagál.

```
GEvent.addListener( map, 'moveend', function() {
      if ( map.getBounds().contains( houseMarker.getPoint() ) )
            houseMarker.setPoint( map.getBounds().getCenter() );
} );
```

6.4 kód Ikon újrapozícionálása

A dolog itt véget is érhetne, de a számunkra lényeges dolgok csak most következnek. Korábban említettük, hogy el kell helyeznünk a házakat és vissza is kell azokat keresnünk az adatbázisból és szintén pozícionálnunk. Ehhez három különféle működési módra lesz szükségünk a szerveroldalon. Az első az elhelyezés, amit csak akkor kell lehetővé tennünk, ha a felhasználó jogosult az ingatlan helyének elmozdítására (belépett és saját ingatlanról van szó). A második a pozíció elmentése, ami ténylegesen felviszi a koordinátákat az adatbázisba – itt szintén jogosultság-ellenőrzés szükséges. Végül a harmadik az adatok visszakeresése, ami nyilvános, de csak adott városra történhet, hiszen az összes hirdetés elhelyezése igencsak terjedelmes eredményt hozhat. Minden esetben igénybe vesszük az előző fejezetben megtanult Ajax ismereteinket is, a gyors működés érdekében.

Az elmentés nem történik meg automatikusan, a felhasználónak meg kell azt erősítenie, ezt az előzőleg elhelyezett ikonra való kattintással teheti meg. Ekkor a buborékablakban megjelenik egy gomb, amire kattintva egy ajax-lekérésen keresztül a szükséges adatok elmentésre kerülnek. Ahhoz, hogy mindezt megvalósítsuk, haladjunk végig a lépéseken. Elsőként elhelyezzük az ikont, ezt már biztosítottuk az ikon mozgathatóságával. A második lépcső az ikonra való kattintáskor megjelenő form. Ehhez újból segítségül hívjuk az eseménykezelőket. A form tartalma látszólag egyetlen gomb lesz, azonban három másik mezőt is tartalmaz: a két koordinátát és a hirdetés azonosítóját. Ezeket továbbítjuk a szerver felé, természetesen ajax segítségével, az pedig válaszol, hogy sikeres-e a mentés, avagy sem. A Google rendelkezésünkre bocsájt egy GXmlHttp objektumot is, amivel az AJAX objektumot hozhatjuk létre. Az ezekhez szükséges kód a 6.5-ös kódban látható.

```
GEvent.addListener(marker, "click", function() {
var inputForm=document.createElement("form");
inputForm.innerHTML='<fieldset style="width:150px;">'
+ '<legend>Hely mentése<\/legend>'
+ '<input type="hidden" id="lng" value="'+marker.getPoint().lng()+'"/>'
+ '<input type="hidden" id="lat" value="'+marker.getPoint().lat()+'"/>'
+ '<input type="button" value="Mentés" onclick="savePos(); return false;"
/>'
+ '<\/fieldset>';
marker.openInfoWindow(inputForm);
});
```

6.5 kód Google-féle AJAX

A kérdéses marker biztosan van valahol (hiszen azt az érkezéskor elhelyezzük a térképen), így a koordinátái is léteznek. A mentést a form gombjának onclick eseménykezelőjében hivatkozott savePos() függvény végzi, melynek alakja a 6.6-os kódban található.

```
function savePos() {
      var lng = document.getElementById( "lng" ).value;
      var lat = document.getElementById( "lat" ).value;
      var hirdetes = document.getElementById( "hirdetes" ).value;
      var request = GXmlHttp.create();
      request.open( 'GET',
'mapsets.php?type=save&hirdetes='+hirdetes+'&longitude='+lng+'&latitude'=la
t, true );
      request.onreadystatechange = function() {
            if ( request.readyState==4 )
                  marker.openInfoWindowHtml( request.responseText );
      }
      request.send( null );
      return false;
}
```

6.6 kód Koordináták elküldése a szervernek

Ezekkel a kódokkal tehát képes lesz a felhasználó elmenteni az ingatlan koordinátáit. Kipróbálni úgy tudjuk, hogy felteszünk egy hirdetést, majd a mellette megjelenő földgömb ikonra kattintunk a http://gjtester.extra.hu/hasznaltlak/ weboldalon. A következő lépcső az ingatlanok visszakeresése, amihez már az XML-re is szükségünk lesz.

Ingatlanok visszakeresése

Miután a felhasználók elmentették az adatokat, lehetőségünk lesz azok megjelenítésére is. Az adatbázisban adottak tehát a feltételek, így már csak rendezni kell őket valamilyen formában és megjelenítenünk mindhez egy-egy nem mozdítható markert, amire kattintva egy rövid leírást találunk majd az ingatlanról. A rendezéshez kézenfekvő az XML használata, hiszen jól strukturált adatokról van szó. A szervernek tehát ki kell keresse a kérdéses városhoz tartozó hirdetéseket és a azt XML formában vissza kell adja, melynek a szerkezete a következő formát fogja követni:

```
<?xml version="1.0"?>
<ingatlanok varos="VAROS">
      <ingatlan>
            <helyszin longitude="LNG" latitude="LAT"/>
            <leiras>
                  <tipus>Ingatlan típusa (eladó vagy kiadó)</tipus>
                  <cím>Utca, házszám</cim>
                  <ar>Ingatlan ára</ar>
```

```
                    <kep>Ingatlanhoz tartozó kép</kep>
            </leiras>
      </ingatlan>
…
</ingatlanok>
```

6.7 kód Az ingatlan adatait hordozó XML fájl

A kliens oldal ezt fogja megkapni, vagyis ezt fel kell dolgoznunk és az adatok alapján markert kell készítsünk mindegyikből. Első lépésként tehát szert kell tennünk a dokumentumra, amit szintén AJAX segítségével valósítunk meg. A feldolgozás kódja a következő lesz:

```
var xmlRequest = GXmlHttp.create();
xmlRequest.open( 'GET', "mapsets.php?type=show&varos={$varos_id}", true );
xmlRequest.onreadystatechange=function() {
if ( xmlRequest.readyState==4 ) {
var xmlDoc = xmlRequest.responseXML;
var ingatlanok=xmlDoc.documentElement.getElementsByTagName("ingatlan");
var helyszinek=xmlDoc.documentElement.getElementsByTagName("helyszin");
var leirasok=xmlDoc.documentElement.getElementsByTagName("leiras");
for ( var i=0; i<ingatlanok.length; i++ ) {
var point = new GPoint( parseFloat( helyszinek[i].getAttribute( "longitude"
) ), parseFloat( helyszinek[i].getAttribute( "latitude" ) ) );
map.addOverlay( markerKeszit( point, leirasok[i] ) );
}
}
}
xmlRequest.send(null);
```

6.8 kód XML fájl feldolgozása

A feldolgozás viszonylag egyszerű, lekérjük a dokumentumot és nekikezdünk. Elsőként a dokumentumból kivesszük az ingatlan címkéjű elemeket, amik közrefogják a lényeges adatot. A Google Maps számára a pozícionáláshoz szükséges egyetlen lényeges információt a helyszin elemek longitude és latitude attribútumai hordozzák, ezeket a getAttribute metódusokkal érhetjük el, a kinyert koordinátákból pedig GPoint-ot készítünk.

Hivatkozunk egy markerKeszit nevű függvényre, amivel a leírás és a pont alapján létrehozzuk a hirdetés markerét. Ebben az előzőleg már ismertetett lépések szerepelnek, így ezekre nem térek ki. A markerKeszit kódja a 6.9-es kódban látható.

```
function markerKeszit( terkepPont, leiras ) {
      var icon = new GIcon();
      icon.image="images/site/house.gif";
      icon.iconSize = new GSize( 32, 32 );
      icon.iconAnchor = new GPoint( 16, 27 );
      icon.infoWindowAnchor = new GPoint( 16, 5 );
      var ingatlanMarker = new GMarker( terkepPont, icon );
      GEvent.addListener( ingatlanMarker, "click", function() {
```

```
            leiras.getElementsByTagName("ar")[0].firstChild.data;
            informacio = '<div style="width: 300px; height: 130px;"><b>'

      +(leiras.getElementsByTagName("tipus")[0].firstChild.data) +
"</b><br>"
            +'<img
src="showimage.php?id='+(leiras.getElementsByTagName("kep")[0].firstChild.d
ata)+'&img=1&size=s" align="right">'
            +(leiras.getElementsByTagName("cim")[0].firstChild.data)+'<br>'
            +'Ára: '+(leiras.getElementsByTagName("ar")[0].firstChild.data)
            +"</div>";
ingatlanMarker.openInfoWindow( informacio );
      } );
      return ingatlanMarker;
}
```

6.9 kód A markereket elkészítő függvény

Természetesen a markerünk rendelkezik egy eseménykezelővel, a kattintásnál pedig megjeleníti a szükséges információt. Megfigyelhetjük, hogy az adatok kinyeréséhez firstChild.data elemeket is felhasználjuk, hiszen a lényeges információhordozók ezek. A Google Maps korábbi verziójában lehetőségünk volt arra, hogy ezt ne ilyen kerülővel kelljen megoldani, hanem az XML dokumentumok formázására is szolgáló XSL stíluslapokkal dolgozhassunk. A 2-es verzió a Macintoshon futó Safari nevű böngészője miatt ezt már nem támogatja, azonban az érdekesség kedvéért kitérek rá, ha netán később mégis alkalmazhatóvá válna ez a lehetőség. A függvényben az egyetlen változás az eseménykezelőnél történik, az a következő sorra fog módosulni:

```
GEvent.addListener( ingatlanMarker, "click", function() {
  ingatlanMarker.openInfoWindowXslt( description, "hasznaltlakMap.xsl" );
});
```

6.10 kód XSLT-hez szükséges módosítás

Láthatóan egyszerűbb a leprogramozása, most már csak a hivatkozott XSL fájlt kell elkészítenünk. Az XSL rendkívül sok lehetőséget rejt még ezen kívül is, de ezekre nem térek ki külön, a netes fórumok és a Wikipédia nagy segítséget nyújthat ezen a téren. A hasznaltlakMap.xsl fájl tartalma a következő lesz:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XLS/Transform">
<xsl:template match="/">
<xsl:apply-templates select="leiras"/>
</xsl:template>

<xsl:template match="leiras">
<div style="display: block; width: 300px; height: 130px;">
```

```
<b><xsl:value-of select="tipus"/></b><br>
<xsl:apply-templates select="kep"/>
<xsl:value-of select="cim"/><br>
Ára: <xsl:value-of select="ar"/> Ft
</div>
</xsl:template>
<xsl:template match="kep">
<img>
  <xsl:attribute name="src"><xsl:value-of select="."/></xsl:attribute>
  <xsl:attribute name="align">right</xsl:attribute>
</img>
</xsl:template>
</xsl:stylesheet>
```

6.11 kód Az XSLT stíluslap

Láthatóan ez is követi a szabványos XML-sémát, és teljesen logikusan leírja a megjelenítendő elemeket. Ezen felül nagy előny, hogy nem terheljük vele a JavaScript motort és így a böngészőt sem, tehát gyorsabb a használat és ezt a fájlt csak egyszer kell betölteni, elhelyezhető a gyorstárban, tehát nem generál plusz forgalmat sem. Remélhetőleg a Google Maps következő fejlesztéseinél visszaveszik a támogatását. A feldolgozáshoz és az XSL fájlhoz a Tanuljuk meg az XML használatát 24 óra alatt című könyvben találtam segítségre.

Összefoglalás

A fejezetben szert tettünk egy kevés térképészeti ismeretre és megismertük a GoogleMaps azon szolgáltatásait, amivel saját térképet készíthetünk és elhelyezhetünk azon számunkra fontos információkat, valamint jelentősen megkönnyítettük a honlap látogatói számára az ingatlanok megtalálását. Láthattuk, hogyan helyezzük el adatainkat XML fájlban és hogyan dolgozzuk fel azt JavaScript segítségével kliensoldalon. Kitekintettünk egy új irányba is, amivel az XML lehetőségeit is megmutattuk.

A fejlődés természetesen nem áll meg, ha netán a Hold felszínén is lesznek eladó ingatlanok, úgy azt is megjeleníthetjük majd egy igencsak hasonló alkalmazás segítségével, de erre még minden bizonnyal várni kell. Addig viszont érdemes megnézni a http://moon.google.com/ weblapot, ahol a Holdról készült, valamint a holdra szállásról tekinthetünk meg képeket és persze helyszíneket.

Szintén nem használtuk ki az útvonaltervezés lehetőségét sem, ez egy további fejlődési pontot jelenthetne, hiszen ha a látogató szeretné megnézni élőben az ingatlant, a Google Maps segítséget nyújthat az ingatlanhoz való navigálásra. Fontos még megemlíteni, hogy a Google Maps-en kívül is léteznek on-line térképészeti alkalmazások, azok azonban fizetősek. És egy ingyenes alkalmazás, ami legalább annyit nyújt – ha nem többet – mint fizetős társaik, azt hiszem lényegesen jobb befektetés.

Összefoglalás

A fejezetek folyamán megismerkedtünk a rendszertervezés néhány fontos részletével, a PHP új felületével, az AJAX technológia felhasználásával, valamint a Google Maps nyújtotta szolgáltatásokkal is. Megnézhettük azt is, mire lehet felhasználni az UML diagramokat, valamint hogy mennyire lehetnek segítségünkre a különféle tervezési minták a rendszer fejlesztése során. 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 – bennünket – felkeresni.

A rendszerhez tartozó, megjelenítést végző HTML fájlok még a 4.01-es verziónak felelnek meg, és csak kis mértékben alkalmazzák a CSS nyújtotta előnyöket és formázási eszközöket. A felépítés úgynevezett táblázatos elrendezést követ, ami meglehetősen nehézkesen alakítható és elég ingatag, ha egy AJAX technikát is alkalmazó honlapról van szó. Ez minden bizonnyal egy újabb fejlesztési pont lehet az alkalmazás történetében, de annak köszönhetően, hogy a rendszert szétbontottuk három (vagyis csak kettő, de ez a webes környezet miatt van így) fő alkotórészére, ez könnyen megvalósítható. Ha a megrendelő további fejlesztést szeretne, minden bizonnyal szükség lehet az XHTML-re való átállásra, valamint a könnyebb és egyszerűbb CSS szerkezetre.

Mint említettem, a Google Maps könnyen hozzáilleszthető a rendszerhez és így szintén egy jó fejlesztési pontnak bizonyulhat – minden csak a megrendelőn múlik. Átlagosan egy év az az idő, ami alatt felmerül a megrendelőben az igény a fejlődésre, így várhatóan a rendszer hamarosan újabb evolúción fog keresztül menni, aminek részét képezheti az előbb említett két pont is. Szintén nagy segítséget jelenthetnek az ez idő alatt kialakult felhasználói vélemények, hiszen nem mi vagy a megrendelő fogjuk használni a külső felület, hanem a hirdetéseket elhelyezni kívánó vagy csak egyszerűen a honlapra tévedt látogatók. Ennek elősegítése érdekében újabb részt érdemes hozzáfűzni a rendszerhez, egy ún. vendégkönyvet, mivel általában a felhasználók ritkán fejtik ki és küldik el véleményüket e-mailben. Ezeket viszont érdemes elkérni a későbbi fejlesztések megkezdése előtt, hogy láthassuk, min érdemes változtatni, bármennyire is nem számítunk sok üzenetre ilyen téren.

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

Büki András: UNIX/Linux héjprogramozás, Kiskapu Kft., 2002

Captcha kód: http://www.captcha.net/, The Official CAPTCHA Site

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

GoogleGeo: http://mooder.org/googlegeo/, Class for transforming addresses to geographic coordinates

Google Maps API: http://code.google.com/apis/maps/, Google

Grépály Csaba János: Az UML eszközeinek bemutatása egy komplex rendszer tervezésén keresztül, Szakdolgozat, 2007

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

Julie C. Meloni: Tanuljuk meg a MySQL használatát 24 óra alatt, Kiskapu Kft., 2003

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

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

Michael Kay: XSLT 2.0 Programmer's Reference, 3rd Edition, Wrox, 2004

Michael Monchur: Tanuljuk meg a JavaScript használatát 24 óra alatt, Kiskapu Kft, 2002

Michael Morrison: Tanuljuk meg az XML használatát 24 óra alatt, 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

XML: http://www.w3.org/XML/, W3C specifikáció

XSL, rövid leírás: http://www.hik.hu/tankonyvtar/site/books/b10104/ch10s03.html, Kempelen Farkas Digitális Könyvtár