# DIPLOMAMUNKA

*Buzdor Attila*

Debrecen

2010

**Debreceni Egyetem**

**Informatikai Kar**

# Zend Framework:
# a PHP keretrendszere

**Témavezető:**

*Dr. Juhász István*

egyetemi adjunktus

**Készítette:**

*Buzdor Attila*

programtervező matematikus

Debrecen

2010

# Tartalomjegyzék

# Köszönetnyilvánítás

Köszönettel tartozom Dr. Juhász István tanár úrnak a témavezetői feladatainak ellátásáért, és a rendszerfejlesztés technológiai hátteréről átadott tudásért. Megköszönöm még Tóth Imrének – aki főnököm és barátom – a segítséget, nélküle talán nem ismerhettem volna meg a Zend Framework keretrendszert. Továbbá hálával tartozom még Egri Zsolt és Karakas Péter kollégáimnak, akikkel szintén hosszas tanácskozásokat ejtettünk meg a témában. Köszönöm még Szimeonov Miklósnak, hogy szakmai szemmel átolvasta a diplomamunkát, és helyenként segített a megfelelő megoldások felkutatásában. Szeretném megköszönni Buzdor Gabriellának – unokatestvéremnek -, hogy nyelvtanilag átnézte a dolgozatot. Végül, de nem utolsó sorban pedig köszönöm a kitartást és a hajszolást szeretteimnek, akik mindvégig támogattak utamon.

# Bevezetés

Napjainkban a programozás, rendszerfejlesztés sokkal többről szól, mint önállóan működő, néhány ezer soros algoritmusokat írni. Biztosak lehetünk abban, hogy ha csak nem valami forradalmian új ötletünket szeretnénk megvalósítani, akkor az adott problémára már számtalan – mások által korábban megírt – alkalmazás létezik.

A Google fénykorában már az IT területeken tényked̋ők nagy többsége előtt ismert és köztudott, hogy hatalmas jelentősége van a webes környezetre fejlesztett alkalmazásoknak is. Jómagam is számtalan ilyen alkalmazást használok nap, mint nap. Csak a Google cégnél maradva, amióta elkészítették méltán népszerű és az egész világon ismert keresőjüket, rengeteg más hasznos alkalmazásukkal segítik felhasználók millióit mindennapi tevékenységeik során. Ilyen alkalmazás például a Google Calendar – mellyel online határidőnaplót vezethetünk, az eseményekre bárkit meghívhatunk, és ezekről ingyen SMS-ben fogadhatunk értesítőket. Vagy ott van a Google Docs – mely gyakorlatilag egy komplett irodai csomag webes változata (szövegszerkesztő, táblázatkezelő, prezentációkészítő, stb.), ennek minden előnyével (verziókezelt szerkesztés, megosztás másokkal, stb.). Szintén kiemelhető a Google Sites – aminek segítségével néhány kattintással hozhatunk létre webhelyeket, elkerülve az unalmas részeit a webfejlesztésnek. A lista pedig sosem ér véget, folyamatos a munka a cégnél. A Google népszerűségét leginkább annak tulajdoníthatja, hogy minden egyes szolgáltatásukba beletettek valamit, amivel jobb, többet ad a felhasználónak, mint „vetélytársaik” hasonló programjai. És mindezt természetesen ingyen teszik elérhetővé a nagyközönség számára.

Visszatérve azonban bevezető mondataimra, ellentétben a hagyományos asztali alkalmazásokkal, a webes környezetre történő fejlesztésnek bizony bőven vannak még kiaknázatlan területei, vagy épp olyanok, ahol egyes cégek már próbálnak betörni, de az elvárt működést még nem képesek hozni. Ugyan számtalan előnye van a weben történő fejlesztésnek, de az átlag felhasználó is elvárja, ha már „áttér” egy újabb szoftverre, hogy legalább azt a funkcionalitást tudja az új rendszer, mint amit tudott a régi. Bár a mai nagy

technológiák (JAVA, .NET) szintén rendelkeznek az internetre történő alkalmazásfejlesztés eszközeivel, van azonban egy nyelv, ami talán ma még mindig átütő többségben uralja a világhálót, ez a nyelv a PHP. A PHP egy rekurzív mozaikszó, első betűje magát a PHP-t jelenti, teljes jelentése: **P**HP: **H**ypertext **P**reprocessor. A nyelv viszonylag könnyen tanulható, segítségével alapvetően HTML forráskódokba írhatunk script-eket, melyek szintén HTML forrást generálnak, ezt pedig már képesek a web böngészők értelmezni (sajnos eléggé eltérően). Szintaktikája sok mindent örökölt egyéb programozási nyelvektől (C, JAVA, PERL). Amit fontos még megjegyezni, hogy a PHP alapvetően szerveroldali programozási nyelv, így a kódok is egy webszerveren tárolódnak. Létezik kliensoldali változata is, de nem erre lett kitalálva. 1994-ben tervezte Rasmus Lerdorf, de a fejlesztést és a karbantartást a „The PHP Group" névre hallgató csoport végzi a Zend cég technológiai hátterére támaszkodva. Ennek köszönhetően 2004-ben a PHP5 megjelenésével a Zend Engine 2-vel a nyelv már a teljes objektumorientált paradigmát implementálja. E dolgozat írásakor a nyelv legfrissebb verziója az 5.3.1, amit 2009. november 19-én jelentettek be.

A Zend Framework születése 2005-re tehető, ekkor a Zend cég már nagyon szoros kapcsolatban áll a PHP-vel. Az ötlet egyszerű: alkotni kell egy mindent átfogó, PHP alapú keretrendszert, ami egyesíti a RAD (Rapid Application Development – gyors alkalmazásfejlesztés) előnyeit az egyszerűséggel, miközben a PHP fejlesztőknek tervezési mintákat kínál, ezzel rávezetve őket a szabványos fejlesztésre. Számos eredményt értek el, többek közt a mai legelterjedtebb webfejlesztői kódstílusra vonatkozó konvenció is a Zend nevéhez fűződik. Maga a Zend Framework (amit az egyszerűség kedvéért a továbbiakban leginkább ZF-ként fogok emlegetni) egy széleskörű osztálykönyvtár, mely a leggyakrabban használt komponensektől (űrlapok, adatbázis-kapcsolat, stb.) egészen a legspeciálisabb elemekig (különböző mai „divatos" webszolgáltatások, pld. Twitter, Facebook, stb.) mindenhez nyújt támogatást, amire csak szükségünk lehet. A Zend Framework fejlesztői átlag kéthetente jönnek ki az újabb és újabb verziókkal, így a keretrendszer folyamatosan bővül, és mára elég nagy a támogatottsága. Számos neves cég – többek közt az IBM, a Google, és a Microsoft – adja nevét a keretrendszerhez, ezzel is növelve népszerűségét.

Végezetül a bevezetőben hadd ejtsek még néhány szót jelenlegi munkahelyemről. 2007 novemberében kerültem a WEB-SKY Consulting Informatikai Kft. szárnyai alá. Az ügyvezető az akkoriban induló cég profiljaként a webes alkalmazásfejlesztést jelölte meg. Első feladataim egyike volt, hogy végezzek némi kutatómunkát. Több különböző PHP alapú keretrendszert is kipróbáltunk – melyek közül nekem jutott a Zend Framework ismertetése. Összevetve a keretrendszerek képességeit, végül a Zend Framework mellett döntöttünk, ezt a döntésünket pedig azóta sem bántuk meg. A napokban született céges SWOT analízis során legfőbb erősségeink közt sorolhattuk fel a technológiát.

A diplomamunka célja áttekinteni a Zend Framework keretrendszer fontosabb elemeit, főként kitérve az MVC tervezési mintára. Véleményem szerint a dolgozat témája mindenképpen szerves része lehetne a magyar informatikai felsőoktatásnak. Minthogy egy korszerű, modern, és elismert keretrendszerről van szó, másodlagos célként tűztem ki magam elé, hogy minél több vállalkozó szellemű fejlesztővel megismertethessem a ZF-et.

Diplomamunkám során feltételezem, hogy az olvasó rendelkezik alapszintű PHP ismeretekkel, ugyanis nem célom a PHP nyelv bemutatása, számtalan könyv, szakmai dokumentáció létezik a témában.

A diplomamunka mellé leadott alkalmazás értelemszerűen a Zend Framework segítségével íródott.

# Miért éppen a Zend Framework?

A bevezetőben említettem, hogy a WEB-SKY Consulting Informatikai Kft.-nél mi is sok keretrendszert kipróbáltunk, ezeknek egy része teljesen használhatatlannak bizonyult, vagy megfelelő dokumentáció hiányában nem is nagyon erőltettük a kutatást. A projekt végére csupán néhány különböző keretrendszer maradt versenyben, végül választásunk a Zend Framework-re esett. A következő táblázat a teljesség igénye nélkül egy összehasonlítást ad a legtöbb PHP-s keretrendszerről.

http://phpframeworks.com

| PHP Framework | PHP4 | PHP5 | MVC | Multiple DB's | ORM | DB Objects | Templates | Caching | Validation | Ajax |
|---|---|---|---|---|---|---|---|---|---|---|
| Akelos | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| ash.MVC | - | ✓ | ✓ | - | - | ✓ | ✓ | - | ✓ | - |
| CakePHP | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | - | ✓ | ✓ | ✓ |
| CodeIgniter | ✓ | ✓ | ✓ | ✓ | - | ✓ | ✓ | ✓ | ✓ | - |
| DIY | - | ✓ | ✓ | - | ✓ | ✓ | ✓ | ✓ | - | ✓ |
| eZ Components | - | ✓ | - | ✓ | - | ✓ | ✓ | ✓ | ✓ | - |
| Fusebox | ✓ | ✓ | ✓ | ✓ | - | - | - | ✓ | - | ✓ |
| PHP on TRAX | - | ✓ | ✓ | ✓ | ✓ | ✓ | - | - | ✓ | ✓ |
| PHPDevShell | - | ✓ | - | - | - | - | ✓ | - | - | ✓ |
| PhpOpenbiz | - | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | - | ✓ | ✓ |
| Prado | - | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| QPHP | ✓ | ✓ | ✓ | ✓ | - | ✓ | ✓ | - | ✓ | ✓ |
| Seagull | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Symfony | - | ✓ | ✓ | ✓ | ✓ | ✓ | - | ✓ | ✓ | ✓ |
| WACT | ✓ | ✓ | ✓ | ✓ | - | ✓ | ✓ | - | ✓ | - |
| WASP | - | ✓ | ✓ | - | - | ✓ | ✓ | - | ✓ | ✓ |
| Yii | - | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Zend | - | ✓ | ✓ | ✓ | ✓ | ✓ | - | ✓ | ✓ | ✓ |
| ZooP | ✓ | ✓ | ✓ | ✓ | - | ✓ | ✓ | ✓ | ✓ | ✓ |

Mindent összevetve a következő 5 fő szempont alapján érdemes döntenünk:

- Milyen méretű alkalmazást szeretnénk fejleszteni?
- Van-e szükségünk speciálisabb komponensekre?
- Milyen a keretrendszer dokumentáltsága?
- Milyen időközönként fejlesztik a keretrendszert?
- Mennyire „kötött” a keretrendszer?

Az első kérdésre egyszerűen lehet válaszolni, azonban fontos kérdésről van szó. Alkalmazásunk mérete nem is a keretrendszerek közti választásban létfontosságú, hanem abban a kérdésben, hogy egyáltalán használjunk-e keretrendszert. Mivel a PHP egy HTML-be ágyazott, egyszerű script nyelv, egy bizonyos bonyolultságig a keretrendszerek használatának több hátránya van, mint előnye. A mai korszerű sávszélesség és technológiai háttér mellett ezek a hátrányok eltörpülnek ugyan, és sokkal inkább előtérbe kerülnek az olyan szempontok, mint az újrafelhasználhatóság, a konzisztencia, az átláthatóság, a szabványosság és a továbbfejleszthetőség, de tény viszont, hogy bármilyen keretrendszert használjunk is, mivel jóval több állományt kell betöltenünk minden egyes oldalfrissítésnél, az ilyen weblapok lassabban fognak működni, és nagyobb erőforrás-igényűek, mint egyszerűbb társaik.

Keretrendszerünk kiválasztásakor fontos szempont lehet, hogy a kiszemelt framework implementálja-e azokat az esetleges speciális komponenseket, amiket használni szeretnénk. A Zend esetén – mint azt látni fogjuk – szinte mindenre találunk beépített komponenst, azonban minden helyen implementálhatunk saját magunk is módszereket.

Egy jó keretrendszernél elengedhetetlen a megfelelő dokumentáció. A ZF ilyen téren nagyon jól áll, a néhány órás gyorstalpaló[1] mellett találhatunk részletes minden komponenst érintő használati útmutatót (többezer oldalas kézikönyv),[2] és egy API dokumentációt[3] is. Ezek mellett a Zend támogatottsága miatt rengeteg prezentációt, és néhány könyvet is találhatunk a világhálón, bár igaz, hogy leginkább angol nyelven.

Mindenképp érdemes azt is figyelembe venni, hogy a kiszemelt keretrendszer mögött milyen fejlesztői csapat áll, azok milyen jellegű fejlesztéseket, milyen időközönként vezetnek be. A Zend Framework itt is megfelelő lehet számunkra, az egyes verziók 2-3 hetente jelennek meg. A verziószámokat 3 komponensre osztják, az első szám a „major” verzió, a második a „minor”, a harmadik pedig a „mini” verziószám. Tehát pld. az 1.10.2-es verziót

[1] http://framework.zend.com/manual/en/learning.quickstart.intro.html
[2] http://framework.zend.com/manual/manual
[3] http://framework.zend.com/apidoc/core

előreláthatólag az 1.10.3 követi majd, a fejlesztéseket pedig méretük szerint szintén így osztályozzák. A verziószám harmadik komponensét képviselő úgynevezett „mini release”-ek jelennek meg a leggyakrabban.

Az utolsó kérdésre adott válasz már igencsak eltérő lehet a különböző keretrendszereknél. A Zend Framework-nél engem pont az fogott meg, hogy nincs túl sok automatizáció benne. Így a ZF kicsit lehet „fapadosabb” vetélytársainál (bár ez sem igaz a frissebb verzióknál már), azonban teljesen testreszabott kódokat írhatunk benne. Nem kötelez minket a komponensek használatára, tetszőlegesen használhatjuk a PHP nyelv összes lehetőségét.

# Egy webalkalmazás tervezőeszközei

Akár egyedül dolgozunk egy webalkalmazáson, akár csoportmunka keretein belül (ez a gyakoribb), mindenképpen ajánlott bizonyos tervezési eszközök használata. Ilyen módon segítség lehet számunkra, ha egy kódstílusra vonatkozó konvenciót betartunk, ha a munka előtt megtervezzük az adatok struktúráját – ez az esetek döntő többségében egy relációs adatbázis megtervezését jelenti – valamint használunk valamilyen projektirányítási rendszert. Ebben a fejezetben ezekről lesz kicsit részletesebben szó.

# *Zend konvenciók*

Talán nem kifejezetten tervezőeszközként kell említeni, de egy nagyobb méretű projektnél, ahol sok állományt írunk meg – főleg ha többen is dolgozunk együtt – szinte elengedhetetlen, hogy betartsunk egy bizonyos kódolási stílust. Konvenciók alkalmazásával magasabb minőségű kódokat írhatunk, és egyszerűbben elkerülhetők az esetleges hibák is. És akkor még nem is említettük, hogy ha a csapatba egy új fejlesztő érkezik, aki ismeri a konvenciót, akkor könnyen átlátja, olvashatóbb számára a már megírt kód. Vannak szabvánnyá vált nagyobb konvenciók is, ezek egyike a Zend nevéhez fűződik. A kódolási stílus részletes leírása angol nyelven a Zend weboldalain tanulmányozható.[4]

## PHP állományok formázása

Azokban a fájlokban, amik csak PHP kódot tartalmaznak, a záró `?>` alkalmazása tilos. Ennek figyelmen kívül hagyása problémát is okozhat, ugyanis a kimeneten felesleges whitespace karakterek jelenthetnek meg. A PHP forrás kezdetét jelző elem a `<?php`. A fejlesztés során viszont látni fogjuk, hogy időnként csak egyes kódrészeket szúrunk be a HTML forrásba, ekkor értelemszerűen szükség van a záró delimiter-re is. Ezekben az esetekben a rövid `<? //code here ?>` formát alkalmazzuk. A behúzásoknak 4 szóköznyi szélességűnek kell lenniük, és nem valódi tabulátoroknak. Ez könnyebbé teszi a dolgunkat, ha csapatban dolgozunk és a csapat fejlesztői különféle szerkesztőkkel nyúlnak a kódhoz. A maximális sorhossz a szabvány szerint 80 karakter, azonban ez csak egy ajánlás, a tényleges megkötés 120 karakterre vonatkozik. A sorvégjelek formátuma UNIX típusú. Ez azért fontos, mert kódunk nagy valószínűséggel UNIX alapú operációs rendszeren fog futni.

## Elnevezési konvenciók

- Osztályok: A Zend az osztályok neveire olyan konvenciót alkalmaz, ami alapján az osztályt tartalmazó PHP állomány könnyen megtalálható a könyvtárszerkezetben. A ZF gyökérkönyvtárának neve a Zend, így minden osztály neve Zend-del kezdődik. A keretrendszer összes osztálya ebben a mappában található elég jól strukturált

[4] http://framework.zend.com/manual/en/coding-standard.html

szerkezetben. Az osztályok nevei csak alfanumerikus karaktereket tartalmazhatnak és az alsóvonás (_) karaktert. A számok ugyan engedélyezettek, de nem szokás használni őket. Az alsóvonás karakter pedig a könyvtárszerkezetben lévő elválasztásokat (/) jelzi. Például a `Zend/Db/Table.php` állományban az osztály neve `Zend_Db_Table` lesz. Mint látható, ha az osztály neve több szóból áll, minden szó nagybetűvel kezdődik, azonban az egymást követő nagybetűk nem elfogadhatóak, így például a `Zend_PDF` nem, de a `Zend_Pdf` megfelelő osztálynév. Absztrakt osztályok és interfészek esetén hasonló a helyzet, de az osztálynév végén az `Abstract` (illetve az `Interface`) szónak kell szerepelnie, és azt nem előzheti meg közvetlenül alsóvonás karakter.

- Metódusok: A metódusok nevei szintén alfanumerikus karakterekből állhatnak, viszont itt az alsóvonás használata is tiltott. Számokat itt sem gyakran használunk, bár nem tiltott. A metódusok első betűje mindig kisbetű, ha a metódus neve több szóból áll, minden újabb szót nagybetűvel kezdünk. Ezt egyébként gyakran „camelCase" formázásnak hívják, és a legtöbb nagy technológia alkalmazza. A metódusok neve legyen minél tömörebb, és kifejezőbb arra nézve, hogy mit is csinál. Ahogy azt OO-ban a legtöbb helyen alkalmazzák, az egyes attribútumok elérési metódusait a `get` vagy a `set` szóval kezdjük.

- Változók: A változók nevei is alfanumerikus karaktereket tartalmazhatnak, a számokat nem használjuk, az alsóvonás tiltott, kivéve, ha a változó a `private` vagy a `protected` láthatósági szinttel rendelkezik. ilyenkor a változó nevének első karaktere egy alsóvonás. Itt is „camelCase" formátumot alkalmazunk. Szintén elvárás a beszédes ám tömör nevek használata, az egybetűs változónevek csak ciklusváltozóknál használatosak. Pld: `$userTable`

- Konstansok: Itt számok, betűk és alsóvonás használata is engedélyezett. Minden egyes betű nagybetűvel írandó, ha a konstans neve több szóból áll, alsóvonás karakterrel kell őket elválasztani egymástól. Pld: `THIS_IS_A_CONSTANT`

## Kódstílus

- Karakterláncok (sztringek): Ha a sztring nem tartalmaz változókat, aposztrófok közé kell tenni (szimpla idézőjel). Ha a szöveg aposztrófokat tartalmazna, – a jobb olvashatóság érdekében – kettős idézőjelekkel határoljuk. Ez különösen jól jön SQL utasítások esetén. Ha változókat szeretnénk tenni a sztringbe, kettős idézőjelet használunk.

```
$myString = "It's a " . 'string with a ' . "$var variable";
```

- Tömbök: Az egyszerű számmal indexelt tömbök esetén a negatív indexek tiltottak. A tömb indexelése nem csak nulláról indulhat, de célszerű így eljárnunk. A tömbök elemeit egy vesszővel és egy szóközzel választjuk el egymástól.

```
$sampleArray = array(
    1, 2, 3, 'Zend', 'Framework',
    $a, $b, $c,
    56.44, $d, 500,
);

$assocArray = array(
    'firstKey'  => 'firstValue',
    'secondKey' => 'secondValue',
);
```

Mint az látható, a behúzások segítségével teljesen jól olvasható egy nagyobb tömb deklarációja is. Az utolsó elem után érdemes kitenni a vesszőt, így elkerülve az esetleges hibákat új elem felvitelekor. Asszociatív tömbök esetében is hasonló a helyzet, viszont érdemes az olvashatóság kedvéért a => operátorokat egymás alá helyezni.

- Osztályok: Az osztályok neve utáni nyitó kapcsos zárójelet új sorba kell tenni. Az osztályon belül minden legalább 4 szóköznyi behúzással van írva. Egy PHP fájl maximum 1 osztályt tartalmazhat. Az osztályokon belüli attribútum változók az osztály implementációjának első részében helyezkednek el, ezután következnek a

metódusok. Minden metódusnak és változónak kötelező láthatóságot definiálni (`public`, `protected`, `private`). A metódusok esetén a nyitó kapcsos zárójel szintén új sorban kezdődik az osztályokhoz hasonlóan. A metódus neve és a paraméterlistáját tartalmazó zárójelpár közé tilos szóközt tenni.

```
/**
 * Documentation here
 */
public class MyClass extends SuperClass
{

    private $_id = 'myId';

    public function myFunction($myParam)
    {
        // Code here
    }

}
```

- Függvény- és metódushívások: Az aktuális paraméterlista elemei egy vesszővel és egy szóközzel vannak elválasztva egymástól. Ha túl hosszú sort kellene írnunk, hasonlóan járunk el, mint a tömbök deklarációjánál tettük.

```
threeParameterMethod(
    array(
        'longParameter',
        'anotherParameter',
    ),
    $thisParam,
    404
);
```

- Feltételes utasítás: Az eddigiektől eltérően a nyitó kapcsos zárójelet az `if` kulcsszóval egy sorba írjuk, azt csak egy szóköz előzi meg. A feltételt tartalmazó zárójeleken belüli részben az operátorok szóközzel vannak elválasztva az operandusoktól, az olvashatóság érdekében. Az `if` kulcsszó és a feltételt tartalmazó zárójelpár nyitó tagja közé is kötelező szóközt tenni (míg metódusoknál ugye tilos volt). A behúzások itt is maguktól értetődőek. Ha túl hosszú sort kellene írnunk, a feltételt hasonlóképp

törhetjük meg, mint az eddigiekben láttuk. Az `else` és az `elseif` kulcsszavak mindig az `if` záró kapcsos zárójelével egy sorba kerülnek. A szabvány nem engedi meg a kapcsos zárójelek elhagyását, amennyiben az adott blokkban csak egy utasítás szerepelne, bár a PHP-ban ez engedélyezett.

```
if ($three == 3) {
    echo 'ok';
} elseif ($three == 4 || $three == 2) {
    echo 'nearly ok';
} else {
    echo 'not ok';
}
```

- Többirányú elágaztatás: Teljesen hasonló szintaktika engedélyezett, mint az `if` esetében, az egyes `case` utasítások után kettőspontot teszünk, és alatta behúzva jön a kódrészlet.

```
switch ($numPersons) {
    case 1:
        echo 'one person';
        break;
    case 2:
        break;
    default:
        echo 'lot of people';
        break;
}
```

- Ciklusok: Az eddigi vezérlési szerkezetekhez hasonló.

```
for ($i = 0; $i < 10; $i++) {
    // code here
}
```

### Dokumentáció

Minden dokumentációs blokk („docblock”) a phpDocumentor formátumának kell, hogy megfeleljen, ez nagyjából megegyezik a JavaDoc formátumával. A konvenció nem is tér ki ennek részletezésére, megtekinthető a phpDocumentor weblapján.[5] Magunk dönthetjük el, hogy milyen elemeket adunk meg az egyed dokumentációkban, de célszerű osztályok esetén egy `@package`, `@category`, `@copyright`, és maga a leírás megadása, metódusok esetén a paraméterlista ismertetése (akár POST esetén is), a visszatérési érték ismertetése, valamint az, hogy milyen esetleges kivételeket dobhat az adott metódus (`@param`, `@return`, `@throws`).

[5] http://phpdoc.org

# *Adatbázisterv készítése*

Egyetlen jól felépített webes rendszer sem létezhet átgondolt, strukturált adatbázisterv nélkül. Az adatbázisterv egy vizuális tervezési eszköz, melyen az adatbázis komponenseit láthatjuk: az adatbázistáblákat, azok kapcsolatait, a függéseket, elsődleges kulcsokat. Egy okos adatbázistervező ezeken felül egyéb szolgáltatásokkal is segíti tervezői munkánkat.

Adatbázistervezőnk az adatbázís típusától függetlenül – legyen az MySQL, Postgre, MsSQL vagy Oracle – ki kell, hogy tudja generálni a terv alapján az SQL-DDL utasításokat, melyekkel ténylegesen elkészülhet az adatbázis. Az SQL szintaktikája lényegesen nem, de eltérő lehet az egyes DBMS-ek (DataBase Management System) esetében. Arra is szükségünk lehet időnként, hogy ilyen SQL állományokból tudjunk adatbázistervet előállítani.

A következő szolgáltatás a fentihez hasonló, ez az adatbázissal való szinkronizálás lehetősége. Ekkor megadjuk a tervezőnek az adatbáziskapcsolathoz szükséges hitelesítési adatokat (a szerver nevét, a felhasználói nevet, jelszót és magát az adatbázis nevét), és közvetlenül tudjuk szinkronizálni a tervet az adatbázissal.

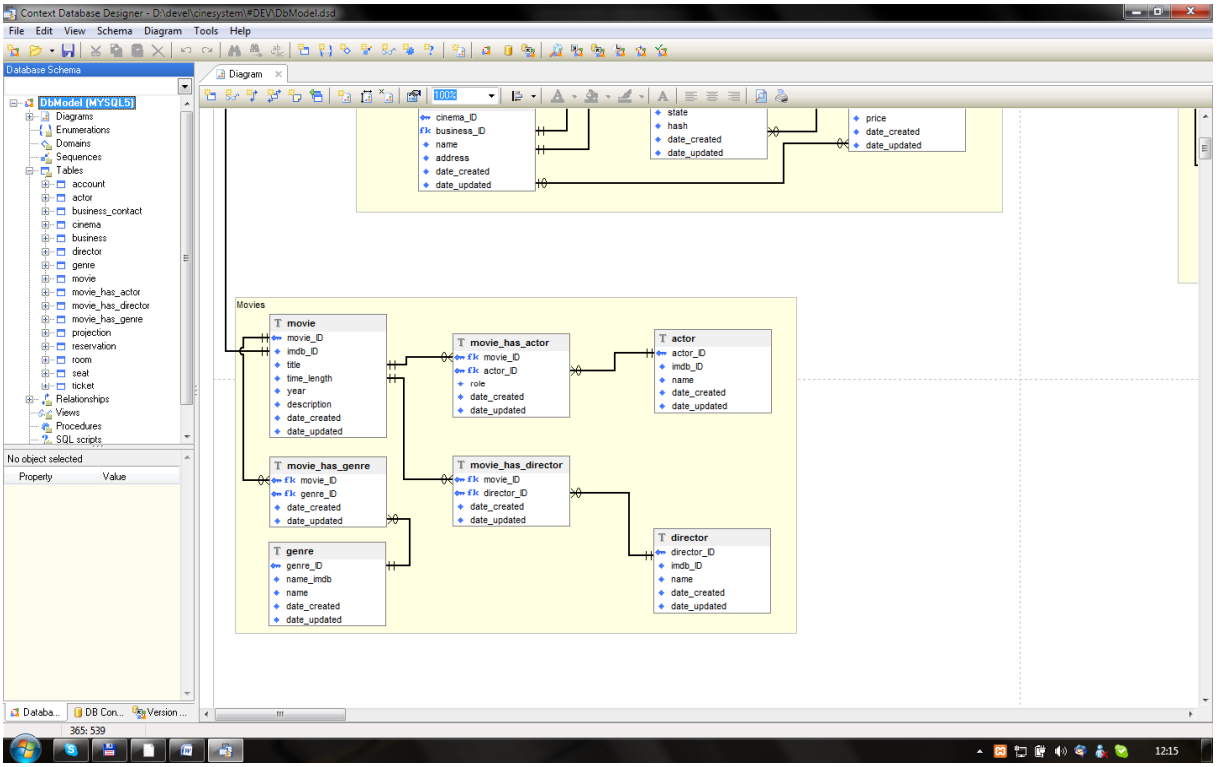
Innen egy újabb lépés, ha szoftverünk képes az egyes adatbázisverziók kezelésére, és azok között tetszőlegesen tud mindkét irányban migrálni. A migrációnak nagyon fontos szerepe van a fejlesztés során. Ha egy új szoftververzióval állunk elő, melyben esetleg változtattunk az adatbázis struktúráján, előfordulhat, hogy nem csak magát a szerkezetet kell változtatnunk, hanem megnehezíti dolgunkat a rendszerünkben már meglévő adatok átkonvertálása. Ilyen esetekben, ha el szeretnénk kerülni az ügyfeleink számára fontos adatok elvesztését, károsodását, migráló kódot kell írnunk. Ez a migráló kód értelemszerűen nem csak SQL utasításokból állhat, szükség lehet egyéb átalakításokra is. A migráció lényege, hogy minden egyes adatbázisverzióhoz készül egy-egy osztály, melyben leimplementáljuk a lefelé, illetve felfelé történő migrálást (ezek az osztályban metódusként fognak megjelenni, például `sqlUp()`, `sqlDown()`, `phpUp()`, `phpDown()`, `preUp()`, `preDown()`, `postUp()`,

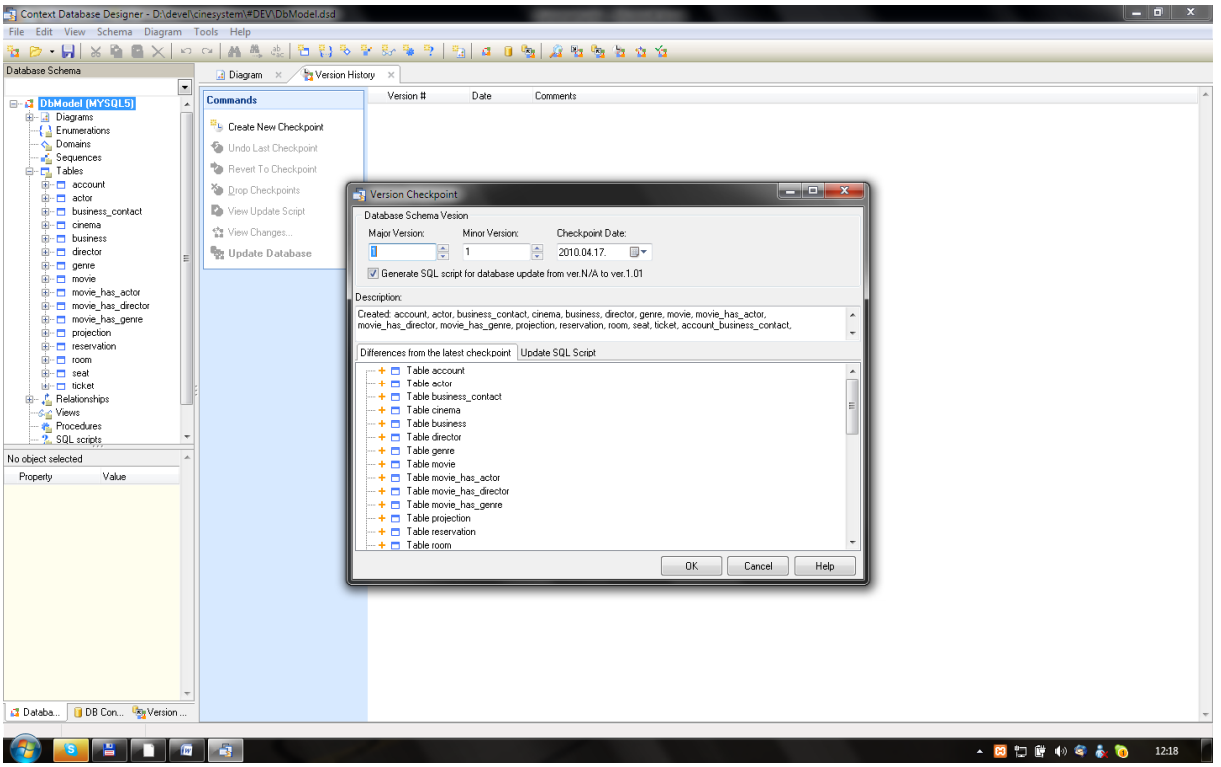
`postDown()`). Így tetszőlegesen válthatjuk az adatbázis verziószámát, és minden adat biztonságban maradhat.

## Context Database Designer

A Context Database Designer a fent említett funkciókat tudja, és nagyméretű adatbázistervek esetén is megfelelő gyorsasággal működik (sajnos ez sok másik hasonló szoftver esetén nem mondható el). A program fizetős ugyan, de igazán nem drága ahhoz képest, hogy mennyire hasznos. Számos hasonló rendszert kipróbáltam már, igazából mindegyiknek volt valami „nyűgje”, végül ennél a tervezőnél maradtam.

A diplomamunkámhoz csatolt alkalmazásom adatbázisterveit szintén a program segítségével készítettem, a dolgozat függelékében megtalálhatók az adatbázistervek. Íme néhány képernyőkép a Context Database Designer-ből:




Context Database Designer - D:\devel\cinesystem\#DEV\DbModel.dsd
File Edit View Schema Diagram Tools Help
Database Schema
DbModel (MYSQL5)
Diagrams
Enumerations
Domains
Sequences
Tables
account
actor
business_contact
cinema
business
director
genre
movie
movie_has_actor
movie_has_director
movie_has_genre
projection
reservation
room
seat
ticket
Relationships
Views
Procedures
SQL scripts
No object selected
Property
Value
Diagram
Version History
Commands
Create New Checkpoint
Undo Last Checkpoint
Revert To Checkpoint
Drop Checkpoints
View Update Script
View Changes...
Update Database
Version #
Date
Comments
Version Checkpoint
Database Schema Vesion
Major Version:
Minor Version:
Checkpoint Date:
1
2010.04.17.
Generate SQL script for database update from ver.N/A to ver.1.01
Description:
Created: account, actor, business_contact, cinema, business, director, genre, movie, movie_has_actor, movie_has_director, movie_has_genre, projection, reservation, room, seat, ticket, account_business_contact,
Differences from the latest checkpoint
Update SQL Script
Table account
Table actor
Table business_contact
Table cinema
Table business
Table director
Table genre
Table movie
Table movie_has_actor
Table movie_has_director
Table movie_has_genre
Table projection
Table reservation
Table room
OK
Cancel
Help
Databa...
DB Con...
Version ...
12:18

# *Hibajegykezelő rendszerek*

A fejlesztés során több mint valószínű, hogy nem egyedül dolgozunk, az esetek döntő többségében egy csoport tagjaként veszünk részt a projektben. A csoportnak majdnem minden esetben van egy vezetője, az ő feladata kiosztani a munkákat a fejlesztők között. Erre a dologra számos szoftveres megoldás született, manapság a legnépszerűbbek az úgynevezett hibajegykezelő rendszerek. Bár a magyar terminológia hibajegyként emlegeti az angol „ticket” szót, nem szabad félreértenünk, nem feltétlenül hibákat fogalmazunk meg hibajegyek formájában, hanem az egyes elvárásokat, feladatokat tűzzük ki bennük. Persze ezek a hibajegykezelők sokkal több szolgáltatást nyújtanak, mint egyszerűen a hibajegyek létrehozását, szerkesztését, törlését, stb. Célszerű olyan rendszer mellett döntenünk, amely beépítetten kezel valamilyen SCM stratégiát (Software Configuration Management), mint például a népszerű Subversion-t, GIT-et, Bazaar-t vagy hasonlókat. Azt sem árt figyelembe vennünk, hogy az adott hibajegykezelő mennyire testreszabható egyéni igényeinkhez igazodva. A következőkben pár hasznos tulajdonságát – előnyét – mutatom be a hibajegykezelésnek.

Maguk a hibajegyek szemléletesen ténylegesen egy „jegynek”, avagy cetlinek felelnek meg, amin számos információ található. Minden hibajegynek van egy rövid címe, egy részletesebb leírása, amiben a probléma van megfogalmazva, esetleges figyelmeztetésekkel, javaslatokkal a megoldásra nézve. A ticket-nek értelemszerűen lesz egy szerzője, aki létrehozta, valamint egy tulajdonosa, aki foglalkozik az adott feladattal. Ezeken kívül számos egyéb adatot is nyilvántarthatunk a hibajegyről, adhatunk neki prioritást, hozzáírhatjuk a feladatra szánt időt, a ténylegesen eltöltött időt, egy elkészültségi százalékot, sőt akár határidőt is. Kapcsolatokat is definiálhatunk az adott feladatok között, például az egyik megelőzi a másikat, vagy kiváltja, így szabályokat definiálhatunk rájuk. A felsorolt tulajdonságok némelyikét a programozók módosítják, míg másik részét a csoportvezetők.


Ami nagyon fontos még, és minden rendszer számon tartja, hogy a ticket-eknek van egy aktuális állapota. Egy hibajegy – életciklusa során – több állapoton is keresztülmegy. Ezeket a státuszokat magunk adminisztrálhatjuk a saját stratégiánkhoz igazodva. A hibajegyek státuszaival úgynevezett workflow-t definiálhatunk, amiben azt adhatjuk meg, hogy mely résztvevők pontosan milyen állapotokból milyen állapotokba helyezhetik át a jegyet. Általában létrehozáskor a ticket „új” vagy „tervezett” állapotba kerül (a „tervezett” állapot azt jelzi, hogy még nincs teljesen letisztázva, így nem felvehető). Ekkor, ha egy programozó elfogadja, ő lesz a tulajdonosa és „folyamatban” állapotba teszi a hibajegyet. Ahogy halad a feladat végrehajtásával, a jegy bizonyos egyéb tulajdonságait módosítja, így például a ledolgozott órákat írhatja fel, megjegyzéseket fűzhet a munkájával kapcsolatban. Ha pedig elkészült a feladatban kiírtakkal, „tesztelésre kész” állapotba helyezi a hibajegyet. Ekkor a csoportvezető, vagy a tesztelő hozzálát a teszteléshez, és attól függően, hogy mennyire elégedett a feladat megoldásával visszadobhatja a hibajegyet, vagy lezárhatja azt. A lezárt ticket-ek minősülnek kész feladatnak, illetve ha bármilyen más okból kell lezárni, „elutasított” állapotról beszélünk. Egy megfelelő hibajegykezelő rendszernél ez a workflow teljesen egyénileg alakítható ki, azaz az állapotok és azok rendszere, hogy mi mit követhet, tetszőlegesen variálható.

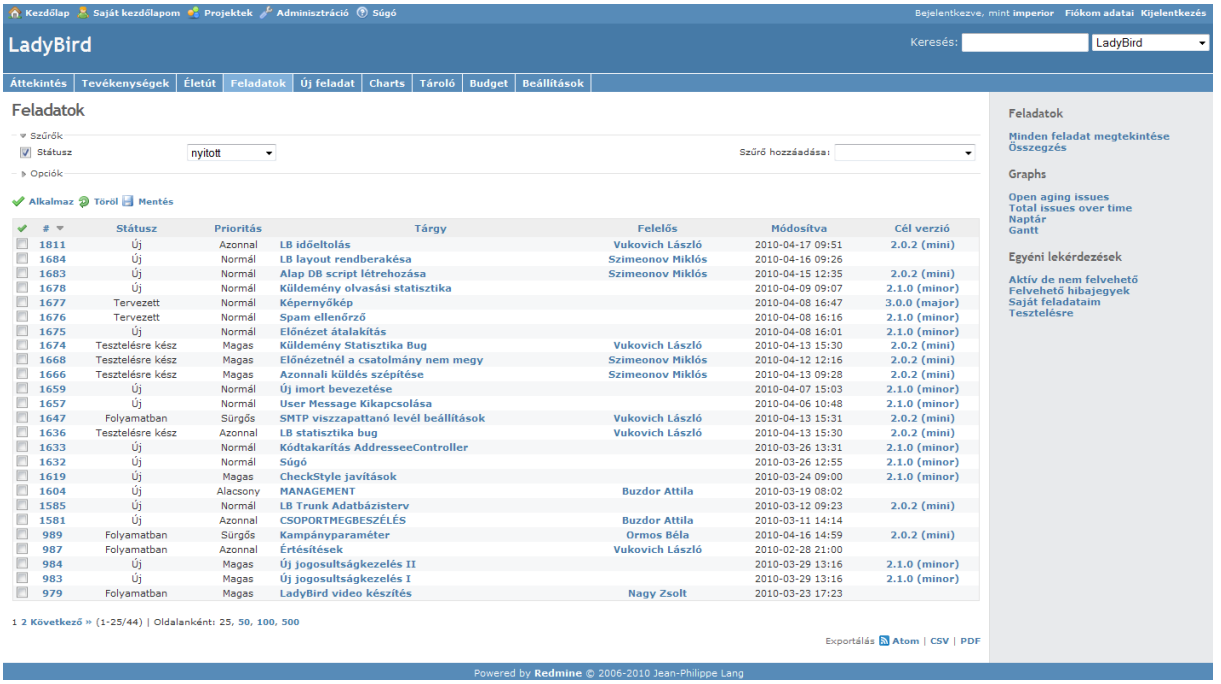
Nem véletlen az sem, hogy a fejlesztő tudja feljegyezni magának a ledolgozott órákat. Ha kötött munkaidővel dolgozunk egy irodában, ennek akkor is van haszna, bár ebben az esetben kevésbé jelentős. Ha viszont távmunkában fejlesztünk, csak így lehet naplózni a munkaidőnket. A hibajegyekről és a ledolgozott órákról egy jól felkészített rendszer részletes statisztikákat vezet és ezt a vezetők számára könnyen értelmezhető és olvasható módon jeleníti meg jelentések és grafikonok formájában. Nem utolsó szempont az sem, ha a választott hibajegykezelő elég népszerű és a felhasználók beépíthető plugin-okat írnak a rendszerhez.

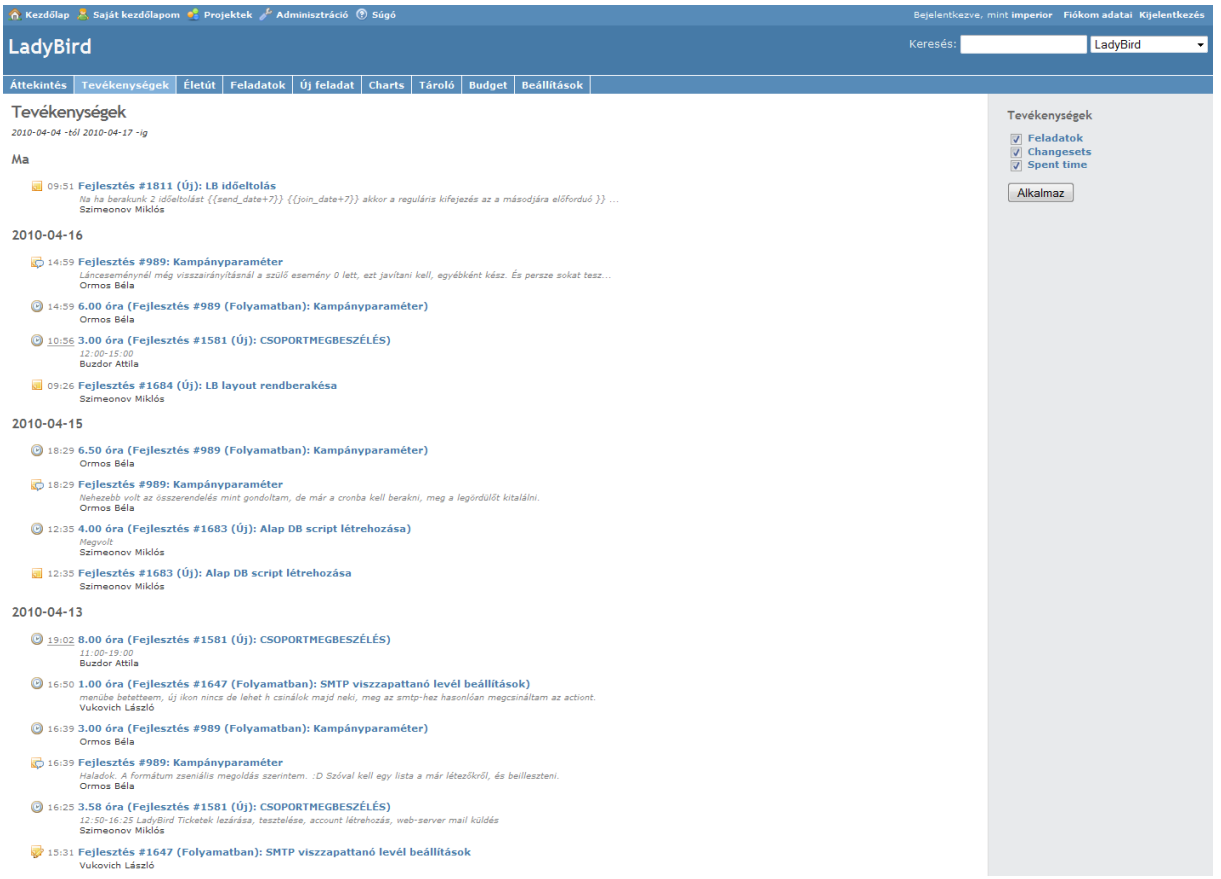
Az SCM kapcsolódásról már volt szó, ugyanakkor ennek is távmunkánál van nagyobb jelentősége. Revízióknak nevezzük a forráskódon történt változtatások egy fejlesztő által végrehajtott egyénileg meghatározott időintervallumban vett halmazát. Ezeket a változásokat egy jó rendszer könnyen átlátható formában képes megjeleníteni. Az egyes revíziókhoz hozzá kell tudnunk kötni a ticket-eket, és magát a fejleszteni kívánt alkalmazást is meg kell, hogy tudjuk tekinteni forráskód szinten.

## RedMine

A RedMine nevű hibajegykezelőt nem olyan régen vezettük be a WEB-SKY Consulting-nál. Sokáig a népszerűbb TRAC rendszert használtuk, azonban számos olyan funkcióval rendelkezik a RedMine alapból, ami a TRAC-nél csak úgynevezett „egg”-ek (plugin) formájában volt elérhető. A RedMine rendszer Ruby nyelven íródott, számos különböző SCM megoldást támogat, több projektet is kezelhetünk benne egyszerre, valamint a felhasználóknak nagyon részletesen beállíthatjuk a jogosultságait. Ismeri a Gantt diagrammot, naptárakat képes megjeleníteni, tetszőleges eseményről küld figyelmeztetést e-mailben akár, és nem utolsó sorban többnyelvű.

Íme néhány képernyőkép a RedMine rendszerből, az első a hibajegyek listáját mutatja, a második pedig az idővonalat (az események időrendben).




Kezdőlap
Saját kezdőlapom
Projektek
Adminisztráció
Súgó
Bejelentkezve, mint imperior
Fiókom adatai
Kijelentkezés
LadyBird
Keresés:
LadyBird
Áttekintés
Tevékenységek
Életút
Feladatok
Új feladat
Charts
Tároló
Budget
Beállítások
Tevékenységek
2010-04-04 -től 2010-04-17 -ig
Ma
09:51 Fejlesztés #1811 (Új): LB időeltolás
Na ha berakunk 2 időeltolást {{send_date+7}} {{join_date+7}} akkor a reguláris kifejezés az a másodjára előforduó }} ...
Szimeonov Miklós
2010-04-16
14:59 Fejlesztés #989: Kampányparaméter
Láncseménynél még visszairányításnál a szülő esemény 0 lett, ezt javítani kell, egyébként kész. És persze sokat tesz...
Ormos Béla
14:59 6.00 óra (Fejlesztés #989 (Folyamatban): Kampányparaméter)
Ormos Béla
10:56 3.00 óra (Fejlesztés #1581 (Új): CSOPORTMEGBESZÉLÉS)
12:00-15:00
Buzdor Attila
09:26 Fejlesztés #1684 (Új): LB layout rendberakása
Szimeonov Miklós
2010-04-15
18:29 6.50 óra (Fejlesztés #989 (Folyamatban): Kampányparaméter)
Ormos Béla
18:29 Fejlesztés #989: Kampányparaméter
Nehezebb volt az összerendelés mint gondoltam, de már a cronba kell berakni, meg a legördülőt kitalálni.
Ormos Béla
12:35 4.00 óra (Fejlesztés #1683 (Új): Alap DB script létrehozása)
Megvolt
Szimeonov Miklós
12:35 Fejlesztés #1683 (Új): Alap DB script létrehozása
Szimeonov Miklós
2010-04-13
19:02 8.00 óra (Fejlesztés #1581 (Új): CSOPORTMEGBESZÉLÉS)
11:00-19:00
Buzdor Attila
16:50 1.00 óra (Fejlesztés #1647 (Folyamatban): SMTP visszapattanó levél beállítások)
menübe betetteem, új ikon nincs de lehet h csinálok majd neki, meg az smtp-hez hasonlóan megcsináltam az actiont.
Vukovich László
16:39 3.00 óra (Fejlesztés #989 (Folyamatban): Kampányparaméter)
Ormos Béla
16:39 Fejlesztés #989: Kampányparaméter
Haladok. A formátum zseniális megoldás szerintem. :D Szóval kell egy lista a már létezőkről, és beilleszteni.
Ormos Béla
16:25 3.58 óra (Fejlesztés #1581 (Új): CSOPORTMEGBESZÉLÉS)
12:50-16:25 LadyBird Ticketek lezárása, tesztelése, account létrehozás, web-server mail küldés
Szimeonov Miklós
15:31 Fejlesztés #1647 (Folyamatban): SMTP visszapattanó levél beállítások
Vukovich László
Tevékenységek
Feladatok
Changesets
Spent time
Alkalmaz

# Elméleti megközelítés: tervezési minták

A szoftverfejlesztésben a tervezési minták általános újrahasználható megoldást biztosítanak egy-egy gyakran felbukkanó problémára, vagy problémakörre a fejleszteni kívánt szoftverünk megfelelő megtervezésével kapcsolatban. Sosem kész megoldásokat nyújtanak, amelyek közvetlenül kóddá formálhatók lennének. Csupán egy sablont, leírást biztosítanak arra, hogy egy adott problémát hogyan oldjunk meg különféle szituációkban.

A legismertebb tervezési minták az objektum-orientált világhoz köthetők. Ezek általában az egyes osztályok és egyes objektumok közti kapcsolatokat, interakciókat mutatják be anélkül, hogy konkrétan megszabnák, hogy mit hogyan használjunk ténylegesen. A gyakorlatban ezek a minták felgyorsítják a fejlesztés menetét azáltal, hogy tesztelt és bizonyítottan jól működő paradigmákat alkalmaznak. A hatékony szoftverfejlesztésben szükségünk lehet előre nem látható problémákat is figyelembe venni, ezt is könnyebbé tehetik számunkra. Általánosan elmondható, hogy rengeteg hibaforrást kiküszöbölhetünk alkalmazásukkal, nem utolsó sorban pedig – ha a fejlesztő vagy a tervező (architect) ismeri a tervezési mintát – a kód sokkal olvashatóbb és átláthatóbb lesz mindenki számára.

A tervezési mintákról rengeteg dokumentációt, könyvet találhatunk akár az interneten is, a dolgozatban csak a Zend Framework által használt design pattern-ekről lesz szó.

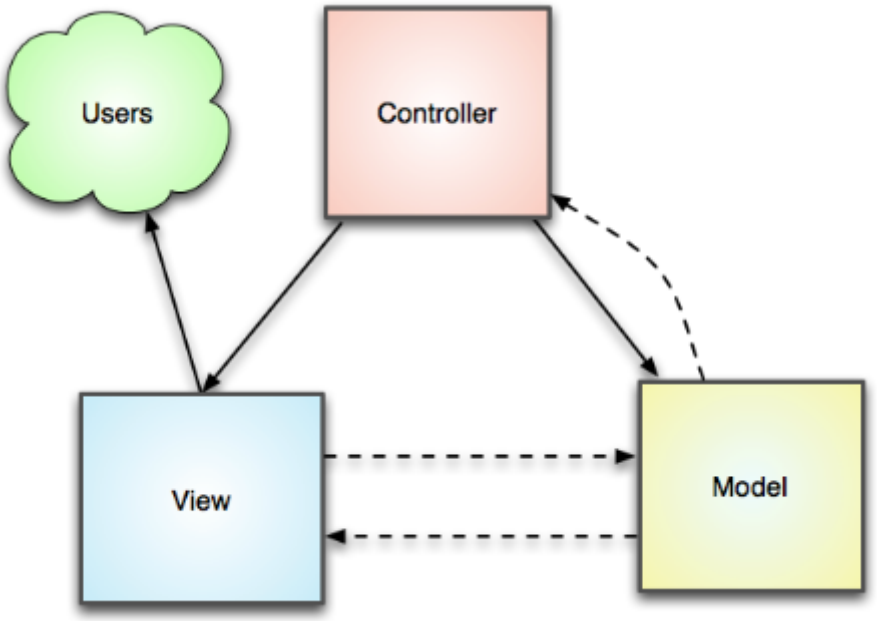
## *MVC Design Pattern*

A legfontosabb tervezési minta, melyet számos keretrendszer – nem csak a Zend Framework – implementál, a Model View Controller Design Pattern. Összetett, sok adatot a felhasználó elé táró számítógépes alkalmazásokban gyakori fejlesztői kívánalom az adathoz (model) és a felhasználói felülethez (view) tartozó dolgok szétválasztása, hogy a felhasználói felület ne befolyásolja az adatkezelést, és az adatok átszervezhetők legyenek a felhasználói felület változtatása nélkül. Az MVC ezt úgy éri el, hogy elkülöníti az adatok elérését és az üzleti logikát az adatok megjelenítésétől és a felhasználói interakciótól egy közbülső összetevő, a vezérlő (controller) bevezetésével.

A mintát Trygve Reenskaug írta le először 1979-ben, miután a Smalltalk-on dolgozott a Xerox kutatói laborjánál. Az eredeti megvalósítás részletesen a nagyhatású „Applications Programming in Smalltalk-80: How to use Model-View-Controller” című tanulmányban olvasható.

Gyakori egy alkalmazás több rétegre való felbontása: megjelenítés (felhasználói felület), tartománylogika és adatelérés. Az MVC-ben a megjelenítés tovább bomlik nézetre és vezérlőre. Az MVC sokkal inkább meghatározza egy alkalmazás szerkezetét, mint az egy átlagos programtervezési mintára jellemző.

Az MVC leginkább a webes alkalmazások esetén elterjedt minta, érthető okokból, hiszen a webalkalmazásoknál a kód 100%-a gyakorlatilag pont erre a három részre osztható. A Zend Framework esetén a tervezési minta a fejleszteni kívánt rendszerünk mappaszerkezetét is meghatározza. Sok programozó – akik már használják az MVC mintát – elengedhetetlen eszközként tekint erre a szeparáltságra, a kódok rendezettekké válnak, és az esetek döntő többségében egyből egyértelműen tudjuk, hogy hol kell változtatni a kódon az esetleges hibák esetén.




Habár az MVC-nek sok értelmezése létezik, a vezérlés menete általánosságban a következőképp működik:

1. A felhasználó valamilyen hatást gyakorol a felhasználói felületre (pld. megnyom egy gombot).
2. A vezérlő átveszi a bejövő eseményt a felhasználói felülettől, gyakran egy bejegyzett eseménykezelő vagy visszahívás útján.
3. A vezérlő kapcsolatot teremt a modellel, esetleg frissíti azt a felhasználó tevékenységének megfelelő módon (pld. a vezérlő frissíti a felhasználó kosarát). Az összetett vezérlőket gyakran alakítják ki az utasítás mintának megfelelően, a műveletek egységbezárásáért és a bővítés egyszerűsítéséért.
4. A nézet (közvetve) a modell alapján megfelelő felhasználói felületet hoz létre (pld. a nézet hozza létre a kosár tartalmát felsoroló képernyőt). A nézet a modellből nyeri az adatait. A modellnek nincs közvetlen tudomása a nézetről.
5. A felhasználói felület újabb eseményre vár, mely az elejéről kezdi a kört.

### Model – Az adatok, amikkel dolgozunk

Az alkalmazás által kezelt információk tartomány-specifikus ábrázolása. A tartománylogika jelentést ad a puszta adatnak (pld. kiszámolja, hogy a mai nap a felhasználó születésnapja-e, vagy az összeget, adókat és szállítási költségeket egy vásárlói kosár elemeihez). Sok alkalmazás használ állandó tároló eljárásokat (mint mondjuk egy adatbázis) adatok tárolásához. Az MVC nem említi külön az adatelérési réteget, mert ezt beleérti a modellbe. A ZF esetében gyakorlatilag táblánként 2 osztály implementációját jelenti a modell elkészítése. Az egyik osztály a táblát fogja reprezentálni, a másik pedig annak egy rekordját. Így mikor lekérdezünk adatokat az adatbázisból, teljesen OO módon tehetjük ezt meg. És hogy mik is kerülnek a modellekbe? Egyik részről, ha megadtuk a szükséges adatokat (táblanév, elsődleges kulcs, kapcsolatok…), akkor magát az adatot már reprezentáltuk is. Másik részről az osztályba tetszőleges metódusokat írhatunk, ez pedig már az adatelérési réteg lenne. Sokan, akik MVC alkalmazást fejlesztenek, abba a hibába esnek, hogy az adatelérési réteget a controller-be veszik, pedig a modellbe kéne.

### View – A megjelenítés művészete

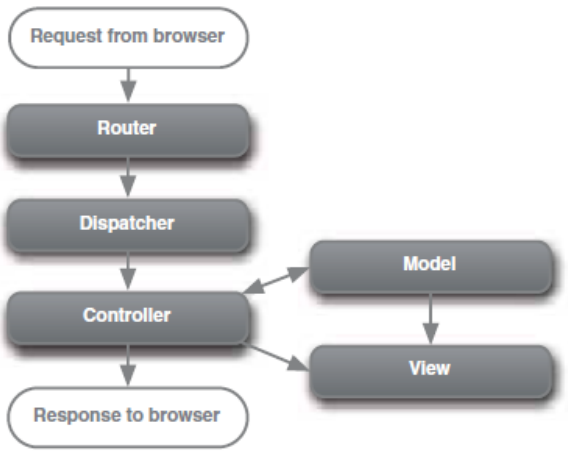
A view az, amit a felhasználó lát maga előtt. Megjeleníti a modellben tárolt adatokat egy megfelelő alakban, mely alkalmas a felhasználói interakcióra, jellemzően egy felhasználói felületi elem képében (táblázat, űrlap, menü, stb…). Különböző célokra különböző nézetek létezhetnek ugyanahhoz a modellhez. A nézet gyakran adatokat gyűjt a felhasználóktól. Az MVC-t használó webes alkalmazásokban ez az a része a kódnak, ahol HTML nyelvvel találkozhatunk. Esetünkben az ún. view script-ek gyakorlatilag phtml állományokat jelentenek. Használható az alapértelmezésben felkínált ZF-es sablonozó rendszer, de akár más, komplexebb ilyen eszközöket is használhatunk (pld. Smarty). Alapvető szabály a view esetében, hogy olyan embernek is ki lehessen adni a felület programozását, aki esetleg nem ért a PHP nyelvhez. Tehát a PHP kódokat minimalizálni kell. Természetesen teljesen nem tudjuk eltüntetni, mert a felületnél is szükségünk lesz iterációra, feltételekre, és a view-nak átadott adatokat valahol meg is kell jelenítenünk. Viszont az egyszerű hívásokon, kiíratásokon, vezérlési szerkezeteken kívül más utasítások nem kerülhetnek ide. A PHP az

ilyen esetekre biztosít egy alternatív szintaxist,[6] amely kifejezetten olyan esetekben hasznos, amikor legfeljebb egy soros PHP script-eket szúrunk be a HTML kódba. A view esetében pont ilyen fájlokat fogunk készíteni.

### Controller – Vezérlők az összekötő kapocs szerepében

A controller köti össze az egész mintát egy nagy egységgé. Az eseményeket, jellemzően felhasználói műveleteket dolgozza fel és válaszol rájuk, illetve a modellben történő változásokat is kiválthat. A Zend-nél a controller-ek osztályok formájában jelennek meg. Az osztályok metódusai az ún. action-ök. Ezek a megkülönböztetett metódusok nagyon fontos szerepet kapnak a fejlesztés során. A vezérlők megtervezésének nagy szerepe van a rendszertervek készítésénél. Általánosságban elmondható, hogy minél több vezérlőt írunk, minél kevesebb kóddal, annál tagoltabb, annál átgondoltabb a rendszer struktúrája.

A Zend MVC komponensei maradéktalanul implementálják a fent vázolt eszközöket. A `Zend_Controller` a controller rétegért felelős, a `Zend_View` a `Zend_Layout` mellett a megjelenítésért, míg a `Zend_Db` a modell réteget valósítja meg. Ezen komponensek egyenként is megvalósítanak különféle tervezési mintákat, a `Zend_Controller` a Front Controller mintát, a `Zend_Layout` a Composite View mintát, a `Zend_Db` pedig a Table Gateway mintát alkalmazza. Ezekről külön-külön is szót ejtünk a későbbiekben.




A Zend Framework esetén az MVC megvalósítása az ábrán látható módon zajlik. A böngészőn keresztül a felhasználótól jön egy kérés GET vagy POST formájában (pld. kattint egy linkre, elküld egy űrlapot, beír egy új címet, stb.). Ekkor az alapértelmezett, vagy akár általunk testreszabott útválasztó (router) és a dispatcher közösen kiválasztják, hogy melyik controller melyik action-je kell, hogy lefusson. Miután megkapta a vezérlést, a

[6] http://php.net/manual/en/control-structures.alternative-syntax.php

megfelelő action a modellel oda-vissza, a megjelenítéssel pedig csak egyirányban kommunikálva állítja elő a megjeleníteni kívánt weblapot, és ezt a böngészőben válaszként (response objektum) megküldi a felhasználónak.

A 3 réteg kommunikációját egy példán szemléltetve a legegyszerűbb bemutatni: Legyen az URL-ünk a következő: http://azendomainem.hu/forum/read/topic/6 Ez azt fogja jelenteni a router és a dispatcher számára, hogy a ForumController nevű vezérlőosztály readAction() nevű metódusát kell meghívnia, méghozzá a topic=6 paraméterrel. Értelemszerűen az adott action kiolvassa a modellből a 6-os azonosítójú fórumtéma bejegyzéseit, azt átadja a view-nak, amely megjeleníti a hozzászólásokat. Minden controller osztály minden metódusához (action) létezik saját view script állomány.

A modell és a nézet kettéválasztásával az MVC csökkenti a szerkezeti bonyolultságot, és megnöveli a rugalmasságot és a felhasználhatóságot. A Model View Controller tervezési mintának későbbi leszármazottja az MVP, amelyet szintén sok nagy technológia használ (pld. Google Web Toolkit).

## *Front Controller Design Pattern*

A Front Controller tervezési minta elég népszerű az MVC-t használó keretrendszerek körében. A ZF esetében ezt a `Zend_Controller_Front` osztály képviseli (ami ugyebár a `Zend/Controller/Front.php` állományt jelenti a konvenciók ismeretében). Ez az osztály Singleton, azaz egyszerre csak egyetlen példánya létezhet. Ez azt jelenti, hogy az osztályt nem tudjuk példányosítani, inkább lekérni tudjuk az egyetlen példányt (a statikus `getInstance()` metódus segítségével). Ahhoz, hogy részletesebben megértsük a Front Controller tervezési mintát, nézzük meg mi történik a HTTP kérésekkel a routing és a dispatching során.

A HTTP kérések elérését a ZF a `Zend_Controller_Request_Http` osztályon keresztül bonyolítja le. Itt található meg az összes kérés – a PHP-ból esetleg már ismert `$_POST`, `$_GET`, `$_COOKIE`, `$_SERVER`, `$_ENV` szuperglobális tömbök. Ezek tartalma, valamint az aktuális controller és action neve lekérhető. A router ezek alapján a paraméterek alapján eldönti, melyik kódrészlet fusson le. Ez a folyamat a ZF-ben felüldefiniálható (`Zend_Controller_Router_Rewrite`). A teljes URL-ünk legyen a következő cím:

```
http://azendomainem.hu/index.php?controller=news&action=list
```

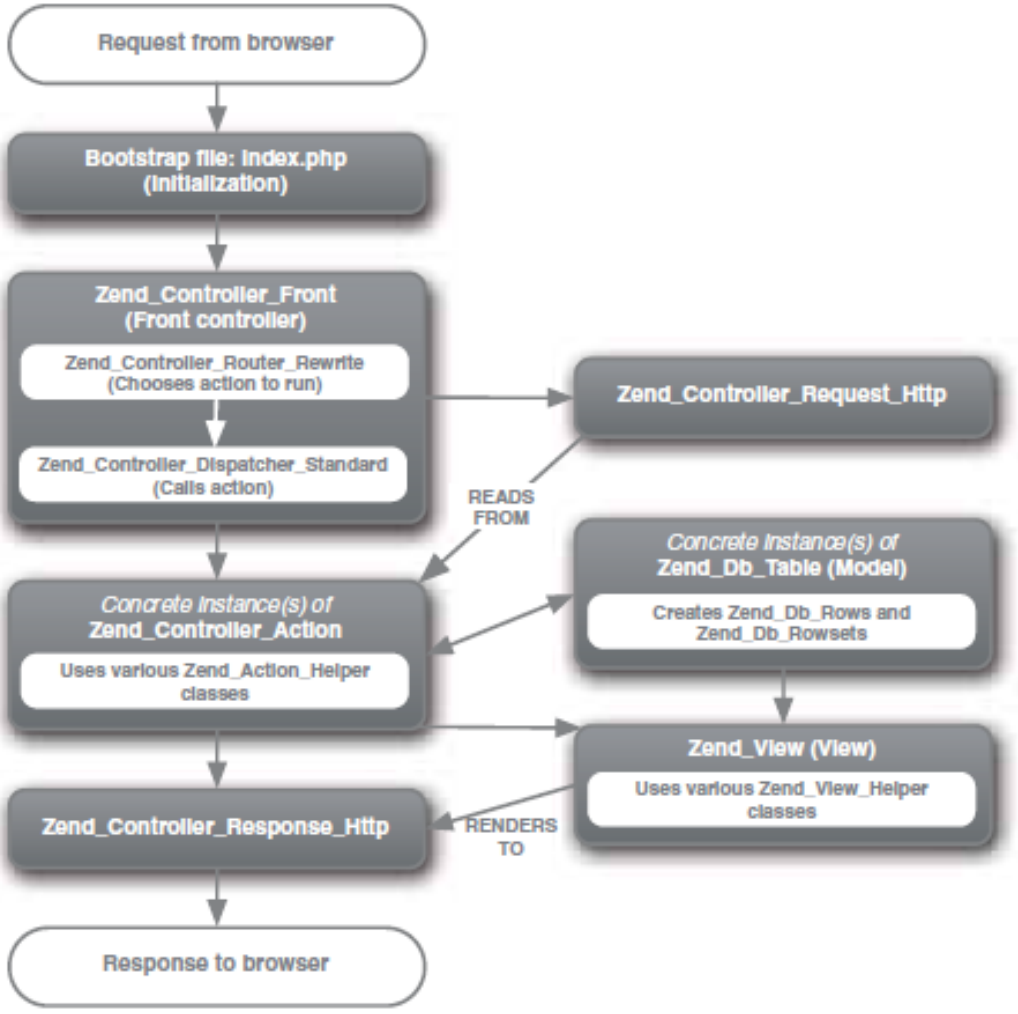
A felbontás ebben az esetben egyszerűen a `$_GET` lekérésével történik, a controller és action paraméterek kiértékelésével. Persze a legtöbb ZF alkalmazásban úgynevezett „csinos” URL-eket alkalmazunk, amelyek a keresőmotorok számára megkönnyítik az indexelést, így a fenti URL formája:

```
http://azendomainem.hu/news/list
```

Routing: A folyamat összefügg a dispatching folyamatával, ezek során dől el, hogy melyik kód fusson le adott cím esetén. Az útválasztót egy olyan osztály implementálhatja, ami alkalmazza a `Zend_Controller_Router_Interface` interfészt (akár magunknak is írhatunk ilyet). A teljes URL három részre osztható: az alap domain az, ahova a web-szerverünk mutat.

A második részről lesz szó a későbbiekben is, ez a baseUrl, ami az `index.php` elérési útját tartalmazza a gyökérhez képest. Esetünkben az `index.php` egyből a domain által mutatott helyen van, azaz a `http://azendomainem.hu` arra a mappára mutat, amelyben az állomány található. Ebben az esetben tehát a baseUrl üres. Látni fogjuk viszont, hogy ez nem minden esetben igaz. Az ezután álló paraméterekkel foglalkozik maga a router. Szeparálja a különböző értékpárokat (controller = news, action = list), és ezt könnyen értelmezhető formába hozza, majd átadja a dispatcher-nek.

Dispatching: Szétbontottuk az URL-t, eldöntjük, hogy mi fusson le, majd meghívjuk a megfelelő eljárásokat. Ezt a folyamatot nevezzük dispatching-nek. Miután meghatároztuk, hogy melyik controller-nek melyik action-je fusson le, az alkalmazásban esetlegesen egyéb kódokat is hívhatunk. Mint minden esetben, a ZF alapértelmezett dispatcher-e is tartalmazza a legtöbb kódot, amire szükségünk lehet, de természetesen saját egyéni dispatcher-t is definiálhatunk a Front Controller számára. Ezen osztályok mindegyike megvalósítja a `dispatch()` metódust, mely példányosítja a megfelelő controller-t és meghívja annak megfelelő metódusát.

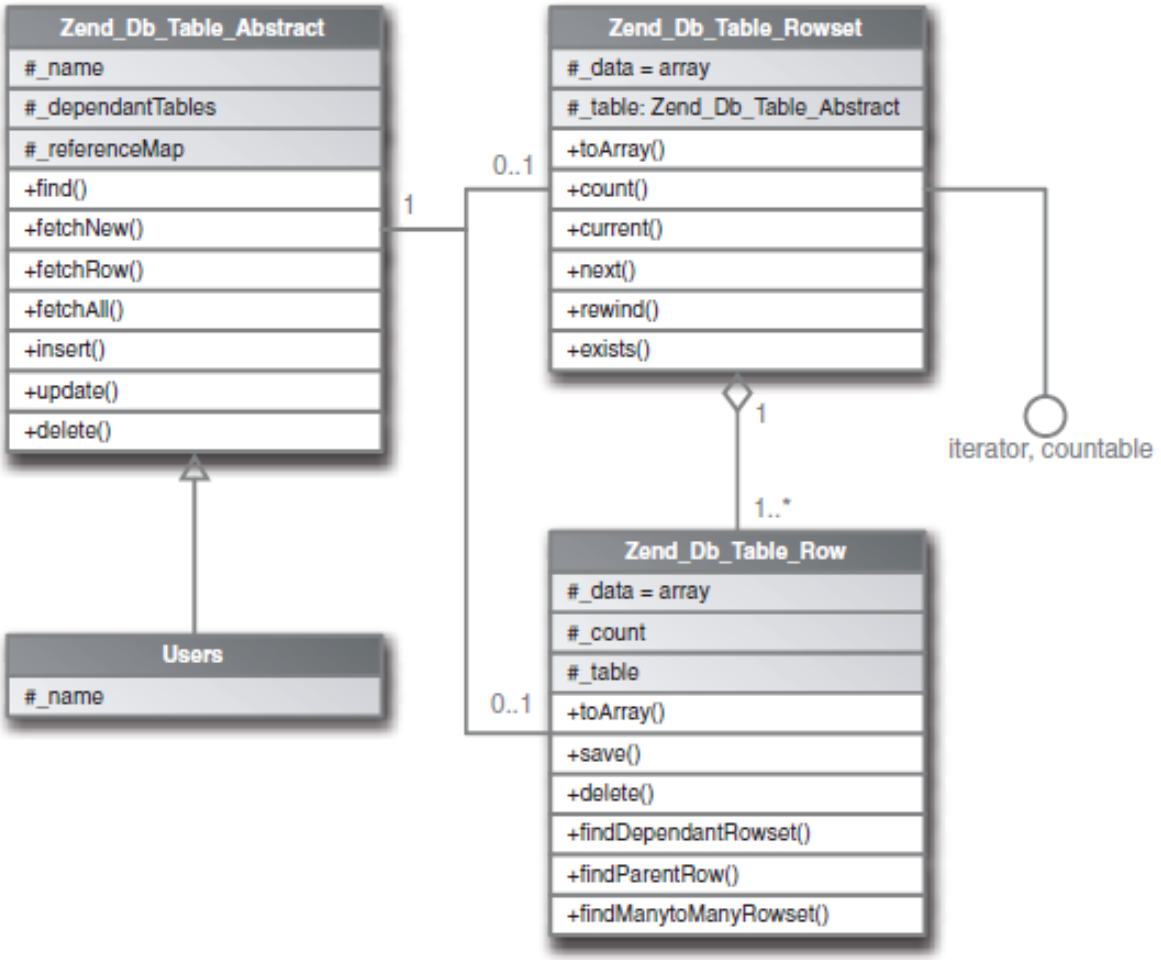



A fenti ábrán az látható, hogy egy MVC alkalmazásba hol és hogyan épül be a Front Controller tervezési minta.

## *Table Gateway Design Pattern*

Az MVC alkalmazásokban a modell képviseli az adatbázis fogalmát. Ez természetesen nem minden esetben relációs adatbázisokat jelent, de az esetek nagy részében igen. Ezért a Zend a relációs adatbázisok absztrakciójára alkalmaz egy újabb tervezési mintát, ez pedig a Table Gateway. Fontos, hogy ne SQL utasításokban gondolkodjunk, hanem erre épüljön egy absztrakciós réteg. Ennek a legegyszerűbb módja, ha osztályokat alkotunk, amik az általuk reprezentált adatokat képesek az adatbázissal szinkronizálni. Az egyes sorok (adatbázistáblabeli rekordok) a Row Data Gateway mintát alkalmazzák.

A `Zend_Db` komponens 3 fő részből tevődik össze a minta szempontjából (természetesen egyéb dolgokat is tud a `Zend_Db`). Ezek a `Zend_Db_Table`, a `Zend_Db_Table_Rowset`, valamint a `Zend_Db_Table_Row`. A `Zend_Db_Table_Abstract` osztályból fognak származni azok az osztályok, melyeket a táblák reprezentációjára készítettünk. Az alkalmazás futása során lekérdezéseket hajtunk végre a táblákon, ezek `Zend_Db_Table_Rowset` típusú objektumokként állnak rendelkezésünkre. Ezt az objektumot bejárva `Zend_Db_Table_Row` objektumokat kapunk, amik az egyes sorokat adják.


Zend_Db_Table_Abstract
#_name
#_dependantTables
#_referenceMap
+find()
+fetchNew()
+fetchRow()
+fetchAll()
+insert()
+update()
+delete()
Users
#_name
Zend_Db_Table_Rowset
#_data = array
#_table: Zend_Db_Table_Abstract
+toArray()
+count()
+current()
+next()
+rewind()
+exists()
Zend_Db_Table_Row
#_data = array
#_count
#_table
+toArray()
+save()
+delete()
+findDependantRowset()
+findParentRow()
+findManytoManyRowset()
1
0..1
1
1..*
0..1
iterator, countable

## *Composite View Pattern*

A view az MVC alkalmazásoknak az a része, amivel a rendszer felhasználói közvetlenül interakcióban vannak, ezért fontos átlátni a működését. Minden webes alkalmazásnak – még a legkisebb honlapoknak is – meg kell jelenítenie bizonyos adatokat az egyedi oldalakon. Ez a legtöbbször kimerül header-, footer-, menü- és tartalmi részekben, de természetesen ennél komplexebb is lehet egy oldal felépítése.

A minta lényege, hogy közös elemeket szeretnénk megjeleníteni minden egyes oldalon, ezért másolni kellene a kódokat viewscript-től viewscript-re. A legegyszerűbb módja ennek, ha két `include()` utasítással a tartalmi rész előtti és utáni részt beillesztjük a kódba, valahogy így:

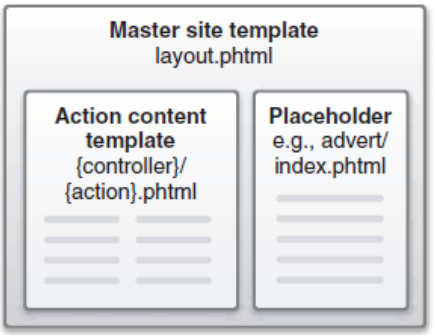
```
<? include('header.phtml'); ?>
    <h1>page title</h1>
    <p>text here</p>
<? include('footer.phtml'); ?>
```

A legfőbb probléma ezzel a megoldással, hogy minden egyes viewscript-be be kell tennünk. Hogy ezt elkerüljük, egy jobb megoldás, ha ez a szétválasztás (header, content, footer stb…) automatikusan történik. Két tervezési minta is van erre megoldásként, a „Two Step View”, valamint a „Composite View”.

A „Two Step View” lényege, hogy először az adatokat mindenféle formázás nélkül megkapja a megjelenítés, majd csak második lépésben specializáljuk a kinézetet.

A „Composite View” nevéből adódóan az összetett megjelenítést valósítja meg egy dinamikus újrafelhasználható sablonrészekből álló formában. Elsősorban minél kisebb atomi egységekből állítsuk össze a weblap vázát, és mindig hagyjuk meg a lehetőséget arra, hogy tetszőlegesen variálhassuk ezeket az elemeket, olyan esetekre, mikor úgy adódna, hogy módosítani szeretnénk a kinézeten.

Zend Framework-ös környezetben ez úgy valósul meg, hogy a controller-ek egyes metódusaihoz (action-ök) tartozik egy-egy viewscript. Ezek a viewscript-ek főleg HTML kódot tartalmaznak, valamint az action tud átadni adatokat neki változók formájában. Ezenkívül a `Zend_Layout` segítségével definiálhatjuk a layout-ot (oldalunk vázát) egy saját phtml-ben. A layout script tartalmazza tehát az oldal azon részeit, amelyek nem függnek az action-öktől. Ilyen részek lehetnek például a fejléc, a lábléc és a menü. A layout biztosítja számunkra a „Composite View” lehetőségét.




Természetesen előfordulhat, hogy egy-egy action esetén más layout-tal szeretnénk megjeleníteni az oldalt, ez tetszőleges controller metódusban megtehető.

# *Bootstrapping*

A korábbi Zend Framework verziókban gyakran úgy emlegettem ezt a részét a fejlesztésnek, mint amit titkolni kell, amit senki sem szeret, mert a kód a bootstrap-ben csúnya volt, rendezetlen, érthetetlen, stb. A legkevésbé a rendszernek ezt a részét lehetett átalakítani objektumorientált formára. Azonban az idők haladtával erre a Zend-es csapat is rájött, és nagy hangsúlyt fektettek a rész finomítására. És bizony meg is lett az eredménye.

Mi is az a bootstrap? Lefordítva nem sok értelme van a szónak, ezzel a szóval írja le egyszerűen az angol terminológia azt az eljárást, amikor a kódot inicializáljuk, valamint felvértezzük mindenféle speciális konfigurációval. Érdemes egy pillanatra belegondolni mi a különbség egy HTTP protokollon futó alkalmazás és egy asztali alkalmazás közt. Míg az asztali alkalmazásoknál folyamatos a kapcsolat (a program egyszerűen FUT), addig a http gyakorlatilag kérés-válasz alapú, állapotmentes protokoll, amint visszajött a válasz, a program nem fut semmiféle formában. Ha elnavigálunk az oldalról, újabb kérés megy a szerver felé, jön a válasz, majd megint leáll a működés. Azaz vannak olyan dolgok, amikre minden egyes „oldalbetöltődéskor” szükségünk lesz. Ilyen például maga a keretrendszer inicializálása, a jogosultságkezelés, az adatbáziskapcsolat, és még sorolhatnám. A bootstrap fájl ezeket a konfigurációkat foglalja magában.

Amióta kijött az 1.8-as Zend Framework, azóta az `application/Bootstrap.php` állományban található meg a Bootstrap osztály. Magát a bootstrapping-et a Zend egy külön komponensbe helyezte át, ez a `Zend_Application` nevet kapta. Ez a rész 3 állományt érint szorosabban, a konfigurációs fájlt (`application/configs/application.ini`), az apache webszerver `.htaccess` állományát, valamint magát a `Bootstrap.php`-t. Szintén a bootstrapping feladata a Zend Framework autoloading eljárása, mely csak a szükséges állományokat tölti be a keretrendszerből, de azokat automatikusan (azaz nincs szükség az egyes osztályok include-olására a kódban).

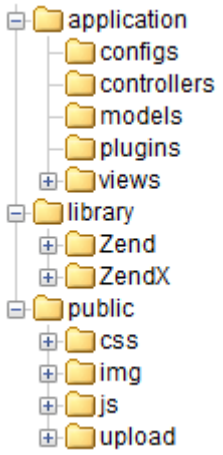
A Zend_Application a bootstrapping-et különálló részekben oldja meg:

- A `Zend_Application` osztály maga a PHP környezet inicializálásáért, a Zend Framework betöltéséért, az elérési utak meghatározásáért, valamint a konkrét bootstrap fájl meghívásáért felelős.
- A `Zend_Application_Bootstrap` interfészeket biztosít a bootstrap számára.
- A `Zend_Application_Bootstrap_Bootstrap` az általunk elkészítendő állomány ősosztálya, a metódusokat természetesen felüldefiniálhatjuk.
- A `Zend_Application_Resource` pedig erőforrásokat biztosít, amik segítségével a bootstrap fájlunkat minimalizálhatjuk. Akár a konfigurációs állományban is megadhatók az egyes resource-ok (mint pld. adatbáziskapcsolat, menü, hozzáférés vezérlő listák, plugin-ek, stb…), és így egy egyszerűbb alkalmazás esetén a bootstrap fájl akár el is hagyható (ilyenkor az ősosztály fut le).

A későbbiekben lesz még szó a fentebb említett állományokról, a bootstrapping, mint tervezési minta a leírtakban kimerül.

# Egy ZF alkalmazás felépítése

Az említett tervezési minták meghatározó szerepűek az alkalmazásaink struktúrájában. Az MVC meghatározza a szoftver alapvető fájljainak elhelyezkedését, a Table Gateway a modellek szervezését oldja meg, a Composite View pedig a viewscript-ek elhelyezkedését és a layout helyét határozza meg. Fontos megemlítenünk, hogy bármelyik alapértelmezés tetszőlegesen felülírható, ez az egyik hatalmas előnye a Zend Framework-nek a többi keretrendszerrel szemben.




Az alapértelmezett felépítést a legegyszerűbben úgy tudjuk megnézni, ha a mappaszerkezetet kigeneráltatjuk a Zend-del. Ha létrehozunk egy ZF projektet, a mappaszerkezet és néhány alapvető állomány készül el. Mint azt korábban említettem, a Zend is biztosít automatikusan generálható kódokat a fejlesztőknek, így egy üres projektet is generáltathatunk a `Zend_Tool` segítségével, ami elkészíti nekünk az ábrán látható szerkezetet. Ennek lépései igen egyszerűek, a Zend dokumentációjában megtalálható a leírás.[7] Az így kigenerált, vagy általunk „összedobott” mappaszerkezet lesz ebben a fejezetben ismertetve.

[7] http://framework.zend.com/manual/en/learning.quickstart.create-project.html

## *Az application mappa*

Az `application` mappában helyezkednek el alkalmazásunk fájljai, forrásállományai. A webszerver számára közvetlenül elérhetetlen a tartalma, ugyanis a webszerver a `public` mappa állományit látja (a legtöbb esetben). Az irányítást a PHP adja át az itt található állományoknak, a böngészőből nem tudjuk elérni a fájlok tartalmát. Mint látható, az `application` mappán belül válik szét fizikailag is az MVC három alkotóeleme.

- Konfigurációs állomány(ok): `application/configs`

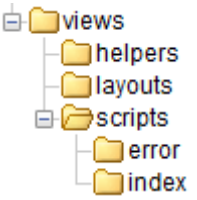
  A Zend kétféle lehetőséget biztosít a konfigurációs kulcs-érték párok letárolására. Ezek az XML és az INI formátumok. Az INI formátum talán az emberi szemnek olvashatóbb, ezért leginkább azt alkalmazzuk. A konfigurációs fájlokról lesz szó egy későbbi fejezetben részletesebben.
- A vezérlő osztályok: `application/controllers`

  Itt találhatóak az MVC-ből már megismert controller-ek. Ezek PHP osztályok, nevük a `ValamiController.php` mintára illeszkedik. Általánosságban elmondható, minél több controller van egy alkalmazásban, annál jobban és logikusabban lett megtervezve. A Zend alapból kigenerál nekünk 2 ilyen osztályt, az egyik az IndexController, mely az alapértelmezés, ha nem adunk meg controller-t, valamint az ErrorController-t, ami a hibák, kivételek kezeléséért felelős.
- A modellek: `application/models`

  Ebben a mappában a modellek találhatók. Szokás tovább bontani attól függően, hogy az adott állomány rekordot, táblát, vagy épp egyéb, modellhez kapcsolódó dolgot valósít meg.
- Beépülők: `application/plugins`

  Az alapértelmezett plugin-okon kívül saját kis kódokat is írhatunk, ezeket helyezhetjük el itt. Mindig érdemes átgondolni, hogy mit kell plugin formájában megírni, általában az olyan php kód igényű beépülő script-eket, amik minden egyes oldal letöltődésnél szerepet játszanak. Tipikus példa erre a kivételkezelés, a jogosultságkezelés, a naplózás, stb.

- A megjelenítés: `application/views`

  Ez a mappa alapértelmezésben is tovább van strukturálva. Minden, ami a megjelenítéshez tartozik, itt van letárolva. Az MVC-ből korábban megismert phtml-es viewscript-ek helye a `views/scripts/controllernév` mappa és a viewscript neve `actionnév.phtml` mintára igazodik. Itt találhatóak a Composite View-ból megismert layout-ok is. A `helpers` almappa pedig olyan PHP segédosztályokat tartalmaz, melyek a megjelenítés előtti közvetlen feldolgozást nyújtják. Ezek a helper-ek a viewscript-ek saját metódusaiként hívhatók, azaz a phtml állományokban a `$this->metódusNév()` hívással érhetjük el őket. Jó példa view helper-re ha egy alkalmazás MySQL-es adatbázisában minden dátumot DATETIME típusként, vagy TIMESTAMP-ként tárolunk, de csak helyenként van szükség a teljes dátum kiíratására, a legtöbb helyen elég csak napi pontossággal megjeleníteni azt. Ekkor helperünk a megkapott formátumból a nekünk tetsző alakra hozza a dátumot. A ZF rengeteg beépített view helper-t támogat.

## A library mappa

A `library` mappában található maga a Zend Framework keretrendszer, illetve ha egyéb keretrendszereket szeretnénk integrálni, azt is itt tehetjük meg. Általános elvárás lehet, hogy a library-ben lévő állományokat nem bántjuk, csak ha a rendszer alatt be szeretnénk frissíteni a ZF-et. Érdemes lehet még megjegyezni, hogy a library-ben csak a hivatalosan kiadott osztályok szerepelnek, az aktuális ZF verzióból. Emellett ha letöltjük a keretrendszert, találunk egy `incubator` névvel ellátott mappát is, ebben a bevezetésre szánt fejlesztői eszközöket találjuk, melyek valamilyen okból még nem érettek meg rá, hogy bekerüljenek a hivatalos library-be, de érdekes lehet egyik másik használata.

Minden ZF alkalmazásban fontos lesz, hogy a library mappa benne legyen az `include_path` tartalmában, azaz, hogy a library mappa tartalmát úgy tudjuk tallózni, mintha az lenne az abszolút hivatkozás gyökérmappája. Így ha egy ZF osztályt szeretnénk példányosítani, akkor nem kell beírnunk a teljes elérési útvonalat, hanem az állományra egyszerűen `Zend/…almappák…/Osztálynév.php`-ként hivatkozhatunk (bár erre az automatikus betöltés miatt nem lesz konkrétan szükségünk). Így a konvencióknak megfelelően, ha a `C:\…apache_root…\projekt_neve\library\Zend\Form.php` állományt szeretnénk betölteni, csupán a `$form = new Zend_Form();` utasításra lesz szükségünk, és nem kell foglalkozni azzal, hogy hol van a file, és hogyan kell azt betölteni.

## A public mappa

A `public` mappa tartalmazza a képeket, stíluslapokat (CSS), JavaScript-eket (JS), és alapvetően minden olyan fájlt, amit a böngésző számára közvetlenül elérhetővé szeretnénk tenni. Itt helyezkedik el az `index.php`, amire – ha alkalmazásunknak saját domain-je van – a webszerver belépési pontként mutat. Találunk még egy `.htaccess` állományt is, ami szintén a webszervernek ad utasításokat (pld. hogy felkészítse a „csinos" url-ek használatára, és megadja a kivételeket). A gyakorlatban egy ilyen apache konfiguráció a következőképpen fog kinézni.

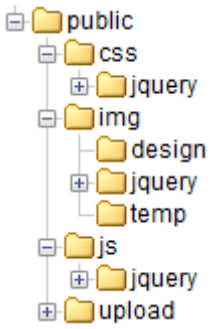
```
SetEnv APPLICATION_ENV devel

RewriteEngine On
RewriteRule !\.(js|gif|jpg|png|css|swf)$ index.php
RewriteRule ^.*/css(.*) css$1 [PT]
RewriteRule ^.*/img(.*) img$1 [PT]
RewriteRule ^.*/js(.*) js$1 [PT]
RewriteRule ^.*/temp_data(.*) temp_data$1 [PT]
RewriteRule ^.*/contrib(.*) contrib$1 [PT]
```

A fájl első sorában egy konstanst definiálunk. Itt azt adtuk meg, hogy az alkalmazásunk jelenleg a devel verzióban tart (azaz még fejlesztői). Nagyon praktikus megoldás ez, ugyanis erre az egyetlen konstansra hivatkozva adhatunk meg további konfigurációkat. Azaz, ha a rendszerünket tesztelési fázisba szeretnénk tenni, vagy épp ki szeretnénk tenni éles verzióként ügyfeleink számára, nyilván más adatbázist fog használni, esetlegesen más hibakezelést (fejlesztésnél értesülni szeretnénk a hibákról, míg az ügyfelek elől érdemes elrejteni azokat). Így ha a rendszert át szeretnénk állítani egy másik állapotba, feltöltés (másolás) után csak ezt az egyetlen szót kell átírnunk a megfelelőre, és semmi máshoz nem kell hozzányúlnunk.

Az állomány további részében az Apache szervernek adunk utasításokat, hogy megfelelően tudja kezelni a Zend-es URL formátumot. Nagyjából lefordítva azt adjuk meg neki, hogy bármit kap az URL-ben, ami nem illeszkedik a felsorolt végződésekre, adja át a vezérlést az `index.php`-nek, hadd döntse el ő, mit kell megjeleníteni. Az egyes kivételeket pedig

megadjuk, hogy milyen mappákban keresse az Apache. Miért van erre szükség? A képeket szintén itt tároljuk a public mappán belül. Ha az oldalon van egy `<img src='/img/fejlec.png'>` képünk, akkor a Zend-es URL formátumnak megfelelően az `img` nevű controller `fejlec.png` nevű action-jét kellene megkeresnünk, ami nyilván nem létezik. Így mivel a végződés `png`, a `.htaccess` látja, hogy ez kivétel, és nem adja át az `index.php`-nek a vezérlést, hanem a `public/img/fejlec.png`-t fogja keresni, és ez a helyes feloldás.




A fentieknek megfelelően a `public` mappa további almappákat tartalmazhat. Érdemes a webalkalmazáshoz tartozó képeket egy mappába tenni, a CSS stíluslapokat és a JavaScripteket szintén külön elhelyezni, illetve – az alkalmazás jellegétől függően – ha a rendszert használó emberek is tudnak file-okat létrehozni a szerveren, azokat is szintén el kell különíteni. Egy példa erre a baloldali ábrán látható.

# A Zend Framework komponensei

Az eddigi fejezetekből megismerkedhettünk egy Zend Framework-ös alkalmazás felépítésével, a használt tervezési mintákkal, és bármilyen hihetetlen, gyakorlatilag az API dokumentációt használva, már képesek vagyunk önállóan is megírni egy ilyen rendszert. Fogalmazhatunk úgy is, hogy ha az eddigieket teljesen tisztába raktuk, akkor a nehezén már túl vagyunk.

Azonban van a keretrendszernek néhány fontos komponense, amelyekről érdemes bővebben is beszélnünk, az alapvető MVC osztályoktól, az űrlapok kezelésén át, az e-mailek kiküldéséig. A Zend Framework a dolgozat megírásának időpontjában a következő (egyébként beszédes nevű) csomagokat, komponenseket kínálja a fejlesztők számára – ezek részletes ismertetése túlmutat a diplomamunka keretein, de félkövérrel szedve láthatóak a bemutatni kívánt eszközök:

- **Zend_Acl: Jogosultságkezelés**
- Zend_Amf: Flash és PHP közti kommunikáció
- **Zend_Application: Bootstrapping, alkalmazás-inicializálás**
- **Zend_Auth: Authentikáció, beléptetés, azonosítás**
- Zend_Barcode: Vonalkódkezelés
- Zend_Cache: Gyorsítótárak használata
- Zend_Captcha: Robotok és valós személyek kiszűrése
- Zend_CodeGenerator: PHP állományok generálása OO eszközökkel
- **Zend_Config: Konfigurációs eszközök**
- Zend_Config_Writer: Konfigurációk dinamikus írása, betöltése
- Zend_Console_Getopt: CLI kezelés, argumentumok szétválasztása
- **Zend_Controller: MVC Controllerek, Front Controller Pattern**
- Zend_Currency: Különböző pénznemek kezelése
- Zend_Date: Dátumok, formátumok, időzónák

- **Zend_Db: MVC Modellek, Table Gateway Pattern**
- **Zend_Debug: Változók teljes tartalmának kiíratása**
- Zend_Dojo: DOJO JavaScript-es függvénykönyvtár kezelése
- Zend_Dom: DOM objektumok, struktúrák kezelése, lekérdezései
- Zend_Exception: Kivételkezelés ősosztálya
- Zend_Feed: RSS és Atom kezelése
- Zend_File: Állománykezelés, írás, olvasás
- Zend_Filter: Bemeneti adatok szűrése, felesleg elhagyása
- **Zend_Form: Felhasználói űrlapok kezelése**
- Zend_Gdata: Google Data API-k, online szolgáltatások elérése
- Zend_Http: HTTP kérések kezelése, feldolgozása
- Zend_InfoCard: InfoCard típusú felhasználói azonosítás kezelése
- Zend_Json: JSON formátum és PHP formátum közti konverzió
- **Zend_Layout: Composite View Pattern, sablonok kezelése**
- Zend_Ldap: Lightweight Directory Access Protocol szolgáltatások
- Zend_Loader: Automatikus betöltések
- Zend_Locale: I18N
- Zend_Log: Naplózás, naplófájlok kezelése
- **Zend_Mail: Elektronikus levelezéssel kapcsolatos tennivalók**
- Zend_Markup: Különböző jelölőnyelven írt adatok feldolgozása
- Zend_Measure: Mértékegységek kezelése
- Zend_Memory: Limitált memóriakapacitás megfelelő lekezelése
- Zend_Mime: MIME típusú üzenetek kezelése
- Zend_Navigation: Dinamikus menük, oldaltérképek
- Zend_Oauth: Külső forrásból történő azonosítás
- Zend_OpenId: OpenID API, beléptetés OpenID-val, külső azonosítás
- Zend_Paginator: Lapszámozás nagyméretű adathalmazok esetén
- **Zend_Pdf: PDF állományok létrehozása**
- Zend_ProgressBar: Feldolgozottságot mutató eszközök
- Zend_Queue: Várakozási sorok kezelése

- Zend_Reflection: PHP Reflection, objektumok, melyek önmagukkal térnek vissza
- **Zend_Registry: Egy HTTP munkameneten belüli adattároló (nem session)**
- Zend_Rest: REST webszolgáltatások
- Zend_Search_Lucene: Apache Lucene keresőmotor
- Zend_Serializer: Szerializáció oda-vissza
- Zend_Server: Szerveroldali kommunikáció, osztályok kezelése
- Zend_Service: Különféle webszolgáltatások elérése
- **Zend_Session: Munkamenetek kezelése**
- **Zend_Soap: Simple Object Access Protocol XML-ek kezelése, kommunikáció**
- Zend_Tag: Címkézés, különböző elemek jelölése
- Zend_Test: Egységtesztekkel kapcsolatos tenniovalók
- Zend_Text: Szöveg alapú rajzok kezelése (ASCII és UTF-8 art), FIGlet Text formátum
- Zend_TimeSync: Internetes pontos idő lekérése
- Zend_Tool: Zend CLI parancsok kezelése
- Zend_Tool_Framework: Zend CLI programozása
- Zend_Tool_Project: Projektek létrehozása Zend_Tool segítségével
- Zend_Translate: Többnyelvűsítés, fordítások
- Zend_Uri: Egységes erőforrás-azonosítók manipulációja
- **Zend_Validate: Adatok ellenőrzése, validálása**
- Zend_Version: Aktuális ZF verzió lekérése
- **Zend_View: MVC View**
- Zend_Wildfire: FireBug és PHP közti kommunikáció
- Zend_XmlRpc: Távoli eljáráshívások XML alapon
- ZendX_Console_Process_Unix: Konzolos környezet, párhuzamos futtatás
- **ZendX_JQuery: JQuery AJAX függvénykönyvtár alkalmazása**

Először nézzük át az MVC egyes komponenseit, majd a leggyakrabban használt elemektől a legkevésbé használt elemekig haladunk. Mint látható, a Zend ettől is sokkal többet nyújt.

# *Adatbáziskezelés modellekkel (Zend_Db)*

A webalkalmazásokban hatalmas szerepet kapnak a relációs adatbázisok. A ZF az adatbáziskezelésre egy komplex komponenskészletet kínál, több absztrakciós szint megvalósításával. A két legfontosabb absztrakció az adatbázis és a táblák OO implementációja. Az adatbázis absztrakciója a PHP kódot függetleníti a háttérben futó adatbázisszervertől. Ez azt jelenti, hogy ha a Zend-ben adatbáziskezelést programozunk, az tetszőleges adatbázisszerveren ugyanúgy fog futni, és bármikor átállhatunk egyikről a másikra, a kódhoz nem kell hozzányúlnunk. A tábla absztrakció az adatbázistáblákat és azok rekordjait PHP objektumokként értelmezi. Ezek segítségével gyakorlatilag úgy tudunk dolgozni a keretrendszerrel, hogy a háttérben futó adatbázisról nem is tudunk semmit.

A `Zend_Db_Adapter` az alaposztálya annak, hogy egy adatbázis-kapcsolatot építsünk ki a PHP alkalmazás és a megfelelő RDBMS (Relational DataBase Management System) között. Különböző RDBMS-ekhez külön adapter osztály létezik. A kapcsolódó interfész PHP Data Object (PDO) mintára illeszkedik. A Zend által így támogatott RDBMS típusok a következők: IBM DB2 és Informix Dynamic Server (`pdo_ibm`), MySQL (`pdo_mysql`), Microsoft SQL Server (`pdo_dblib`), Oracle (`pdo_oci`), PostgreSQL (`pdo_pgsql`), SQLite (`pdo_sqlite`). A `Zend_Db_Adapter`-nek többféle módon is megadhatóak a kapcsolódáshoz szükséges adatok. Az egyik leginkább használt módszer, ha a konfigurációs állományban adjuk meg, majd a bootstrap osztályban a „factory” segítségével hozzuk létre a kapcsolatot.

```
# application/configs/application.ini
resources.db.adapter             = Pdo_Mysql
resources.db.params.charset      = utf8
resources.db.params.host         = localhost
resources.db.params.username     = username
resources.db.params.password     = my_secret_pass123
resources.db.params.dbname       = my_database
```

```
// application/Bootstrap.php
$config = new Zend_Config_Ini(
    APPLICATION_PATH . '/Configs/application.ini',
    APPLICATION_ENV
);
$db = Zend_Db::factory($config->resources->db);
```

Ha elkészült az adatbázis-kapcsolat, innentől kezdve ezzel nem kell foglalkoznunk, egyszerűen dolgozhatunk a modell osztályokkal az egyes action-ökben. Itt jegyezném meg, hogy a Zend Framework – attól függetlenül, hogy MVC keretrendszer – lehetőséget biztosít arra is, hogy ne dolgozzunk modellekkel, hanem minden adatbázis-művelethez az adapterünk példányát használjuk, azonban ennek egy ilyen eszközrendszerrel a kezünkben egyáltalán nem lenne értelme.

A modellek tehát az `application/models` mappában helyezkednek el, érdemes őket külön szedni attól függően, hogy épp táblához, vagy rekordhoz kapcsolódnak. Egy táblához tartozó modell osztálynak ismernie kell a hozzá tartozó tábla nevét (`$_name`), a tábla elsődleges kulcsát (akár egy, akár több mezőből áll - `$_primary`). Emellett tudnia kell, hogy melyik osztály valósítja meg a táblához tartozó rekordokat (`$_rowClass`), valamint a tábla kapcsolatairól is tárol információkat (`$_dependentTables`, `$_referenceMap`). Egy tipikus modell osztály a következő:

```
class Models_Table_Projection extends Zend_Db_Table_Abstract
{
    protected $_name = 'projection';
    protected $_rowClass = 'Models_Row_Projection';
    protected $_primary = 'projection_ID';
    protected $_dependentTables = array(
        'Models_Table_Reservation'
    );
    protected $_referenceMap = array(
        'room' => array(
            'columns' => array('room_ID'),
            'refTableClass' => 'Models_Talbe_Room',
            'refColumns' => array('room_ID')
        ),
    );
}
```

A kapcsolat számosságától függően tartalmazhat tetszőlegesen `$_dependentTables` és `$_referenceMap` attribútumokat. Ha így megadjuk a kapcsolatokat a modellekben, a kódokban alkalmazhatóak a `findDependentRowset()`, a `findParentRow()` és a `findManyToManyRowset()` metódusok, melyek a kapcsolat alapján visszaadják a megfelelő kapcsolt adathalmazokat. A táblákhoz tartozó modellek mellett szükségünk van még a hozzájuk tartozó sorosztályokra is. A fenti modellhez tartozó rekord implementációja a következő lehet:

```
class Models_Row_Projection
    extends Zend_Db_Table_Row_Abstract
{

// my implementation here

}
```

Egy tipikus példája a modellekkel való munkának:

```
// instantiate table model
$mvMdl = new Models_Table_Movie();

// getting a new empty row
$mvRow = $mvMdl->fetchNew();

// setting up the new row with data
$mvRow->title = $fData['movieTitle'];
$mvRow->time_length = $fData['movieMinute'];
$mvRow->year = $fData['movieYear'];
$mvRow->description = $fData['movieDesc'];

// insert row
$movieId = $mvRow->save();

// serialization
$serializedRow = serialize($mvRow);

// delete row
$mvRow->delete();
```

Mint látható a modellek a `Zend_Db_Table_Abstract` és a `Zend_Db_Table_Row_Abstract` osztályokból származnak, azaz használható az összes ott implementált metódus (lásd API).

Fontos beszélni még a lekérdezésekről. Erre a Zend egy külön osztályt biztosít számunkra, ami a `Zend_Db_Select` nevet kapta. Az SQL-es select utasítást valósítja meg OO paradigma alapján metódusokkal. Az osztály generálja helyettünk magát a lekérdezést (query-t), amely adatbázisfüggetlen lesz (az aktuális adapter alapján dől el a szintaktikája), valamint a lehető legbiztonságosabb (SQL injection támadások elleni védelem). Miután összeállítottuk a lekérdezést, egyszerűen lefuttathatjuk azt, mintha csak sztringként írtuk volna meg.

Nem kötelező használni az osztályt, mint fentebb is láttuk egyszerű lekérdezések esetén az adapterek, vagy a modellek által biztosított fetch metódusok is segíthetnek. Ha azonban feltételekhez vannak kötve a lekérdezés egyes részei (rendezés, csoportosítás, szűrés, stb…), vagy komplexebb select-et kell írnunk, mindenképp érdemes ezt használni.

Kétféle módon hozhatunk létre `Zend_Db_Select` objektumot. Az első módszerben akár az adapter, akár a modellünk `select()` metódusát hívhatjuk, míg a második módszerrel az osztály konstruktorát hívjuk segítségül.

```
// way 1: using factory method
$db = Zend_Db::factory(...options...);
$select = $db->select();

// or model method
$myModel = Models_Table_Projection();
$select = $myModel->select();

// way 2: using constructor with default db adapter
$select = new Zend_Db_Select();
```

Minden SQL kulcsszóhoz találunk egy-egy metódust az osztályban:

```
// Create the Zend_Db_Select object
$select = $db->select();

// Add a FROM clause
$select->from( ...specify table and columns... );

// Add a WHERE clause
$select->where( ...specify search criteria... );

// Add an ORDER BY clause
$select->order( ...specify sorting criteria... );
```

Itt ragadnám meg az alkalmat, hogy a Zend Framework által olyan sok helyen alkalmazott, és nekem személy szerint egyik legjobban tetsző tulajdonságára hívjam fel a figyelmet. Ez az úgynevezett „fluent interface”, ami kényelmesebbé és gyorsabbá teszi a kódolást. A dolog egyszerű, vannak olyan metódusok, melyek magával az objektummal térnek vissza, így tetszőlegesen hívhatóak egymás után, akár egyetlen utasításon belül is. Az előbbi példa ezzel a megoldással a következőképpen néz ki:

```
$select = $db->select()
    ->from( ...specify table and columns... )
    ->where( ...specify search criteria... )
    ->order( ...specify sorting criteria... );
```

Minden utasításrészre hasonlóan metódushívásokkal hivatkozhatunk.[8]

[8] http://zendframework.com/manual/en/zend.db.select.html

# *View, és a megjelenítési komponensek (Zend_View, Zend_Layout)*

A `Zend_View` az MVC alkalmazások View alkotója. Teljesen külön áll a modellektől és a controller szkriptektől. Lényegében a view használata 2 fő lépésben zajlik. Az első lépésben a meghívott action példányosítja a `Zend_View`-t, és változókat ad át neki. Második lépésben az action-ben megadhatjuk, hogy jelenítsünk meg egy különálló viewscript-et, ezáltal mi magunk is kezelhetjük azt.

Tetszőleges controller-ünk egy metódusában átadhatjuk az adott action view-jának a kívánt változókat egy egyszerű utasítással: `$this->view->változónév = 'érték';`

```
// use a model to get the data for book authors and titles.
$data = array(
    array(
        'author' => 'Hernando de Soto',
        'title' => 'The Mystery of Capitalism'
    ),
    array(
        'author' => 'Henry Hazlitt',
        'title' => 'Economics in One Lesson'
    ),
    array(
        'author' => 'Milton Friedman',
        'title' => 'Free to Choose'
    )
);

// now assign the book data to a Zend_View instance
$this->view->books = $data;
```

Ezután pedig a viewscript-ben `$this->változónév` néven hivatkozhatjuk a megkapott változót.

```
<? if ($this->books) : ?>
<table>
    <tr>
        <th>Author</th>
        <th>Title</th>
    </tr>

    <? foreach ($this->books as $key => $val) : ?>
        <tr>
            <td>
                <?=$this->escape($val['author'])?>
            </td>
            <td>
                <? echo $this->escape($val['title']) ?>
            </td>
        </tr>
    <? endforeach; ?>
</table>
<? else : ?>
    <p>There are no books to display.</p>
<? endif; ?>
```

A viewscript-ek az `application/views/scripts` mappában helyezkednek el, mégpedig minden controller-hez tartozik egy almappa, és azon belül minden action-höz tartozik egy phtml viewscript. A phtml kiterjesztés arra utal, hogy egy ilyen viewscript tartalma html kódot tartalmaz kisebb PHP beszúrásokkal. Mint az a példában is látható, egy ilyen phtml-ben, mivel nagyjából 1 soros PHP részletek vannak, és a korábban már említett alternatív PHP szintaktikát használjuk. Ez a szintaktika blokkok használata helyett kettőspontot és az adott kulcsszó end-del kezdődő párját jelenti, mint `endif`, `endfor`, `endforeach`, `endwhile`, stb. Ha így használjuk a PHP-t ezen fájlokban, sokkal átláthatóbb script-eket írhatunk. Szintén érdekes lehet még az előbbi kódrészletben a két érték eltérő kiíratása, nem véletlenül csináltam különböző módon. Az echo-ra a viewscript-ekben használhatunk egy gyorsabb szintaktikát is, amiben a nyitó PHP tag használatát egy egyenlőségjel követi majd a kiírni kívánt érték. A fenti két kiíratás tehát ekvivalens.

### View helper osztályok

Előfordulhat, hogy a viewscript-ekben bizonyos komplexebb metódusokat kell elővennünk újra és újra. Ilyenek lehetnek például egy dátum megformázása, űrlap elemek megjelenítése, linkek létrehozása, stb. A Zend beépítve tartalmaz jónéhány előre megírt view helper-t, de saját magunknak is írhatunk.

A helper formáját tekintve egy egyszerű osztály. Mondjuk, hogy szeretnénk egy fooBar nevű view helpert. Az osztálynév első része a PATH-ban is megadott `Zend_View_Helper`, és alapértelmezetten az `application/views/helpers` mappába kerülnek a fájlok. Az osztálynév vége pedig általunk megadható, és „camelCase" szintaktikával kell írnunk, az osztályunk teljes neve tehát `Zend_View_Helper_FooBar` lesz. Az osztálynak minimum 1 metódust kell tartalmaznia, ennek neve megegyezik az osztálynév végével, azaz esetünkben `fooBar()`.

Egy helper használatához a viewscript-ekben egyszerűen csak `$this->helperName()` hívásra van szükség. A háttérben ekkor a ZF példányosítja a `Zend_View_Helper_HelperName` osztályt, majd meghívja annak `helperName()` metódusát. Az objektum perzisztens marad, és a view-ban a további hívásoknál már nem kell újra példányosítanunk.

A view-nál érdemes beszélnünk a layout-ról is. A layout alkotja egy weblap vázát, az állandó, nem változó részeket adjuk meg itt, illetve természetesen a változó tartalmakat is meg kell jelölnünk. A `Zend_Layout` csomag egy ún. master layout viewscript generálását végzi helyettünk, amiben az egyes változó területeket placeholder-ekkel definiálhatjuk. Ide töltődnek be a viewscript-ek, akár action-ök által meghívva, akár más módon. Az említett master layout helye a legtöbb esetben az `application/views/layouts/layout.phtml` lesz, de természetesen ezt felül is definiálhatjuk. Egy webalkalmazáshoz tetszőleges számú layout létrehozható, így könnyen implementálható akár felhasználó által változtatható felület is.

Egy tipikus layout a következő képpen épül fel:

```
<?=$this->doctype()?>
<html>
<head>
    <?=$this->headTitle()?>
    <?=$this->headLink()?>
    <?=$this->headStyle()?>
    <?=$this->headScript()?>
</head>
<body>
    <div class="upBar">
        <div class="logo">
            <div class="menu">
                <?=$this->action('menu', 'index')?>
            </div>
        </div>
    </div>
    <div class="content">
        <?=$this->layout()->content?>
    </div>
</body>
</html>
```

Amint az látható, a layout-ban szintén használhatók a view helper-ek. Maguk a dinamikus területek is helper-ek segítségével töltődnek be a megfelelő helyre.

A Zend elég sok beépített view helper-t kínál készen számunkra. Néhány fontosabbat mindenképp érdemes megnézni részletesebben:

Hivatkozások dinamikus létrehozása: Az oldalon rengetegszer lesz szükségünk hivatkozásokra, ugyanis ez az egyes lapok közötti navigáció leggyakoribb eszköze. A hivatkozás az `<a href="..." ... további paraméterek>Katt</a>` HTML kódrészletet jelenti. Mivel ZF-es környezetben ez a „csinos" URL-ek alkalmazása és az esetleges egyéni routing algoritmus használata miatt speciális, célszerű a Zend által kínált Url nevű view helper segítségével generálni a linkeket. A `$this->url($optionsArray, $routerObj, $reset)` hívással érhetjük ezt el. Első paraméterként egy asszociatív tömbben átadjuk, hogy mit szeretnénk megjeleníteni (melyik controller melyik action-ét milyen paraméterekkel), megadhatunk egy alternatív router objektumot, valamint a harmadik paraméter arra szolgál, hogy a nem felsorolt paramétereket törli az url-ből (azaz nem marad „nem kívánatos"

paraméter), de néha pont azt szeretnénk, hogy megmaradjon. Egy tipikus hívás a view script-ből tehát a következőképpen néz ki: `<?=$this->url(array(controller=>'forum', action=>'viewtopic', topicId=>6), null, true);?>`

Felhasználói űrlapok, és azok elemeinek generálása: Mivel a form-ok kifejezetten a felhasználókkal való interakció eszközei, értelemszerűen a hagyományos OO módszer mellett létrehozhatjuk azokat view helper-ek segítségével is. Minden egyes felületelemhez van külön osztály a Zend-ben. Általánosan elmondható, hogy első paraméterként a nevüket várják, második paraméterként az értéküket (ami pld. gombok esetén a felirat, szöveges mező esetén az alapértelmezett tartalom, stb…), majd tömb formájában az egyes HTML attribútumokat. Így például egy egyszerű szöveges input mező létrehozása így történik: `<?=this->formText('username', 'Ide írja a felhasználói nevét', array(size => 20);?>` A felhasználói űrlapok létrehozására sokféle megoldás van, ezek közül csak az egyik a helper-ek használata. A többi módszerről is szó lesz a megfelelő alfejezetben.

BaseURL lekérdezése: Gyakran van szükségünk arra, hogy az `index.php` állomány helyét megtudjuk. Ha domainünk nem közvetlenül erre a mappára mutat, problémánk adódhat abszolút hivatkozások esetén. Csoportmunka során pedig szinte biztos, hogy minden fejlesztőnek más és más lesz a beállított docRoot mappája. Arról már volt szó, hogy az egyes JavaScript-eket és képeket hol tároljuk: helyük a `public` mappa megfelelő alkönyvtára. A példa kedvéért tegyük fel, hogy a képekre így szeretnénk hivatkozni: `/img/kepneve.png`. Az első perjel azt jelenti, hogy a gyökértől nézzük abszolút hivatkozásban az elérési utat, azaz ha localhostunk, vagy a megfelelő domain-ünk nem a public mappára van beállítva, már biztosan nem is fogjuk elérni ezeket az állományokat. A relatív hivatkozások jelenthetnének megoldást erre, de azokkal is probléma akad, ugyanis az URL-ben leginkább controller/action formában olvashatjuk ki, hogy éppen hol tartózkodunk, és bár így a webböngésző meg fogja találni a fájlokat, még mindig lesz egy problémánk. Minden egyes controller/action helyen más és más képként fogja kezelni ugyanazt a képet a böngészőnk (mert az url alapján szerinte máshol található), így aztán minden helyen újra és újra le kell töltenie azt a szerverről. Minden korszerű böngésző tud már gyorsítótárazni (cache), ne vegyük el tőlük ezt a lehetőséget! Ha tehát a gyökér mondjuk a könyvtárszerkezetben 2 szinttel kívülre mutat a

public mappához képest, a baseUrl helper vissza fogja adni a következő karaktersorozatot: `/szülőmappa1/szülőmappa2/public`. Így minden esetben megfelelő lesz az elérési út, és a böngészőknek is csak egyszer kell letölteniük az adott képet, vagy egyéb állományt.

HTML HEAD elemek beállítása, lekérdezése: Ez is a form-okhoz hasonlóan egy csoportnyi helper-t foglal magában. Egy oldalon a fejrészben, azaz a `<head>` és a `</head>` html tagek között nagyon sok finomságot beállíthatunk. Fontos még megjegyezni, hogy ezek egy része főleg a layout-ban kap szerepet, így a legtöbbet ott célszerű használni. Például az oldal címét beállítani a headTitle 1 paraméteres metódussal tudjuk, míg lekérdezni és kiíratni a `<title>Oldal címe</title>` részt a paraméter nélküli változatával lehet. Előfordulhat még, hogy egy CSS-t vagy JavaScript-et a head-hez szeretnénk csatolni, de nincs rá minden oldalon szükség. Ekkor csak az adott view script-ben a headScript vagy a headStyle hívások segítségével tudjuk ezt megtenni.

A részletezett helper osztályokon kívül számos egyéb is található még beépítetten a ZF-hez. Elmondható, hogy a keretrendszer egyik legbővebb, és leginkább gyorsan fejlődő része. Találhatunk még navigációs segédosztályokat, (`Zend_View_Helper_Navigation_*`), kifejezetten a layout-tal kapcsolatosakat (`Zend_View_Helper_Placeholder_*`), különböző nyelvekre való fordításhoz szükséges megoldást (`Zend_View_Helper_Translate`) és egyéb hasznos ötleteket, melyekről természetesen szintén találhatunk részletes leírást a Zend dokumentációjában a megfelelő manual-ban.[9]

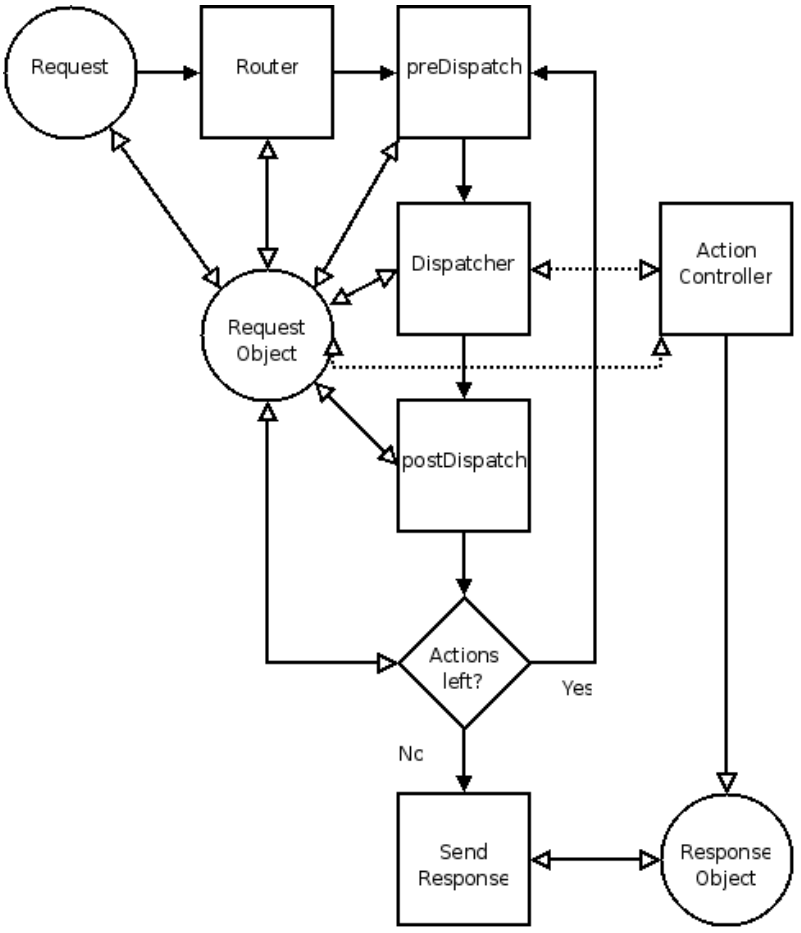
[9] http://framework.zend.com/manual/en/zend.view.helpers.html

# *A vezérlőosztályok (Zend_Controller)*

A controller-ek szintén fontos elemei az MVC alkalmazásoknak. A controller irányítja az alkalmazást, elkapja a felhasználói felületről érkező interakciókat, kéréseket, azok alapján a modellel kommunikálva végzi az adatok lekérését, feldolgozását, majd válasz formájában adatokat ad át a view-nak, ami újra felhasználói közbeavatkozásra vár, és így a kör kezdődik elölről.

A `Zend_Controller` úgy lett megalkotva, hogy a lehető legjobban lehessen azt bővíteni. Kihasználhatjuk az objektum orientáltság minden előnyét, leszármaztathatjuk az osztályokat, írhatunk interfészeket, absztrakt osztályokat, beépülő plugin-okat, és még sok más hasznos eszközt, hogy a rendszer funkcionalitását manipuláljuk, hatékonyságát növeljük.

A következő ábra részletes áttekintése esetén teljesen tisztázódhat számunkra a vezérlés felépítése, annak lépései. Az ábra beépül az eddig tanultakba, egyes részek már ismerősek lehetnek.




Mint látható, a `Zend_Controller` workflow-ja számos alkotóelemből épül fel. A már ismertetett Front Controller tervezési minta alapján a `Zend_Controller_Front` átveszi az ábra felső részén látható elemeket.

## A request objektum

A kérés egy objektum formájában vándorol osztályról osztályra. Tartalmazza az aktuális controller és action nevét, és az opcionális paramétereket. Tetszőleges kérési környezetben működik, legyen az HTTP protokoll, a Zend klienses konzol alapú alkalmazása, vagy akár a PHP-GTK (a PHP nyelv kliensoldali változata asztali alkalmazásokhoz). Az aktuális paramétereket lekérhetjük, illetve beállíthatjuk a következő metódusokkal: `getControllerName()`, `setControllerName()`, `getActionName()`, `setActionName()`, `getParams()`, `setParams()`, `getParam()`, `setParam()` értelemszerűen.

### Az alapértelmezett router

A Zend Framework keretrendszerben az alapértelmezett útválasztó osztály a `Zend_Controller_Router_Rewrite` nevet viseli. A rewrite az osztály végén arra utal, hogy a „csinos” URL-eket az apache webszerverek esetében a `mod_rewrite` modul alkalmazásával tudjuk engedélyezni. Mivel most már tisztában vagyunk a baseUrl fogalmával, a routing-ot a legegyszerűbben úgy lehet elmagyarázni, hogy az URL baseUrl utáni részét tagoljuk, és dolgozzuk fel. Így meghatározzuk, hogy pontosan minek kell futnia. Saját router-t is írhatunk, ennek mikéntjéről részletesen olvashatunk a dokumentációban.[10]

### Dispatching

A dispatching a routing utáni lépcső, a széttagolt URL-t feldolgozzuk, és meghívjuk a megfelelő kódrészeket (action-öket). Ha a controller, vagy az action bármelyike nem lenne megadva, akkor az alapértelmezést hívjuk meg. A `Zend_Controller_Dispatcher_Standard` alapértelmezései az ’index’ nevet viselik. Ez azt jelenti, hogy a `http://azendomainem.hu/` beírása esetén az `IndexController indexAction()`-je fog lefutni, de ugyanez igaz akkor is, ha `http://azendomainem.hu/index/` vagy `http://azendomainem.hu/index/index/` az url. Ugyanígy tetszőleges más controller esetén az action nevének elhagyása esetén az adott controller index action-je kerül meghívásra. A Zend lehetőséget biztosít arra is, hogy az alapértelmezéseket felülbíráljuk, erre valók a `setDefaultController()`, `setDefaultAction()` metódusok.

### Az action controller-ek

Az action controller-t a `Zend_Controller_Action` implementálja. Ebből származnak a controller-eink, amiket mi írunk. Az ilyen osztályoknak legalább 1 action-t kell tartalmaznia, ami az osztály egy publikus metódusa lesz. Az action neve a következőképp alakul: `listAction()`, `editAction()`, `deleteAction()`, stb… Látható, hogy a korábban ismertetett Zend konvenciókhoz igazodik az elnevezés. A Zend routing és dispatching módszerei az összes ’Action’-re végződő nevű metódust potenciális action metódusoknak veszik. Egy általános controller a következőképpen néz ki:

[10] http://framework.zend.com/manual/en/zend.controller.router.html

```
class FooController extends Zend_Controller_Action
{
    public function barAction()
    {
        // do something
    }

    public function bazAction()
    {
        // do something
    }
}
```

Természetesen az action metódusokon kívül számos egyéb függvényt is implementálhatunk a vezérlőkbe. Elegendő csak megnézni az ősosztály metódusait, azokat felüldefiniálva számos módon szabhatjuk testre az adott osztályt. Az `init()` metódusban megadott konfigurációk például az adott controller összes action-jére lefutnak. A `preDispatch()` és `postDispatch()` metódusok használatával úgynevezett hook-scripteket (horgokat) írhatunk, melyek a dispatch előtt illetve után futnak le. Tökéletes helye lehet például itt a jogosultságkezelésnek, mert így egészen biztosan nem fog lefutni az adott action, ha már a `preDispatch()`-ben átirányítjuk a felhasználót egy olyan oldalra, ahol az a hibaüzenet várja, hogy „Önnek nincs jogosultsága az oldal megtekintéséhez.” Fontos különbség az `init()` és a `preDispatch()` közt, hogy az `init()` metódust inkább inicializálásra használjuk, mintegy a konstruktor kiterjesztéseként.

### Action helper-ek, plugin-ok

Néhány előre beépített helper és plugin adott a Zend részéről, azonban magunk is írhatunk, és itt egy kis fantáziával elég merész dolgokat valósíthatunk meg. Nincs más teendőnk, mint egy osztályt írni, mely leszármaztatja a `Zend_Controller_Action_Helper_Abstract` osztályt, és megírni az ismertetett metódusokat, hook script-eket.

A legfontosabb alapértelmezésben is a rendelkezésünkre álló action helper talán az ActionStack. Gyakorlatilag action-öket pakolhatunk várakozási sorba a segítségével, és tetszőlegesen átnavigálhatunk egyik action-ről a másikra, hívhatjuk őket beágyazva. Ez az action-ökben a `$this->_helper->actionStack()` hívással tehető meg, melynek

paraméterként át kell adni, hogy pontosan mit szeretnénk meghívni. Első paramétere az action, majd a controller, és esetleges további hívási paraméterek. Az actionStack kiválóan alkalmas modulárisabb oldalak létrehozására, ahol egy felületen szeretnénk több különböző action-t megjeleníteni. Tipikusan az `indexAction()`-ökben találhatunk actionStack helper utasításokat.

# *Felhasználói űrlapok (Zend_Form)*

A `Zend_Form` segítségével az űrlapokat készíthetjük, dekorálhatjuk, validálhatjuk, stb. Mint azt korábban említettük, rendkívül sok lehetőségünk van, ha form-okkal szeretnénk dolgozni, most a legegyszerűbb objektum-orientált lehetőséggel ismerkedhetünk meg. Egy gyakorlati példán szemléltetve fogjuk megismerni a metódusokat.

Első lépésként hozzunk létre egy új form objektumot a Zend_Form példányosításával.

```
$form = new Zend_Form();
```

Specializáljuk a `<form>` elem action és method paraméterét. A method azt adja meg, hogy az adatok átadása a szervernek milyen módon történjen. Ez kétféle lehet, post, vagy get. A post az elterjedtebb és biztonságosabb, a get az url-ben adja át az adatokat. Az action nevű paraméter azt adja meg egy form esetén, hogy melyik PHP szkript dolgozza fel a megkapott adatokat, így nálunk az action paraméter értéke egy MVC-s action lesz értelemszerűen, azonban a két fogalom nem keverendő (véletlen egybeesés csupán az action név mindkét esetben). Esetlegesen egyéb HTML attribútumokat is beállíthatunk a form-nak (vegyük észre, hogy a „fluent interface” itt is használható).

```
$form->setAction('/resource/process')
     ->setMethod('post')
     ->setAttrib('id', 'login');
```

A form jelenlegi alakjában még üres, kezdjünk el elemeket hozzáadni. Az egyes elemek saját osztályt kaptak a ZF-ben. Ezek az elemtípusok a különféle gombok (`button`), mint a form-beli elemeket űrítő gomb (`reset`), a képként megadott gomb (`image`), valamint az űrlapot elküldő gomb (`submit`). A felhasználó egy vagy több lehetséges elemet a jelölőnégyzet (`checkbox`) segítségével választhat, valamint ide tartozik logikailag a rádiógomb (`radio`), és a legördülő lista (`select`). A különböző szöveges adatok bevitelére való a rövid szöveges beviteli mező (`text`), és a hosszabb szövegekhez alkalmazott szövegterület (`textarea`).

Jelszavakat a jelszómező (`password`) segítségével vihetünk be. A rejtett mező (`hidden`) segítségével pedig olyan láthatatlan elemet hozhatunk létre, ami a felületen nem szerepel, de értéke és neve a többi elemhez hasonlóan van.

Kétféle módon lehet a formunknak megadni ezeket az elemeket, vagy példányosítjuk az adott elem osztályát és ezt adjuk át a formnak, vagy csak név alapján a form egy metódusával is megtehetjük, ekkor a Zend példányosítja helyettünk a megfelelő osztályt.

```
// Instantiating an element and passing to the form object:
$form->addElement(new Zend_Form_Element_Text('username'));

// Passing a form element type to the form object:
$form->addElement('text', 'username');
```

A form innentől kezdve, bár kezdetleges formában, de elkészült. Még meg kell jelenítenünk magát a form-ot valahogyan a view-ban. A renderelés egyszerű, vagy meghívjuk a form objektum `render()` metódusát – ebben az esetben akár más, nem ide tartozó view-hoz is rendelhetjük az űrlapot – vagy egyszerűen csak echo-zzuk ki, adjuk át a view-nak, majd ott írassuk ki, mint változót.

```
// Explicit call render, passing an optional view object:
echo $form->render($view);

// Assuming a view object has been set via setView():
echo $form;
```

A következő fejezetben tovább fejlesztjük a formot validációval és szűrők alkalmazásával.

# *A begyűjtött adatok ellenőrzése (Zend_Validate, Zend_Filter)*

A biztonság alapvető szerepet kell, hogy kapjon egy webes alkalmazás esetében is. Van egy szabály, „filter input – escape output”. A program felhasználóiban fejlesztői szempontból nem szabad megbíznunk, minden egyes adatot, melyet ők írnak be szűrni, alakítani és ellenőrizni (validálni) kell, mielőtt az adatbázisba kerülnének. A kimenetnél pedig mindig figyelnünk kell az adatbázisba esetlegesen mégis hibásan került adatokra, ezért az escaping elhagyhatatlan (ekkor az esetleges HTML, vagy tetszőleges kódok, amik akár ártalmasak is lehetnek, nem fognak lefutni, csak a forrásuk íródik ki ehelyett).

A `Zend_Validate` csomagban egy sor előre definiált és gyakran használt validátor található. Egy validátor általánosan a következőt jelenti: a bemenetként kapott tetszőleges karaktersorozaton bizonyos ellenőrzéseket végez el, majd attól függően, hogy az ellenőrzéseken a sztring átment-e, igaz vagy hamis értékkel tér vissza. Természetesen, ha több validátort is igényel egy bevitt érték, akkor visszatérhetünk a nem teljesülő feltételek listájával, hogy ne kelljen nagyméretű formok esetén minden egyes hiba alkalmával újra és újra kitölteni az adatokat.

Például egy webalkalmazás megkövetelheti, hogy a felhasználói név 6-12 karakter hosszú legyen, csak ékezet nélküli betűket és számokat tartalmazhat. A validátor ezesetben eldönti, hogy megfelel-e a felhasználói név ezeknek a követelményeknek, ha pedig nem, akkor visszaadja, hogy melyik követelmény(ek) nem teljesült(ek).

Ha saját validátort szeretnénk írni, olyan osztályt kell leprogramoznunk, mely implementálja a `Zend_Validate_Interface` interfészt, és meg kell valósítanunk annak az `isValid()` és a `getMessages()` metódusát. Az `isValid()` TRUE értékkel tér vissza, ha a validálás sikeres, egyébként a `getMessages()` visszaadja a hibaüzeneteket. A beépített validátorok természetesen ugyanilyen módon lettek megírva. Példaként a `library` mappánk

`Zend/Validate/` alkönyvtáraiban található osztályokat érdemes megnéznünk. Ezek használata igen egyszerűen történik:

```
$validator = new Zend_Validate_EmailAddress();

if ($validator->isValid($email)) {
    // email appears to be valid
} else {
    // email is invalid; print the reasons
    foreach ($validator->getMessages()
             as $messageId => $message) {
        echo "Validation failure '$messageId': $message\n";
    }
}
```

A beépített validátor osztályok a következők:

- Alnum: akkor ad vissza igazat, ha a vizsgált érték csak betűket és számokat tartalmaz (angol abc), paraméterrel megadható, hogy emellett még szóköz karaktert is elfogadja.
- Alpha: csak a betűket fogadja el (szintén angol abc)
- Barcode: akkor tér vissza igaz értékkel, ha a megadott érték reprezentálható valamelyik vonalkód formátumban (code25, code25interleaved, code39, code39ext, code93, code93ext, ean szabványok, gtin szabványok, issn, stb…)
- Between: azt figyeli, hogy a megadott intervallumba esik-e az érték, vagy sem
- CreditCard: a különböző bankkártyatípusok egyes adatait tudjuk ellenőrizni ezzel a validátorral
- Date: a kapott értékről megállapítja, hogy dátum-e
- Db_RecordExists, Db_NoRecordExists: a kapott értéknek szerepelnie kell, vagy épp tilos szerepelnie egy előre megadott adatbázistábla egy oszlopában
- Digits: csak számjegyeket fogadunk el a stringben
- EmailAddress: ellenőrzi, hogy a kapott érték lehetséges e-mail cím-e. Nem csak szintaktikailag ellenőriz, szolgáltatót is kereshet akár.
- Float: lebegőpontos érték ellenőrzésére szolgál
- GreaterThan: akkor igaz, ha egy alsó határtól nagyobb a kapott érték

- Hex: csak hexadecimális számjegyeket fogad el
- Hostname: szerverneveket ellenőrizhetünk vele (szintén nem csak szintaktikát ellenőriz)
- Iban: IBAN bankszámlaszám formátum ellenőrző
- Identical: azonosságot vizsgál
- InArray: benne van-e az érték egy megadott tömbben
- Int: a kapott értéknek egész számnak kell lennie
- Ip: a kapott érték valós IP cím kell, hogy legyen (akár IPv4, akár IPv6)
- Isbn: ISBN validátor
- LessThan: akkor igaz, ha egy felső határtól kisebb a kapott érték
- NotEmpty: a kapott érték nem lehet üres
- PostCode: irányítószámot vizsgál, megadható a terület
- RegEx: megadott reguláris kifejezésre validál
- StringLength: a sztring hosszát ellenőrzi

A szűrők általában arra valók, hogy nemkívánatos részeket távolítsanak el a beérkező adatokból. Sok más megfogalmazásban a szűrő inkább egy operátor, amely a bemenetnek egy részhalmazával tér vissza. Webes alkalmazásoknál kiválóan alkalmazható, a nem engedélyezett adatokat mellőzi, esetleg levágja a felesleges whitespace karaktereket. A Zend esetén tehát a filter mindig valamilyen transzformációt fog végrehajtani a neki átadott adatokon.

A beépített filterek egy része hasonló a validátorokhoz, kivéve, hogy nem ellenőriz, hanem átalakít. Ezek a következők:

- Alnum: a megkapott karaktersorozatból kiszűr mindent, ami nem alfanumerikus karakter (vagy opcionálisan szóköz)
- Alpha: csak a betűket hagyja meg a bemenetből és azzal tér vissza
- BaseName: teljes elérési úttal megadott fájlnév esetén csak a konkrét fájlnévvel tér vissza

- Boolean: a megkapott értéket boolean értékké konvertálja (integer esetén 0 a false, float esetén 0.0 a false, string esetén az üres string a false, a zéró karakter ’0’ szintén false, az üres tömb, a null, a „false” string lesznek még false értékűek)
- Compress: a megkapott stringet, állományt, vagy mappát tömöríti, a decompress pedig kibontja. Támogatja a bz2, tar, zip, rar, gz, lzf típusokat.
- Decrypt: OpenSSL, vagy MCrypt segítségével kódolt szövegeket dekódol
- Digits: csak a számjegyeket hagyja meg a stringben
- Encrypt: OpenSSL, vagy MCrypt segítségével való kódolás
- HtmlEntities: a megkapott szöveget átalakítja a megfelelő html kódolt karakterek formájában
- Int: visszatér az (int) $value értékkel, azaz típuskényszerít
- LocalizedToNormalized: a megfelelő helyi formátumban megadott értéket alakítja az alapértelmezett formára
- NormalizedToLocalized: az előbbi fordítva
- Null: Null értékké alakítja a megkapott értéket, ha megfelel a kritériumoknak
- PregReplace: reguláris kifejezések alapján történő csere
- RealPath: tetszőleges elérési útból abszolút kanonikus alakú hivatkozást készít
- StringToLower: kisbetűssé alakítja a kapott string-et
- StringToUpper: nagybetűssé alakítja a kapott string-et
- StringTrim: trimmelést hajt végre a string elején és végén
- StripNewlines: visszaadja a többsoros string-et újsor karakter nélkül
- StripTags: visszaadja a kapott értéket bármilyen HTML vagy PHP tag-től mentesen, kivéve, ha az adott tag-re külön engedélyt adtunk meg

# *Authorizáció (Zend_Auth, Zend_Acl)*

A legtöbb webes alkalmazás az egyes felhasználók számára mást mutat, nem ugyanazokat az elemeket engedi megtekinteni. Ez alapvető követelmény egy korszerű alkalmazás esetén. Ha például egy webáruházban szeretnénk vásárolni, ahhoz, hogy ezt megtehessük, a rendszernek valahonnan tudnia kell, hogy kik vagyunk. Onnantól kezdve, hogy azonosított minket a rendszer, tudnia kell, hogy mihez van jogosultságunk, és mihez nincs. Ezeket a funkcionalitásokat hívjuk együttesen authorizációnak. Nem összetévesztendő az authentikációval, ugyanis az csak az azonosításra vonatkozik, és így csak egy szelete az authorizációnak.

A `Zend_Auth` valósítja meg ezt az első szeletet, az egyedek (legtöbb esetben a felhasználók, de persze nem minden esetben) azonosítását. A `Zend_Auth` sok más ZF-es osztályhoz hasonlóan Singleton, azaz egyszerre csak egyetlen példánya létezhet. Nem példányosítható, a létező példány csupán lekérdezhető. Ez a `Zend_Auth::getInstance()` statikus metódus segítségével tehető meg. A `Zend_Auth` ún. adapterek segítségével valósítja meg a különféle hitelesítési szolgáltatásokat (LDAP, RDBMS, állomány alapú, stb…). Az egyes adapterek eltérők lehetnek bizonyos mértékig, de nagyjából a lényeg ugyanaz. A működési elve mindegyiknek az, hogy fogadja a hitelesítési adatokat a felhasználótól, értelmezi azokat, eldönti, hogy sikeres-e a hitelesítés, majd valamilyen formában visszatér az eredménnyel.

Mint azt megszokhattuk a Zend-től, magunk is írhatunk ilyen adaptereket, bár előre megírva is kínál számunka megfelelőeket. Minden `Zend_Auth` adapterosztály implementálja a `Zend_Auth_Adapter_Interface` interfészt. Az interfész egyetlen metódust definiál, az `authenticate()` névre hallgatót. Az authenticate végzi a hitelesítést, majd visszatér egy `Zend_Auth_Result` példánnyal, vagy kivételt dob.

Miután azonosítottuk magunkat, el kell döntenünk, mihez van, és mihez nincs jogosultságunk. Érdemes előre eldönteni, hogy felhasználói csoportoknak szeretnénk

jogosultságokat definiálni, vagy az egyes felhasználóknak külön-külön, esetleg teljesen más struktúra alapján osztjuk ki őket.

A `Zend_Acl` úgynevezett hozzáférés vezérlő listák (Access Control List, ACL) segítségével határozza meg a jogosultságokat. Két különböző típusra osztja az objektumokat, megkülönböztet resource-okat, és role-okat. A resource az az objektum, amihez definiáljuk a jogosultságot, a role pedig az az egyed, aki hozzáférést kap, vagy nem kap az egyes resource-okhoz. Fogalmazzuk meg egyszerűen egyetlen mondatban: az egyes szerepek (role) hozzáférést igényelnek az egyes erőforrásokhoz (resource).

Például mikor a luxusszállodák előtt a parkoló segéd elkéri tőlünk az autó kulcsait, akkor ő van a role szerepében, az autó pedig az erőforrás, amihez megadjuk neki a jogosultságot, de ettől még természetesen nem minden embernek adunk hozzáférést. A legalkalmasabb webes alkalmazások esetén, ha erőforrásoknak az egyes action metódusokat választjuk, míg role-oknak az egyes felhasználói csoportokat választjuk.

Resource létrehozása a `Zend_Acl` segítségével nagyon egyszerűen megy. Az osztálynak implementálnia kell a `Zend_Acl_Resource_Interface`-t, melyben a `getResourceId()` metódussal kérhetjük le az erőforrás ACL-beli azonosítóját. Az erőforrás objektumok egymástól örököltethetők. Valamint a `Zend_Acl` segítségével privilégiumokat is definiálhatunk az erőforrásokhoz (mint pld. olvasás, írás, törlés, stb…). Tehát egy erőforráshoz több különböző jogosultsági szint megadható.

A role-ok létrehozása szintén egyszerűen történik. Az implementálandó interfész neve itt értelemszerűen `Zend_Acl_Role_Interface`, az azonosító metódus neve pedig `getRoleId()` lesz. Az öröklődtetésnek itt nagyobb szerepe van. Ha például veszünk egy vendéget az oldalunkon, akkor a hitelesített felhasználóknak a vendég jogosultságai szintén megvannak, mellette persze extra jogosultságokkal, amik a vendégnek nincsenek meg. Elmondható tehát, hogy a hitelesített felhasználó örökli a vendég jogosultságait. Az adminisztrátori jogosultságkör általában minden más szerepkörtől örökli a jogokat.

```
$acl = new Zend_Acl();
$acl->addRole(new Zend_Acl_Role('guest'))
    ->addRole(new Zend_Acl_Role('member'))
    ->addRole(new Zend_Acl_Role('admin'));

$parents = array('guest', 'member', 'admin');

$acl->addRole(new Zend_Acl_Role('someUser'), $parents);
$acl->add(new Zend_Acl_Resource('someResource'));

$acl->deny('guest', 'someResource');
$acl->allow('member', 'someResource');

echo $acl->isAllowed('someUser', 'someResource') ?
    'allowed' : 'denied';
```

A kódrészlet bemutatja a `Zend_Acl` használatát. Nézzük át, mit is csinál ez az algoritmus: Definiálunk 3 db alap role-t (guest, member, admin). Ezután létrehozunk egy someUser nevű role-t, ami a 3 alaptól örökli a jogosultságait. Fontos, hogy a `$parent` tömbben milyen sorrend alapján értékeljük ki az öröklődést. Mivel a someUser-nek nincs definiálva konkrét jogosultság a someResource-hoz, ezért megnézzük a szülő role-okat. A tömbön jobbról balra haladva megtaláljuk az admin szülőt, szintén nincs hozzárendelve jogosultság az erőforráshoz. Haladunk tovább, megtaláljuk a member-t, mint szülőt, és észrevesszük, hogy neki engedélyezve van a hozzáférés a someResource-hoz.

Ha tovább haladnánk, kisebb ellentmondásba keverednénk, mert a guest-nek meg van tiltva a hozzáférés. Ezt a Zend úgy oldja fel, hogy úgymond „rövidzár” módon az első illeszkedő és pontosan eldönthető párig nézi a hozzáférést, és nem halad tovább. Tehát esetünkben a kódrészlet kiírja az „allowed” karaktersorozatot.

# *Konfiguráció, ideiglenes tárolók (Zend_Config, Zend_Registry, Zend_Session)*

A bootstrap-ről már volt szó a tervezési mintáknál. Az ebben a fejezetben ismertetett eszközök leginkább a bootsrap-hez kapcsolódnak, ugyanis a konfigurációk, és az alkalmazás inicializálása itt történik.

A `Zend_Config` alapvetően arra való, hogy szöveges állományokban kulcs-érték párok formájában beállításokat tároljunk, és onnan egyszerűen lekérdezhessük azokat. Hierarchikus szerkezetet támogató formátumnak kell lennie a fájlnak, ezért a Zend jelenleg 2 fájltípust kezel beépítetten, ezek az XML és az INI formátumok. A `Zend_Config_Xml` előnyeit nem igazán kell ecsetelgetni, az XML szabvány eléggé elterjedt és közismert. Azonban az emberi szem számára olvashatóbb – igazából semmilyen más oka nincs annak, hogy a legtöbben az INI formát használják – a másik, `Zend_Config_Ini` komponens.

A konfigurációk „asszociatív tömb” szerűen érhetők el, ami nyilván akár többdimenziós is lehet, attól függően, hogy a lekérdezni kívánt érték mennyire speciális vagy általános. Az egyes dimenziókban a kulcs érték párok a konfigurációs fájlban leírtakkal egyezik meg. Tehát ha le szeretnénk kérdezni a `resources.layout.layoutPath` értékét, akkor az objektum `$config->resources->layout->layoutPath` elemére fogunk hivatkozni. A `Zend_Config` implementálja a `Countable` és az `Iterator` interfészeket, ennek minden előnyével együtt.

Az INI formátumban az egyes „dimenziók” közti elválasztó karakter a pont (.). Ezenkívül a fájl szekciókra osztható `[szekciónév]` címkék segítségével. Az egyes szekciók a beállításokat örökölhetik egymástól, ezt úgy tudjuk megadni, hogy a név után kettősponttal odaírjuk a szülő szekció nevét. Pld: `[szekció1 : szekció2]`.

Egy példa állomány a következőképpen nézhet ki ini formában (az állomány helye a mappaszerkezetben az `application/configs/application.ini` lesz):

```
; Production site configuration data

[production]
webhost                  = www.example.com
database.adapter         = pdo_mysql
database.params.host     = db.example.com
database.params.username = dbuser
database.params.password = secret
database.params.dbname   = dbname

; Staging site configuration data inherits from production
; and overrides values as necessary

[staging : production]
database.params.host     = dev.example.com
database.params.username = devuser
database.params.password = devsecret
```

Amint az látható a példában, a megjegyzések ebben a formátumban pontosvessző karakterrel kezdődnek. A production szekcióban beállítottuk, hogy MySQL adaptert használjunk, a staging szekcióban pedig már ez az alapértelmezés, mert örökli a tulajdonságokat a production-től, ugyanakkor az adatbázis kapcsolódáshoz szükséges adatok már eltérőek lesznek. A `.htaccess` első sorában ezt a konstanst adtuk meg korábban, hogy milyen szekcióval töltődjön be alkalmazásunk. Ez azt jelenti, hogy ha kellően jól konfiguráljuk a rendszert az ini fájlban, akkor tényleg csak egyetlen apró változtatással tudunk váltani a program egyes változatai között (pld. itt production, staging). Az ini fájl a következőképpen tölthető be (leginkább a boostrap-ben van rá szükség):

```
$config = new Zend_Config_Ini(
    '/path/to/config.ini',
    'staging'
);

// prints "dev.example.com"
echo $config->database->params->host;

// prints "dbname"
echo $config->database->params->dbname;
```

Ezt a `$config` objektumot célszerű letárolnunk legalábbis egy oldalletöltődés erejéig, hogy tetszőlegesen a kódban bárhol anélkül, hogy újra példányosítanunk kellene az osztályt, hozzáférjünk a konfigurációkhoz. Erre való a `Zend_Registry` komponens. A `Zend_Registry` két statikus metódusával valósítja meg az ideiglenes tároló szerepét. Ezek a `Zend_Registry::set('nev',$ertek);` és a `Zend_Registry::get('nev');` Esetünkben a második utasítás visszaadja az `$ertek` változó pontos tartalmát.

```
// load the config object into registry
Zend_Registry::set('config', $config);

// anywhere int he code, if needed
$config = Zend_Registry::get('config');
```

Nem szabad azonban a Registry-t összekeverni a munkamenetekkel (session), mert ez két oldalletöltés között nem őrzi meg az értékét. Azaz, a forum/list helyen letárolunk valamit a registry-ben, akkor azt egy másik oldalon biztosan nem fogjuk elérni. Ez is a HTTP protokoll állapotmentességéből adódik. Persze munkamenetkezeléssel megoldható ez is, elég nagy probléma lenne, ha nem tudnánk megvalósítani. Gondoljunk csak bele, webalkalmazásunk minden oldalon elfelejtené, hogy kik vagyunk, és újra be kéne jelentkeznünk, és egyéb érdekes dolgok történnének…

A Zend nem sokat vacakolt a munkamenetkezeléssel, aminél alapvetően szükség van erre (pld. `Zend_Auth`), ott beépítette a komponensbe, nem kell vele külön foglalkoznunk. Magunk is készíthetünk azonban munkamenet névtereket, ahova tetszőleges számban vihetünk fel olyan fontos adatokat, melyeknek meg kellene maradnunk legalább a munkamenet végéig. Egy munkamenet sokféleképp véget érhet: ha lejár az ideje (szerverenként más a TTL értéke a munkameneteknek), ha ürítjük a kliensoldalon cache-ként tárolt párját (pld. a böngészőben nyomunk egy ctrl + shift + delete billentyűkombinációt), vagy egyszerűen kézzel öljük meg a munkamenetet (kijelentkezünk).

A `Zend_Session_Namespace` szintén névtereken keresztül tárolja az adatokat (hasonló hierarchiával, mint a config tette), hogy konzisztens maradjon önmagához. Itt is tetszőleges mélységig kulcsokkal tudunk hivatkozni a speciálisabb elemekre.

Tehát egy munkamenetváltozót a kódban tetszőleges helyen, a következő utasítással tudunk létrehozni:

```
$mySession = new Zend_Session_Namespace('myNameSpace');
```

A konstruktor paramétere elhagyható, ekkor az alapértelmezett névteret hívjuk segítségül. Ezután a következő utasítás segítségével tehetünk adatot a session-be:

```
$mySession->key = $value;
```

Ilyen egyszerű az egész! Ezután nincs más dolgunk, mint a tárolt értéket lekérni valahol a kódban (ha már egyszer létrejött, akkor megmarad az egyes oldaltöltések között is):

```
$mySession = new Zend_Session_Namespace('myNameSpace');
echo $mySession->key;
```

## „Debuggolás” (Zend_Debug)

A `Zend_Debug` osztálynak egyetlen statikus metódusa van, ez a `dump()`. Tetszőleges változó tartalmának kiírására való, legyen az egyszerű változó, bármilyen tömb, vagy objektum, akármilyen összetettségben. Érdemes használni, mert olvasható, tagolt formában mindenféle extra debug keretrendszer használata nélkül működik. Egy tipikus hívása a következőképp történik:

```
Zend_Debug::dump($myVariable);
```

# A ZF krémje – webes szolgáltatások

A web service-ek, azaz webszolgáltatások igen népszerű, és divatos elemei a mai modern webalkalmazásoknak. Számos nagy cég az általa kifejlesztett rendszerhez API-kat biztosít, melyeken keresztül egy-egy szolgáltatást részben, vagy teljes egészében beépíthetünk saját rendszerünkbe. Méltán népszerű és kellően nagy támogatottságú oldalak a Twitter, vagy a Facebook, a Google különböző szolgáltatásairól (YouTube, Search, Calendar, Docs, Sites, Gmail…) már nem is beszélve. A Zend próbál lépést tartani a modern vívmányokkal, és ha erre lehetőség akad, minél több ilyen web service-t támogat beépített osztályai segítségével. Most a speciálisabb komponensekbe nem merülnék bele, azokat bárki megtalálja a ZF kézikönyvben[11, 12]. Nézzünk meg inkább néhány általánosabb jellegű webes szolgáltatást, mint az e-mail küldés, a SOAP jellegű kommunikáció és a JQuery AJAX-os függvénykönyvtár használata. Ezek részletezése nem célom (A JQuery-ről gyakorlatilag külön diplomamunkát lehetne írni, számtalan könyv foglalkozik a népszerű eszközzel), mindegyikből csak egy kevés ízelítőt szeretnék nyújtani.

## E-mail küldés – Zend_Mail

Kezdjük egy egyszerű kódrészlettel, ami egy levelet küld általunk megadott paraméterekkel:

```
$mail = new Zend_Mail();
$mail->setBodyText('This is the text of the mail.');
$mail->setFrom('somebody@example.com', 'Some Sender');
$mail->addTo('somebody_else@example.com', 'Some Recipient');
$mail->setSubject('TestSubject');
$mail->send();
```

A meghívott metódusok elég nyilvánvalók, beállítjuk a levél szövegét, feladóját, címzettjét (címzettjeit), tárgyát, majd elküldjük a levelet. Természetesen a `Zend_Mail` támogatja az SMTP alapú levélküldést is, ami egy fokkal biztonságosabb, mint ez a gyors módszer. HTML alapú leveleket szintén kezelhetünk az osztály segítségével. Küldhetünk csatolmányokat levelünk mellett, és megadhatunk speciális header elemeket az e-mailnek.

[11] http://zendframework.com/manual/en/zend.service.html
[12] http://zendframework.com/manual/en/zend.gdata.html

## Távoli kommunikáció – Zend_Soap

A webszolgáltatások egyik legelterjedtebb kommunikációs platform-ja a SOAP (Simple Object Access Protocol). Gyakorlatilag egy XML alapú üzenetküldésre használt formátum, a továbbítás leggyakrabban HTTP protokollon keresztül zajlik. A SOAP szabványt a W3C (World Wide Web Consortium) tartja karban jelenleg.

A Zend mind szerver-, mind kliens-oldali implementációt támogat. A `Zend_Soap_Server` kétféle módban érhető el, WSDL és nem WSDL módszerrel. Használata elég egyszerű. Írnunk kell egy tetszőleges osztályt, melyet távolról szeretnénk elérni, és ezt a szerver objektumnak át kell adnunk (`setClass()` és `setObject()` metódusokkal). Ezután a távolról elérni kívánt metódusokat hozzáadjuk a szerverhez (`addFunction()`), majd a `handle()` utasítással megadjuk a szervernek, hogy fogadja a kéréseket.

A kliens is egyszerűen implementálható: A megadott WSDL fájl alapján példányosítjuk a `Zend_Soap_Client` osztályt, majd egyszerűen saját metódusaként hívjuk az ismert függvényeket, amik a szerveroldalon hívódnak meg.

Természetesen a SOAP ettől sokkal kifinomultabb, és a Zend Framework-kel is tetszőlegesen konfigurálhatjuk a kommunikációt.

## JQuery – AJAX megoldások felsőfokon!

A `library` mappában találjuk a Zend mellett a ZendX mappát. Ez a mappanév a Zend Framework Extensions névre utal, ugyanis az itt található eszközök nem szerves részei a keretrendszernek, azonban elég sokan foglalkoznak JQuery-vel ahhoz, hogy a Zend ezt szó nélkül hagyhatta volna. Nem csak a JQuery integrálható a ZF-be, egy verzióval korábban bekerült egy másik AJAX-os függvénykönyvtár, a DOJO.

Mi is a JQuery? A JQuery egy gyors és a végletekig tömörített formájú JavaScript-es függvénykönyvtár, mely fejlett HTML kijelölő eszköztárral rendelkezik, egyszerűsíti az eseménykezelést, az animációk integrálását, és az asszinkron kérésekkel való munkánkat,

mindezt a gyors alkalmazásfejlesztés (RAD) filozófiáján alapulva. A JQuery saját weboldalán ezt olvashatjuk: „JQuery is designed to change the way that you write JavaScript", azaz megreformálják a JS-ekkel való munkát. Számtalan előnye, hogy böngészőtől függetlenül minden JQuery elem ugyanúgy fog megjelenni, mindig, mindenhol. A teljes JS, amit be kell töltenünk mindehhez, összesen 24 Kbyte méretű. A JQuery hivatalos honlapján gyönyörű bemutatók, példák, leírások találhatók a használatához.[13] A diplomamunkához csatolt alkalmazásban is sok JQuery-s elemet használok.

A ZF-ben 3 különböző módon férhetünk hozzá jelenleg a JQuery-s eszközökhöz (bár ez a része elég gyorsan változik a keretrendszernek, szóval elképzelhető, hogy a következő verzióban már újabb JQuery implementációk köszönnek majd vissza). Ezek a módszerek a következők:

- Beépített view helper segítségével, ami betölti számunkra a Core és a UI környezeteket is.
- JQuery UI specifikus view helperek használatával
- JQuery UI specifikus form elemek használatával

Jelenleg a Zend a JQuery-nek csak egy nagyon kis hányadát támogatja, de egyszerűen tudunk kezelni ilyen objektumokat saját módszerekkel. Ezen a területen is számos dokumentumot találunk a világhálón, hogy a JQuery és a Zend Framework hogyan tud a legjobban együtt működni. Elég csak kedvenc keresőnkbe a „Zend Framework + JQuery" kifejezésre rákeresni, rengeteg demo, tutorial, leírás vár bennünket a témában.

[13] http://jquery.com/

# Összefoglaló

Annyit mindenképp elmondhatunk a Zend Framework keretrendszerről, hogy robosztus, a legtöbb területet – amire egy webes alkalmazás készítése során szükségünk lehet – érinti. Nem is csak egyszerűen érinti, hanem kész mintákat kínál a használatukra. Azonban szabad kezet is ad nekünk a fejlesztésben, mindenhol magunk is implementálhatunk eszközöket, és használhatjuk azokat az előre adottak helyett.

A keretrendszer kiváló eszköz azoknak, akik ismerik a PHP nyelvet, és az objektumorientált paradigmával tisztában vannak. Egyszerű kezelni, és ha a programozó egyszer ráérez a dolgokra, rájön, hogy hasonló konvenciókat követ minden építőelem. Mindenképp hasznos eszköztárat nyújt csoportos fejlesztéshez. A kódok érthetőbbek, tisztábbak, konzisztensebbek, tagoltabbak, tesztelhetőbbek, stabilabbak lesznek, ha betartjuk a keretrendszer által favorizált tervezési mintákat.

Én eddig is elég sok alkalmazást fejlesztettem, fejlesztek jelenleg is a keretrendszer segítségével. Számos olyan területe van azonban a ZF-nek, amiről nagyon keveset tudok, és egyáltalán nem használtam még. Ez persze nem jelent problémát, minden vállakozó szellemű fejlesztőnek bátran ajánlom kipróbálásra a keretrendszert, mert bármikor adódhat olyan helyzet, hogy valami újjal kelljen megismerkednünk a Zend világában.

# Felhasznált irodalom

Rob Allen, Nick Lo, Steven Brown: **Zend Framework in Action**

*(Manning Publications Co. 2009 – Greenwich)*

Keith Pope - **Zend Framework 1.8 Web Application Development**

*(Packt Publishing Ltd. 2009 – Birmingham, UK)*

W. Jason Gilmore – **Easy PHP Websites with the Zend Framework**

*(V.J. Gilmore, LLC 2009 – Ohio, USA)*

Cal Evans – **php|architect's Guide to Programming with Zend Framework**

*(Marco Tabini & Associates, Inc. 2008 – Toronto, Canada)*

Internetes tartalmak:

http://php.net

http://phpframeworks.com

http://framework.zend.com/manual/en/

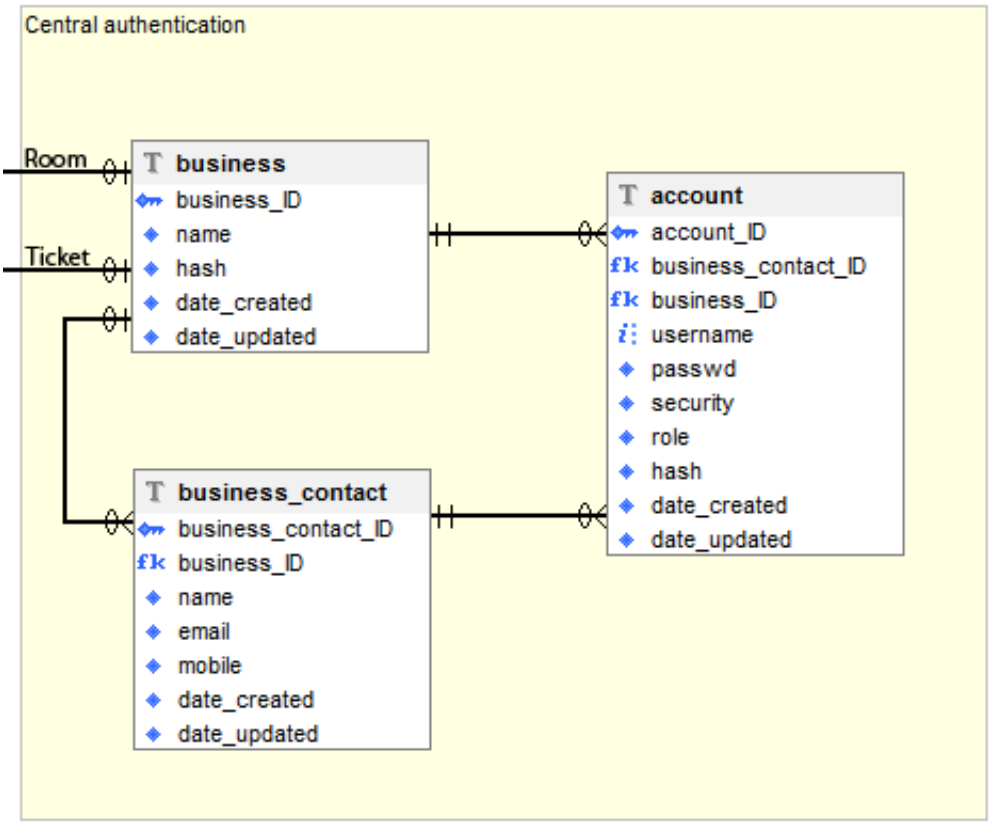
http://en.wikipedia.org/wiki/Design_pattern_(computer_science)

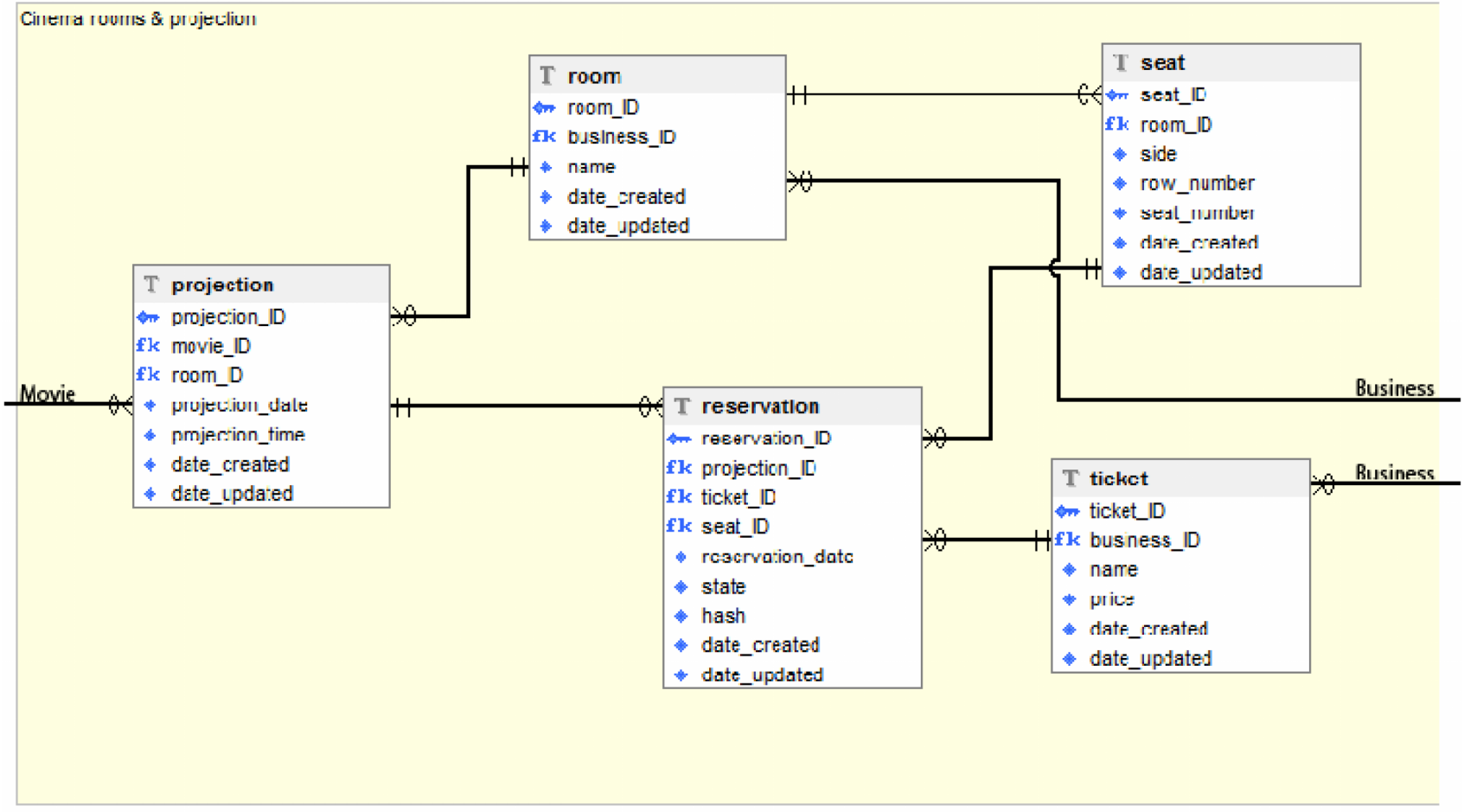
http://hu.wikipedia.org/wiki/Modell-nézet-vezérlő

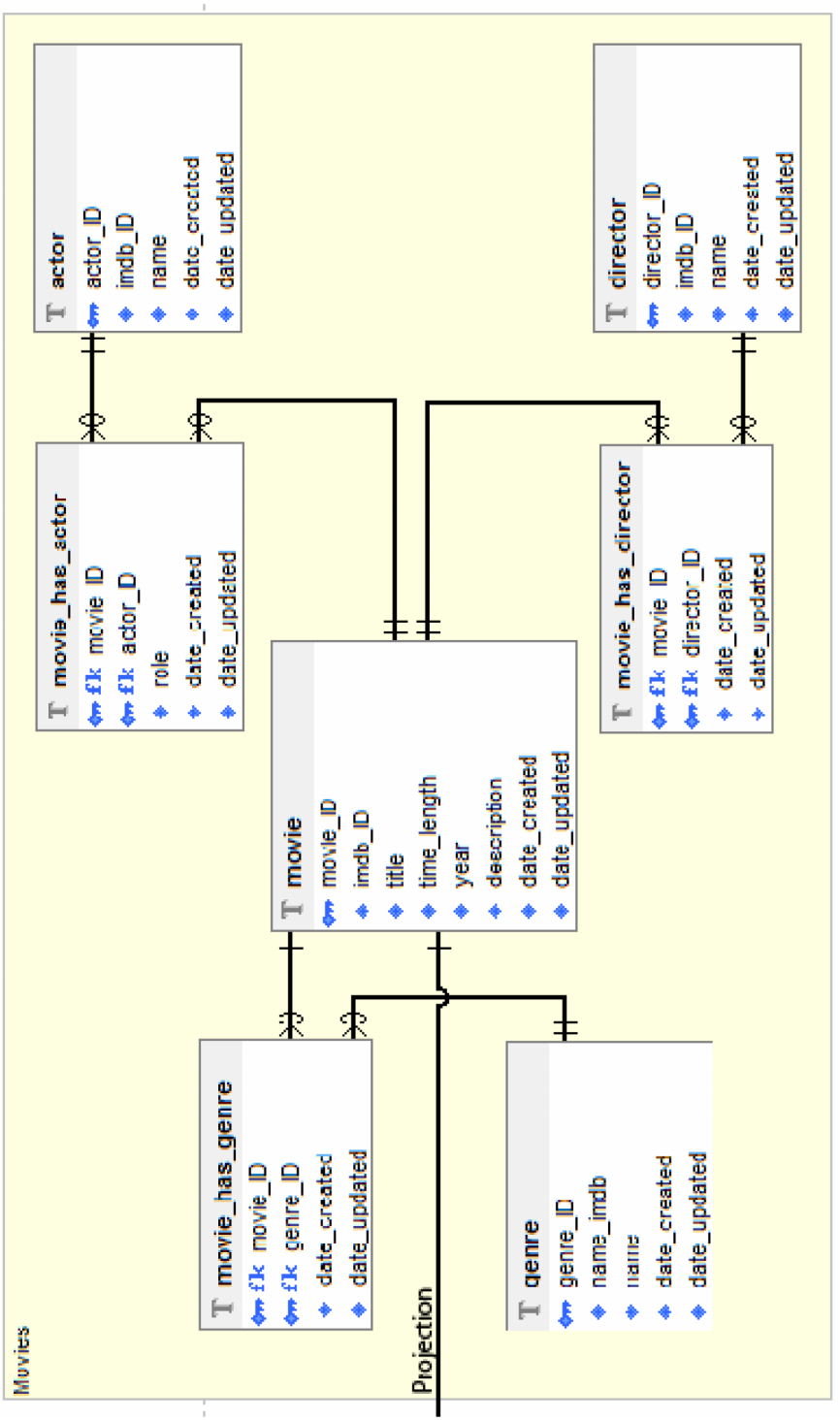
http://jquery.com

http://hu.wikipedia.org/wiki/SOAP

# Függelék– CineSystem adatbázistervek




Cinema rooms & projection
room
room_ID
business_ID
name
date_created
date_updated
seat
seat_ID
room_ID
side
row_number
seat_number
date_created
date_updated
projection
projection_ID
movie_ID
room_ID
projection_date
projection_time
date_created
date_updated
Movie
reservation
reservation_ID
projection_ID
ticket_ID
seat_ID
reservation_date
state
hash
date_created
date_updated
Business
ticket
ticket_ID
business_ID
name
price
date_created
date_updated
Business


Movies
actor
actor_ID
imdb_ID
name
date_created
date_updated
director
director_ID
imdb_ID
name
date_created
date_updated
movie_has_actor
movie_ID
actor_ID
role
date_created
date_updated
movie_has_director
movie_ID
director_ID
date_created
date_updated
movie
movie_ID
imdb_ID
title
time_length
year
description
date_created
date_updated
movie_has_genre
movie_ID
genre_ID
date_created
date_updated
genre
genre_ID
name_imdb
name
date_created
date_updated
Projection

# Függelék – Felhasználói kézikönyv

A mellékletként beadott webalkalmazás a CineSystem nevet kapta, és funkcióját tekintve egy adminisztrációs rendszer mozik számára. Főbb tulajdonságai:

- Filmek kezelése akár IMDb adatok alapján is. A filmekhez felvihetjük a film rendezőit, szereplőit, műfaját, címét, játékidejét. Ezeket kézzel is felvehetjük, illetve az IMDb adatbázisából is könnyen be tudjuk tölteni az adatokat. A rendszerbe már egyszer felvitt adatokat a szoftver „megtanulja”, és legközelebb már felajánlja az adott nevet.
- Jegyek kezelése, egyszerű módszerrel. A mozijegyeknek van neve (pld. Diákjegy) és ára. A jegyek szerkeszthetők, törölhetők.
- Termek kezelése, tetszőleges számú mozi termet felvihetünk a rendszerbe, megadhatjuk, hogy a terem hogyan néz ki (vizuális ülőhely-vázlat alapján).
- Vetítéseket vehetünk fel, megadható, hogy melyik filmet melyik teremben mikor adják. A rendszer intelligensen szűri a vetítéseket, hogy a felhasználó lássa, mit vihet még fel melyik terembe.
- Helyfoglalást intézhetünk, egy grafikus felületen foglalhatjuk le az ülőhelyeket a nézők számára. Ez párhuzamos hozzáférés esetén (a rendszer több felhasználós) sem okoz problémát, nem lehet kétszer ugyanazt az ülőhelyet eladni egy adott vetítésre.
- A felhasználókat adminisztrálhatjuk, akik meghívásos alapon kerülnek be a rendszerbe, jogosultságokat rendelhetünk hozzájuk, és választhatunk számukra biztonságos beléptetést. Ha egy felhasználónál be van kapcsolva a biztonságos beléptetés, akkor a felhasználói név és jelszó beírása után a saját telefonszámára SMS-ben kap egy egyedi azonosítót, hogy be tudjon lépni a rendszerbe.

Az alkalmazás tervezésénél fontos szerepet játszott az a szempont, hogy a felhasználói felület minél intelligensebb, önállóbb, és ennél fogva a lehető legegyszerűbb legyen. Egy ilyen rendszer esetén elengedhetetlen, hogy az egyes funkciókat minél gyorsabban, és átláthatóbb módon kezelhessék a felhasználók. A rendszer, amit lehet, automatikusan megtanul, és mindenhol magyar nyelven szól hozzánk.

# *Bejelentkezés a rendszerbe*

A rendszer használatához Önnek szüksége lesz egy felhasználói névre és egy jelszóra. A hozzáférést a rendszer adminisztrátora tudja biztosítani az Ön számára. A felhasználói neve a könnyebb megjegyezhetőség, és az egyediség megkönnyítése érdekében az e-mail címével egyezik meg. Jelszavát saját magának állíthatja be, miután meghívást kapott a rendszer használatához.

Bejelentkezése esetén az Ön adminisztrátora döntésétől függően elképzelhető, hogy a felhasználói név és a jelszó megadása után egy kódot is meg kell adnia ahhoz, hogy be tudjon lépni. Ezt a kódot a rendszer automatikusan küldi bejelentkezéskor az Ön mobiltelefonjára SMS formájában. Természetesen amennyiben az adminisztrátor nem él ezzel a lehetőséggel, nem kell SMS-ban kapott kódot megadnia.

Sikeres bejelentkezés után a rendszer menüjében egyszerűen navigálhat. Az egyes menüpontok részletesen is ismertetve vannak a felhasználói kézikönyv további részeiben. A CineSystem-ből való kilépéshez használja a Kijelentkezés menüpontot.

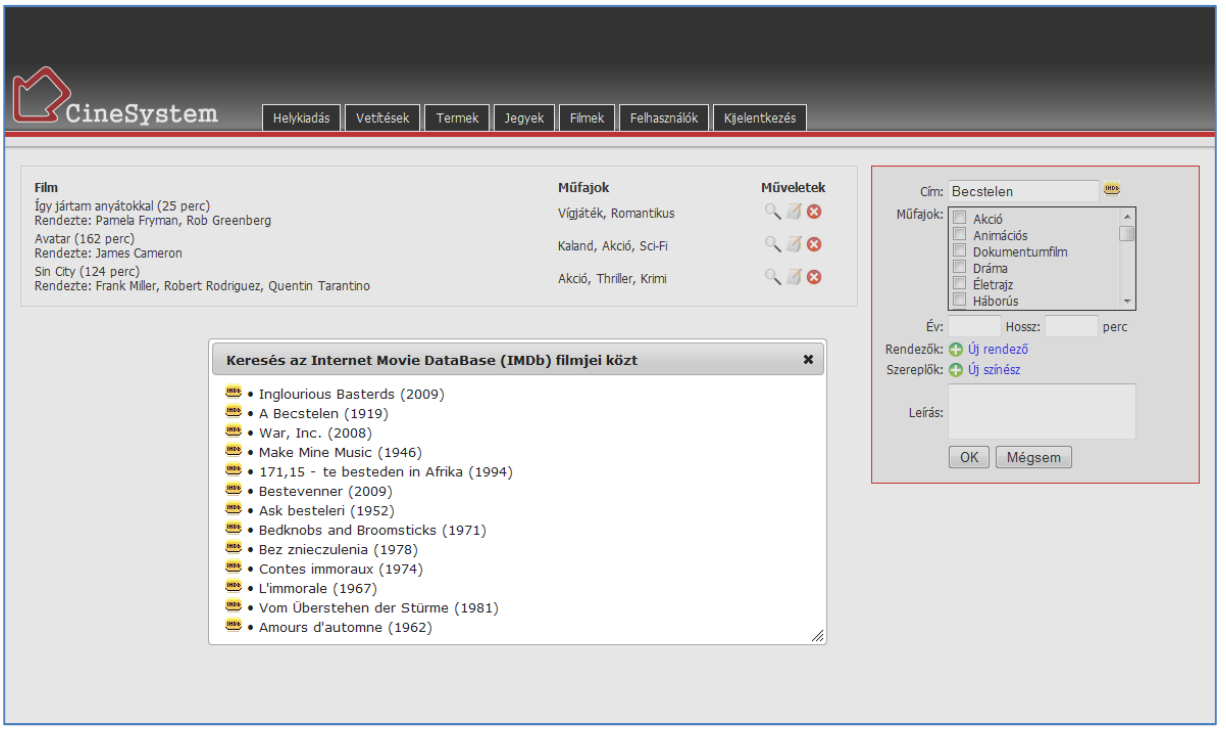
## *Filmek menüpont*

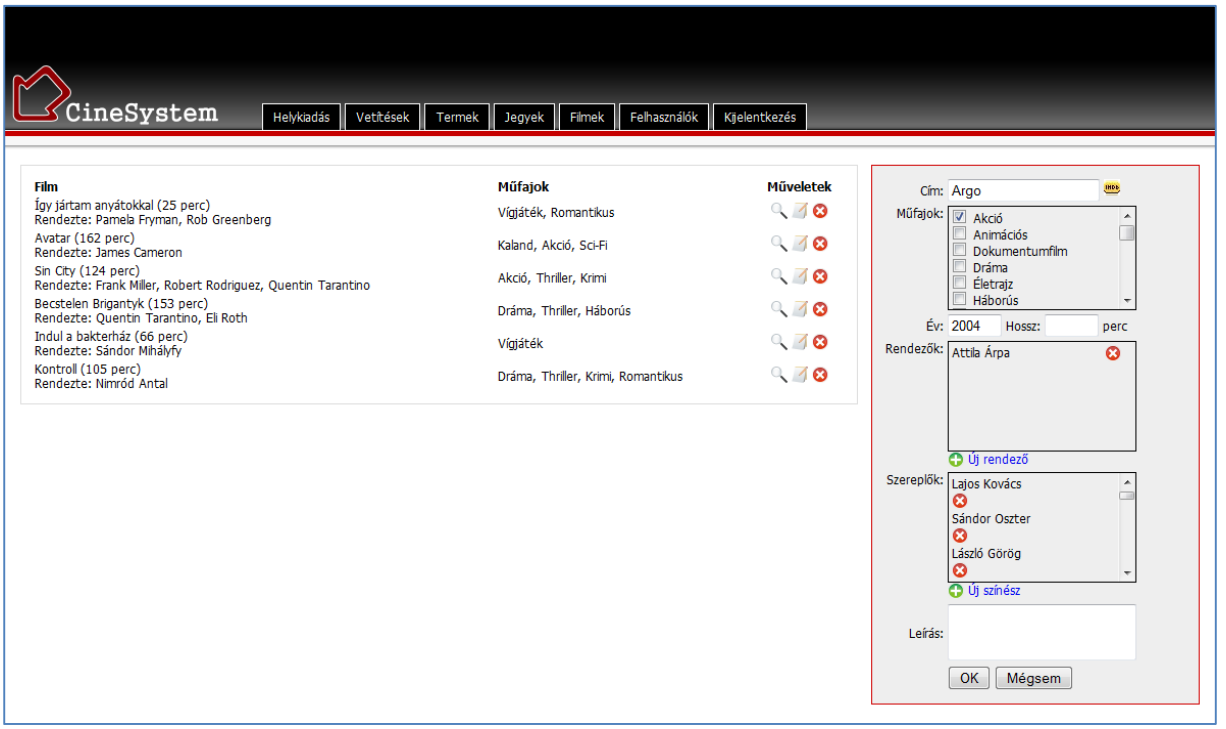
Ebben a menüpontban tud a rendszerbe filmeket feltölteni, és azokat szerkeszteni, törölni. A filmek listáján a következő adatokat láthatja a táblázatban cím szerinti ABC sorrendben: A film címe, hossza percben, a rendező neve (vagy rendezők nevei), és a film műfaji besorolásai. Ha a táblázat végén látható ikonokon kattint, a film részletes adatait hozhatja elő, illetve szerkeszthet vagy törölhet egy már korábban felvitt filmet. A menüponthoz adatfeltöltői, vagy adminisztrátori jogkörrel férhet hozzá.

Filmet két módon tud felvinni a rendszerbe. Az első módszer a kézi hozzáadás, ekkor megadhatja a film címét, kiválaszthatja a hozzá tartozó műfajokat, megadhatja a film hosszát, megjelenési évét, és leírását. Szintén itt tud hozzárendelni a filmhez rendezőket, és szereplőket az új rendező, vagy az új színész linkre való kattintással. Szereplők esetén megadhatja a karakter nevét is, amelyet a színész a filmben játszott. Mindkét típus esetén a zöld pipára kattintva tudja felvinni a személyeket. A rendszer a már felvitt színészeket és rendezőket automatikusan megtanulja, ha ugyanazt az embert szeretné újra felvenni, a nevek első betűinek begépelésével felajánlja Önnek őket.

A másik módszer, ha ugyanezeket az adatokat nem szeretné kézzel megadni. Ekkor a világ legnagyobb filmes adatbázisából, az IMDb (http://www.imdb.com) filmjei közül tallózhatja ki a felvinni kívánt filmet. Csak a cím egy részletét kell begépelnie, majd az IMDb ikonra kattintva a rendszer egy előugró ablakban felajánlja az IMDb-s találatokat. Ezek közül kiválaszthatja az Önnek tetszőt, és egyetlen kattintással betöltheti a film adatait az űrlapra, mintha csak Ön gépelte volna be azokat. A felugró ablakban átnavigálhat a találatok IMDb-s oldalára, ha esetleg nem tudna dönteni.

A filmek szerkesztése esetén ugyanezt az űrlapot használhatja. Filmeket bármikor törölhet a rendszerből, azonban ebben az esetben törlődik a filmhez tartozó összes vetítés is. A nagyító ikonra kattintva egy előugró ablakban megtekintheti a film részletesebb leírását, a filmben játszó színészeket és azok karaktereit, amiket a listában nem láthat.


CineSystem
Helykiadás
Vetítések
Termek
Jegyek
Filmek
Felhasználók
Kijelentkezés
Film
Műfajok
Műveletek
Így jártam anyátokkal (25 perc)
Rendezte: Pamela Fryman, Rob Greenberg
Vígjáték, Romantikus
Avatar (162 perc)
Rendezte: James Cameron
Kaland, Akció, Sci-Fi
Sin City (124 perc)
Rendezte: Frank Miller, Robert Rodriguez, Quentin Tarantino
Akció, Thriller, Krimi
Keresés az Internet Movie DataBase (IMDb) filmjei közt
• Inglourious Basterds (2009)
• A Becstelen (1919)
• War, Inc. (2008)
• Make Mine Music (1946)
• 171,15 - te besteden in Afrika (1994)
• Bestevenner (2009)
• Ask besteleri (1952)
• Bedknobs and Broomsticks (1971)
• Bez znieczulenia (1978)
• Contes immoraux (1974)
• L'immorale (1967)
• Vom Überstehen der Stürme (1981)
• Amours d'automne (1962)
Cím: Becstelen
Műfajok:
Akció
Animációs
Dokumentumfilm
Dráma
Életrajz
Háborús
Év:
Hossz:
perc
Rendezők:
Új rendező
Szereplők:
Új színész
Leírás:
OK
Mégsem


CineSystem
Helykiadás
Vetítések
Termek
Jegyek
Filmek
Felhasználók
Kijelentkezés
Film
Műfajok
Műveletek
Így jártam anyátokkal (25 perc)
Rendezte: Pamela Fryman, Rob Greenberg
Vígjáték, Romantikus
Avatar (162 perc)
Rendezte: James Cameron
Kaland, Akció, Sci-Fi
Sin City (124 perc)
Rendezte: Frank Miller, Robert Rodriguez, Quentin Tarantino
Akció, Thriller, Krimi
Becstelen Brigantyk (153 perc)
Rendezte: Quentin Tarantino, Eli Roth
Dráma, Thriller, Háborús
Indul a bakterház (66 perc)
Rendezte: Sándor Mihályfy
Vígjáték
Kontroll (105 perc)
Rendezte: Nimród Antal
Dráma, Thriller, Krimi, Romantikus
Cím: Argo
Műfajok:
Akció
Animációs
Dokumentumfilm
Dráma
Életrajz
Háborús
Év: 2004
Hossz:
perc
Rendezők:
Attila Árpa
Új rendező
Szereplők:
Lajos Kovács
Sándor Oszter
László Görög
Új színész
Leírás:
OK
Mégsem

## *Jegyek menüpont*

Ebben a menüpontban az eladni kívánt jegytípusokat adminisztrálhatja. A menüponthoz adminisztrátori, vagy adatfeltöltői jogkörrel férhet hozzá.

Minden jegytípusnak van egy neve, ami alapján beazonosíthatja a rendszerben (pld. Diákjegy), valamint egy ára. A táblázatban ezeket az adatokat láthatja, valamint a táblázat végén található ikonokat választva tudja szerkeszteni, vagy törölni a jegyeket.

A jegyek árát egyszerű számként adhatjuk meg, a rendszer azonban formázottan jeleníti meg a listában az ellenértéket. (pld. a beírt 1500-at a listában 1 500 Ft-ként láthatja)

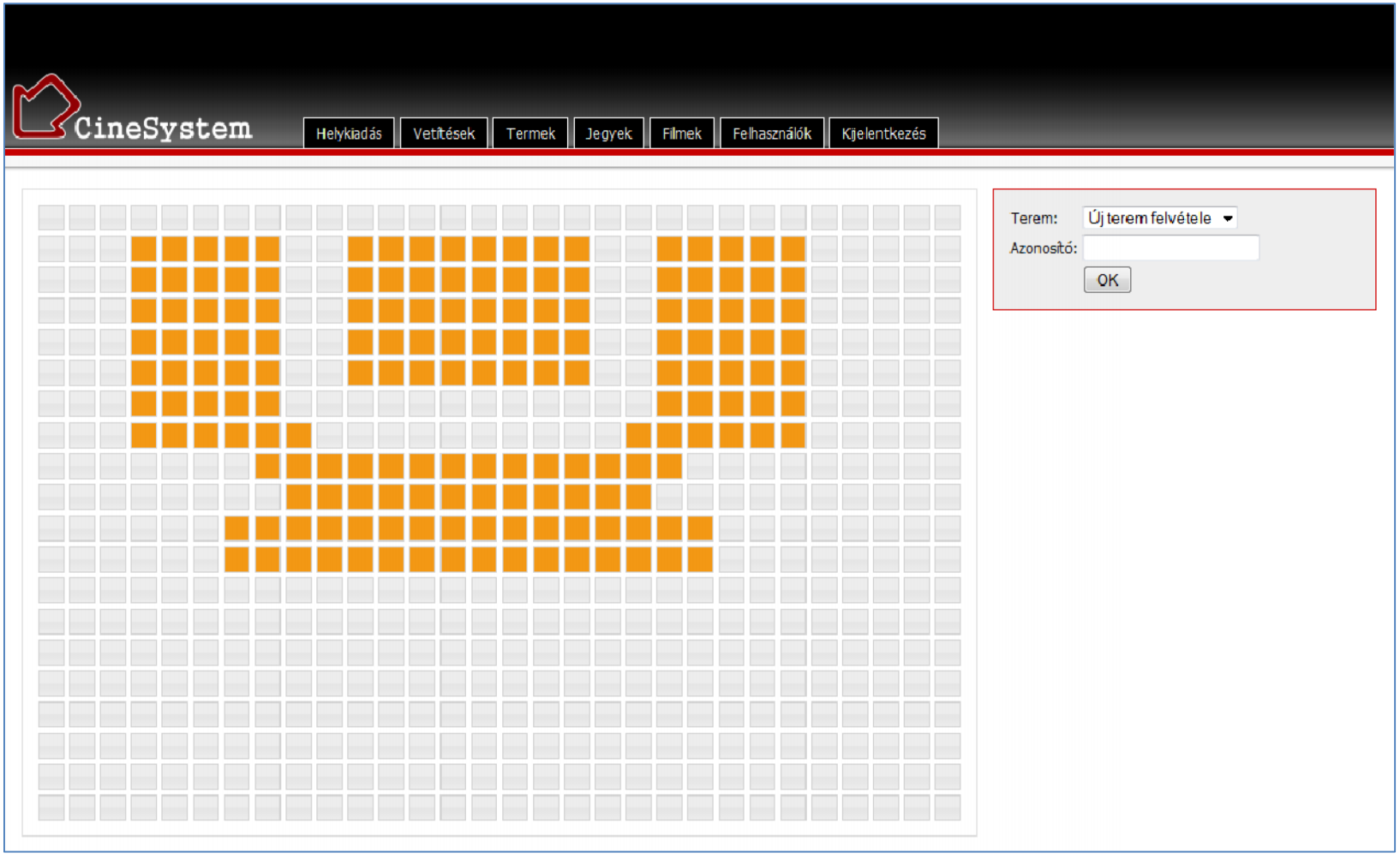
A jegyek szerkesztésére ugyanazt az űrlapot használhatja, mint ahol új jegyet tud felvenni. A jegyeket bármikor törölheti a rendszerből.

## *Termek menüpont*

Itt adminisztrálhatja a mozitermeket. Az oldalon egy virtuális mozitermet láthat, ahol be tudja jelölni, hogy az Ön terme pontosan hogy néz ki. A szürke négyzeteken kattintva tudja megadni, hogy az adott helyen található-e ülőhely az Ön mozijában. Az egérgombot lenyomva tartva több ülőhelyet is kijelölhet, csak húzza el az egeret a négyzetek fölött. A CTRL billentyű lenyomva tartása segítségével az eddigi kijelölés nem vész el, újabb elemeket jelölhet ki, ahogy azt számítógépes asztali alkalmazásaiban is megszokhatta. A menüponthoz adatfeltöltői, vagy adminisztrátori jogkörrel férhet hozzá.

A mozi vizuális megtervezése után adjon nevet a teremnek (Azonosító), majd az OK gombra kattintva mentheti el az adott elrendezést. A legördülő listából kiválaszthatja, hogy egy már meglévő termet szeretne szerkeszteni, vagy új termet szeretne felvinni. Amint kiválasztja a listából az adott lehetőséget, az oldal dinamikusan betölti azt.

Ha a termet törölni szeretné, az űrlap alatt található terem törlése szövegre kattintva teheti ezt meg. Terem törlése esetén a teremhez tartozó ülőhelyek, és a terembeli vetítések is automatikusan törlődnek a rendszerből.

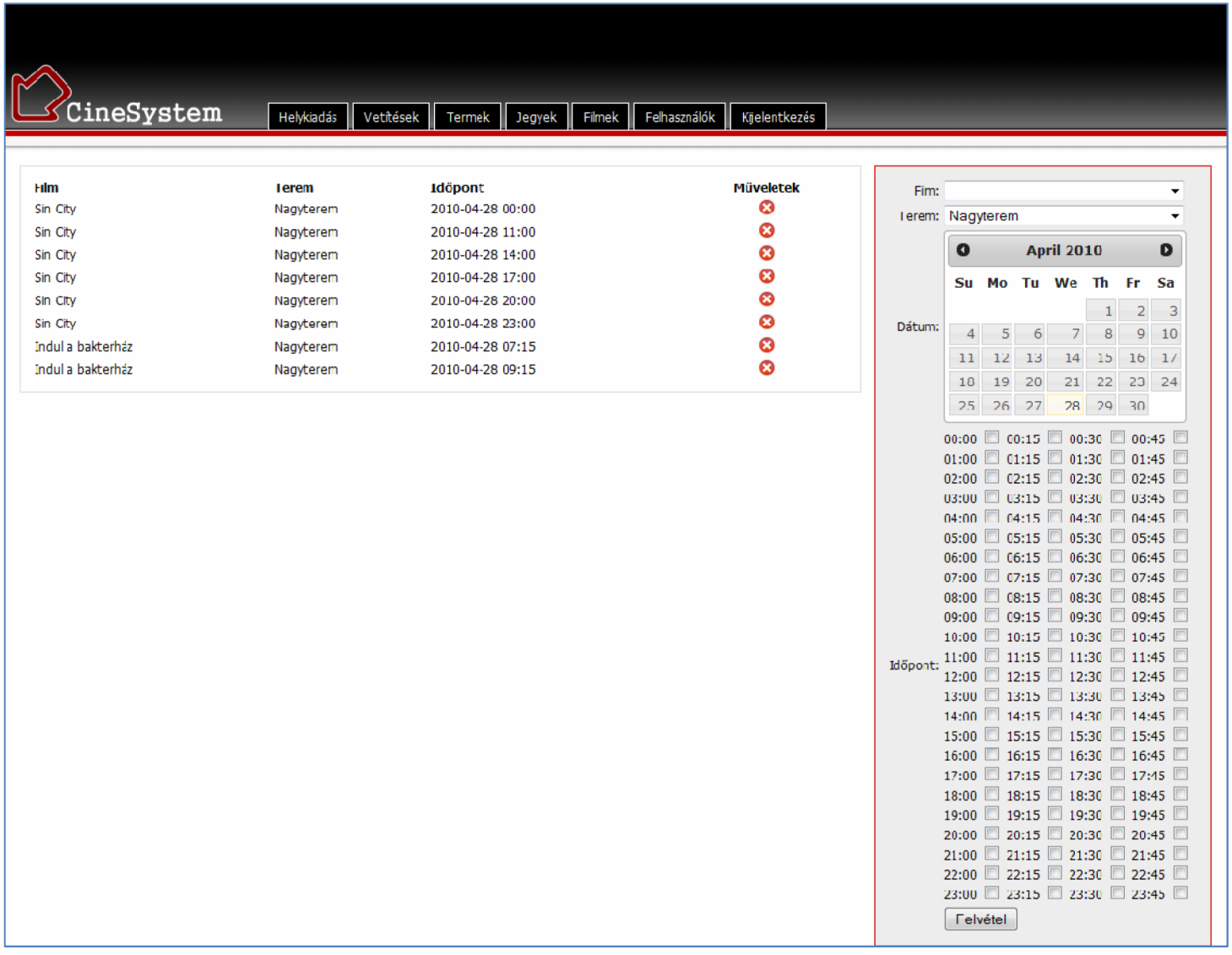

CineSystem
Helykiadás
Vetítések
Termek
Jegyek
Filmek
Felhasználók
Kijelentkezés
Terem:
Új terem felvétele
Azonosító:
OK

## *Vetítések menüpont*

Itt az egyes filmek vetítéseit veheti fel. A menüpont adatfeltöltői, vagy adminisztrátori jogkörrel férhet hozzá.

A baloldalon található táblázat attól függően változik, hogy a jobboldali kezelőfelületen mit választ ki. Alapértelmezésben ki van választva a mai nap a naptárban, ezért a listában az aktuális napi vetítéseket láthatja. Kiválaszthatja a termet, ahol a vetítés lesz (ekkor a táblázat szűkíti a találatokat, csak az adott teremben lévő vetítéseket láthatja). Ezután válasszon filmet (már csak azok az adatok látszanak, amelyek mindkét szűrőfeltételnek eleget tesznek). A naptárból választhat egyéb dátumot (a táblázat ismét változik megfelelően).

Ezek után a táblázat már a legszűkebb szűrés állapotában áll, és vihet fel vetítéseket. Vetítések felviteléhez válassza ki a kiválasztott napra az egyes vetítések kezdő időpontjait. Az űrlap elküldésével (az OK gombra kattintva) a vetítéseket sikeresen elmentette, és a legközelebbi szűrés esetén már látható a táblázatban is. Vetítéseket törölhet is a rendszerből a táblázatban látható törlés ikonra kattintva.


CineSystem
Helykiadás
Vetítések
Termek
Jegyek
Filmek
Felhasználók
Kijelentkezés
Film
Terem
Időpont
Műveletek
Sin City Nagyterem 2010-04-28 00:00
Sin City Nagyterem 2010-04-28 11:00
Sin City Nagyterem 2010-04-28 14:00
Sin City Nagyterem 2010-04-28 17:00
Sin City Nagyterem 2010-04-28 20:00
Sin City Nagyterem 2010-04-28 23:00
Indul a bakterház Nagyterem 2010-04-28 07:15
Indul a bakterház Nagyterem 2010-04-28 09:15
Film:
Terem: Nagyterem
Dátum:
April 2010
Su Mo Tu We Th Fr Sa
Időpont:
Felvétel

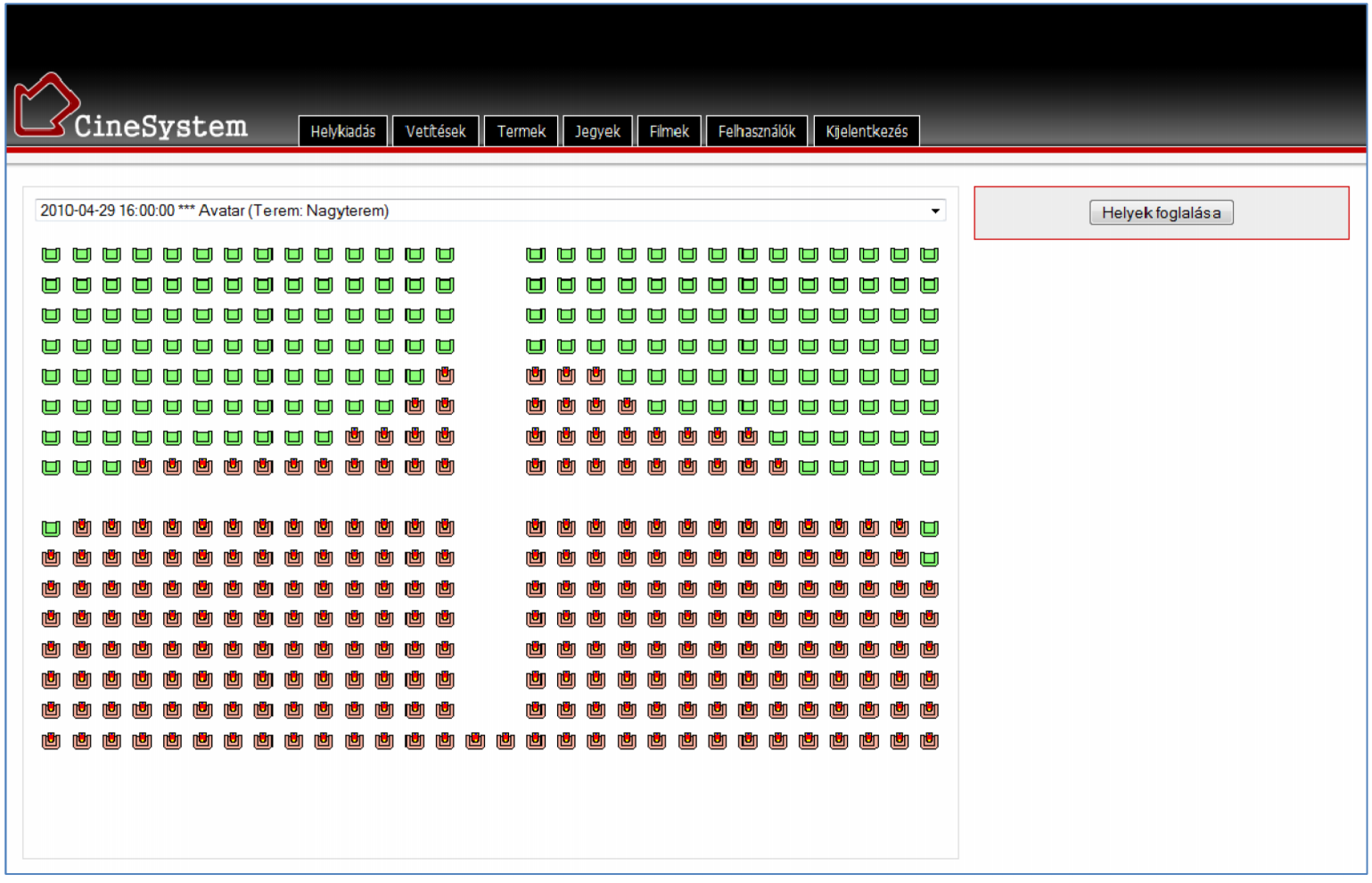
## *Helykiadás menüpont*

Ha Ön értékesíti a jegyeket, ezt a menüpontot fogja használni leggyakrabban. A rendszerbe felvitt vetítésekre adhat el jegyeket. A menüponthoz pénztárosi, vagy adminisztrátori jogkörrel férhet hozzá.

Az oldalon kiválaszthatja, hogy melyik vetítésre szeretné lefoglalni a helyet. Ezt a megjelenő nagyméretű legördülő listával teheti meg. Ebben az egyes vetítések időpontja, a vetített film és a terem jelenik meg. A már elkezdődött vetítések automatikusan kikerülnek a listából. Legfelül láthatja a soronkövetkező előadásokat, minél későbbi időpontban kezdődik a vetítés, annál lejjebb található meg itt.

Ha sikerült megtalálnia a megfelelő vetítést, a rendszer betölti az adott vetítéshez tartozó termet a vizuális felületre, és itt ki kell választania, hogy melyik helyeket foglalja le. A kézikönyvben a termeknél már megismert szerkesztő segítségével tudja a helyeket lefoglalni. Ha a szerkesztőben az ülőhelyen egy zöld fotelt lát, akkor az azt jelenti, hogy az adott hely még szabad. A foglalt helyeket piros fotelben ülő kisember jelzi a felületen. Egyszerre több helyet is lefoglalhat az egérgomb nyomvatartása mellett, ha végigviszi a kurzort a kívánt elemeken. A CTRL billentyűvel pedig az eddigi kijelöléshez vehet fel újabb elemeket anélkül, hogy eltűnne a korábbi kijelölés.

Ezután a jobboldali részen kattintson a „Helyek lefoglalása” feliratú gombra. Ezután kiválaszthatja, hogy az eladni kívánt jegyek milyen típusúak. A „Foglalás véglegesítése” gombra kattintva két eset lehetséges. Amennyiben a lefoglalni kívánt helyeket időközben egy kollégája már lefoglalta, erre a rendszer figyelmezteti Önt, és új helyeket kell kiválasztania. Sikeres foglalás után a program továbblép, és a rendszer kiszámolja Önnek, hogy a jegyekért összesen mennyit kell fizetnie a vásárlónak és ezzel a jegyek eladásra kerülhetnek.


CineSystem
Helykiadás
Vetítések
Termek
Jegyek
Filmek
Felhasználók
Kijelentkezés
2010-04-29 16:00:00 *** Avatar (Terem: Nagyterem)
Helyek foglalása

## *Felhasználók menüpont*

Ebben a menüpontban adminisztrálhatja felhasználóit. A menüpontot csak adminisztrátori jogosultságokkal érheti el. Ha a menübe lép, egy listát talál a rendszerbe felvitt felhasználóiról, valamint a jobboldali mező űrlapját kitöltve tud új felhasználókat meghívni a rendszer használatára. A listában a következő adatokat láthatja a felhasználókról: név, felhasználói név (ami az e-mail címmel egyezik meg), e-mail cím, telefonszám, a regisztráció dátuma, valamint a lista végén található ikonokkal tudja törölni, illetve szerkeszteni az adott felhasználót.

Megadható a felhasználó neve, kiválaszthatja, hogy milyen szerepkörben dolgozzon a felhasználó a rendszerben, meg kell adnia az illető e-mail címét, és rövid szöveget írhat a felhasználónak, hogy miért veszi fel őt a rendszerbe.

A felhasználó e-mail címe lesz egyben a felhasználói neve is, és miután a Meghívás feliratú gombra kattintott, a felhasználó erre az e-mail címre küldött levélben kapja meg a regisztráció megerősítéséhez szükséges linket.

A szerepkör a következők valamelyike lehet: pénztáros, adatfeltöltő, adminisztrátor. A pénztárosnak csak a helykiadáshoz vannak jogosultságai. Az adatfeltöltők a filmek, termek, jegyek, és vetítések menüpontokhoz férnek hozzá. Az adminisztrátor értelemszerűen az összes menüpontot eléri.

Meghívás után a felvenni kívánt felhasználó kap egy e-mailt a postafiókjába, amelyben egy linkre kattintva tudja megadni saját adatait a rendszerben (regisztráció). Amíg a felhasználó nem erősítette meg regisztrációval e-mail címét, a felhasználók listájában megkülönböztetve láthatja őt (dőlt betűvel szedve, szürke színnel).

A felhasználó szerkesztése esetén a felhasználói néven és jelszón kívül bármely adatát módosíthatja az adott embernek. Valamint itt jelölheti be azt is, hogy ha a felhasználó be

szeretne jelentkezni a rendszerbe, a belépési adatok megadása után egy biztonsági SMS üzenet formájában kelljen megerősítenie belépési szándékát. Ebben az esetben a rendszer egy One Time Password azonosítót generál a felhasználónak, azt a megadott mobiltelefonszámra SMS-ként elküldi, és csak a kód beírása után engedi a felhasználót belépni a rendszerbe.

A felhasználókat bármikor törölheti a rendszerből, a listán található törlés ikonra kattintva.