Debreceni Egyetem
Informatika Kar

Közösségi portál fejlesztése PHP és ZEND FRAMEWORK segítségével

Témavezető:
Dr. Kuki Attila
egyetemi adjunktus

Készítette:
Piros Attila Sándor
programtervező informatikus

Debrecen
2009.

Köszönetemet fejezem ki elsősorban Dr. Kuki Attila egyetemi adjunktus úrnak, hogy felügyelte és tanácsaival elősegítette dolgozatom elkészítését, továbbá a Debreceni Egyetem Informatikai Kar tanárainak, hogy az évek során tapasztalt elkötelezett munkájukkal hozzásegítettek az informatikai tudásom megszerzéséhez.

Tartalomjegyzék

1. Bevezetés

A szakdolgozatom a FŐNIX (gyorskorcsolya) egyesület közösségi portáljának kialakítását, működését, kialakításának lépéseit mutatja be PHP és ZEND Framework technológiák segítségével.

Először szeretnék említést tenni a PHP illetve a ZEND Framework technológiákról.

A **PHP** (PHP: Hypertext Preprocessor) nyílt forráskódú, számítógépes szkriptnyelv, legfőbb felhasználási területe a dinamikus weboldalak készítése. Emiatt a PHP-t jórészt szerver-oldalon használják, bár létezik parancssori interfésze is, illetve önálló, grafikus felületű alkalmazások is létrehozhatóak vele.

A nyelvet eredetileg Rasmus Lerdorf alkotta meg 1994-ben, de a ma létező egyetlen (és hivatalos specifikáció híján *de facto* szabvánnyá vált) PHP implementációt már a PHP Group tartja karban és fejleszti. A PHP a saját licensze alatt kerül kiadásra, a Free Software Foundation így szabad szoftverként tartja számon.

A PHP a legtöbb webszerverre, operációs rendszerre és platformra ingyenesen telepíthető. Manapság több mint 20 millió weboldal és egymillió szerver futtat PHP-t, bár a nyelvet használó oldalak száma 2005 augusztusától kezdve folyamatosan csökken. A PHP emellett az Apache webszerver egyik legnépszerűbb beépülő modulja.

A PHP oldalak elkészítésénél a HTML-t gyakorlatilag csak mint formázást használják, ugyanis ezen lapok teljes funkcionalitása a PHP-re épül. Amikor egy PHP-ben megírt oldalt akarunk elérni, a kiszolgáló először feldolgozza a PHP utasításokat, és csak a kész (HTML) kimenetet küldi el a böngészőnek, így a programkód nem is látható kliens oldalról. Ehhez egy ún. interpretert (értelmezőt) használ, amely általában egy külső modulja a webszervernek.

A PHP nyelv lényegében nagymértékű kiegészítése a HTML-nek, ugyanis rengeteg olyan feladat végezhető el vele, amelyre az ügyféloldali szkriptek nem képesek (vagy ha igen, korlátozottan). Ilyen például a bejelentkezés, az adatbáziskezelés, filekezelés, kódolás, adategyeztetés, kapcsolatok létrehozása, e-mail küldése, adatfeldolgozás, dinamikus listakészítés stb. Minden olyan esetben, ahol nagyszámú ismétlődő feladatsort kell végrehajtani (például képek listázása és linkelése, listakészítés stb.), ott ez a programnyelv nagyszerű segítség.

A PHP programok futhatnak közönséges (parancssori) programként is, nem HTML oldalba építve. Ezt azonban ritkán használják. Ezen módot sokszor weblapokkal kapcsolatos időzített folyamatok megvalósításához alkalmazzák, mivel azonos programnyelven, azonos megközelítési módon hajthatók azok végre.

A **Zend Framework** egy nyílt forrású, objektumorientált webes keretrendszer, PHP 5-ben megvalósítva és a New BSD License alatt terjesztve. A Zend Framework – gyakran csak **ZF** – fejlesztésének célja a webfejlesztés megkönnyítése, és közben követendő gyakorlatok bevezetése a PHP közösség köreibe. A ZF igény szerint használható felépítése lehetővé teszi a fejlesztők számára, hogy újrahasznosítsák az összetevőket amikor és ahol az alkalmazásukban értelme van, anélkül, hogy egyéb ZF összetevőket követelnének a legalapvetőbb függőségeken túl. Így aztán nincs olyan fejlesztési minta, amit a használóknak követniük kell, habár a ZF kínál összetevőket a Modell-Nézet-Vezérlő és a Tábla Átjáró programtervezési mintákhoz, melyeket a legtöbb ZF alkalmazás használ. A Zend Framework önálló összetevőt nyújt sok más, webalkalmazás fejlesztésekor gyakori követelményhez, beleértve az azonosítást és a jogosultságkezelést hozzáférésvezérlő listákon keresztül, alkalmazásbeállítást, adatok gyorsítótárazását, a felhasználók által küldött adatok szűrését/ellenőrzését a biztonság és az adatok sértetlensége érdekében, nemzetköziesítést, felületeket AJAX funkciókhoz, emailek összeállításához/küldéséhez, Lucene formátumú keresőjegyzékelést és -lekérdezést, illetve az összes Google Data API-t, sok más népszerű webszolgáltatás mellett. Laza kötődésű tervezésük folytán a ZF összetevőit gyakran használják más PHP-s webes keretrendszerek összetevői mellett.

A Zend Framework gondolata 2005 elején született meg, amikor sok új keretrendszer, mint például a Ruby on Rails és a Spring Framework nészerűvé váltak a webfejlesztői

közösségben. A ZF-et először az első Zend Conference-en jelentették be. Ezidőtájt nem volt széles körben elterjedt, a PHP közösség rendelkezésére álló keretrendszer, ami kielégítette volna a hasonló webfejlesztői igényeket. A Zend Framework tervezői megkísérelték egyesíteni ezen keretrendszerek könnyű használhatóságát és a gyors alkalmazásfejlesztési (rapid application development – RAD) képességeiket az egyszerűséggel, nyíltsággal és a való világbeli praktikummal, melyeket a PHP közösség nagyra értékel.

A specifikus fejlesztési feladatok megvalósítása jellemzően sokkal általánosabb szoftver összetevők felhasználásával, önműködő beállításokon és/vagy kód generáláson keresztül történik. Az eddigi kiadások során a Zend Framework közösség ezen felszín alatt meghúzódó összetevők fejlesztésének teljes elvégzését és tesztelését választotta, a fejlesztési feladatok megkönnyítésén – mint az adatbázis migrációk, állványzatok (scaffolding) generálása, projektek létrehozása és beállítása – való munka megkezdése előtt. Ez a gyakorlat kritikák tárgyát is képezte, mivel sokak által mai webes keretrendszerek megjelenéséhez elengedhetetlennek érzett funkciók a Zend Framework jövőbeli kiadásaira lettek ütemezve. Sok ZF felhasználó azonban sokkal jobban felhasználhatónak és kiterjeszthetőbbnek érezte ezeket az általános jellegű összetevőket alkalmazásaik megvalósításakor. Emellett a Zend Framework keresi a lehetőséget követendő webfejlesztési gyakorlatok a PHP közösségben való alkalmazásának elősegítésére; a ZF-ben közös megállapodások ritkábban használatosak, mint sok más keretrendszerben, ehelyett az ajánlottak ésszerű alapbeállításokban jelentkeznek, melyek felülbírálhatók az egyes ZF alkalmazások követelményeinek megfelelően.

A Zend Frameworkre az Open Source Initiative (OSI) által jóváhagyott New BSD License feltételei vonatkoznak, és minden közreműködőnek alá kell írnia egy az Apache Software Foundation CLA-ján alapuló Közreműködői Licenc Megállapodást (Contributor License Agreement). A licenc és közreműködés szabályai annak érdekében lettek megállapítva, hogy elkerüljék a ZF kereskedelmi felhasználóira irányuló intellektuális tulajdonnal kapcsolatos pereket.

A PHP mag közreműködői, Andi Gutmans és Zeev Suraski által közösen alapított Zend Technologies a Zend Framework vállalati támogatója. A technológiai partnerek közé tartozik az IBM, a Google, a Microsoft és a StrikeIron.

2. Az oldal funkcionalitása

Egy weboldal létrehozásakor el kell dönteni néhány alapvető funkcionalitásbeli dolgot. Mégpedig, hogy milyen céllal, tartalommal, felépítéssel kell rendelkeznie. Egy átlagos weboldal nem túl bonyolult és minden látogató számára átláthatónak kell lennie. Igyekeztem betartani azt a szemléletet, miszerint egy átlagfelhasználó kettő – három mélységnél tovább nem tudja követni a weboldal felépítését és „elvész" benne. Ezért az oldal letisztult és egyszerű használni, kezelni.

Egy ilyen egyesületi közösségi weblap tulajdonképpen nem csak arra szolgál, hogy az egyesületi tagok között fennálljon a kommunikáció, hanem az ismeretlen böngészők számára információt nyújt eme csodálatos sportágról.

Az oldal hat nagyobb modulból épül fel, adja az oldal tartalmát. Ezek sorrendben:

- Kezdőlap
- Csapattagok
- Fórum
- Képgaléria
- Versenynaptár
- Elérhetőségek

A látogatók a kezdőlapon találják a legfrissebb információkat, egy alap bevezetést magáról a sportágról.

A csapattagok menüpont egy rövid ismertetést nyújt a regisztrált felhasználókról (csapattagokról).

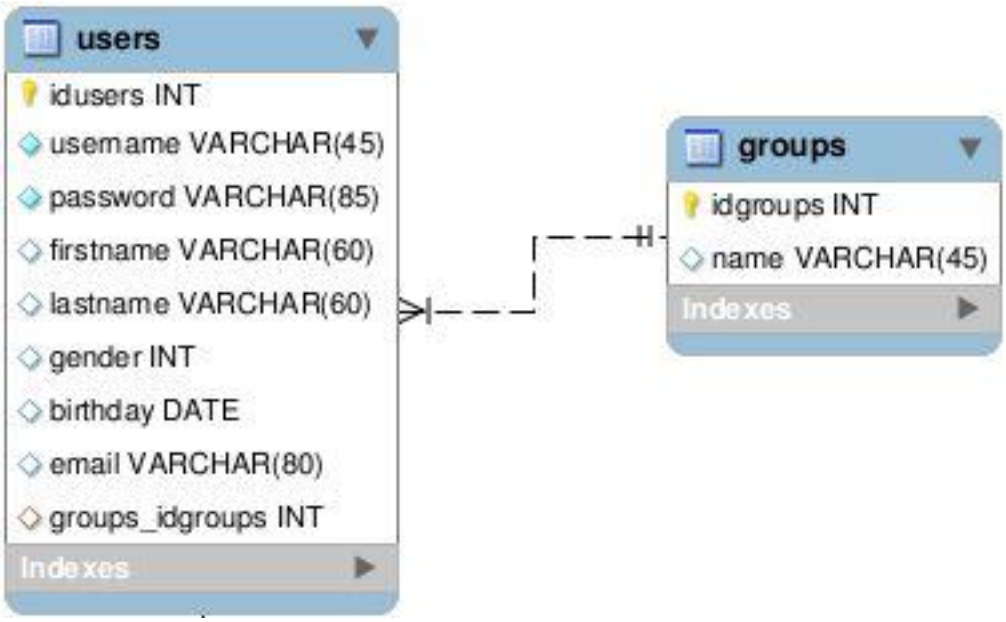
Minden regisztrált felhasználó jogosult a fórum használatára, mely az adott témákhoz való hozzászólást jelent.

Edzéseken, versenyeken készült képek a képgaléria menüpont alatt érhető el.

A versenynaptár az aktuális szezon versenyeiről nyújt információkat.

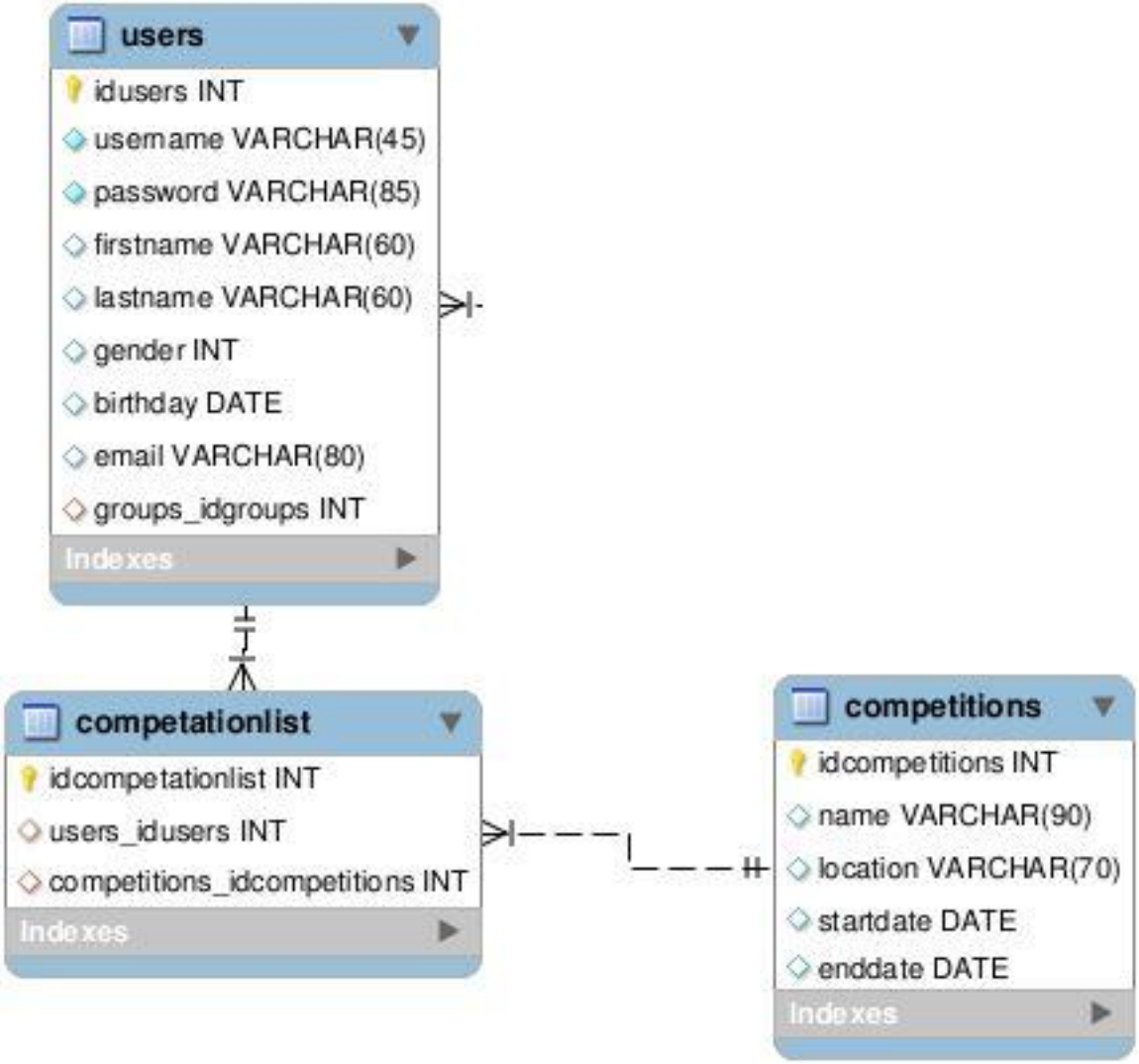
Elérhetőségek alatt az edző, illetve a szakosztályvezető elérhetőségei találhatóak meg.

Mivel ez az oldal egy kisebb közösségi portálnak fogható fel, ezért szükség van bizonyos felhasználói szintek bevezetésére. Ezt négy különböző jogosultságú felhasználói jogosultságú szintre tagoltam. Ezen szintek szoros összefüggésben vannak a regisztrált felhasználókkal, melynek az adatbázis felépítése a következő:



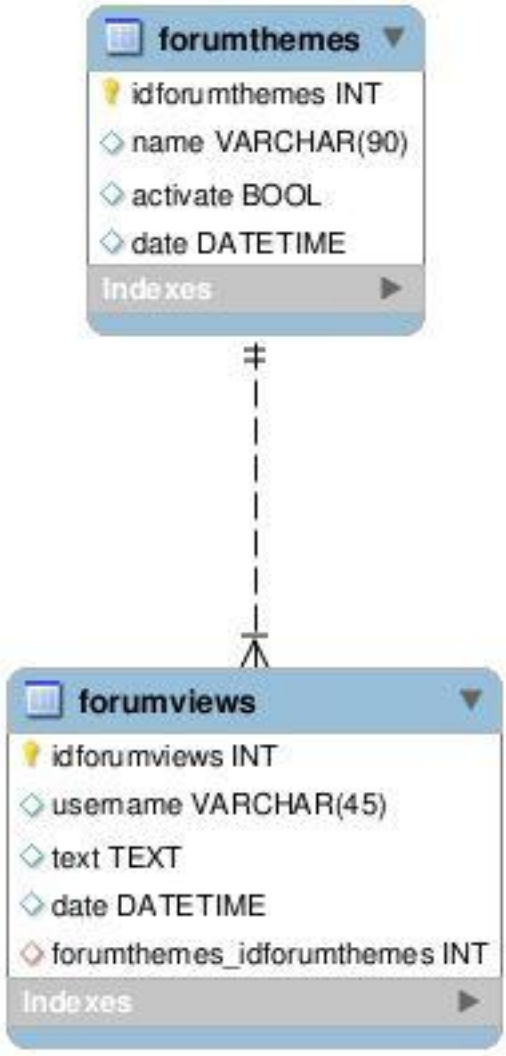
Látható, hogy az egyes csoportok tökéletesen elhatárolják a felhasználókat és ezáltal beazonosításuk is egyszerű.

A regisztrált felhasználóknak továbbá lehetőségük van egyfajta (egy úgynevezett *competition* listára) jelentkezésre, amely információt nyújt arról, hogy az adott szezonban melyik versenyen szeretne, illetve tud elindulni. Ez az előzetes szállásfoglalást segíti a szervezőknek.



A három tábla együttesen jól szemlélteti a köztük lévő kapcsolatokat.

A fórum az alkalmazás talán legösszetettebb része. Az anonymus (nem regisztrált) felhasználók, csak böngészhetik, olvashatják a hozzászólásokat, azonban nincs jogosultságuk bejegyzéseket írni a fórumtémákba. A következő két tábla azt szemlélteti, hogy az egyes fórumtémákhoz hogyan kapcsolódnak a bejegyzések:



A forumthemes táblában megfigyelhető, hogy nem a users táblában lévő usersid került bele mint kűlső kulcs, hanem egy name (név) mező szolgálja a felhasználó azonosítását. Ennek kialakítása azért történt így, mert a későbbiekben, a továbbfejlesztés során elképzelhető, hogy ne csak regisztrált felhasználók, hanem vendég (anonymus) felhasználók is hozzászólhassanak bizonyos témákhoz.

3. Az oldal kialakítása

3.1. A ZEND Framework beállításai

Ahhoz, hogy el tudjuk kezdeni a weboldal fejlesztését, néhány alapvető konfigurációs dolgot be kell állítani. Ezen beállításokat a bootstrap.php valamint a Initilization.php fájlokban kell elvégezni. A bootstrap minden alkalommal elsőként töltődik be, ezért általában az adatbázis kapcsolatot szoktuk definiálni. Az Initializerben található többek között a layout(ok), modul(ok) (modules), kontroller(ek) (controllers), kinézet(ek) (views) beállítási lehetősége. Megadható bármelyikhez az alapértelmezett könyvtár elérési útja, valamint neve. A Framework egy jól strukturált rendszer, melyben ezen apró beállításokat elvégezve hozzákezdhetünk a weblapunk fejlesztéséhez.

```
$db = new Zend_Db_Adapter_Pdo_Mysql(array(
    'host'     => 'localhost',
    'username' => 'root',
    'password' => '          ',
    'dbname'   => 'fonix'
));
```

3.2 Modellek (models)

A modellekben találhatók az adatbázis kapcsolatokhoz szükséges osztályok. Az osztályokban található metódusok segítségével definiálhatóak az adatbázis táblák rekordjai, köztük lévő külső kapcsolatok, lekérdező (select), inzertáló (insert), módosító (update) metódusok.

3.2.1 Users

A modell osztályokat mindig a Zend_Db_Table_Abstract osztály kiterjesztett osztályaként kell létrehozni, nincs ez másképp a Users osztály esetében sem. Protected változóként definiálni kell a tábla nevét (protected $_name), valamint opcionális lehetőség, hogy megadjuk az elsődleges kulcsot (protected $_primary) (természetesen ha nem tartalmaz elsődleges kulcsot, nincs mit megadni). Ezt azért célszerű definiálni, mert a keresés ez alapján egyszerűsödik. Opcionálisan megadható a táblák közötti kapcsolatok,

```
protected $_name = 'users';
protected $_primary = 'idusers';

protected $_referenceMap = array(
    'groups' => array(
      'columns' => 'groups_idgroups',
      'refTableClass' => 'Groups',
      'refColumns' => 'idgroups'
    )
    );
```

melyet a $_referenceMap változó tárol.

A továbbiakban olyan metódusok definiálása történik, melyben a lekérdező, beszúró és módosító sql utasításokat használok ZEND specifikusan.

A *getUserIdByName* egy egyszerű lekérdező metódus, melyre azért volt szükség, hogy egy adott felhasználónévhez tartozó azonosítót (idusers) nyerjek ki. Mivel a bejelentkezéskor felhasználónév és jelszó párossal lehet belépni, azonban a users tábla elsődleges kulcsa az idusers, ezért mindenféleképpen szükség van erre a metódusra.

Belépéskor (loginkor) a *getUser* segít az azonosításban, melynek visszatérési értéke egy boolean érték. True értékkel tér vissza, ha az adatbázisban szerepel a felhasználónév és jelszó páros, valamint false az értéke abban az esetben, ha nem található a felhasználó az adatbázisban.

```
public function getUser($username, $password) {
  $row = $this->fetchRow($this->select()->where("username = ?", $username)->where("password = ?", $password));
  if (!$row) {
    return false;
  }
  else return true;
}
```

getAllUsers: Az adatbázisban lévő összes regisztrált user információit adja vissza egy tömb változóban. Ebben egy ZEND specifikus beépített függvény segítségét kérem, ez pedig a fetchAll. Paraméterként egy select utasítást vár, melynek string formátumúnak kell lenni. A PHP igen kényelmes nyelvnek mondható, hiszen dinamikusan értelmezi a változókat, azonban néhány esetben előfordul, hogy típuskényszerítést kell alkalmazni. Erre a __toString() metódus alkalmas, amely szintén egy ZEND specifikus beépített függvény.

A bejelentkezéskor eldől, hogy ki milyen jogosultságokkal rendelkezik. Ebben a *getGroupsById* függvény segít. Ezen metódus egy összetett select utasítást tartalmaz, melyben join kapocsolat áll fenn a users és a groups táblák között. A lekérdezés után tehát minden felhasználó a megadott csoport tulajdonságaival rendelkezik.

Az *updateProfile* metódusban az eddigiektől eltérően nem select, azaz lekérdező utasítás hajtódik végre, hanem egy update. Hét paraméterrel rendelkezik, melyek közül mindegyik megadása kötelező, ellenkező esetben az update nem fut le, azaz false értékkel tér vissza.

Ebben a lekérdezésben érdekesség még az is, hogy a kűlső kulcs segítségével történik az azonosítás, melyben a *getUserIdByName* metódus segítségül hívva kapok meg.

A regisztrációkor hívódik meg a *useraddAction* függvény. Összesen hat paraméterrel rendelkezik és insert sql utasítást hajt végre. Sikeres lefutása esetén true, sikertelen lefutás esetén false értéket ad, tehát egy boolean típus dönt a lefutás kimeneteléről.

Amennyiben felhasznló törlésére kerül sor, abben a *userdel*-t hívom meg, amely összesen egy idusers azonosítót vár, és amennyiben nincs függő bejegyzései abban az esetben sikeresen törli a kiválasztott felhasználót a táblából.

3.2.2. Groups

A groups modell egy viszonylag egyszerű osztály. Összesen egyetlen egy metódust tartalmaz, amely a regisztrációkor tölt be fontos szerepet. Ez pedig nem más, mint az adatbázisban tártolt összes csoportot listázza a sor tulajdonságaival együtt. Definiálom tábla nevét, amely a groups, valamint a függő (dependent) táblának a nevét is. A groups tábla függ a users táblától, ezért ennek a definiálása itt elkerülhetetlen.

3.2.3. Forum

Ebben a modellben vannak a fórum témáival kapcsolatos metódusok. Természetesen a forum osztály is a Zend_Db_Table_Abstract kiterjesztett osztálya. Védett (protected) változóként bejegyzésre kerül a tábla neve (forumthemes) és az elsődleges kulcs is. Mivel ezen táblához is tartozik referencia táblázat, ezért egyértelműen ezt is definiálom. Ilyenkor meg kell adni a táblázatban lévő rekordjának a nevét, a referencia táblázatban elhelyezkedő modell osztályának a nevét, valamint a referencia táblázat rekordjának nevét.

A *getAllForumthemes* azon fórumtémák neveit adja értékűl, melyek aktív témák, azaz a root vagy a fórum admin felhasználó aktív témának állít be. Egy egyszerű lekérdező script hajtódik végre, melynek feltételében az aktívnak titulált témákra történik a szűrés. Visszatérési értéke függ az sql lekérés kimenetelétől. Sikeres lefutás esetén true, ellenkező esetben pedig false.

A *getThisForumthemes* meghívásakor egy adott jól definiált fórumtéma tulajdonságaival leszek gazdagabb. Ezt akkor tudom felhasználni, mikor egy felhasználó bejegyzést kíván írni valamelyik témához. Sikertelen lefutáskor false, sikeres lefutáskor true értéke van.

Opcionális metódusok közé tartozik a

- updateForumTheme
- deleteForumTheme
- createForumTheme

Ezeket csak a root admin és a fórum admin tudja meghívni.

updateForumTheme: Három paraméteres függvény, melynek első két paramétere az update adat részénél játszik szerepet, majd a harmadik paraméter a feltétel szűrésekor kap szerepet. Amennyiben sikerült az update, boolean típus dönt arról, hogy sikeresen vagy sikertelenül hajtódott végre.

Emennyiben törölni akarok egy témát, abban az esetben a *deleteForumTheme*-t hívom meg, amely egy azonosító alapján kiválasztott témát töröl az adatbázisból.

Amennyiben a törléssel ellentétben új téma bejegyzést szeretnék létrehozni, a *createForumTheme* függvényt alkalmazom. Sikeres lefutáskor két paraméter adódik át a servernek. A harmadik paraméter előállításához egy beépített PHP függvényt alkalmazok, amely az éppen aktuális (szerver) dátumot generálja Év-Hónap-Nap Óra:Perc:Másodperc formátumban.

```
public function deleteForumTheme($id) {
  $where = $this->getAdapter()->quoteInto('idforumthemes = ?', $id);
  $delete = $this->delete($where);
  if (!$delete) {
    return false;
  }
  else return true;
}
```

3.2.4 Forumview

Ebben a modellben is megadom azon értékeket, amelyek szükségesek a többi táblával való kapcsolat kialakításában, valamint a tábla nevét illetve elsődleges kulcsát, azonosítóját.

a *countViewsOneThemes* metódus az eddigiektől eltérően nem egy select utasítás eredményét adja adatként, hanem egy integer típusú számot. Ez azt számolja meg, hogy hány bejegyzés van egy-egy fórumtémában. Egy paramétere van, ez pedig egy fórumtéma azonosító. Hamis értékkel egy esetben térhet vissza, amikor nem létező azonosítóval van paraméterezve.

Hogy ki milyen bejegyzést írt a témákhoz, természetesen ez is egyszerűen egy select utasítás lekérdezése után nyerhető ki. Ebben a *getComments* publikus függvény segít. Érdekessége, hogy a kommenteket a hozzászólások beérkezésének sorrendjében listázza, tehát a legfrisebb bejegyzés kerül majd a lap tetejére.

Ahhoz, hogy tartalom is legyen egy témában, szükség van bejegyzésekre is. Az *insertComment*tel ezen bejegyzések bejegyezhetőek. Három paramétere van, melyek a név, a bejegyzés szövege valamint a téma azonosítója. Az aktuális dátumot ismételten a *date* beépített PHP függvénnyel határozom meg. Amennyiben sikeres az adatfelvitel true, sikertelen esetben false az értéke.

A *deleteViewsByForumTheme* és a *deleteViewsByRoot* metódusok szorosan összefüggnek egymással. Hiszen, amennyiben bejegyzések találhatóak egy fórum témában, akkor természetesen a téma törtlése sql hibaüzenettel térne vissza. Éppen ezért, mindenek előtt törölni kell a bejegyzéseket, majd utána törölhető a fórumtéma. Mind két metódus tehát egy delete utasítást tartalmazó sql scriptet futtat. Paramétereik azonosak, egy azonosítót várnak, a fórumtéma azonosítóját.

3.2.5. Competition

A Competition modell osztály, a versenynaptárnál lesz jelentős. A modell táblaneve a competitions, elsődleges kulcsa az idcompetitions. Két publikus metódusa a listázásnál illetve a felvitelnél lényeges.

getCompetition: Az eddigiekhez megszokott módon, ez a metódus egy lekérdező sql selectet használ, melynek sikeres lefutásakor a visszatérési értéke egy tömb. Ebben a tömbben az összes rekord értéke megjelenik, így könnyen listázható. Amennyiben hibával szál ell úgy false értékkel tér vissza.

A versenyek felviteléhez az *addCompetition* függvényt használom. Négy paramétere van, ezek sorrenben: $name, $location, $startdate, $enddate. A név a verseny nevét, a location a verseny helyszínének helyét adja meg. A startdate és az enddate egy-egy dátum típusú változók, ezek a verseny kezdetét illetve a végét adják meg. A felvitelhez egy insert metódus van segítségre és true vagy false értéke határozza meg a felvitel kimenetelét.

3.3. Kontrollerek

A Zend Frameworkben és mint a többi MVC rendszerben, itt is a rendszer szívét töltik be a kontrollerek. A Zend_Controller egy komplett rendszert jelent, amely a már meglévő subclass osztályokat dolgozzák fel. Azonban létrehozhatók új osztályok is, amelyek különböző interfészek vagy elvont osztályok, melyeknek cselekvő képességét adatkezelő családosztályokban, pluginekben, vagy helperekben lehet menipulálni. A kontrollereknek jutott az a szerep, hogy a PHP fordító által generált kód magját hozzuk létre bennük. Ezek minden esetben a *Zend_Controller_Action* kiterjesztett osztályai. Általában az osztályban szereplő metódusoknak publikus hozzáférést szoktunk adni.
A kontrollerek esetében az első lépés az, hogy ellenőrizni kell a file rendszer felépítését, melynek egy tipikus elrendezése a következő:

```
application/
     controllers/
         IndexController.php
     models/
     views/
         scripts/
             index/
                 index.phtml
```

```
        helpers/
        filters/
html/
    .htaccess
    index.php
```

Minden kontrollernek van egy alapértelmezett metódusa, amely alap állapotban az IndexAction. Azonban ez az Initializer.php-ben megváltoztatható, módosítható. Az hogy melyik kontroller töltődik be, az URL path segítségével dönti el a rendszer. A http://pirosattila.info/kezdolap/index URL-ből az szűrhető le, hogy a kezdőlap kontroller töltődik be automatikusan és annak is az indexAction metódusa. Minden kontrollerhez tartozik egy nézet script is, melyet kötelezően létre kell hozni, ellenkező esetben egy kivételt dob a kivételkezelő, mely figyelmeztet minket, hogy létre kell hozni.

3.3.1. IndexController

Ez a kontroller nálam csak egyetlen egy dolgot végez, méghozzá azt, hogy egy fileból betölti a kezdolap.phtml tartalmát. Ebben a fileban van a kezdőlapon elhelyezkedő statikus szöveg. A fileban való tárolásra azért volt szükség, hogy a későbbiekben megnövekedett látogatók számával az adatbázis nagy valószínűséggel nagyobb terhelést kap és ezt szerettem volna ezzel tehermentesíteni.

3.3.2. KezdolapController

Ebben a kontrollerben sem történik semmilyen lényeges. Egyszerűen egy redirekt hajtódik végre, mely az IndexController tartalmára visz. Ennek lényege abban nyilvánul meg, hogy az URL-ben szebben jelenhessen meg.

3.3.3. AuthController

Az első érdemleges dolog ebben a kontrollerben van implementálva. Az osztályon belül található kettő metódus. Az egyik a bejelentkezéskor, azaz a loginkor lesz lényeges, a másik pedig a kijelentkezésnél (logoutnál).

A *loginAction* metódus a bejelentkezésért felelős.

```
public function indexAction() {
  // TODO Auto-generated AuthController::indexAction() default action
}

public function loginAction() {
  $this->view->langs = Zend_Registry::get("langs");
  $this->view->isTrue = false;
  if ($this->_request->isPost()) {
    $request = $this->_request->getParams();
    $users = new Users();
    $isUser = $users->getUser($request["username"], $request["password"]);
    /*
     * Sikeres bejelentkezes
     */
    if ($isUser == true) {
      $_SESSION["username"] = strtolower($request["username"]);
      $_SESSION["userid"] = strtolower($users->getUserIdByName($request["username"]));
      $_SESSION["group"] = strtolower($users->getGroupsById($request["username"]));
      $errorString = "Sikeres bejelentkezés ".$request["username"]."!";
      $this->view->isTrue = true;
```

A legelején egy isTrue változónak false értéket állítok be, ezt azért alkalmazom, mert alapjába véve feltételezem minden látogatóról, hogy nincs bejelentkezve. A $this->_request->isPost() belső függvény segítségével megvizsgálom, hogy történt-e form küldés, azaz megtörtént-e a bejelentkezés űrlapnak a kitöltése. Amennyiben igen, le kell ellenőrizni, hogy az adott felhasználónév és jelszó páros létezik az adatbázisban, vagy sem. Miután megtörtént az űrlap kitöltése az $isUser változó újra értéket kap a kiértékelésnek megfelelően. Ha sikeres a bejelentkezés, true értéket vesz fel és pár dolog alapvetően tárolásra kerül a későbbiek azonosítása végett. Azt, hogy a felhasználó a bejelentkezés után tudja használni a regisztrált felhasználók előnyeit a session segítségével érem el. Egy munkamenetet nyitva tárolom el a felhasználónevét, a felhasználó azonosítóját, valamint hogy melyik csoport tagja. Ezután az $this->view->isTrue értéke true értéket kap, majd meghívódik újból az oldal, amely ezután már azonosítva a bejelentkezett felhasználót töltődik be. Ki kell térnem arra az esetre, amikor sikertelen a bejelentkezés, ilyenkor egy errorString nevezetű változó a „Sikertelen bejelentkezés!” értéket kapja. Ezen változónak az *authControllerhez* tartozó nézetben lesz jelentősége, melyre a későbbiekben kitérek majd.

Természetesen, ha volt bejelentkezés, egyszer ki is kell jelentkeztetni egy belépett usert. Ilyenkor a logoutAction fut le, melyben az eddig lefoglalt munkamenetet szabadítom fel az összes benne lévő tartalommal együtt. Ilyenkor a felhasználó számára megszűnnek a bejelentkezéssel járó jogai és ismét anonymuskét tud böngészni. Amennyiben újra szeretne például fórumbejegyzést írni, ismételt bejelentkezés szükséges.

3.3.4. CsapattagokController

Három metódusos osztály.

Ebből az első az *indexAction*, melyben az összes eddig regisztrált felhasználó listázásra kerül. A users modulban található *getAllUsers* függvényt hívva egy rendezett tömböt kapunk. a $params változó egy opcionális paraméter, amely egy rendezés funkciót tölt be a megadott paraméternek megfelelően.

A *personalmodifyAction* az a metódus, melyben a regisztrált felhasználók adatmódosítást tudnak végrehajtani. A $params változó a GET illetve a POST paramétereket egyaránt magában hordozza. Ezálltal nagyon könnyen feldolgozhatóak az adatok. A $groups változó egy kontroller helperben megírt függvény értékét kapja meg, amely azt, mondja meg hogy melyik csoportban van az éppen bejelentkezett felhasználó.

```
public function personalmodifyAction() {
  $users = new Users();
  $params = $this->getRequest()->getParams();
  $groups = $this->_helper->groups->getGroups();

  if ($this->_request->isPost() and !$params["user"]) {
    if (empty($params["username"]) or
      empty($params["password1"]) or
      empty($params["password2"]) or
      empty($params["lastname"]) or
      empty($params["firstname"]) or
      empty($params["gender"]) or
      empty($params["birthday"]) or
      empty($params["email"])
      ) {
      $errorString = "Minden mező kitöltése kötelező!";
    }
    elseif (strtolower($params["password1"]) != strtolower($params["password2"])) {
      $errorString = "A jelszavak nem egyeznek!";
    }
```

Ezek után egy jól megszokott rutin ellenőrzés hajtódik végre, ez pedig az űrlap elküldésének ellenérzése. Ha az adatmódosító form elküldésre került, akkor le kell ellenőrizni, hogy az összes input (beviteli) mezők ki lettek-e töltve. Amennyiben nem, abban az esetben az $errorString változó a „Minden mező kitöltése kötelező!” értéket veszi fel. Ezek után mivel ezen űrlapon a jelszavak is módosíthatóak, leellenőrzöm, hogy a jelszó mezők egyező értékeket vesznek fel, avagy sem. Ezután egy rutin e-mail cím ellenőrzés hajtódik végre, mely a helyes szintaktikát figyeli. Erre egy regexp kifejezést használok, mely a következő:

'^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z0-9]+$'

Ezek után ha minden kivételkezelésen sikeresen átment, meghívódik egy módosító metódus, melynek paraméterül adva az űrlapon kitöltött értékeket felülírja az adatokat. Az updateProfile lefutásától függ az $errorString értéke. Sikeres lefutáskor a „Sikeres adatmódosítás” értéket veszi fel.

Az űrlap elküldése előtt a bejelentkezett felhasználó láthatja eddigi adatait. Azonban egy plusz funkcióval egészül ki amennyiben root adminisztrátori jogosultságai vannak egy felhasználónak, hiszen nem csak a saját adatait módosíthatja, hanem bárki másét.

A következő action a *usersadAction*. Ebben van leimplementálva a regisztrációval kapcsolatos kód valamint cseles módon elrejtve benne a törléssel kapcsolatos dolgok. A felvitelben természetesen egy form áll rendelkezésre, melyben megadható paraméterek egy rutin ellenőrzés után bekerülnek az adatbázisba. Ezen funkcióját az oldalnak csak és kizárólag a root admin számára aktív. Megbizonyosodva arról, hogy a bejelentkezett felhasználó adminisztrátori jogosultságokkal rendelkezik és kitöltötte az űrlapot, valamint egy hidden (rejtett) POST paraméterként megkapja a „useradd” értéket, a regisztrációs kódrészlet fog lefutni. Ezek után, amennyiben az összes mező kitöltésre került, valamint az adatbázisban nincs már ilyen nevű felhasználó semmilyen más akadálya nem merül fel annak, hogy egy új bejegyzés kerülhessen az adatbázisba.

```
if ($params["useradd"]) {
  if(empty($params["username"]) or empty($params["password"]) or empty($
    $errorString = "Minden mező kitöltése kötelező!";
  }
  elseif ($users->isUser($params["username"]) == true) {
    $errorString = "Ilyen felhasználónév már létezik!";
  }
  else {
    $useradd = $users->useraddAction($params["username"], $params["passw
    if (!$useradd) {
      $errorString = "Nem sikerült felvinnem az új tagot!";
    }
    else $errorString = "Az új tag sikeresen bekerült az adatbázisba!";
  }
}
elseif ($params["userdel"]) {
```

Am ennyiben törölni szeretnék egy már előzőleg beregisztrált tagot, abban az esetben egy „userdel” POST paramétert kell figyelni. ilyenkor egy listából választva tudom megadni hogy melyik usert szeretném törölni. Ha megtörtént a kiválasztás, akkor hajtódik végre a *userdel*

metódus, melynek egy paramétere a felhasználó azonosítóját várja, mely minden bejegyzést töröl az adott felhasználóról.

3.3.5. ElerhetosegekController

Az ebben implementált kód nagymértékben hasonlít a kezdolapéhoz. A cél itt is ugyan az volt, hogy tehermentesítsem az adatbázist, éppen ezért egy file tartalmának a betöltése hajtódig végre egy egyszerű filekiolvasással. A kiolvasott file neve: elerhetosegek.phtml. Ennek a filenak a későbbiekben mindenféleképpen írási joggal kell rendelkeznie, hiszen csak abban az esetben kerülhető bele bármiféle módosítás.

3.3.6. ForumController

Ez a kontroller az egyik legösszetettebb kódját tartalmazza az alkalmazásnak. Ezt jelzi az is, hogy öt komolyabb metódussal rendelkezik. A fórum megírásakor törekedtem arra, hogy az egyszerű alapvető dolgokat tartalmazza, azonban többet tudjon mint egy egyszerű vendégkönyv.

Az *indexAction*, azaz a fórumtémák listázását teszi elérhetővé, illetve adminisztrátori jogosultságokkal rendelkező felhasználók módosíthatják a fórumtéma nevét, illetve beállítható, hogy egy aktív illetve egy nem aktív fórumtémáról van szó. Valamint ezen felhasználók új témákat is létre tudnak hozni. Ekkor meg kell adni a téma címének a nevét, valamint azt, hogy aktív vagy inaktív a téma.

```
  if ($this->_request->isPost()) {
    if (empty($params["themename"]) or empty($params["activate"])) {
      $errorString = "Minden mező kitöltése kötelező!";
    }
    else {
      if($params["activate"] == "active") $activate = 1;
      else $activate = 0;
      $newForumTheme = $forumtheme->createForumTheme($params["themename"], $
      if (!$newForumTheme) {
        $errorString = "Nem tudtam létrehozni a témát!";
      }
      else $errorString = "Sikeresen létrehoztam a témát!";
    }
  }
  $this->view->errorString = $errorString;
}

if ($groups == "Root admin" or $groups == "Fórum admin") $forumthemes = $for
else $forumthemes = $forumtheme->getAllForumthemes(1);
```

A metódus legvégén hívódnak meg azon lekérdezések melyek az adatbázisból az adatokat kérdezik le. Erre azért volt szükség, hogy az adatok elküldésével és módosításával egyidejűleg csak egyszer kell lekérni a friss adatokat.

A *thisAction* metódusnak a lényege az, hogy egy adott fórumtémán belüli adatokat listáz, valamint regisztrált felhasználók számára bejegyzéseket lehet felvinni az egyes témákhoz. Hozzászólások bevitele azonban opcionális, ha valaki csak olvasni akar a bejegyzések között, akkor csak olvas, ha hozzá is akar szólni, akkor hozzá is szólhat. A szokásos űrlap elküldés vizsgálata után már csak azt ellenőrzöm, hogy a beviteli mező ki van-e a töltve. Amennyiben nincs üres bejegyzés nem kerülhet az adatbázisba. A metódus legvégén beállításra kerül pár nézetben használatos változó:

- $this->view->groups = $groups;
- $this->view->comments = $comments;
- $this->view->thisForumTheme = $thisForumThemes;
- $this->view->errorString = $errorString;

.

A *modifyAction* csak az adminisztrátorok számára elérhető funkció. Egy kiválasztott téma nevét illetve azt hogy aktív vagy inaktív téma lehet módosítani. A szokásos rutin formküldés után egy általános input beviteli mezőnek a meglétét vizsgálom, melynek tartalom nélküli állapota hibakezelést igényel. Amennyiben a téma neve mező üresen van elküldve, visszaugrat az oldalra a következő hibaüzenettel: „A téma neve nem lehet üres!”. Azonban ha minden adat helyesen lett kitöltve, akkor az adatbázisba egy mentés készít a változtatott adatokkal. Ha egy adminisztrátori jogosultsággal nem rendelkező felhasználó szeretné mégis ezt a funkciót igénybe venni, a rendszer annyira intelligens, hogy még csak véletlenül sem tudja használni és átugratja a főoldalra, mintha egy ismeretlen oldalt szeretett volna betölteni.

DeleteAction: Természetesen ez is adminisztrátori jogosultságot igényel. Egy fórumtéma törtlése kapcsolatban van az adott témához bejegyzett adatokkal is. Egészen addig nem törölhető egy téma, amíg abban létezik bejegyzés. Éppen ezért szükséges a bejegyzések

elsődleges törlése. Egy kiválasztott téma törlésénél először egy biztonsági oldalra irányít át, melyben megadható hogy valóban kívánja az adminisztrátor törölni a témát vagy sem. Amennyiben nem törli, visszaugrat a témákhoz. A törlés elfogadásakor minden bejegyzésével együtt törlődik a fórum téma is, mely egy helyre nem állítható folyamat, azaz ha egyszer kitöröltem valamit, már nem hozható vissza.

Azonban lehetőség van arra is, hogy egy fórum témán belüli bejegyzést töröljenek az admin felhasználók. Erre a *deletethisAction* függvényem lesz segítségemre. Teljesen hasonlóan működik mint az előző metódus, annyi különbséggel, hogy csak egy bejegyzés törlése engedélyezett egy időben. A törléskor egy űrlap fogad, melynél ki lehet választani, hogy valóban törlésre kerül a bejegyzés. Amikor igen válasz érkezik a fordítóhoz, akkor a bejegyzés meghívja a *deleteViewsByRoot* metódust, mely a Forumview modulból példányosított függvényként érhető el. Sikeres törlés esetén már nem jelenik meg a bejegyzés az adatbázisban, ellenkező esetben hibaüzenet fogad bennünket: „Nem tudtam törölni a bejegyzést!”.

3.6.7 VersenynaptarController

Ebben a kontrollerben két metódus található. Az első igen egyszerű, míg a másik egy kicsit komplikáltabb kódot tartalmaz.

Az *indexAction* egy egyszerű sql select segítségével lekérdezi az adatbázisból a versenyekről tárolt információkat (getCompetition), majd egy tömb típusú változónak adja át értékül.

A versenylista feltöltéséről a *competitionupdateAction* gondoskodik. Ez a függvény egy file alapján dolgozza fel a tartalmat és importálja az adatbázisba.

A feldolgozandó file felépítése:

verseny neve|helyszín|verseny kezdete|verseny vége

Mikulás Kupa|HU, 1021 Budapest Jagelló út 32|2009-04-12|2009-04-14

Hajdú Kupa|HU, 1021 Debrecen Nagyerdei út 27|2009-06-20|2009-06-21

Szeged Kupa|HU, 1021 Szeged Jagelló út 32|2009-04-12|2009-04-14

Látható, hogy a tartalom feldolgozás nélkül nem igazán értelmes adatokat tartalmaznak. Azonban feldolgozás után az adatbázisba kerülhetnek. A metódus elején tisztázni kell a file helyének abszolút elérésének a helyét, ez alapértelmezetten a application/versenyek. Egy

biztonsági vizsgálat után a file megnyitásra kerül és soronként történik a feldolgozás. Minden sor az adatbázisba kerül egészen a file végéig. Az egyes sorokon belül a pipeline („|”) elválasztóval vannak elkülönítve egymástól. Ezeket az explode függvénnyel dolgozom fel, majd ezután következhet az adatbázis feltöltése. Arra kell figyelni, hogy amikor egy ilyen feldolgozás történik, akkor az előzőekben felvitt adatok törlődnek az adatbázisból és helyükre az új adatok kerülnek bejegyzésre.

```
public function competitionupdateAction() {
  $competitions = new Competition();
  $groups = $this->_helper->groups->getGroups();
  $filename = "../application/versenyek";
  if($groups == "Root admin") {
    if(file_exists($filename)) {
      $rows = file($filename);
      foreach ($rows as $value) {
        $data = explode("|", $value);
        $competitions->addCompetition($data[0], $data[1], $data[2], $data[3]);
      }
      print "Sikeresen feltöltöttem a versenyeket!";
    }
    else {
      print "Jelenleg a legújabb versenynaptár található az oldalon!";
    }
  }
}
```

3.6.8 ModifyController

Ebben az osztályban egy metódus található, amely az előzőekben említett statikus fileok tartalmát képes módosítani a változtatott értékek alapján.

Az *indexAction* metódusban zajlik le e funkció. Az URL-ben átadott paraméter határozza meg, hogy melyik file töltődik be a következő keppen:

$filename = "../application/fonix/files_data/".$params["page"].".phtml";

A $params["page"] csak az elérhetőségek illetve a kezdőlap paramétereket kaphatja, ellenkező esetben nem történik semmi. A filenév meghatározása után írásra nyitódik meg a file és az újonnan felvitt adatokat menti el. Sikeres módosításnál legközelebbi újratöltésnél már a frissen módosított adat jelenik meg.

3.6.9. ErrorController

Ebben az osztályban egyetlen metódus szerepel, mely egy Zend specifikus függvény. Minden olyan kivétel ide fut bele, amelyet a Zend Framework írói készítettek. Érdemes megfigyelni ennek a felépítését:

```
public function errorAction()
{
    $errors = $this->_getParam('error_handler');
    switch ($errors->type) {
        case Zend_Controller_Plugin_ErrorHandler::EXCEPTION_NO_CONTROLLER:
        case Zend_Controller_Plugin_ErrorHandler::EXCEPTION_NO_ACTION:
            // 404 error -- controller or action not found
            $this->getResponse()->setRawHeader('HTTP/1.1 404 Not Found');
            $this->view->title = 'HTTP/1.1 404 Not Found';
            break;
        default:
            // application error; display error page, but don't change
            // status code
            $this->view->title = 'Application Error';
            break;
    }

    $this->view->message = $errors->exception;
}
```

Ez a modul azt várja hogy egy vezérlő átugrassa és kezelje a hibákat. Az alapértelmezett modul ebben az osztályban az *errorAction*.

3.4. Nézetek (Views)

A Zend_View egy osztályban dolgozik a "view" résszel, a modell-nézet-vezérlő mintával. Lényegében két fő lépésből áll:

1. Saját vezérlő parancsok létrehozása a Zend_View példányaként
2. A Zend_View ellenőrzi a kimenetelt és megjeleníti a végeredményt.

Minden nézet elején definiálható a title, ezáltal minden oldalon más-más érték adható meg neki.

3.4.1. Auth

Az auth nézetben található a login nézet.

Jól látható a két kép közötti különbség. Az egyik a bejelentkezés előtt állapotot jeleníti meg, míg a másik a login utáni állapotot ábrázolja.

3.4.2. Index

Az index nézet a kezdőlapon elhelyezkedő információkért felelős. Beállítom benne a title-t és az oldal placeholderét.

3.4.3. Kezdolap

Mivel ez a nézet teljesen megegyezik az Index viewval, ezért erre nem térnék ki.

3.4.4. Csapattagok

Ez a nézet 3 külön oldalból áll:

- index.phtml
- personalmodify.phtml
- useradd.phtml

Az index view egy álalános információ megjelenítésében segít. Az érdekessége, hogy a megjelenítéshez a jquery nevezetű javascript gyüjteményt használom, amely érdekes effektjeivel hangulatossá, érdekessé teszi az oldalt.

A felső részen látható, hogy az oldalon linkek segítségével rendezhető a listázási sorrend. Valamint a névre, illetve arra a boxra klikkelve amiben a név is megjelenik, egy látványos effektel „átgördül” a kiválasztott névre. Listázásnál kerülendő a táblázatos elrendezés, ugyanis a méretezéskor minden böngésző máshogyan viselkedik. Éppen ezért a div megoldást alkalmazom helyette.

A personalmodify.phtml tartalma egy módosító form mező. Ezen az oldalon az adminisztrátorok bárki azonosságát módosíthatják. Különlegessége ennek az oldalnak, hogy a születési dátum mező könyebb tárolása-megjelenítés érdekében szintén egy jquery-s modult alkalmaztam. Ennek a neve a datepicker.

A felhasználók hozzáadása, azaz az useradd is egy form kitölrését valósítja meg. Illetve ehhez tartozik még a felhasználók törlése is. A beviteli input mezőkön kívül select boxok is találhatóak a megjelenített oldalon.

A felhasználó hozzáadása:

Felhasználók törlése:

3.4.5. Elérhetőségek

Az elérhetőségeknél a csapattal kapcsolatos információk láthatóak. Leginkább a kontakt információk megjelenítésére kell itt gondolni.

3.4.5. Forum

Alapértelmezésként csak a fórumtémák jelennek meg, azonban adminisztrátor hozzáféréssel több funkció válik aktívvá. Öt nézetre bontható a forum view:

1. delete.phtml
2. deletethis.phtml
3. index.phtml
4. modify.phtml
5. this.phtml

.

Ezek sorrendben a következőképpen néznek ki,

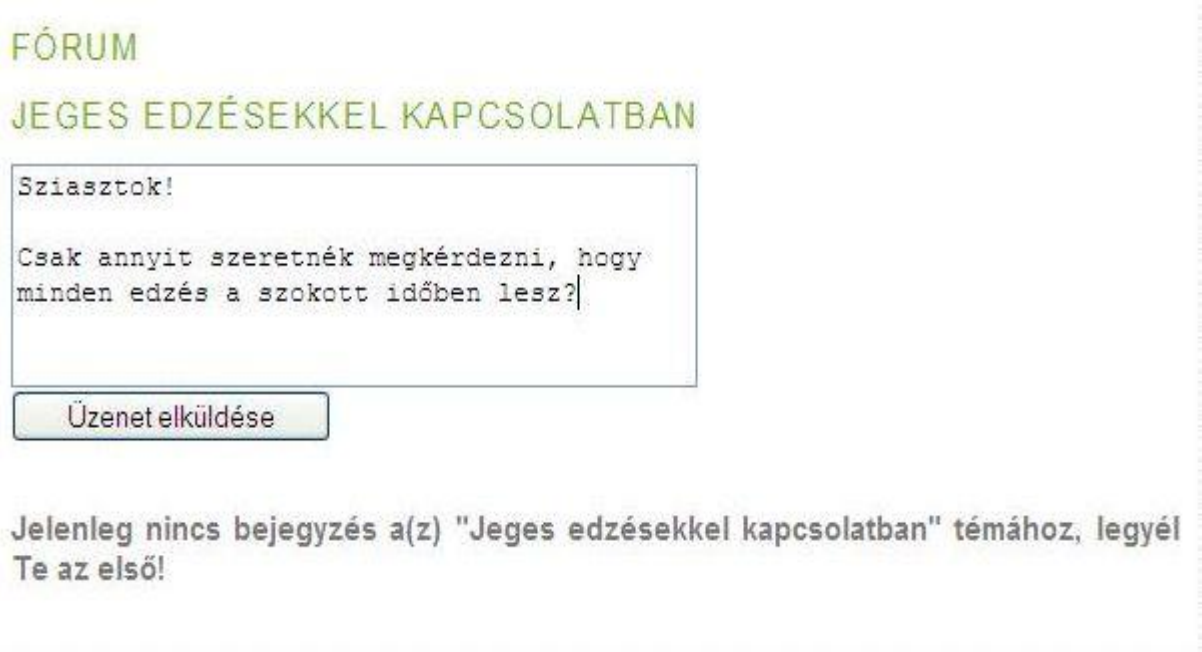
- delete.phtml

- deletethis

- index.phtml

- modify

- this



3.4.6. Versenynaptár

A versenynaptár az aktuálisan felvitt versenyekröl nyújt információkat. Itt lehet tájékozódni, hogy a soron következő versenyek mikor lesznek.

VERSENYNAPTÁR

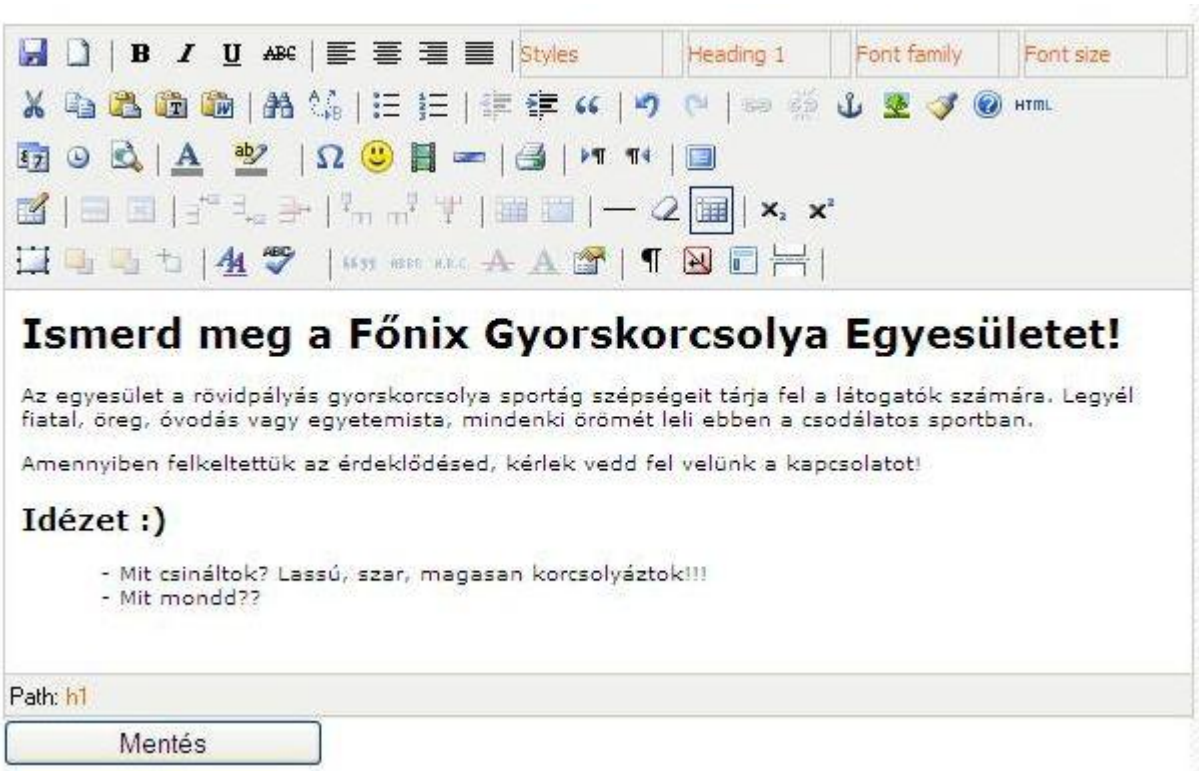
Név	Hely	Dátum
Mikulás Kupa	HU, Budapest - Gyakorló jégcsarnok	2008-04-16 - 2008-04-19
Hajdú kupa	HU, Debrecen - Debreceni jégcsarnok	2008-03-03 - 2008-04-06
Mikulás Kupa	HU, 1021 Budapest Jagelló út 32	2009-04-12 - 2009-04-14
Hajdú Kupa	HU, 1021 Debrecen Nagyerdei út 27	2009-06-20 - 2009-06-21
Szeged Kupa	HU, 1021 Szeged Jagelló út 32	2009-04-12 - 2009-04-14
Miskolc Kupa	HU, 1021 Miskolc Jagelló út 32	2009-04-12 - 2009-04-14
Pécs Kupa	HU, 1021 Pécs Jagelló út 32	2009-04-12 - 2009-04-14

Jól látható az egyszerű elrendezés, amely az olvashatóság és az egyszerűbb elrendezés végett került kialakításra.

3.4.7. Modify

Ez a nézet a kezdőlap és az elérhetőségek módosításánál szükséges. Egy nagyon látványos javascript alkalmazással segítve az egyszerűbb szövegfelvitelt. Ennek a neve a Tiny mce. Egy másdik generációs WYSIWYG szerkesztőről van szó. Kezelhető vele a szövegen kívül mindenféle html szabványnak megfelelő kód elhelyezése is. Ezzel nagymértékben könnyítve az adatok felvitelét még a nem html hozzáértők számára is. Egy Word szerű kezelőfelületet látunk, melynek alkalmazása teljesen megegyezik a Wordével.

Példa a kezdőlap módosítására:

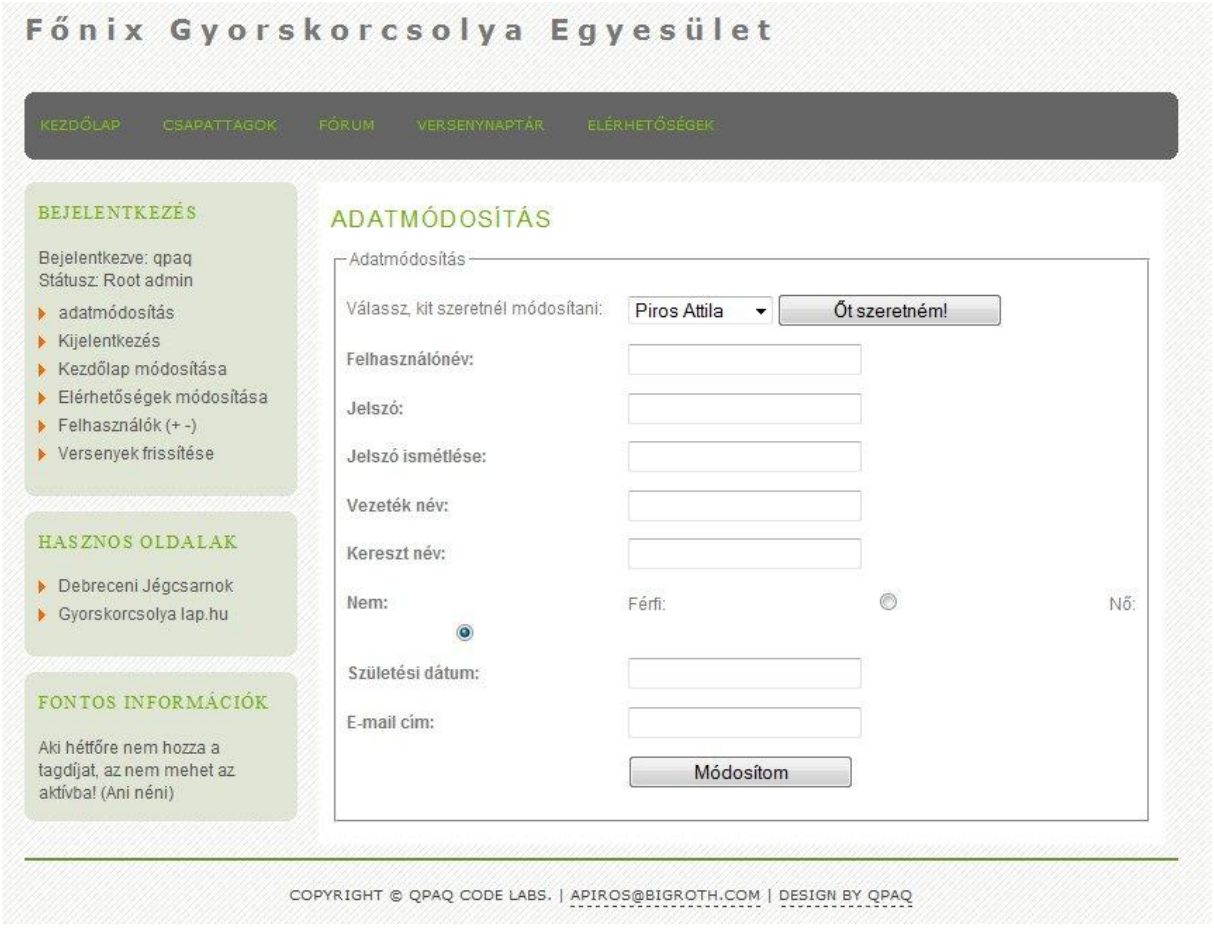


4. Képek

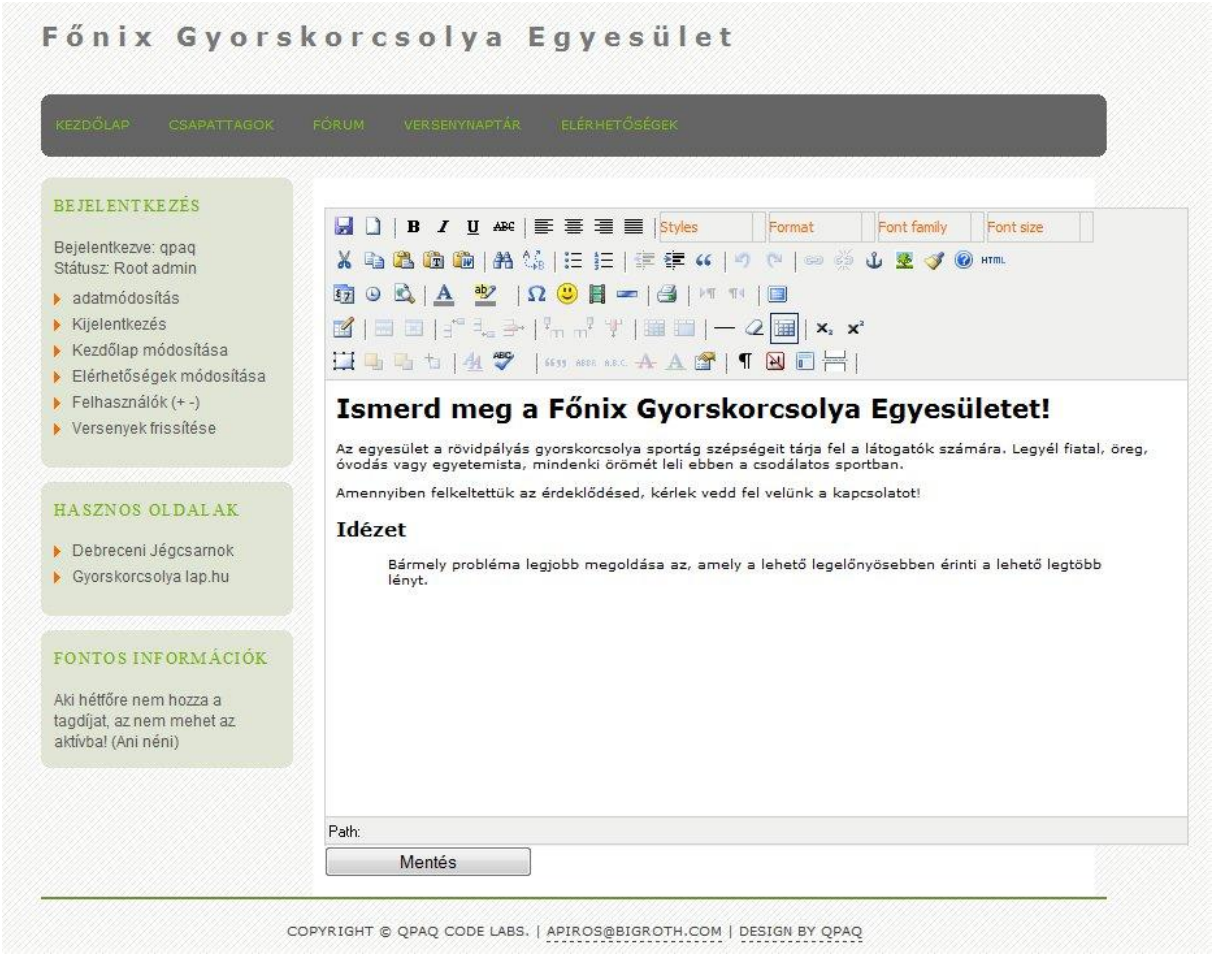
Kezdőlap

Csapattagok

Versenynaptár



Adatmódosítás



Kezdőlap módosítása

5. Összefoglalás

Az alkalmazás az elkészítése előtt eltervezett, kigondolt funkciókat teljes mértékben véghez tudja vinni. Minden funkciója működőképes. A felhasználói szintektől függően lehet böngészni az oldalt, adminisztrátori feladatokat elvégezni, egyszóval minden adott, hogy kellemesen eltölthesse az idejét a látogató.

Az alkalmazás kinézete igazodik a mai igényekhez, egy teljesen egyedi, jól használható designről van szó, bár lehetne rajta pár csicsásabb kép, ami feldobja az oldalt, de úgy gondolom, egy ilyen közösségi oldalnak, nem ez az elsődleges célja. Az oldal Internet Explorer és Firefox alá is optimalizálva lett, így bármilyen platformon elérhető és élvezhető.

A háttérkód jól strukturált, az MVC szabványnak teljes mértékben megfelel. Bár igen átlátható és követhető a kód, a fejlesztés után némi változtatást azért elvégeznék rajta. A Zend framework tökéletes keretrendszer és örülök, hogy ezt választottam.

A fejlesztés ideje – haladó PHP fejlesztő révén – nem vett sok időt igénybe, ez 1 hetet jelent. Egy hasonló weboldal elkészítése alapos tervezés és kivitelezés mellett 2 hetet venne igénybe. A fejlesztés során igyekeztem az oldalt modern jQuery technikák révén izgalmasabbá, valamint WYSIWYG szerkesztő segítségével az adatmódosítást könnyebbé tenni.

Mivel a PHP nyelv ismerete számomra nem újdonság, ezért tudtam mit várhatok ettől a fejlesztői nyelvtől, nem csalódtam benne és továbbra is szeretnék hasonló témákkal foglalkozni a jövőben.

6. Felhasznált technológiák, alkalmazások

APACHE 2.2.9 (Linux)

MYSQL 5.0.67

PHP 5.2.6

Zend Studio for Eclipse 6.0

Zend Framework 1.6.0

PHPMYADMIN 2.11.8

GIMP 2.6.1

MYSQL WORKBENCH

7. Irodalomjegyzék

Peter Moulding : PHP Fekete könyv

Felhasznált Internetes adat:

1. A PHP nyelv hivatalos dokumentációja: http://hu.php.net/manual/en

2. http://framework.zend.com/

3. http://www.w3schools.com/

4. http://hu.wikipedia.org/wiki/PHP

5. http://hu.wikipedia.org/wiki/Zend_Framework

8. Függelék

A README.txt fájl tartalmazza a program összes funkciójának futásához szükséges beállításokat.