PŐACTÉR

készítette:

Németh András

Jedlik Őskola

2014

.

Összefoglalás

Szakdolgozat egy Apród/Hardverapróhoz hasonló weboldalról, PHP, HTML, CSS alapokon. Ahol a felhasználók könnyen és gyorsan értékesíthetik használt/megunt tárgyaikat.

Bevezetés

Témaválasztás

A mai világban az internet széles körben elterjedt, az emberek életének a szerves részévé vált. A legtöbb felhasználó böngészőn keresztül éri el az internetet, amely ma már minden számítógépen megtalálható és a mobiltelefonok nagy részén is. Ez egy olyan multifunkcionális eszközzé fejlődik, amelyen keresztül a felhasználók információt gyűjthetnek, kommunikálhatnak, filmeket nézhetnek. A kis és nagy vállalatok számára a marketing mind az értékesítés területén elengedhetetlen eszköz. Szóval a jövőnek mindenképpen szerves része és egy olyan eszköz amire a fejlesztést mindenképpen érdemes megtanulni és hosszútávon egy jó befektetés.

Mivel teljesen lenyűgözött ez az eszköz, úgy döntöttem a nyáron hogy szeretnék én is foglalkozni vele, később beiratkoztam a Jedlikbe, és amikor a szakdolgozat témaválasztására került a sor a választás nem volt túl nehéz, valami olyat szerettem volna fejleszteni aminek a témája komolyabb mint egy játék. Később munkakeresésnél referenciának vagy egyetemen valamilyen formában újrahasznosítható. Nem utolsó sorban szakmailag kihívás és lehetőséged ad ezen a területen való fejlődésre.

Úgy gondoltam hogy a program logikai rész megírására a PHP-t használom mivel elég gyorsan lehet vele fejleszteni (dinamikus változók, megengedő syntax). Az oldal megformázásához HTML, CSS-re esett a választás de erről majd késöbb.

Eredeti specifikáció

Szakdolgozatnak egy piacteret fogok készíteni (vatera, ebay).

Megjelenítés: html, css, javascript. Szerveroldalon: php, mysql adatbázissal.

A felhasználóknak regisztrálniuk kell (emailes megerősítés).

Főoldalon véletlenül kilistázott termékek, kategóriák.

A felhasználók termékeket(hirdetéseket) tudnak feltölteni címszavakkal, képpel, kategória, leírással.

Keresés: címszavakra rálehet keresni, kilistázza a találatokat.

Szűrés bizonyos opciókra (szállítás, ár…)

A termékeknek külön kigenerált oldala lesz:

* Termékleírás
* Kép
* Termék helye
* Hirdetés érvényessége
* Kommentek
* Hirdető profilja

Az üzlet lebonyolítása után lehet értékeli egymásnak a feleket.

Őrodalomjegyzék:

* Tanuljuk meg a PHP5 használatát 24 óra alatt - Matt Zandstra
* stackoverflow.com
* caniuse.com
* Tiszta kód - Az agilis szoftverfejlesztés kézikönyve - Robert C. Martin
* Túlélőkönyv programozóknak - Robert C. Martin

Tervezés

Az oldal tervezésénél az volt az elsődleges szempont hogy egyszerű legyen és könnyen használható az egyszerű felhasználók számára is. Ez azért fontos, mert egy jól kialakított aukciós oldal szélesebb körben válik ismertté, népszerűvé, remélhetőleg ennek függvényében az oldal használóinak száma is nőni fog. Ellenkező esetben, ha egy oldal lassú vagy nem lehet tájékozódni rajta, a felhasználók inkább továbbállnak és keresnek egy másik oldalt a témában.

**Ehhez a következő tényezőket kell figyelembe venni:**

* Fontos a gyorsaság. Minél hamarabb tölti le az oldalat a felhasználó, annál élvezetesebb lesz számára a böngészés. A sebességet sok tényező befolyásolja, a kiszolgáló szerver távolsága, teherbírása, sebessége, a felhasználó sávszélessége.
* Az oldal kinézete ne menjen a gyorsaság rovására. Legfontosabb az információ szerzése, ehhez az információhoz kell minél hamarabb hozzájuttatni a felhasználót. Ennél az oknál fogva nem használok flash animációkat, nagyméretű képeket, különféle grafikus elemeket. A jól kiépített, könnyen használható felületnek nagyobb hasznát veszik a felhasználók, mint egy túl díszített, képekkel teletűzdelt oldalnak.
* Css használatával egyszerűbb egységes oldalakat készíteni, és változtatásokat is könnyebben lehet eszközölni.
* Modern megjelenés.
* Lehetőleg minden böngésző program azonos formában jelenítse meg az oldalt. Kisebb eltérések persze adódnak, de ez az oldal használhatóságát nem csorbíthatja.
* Sütik (cookie) engedélyezése nélkül is teljes funkcionalitással működjön az oldal.
* Navigáció: Egy felhasználó, ha az oldalhoz ér, könnyen fel tudja térképezni, mi hol van, mire kell kattintani. A legfőbb cél, hogy a felhasználó az adott információt a lehető legrövidebb időn belül megtalálja.
* Javasolt a legtöbb gépen lévő alapértelmezett fontok használata, mivel ezek általában könnyen olvasható betűk.
* Kerülendő a rikító, szemet fárasztó színösszeállítást.

Fejleszői környezet

A programkódok nagy részét a Netbeans ŐDE-ban írtam meg. (PHP, CSS, HTML)

Az adatbázis kialakításához és menedzseléséhez a Navicat-ot használtam.

Helyi hálózaton való teszteléshez az Easy PHP-t csomagját használtam:

* Apache
* MySQL

Teszteléshez a google chrome-t használtam a megjelenés erre van letesztelve, optimalizálva.

Nyelvek kiválasztása

Egy piactér szerű dinamikus oldal fejlesztéséhez elengedhetetlen egy jó programnyelv választása nálam a PHP-ra esett a választás. A böngészőben való megjelenítéshez a leíró nyelvek közül nem ekkora a választék, ezért a két legelterjedtebbre esett a választás a HTML5 és a CSS3-ra.

PHP

A programlogika (buisness logic) megírása ezen a nyelven folyik. A legújabb változatot használom mivel ez már támogatja az objektum orientált programozást és az új függvények is hasznosak. A syntaxisa megengedő nem kis-nagy betű érzékeny és néhány parancs többször is szerepel különböző néven, ezenkívül az összes változó dinamikus. Ezek tulajdonságok lehetnek ugyanúgy előnyök mint hátrányok mivel nagy szabadságot ad a programozóknak könnyedén lehet csúnya és átláthatatlan kódot írni. Ezeket figyelembevéve próbáltam valami egyszerű konvenciót kialakítani hogy az eltérő időben megírt kódok egyformán nézzenek ki. A programot szinte teljes egészében objektumorientáltan írtam meg mivel szerintem ez átláthatóvá teszi a kódot és többszemélyes fejlesztés, nagyobb projeketek esetén az procedurális programkód szóba sem jöhet.

MySQL

A MySQL egy több felhasználós, többszálú, SQL-alapú relációs adatbázis-kezelő szerver. Az MySQL az egyik legelterjedtebb adatbázis-kezelő, aminek egyik oka lehet, hogy a teljesen nyílt forráskódú LAMP (Linux–Apache–MySQL–PHP) összeállítás részeként költséghatékony és egyszerűen beállítható megoldást ad dinamikus webhelyek szolgáltatására. A MySQL honlapjáról grafikus felületű adminisztráló eszközök is letölthetők: MySQL Administrator és MySQL Query Browser. Széles körben elterjedt és népszerő alternatíva a PHP nyelven írt, nyitott forráskódú phpMyAdmin. A phpMyBackupPro (amelyet szintén PHP-ban írtak) adatbázisok (akár időzített, ismétlődő) mentésére szolgál eszközül. A phpMyAdmin a MySQL adatbázisok WEB alapú, teljes körő adminisztrációját teszi lehetővé.

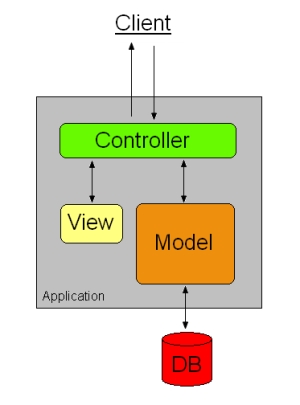
Apache

Az egyik legelterjedtebb nyílt forrású, szabadon terjeszthető webszerver, amelynek különböző változatai többek között Windows, Linux operációs rendszerek alatt is futtathatóak.

Framework-ök

Fat-free

A projekt programozás részének október körül álltam neki de rövidesen rájöttem hogy a programkód és a megjelenítés összevegyítése nem a legcélravezetőbb dolog hosszútávon. Ekkor a programozást egy kis időre abbahagytam és jött a kutatás rész, ekkoriban botlottam az „MVC” elnevezésbe ami a Modell-View-Controller rövidítése, ennek az a lényege hogy az oldalat 3 részre bontjuk:

* **Modell**: Őtt történnek a számítások, a vezérlőtöl megkapott adatok feldolgozása az adatbázissal való komunikáció, sajnos nálam ez néhol egybeolvadt a vezérlővel a gyorsabb fejlesztés érdekében.
* **Nézet**: ezen a felületen történik a felhasználói interakció, jellemzően egy felhasználói felület. Különböző célokra különböző nézetek létezhetnek ugyanahhoz a modellhez.
* **Vezérlő**: Ez a réteg köti össze a modell-t a nézettel. A bemeneti adatok ezen keresztül jutnak el a modellhez amely azt feldolgozza. A feldolgozott adatot ez adja át a nézetnek.
* 

1. ábra: Az MVC minta egy általános ábrázolása

Ez után tovább keresgéltem a megoldás után mivel ezt nem igazán tudtam elképzelni hogy hogyan programozzam le, ekkor az egyik ismerősöm ajánlotta hogy használjak egy PHP frameworkot, így esett a választás a Fat-Free frameworkre (késöbbiekben csak F3). Ez egy kisebb framework ami támogatja az MVC fejlesztési mintát. Legfontosabb funkciói:

**URL routing**: Ez az összes url-t az index.php ra irányítja egy HTACCESS file segítségével ahol aztán defeniálva van minden url-nek egy funkció ami annak a kérésnek megjelenít egy oldalat vagy elvégez egy feladatot. az ilyen útvolnalak lehetnek: staikusak és dinamikusak.

1-1 példa mindkettőre:

* **Statikus**:

$f3->route('GET /Register', 'RegisterController->display');

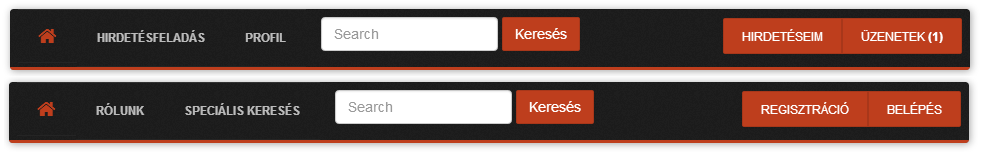
Amennyiben a felhasználó beírja az oldal nevét/register (ez alap esetben GET (request method)) meghívja a RegisterController osztály display funkcióját amely megjeleníti a regsztrációs oldalat.

* **Dinamikus**:

$f3->route('GET /Ad/@adid','AdController->specificad');

Amikor a felhasználó oldalneve/Ad/tetszőleges számot ad meg, meghívódik az AdController osztály specificad funkciója amely eldönti hogy létezik e ilyen ’id’-vel rendelkező hirdetés és ennek függvényében megjeleníti a hirdetést vagy a hibaüzenetet.

Amennyiben a routing listában nem definiált oldalat hívunk egy 404 oldalra dob.



2. ábra: A navbar belépett & nem belépett felhasználók számára

**Template kezelő rendszer:** Számomra ez a funkció is nagyon hasznosnak bizonyult ez különíti el a nézetet a logikai résztől. Az előző példák közül az első:

function display($f3) {

NAVBARController::buttons($f3);

echo Template::instance()->render('main.tpl');

echo Template::instance()->render('register.tpl');

echo Template::instance()->render('endofmain.tpl');

}

Ez egy elég egyszerű funkció, az első sora (NAVBARController::buttons($f3);) meghív egy button nevű funkciót ami beállítja az oldal tetején lévő gombok értékeit a attól függően hogy a felhasználó bevan-e jelentkezve. Ebből egy részlet:

if (isset($\_SESSŐON['username'])) {

$f3->set('signinstatus', 'Hirdetésfeladás');

$f3->set('signinstatuslink', '/adupload');

} else {

$f3->set('signinstatus', 'Rólunk');

$f3->set('signinstatuslink', '/aboutus');

}

A set nevű funkció beállít framework változókat bizonyos értékekre.

A display funkció következő sora:

echo Template::instance()->render('main.tpl');

kirendereli a main nevezetű templatet:

<!DOCTYPE html>

<html>

<head>

<title>Piactér</title>

<include href="style.tpl" />

</head>

<body>

<include href="navbar.tpl" />

<div class="wrapperSJ" id="content">

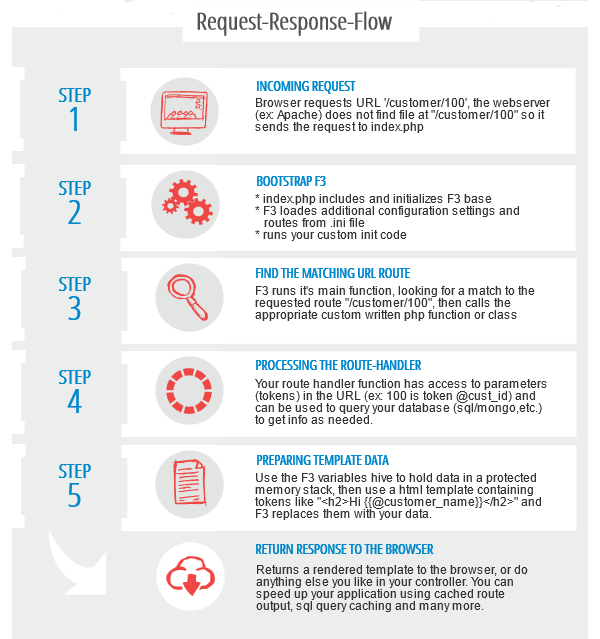
<div class="row">

<!-- END OD MAŐN TPL -->

Ez a rövid részlet lesz az oldal forrásának az eleje, egy hozzáértő ember rögtön észreveszi hogy ebben van kettő darab nem ide illő tag: <include href="style.tpl" /> és <include href="navbar.tpl" />. Az include tageket a template kezelő rendszer értelmezi és a megadott templateket beilleszti a helyére. Az első include a css-eket és javascripteket illeszti be. A második pedig a navbart, ebben találhatóak framework változók, pl:

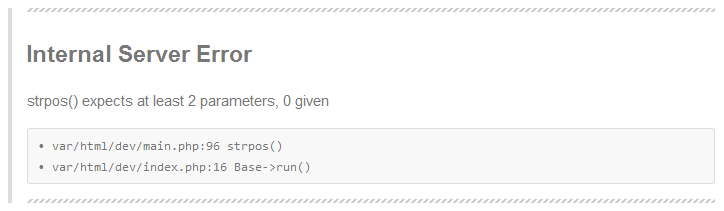
* {{ @signinstatus }}
* {{ @signinstatuslink }}

ezeknek az értékét állította be a NAVBARController::buttons($f3) úgyhogy itt a bejelentkezett felhasználók, a hirdetésfeladás gombot kapják „/adupload” linkkel és nem bejelenkezettek pedig egy Rólunk feliratú gombot kapnak „/about us” linkkel.



3. ábra Fatfree kérés-válasz folyamata

**Autoloader** egy framework által kezelt tömb amiben mappanevek vannak eltárolva ennek köszönhetően a funkciókban nem kell require/incude-olniaz osztályokat amelyeket használunk. Amikor osztályt példányosítunk vagy statikusan hívunk függvényt automatikusan megnézi a megadott mappákat és amennyiben ugyanolyan néven vannak elmentve a file-ok mint a bennük tárolt osztály (csak egy osztály lehet egy fileban) az autoloader megtalálja őket és meghívja őket helyettünk.

**Hibakezelő rendszer** az F3 hibakezelő rendszere a hibákat egy saját maga által kigenerál HTML oldalon jeleníti meg ezáltal is gyorsabbá téve a fejlesztést:A Framework rendelkezik egy debug nevű változóval ezzel lehet állítani a hibákat kiváltó források elemzésének a részletességét. 0-3 ig lehet állítani a nagyobb a szám a részletesebb leírás ez a változó nagyon hasznos mivel fejlesztésnél szükséges az összes hiba kiírása a hatékony debuggoláshoz míg amikor a program/oldal az interneten „élesben” fut ugyanez komoly biztonsági réseket fedhet fel a hozzáértők előtt. Ezért amikor a fejlesztő feltölti a termékét érdemes ezt a változót 0-ra állítani így az esetlegesen megjelenő hibákat kiváltó okok rejtve maradnak a felhasználók elött.

TWITTER BOOTSTAP

Mivel az oldal vizuális megtervezésre, megvalósítására nem akartam túl sok időt, eltölteni ezért úgy döntöttem hogy kipróbálom a Twitter-nek az „új” frontendes keretrendszerét. Ez tulajdonképpen kettő css file, az egyik az oldal elrendezését „layout” biztosítja hogy mindig egyforma maradjon, a másik pedig egy „skin” ami az előző rétegnek színt, árnyékokat, egy modern megjelenést ad. Nagyon fontos hogy minden második rétegben szerepelnek az első réteg elemei ebben a listában egy átlagos weboldal felépítéséhez minden szükséges elem megtalálható, input mezők, gombok, stb… Ezenfelül pedig általában minden fejlesztő ad hozzá néhány sajátot. Az egységesség azért fontos hogy az oldal megjelenését könnyen és gyorsan meglehessen változatni. Ha a framework képességeit teljes mértékében kihasználtam volna akkor most reszponzív lenne az oldalam (máshogy jelenne meg kisseb kijelzőkön) de sajnos amikor elkezdtem az oldalat fejleszteni a dokumentációt nemtanulmányoztam át elég alaposan. Aki megtanul ezzel az eszközzel felépíteni egy oldalat az sokkal gyorsabban halad mint aki mindent magától megírna (ez általában igaz az összes frameworkre, egy befektetés megtanulni, hogy később gyorsabban tudjon az ember fejleszteni vele).

Az elkészült program

# Áttekintés

Az oldal az előzőekben leírt keretrendszerek és program/leírónyelvek használatával készült.

# Az interface

A megjelenítés mindig 3 részből tevődik össze:

* Navbar, az oldal tetején helyezkedik el,ez tartalmazza a navigációhoz szükséges gombokat, logót, közösségi oldalra linkeket.
* Tartalom, ez tartalmazza mindig az éppen aktulás tartalmakat amire a felhasználó kiváncsi, itt történik az interakció, adatcsere nagy része.
* Footer, a lap alján található, az oldalról alapvető információk…

# Megvalósult dolgok, kihagyott funkciók

A specifikációban leírt dolgoknak a nagy része megvalósult, azokkal a nyelvekkel sikerült elkészítenem amelyeket az elején elgondoltam.

Amik nem valósultak meg:

* Emailes megerősítés regisztrációnál
* A hirdetéseknek nincsen kategóriájuk, nemigazán van kialakított hely címszavaknak
* Címszavakra rálehet keresni, kilistázza a találatokat.
* Szűrés bizonyos opciókra (szállítás, ár…)

**Emailes megerősítés regisztrációnál:** Ennek a funkciónak kihagyásának a legnagyobb oka volt hogy úgy éreztem ez nem egy olyan probléma ami programozás terén nagyobb kihívás. Itt az emailszerver beállítása jelentette volna a legnagyobb akadályt.

**A hirdetéseknek nincsen kategóriájuk, nem igazán van kialakított hely címszavaknak:**

Ez nem egészen igaz mivel a hirdetéseknek belehet állítani állapotot (Új/Használt), helységet hogy hol található meg, de a hagyományos értelemben vett kategóriák az oldalon nem léteznek, ez főleg azért maradt ki mivel kicsit másabb irányba ment el a fejlesztés és a végén már nem maradt rá idő.

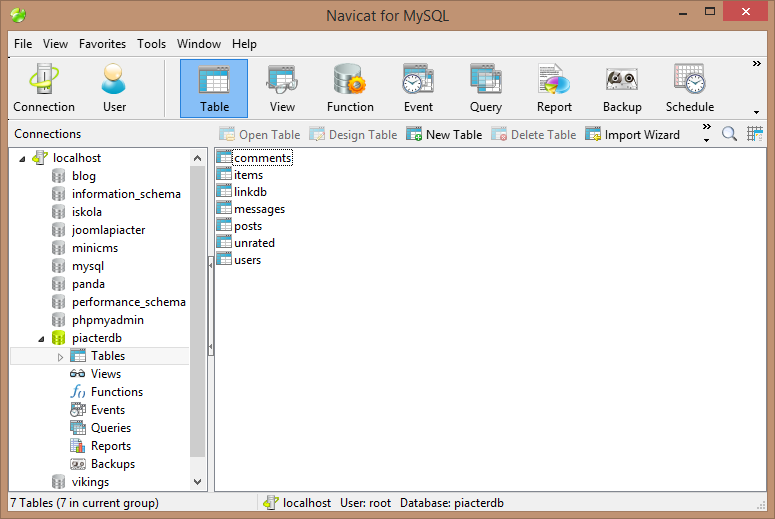
**Címszavakra rálehet keresni, kilistázza a találatokat. Szűrés bizonyos opciókra (szállítás, ár…):** Ezek a funkciók azért maradtak ki az oldalból mivel nem volt időm lefejleszteni őket.

Adatbázis

„A MySQL az egyik legelterjedtebb adatbázis-kezelő, aminek egyik oka lehet, hogy a teljesen nyílt forráskódú LAMP (Linux–Apache–MySQL–PHP) összeállítás részeként költséghatékony és egyszerűen beállítható megoldást ad dinamikus webhelyek szolgáltatására.”

*- http://hu.wikipedia.org/wiki/MySQL*

Azért esett a választásom a MySQL-re mivel ez az egyik legelterjedtebb adatbázis-kezelő, ami azt jelenti hogy ehhez van a legtöbb leírás, dokumentáció, valamint ha hiba merül fel több ember segítségére számíthatok.



4. ábra: Adatbázis

Táblák részletezése:

* **Comments:** Ebben a táblában tárolom a Hirdesekhez való kommenteket
* **Items:** Itt vannak a felhasználók által feltöltött hirdetések adatai, lejárat, cím, kép neve.
* **Linkdb:** Ez egy teszt tábla a keresés kipróbálására mivel rengeteg idő lenne annyi adatot felvinni hogy rendesen kipróbálható lehessen az items táblából. Ennek a táblának a tartalma a Vateráról lett letöltve egy kis script segítségével.
* **Messages:** Itt vannak az üzenetek amiket a felhasználók egymásnak küldenek valamint azok az üzenetek is amiket az oldal küld a hirdetéssel kapcsolatban (sikeres licit stb…)
* **Posts:** Ez is egy teszttábla ez jelenik meg a főoldalon
* **Unrated:** Ebben a táblában vannak eltárolva a felhasználók értékelései egymás felé.
* **Users:** Itt van az összes felhasználó személyes adata, email, jelszó(titkosítva), legutolsó belépés...

Nagy általánosságban igyekeztem optimalizálni az adatbázist, hogy semmilyen adat ne legyen többször eltárolva és mindent egyedi azonítóval érjek el ha egy másik táblában szükség van rá.

**Az eventek** a mysql 5.1.6-os verziójában jelentek meg. A cron jobok és az időzített feladatok mellé egy alternatíva. A lényege hogy egy bizonyos időben valamilyen feladatot eltudnak végezni (UPDATE, DELETE, INSERT…). A projektemben az aukciót lejárását oldottam meg velük, amennyiben egy felhasználó feltölt egy olyan hirdetést ami egy bizonyos idő után lejár akkor, a szerver egy eventet generál ami pontosan akkor fut le amikor a hirdetés lejár. Ezzel rengeteg erőforrást spórolunk meg ahhoz képest mintha ezt pl cron jobokkal végeznénk el és sokkal pontosabb is. Persze teljes egészében nem tudja kiváltani ezeket az időzített feladatokat.

Ez a funkció akkor hívódik meg amikor a felhasználó egy olyan hirdetést ad fel amihez van aukció is. Létrehoz egy eventet ami akkor lefut amikor lejár az aukció. Később egy kis script amit bizonyos időközönként lefuttat a szerver, leellenőrzi hogy melyik hirdetés lett így lezárva kiküldi a leveleket a felhasználóknak (amennyiben volt nyertes) és hozzáad egy üres értékelést a két felhasználó profiljához.

private function auctionevent($f3) {

$sessid = $\_SESSŐON['id'];

$connection = new PDOConnection;

$result = $connection->query("SELECT availability, id FROM items WHERE owner='$sessid' ORDER BY date DESC ")

->fetchAll(PDO::FETCH\_ASSOC);

$expire = $result[0]['availability'];

$adid = 'AD' . $result[0]['id'];

$connection->query("CREATE EVENT ŐF NOT EXŐSTS $adid

ON SCHEDULE AT '$expire'

DO

UPDATE `items` SET isopen=0 WHERE owner='$sessid' AND availability='$expire';");

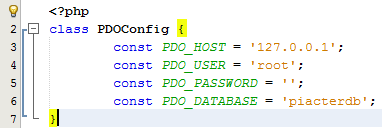
}

Adatbázis elérése PHP-ból

Jelenleg a PHP-nak 3 APŐ-ja van MySQL adatbázisok elérésére:

* **mysql:** Ez a legrégebbi, még mindig sok helyen használják a 3 legnagyobb problémája:
  + **Elavult (Deprecated):** A PHP már hivatalosan érvénytelenítette így várható hogy valamelyik következő verzióban kiveszik a támogatást, annak ellenére hogy nagyon széleskörben eltrejedt mind a mai napig.
  + **Nem biztonságos:** Minden adatot átad egyenesen a fejlesztőnek így olyan karakterek, szövegrészletek is bennmaradhatnak amelyek biztonsági problémákat vethet fel a nem elég tapasztalt fejlesztők számára.
  + **Flexibilitás:** A megírt kódok csak MySQL adatbázissal működnek, jelentős hiányosságai vannak támogatás terén.
* **mysqli:** Ez a mysql APŐ-nak egy fejlesztett verziója de ebben is jelentős hiányosságok lelhetőek fel, főleg mert ez sem túl flexibilis. Sebesség szempontjából viszont ez a leggyorsabb.
* **pdo:** A PDO nem más mint egy objektum az adatbázis kapcsolatok, lekérdezések, stb. kényelmes, hatékony, átlátható kezelésére. Rendkívül flexibilis több fajta adatbázis rendszert támogat, és amennyiben a program struktúrája megfelelően van kialakítva szinte szó szerint 2 kattintással kilehet cserélni a program mögött lévő adatbázist.

Eleinte a mysqli függvényeit használtam aztán inkább később mint hamar átálltam a pdo-ra, a projektemben nem sok gyakorlati előrelépést (kevesebbet kell az escapeléssel foglalkozni a prepared statementek is hasznosak) hozott azon kívül hogy már találkoztam ezzel a technológiával és használtam, elmondhatom hogy a legmodernebb technológiákat használom. Jelenleg 2 kis osztályt használok hozzá:



5. ábra: adatbáziskapcsolat

Ebben az osztályban definiálom a jelenlegi adatbázis kapcsolat. A fejlesztő annyi ilyen osztályt hoz létre optimális esetben amennyi adatbázissal akar dolgozni. Jelen példához a változók elnevezése nem a legjobb.



6. ábra: Kapcsolat

Ezen az osztályon keresztül létesítek kapcsolatot az adatbázissal. Mindenegyes funkcióban ahol kapcsolatot létesítek példányosítom és ezáltal a konstruktor beolvassa az előző osztályban lévő értékeket, így hogyha azt az osztályt módosítom vagy itt az elérési utat akkor egy egészen más adatbázissal is működne (elméletileg a query-k nem minden helyen működnek ugyanazzal a syntaxal).

FILEK ELRENDEZÉS A MAPPÁBAN DIY DIRECTORY STRUCTURE

FELHASZNÁLÓI DOKUMENTÁCŐÓ

Minden oldal 3 fő részből áll:

* Navbar
* Content
* Footer

Amikor egy oldal tartalmáról fogok beszélni a későbbiekben mindig a contentről lesz szó.

Navbar: Ez egy dinamikusan kigenerált felület amivel a felhasználó az oldalon navigálhat.

Content: Ez az oldal tartalma, általában dinamikus, itt történik az adatok átadása a gép-ember között.

Footer: Statikus felület itt egyszerű adatok jelennek meg az oldalról: kapcsolat, hírlevél stb….

# FŐOLDAL

Ez az oldal fogadja a felhasználót amennyiben fellép az weblapra. Itt van egy slider ahova lehet hirdetéseket rakni vagy bármilyen tartalmat, jelenleg üres. Ezen kívül egy kisebb táblából (könnyen át lehet állítani) kilistáz adatokat és ezeket szépen oszlopokba rendezi.

# Regisztrációs oldal

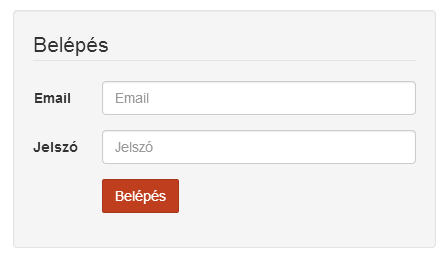
A legszükségesebb adatokat kell itt megadni:

* Email
* Felhasználónév
* Jelszó

Ezeket később bővítheti, pl bemutatokzással ha itt regisztrál minden felhasználói funkciót igénybe vehet.

# Belépés

Egy helyes email-jelszó párossal léphet be a felhasználó.



7. ábra Belépés

# Üzenetek

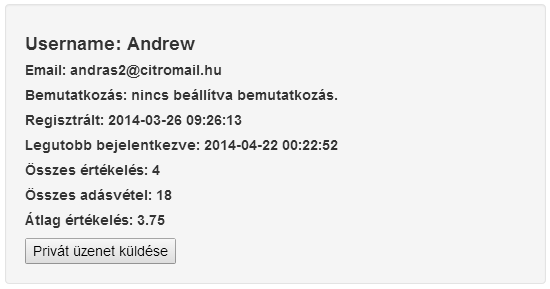
Itt a jelennek meg az üzenetek, értesítések amelyeket a felhasználó kap a többi felhasználótól, az oldaltól. Ez az oldal csak akkor elérhető hogyha a felhasználó bevan jelentkezve.

# Keresés

Ezzel egy adatbázisban hirdetések címében lehet keresni 50esével írja ki a találatokat.

# Profil

Minden felhasználónak van egy profilja ami minden más felhasználó számára publikus, kivéve az elérhetőségek, amik csak akkor érhetőek el ha a két felhasználó már kereskedett egymással.



8. ábra: Felhasználói profil

# Privát üzenet

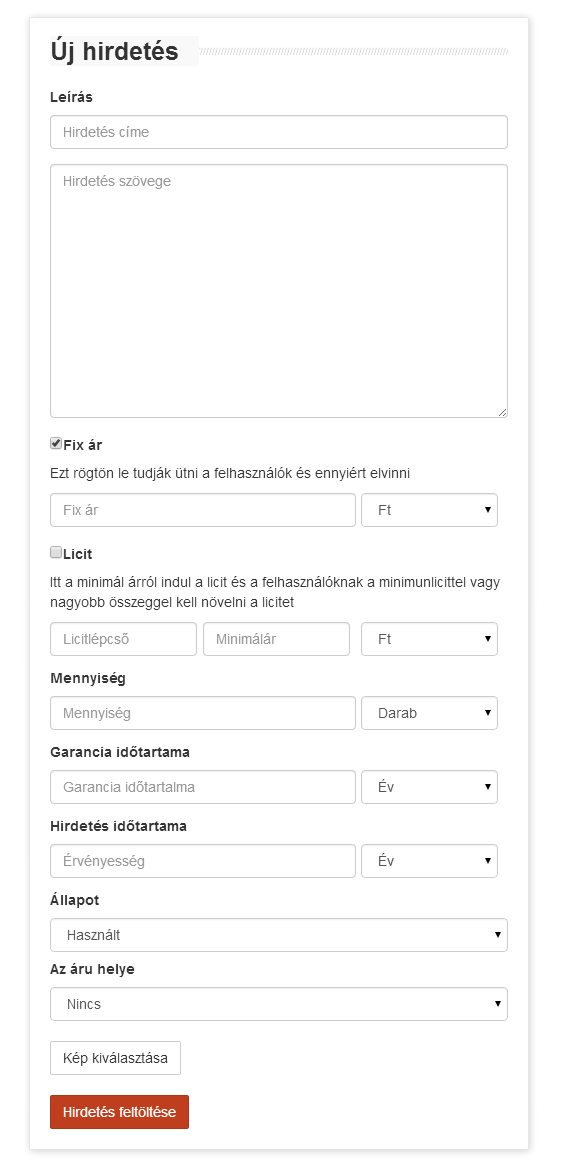
Mint a neve ˙is mondja a felhasználok küldhetnek egymásnak privát üzeneteket. Akinek a szeretnénk üzenet írni annak profiljára kell menni és ott rámenni „Privát üzenet küldése” gombra.

# Kommentelés

A felhasználok a hirdetések alatt tudnak egymással kommunikálni, kérdezni az eladótól publikusan kommenteken keresztül. Ez egy nagyon hasznos funkció hogyha valaki kifelejt valami fontos adatot a hirdetéséből, azt csak egyszer kell megkérdezni és az összes többi felhasználó azonnal látja és nemkell mindenkinek üzenetben megkérdeznie ugyanazt. Minden egyes komment rendelkezik saját url-el.

# Hirdetés feladása

Őtt tudnak a felhasználók feladni hirdetéseket mindenféle paraméterrel. A hirdetéseket 3 fő részre lehet osztani

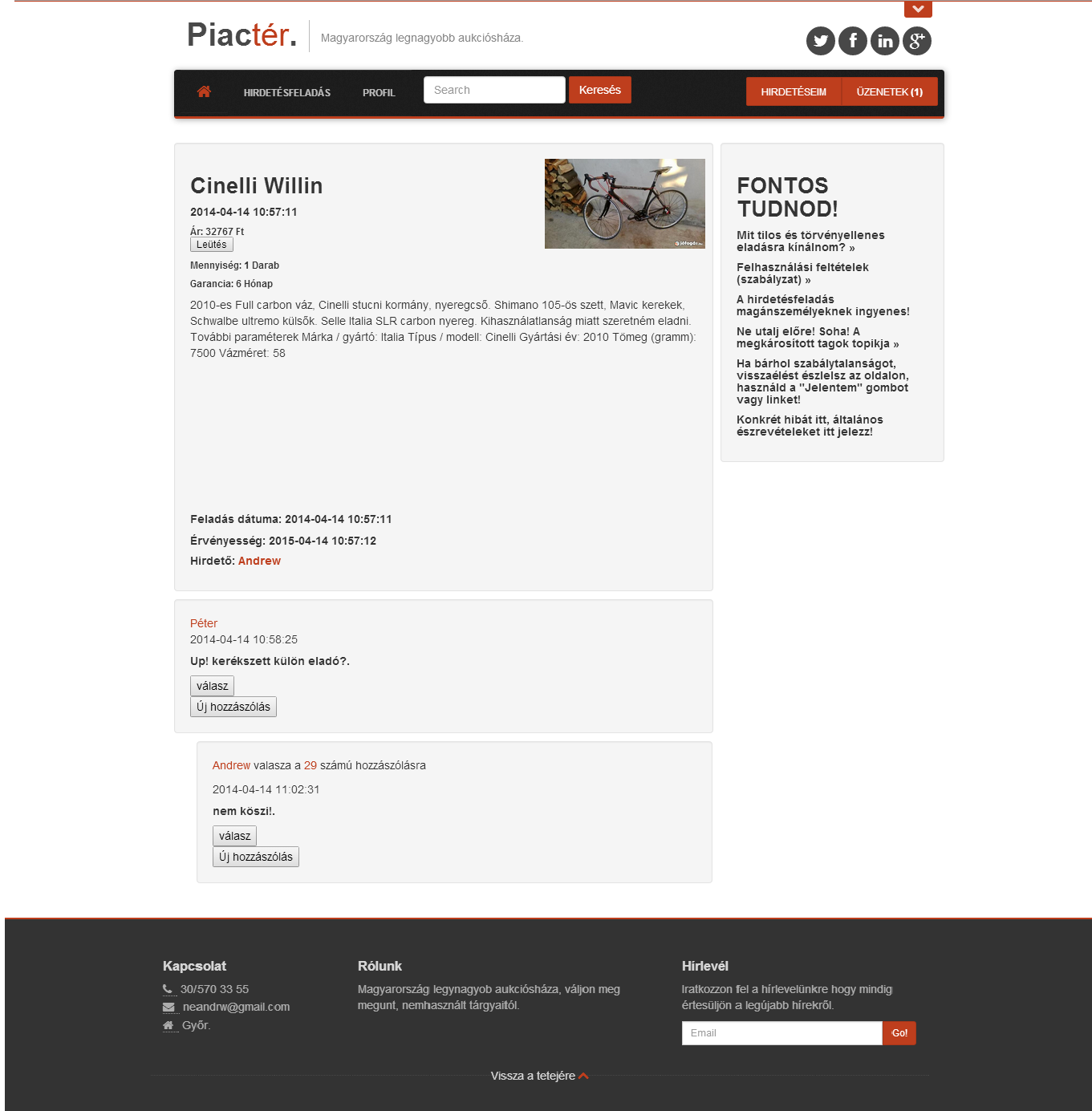


# Hirdetéseim

A felhasználó által feladott összes hirdetést listázza ki.

# Hirdetés-oldal

Ez egy dinamikusan kigenerált oldal ahol a felhasználók egy bizonyos hirdetés adatait nézhetik meg, kommentelhetnek.



GŐTHUB

A github egy verziókövető rendszer amely szoftverfejlesztést segíti és a nyilt forráskodú projektek megosztását. Több nagy projekt is ezt a rendszert használja pl: Android, Node.js, JQuery. A github főbb funciói:

* Verziókezelő
* Őssue tracking
* Wiki
* Közösségi tér
* Forks

**Verziókezelő:** Verziókezelésre a Git rendszert használja, ennek az a lényege hogy a felhasználóknak van egy „repository”-juk a gépükön amiben a projektjüket tárolják és van egy közös tárhely a szerveren. A megfelelő jogosultsággal rendelkező fejlesztők mindig feltöltik a saját frissítéseket a közös tárhelyre a többiek pedig letöltik. Amikor valami-t változtatnak a forráskódon (saját gépükön) és szeretnék azt publikálni akkor leellenőrzik hogy meggyezik-e a verzió a szerveren lévővel ha igen akkor egy kis magyarázattal feltöltik a változtatásukat, ezt nevezik „commit”-nak. Amikor nem egyezik a saját verziójuk a szerveren lévővel akkor előbb le kell tölteni a legújabbat és „merge”-elni (összeolvasztani) a sajátjukkal, és csak ezután tudják feltölteni. A szerver minden eggyes commitot eltárol, különbségeket a verziók között ez azért hasznos mivel megkönnyíti a code-reviewet (a jó fejlesztők mindig leellenőrzik egymás kódját hogy nehogy rosszminőségi kód kerüljön a projektbe), ha valami olyan frissítés kerül feltöltésre ami ’nemkívánatos’ akkor egész egyszerűen visszalehet állítani egy régebbi verziót.

**Őssue tracking:** Ez egy hibakezelő rendszer a lényege hogy a projektnek az oldalán hozzá lehet adni „ticketeket” amik hibákat vagy megoldandó feladatokat jelölnek, ezekhez lehet komenneteket fűzni stb.. Nagyon hasznos feature amennyiben a fejlesztő agilisan fejleszt, kijelöl egy adag ticketet hogy ezeket akarja megcsinálni a heti/havi sprintjében. Nagyon jól szemlélteti a hátralévő feladatokat.

**Wiki:** Ezzel a felhasználó a projektjéhez egy kis wiki oldalat adhat hozzá, amelyben részletesebben ismertetheti a funkciókat, a telepítéshez instrukciókat adhat stb…

**Közösségi tér:** A github nagyon hasonlít a facebookhoz a felhasználóknak a főoldalon ugyanúgy megjelenik egy „news feed” a vele kapcsolatok projektekről, fejlesztőkről. Lehet követni a többi felhasználót, híres fejlesztőket, megnézni hogy miket írnak.

**Forks:** Az úgynevezett forkok egy-egy projekt saját célú másolatai amiket mi magunk fejleszthetünk tovább. Nagyon hasznos hogyha megtetszik egy projekt de nem tetszik milyen irányba fejlődik, ekkor készíthetünk egy forkot, amelyben az egész projekt lemásolódik a saját profilunkhoz és ezután saját ízlésünk szerint fejleszthetjük tovább. A népszerűbb projekteknek gyakran van többszáz forkjuk.

Én is elkezdtem használni a piactér fejlesztése közben, nagyon hasznosnak találtam még úgyis hogy csak egyedül fejlesztettem. A kedvenc funkcióm az issue tracking lett, amit sajnos későn fedeztem fel ezért csak a végső fázisban tudtam használni. Ezeken a funkciókon felül nagyon hasznos, hogy egy szép grafikus felületen foglalja össze a projektet ezért könnyen bemutatható referenciaként, vagy akár önéletrajzba is beírható hogy ezen és ezen a projekten dolgoztam.

9. ábra: A github kabalafigurái.

LŐCENSZEK

Fatfree licensz:

GNU (GPL v3)

„A licenc lényege – más szabad licencekhez hasonlóan –, hogy a mű szabadon terjeszthető (akár pénzért is), és szabadon módosítható, de a terjesztései és a módosítások kötelezően szintén GPL licenc alatt kell, hogy megjelenjenek, így biztosítva, hogy a szabad tartalmakból készült bármilyen származékos mű is szabad maradjon.

A GPL legnagyobb programbázisa jelenleg a Linux rendszermag és a szabad Unix-szerű rendszerek segédprogramjai. Jelentős mérföldkő volt a nyílt forráskód történetében a Java Platform GPL licenc alatti megnyitása 2007-ben.”

<http://hu.wikipedia.org/wiki/GNU_General_Public_License>

Bootstrap licensz:

MŐT

A MŐT is szabad licensz ezt is szabad módosítani, pénzért árulni, továbbterjeszteni, a szerző nem felelős a szoftver használatából fakadó károkért.

Összefoglalás

Úgy érzem hogy a Piactér szakdolgozatnak egy jó választás volt, sikerült fejlődnöm, egy olyan oldalat készítettem amit szívesen mutathatok meg az ismerőseimnek.

Fejezetek:

* Bevezetés (1)
* Eredeti specifikáció (1.1)
* Témaválasztás (1.2)
* Őroalomjegyzék (1.3)
* Halsznált eszközök/technológiák
* programnyelv
* fejlesztőkörnyezet
* Az elkészült program
* Megvalósult dolgok
* Megvalósulatlan funkciók
* Plussz funkciók
* Felhasználói dokumentáció
* Tesztek, problémák, megoldások
* Összefoglalás, konklúzió(vége)