SZAKDOLGOZAT

Tóth Tamás

Debrecen

2010

Debreceni Egyetem
Informatikai Kar

ONLINE FOGADÓJÁTÉK MEGVALÓSÍTÁSA INGYENES WEBES ESZKÖZÖKKEL

Témavezető:
Pánovics János
egyetemi tanársegéd

Készítette:
Tóth Tamás
programtervező
informatikus BSc

Debrecen
2010

Tartalomjegyzék

1. Bevezetés

Az internet a számítógép hálózatokat összekapcsoló globális rendszer, amely a szabványos TCP/IP internet protokoll révén ma a felhasználók milliárdjait köti össze. A világháló angolul World Wide Web vagy egyszerűen csak web, az interneten működő, egymással hiperlinkekkel összekötött dokumentumok rendszerét jelenti, melyet webböngésző programok segítségével lehet elérni.
A web alapelveit 1989-ben Tim Berners-Lee dolgozta ki. A CERN részecskefizikai kutatóközpont munkatársának elsődleges célja a világ több pontján elhelyezkedő kutatóintézetek közötti kutatási eredmények megosztása volt. A CERN 1993. április 30-án a világhálót mindenki számára szabadon elérhetővé tette, ezzel elindítva a Web gyors iramú fejlődését.
1994 októberében Tim Berners-Lee megalapította a World Wide Web Konzorciumot (angolul: World Wide Web Consortium, rövidítve: W3C), melynek célja, hogy minél jobban kihasználja az internet (web) nyújtotta lehetőségeket szabványok és iránymutatások kidolgozása révén, melyek egyúttal biztosítják a web hosszú távú fejlődését is.

Az internet felhasználók száma évről évre növekedett, ami újabb és újabb igényeket támasztott a webhasználattal szemben. Kezdetben a csak szövegeket tartalmazó statikus weboldalaktól eljutottunk a dinamikus webes alkalmazásokig, melyek megvalósításához egyre fejlettebb webes technológiák fejlesztésére volt szükség. Mára már asztali alkalmazások funkcióinak megfelelő weboldalak létrehozása lehetséges.
A web alkalmazások népszerűségének oka, hogy az őket használó webböngésző kliensek szinte minden gépen rendelkezésre állnak és karbantarthatóak a kliens gépek szoftverének változtatása nélkül. Ezzel szemben az asztali alkalmazásokat telepíteni kell és nem függetlenek a különböző platformoktól.
Az internet felhasználók száma a világon 2009. szeptemberére elérte az 1.73 milliárdot.

1.1. Témaválasztásom

A bevezetőben elhangzottak alapján, bátran kijelenthetjük, hogy a webes alkalmazások elképesztő népszerűséggel bírnak a felhasználók körében. Aki egyszer is kipróbálta az internetet, meggyőződhetett róla, mekkora lehetőségeket rejt magában. Az egyik legjobb, legtöbb lehetőséget nyújtó találmány hosszú évtizedek óta. Azt, hogy mekkora jelentőségre tett szert mindennapjainkban, mennyire hasznos és sokoldalú eszközről van szó, felesleges hosszasan tárgyalni.

A technika fejlődése, és az egyre csökkenő tarifák segítenek abban, hogy minden háztartásba eljuthasson. A gyerekek lassan billentyűzettel a kezükben fognak felnőni, és az idősebb korosztály is lassacskán felfedezi az internet nyújtotta lehetőségeket.

A felhasználók nemcsak személyes, illetve üzleti célokra, például információkeresésre használhatják, hanem szolgáltatások és áruk megrendelésére, kapcsolattartásra, szórakozásra, stb. A távlatok, mondhatni beláthatatlanok.

Az internet rohamos elterjedésének szülötte az online sportfogadás is, hisz a világ bármely pontjáról biztonságos körülmények között lehet sporteseményekre fogadni, válogatva számos nemzetközi fogadóiroda közül.

A gomba módjára elszaporodott, újabb és újabb internetes fogadó portálok a média segítségével is igyekeznek behálózni bennünket. Reklámjaik visszaköszönnek a televízióból, rádióból, újságokból, weboldalakról, sőt még a labdarúgók mezeiről is.

Amit a különféle sporteseményekre való tippelés szolgáltathat, az lehet egyfajta hobby, ugyanakkor komoly pénzkeresési lehetőség is, vagy csupán az izgalom és a szórakozás egyik megfelelője. Mindegyik alapja tehát a pénz, mely hiányában egyik sem valósulhat meg.

Ezért is választottam szakdolgozatom témájául egy olyan fogadó játék megvalósítását, mely teljesen független az anyagiaktól. A kivitelezéséhez és a használatához se legyen szükség pénzre. Egy olyan interaktív webes alkalmazás jöjjön létre, mely a felhasználó számára egyszerűen kezelhető, látványos, és nem utolsósorban szórakoztató is.

A weboldalamnak a BetGame.hu nevet adtam, ezzel is utalva arra, hogy fogadást valósít meg, de csakis játékos formában. A webalkalmazásomat Firefox böngészőre

optimalizáltam, és egy Apache HTTP szerveren hoztam létre, melyen PHP5 fut, egy MySQL adatbázis-kezelő szerverrel. A HTML leíró és CSS stílusleíró nyelven kívül használtam még a JavaScript és az AJAX programozási nyelveket a weboldal dinamikusabbá tételére. A webalkalmazás használ XML dokumentumokat is, melyben az adatok elérését XPath-al valósítottam meg.

A szakdolgozatomban szeretném ismertetni a felhasznált technológiákat és programokat, valamint a fejlesztés menetét is. Az implementáció folyamatából a fontosabb részek megvalósítását mutatom be példakódokkal, és a különböző technológiák együttműködését. A webalkalmazás teljes kódja a szakdolgozathoz mellékelt DVD-n megtalálható. A ***www.betgame.hu/index.php*** oldalon pedig megtekinthető és használható.

2. Alkalmazásom fejlesztő eszközei

A szakdolgozatom címének megfelelően, a weboldalam létrehozásához kivétel nélkül ingyenes eszközöket használtam. Az alábbiakban ezen nyelveket és szoftvereket szeretném bemutatni és jelentőségüket kiemelni.

2.1. HTML/XHTML

A HTML (**H**yper**T**ext **M**arkup **L**anguage) egy leíró nyelv, melyet kifejezetten weboldalak készítéséhez fejlesztettek ki, és mára már internetes szabvánnyá vált a W3C támogatásával.

HTML 1.0-ás verzióját 1990-ben specifikálták. Itt jött létre az alapvető szerkezet, a nyitó és záró címkék „<html></html>”, a fejléc „<head></head>” és a törzs "<body></body>", amiben a tartalom helyezhető el. Már ekkor lehetőség volt a kezdőcímke előtt dokumentum típus deklaráció (DTD)[1] elhelyezésére, ami mára különösen fontossá vált a HTML dokumentumok kiértékelése és érvényessége szempontjából.

Az első, a World Wide Web Konzorcium által kiadott változat, a HTML 2.0. Bevezetésre kerültek az űrlapok, és az ezen belüli többsoros szövegbevitel, és a kiválasztható opciók.

A HTML 3.2 verziót 1996-ban fogadta el a W3C, ami jelentős változásokat hozott. Innentől van lehetőségünk Java appletek beágyazására, és scriptek használatára, táblázat készítésére, a betűtípus beállítására.

A HTML 4.0 verziót 1997 nyarán tette hivatalos ajánlássá a W3C, ami megfelel az ISO 8879[2] előírásainak. Ez már igazából egy SGML[3] alkalmazás. Tovább fejlesztették az űrlapok és táblázatok használhatóságát, a keretek (frame-ek) használata hivatalossá vált. Gyarapodtak a formázóelemek, és elvetettek pár korábbi címkét. Megjelenik a

[1] **Document Type Definition (DTD):** Típusdefiníció, amely leírja, hogy az adott dokumentumtípushoz tartozó dokumentumok milyen elemeket, attribútumokat, értékeket és hivatkozásokat tartalmazhatnak. Itt kell megadnunk a használatos karakterkészleteket, és azt, hogy az egyes elemek hogyan helyezkedhetnek el a dokumentumban. A DTD-vel adjuk meg a dokumentum hierarchikus szerkezetét

[2] **ISO 8879 szabvány:** Az SGML-t meghatározó, leíró szabvány. Az információ szerkezetének megadására szolgáló metanyelvet és az SGML dokumentumok tárolási formátumának leírását definiálja.

[3] **SGML (Standard Generalized Markup Language):** szabványos általános jelölőnyelv, ISO 8879 szabványa által meghatározott jelölőnyelv dokumentumformátumok leírására.

nemzetközi karakterkészletek nagyobb mértékű támogatása is, de a HTML 4.01-es javított verzióban éri el mai szintjét.

A HTML eddigi fejlődése mind azt célozta, hogy minél összetettebb szabványos dokumentum szerkezeteket hozhassunk létre. Azonban kliens oldalon elindult a böngészők közötti verseny a felhasználókért, mely egészen odáig vezetett, hogy bizonyos gyártók eltérve a HTML 4.01 ajánlásától saját tageket (címkék angol megfelelője, amit a magyar szaknyelv is átvett) és formázó elemeket vezettek be. Válasz lépésként a W3C létrehozott egy robusztus és letisztult HTML formátumot, ez a XHTML (XML alapú HTML), mely lényegében egy letisztított 4.01-es verzió XML (e**X**tensible **M**arkup **L**anguage – lásd 2.2. fejezet) adottságokkal integrálva. Így kompatibilis minden XML és HTML alapú böngészővel. A fejlesztők célja a tartalom, és a megjelenítés különválasztása volt, ahol a dokumentumok struktúrájáért az XHTML, míg a megjelenés vezérlésért a stíluslapok (CSS , lásd 2.3. fejezet) lettek felelősek.

A W3C szervezet nagy erőkkel kezdett az XHTML 2.0-ás verziójának fejlesztésébe a magasabb szintű XML integráltság elérése érdekében. 2009 júliusában befejezték a fejlesztéseket, arra hivatkozva, hogy a főleg mobiltelefonok által használt XML alapú böngészők, a vártnál sokkal kisebb számban terjedtek el. Ezért átcsoportosították erőiket a közben már elkezdett HTML5 fejlesztésére, ami napjainkban is tart. Egyik tervezési céljuk, hogy a webes alkalmazásokhoz ne legyen szükség plugin-ek[4] telepítésére (pl.: Adobe Flash, Microsoft Silverlight, Sun JavaFX). Noha léteznek már stabil részei, melyek ma is használhatóak, de a W3C előzetes javaslattervére még 2012-ig várnunk kell. Ian Hisckson, a specifikáció szerkesztője szerint, a végleges ajánlás 2022-re várható. Ez elég távolinak tűnik, viszont Hisckson véleménye szerint, a HTML 4.01-es verziója tíz év fejlődés után sem érte el azt a szintet, amit a HTML5-el el akarnak érni. Így már reálisabbnak tűnik a 2004-ben kezdett fejlesztés elhúzódása 2022-ig.

2.2. XML

Az XML megnevezés, az E**X**tensible **M**arkup **L**anguage (kiterjeszthető jelölőnyelv) kifejezés rövidítése. A W3C, a korábbi jelölőnyelvek fejlesztése során szerzett tapasztalataira alapozva hozta létre.

[4] **Plugin:** böngészőkbe opcionálisan beépíthető, annak képességeit bővítő vagy módosító kiegészítő modul.

Az XML alkalmazási lehetőségei szinte korlátlanok. Két fő felhasználási területe van, az adatcsere-formátum és dokumentumszerkesztési formátum megadása. E két alkalmazási mód gyakran kiegészíti egymást, mivel az adatok emberek által történő felhasználásra is formázhatók.

Az XML adatformátum első ránézésre túlságosan leegyszerűsítettnek tűnhet, azonban számos megszorítása létezik. Ahhoz, hogy egy XML dokumentum helyes legyen, a következő követelményeknek kell megfelelnie:
Helyesen formázottság (well-formed): egy helyesen formázott XML dokumentum, megfelel minden XML szintaxis szabálynak. Ha egy XML dokumentumban megsértjük ezen szabályok valamelyikét is, az elemzőnek meg kell tagadnia a feldolgozását.
Érvényesség (valid): egy érvényes XML dokumentum olyan adatokat tárol, ami megfelel a felhasználó által definiált tartalmi szabályoknak. Ezek a szabályok DTD-vel, vagy az újabb keletű XML Schema[5]-val adhatók meg, így határozva meg a dokumentum struktúrát az érvényes elemek listájával.

Mint mindennek, az XML-nek is megvannak az előnyei és hátrányai. Néhány szó az előnyeiről:
Az XML birtokában van azon tulajdonságoknak, amelyek alkalmassá teszik adattovábbításra mind ember, mind gép számára olvasható formátumban. Támogatja az Unicode[6] karakterkészletet, ami lehetővé teszi bármely információ, bármely emberi nyelven történő közlését, és képes a legtöbb általános számítástudományi adatstruktúra ábrázolására (rekord, lista, fa…). Öndokumentáló formátum, amely struktúra- és mezőneveket ír le speciális értékeikkel együtt. Szigorú szintaktikus és elemzési követelményeket támaszt, ami biztosítja, hogy a szükséges elemzési algoritmus egyszerű, hatékony és ellentmondásmentes maradjon.

Dokumentumtárolási és feldolgozási formátumként – mind online mind offline módban – több előnnyel jár, mivel egyszerű szöveg formátumban valósul meg, így licencektől és

[5] **XML Schema:** egy újabb keletű XML séma nyelv, amit a W3C a DTD utódaként definiáltak. Az XML Schema lényegesen többre képes a DTD-nél az XML nyelvek leírása terén. Sokoldalú adattípus rendszert használ, ami részletesebb megkötéseket tesz lehetővé az XML dokumentum logikai szintjén, de ezért sokkal robusztusabb érvényesítő keretrendszert követel meg.
[6] **Unicode:** nemzetközi karakterek ábrázolását megvalósító karakterkódolás

korlátozásoktól mentes, valamint platform-független is, ezáltal viszonylag immúnis a technológiai változásokkal szemben. internetes szabványokon alapuló, erőteljes, logikailag ellenőrizhető formátum, a hierarchikus struktúrája megfelel a legtöbb (de nem mindegyik) dokumentum típusnak.

Bizonyos alkalmazások szempontjából hátrányokkal is rendelkezik. A szintaxisa elég bőbeszédű és részben redundáns, ez nehezítheti az emberi olvashatóságot és az alkalmazások hatékonyságát. Nagyobb tárolási költséggel jár, ami nehézzé teszi az XML alkalmazását korlátozott sávszélesség esetén. Ez főleg a telefonokon és a PDA-kon futó multimédiás alkalmazásoknak okozhat problémát, mivel XML-t szeretnének használni képek és videók leírására. Bizonyos esetekben a tömörítés megoldást jelenthet.
Külön erőfeszítést igényelhet az egymást részben átfedő (nem hierarchikus) adatstruktúrák modellezése, valamint az XML relációs, és objektumorientált paradigmához kötése is.

2.3. XPath

Az XPath olyan W3C ajánlás, amely egy, kifejezéseket használó nyelvet definiál. Elsősorban XML dokumentumokban való keresésre, kapcsolódások kialakítására, és különböző elemtípusokra alkalmazható környezetfüggő formázási lehetőségek megvalósítására (navigációra) használható. Az ajánlás definiálja a szövegsztringek szintaxisát, amiket kifejezésnek nevezünk, és amely objektumok például XML dokumentumon belüli elemek kijelölésére szolgáló utasításokból állnak.

2.4. CSS

A **CSS** (**C**ascading **S**tyle **S**heets) egy stílusleíró nyelv, mellyel HTML vagy XHTML nyelven írt strukturált dokumentumok megjelenését állíthatjuk be. Tehát a struktúra és a megjelenés elkülönül, aminek több haszna is van. Egyrészt növeli a weblapok használhatóságát, rugalmasságát, és a megjelenés kezelhetőségét, másrészt csökkenti a dokumentum tartalmi struktúrájának komplexitását. A CSS ugyancsak alkalmas arra, hogy a dokumentum stílusát a megjelenítési módszer függvényében adja meg, így

elkülöníthető a dokumentum formája a képernyőn, nyomtatási lapon vagy éppen egy hangos böngészőben[7].

Nyelvtana egyszerű, csupán néhány angol nyelvű kulcsszót használ a stílusok tulajdonságainak leírására. A *stíluslap* maga a stílust leíró szabályok sora. Minden szabályhoz tartozik egy *szelektor* és egy *deklarációs szakasz*. Ez utóbbi kapcsos zárójelek között pontosvesszővel elválasztott *deklarációkat* tartalmaz. A deklarációk formája a következő: a *tulajdonság neve*, egy kettőspont, majd az adott *tulajdonság értéke*. Az 1. ábrán látható példakód, a <warning> címkének a stílusát állítja be, így a tartalmazott szöveg mérete 15 pixel nagyságú, és piros színű lesz.

```
9   warning{
10      font-size:15px;
11      color:#FF0000;
12  }
```

1. ábra.

A CSS információkat a HTML/XHTML dokumentumokhoz több módon lehet megadni.
Kliens stílus:

- a kliens vagy a böngésző által meghatározott, alapértelmezett stílus.

Felhasználói stílus:

- a felhasználó egy helyi, saját CSS fájlt adhat meg a böngésző segítségével az összes dokumentumra. Beállítható, hogy a szerző és a saját felhasználói stílus közül melyik élvezzen nagyobb prioritást.

Szerzői stílus:

- dokumentumból hivatkozható a külső CSS fájl vagy fájlok,
- beágyazhatók a dokumentum szövegébe,
- felülírható az általános stílust egy konkrét esetre.

A CSS fontos tulajdonsága még, amit a nevében szerepelő cascading szó is jelez, hogy *rangsorolt* stíluslapokat definiál. Ez a rangsor szó a stílusok közötti esetleges ütközések

[7] **Hangos böngésző:** képes a weboldalak felolvasására, illetve a látássérültek számára speciálisan megformázott szövegeken könnyű navigációt tesz lehetővé.

feloldásának módjára vonatkozik, melyet az *öröklés* és a *szűkítés* alapelvek határoznak meg pontosan.
Az *öröklés* azon a tényen alapszik, hogy a HTML/XHTML dokumentumban az elemek egymásba ágyazódnak, a köztük lévő viszony hasonló ahhoz, mint amilyen a szülő és gyermek, illetve az ős és a leszármazott között van.
A *szűkítés* lehetővé teszi, hogy a stílusoknak rangja legyen, mégpedig egy adott stílus pontossága alapján. A szűkebb (pontosabb) stílusok súlya nagyobb, mint a tágabbaké, ezért a böngésző azokat fogja használni a formázáshoz.
Az általam felsorolt stílus megadási módok, fentről lefelé haladva egyre szűkebbek, azaz súlyuk egyre nagyobb a megjelenítendő külső szempontjából.

A CSS specifikációját szintén a W3C felügyeli.

2.5. PHP

Születésekor (1995) csupán egy makrókészlet létezett, amely személyes honlapok karbantartására készült. Innen ered a neve is: **P**ersonal **H**ome **P**age Tools. Később a PHP képességei kibővültek, így egy önállóan használható programozási nyelv alakult ki, amely képes nagyméretű webes adatbázis-alapú alkalmazások működtetésére is. A PHP ma már a **P**HP: **H**ypertext **P**reprocessor hivatalos elnevezést használja.

Ez egy olyan kiszolgálóoldali programozási nyelv, amit jellemzően HTML oldalakon használnak. A hagyományos HTML lapokkal ellentétben, a kiszolgáló a PHP parancsokat nem küldi el az ügyfélnek, hanem azokat a kiszolgáló oldalán a PHP-értelmező dolgozza fel. A webes alkalmazásokban lévő HTML elemek érintetlenül maradnak, de a PHP kódok lefutnak, ezáltal a kódok kimenetele a megadott HTML elemekkel együtt kerül az ügyfélhez. A PHP kódok végezhetnek adatbázis-lekérdezéseket, dinamikusan létrehozhatnak képeket, fájlokat olvashatnak és írhatnak, kapcsolatot létesíthetnek távoli kiszolgálókkal, a lehetőségek száma végtelen.

A szakdolgozatom írásakor a legfrissebb stabil verzió, a PHP 5.3.2, amely 2010 március 04-én jelent meg.

2.6. JavaScript

A JavaScript, mint nevéből is kitűnik script nyelv, viszont a Java szó megtévesztő lehet, hisz nincs sok közös bennük. A Java önálló objektumorientált programozási nyelv, melynek szüksége van fordító programra, amivel a forráskódból bájtkódot készíthetünk. A JavaScript pedig egy parancsnyelv, futtatásához nem kell más, csak egy böngésző, amely kezelni képes a JavaScript-et. Hasonlóságot talán a két nyelv szintaxisában fedezhetünk fel, mivel mindkettő a C nyelv szintaxisára hasonlít. Egyes szélsőséges vélemények szerint a két nyelv csupán a nevük első négy betűjében egyezik meg.

A JavaScript-et és elődjét, a LiveScript-et egyaránt a Netscape fejlesztette ki saját böngészőjéhez. Elsőként a Netscape 2.0-ban jelent meg, célja az volt, hogy a weboldalakat és a böngészőket dinamikussá tegye. Mivel a JavaScript kliens oldalon fut, ezért az oldal teljes frissítése nélkül képesek vagyunk olyan feladatokat elvégezni, amelyeket előre tudunk definiálni és így a szervert nem terheljük. Ilyen feladat lehet: az űrlapok mezőinek tartalmi ellenőrzése, felhasználók azonnali figyelmeztetése hibák esetén, és az oldal elemeinek dinamikus változtatása. Mára persze minden elterjedt böngésző tudja értelmezni a JavaScript-eket.

A legnagyobb hátránya, hogy mivel interpretált nyelv, ezért az olykor nehézkesen megírt JavaScript függvényeinket bárki kiszedheti a forráskódunkból, és beteheti a sajátjába. Másik hátulütője (főleg régebben), hogy a felhasználók eleve letiltották a JavaScript futtatási lehetőséget a böngészőben. Ugyancsak hátrányként említhető, hogy a hibakeresés nehézkes lehet.

2.7. AJAX

Az AJAX mozaikszó, az **A**synchronous **J**avaScript **a**nd **X**ML (aszinkron JavaScript és XML) kifejezésből ered. Központi eleme az XML DOM (Document Object Model, Dokumentum-Objektummodell) részét képező XMLHTTPRequest (XHR) objektum, amely az AJAX legfőbb lényegét és erőségét megvalósító eszköz. Segítségével lehetővé válik, hogy a kliens oldaláról HTTP-kérelmeket küldjünk a szerverhez, a böngésző frissítése nélkül.

Az XML dokumentum-objektummodell, az XML dokumentumok elérésének és módosításának szabványos módját határozza meg. A DOM hozzáférést nyújt az XML, illetve az XHTML dokumentumok szerkezetét alkotó elemekhez, ezáltal teljes elérést nyújt a JavaScript számára. A hozzáférést a JavaScript DOM-kezeléssel foglalkozó belső objektumhalmaza teszi lehetővé.

Az AJAX olyan nyelvek hatékony gyűjteménye, amelyek együtt könnyen használható felületeket és ügyféloldali párbeszédet eredményeznek, mindezt aszinkron működéssel, vagyis a böngésző aközben is használható, míg az elküldött kérésre válasz érkezik. Így lehetővé teszi, hogy adatcserét folytassunk a kiszolgálóval, HTTP-állapotkódokat fogadjunk, adatbázisba mentsünk, illetve meghatározzuk, hogy mi jelenjen meg a felhasználó számára, anélkül, hogy egyszer is frissíteni kellene az oldalt. Az asztali alkalmazásokhoz hasonlóan, ez a kérelem-válasz körforgás folyamatosan fennállhat, az AJAX-szal együttműködő webalkalmazások azonban a Weben találhatók, tehát bárki, aki internetkapcsolattal rendelkezik, elérheti azokat.

2.8. MySQL

A MySQL egy többfelhasználós, többszálú, SQL-alapú relációs adatbázis-kezelő szerver.
A többszálú kifejezés azt jelenti, hogy minden egyes új kapcsolat létesítésekor elindul egy új kiszolgálófolyamat. A MySQL-hez kötődő kapcsolatok nem osztják meg egymással a folyamataikat, így amikor egy folyamat valamilyen hiba következtében leáll, vagy valamilyen módon túlterheli a kiszolgálót, csak az adott folyamat áll le, és nem az egész kiszolgáló. Ez a tulajdonság a MySQL sebességére is kedvező hatással van.
Amellett, hogy tárolja tábláinkat, oszlopainkat, sorainkat (és a bennük lévő adatainkat), egységes egészként kezeli őket, valamint lehetővé teszi a felhasználók és a hozzájuk tartozó jogosultságok megadását, naplózza az egyes felhasználók tevékenységeit, és az adott válaszokat lekérdezésekbe szervezi.
A MySQL-t az is népszerűvé teszi a fejlesztők körében, hogy gyakorlatilag minden nyelv – PHP, Perl, C, C++, illetve a Java és a Python – alkalmas arra, hogy felhasználói felületet írjanak hozzá.

Az operációs rendszer lecserélésével járó gondok is egyszerűen áthidalhatóak: ha bármikor operációs rendszert vagy nyelvet kell váltanunk, egy paranccsal (*mysqldump*) kinyerhetjük adatainkat, és egy másik parancs segítségével (*mysqlimport*) visszavihetjük őket a rendszerbe.
Nemhiába vált a világ egyik legelterjedtebb nyílt forrású adatbázis-kezelőjévé. Több millió felhasználója van, az egyéni fejlesztőktől az olyan nagyvállalatokig, amelyek nagy forgalmú oldalak adatbázismotorjaként alkalmazzák.

2.9. Apache

Az Apache HTTP Server nyílt forráskódú webkiszolgáló alkalmazás, egy szabad szoftver, mely kulcsfontosságú szerepet játszott a World Wide Web elterjedésében. A projekt célja egy, olyan webszerver program létrehozása, karbantartása és fejlesztése, amely megfelel a gyorsan változó internet követelményeinek, biztonságos, üzleti, vállalati felhasználásra is megfelelő és szabadon használható.
Az Apache egy robosztus, erőteljes és rugalmas webszerver, amely kompatibilis a HTTP/1.1 (RFC2616) protokollal.
Az Apache volt az első használható alternatíva a Netscape Communications Corporation webszerverrel szemben (mai neve: Sun Java System Web Server). A későbbiekben továbbfejlődött és más unix alapú webszerverekkel is felvette a versenyt, funkcionalitás és teljesítmény tekintetében.
Elsősorban a következő operációs rendszerekhez készítették el az Apache-ot: Unix, FreeBSD, Linux, Solaris, Novell NetWare, Mac OS X és Microsoft Windows.
Az említett webszerver sok olyan szabványt támogat, melyeknek nagy része lefordított modulok formájában áll rendelkezésre, a mag kiegészítéseként. Ezek a modulok sok területet lefednek, a kiszolgálóoldali programnyelv-támogatástól kezdve a hitelesítési sémákig. Az ismertebb, támogatott programnyelv modulok: *mod_perl*, *mod_python* és a PHP.

A PHP, MySQL, Apache együttes alkalmazása, jól használható dinamikus weblapok előállítását teszi lehetővé.

2.10. Fejlesztést segítő ingyenes programok

A folytatásban azon programokról szeretnék írni, melyek nagyban segítették munkámat a fejlesztés során. Ezen programok kiválasztásánál is törekedtem arra, hogy ingyenes, szabad szoftvereket használjak.

2.10.1. WampServer

A WampServer lényegében egy webfejlesztő-keretrendszer, Windows operációs rendszerhez. Tartalmazza a PHP, Apache, MySQL rendszereket, mindezeket egy intuitív csomagba ágyazva, amely segítségével nagyon egyszerűen tudunk helyi gépen reális tárhelyet létrehozni.

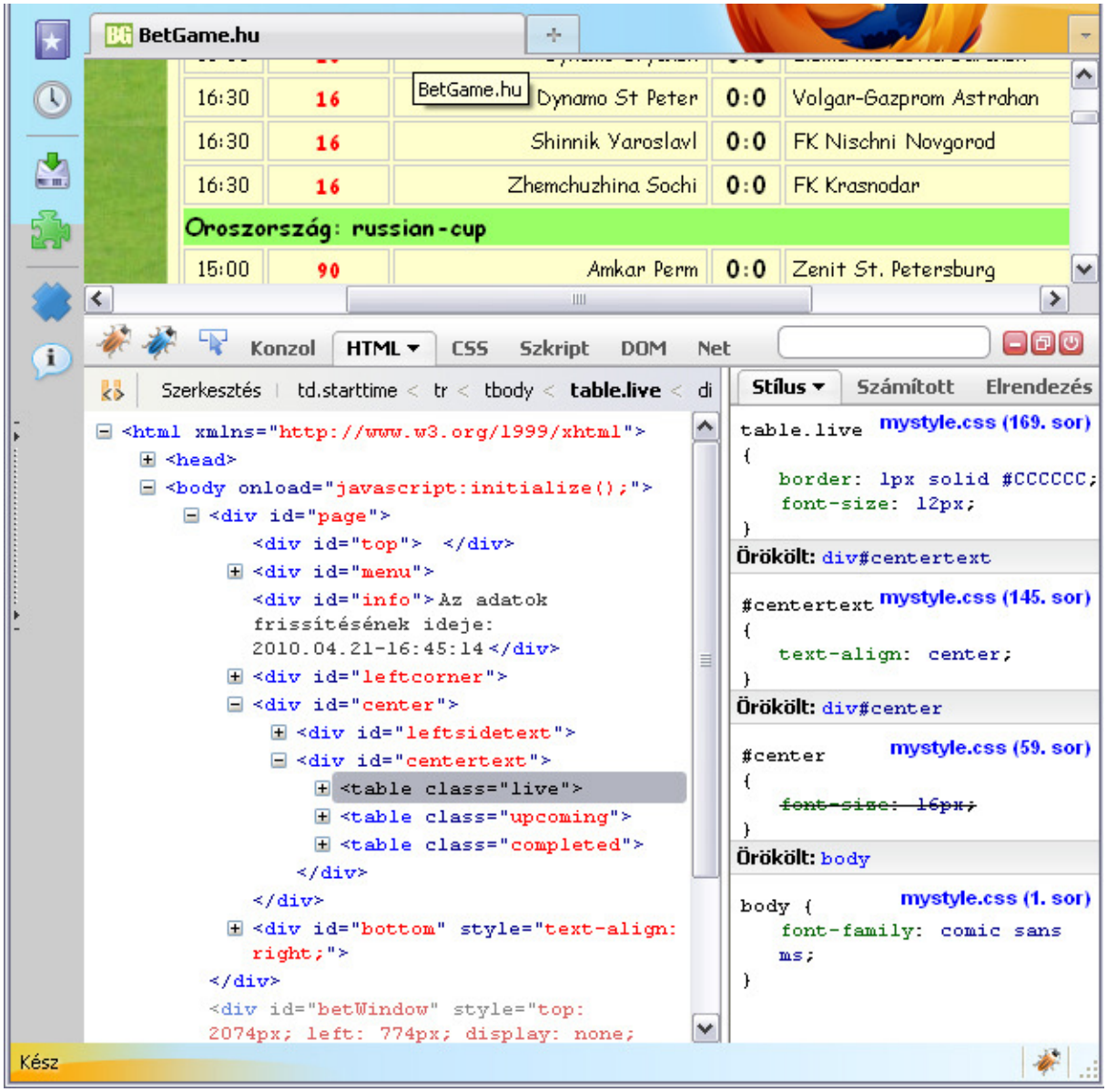
A program indítását követően, mind a PHP, az Apache és a MySQL használni kívánt verziói kiválaszthatóak, és a beállításaik könnyen engedélyezhetőek vagy letilthatóak. Az előre beépített PHP kiterjesztéseket és az Apache modulokat is beállíthatjuk.

A szoftver nagy segítséget nyújt abban, ha különböző okok miatt nem tudjuk elérni a tárhelyünket az interneten, vagy előbb saját gépen akarunk fejleszteni, és csak a stabil verziót szeretnénk feltölteni a webszerverünkre.

2.10.2. FireBug

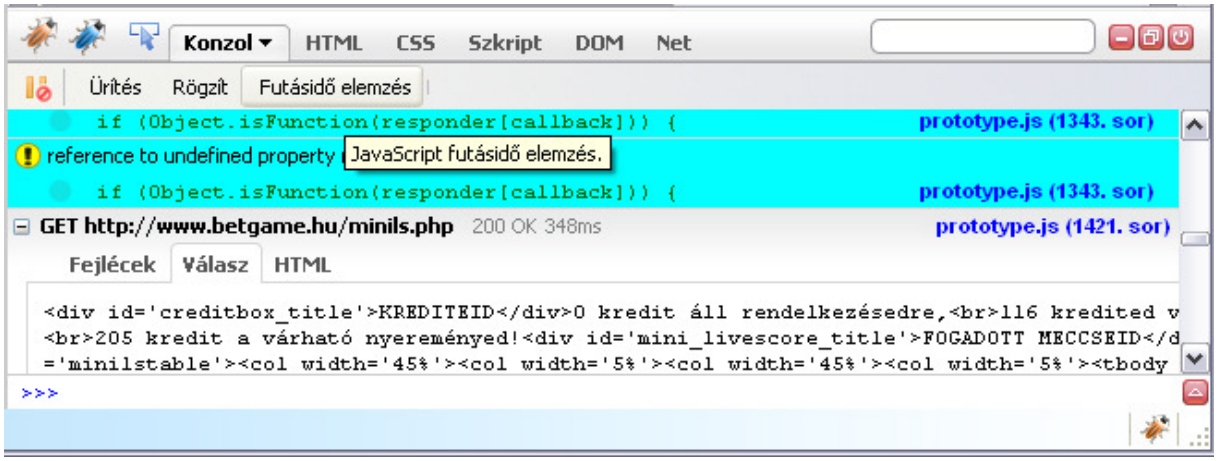
Webalkalmazásom fejlesztése során, a legtöbb segítséget nyújtó szoftver a FireBug volt. Ez a program csak Firefox böngészőhöz telepíthető kiegészítés, melynek segítségével élőben lehet szerkeszteni, hibakeresést végezni (debugolni) és monitorozni, bármely webes oldal CSS, HTML és JavaScript kódját.

A szerkesztés úgy értendő, hogy szabadon megváltoztatható a böngészőnkben megjelenő kódok bármelyike (a változtatás eredménye rögtön látható a böngészőben), de nincs hatással a tárhelyen elhelyezkedő tényleges kódra. Gyakorlati haszna mégis hatalmas. Az oldalam megfelelő külső megjelenésének kialakításában nyújtotta a legtöbb segítséget. Ha nem voltam elégedett egy elem stílusával, akkor szabadon próbálkozhattam a FireBug segítségével, majd elég volt a CSS fájlomat egyszer módosítani a elvégzett változtatásoknak megfelelően. Mindemellett kiválóan szemlélteti az elemek stílusát meghatározó tulajdonságok öröklődését.



2. ábra. *A FireBug HTML nézete. Baloldalon látható a weblapom HTML felépítése, míg jobboldalt a „live” osztályú táblám CSS által meghatározott stílusa. Jól látszik (lentről felfele), hogy sorban a legkülső* elemtől *a legbelsőig, milyen tulajdonságokat örököl. A szűkítésre is van egy példa, mégpedig a „center” nevű* *elemem „font-size: 16px;” (betű-méret: 16 pixel) tulajdonsága helyett, a „live” osztály szűkebb „font-size: 12px;” tulajdonsága érvényesül a stílus meghatározásában.*

A FireBug másik fontos eszköze a konzol. Itt jelenik meg a JavaScript által küldött összes figyelmeztetés (warning), hiba (error), és a felhasználó ide írathat ki számára fontos információkat a működés során. Az AJAX által küldött kérések, azok állapota, a válaszidő, és a válasz eredménye is könnyen követhető általa.



3. ábra. *A FireBug-konzol működés közben. Egy figyelmeztetés látható a képen, és az AJAX által küldött kérésre kapott válasz eredménye.*

2.10.3. Notepad++

A Notepad++ egy Microsoft Windows platformra írt szabad, szövegszerkesztő szoftver. Kódszínezése miatt, forráskódszerkesztésre is kiválóan alkalmas, mely számos programozási nyelv szintaxisát támogatja. Az általam használt PHP, JavaScript, HTML, XML, CSS nyelveken kívül még a következő nyelvekét is: C, C++, Java, C#, makefile, ASCII art, doxygen, ASP, Visual Basic/VBScript, Shell script, BAT, SQL, Objective-C, Pascal, Perl, Python, Lua, TCL, Assembly, Ruby, Lisp, Scheme, Diff, Smalltalk, Postscript, VHDL.

2.10.4. Hornil StylePix képszerkesztő program

Ingyenes, profi felhasználói felülettel rendelkező, gyors képmanipulációs eszköz a Hornil StylePix, melyet kifejezetten Windows operációs rendszerre terveztek. A szoftvert kimondottan olyan személettel készítették, hogy a grafikában kevésbé jártas emberek is gyors sikerélményeket érhessenek el benne.

Minden fontos képszerkesztő eszközt tartalmaz: vágás, forgatás, tükrözés, átméretezés, szűrők, szöveg elhelyezés a képen. Rétegeket is kezel, és a képfájlokat minden elterjedt formátumban menthetjük (jpg, gif, png, bmp, tif, tga).

3. Webes alkalmazásom létrehozása

Legyen szó bármilyen szoftverről vagy alkalmazásról, egész életútja alatt - az ötlettől a megvalósulásig, majd a későbbi esetleges üzemen kívül helyezésig - nagyon sok fázison kell áthaladnia. Ezen állomások a következők:

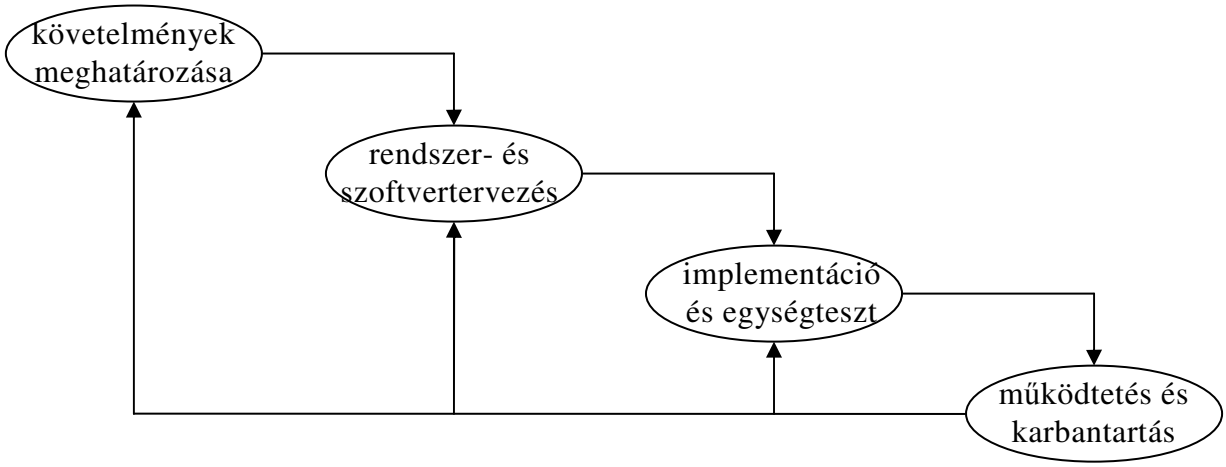
- Vízió
- Követelmények feltárása
- Elemzés
- Architekturális tervezés
- Tervezés
- Implementálás
- Tesztelés
- Üzembe helyezés
- Üzemeltetés
- Karbantartás
- Evolúció
- Üzemen kívül helyezés.

A szoftverfejlesztési életciklus lényege, hogy egy absztrakt megközelítéstől egy teljesen konkrét megközelítés felé megyünk el valamilyen módon, valamilyen alkalmazott tevékenységsorozat alapján. Arra a kérdésre, hogy milyen módon vagy tevékenység sorozat mentén haladjunk, a különböző folyamatmodellek adhatják meg a választ.
Minden folyamatmodell más-más szemléletben tekint a fejlesztés folyamatára.
A folyamatmodellek az elmúlt negyven évben fokozatosan alakultak ki, és mindegyiknek megvannak az előnyei és hátrányai, az alkalmazási területei, mikor célszerű és mikor nem célszerű használni.
Az első modellek más problémák mellett alakultak ki, mint a közelmúlt modelljei. Régen kisebb, egyszerűbb, jól meghatározott és kevesebb kihívást igénylő szoftvereket fejlesztettek, mint ma. A jelenben egyre nagyobb, komplexebb szoftverekről beszélhetünk, amin egyszerre több ember dolgozik. Ehhez kapcsolódik még a költség, idő és minőség hármasa, melyek jó projektmenedzseri munkát feltételeznek, és nem beszélve

arról, hogy a megrendelők által támasztott követelmények állandó változására is fel kell készülni.

Persze ez nem azt jelenti, hogy a korábbi folyamatmodellek elavultak lennének, hisz a folyamatmodellek mindegyike ma is él. El kell tudni dönteni, hogy egy adott szoftver fejlesztésénél melyik modellt követjük, beleértve azt is, hogyan keverjük őket. Az adott lépésben, az adott részfolyamatban, az oda legjobban megfelelő folyamatmodell alkalmazandó.

A webes alkalmazásom fejlesztésének első két lépéséhez – követelmények meghatározása és szoftvertervezés - a vízesésmodellt választottam. Ez az első folyamatmodell, mely a történelem során kialakul, 1970-ben definiálták. Nevét onnan kapta, hogy a folyamatelemek meglehetősen mereven egymásra épülnek, lépcsőzetesen kapcsolódnak egymáshoz, mint ahogy a vízesés folyik le a sziklán.



4. ábra. *A vízesésmodell folyamatábrája.*

Hátránya az, mereven egymásra épülnek a folyamat fázisai. Minden lépés teljeskörűen lezárul, mielőtt elindul a következő. Ha bármelyik lépést elrontottuk, akkor az összes rákövetkezőt újra el kell végeznünk.

Előnye, kisméretű és jól meghatározható követelményekkel rendelkező rendszereknél jelentkezik. Ezen két tulajdonság az én alkalmazásomra is igaz, mivel a weboldallal szemben támasztott követelményeket előre, pontosan le tudom írni, alkalmazásom pedig kicsi. Ezért választottam a vízesésmodellt a követelmények meghatározásához és szoftvertervezéshez.

3.1. Követelmények meghatározása

Első és nagyon fontos lépés a követelményeink meghatározása. Itt fogalmazzuk meg, hogy mit várunk el a rendszerünktől, milyen szolgáltatásokat nyújtson különböző megszorítások mellett. Az is a követelmények közé tartozik, hogy mit nem valósít meg a rendszerünk.

3.1.1. Általános leírás

A BetGame.hu egy online számítógépes játék, melyen keresztül valós labdarúgó mérkőzésekre lehet fogadni.
Teljesen nélkülözi a pénz fogalmát, helyette a fogadásokhoz krediteket használ. Törekszik az interaktivitásra, így percenként frissíti és nyilvántartja az élő meccsek eredményeit, az elkövetkezendő találkozók listáját és a hozzájuk tartozó fogadási szorzókat, a felhasználók krediteit, kredit nyereményeit és veszteségeit. A lejátszott mérkőzéseket csak tizenkét órára visszamenőleg raktározza.
A felhasználók, a fogadott mérkőzéseket külön intelligens táblázatban figyelhetik, amely hanggal jelez, ha gólt szerzett egy csapat és kiemeli az együttes hátterét is külön színnel. Ezen kívül különböző hangulatjelek mutatják, hogy az adott labdarúgó mérkőzés aktuális állapota mennyire megfelelő a felhasználó által megfogadott végkimenetel szempontjából.
Minden felhasználó ugyanazon jogokkal rendelkezik, nem éreztem szükségét külön adminisztrátori jogokkal rendelkező felhasználó típust deklarálni, mivel a rendszer a legfontosabb adminisztrációs feladatokat elvégzi magától. Azon felhasználóknak, akik elveszítik összes kreditjüket, újra kell regisztrálniuk, ezzel is arra sarkalva a játékosokat, hogy jól gondolják meg fogadásaikat.

Az egész rendszer tulajdonképpen egy – a mai elvárásoknak megfelelő – felhasználóbarát, interaktív, teljesen automatikusan működő fogadójáték.

3.1.2. A fogalomszótár

A fogalomszótár tartalmazza a fejlesztés alatt használt fogalmak meghatározását. Mivel ezek más és más szakterületről származhatnak, ahol mást és mást jelenthetnek, ezért

nélkülözhetetlen fejlesztésünk szempontjából a pontos leírásuk, megfogalmazásuk. A szótáram azon fogalmakat tartalmazza, amiket a fejlesztés alatt leggyakrabban használtam és amik magyarázatra szorulnak. A fogalmak neve mellett feltüntettem az angol megfelelőjüket, mivel a kódolás és adatbázis létrehozása során törekedtem az angol szavak használatára.

Felhasználó (user): olyan személy, aki regisztrált a BetGame.hu-ra, és szerepel az adatbázisban.

Profil (profile): minden felhasználó rendelkezik egy *profil*lal, melyben szerepel a regisztráció során megadott adatai. Ezen adatok: felhasználónév, jelszó, e-mailcím, születési év, neme, és egy profilkép, melynek megadása nem kötelező. A *profil*hoz hozzá tartozik még a felhasználó által britokolt *kredit*ek száma.

Mérkőzés/meccs (match): két létező labdarúgó klub egymás ellen vívott - valóságban is létező - labdarúgó mérkőzése. Tulajdonságai: *hivatalos kezdési időpont*, *tényleges kezdési időpont*, státusz, *hazai csapat*, *vendég csapat*, eredmény, *tét*, *bajnokság*.

Hazai csapat (home team): a *mérkőzés* kiírásában elől szereplő labdarúgó csapat neve. A gyakorlatban is mindig a pályaválasztó, vagy hazai pályán játszó csapatot nevezik meg először, és csak utána a *vendég csapatot*.
Például, ha a Real Madrid és a Barcelona labdarúgó csapatok játszanak egymás ellen, és hivatalosan a Barcelona a pályaválasztó vagy ő játszik hazai pályán, akkor a *mérkőzés* kiírása a következő: Barcelona – Real Madrid.

Vendég csapat (away team): a *mérkőzés* kiírásában másodikként szereplő - nem pályaválasztó, vagy nem hazai pályán játszó - labdarúgó csapat neve.

Élő mérkőzés/meccs (live match): azon *mérkőzés*t nevezem *élő mérkőzés*nek, melynek *hivatalos kezdési időpont*ja a jelen pillanatban, van vagy már elmúlt, de a *mérkőzés* még nem ért véget.

Elkövetkezendő mérkőzés/meccs (upcoming match): azon *mérkőzés*, amely *hivatalos kezdési időpont*ja még nem jött el.

Befejezett mérkőzés/meccs (completed match): azon *mérkőzés*, amely véget ért.

Hivatalos kezdési időpont (start time): azon időpont, amelyre a *mérkőzés* kezdete hivatalosan ki van írva.

Tényleges kezdési időpont (live time): azon időpont, amikor a bíró ténylegesen elindítja a *mérkőzés*t.

Bajnokság (league): a ligák, a kupák, a bajnokságok összefoglaló neveként használom. Célja, hogy információt adjon a felhasználónak, arról, hogy milyen keretek között kerül megrendezésre a *mérkőzés*.

Például: az angol bajnokságban szereplő Manchester United és Arsenal FC csapatok találkozhatnak egymással az Angol Bajnokságban (Premier League), az Angol Kupában (FA Cup), az Angol Ligakupában (Carling Cup) vagy a Bajnokok Ligájában (Champions League) is. Az összefoglaló fogalmamat használva, ezen *bajnokságok* más presztízzsel rendelkeznek, ezért a csapatok is más erőkkel készülnek rájuk, tehát a felhasználói fogadás szempontjából hatalmas információs értékkel bír, hogy az adott *mérkőzés*re, melyik *bajnokság*ban kerül sor.

Ország (country): *mérkőzés*ekhez tartozó tulajdonság, információ, amely megmondja, hogy az adott *mérkőzés*t, mely országhoz tartozó *bajnokság*ban rendezik. Nemzetközi *mérkőzés*ek esetén az ország értéke=Nemzetközi.

Kredit (credit): a rendszerem virtuális fizetőeszköze. A felhasználó a játék folyamán *kredit*ekkel rendelkezik, *kredit*ekkel fogadhat, és *kredit*eket nyerhet.

Kockáztatott kredit (stake): egy *fogadás* során a felhasználó által, az adott *mérkőzés* egy végkimenetelére feltett *kredit* értéke.

Rendelkezésre álló kredit (credits): a felhasználóhoz tartozó *kredit* érték, nem számítva a *kockáztatott kredit*eket.

Szorzó (odds): valós számok halmazába tartozó érték. A labdarúgó *mérkőzés* mindhárom végkimeneteléhez (hazai győzelem, döntetlen, vendég győzelem) tartozik egy-egy *szorzó*.

Fogadás (bet): felhasználó által egy *mérkőzés*, egy kiválasztott lehetséges kimenetelére, a rendelkezésre álló *kredit*einél nem nagyobb *kredit* értéket tesz fel.

Várható nyeremény (prospective win): a felhasználó által fogadott meccs lehetséges végkimeneteléhez tartozó *szorzó* és a *kockáztatott kredit* szorzatából kapott *kredit*érték.

3.1.3. A forgatókönyv

A forgatókönyv egy olyan szöveges dokumentum, amelyben a rendszerünkhöz kapcsolódó összes fontos interakciót írjuk le. Nem tartalmaz informatikai szakkifejezéseket és fogalmakat sem. Egyszerű köznyelven írjuk le alkalmazásunk működését, így bárki számára értelmezhető.
A BetGame.hu forgatókönyve a következő:

1. Regisztráció/Profil elkészítése
A nyitóoldalon (***www.betgame.hu/index.php***) a „Regisztrálj itt” szövegre kattintva érhető el a regisztrációs űrlap. A leendő felhasználónak itt kell megadnia a felhasználónevét, jelszavát kétszer, e-mailcímét, születési évét, nemét és egy képet a profiljához, ami nem kötelező. Az oldalon lévő információs sávban szöveges üzenetek jelennek meg, a megadott adatok formai követelményének helyességéről, vagy hibájáról.
Az „OK” gombra kattintva a megadott információk elküldhetők, viszont hibás adatbevitel esetén az oldalon marad a felhasználó, minden megadott információt megőriz a rendszer, kivéve a hibásakat.
Helyes regisztráció esetén az adatok bekerülnek az adatbázisba és a nyitóoldalra léptet a rendszer, ahol a Bejelentkezés található.

A felhasználó a „Kilép” gombra kattintva megszakíthatja regisztrációját, az adatbázisba így nem kerül semmi és a kezdőoldalra lép.

2. Bejelentkezés

Csak az a felhasználó jelentkezhet be, aki regisztrált az oldalra. A nyitóoldalon a bejelentkezési felületen, a felhasználónév és jelszó megadása után, a „Belépés” gombra kattintva lehet bejelentkezni. A rendszer közli, ha hibás a felhasználónév vagy a jelszó, és nem engedi tovább a felhasználót.

Helyes adatok megadása után a felhasználó a főoldalra kerül, itt jelenik meg az oldal tartalma az összes funkciójával. Első bejelentkezés esetén a felhasználó megkapja 1000 kreditjét.

3. Adataim megváltoztatása

Csak a bejelentkezett felhasználó változtathatja meg az adatait. A főoldali menü „Adataim megváltoztatása” menüpontjára kattintva, a jelszó kétszeri megadásával bármely adat megváltoztatható, kivéve a felhasználónevet.

Az információk megadása közben szöveges üzeneteket közöl a rendszer a megadott információk formai követelményének helyességéről vagy hibájáról.

Az „OK” gombra kattintva a megadott információk elküldhetők, viszont hibás adatbevitel esetén az oldalon marad a felhasználó, minden megadott információt megőriz a rendszer, kivéve a hibásakat. Ha mindent helyesnek talál a rendszer, akkor a változtatások bekerülnek az adatbázisba és a főoldalra visszaugrunk.

A „Kilép” gombra kattintva elhagyhatjuk a lapot, az adatbázisba nem történik változás és szintén a főoldalra kerülünk.

4. Fogadás

Csak bejelentkezett felhasználó fogadhat. A főoldal „Következik:” táblájában felsorolt meccsekre lehet fogadni, mivel ezek még nem kezdődtek el. A csapat nevei mellett szereplő (sorban: hazai szorzó, döntetlen szorzó, vendég szorzó) értékekre kattintva lehet indítani a fogadás menetét. Miután a felhasználó eldöntötte, hogy a hazai csapatra, vendég csapatra vagy döntetlenre fogad, rákattintva a megfelelő szorzó értékre, a felugró fogadási ablakban adhatja meg a kockáztatott kreditek számát.

A „Mégsem” gombra kattintva bezáródik a fogadási ablak, és így a fogadás nem történik meg.

Az „OK” gombra kattintva a rendszer elkezdi a fogadás feldolgozását. Ha nem egész számot, vagy több kreditet adott meg a felhasználó, mint amennyivel rendelkezik, akkor a fogadási ablakban megjelenik a hibára utaló értesítési szöveg, az „Újra” és a „Kilép” gombok. Az „Újra” gombra kattintva ismét próbálkozhat a kredit érték megadásával a felhasználó, a „Kilép” gombra kattintva bezáródik a fogadási ablak, és így a fogadás nem történik meg.

Az „OK” gombra kattintva, ha a rendszer minden feltételnek megfelelő kredit összeget talált, akkor a fogadást elmenti az adatbázisba, felkerül a főoldalon található „Fogadott meccseid:” nevű táblázatba is, és a kockáztatott kredit értékét levonja a felhasználó összes kreditjéből. A fogadási ablakban értesítést is kap a felhasználó a fogadás sikerességéről, megjelenik egy „Újra” és egy „Kilép” gomb. Az „Újra” gombra kattintva módosíthatjuk fogadásunkat. A „Kilép” gombra kattintva bezárhatjuk a fogadási ablakot.

5. Fogadás módosítása

Csak bejelentkezett felhasználó módosíthatja fogadását és csak azon fogadott meccseket lehet módosítani, amelyek a „Következik:” táblában szerepelnek.

A fogadott meccsek módosításának menete megegyezik a fogadáséval. Bármilyen módosítást is végez el a felhasználó (más végkimenetelre fogad; más kredit összeget kockáztat) minden változást elment a rendszer az adatbázisba és a változás megjelenik a „Fogadott meccseid” táblázatban is.

6. Fogadás törlése

Csak bejelentkezett felhasználó törölheti fogadását és csak azon fogadott meccseket lehet törölni, amelyek a „Következik:” táblában szerepelnek.

A „Fogadott meccseid” táblázatban, a törölni kívánt meccs sorára kattintva megjelenik a törlési ablak. A „Mégsem” gomb kiválasztásával nem történik változás. A „Törlés” gombbal lehet törölni fogadást, melynek következménye, hogy a fogadásunk törlődik az adatbázisból, és a „Fogadott meccseid” táblázatból, a kockáztatott kreditet pedig a felhasználó visszakapja. A törlés sikeréről vagy sikertelenségéről a rendszer szöveges üzenetet küld a törlési ablakba, amely ezután a „Kilép” gombbal zárható be.

7. Nyeremény és veszteség elkönyvelése

A nyeremények és veszteségek elkönyvelését a rendszer automatikusan végzi. Amint véget ért egy meccs kialakul, a végeredmény. Aki fogadott az adott meccsre és eltalálta a meccs három végkimeneteléből (hazai győzelem, döntetlen, vendég győzelem) a helyeset, annak krediteihez hozzáadja a rendszer a szorzó és a kockáztatott kredit szorzatából megkapott kreditértéket. Aki nem találta el a meccs végkimenetelét helyesen, az elveszti az adott meccsre feltett kockáztatott krediteit.

8. Meccsek adatainak frissítése

A rendszer percenkét végzi a meccsek adatainak frissítését teljesen automatikusan.

9. Felhasználó törlése

Csak regisztrált és bejelentkezett felhasználó törölheti magát. A főoldal „Adataim megváltoztatása" menüpontja alatt található „Törlöm magam" gomb megnyomásával, a rendszer törli a felhasználó összes adatát az adatbázisból a fogadott meccseivel együtt, és a kilépés is végrehajtódik.

10. Kilépés

Már korábban bejelentkezett felhasználó léphet csak ki az oldalról. A menü „Kilépés" menüpontjára kattintva, a rendszer kijelentkezteti a felhasználót, és a nyitóoldalra kerül.

3.1.4. Rendszerkövetelmények

A rendszer bármilyen platformon használható, amely rendelkezik internet-kapcsolattal és Firefox 3.0, vagy magasabb verziójú böngészővel.

3.2. Tervezés

A fejlesztés következő lépcsőfoka a tervezés. Az alkalmazásfejlesztésnek egyik problémája, hogy a tervezésnél elviekben nem lenne szabad figyelembe venni implementációs dolgokat, a terv egy absztrakció, az alkalmazás absztrakciója, mely független a konkrét implementációtól. Tervezni viszont nem lehet anélkül, hogy implementációs problémákat ne hoznánk fel a tervezés szintjére. Ez egy ellentmondás.
Az adatbázisom tervezésénél én sem tudtam teljesen függetleníteni a tervet a későbbi megvalósítástól. Az adataimat a ***www.wsn.com*** livescore szolgáltatást nyújtó weboldalról kapom, így ezen adatok nagymértékben meghatározták az adtabázisom tervét.

3.2.1. Az adatbázis tervezése

Az adatbázis tervezése során a legfontosabb, hogy a tábláinkat, a táblákban szereplő adatok típusait és a táblák között lévő relációkat megfelelő módon határozzuk meg.
Egy táblában csak a konkrétan összetartozó adatokat érdemes szerepeltetni. Az oszlopokban tárolt adatok típusának megfelelő kiválasztásával csökkenthetjük adatbázisunk tárhely-igényét.
Adatbázisomban öt táblát hoztam létre, melyek felépítése és egymás közötti viszonya az 5. ábrán látható.

Adatbázisom táblái:
Users tábla: itt szerepel a felhasználó által regisztrációkor megadott összes adat, ezen felül pedig az adminisztratív információk (reg_date = regisztráció dátuma; last_login = utolsó bejelentkezés dátuma), és a rendelkezésre álló kreditek száma. A user_id az elsődleges kulcs a táblában, melyet a MySQL határoz meg az új felhasználó felvételekor automatikus inkrementáció segítségével. A kép fájlokat nem az adatbázisba menti a rendszer, csak azok tárhelyen lévő elérési útját tartalmazza, ezzel jelentősen csökkentve az adatbázis méretét.

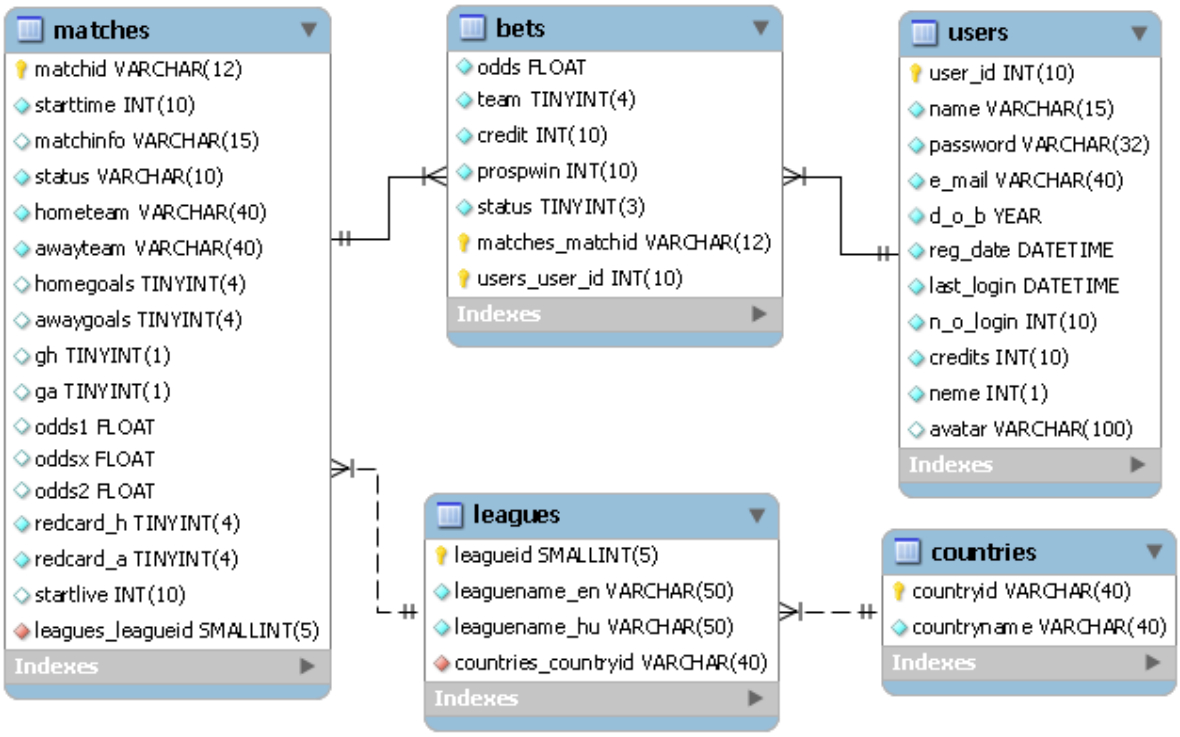
Matches tábla: tartalmazza a mérkőzéseket, a hozzájuk kapcsolódó adatokat, mint a hivatalos kezdési idő, a tényleges kezdési idő, az odds-ok, a piros lapok, az eredmény.

Mivel az oldal funkcióit tekintve felesleges csapatokat tartalmazó tábla létrehozása, ezért itt tárolódik a mérkőzést játszó hazai és vendég csapat neve is.

Bets tábla: az egyes felhasználók által fogadott meccseket és a fogadásokhoz tartozó adatokat tartalmazza. Ahogy az 5. ábrán is látszik, a bets tábla N:1 kapcsolatban van mind a matches, mind pedig a users táblával, tehát egy meccshez és egy felhasználóhoz több fogadási bejegyzés is tartozhat, vagy másként fogalmazva, egy felhasználó user_id-ja és egy mérkőzés matchid-ja határoz meg egyértelműen egy fogadást a bets táblából.

Leagues tábla: a bajnokságokat tartja nyilván. Csak a matches táblával van kapcsolatban, mégpedig 1:N relációban, mivel egy bajnokságban több mérkőzést is rendezhetnek. A matches táblában megjelenik a leagueid oszlop külső kulcsként.

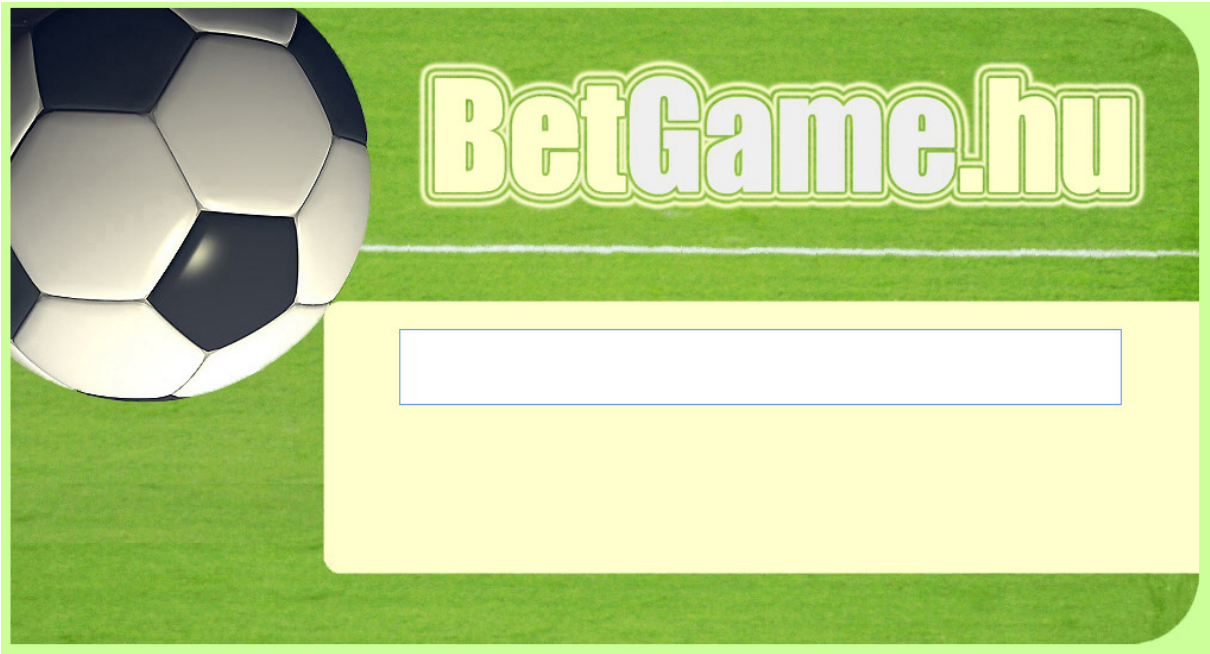
Countries tábla: az országok neveit tartalmazza. A leagues táblával van 1:N relációban, mivel egy országban több bajnokságot is rendezhetnek. A leagues táblában megjelenik a countryid oszlop külső kulcsként.



5. ábra. *Az adatbázistábláim és köztük lévő relációk.*

3.2.2. A felhasználói felület megtervezése

Ezen tervezési folyamat végterméke egy képfájl, amit a 2.10.4. fejezetben bemutatott Hornil StylePix képszerkesztő szoftverrel készítettem el. A 6. ábrán látható a tervezés eredményeként létrejött kép, melynek felhasználásáról a 3.3.1. fejezetben írok.



6.ábra.

3.3. Implementálás

A követelmények meghatározása és a tervek elkészítése után lehet elkezdeni az implementálást. A programozás alatt, verziók sorozatán keresztül jutottam el a végső, teljes megvalósításig. Minden verzióban egy-egy fontosabb funkciót építettem be az oldalba, és minden verzió végén, egy nem teljes, de használható weboldal állt rendelkezésre.
A következő alfejezetekben a legfontosabb lépéseket szeretném bemutatni, egyes részleteket kiemelni, és a megoldást elmagyarázni, indokolni.

3.3.1. A weboldalam struktúrája és külseje

Az első feladat egy jól strukturált HTML kód implementálása, melynek létrejötte után kialakuló vázra, CSS segítségével a felhasználó által látható külső megjelenés „ráhúzható”.

Még ma is van rá példa, hogy egy weboldal egész felépítését táblaszerkezettel valósítják meg, ami teljes mértékben kerülendő megoldás. Ehelyett a HTML általános tároló címkéjét, a taget használjuk, ami a weboldalak felosztását elősegítő jelölés. A semmilyen jelentést nem rendel ahhoz a tartalomhoz, amit közrefog, hanem egy általános módszert biztosít, hogy a tartalmat, olyan saját struktúrába vagy csoportokba rendezzük, amelyre nincs más használható HTML elem. Ha az ilyen csoportokhoz, szemantikus osztályokat vagy azonosító neveket rendelünk, akkor egy CSS fájl segítségével már könnyen definiálhatjuk a külső megjelenésüket.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <link rel="stylesheet" type="text/css" href="mystyle.css" />
  <meta http-equiv="content-type" content="text/html; charset=utf-8" />
  <title>BetGame.hu</title>
  <link href='pic/icon.ico' rel='shortcut icon'/>
</head>
<body>
    <div id="page">
        <div id="top"> </div>
        <div id="menu"> </div>
        <div id="info"> </div>
        <div id="leftcorner">
            <div id="avatar"> </div>
            <div id="avatar_keret"> </div>
        </div>
        <div id="center">
            <div id="leftsidetext"> </div>
            <div id="centertext"> </div>
        </div>
        <div id="bottom"> </div>

    </div>
</body>
</html>
```

7. ábra.

A 7. ábrán látható a HTML kód, a 8. ábrán pedig CSS segítségével, a mystyle.css fájlban definiált stílusok alapján, a böngészőben megjelenő külső. Az ábrán külön jelöltem, hogy az egyes elemek, melyik részeket határozzák meg. Az oldal három fő részre tagolható (felső, középső és alsó), melyeket a „page” azonosítóval ellátott elem tartalmaz.

A felső részhez tartozik a „top”, „menu”, „info” és „leftcorner” azonosítóval ellátott elemek, amiket nem helyeztem el külön tagben, mert nem láttam szükségét. Ezen elemek hátterei, az oldal külső megjelenésének tervezésekor, a Hornil StylePix képszerkesztő programmal előállított képfájl felszabdalt részei. A mystyle.css fájlban, ezen elemek elhelyezkedésére vonatkozó tulajdonságokat, pixel pontosan kellett megadni a helyes illeszkedés miatt.

8. ábra.

A középső rész megegyezik a „center” azonosítóval rendelkező taggel. Ez további két elemet tartalmaz, a „leftsidetext” és a „centertext” azonosítójú elemeket, ahol a főoldal lényegi tartalma jelenik meg. A „leftsidetext”-ben a felhasználó által fogadott mérkőzések táblája, és a „centertext”-ben sorrendben az élő, a következő és a lejátszott mérkőzések táblái. Fontos, hogy a „center” tartalma állandóan változhat, ezért CSS-sel nem határoztam meg magasságáért felelős *hight* tulajdonságot, melynek értéke pixelben adható meg. Az alsó részt képviselő, „bottom” azonosítójú elemnek illeszkednie kell a „center” elem aljához, és vele együtt mozogni a tartalom változásával. Ennek megoldása nehézséget jelentett számomra, azonban az interneten rákeresve a probléma megoldására, kiderült, hogy másnak is.

Több bonyolult módszer után találtam meg a legegyszerűbbet a ***http://www.sitepoint.com/examples/clearing_floats/example2.php*** angol nyelvű oldalon, és ekkor jöttem rá, hogy a megoldás már a kezemben volt, viszont rosszul használtam. A CSS definiálja az *overflow* tulajdonságot, aminek segítségével reagálhatunk arra, ha egy elem tartalma „túlcsordul” az adott határain. Több értéket is felvehet a tulajdonság, viszont ebben az esetben az „*auto*” értéket kell megadnunk a CSS fájlban, viszont nem mindegy, hogy melyik elem tulajdonságaként. Az oldalam esetében a „leftsidetext” vagy a „centertext” azonosítóval rendelkező elemek tartalma

„csordulhat” túl, viszont az „*overflow: auto;*” tulajdonságot nem ezen elemek stílus-meghatározójaként, hanem az őket tartalmazó „center” azonosítójú elem tulajdonságaként kell definiálni. Így előállt a későbbi dinamikus tartalomra megfelelően reagáló, jól strukturált weboldal felépítése és külseje.

3.3.2. A regisztráció megvalósítása

A regisztráció kódja a newuser.php fájlban található. A fájl tartalma kitűnő példát nyújt többféle technológia együttműködésére, mivel itt egyszerre jelenik meg HTML, PHP és JavaScript kód.
A regisztrációs oldal felépítése megegyezik az előző pontban leírtakkal. A kitöltendő HTML <form> a „centertext” azonosítójú elemben jelenik meg, míg a <form> egyes <input> mezőire vonatkozó megszorításokat a newuser.js JavaScript fájl ellenőrzi, és az esetleges hibákat az „info” elemben közli a felhasználóval.

A newuser.js JavaScript nyelven írt függvényeket tartalmaz. A <form> minden <input> mezőjéhez tartozik egy függvény, amivel az <input> mezőben megadott adatok helyességét vizsgálja, mégpedig úgy, hogy a HTML elemeknek megadhatóak bizonyos események bekövetkeztét figyelő attribútumok, melynek értékei JavaScript függvények hívása lehet.
A 9. ábrán látható a HTML, a PHP és a JavaScript együttműködésére egy példa. Az ábra tartalmazza, egy HTML táblába rendezett, <form> kódját. A <form> HTML elem deklarációjában már látszik, hogy az adatok küldését, a *validate* JavaScript függvény engedélyezheti (*onsubmit=”return validate(this.id);”*), minden általa támasztott feltétel teljesülése esetén, ha bekövetkezett az *onsubmit* esemény. Ez akkor történhet, ha egy *submit* típusú gombra kattint a felhasználó. Esetünkben ez lehet az „OK” vagy a „Kilép” gomb is, tehát a *validate* függvény, nemcsak a <form>-ban közölt adatok helyességét vizsgálja, hanem a kiléptetést is indíthatja (a függvény kódja a 10. ábrán látható). A <form> elem deklarációjában szintén látható, hogy az adatok továbbítása a POST PHP metódussal (*method=”post”*) valósul meg, és a feldolgozást a newuser.php fogja elvégezni (*action=”<?php echo $_SERVER[„PHP_SELF”];?>”*).

```
<form action="<?php echo $_SERVER["PHP_SELF"];?>" method="post"
enctype="multipart/form-data" onsubmit="return validate(this.id);">
<table>
<tr><td>Név:</td><td><input type="text" id="uN"
    style="height:14px" onkeyup="userNameCheck(this.id)"></td></tr>
<tr><td>Jelszó:</td><td><input type="password" id="uJ"
    onchange="passwordCheck(this.id)"></td></tr>
<tr><td>Jelszó ismét:</td><td><input type="password" id="uJM"
    onchange="passwordEqual(this.id)"></td></tr>
<tr><td>E-mailcim:</td><td><input type="text" id="eM"
    onchange="emailCheck(this.id)"></td></tr>
<tr><td>Születési év:</td><td><input type="text" id="uD"
    onchange="dateOfBirthCheck(this.id)"></td></tr>
<tr><td>Neme:</td><td><select name="uNeme" id="uN">
    <option value="0">férfi</option>
    <option value="1">nő</option>
    </select></td><td></td></tr>
<tr><td>Profil kép:</td><td><input type="file"
                name="uPic" size="15"></td></tr>
</table>
<input type="submit" name="uOK" id="uOK" value="OK"
    onclick="btnWhichButton=this">
<input type="submit" name="uKilep" id="uKilep" value="Kilép"
    onclick="btnWhichButton=this">
</form>
```

9. ábra.

Mivel a kitöltött <form>-ot szintén a newuser.php dolgozza fel, így a PHP kódnak három esetről kell döntenie:

- ha a felhasználó most került az oldalra, akkor üres <form>-ot küld a kliens böngészőjébe,
- ha a felhasználó kitöltötte a <form>-ot és elküldte, akkor egy biztonsági vizsgálatot elvégez közvetlenül mentés előtt, hogy nem került-e azóta az adatbázisba a menteni kívánt felhasználónévvel azonos név,
    - ha igen, akkor az „info” elemben közli a felhasználóval a hibát, és kiírja újra a <form>-ot, viszont a könnyebbség érdekében a többi megadott adatot (melyeket a POST metódussal történt küldésnek köszönhetően, a $_POST tömb tartalmaz) beírja a <form>-ba, hogy a felhasználónak ne kelljen újra kitöltenie azt,

- ha nem, akkor elmenti az adatbázisba a megadott adatokat (szintén a $_POST tömb segítségével) és az index.php-ra irányítja a felhasználót, ahol bejelentkezhet.

```
192   //Űrlap validálása vagy kiléptetés
193   function validate(x){
194     var submitIt=true;
195     var text="";
196
197     if(btnWhichButton.value=="Kilép"){return true;}
198     if(!userNameCheck("uN"))
199       {submitIt=false;text=text+"Hibás nevet adtál meg! ";}
200     if(!passwordCheck("uJ"))
201       {submitIt=false;text=text+"Hibás jelszavat adtál meg! ";}
202     if(!passwordEqual("uJM"))
203       {submitIt=false;text=text+"Nem egyezik a két megadott jelszavad! ";}
204     if(!emailCheck("eM"))
205       {submitIt=false;text=text+"Hibás e-mailcimet adtál meg!";}
206     if(!dateOfBirthCheck("uD"))
207       {submitIt=false;+"Hibás születésiévet adtál meg!";}
208     //hibák kirása az info div-be
209     document.getElementById('info').innerHTML="<warning>"+text+"</warning>";
210
211     return submitIt;
212   }
```

10. ábra.

A *validate* függvény utolsó előtti utasítása az „info” elembe írja, hogy milyen hibákat talált, vagy helyesen megadott adatok esetén egy üres sztringet.

Ahhoz, hogy a JavaScript-tel mindezt el lehessen végezni, az oldalak betöltésekor létrehozza az adott weboldal DOM felépítését, vagyis a Document Object Modelt. A DOM tartalmazza az adott oldal összes elemét reprezentáló objektumot, mégpedig az oldal struktúrájának megfelelően felépített fa szerkezetben. A fa gyökéreleme a *window* objektum, mely a böngészőablakot képviseli, ennek gyermeke a *document* objektum. Ezen objektum segítségével elérhető és megváltozatható, bármely elem az oldalon. Az én példámban az látható, hogy a *document* objektum *getElementById* függvényével elérem az „info” azonosítójú elemet, ami szintén egy objektum a DOM-ban, melynek *innerHTML* függvényével új értéket adok az elem által tartalmazott HTML kódnak.

Ez csak egy apró példa arra, hogy mit lehet elérni a JavaScript-el. Ezen kívül bármely elemet megváltoztathatunk, törölhetünk, vagy akár újabb elemeket hozhatunk létre, ott ahol csak akarunk az oldalon.

3.3.3. A bejelentkezés megvalósítása

Egy webes oldalon történő beléptetés több módon is implementálható. Elsőre egyszerűnek tűnhet a megoldás, mivel általában két sztring értéket (a felhasználónevét vagy e-mailcímét, és a jelszavát) kell egy felhasználónak megadnia helyesen és már a főoldalra is kerül. Viszont törekednünk kell a biztonságra. A beléptetés során a legalapvetőbb biztonsági intézkedések a következők:

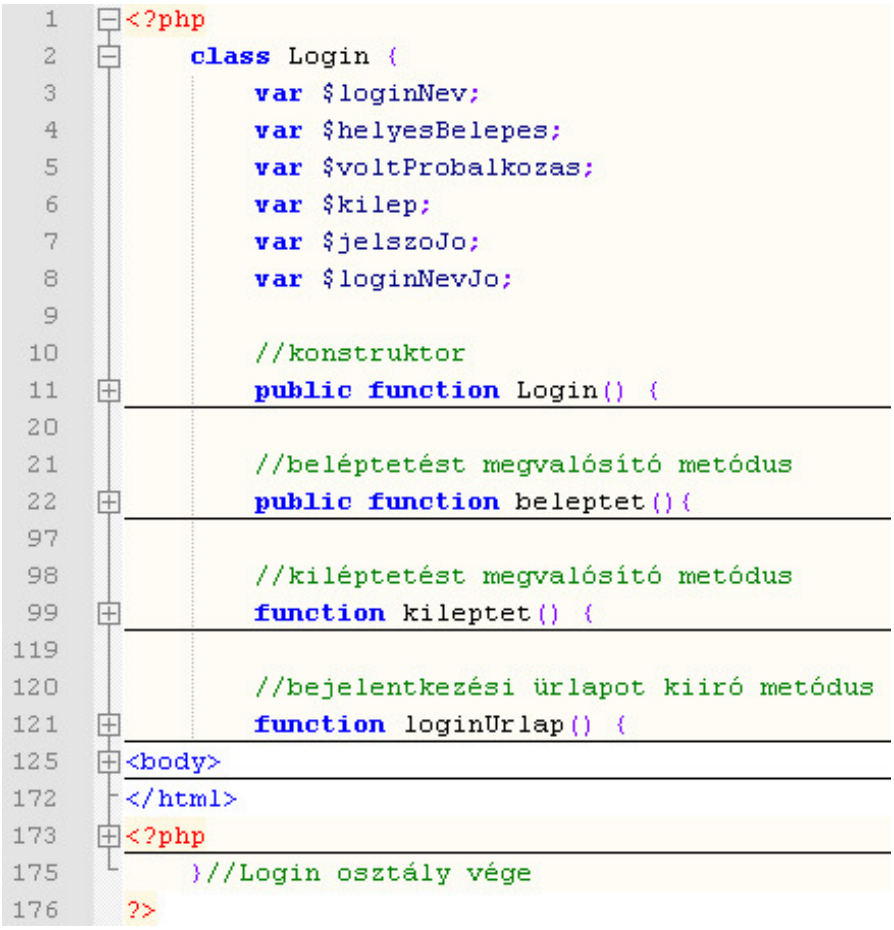
- A belépésnél a jelszó megadására *password* típussal ellátott, <input> HTML elemet használjunk, hisz ekkor még véletlenül sem láthatja meg más, hogy a felhasználó mit gépelt be a mezőbe.
- A regisztrációkor megadott jelszót valamilyen titkosítási eljárással mentsük az adatbázisba. Erre kitűnően alkalmas a PHP *md5* hash függvénye, ami bármekkora sztring bemenethez generál egy 32 karakterből álló hexadecimális számot. Ez a titkosítási függvény egyirányú, tehát a generált 32 karakteres kódból nem tudjuk visszakapni az eredeti sztringet, viszont a bejelentkezéskor helyesen megadott jelszót, *md5* függvénnyel átalakítva ugyanazon kódot kapjuk, ami az adatbázisunkban van, így a bejelentkezés megtörténhet.
- A bejelentkezéskor beírt adatokat csakis POST metódussal küldjük, mert a GET metódus esetén, az adatok megjelennek a böngészőben a webcím után.

Az oldalam bejelentkezésének megvalósításában, és a további működésében is fontos szerepe van még a PHP-hez tartozó munkameneteknek, vagyis a session-öknek. A munkamenetek egy egyedi azonosítót biztosítanak a felhasználóknak, ezen keresztül lehet információkat oldalról oldalra vinni, megtartani. Minden egyes munkamenet létrehoz egy fájlt a szerveren, egy egyedi azonosítóval, és ebben tárolja a felhasználóhoz (klienshez) tartozó aktuális változókat. Kliens oldalon ez az egyedi azonosító, a session

id, ami sütiben (cookie)[8] vagy ritkább esetben a böngésző címsorában tárolódik. Így minden kliens egyedi kulcsot kap, amihez csak ő fér hozzá, és ezzel a kulccsal tud a szerveren az egyik lapról a másikra információt átvinni.
A PHP több függvénnyel rendelkezik a session-ök kezelésére. Ha használni szeretnénk, akkor az aktuális fájl elején el kell helyeznünk a következő sort: *<?php session_start();?>*. Fontos, hogy a függvényhívás előtt nem szerepelhet semmilyen kód, ami outputot szolgáltat a böngésző számára.
A PHP-ben létezik egy szuperglobális tömb, ez a *$_SESSION* tömb. Miután elindítottuk a *session_start* függvénnyel a session-t, ebben a tömbben tárolhatjuk azon változókat, amelyeket szeretnénk megőrizni. A tömbből változókat törölni az *unset* függvénnyel lehet, ha paraméterként megadjuk a törölni kívánt változó nevét. A *$_SESSION* tömb teljes tartalmának törlését, a *session_unset* és a *session_destroy* függvények, ilyen sorrendben történő meghívásával lehet elérni. A PHP magától is törölheti a *$_SESSION* tömb tartalmát, ha nem észlel aktivitást, a php.ini fájlban meghatározott ideig.

A BetGame.hu-n való bejelentkezéséhez létrehoztam egy *Login* nevű osztályt, aminek PHP kódját a login.php tartalmazza. A 11. ábrán látható a *Login* osztály változói és metódusai. A *$loginNev* változó tartalmazza a felhasználó nevét a későbbi azonosításra, a többi változó mind logikai értéket vehet fel és arra szolgálnak, hogy a *beleptet* és *kileptet* metódusok, ezen logikai értékek alapján eldöntsék, milyen megfelelő műveletet hajtsanak végre.

[8] **Cookie**: egy a felhasználó gépén tárolt információs fájl, legtöbbször beállításokat, preferenciákat tartalmaz annak érdekében, hogy egy későbbi látogatás során a felhasználót és az általa használt beállításokat automatikusan, újabb beállítások nélkül be lehessen tölteni.



11. ábra.

Az index.php és login.php egymással közreműködve végzik a beléptetést, kiléptetést és annak eldöntését, hogy a felhasználó éppen be van e jelentkezve vagy nem. Az egész folyamat a session-re, és a *$_SESSION* szuperglobális tömbben elhelyezett *Login* osztály aktuális példányára épül.

Az index.php elején a *session_start* függvénnyel indítom a munkamenetet, a függvény hívása előtt azonban csatolnom kell, vagy programozók között elterjedt kifejezéssel élve, include-olnom kell a login.php-t. Mindezt azért fontos, mert ha a *$_SESSION* tömb már tartalmazza a *Login* osztály egy példányát, akkor nem fogja tudni meghatározni a PHP az elem típusát, hacsak nincs előtte deklarálva. A következő PHP részben include-olom a config.php fájlt, amiben az adatbázis csatlakozásához szükséges utasítások szerepelnek. Ezeket érdemes mindig külön fájlba tenni, ha megváltozna a host, az adatbázis neve, az adatbázis tulajdonosának neve vagy jelszava, akkor elég csak a config.php fájlban elvégezni a változtatásokat.

Az index.php két dolgot vizsgál, létezik-e egy *Login* példány a *$_SESSION* tömbben, vagy nem. Tulajdonképpen mindkét esetben a beleptet metódus hívja meg az index.php, viszont ha nem létezik *Login* példány a *$_SESSION* tömbben, előbb példányosít egyet és elhelyezi abban.
A beleptet metódus dönti el a *Login* példány változóinak aktuális értékei alapján, hogy a felhasználó szeretne bejelentkezni, rossz adatokat adott meg a bejelentkezésnél, vagy már bejelentkezett, és ezután a megfelelő felületet biztosítja a felhasználó számára.
A kileptet metódus törli a *$_SESSION* tartalmát, és a PHP *header* függvényével az index.php-ra irányítva az megjeleníti a belépési felületet.

3.3.4. A weboldalam adatainak frissítése

A programozási idő legnagyobb részét ezen szakasz megvalósítása tette ki. Itt kellett implementálnom a mérkőzések adatainak megszerzését, adatbázisba mentését, és onnan a felhasználónak való megjelenítését.

Az interneten nagyon sok, úgynevezett livescore (tükörfordításban: élőeredmény) szolgáltató oldal létezik, ahol folyamatosan frissülő adatokat közölnek különböző - nem csak labdarúgó - mérkőzésekről. Mielőtt bármibe is belekezdtem volna, meg kellett találnom a követelményeimnek legmegfelelőbb livescore szolgáltatást, hisz ez teljesen meghatározza az implementálás további menetét.
Hosszas böngészés és keresés után a következő lehetőségek rajzolódtak ki számomra.
A legrobosztusabb rendszereket fizetős szolgáltatásra építve lehet létrehozni. A mérkőzések adatait XML fájlokban kapjuk, és legtöbb esetben szorzókat is tartalmaznak. Viszont szinte kivétel nélkül mind külföldi szolgáltatás, és így elég drágák. Egyetlen, magyarul is elérhető szolgáltatást találtam (***www.xml-scores.com***), melynek szintén borsos ára van (160 Euro/hónap).
Milyen ingyenes lehetőségeink vannak? Sokáig ragaszkodtam a megfelelő XML alapú szolgáltatás megtalálásához, viszont egyik sem felelt meg a követelményeimnek, mivel csak mérkőzések eredményeit tartalmazó, vagy csak a mérkőzések szorzóit tartalmazó XML szolgáltatást találtam. Két ilyen XML dokumentum összefűzése elég nehézkes vállalkozásnak tűnhet, mert nem egyszerű kötni az egyik XML dokumentum

mérkőzéseihez, a másik XML dokumentum szorzóit. A másik hátrányuk, hogy nem frissülnek gyakran.
Sok livescore oldal ajánl fel a saját oldalunkon elhelyezhető widget[9]-eket, viszont ezekből nem lehet adatokat kinyerni, és gyakran különböző reklámok megjelenésével is együtt jár, tehát ez sem megoldás. Mellesleg ezek is mind angol nyelvűek, viszont van egy állandóan fejlődő, magyar nyelven is elérhető oldal, a ***www.eredmenyek.com***, amit ajánlok minden olyan felhasználó figyelmébe, aki több információt is szeretne megtudni az élő mérkőzésekről (sárgalapok, piroslapok, gólszerzők, kezdőtizenegyek).
Számomra az egyetlen ingyenes megoldást azon livescore oldalak nyújtják, melyek adatai nincsenek védve. Ezen oldalak kliens oldali HTML kódja lemásolható, és az adatok kinyerhetőek. Ez sem robosztus megoldás, ha megváltozik az oldal struktúrája vagy működése, akkor az oldalunkra is hatással van, viszont ingyenes megoldások közül a legjobb.

Én a ***www.wsn.com*** oldalt választottam. Bár angol nyelvű, de minden számomra fontos információ elérhető rajta egyetlen helyen, a ***www.wsn.com/football/livescores/*** linken. Tehát az élő mérkőzések, a következő mérkőzések, a lejátszott mérkőzések, a hozzájuk tarozó szorzók is megtalálhatóak itt. Ezen kívül a szingapúri másodosztálytól kezdve, a Magyar NB1-en át, az angol Premier League-ig minden labdarúgó mérkőzésről élő eredményeket közöl az oldal.

Mérkőzések adatainak megszerzése a wsn.com weboldalról
Célom az volt, hogy a ***www.wsn.com/football/livescores/*** tartalmát lementsem egy dokumentumba, ebből egy jól-formát XML-t nyerjek, majd a számomra fontos adatokat, XPath segítségével elérve, eltároljam az adatbázisomban. Mindezek megvalósítására a PHP kitűnő eszközöket szolgáltat.

Egy weboldal tartalmát két módon is le lehet menteni: az egyszerűbb eset a *file_get_contents* PHP függvénnyel valósítható meg, azonban a *file_get_contents* használata okozhat időtúllépést, ha nem elérhető a weboldal, így megállítja a kódunkat.

[9] Widget: Az angol szó eredeti jelentése, kütyü vagy bigyó. A számítástechnikában a widget rendszerint kicsi, valamely alkalmazáson belül futó, vagy az operációs rendszerrel induló hasznos segédprogramot jelent, mely egyedi felépítése kifejezetten egy bizonyos funkció ellátására lett kialakítva.

Én inkább a *cURL library* segítségével oldottam meg ezt. A *cURL library* függvényeivel csatlakozni tudunk többféle típusú szerverhez, és kommunikációt végezhetünk velük, többféle protokoll segítségével.

A 12. ábrán látható két függvény valósítja meg a ***www.wsn.com/football/livescores/*** weboldal tartalmának fájlba mentését. Mindkét függvény a getdata könyvtár, getdata.php fájlban található.

A *get_url* függvény szerzi meg a paraméterként átadott URL tartalmát, ami esetünkben mindig a ***www.wsn.com/football/livescores/*** lesz. A *curl_init* függvény segítségével inicializálok egy *cURL* munkamenetet. Ezután beállítom a munkamenet tulajdonságait, amit a *curl_setopt* függvénnyel lehet elvégezni. A *curl_setopt* paramétereként mindig meg kell adni a beállítani kívánt munkamenet azonosítóját, az opció nevét (ami CURLOPT_ karakterekkel kezdődik), és az opciónak beállítani kívánt értéket. Az én beállításaim a következők:

- CURLOPT_REFERER: ezzel a hivatkozó oldal URL-jét lehet beállítani. Esetemben azt érem el, mintha nem a BetGame.hu, hanem a ***http://www.wsn.com*** hivatkozná a *$url* változóban megadott oldalt,
- CURLOPT_HEADER: a HTML header részét vágja le,
- CURLOPT_RETURNTRANSFER: a *curl_exec* függvényre van hatással. Ha igazra állítjuk, mint én, akkor a *curl_exec* függvény a kért URL tartalmát sztringben adja vissza,
- CURLOPT_URL: az URL megadása, ami esetemben a következő lesz: ***http://www.wsn.com/football/livescores/***.

A *curl_exec* függvény hívásával lefut a kérés és a beállításoknak megfelelően egy sztring visszatérési értékben megkapom az oldal tartalmát, amit elmentek a *$data* változóban. Ezután le kell zárni a munkamenetet a *curl_close* függvénnyel, és visszatérek a *$data* értékével.

A *get_url_data* függvényemmel mentem el a *$data* tartalmát az xml.txt fájlba, mégpedig egy *DOMDocument* típusú objektum segítségével, amit a PHP5 DOM kiterjesztése definiál. A DOM kiterjesztés lehetővé teszi XML dokumentumokon történő különböző műveletek végrehajtását.

A *DOMDocumentum loadHTML* függvényével sztringből tudunk HTML tartalmat betölteni egy *DOMDocument* típusú objektumba. A *loadHTML* függvény elemzi a sztringben megadott tartalmat, de nem követeli meg a jól-fomáltságot. Ezután megnyitjuk az xml.txt fájlunkat (ha még nem létezik, akkor létrejön az *fwrite* függvény hatására), és belemásoljuk a *$dom* változóban tárolt XML reprezentációt a *saveXML* függvény segítségével. A fájlunkat lezárjuk.

```
27  function get_url($url){
28      $curl = curl_init();
29      curl_setopt($curl,CURLOPT_REFERER,"http://www.wsn.com");
30      curl_setopt($curl, CURLOPT_HEADER, 0);
31      curl_setopt($curl, CURLOPT_RETURNTRANSFER, 1);
32      curl_setopt($curl, CURLOPT_URL, $url);
33      $data = curl_exec($curl);
34      curl_close($curl);
35      return $data;
36  }
37  function get_url_data(){
38          $dom = new DOMDocument('1.0', 'UTF-8');
39          $dom->loadHTML(get_url("http://www.wsn.com/football/livescores/"));
40          $file = fopen ("xml.txt", "w");
41          fwrite($file,$dom->saveXML());
42          fclose ($file);
43  }
```

12. ábra.

Ezek után egy jól-formázott XML dokumentumot kell létrehoznom az xml.txt fájlból, amit a *toXmlFile* nevű függvényem fog elvégezni. Ez szintén a getdata.php fájlban található.

A függvény kódját a 13. ábrán is meg lehet tekinteni, működését pedig azért emeltem ki a szakdolgozatomban, mert több probléma is felmerült implementálása közben.

Az xml.txt fájl tartalmát először is másoljuk ki egy sztringbe, mivel ezt egyszerűbb módosítani, mint magát a fájlt. Elsőnek távolítsuk el a *$#13;* karaktersorozatokat a sztingből. Az XML, HTML és XSLT dokumentumokban a sorvégét, ezekkel a karakterekkel reprezentálják, ASCII és Unicode karakterkészletekben a decimális értéke egyenlő 13-al.

A wsn.com weboldal HTML szerkezetét feltérképezve a FireBug szoftver segítségével, láthatóvá vált számomra, hogy az összes mérkőzést és azok adatait, egy „ti_liveScore”

azonosítóval ellátott elem tartalmazza. A *toXmlFile* függvény a következő lépésben levágja a sztring elejét és végét úgy, hogy csak ez a elem maradjon benne. Ezzel azt próbálom megelőzni, ha bármilyen kód veszélyeztetné a jól-formáltságot, a számomra szükséges részen kívül, akkor az ne befolyásolja a BetGame.hu működését. A *trim* függvény segítségével minden whitespace karaktert[10] törlünk a sztring elejéről és végéről, majd a sztring tartalmát, úgymond keretbe foglaljuk, azaz megadunk egy gyökér elemet a tartalomnak, és az elején egy fejlécet, ami meghatározza az XML dokumentum verzióját és a karakterkódolást.
A végén elmenti a függvény a sztring tartalmát az xml3.txt nevű fájlba.

```
46  function toXmlFile(){
47      $str=file_get_contents("xml.txt");
48      $doc=str_replace(chr(13), '',$str );
49
50      $doc=strstr($doc,'<div id="ti_liveScore"');
51      $veg=strripos($doc,'<!--End live score-->');
52      $doc=substr($doc,0,$veg);
53
54      $doc=trim($doc);
55
56      $doc="<?xml version='1.0' encoding='UTF-8'?>".
57              "<document>".
58                  $doc.
59              "</document>";
60
61      $newfile="xml3.txt";
62      $file = fopen ($newfile, "w");
63      fwrite($file, $doc);
64      fclose ($file);
65  }
```

13. ábra.

Az adatbázis adatinak frissítése

Az xml3.txt fájlban lévő adatokat a PHP, *xpath* függvényével értem el, és adatbázis kezelő függvényei segítségével mentettem az adatbázisba.

[10] **Whitespace karakterek:** szöveg tagolására alkalmas karakterek, mint a szőköz, újsor, tabulátor.

Az *xpath* függvény az XPath útvonalleíró nyelv használatát valósítja meg PHP-ban, viszont nem XML dokumentumokon dolgozik, hanem *SimpleXMLElement* típusú objektumokon. A *SimpleXML* PHP kiterjesztés egyszerű eszközkészletet biztosít arra, hogy egy XML dokumentumot olyan objektumra konvertálhassunk, aminek segítségével már könnyen végezhető el az XML tartalmának feldolgozása. Ez az objektum a *SimpleXML* kiterjesztés által definiált, *SimpleXMLElement* osztályból származtatott objektum. Függvényeivel elérhető az objektum által tartalmazott XML elemek nevei, névterek információi, ezen kívül szintén lekérdezhetőek és be is állíthatóak, a gyermekeik, az attribútumaik és értékeik. XML kód betölthető fájlból, és sztringből is egy *SimpleXMLElement* objektumba, és az *xpath* függvénynek sztring paraméterként megadott XPath kifejezésekkel, bármely eleme illetve elemei leválogathatóak egy *SimpleXMLElement* típusú objektumokat tartalmazó tömbbe.

A getdata.php fájlban három hasonló működésű függvényt hoztam létre, az élő, a következő, és a befejezett mérkőzések adatbázisba való mentésére. Ezen függvények nevei sorban *getLiveMatchesToDB*, *getUpcomingMatchesToDB* és *getCompletedMatchesToDB*. A mérkőzésekről könnyű eldönteni, hogy éppen élőben játsszák, még csak ezután fognak következni vagy már befejeződtek, mivel minden, a wsn.com weboldalon lévő mérkőzést tartalmazó sor rendelkezik egy „status” attribútummal, mely értéke egyértelműen meghatározza ezt. Így a három lehetséges érték a „Live”, az „Upcoming”, és a „Completed”. Mindhárom függvényem elején, miután az xml3.txt fájlt betöltöttem egy *SimpleXMLElement* objektumba, *xpath* függvénnyel leválogatom a megfelelő státuszú meccseket, és egy *foreach* ciklussal végigmenve az elemeken feldolgozom azokat, ahol két esetet kell elkülönítenem:

- ha a mérkőzés már benne van az adatbázisban, akkor az adatbázis műveleteken és ezáltal a feldolgozási időn is spórolva, nem kell minden adatot újra elmenteni,
- ha nincs még benne a mérkőzés az adatbázisban, akkor az adott mérkőzésről minden adatot el kell mentenem.

A mérkőzések adatainak megjelenítése a felhasználó számára

Miután minden adat mentésre került az adatbázisba, a mérkőzéseket meg kell jeleníteni a felhasználó számára a böngészőben. A rendszeremben az adatok nem közvetlenül az

adatbázisból kerülnek a felhasználóhoz, hanem előbb egy általam létrehozott XML dokumentumba mentem őket, és ennek a fájlnak a segítségével jelenítem meg. Az adatbázis matches.xml dokumentumba mentését a getdata.php fájlban található *getDBtoXML* függvény végzi, míg a felhasználónak történő megjelenítést a refresh.php fájlban implementált, *liveMatchesToTableView*, *upcomingMatchesToTableView* és *completedMatchesToTableView* függvényekkel valósítom meg.

Az matches.xml fájl létrehozása feleslegesnek tűnhet, azonban a percenkénti adatfrissítés szempontjából nem az. A magyarázatot a 3.3.6. fejezetben írom le.

```
1   <?xml version="1.0" encoding='UTF-8'?>
2   <document>
3       <live>
4           <country_league class='Oroszország: 1st-division'>
5               <match class='live_166235'>
6                   <starttime>1272546000</starttime>
7                   <matchinfo>75</matchinfo>
8                   <hometeam>FK Nischni Novgorod</hometeam>
9                   <awayteam>FK Krasnodar</awayteam>
10                  <homegoals>4</homegoals>
11                  <awaygoals>3</awaygoals>
12                  <gh>0</gh>
13                  <ga>1</ga>
14                  <odds1>2.57</odds1>
15                  <oddsx>2.84</oddsx>
16                  <odds2>2.67</odds2>
17                  <startlive>1272549609</startlive>
18                  <redcard_h>0</redcard_h>
19                  <redcard_a>0</redcard_a>
20              </match>
21              <match class='live_166246'>
37              <match class='live_166245'>
53              <match class='live_166264'>
69          </country_league>
70      </live>
71      <upcoming>
487     <completed>
529 </document>
```

14. ábra.

A 14. ábrán látható a *getDBtoXML* függvény által létrehozott, és az xml könyvtárba mentett, matches.xml dokumentum szerkezete. A *getDBtoXML* függvényemben arra

törekedtem, hogy a létrejövő XML dokumentum struktúrája úgy épüljön fel, hogy a refresh.php függvényeinek, elemsorrend szerint végigszaladva a matches.xml fájlon, egyszerű legyen a HTML kód generálása. Ezt, egy jó SQL lekérdezés már részben garantálja, amire példát a 15. ábrán láthatunk, az élő mérkőzésekre vonatkozóan. Mivel a labdarúgó szezon közepén egy hétvégi játéknap esetén, akár több mint háromszáz mérkőzés is lehet, fontos a rendezett megjelenítés. A felhasználó így könnyebben találhatja meg a számára fontos adatokat.

A mérkőzések sorrendjét először is az határozza meg, hogy élő, következő, vagy már lejátszották e. Ezután abc sorrendben az ország neve, aztán szintén abc sorrendben a bajnokság neve, majd a kezdési időpont növekvő sorrendje határozza meg a felsorolást. Azonos kezdési időpont esetén a hazai csapatok neveinek abc sorrendje dönt.

```
780     $query="SELECT * FROM matches AS t1,
781                           leagues AS t2,
782                           countries AS t3
783             WHERE status='Live' AND
784                   t1.leagueid=t2.leagueid AND
785                   t2.countryid=t3.countryid
786             ORDER BY t3.countryname,
787                   t1.leagueid,
788                   starttime,
789                   hometeam";
790     $result=mysql_query($query);
```

15. ábra.

3.3.5. AJAX használata a weboldalamon

Bemutató jellegű weboldalak létrehozása szabványos HTTP kérelmekkel kitűnően megvalósítható. Azonban olyan oldalaknál alkalmazva, ahol a felhasználóknak különböző feladatokat lehet megoldaniuk, már körülményessé teszi a működést, és rontja a felhasználói élményt.

Az AJAX használatával olyan weboldalakat készíthetünk, melyek asztali alkalmazás hatását keltik. Mindezt az AJAX-motor szívével, az *XHR* (*XMLHTTPRequest*) objektummal valósíthatjuk meg. Ez az objektum teszi lehetővé, hogy az oldal háttérkérelemként kapjon adatokat a kiszolgálótól (GET eljárás segítségével), illetve

küldjön adatokat a kiszolgálónak (a POST eljárás segítségével), ami azt jelenti, hogy a folyamat során nem frissíti a böngészőt. Az *XHR* használatakor nem kell a kiszolgálóra várni, a felhasználónak nem kell tudnia ezen folyamatokról, ehelyett a szolgáltatást használó aktuális feladatra összpontosíthat.
Minden AJAX-kérelem egy ügyféloldali művelettel indul, amelyet egy JavaScript kód kezel. Egy *XHR* objektum példányosítása után, tagfüggvényei segítségével indíthatunk kérelmeket, figyelhetjük ezek állapotát, és kérhetjük el a kapott válaszok tartalmát.

A BetGame.hu AJAX kérelmeinek kezelését a Prototype Javascript Framework használatával oldottam meg. Ez egy JavaScript nyelven írt, objektumorientált eszközkészlet, mely dinamikus, webes alkalmazások megvalósítását teszi egyszerűbbé. Forráskódja nyílt, tehát bárki szabadon letöltheti a ***http://www.prototypejs.org/*** oldalról, és szabadon használhatja saját alkalmazásában.
Az egész eszközkészlet JavaScript kódját egyetlen fájl, a prototype.js tartalmazza, melyet feltöltve a szerverünkre, már csak include-olni kell a HTML fájlunkba. A 16. ábrán látható index.php kódrészletének első sora tartalmazza a prototype.js fájl betöltését, melyet a HTML <head> elemében helyeztem el. Ezáltal már használható a Prototype által biztosított osztályok objektumai.
A 16. ábrán látható kódban továbbhaladva található az *initialize* JavaScript függvény implementációja. Ez a függvény végzi a mérkőzések percenkénti frissítését, amit egyszer kell meghívnia a rendszernek a felhasználó bejelentkezése után közvetlenül. Erre kiválóan alkalmas a <body> HTML elem, *onload* eseménye, melynek értékeként megadott JavaScript függvény az oldal betöltésekor meghívódik. Tehát a weboldalam az *initialize* függvényt hívja a főoldal betöltésénél (a 16. ábra legalsó sorában látható).
Az *initialize* egyetlen utasítást tartalmaz, mellyel a Prototype által implementált *Ajax.PeriodicalUpdater* osztályt példányosítja. Ha a konstruktorban megadott paramétereket nézzük, már ki lehet találni, hogy az objektum működése mit fog eredményezni. Segítségével, bizonyos időközök elteltével, újra és újra el tudjuk indítani ugyanazon kérést.
Az első paraméterben megadott sztring érték annak a HTML elemnek (ahogy a Prototype fogalmaz: tárolónak) az azonosítója, amelyben szeretnénk megjeleníteni a szervertől kapott választ. A második paraméter, szintén sztring értékben megadva, annak a PHP

fájlnak a neve, amelynek a kérést küldjük, és a harmadik paraméterben az objektum működésének opcióit állíthatjuk be. Az opciókat úgy adtam meg, hogy GET eljárással, 60 másodpercenként, olyan adatokat kérek a refresh.php fájltól, amelyek tartalmazhatnak scriptet is. A *decay* és a *frequency* opciók összetartoznak, amellyel a következőket állíthatjuk be: ha az egymás utáni kérésekre kapott válaszok teljesen megegyeznek, akkor a következő kérés indítása, a *decay* és a *frequency* értékének szorzatából kapott másodpercérték múlva történik. Amint különböző válasz érkezik, a frissítés periódusa visszaáll az eredeti értékre. Mivel én a *decay* opciónak 1-es értéket adtam, ez soha sem lesz hatással a frissítési időközökre.

Ebben a példában láthattuk, hogy milyen egyszerűen lehet bonyolultabb AJAX kérelmeket megvalósítani a Prototype Framework-el.

```
30  <script type="text/javascript" src="ajax/prototype.js"></script>
31  <script type="text/javascript">
32  function initialize(){
33    new Ajax.PeriodicalUpdater(
34          'centertext',
35          'refresh.php',
36          {method: 'get',evalScripts: true,frequency: 60,decay: 1}
37      );
38    }
39  function betf(matchid,odds,which,event){
47  function betdelete(matchid,event,credit){
52  </script>
53  </head>
54  <?php
99  <body onload="javascript:initialize();">
```

16. ábra.

3.3.6. A teljes frissítési mechanizmus

Ha egyszerre csak egy személy használhatná a BetGame.hu weboldalt, akkor elegendő is lenne a 3.3.5. fejezetben Prototype Framework-el megvalósított percenkénti adatfrissítés. Viszont egy időintervallumban több felhasználó is be lehet jelentkezve a saját böngészőjén keresztül, és ekkor a szervernek nem percenként egyszer, hanem az online felhasználók számától függően, akár tízszer vagy százszor is kellene frissíteni az adatbázist. Ez természetesen ahhoz vezetne, hogy a szerver nem tudná kiszolgálni a kliensektől érkező kéréseket, így a játék nem működne.

Ennek a helyzetnek az elkerülésére hozza létre a rendszerem a 3.3.4. fejezetben említett, matches.xml dokumentumot. A refresh.php kódja, az *Ajax.PeriodicalUpdater* objektummal közösen, és a matches.xml segítségével védi a szervert és az adatbázist a túlzott igénybevételtől a következőkben leírt módon.

Az *Ajax.PeriodicalUdater* percenként lefutatja a refresh.php kódját, viszont a refresh.php mielőtt elkezdené a teljes frissítési folyamatot – indulva a wsn.com oldal adatainak kinyerésétől, az adatbázisba mentésen keresztül, a megjelenítésig – megvizsgálja, hogy mikor módosították utoljára a matches.xml dokumentumot. Ha úgy véli, hogy az adatok még frissek, azaz negyven másodpercnél kevesebb ideje volt az utolsó módosítás, akkor a matches.xml fájlból jeleníti meg az oldalon az adatokat.

Mindezzel azt értem el, hogy bármennyi felhasználó is van bejelentkezve az oldalra, negyven és hatvan másodperc közötti időnként, valamelyik felhasználó kérése biztos, hogy elvégzi a teljes frissítési folyamatot, és a többi felhasználó kiszolgálását a matches.xml fájl segítségével oldja meg a szerver.

Az XML dokumentum módosítási idejének vizsgálatakor, nem fontos pontosan negyven másodpercet beállítani, de az értéknek harmincnál többnek, és hatvannál kevesebbnek kell lennie. Így garantálható, hogy a matches.xml ne frissüljön többször is, mire az újra megjelenítésre kerül a felhasználónál.

4. Továbbfejlesztési lehetőségek

A BetGame.hu jelen állapotában is egy használható, kerek-egész online játék. Viszont újabb funkciók integrálásával továbbfejleszthető az oldal, még élvezetesebbé téve a játékot.

Ilyen újítás lehetne az egyes mérkőzésékhez kapcsolódó fórum, vagy chat beépítése. Például egy mérkőzésre kattintva, a felugró ablakban lehetne követni az élő eredményt és rövid üzeneteket küldeni egymásnak, amiben a felhasználók párbeszédet folytathatnának a mérkőzésről, vagy amiről csak akarnak. Ezzel a játék közösségi oldal jelleget is kapna, ami újabb ötleteket ad a fejlesztésre. A felhasználók ismerősnek jelölhetnék egymást, klubokat alakíthatnának (például szurkolói klubokat), belső levelezési rendszeren keresztül kommunikálhatnának.

Egy „Statisztikák” menüpont alatt a legkülönfélébb, felhasználókra, mérkőzésekre, fogadásokra vonatkozó statisztikákat lehetne közölni. Például a nap/hét/hónap legjobb fogadói és legrosszabb fogadói, egy fogadással nyert legnagyobb nyeremények és legnagyobb bukások, a legtöbb felhasználó által fogadott mérkőzések, legtöbb nyereményt hozó mérkőzések és ezek ellentétjei.

A befejezett mérkőzések végleges adatbázisba való mentésével, a felhasználó megtekinthetné fogadásait a regisztráció időpontjáig visszamenőleg. Ezen kívül a bajnokságok tabelláit is nyílván lehetne tartani (csapatonkénti helyezések, megnyert/döntetlen/elvesztett mérkőzések száma, rúgott és kapott gólok, pontok, hazai mérleg, idegenbeli mérleg).

5. Összefoglalás

A szakdolgozatomban szereplő fogadójáték megvalósításához különböző technológiák, illetve technikák együttes használatára volt szükségem. Úgy gondolom, hogy sikeresen vettem a munkám során felmerülő akadályokat, hisz nem minden programnyelvet használtam korábban. Ezek ismeretét a fejlesztés során sajátítottam el, melyeket a gyakorlatban azonnal tudtam használni.

A különböző internetes szakmai fórumokon, az oly sokat dicsért PHP, MySQL és Apache (sokak által csak „szentháromságként" emlegetett) hármas beváltotta a hozzá fűzött reményeimet. Kiválóan alkalmas dinamikus weboldalak létrehozására és fejlesztésére, valamint a JavaScript és az Ajax beépítésével és együttműködésével, tovább fokozható a felhasználói élmény. Az implementációs részben szereplő példák bemutatásával is törekedtem ennek bizonyítására.

A célom tehát sikerült, hisz egy olyan interaktív webes alkalmazást hoztam létre, mely teljes egészében felhasználóbarát és nem utolsósorban mindenki számára elérhető a ***www.betgame.hu/index.php*** oldalon.

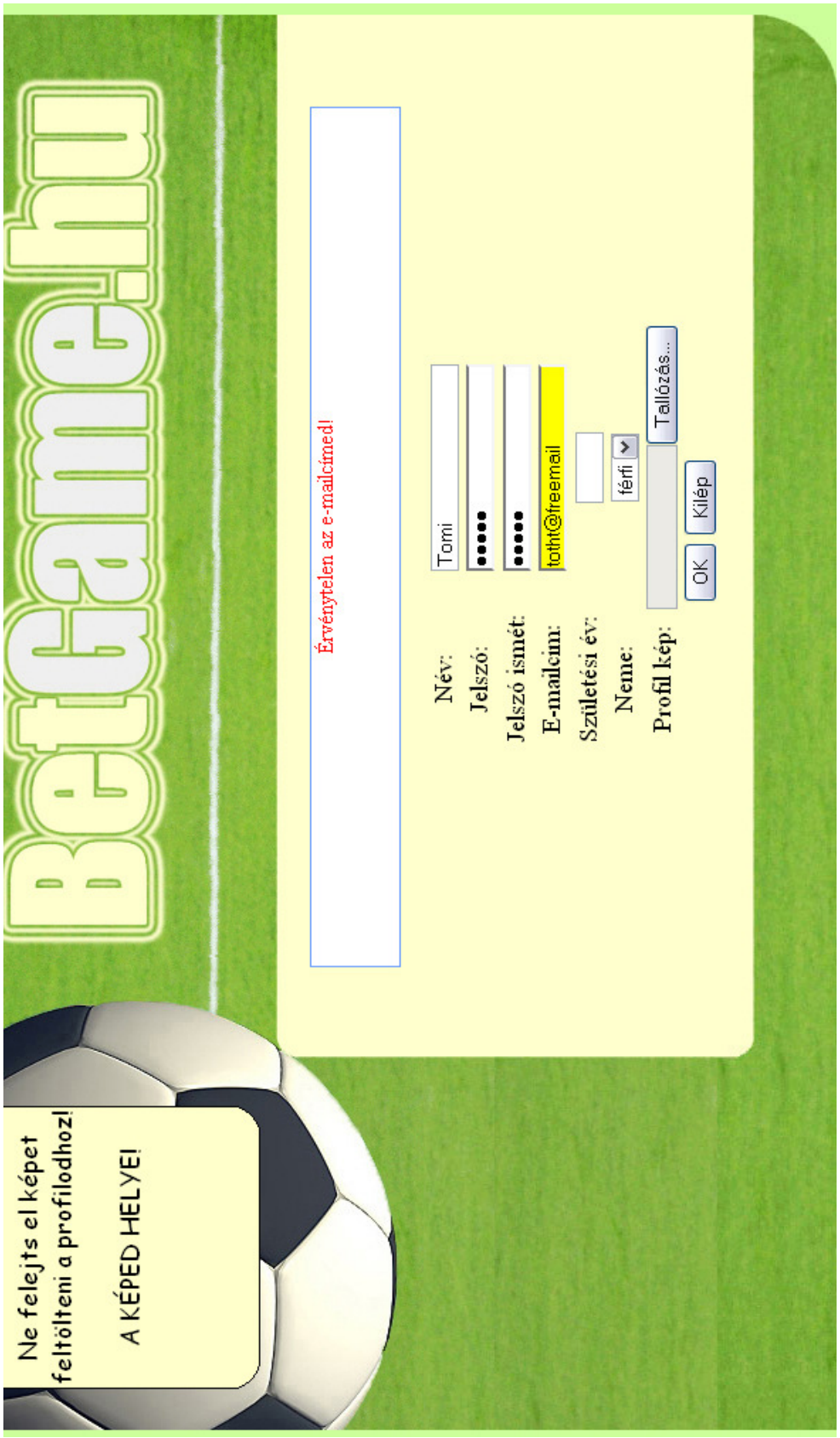
6. Irodalomjegyzék

- Neil Bradley – Az XML-kézikönyv, SZAK Kiadó, 2005
- Virginia DeBolt – HTML és CSS, Webszerkesztés stílusosan, Kiskapu Kft, 2005
- Matt Zanstra – Tanuljuk meg a PHP5 használatát 24 óra alatt, Kiskapu Kft, 2005
- Peter Moulding – PHP Haladóknak, Fekete Könyv, Perfact-Pro Kft, 2002
- Kris Hadlock – Webalkalmazások fejlesztése AJAX segítségével, Kiskapu Kft, 2007
- Julie C. Meloni – Tanuljuk meg a MySQL használatát 24 óra alatt, Kiskapu Kft, 2003
- Juhász István – Rendszerfejlesztési technológiák tárgy jegyzete
- http://www.w3schools.com/
- http://www.php.net/manual/en/
- http://dev.mysql.com/doc/

7. Függelék

7.1. Bejelentkezés

7.2. Regisztráció



7.3. Főoldal


BetGame.hu
Adataim megváltoztatása Kilépés
Az adatok frissítésének ideje: 2010.05.02-14:10:29
ÉLŐ:
Anglia: championship
14:00 11 Blackpool FC 0:0 Bristol City 1.46 4.08 6.78
14:00 14 Coventry City 0:0 Watford FC 2.15 3.28 3.27
14:00 14 Derby County 0:0 Cardiff City 2.43 3.27 2.77
14:00 14 Ipswich Town 0:0 Sheffield United 2.06 3.31 3.46
14:00 14 Leicester City 0:0 Middlesbrough 2.12 3.28 3.32
14:00 14 Plymouth Argyle 0:0 Peterborough United 1.89 3.4 3.96
14:00 14 Queens Park Rangers 0:0 Newcastle United 3.08 3.34 2.21
14:00 14 Reading 0:0 Preston North End 1.67 3.56 5.02
14:00 14 Scunthorpe United 0:0 Nottingham Forest 2.74 3.32 2.43
KREDITEID
1211 kredit áll rendelkezésedre,
100 kredited van játékban,
409 kredit a várható nyereményed!
FOGADOTT MECCSEID
Liverpool FC 14:30 Chelsea FC 409
Mtk Hungaria FC 2:3 Debreceni VSC
Manchester City 3:1 Aston Villa
RCD Espanyol 0:2 CF Valencia
Villarreal CF 1:4 FC Barcelona
Tomi