Debreceni Egyetem
Informatikai Kar

A WATS alkalmazás fejlesztése

Diplomamunka

Témavezető:
Kósa Márk
egyetemi tanársegéd

Készítette:
Kozma Zoltán
programtervező matematikus

Debrecen
2010

Tartalomjegyzék

1. Köszönetnyilvánítás

Köszönöm oktatóimnak, hogy olyan sokat tanulhattam tőlük!

Köszönöm Kósa Márknak, hogy elvállalta a diplomamunkám vezetését.

Köszönöm Uzonyi Nikolettának a sok hasznos tanácsát, továbbá azt is, hogy átolvasta és véleményezte a diplomamunkámat.

Köszönöm Asztalos Zsoltnak és Bónizs Attilának a sok közös tanulást és a vizsgákra-, szigorlatokra való közös felkészülést.

Köszönöm a családomnak, hogy lehetővé tették, hogy a Debreceni Egyetemen tanulhattam.

2. Bevezetés

2.1. A WATS jelentése

A WATS egy mozaikszó, mely az angol Web-based Automatic Testing System kifejezésből ered, ami magyarul Web-alapú Automatikus Tesztelő Rendszert jelent.

2.2. Diplomamunkám témája

Jelen dolgozat egy konkrét alkalmazás fejlesztését írja le, melyet a következő programozási nyelvek segítségével valósítottam meg: PHP, JavaScript, HTML és CSS. Az alkalmazás célja, hogy az egyetemen tanított „*Programozás 1-2.*" és „*Magas szintű programozás nyelvek 1-2.*" tantárgyak beküldendő házi feladatait a hallgatók webes felületen tudják beküldeni, ott tesztelni, a tanárok pedig az általuk kiírt feladatokat automatikusan tudják leellenőriztetni e program segítségével.

2.3. Témaválasztás

Az egyetemi képzés évei alatt főként az elméleti tudásra helyeződik a nagyobb hangsúly a programtervező matematikus hallgatók számára. Gyakorlati tudást, illetve az elméletben tanultak alkalmazását csak néhány speciális esetben van lehetőség kipróbálni. Az egyik ilyen a „*Rendszerfejlesztés technológiája*" c. tárgy, a másik a diplomamunka. Témaválasztásomban is ez játszott szerepet. Megpróbáltam követni egy teljes alkalmazásfejlesztési módszert a tervezéstől kezdve az implementáción át a tesztelésig.

Egyetemi éveim alatt számos programozási paradigmát illetve nyelvet ismertem meg és sajátítottam el. A világ minden részén egyre inkább megjelennek az „e-szolgáltatások", azaz mindent az interneten keresztül próbálnak lebonyolítani a banki átutalásoktól kezdve a tantárgyakra- és vizsgákra való jelentkezéseken át a különböző termékek vásárlásáig. Jelen diplomamunkám során az egyetemen e kevésbé

hangsúlyos, ellenben a világban erősen jelen levő webhez kötődő programozási nyelvek egyikével kívánok foglalkozni. A – jelenlegi – álláshirdetések nagy részében is ezen programozás nyelvek alapvető ismeretét követelik meg.

A fentebb felsoroltak miatt is döntöttem úgy, hogy diplomamunkámként ezzel a területtel foglalkozom.

3. Áttekintés

3.1. Webes működtető technológiák

Legelőször is meghatározom a főbb működtető technológiákat és irányelveket, amelyeket a munkám során alkalmazni fogok, majd az alpontokban kifejtem, hogy a választásom miért arra a technológiára esett, röviden jellemzem azt, és bemutatom egy-egy rövid kódrészlettel.

Markup	XHTML 1.0 Strict
Hypertext Preprocessor	PHP 5
Styles	CSS 3
Client-side Scripting	JavaScript 1.2

3.1.1. HTML/XHTML

A napjainkban napról-napra megjelenő újabb és újabb böngészők kihívást jelentenek a webes programozásnak mind működési-, mind megjelenítési területén. Ez az XHTML 1.0 Strict, amelyre neve is utal, a legszigorúbb megkötéseket alkalmazza, amely lehetőséget teremt a legmagasabb böngésző-kompatibilitási elvárások minél jobb megközelítésére.

A kód kinézete:

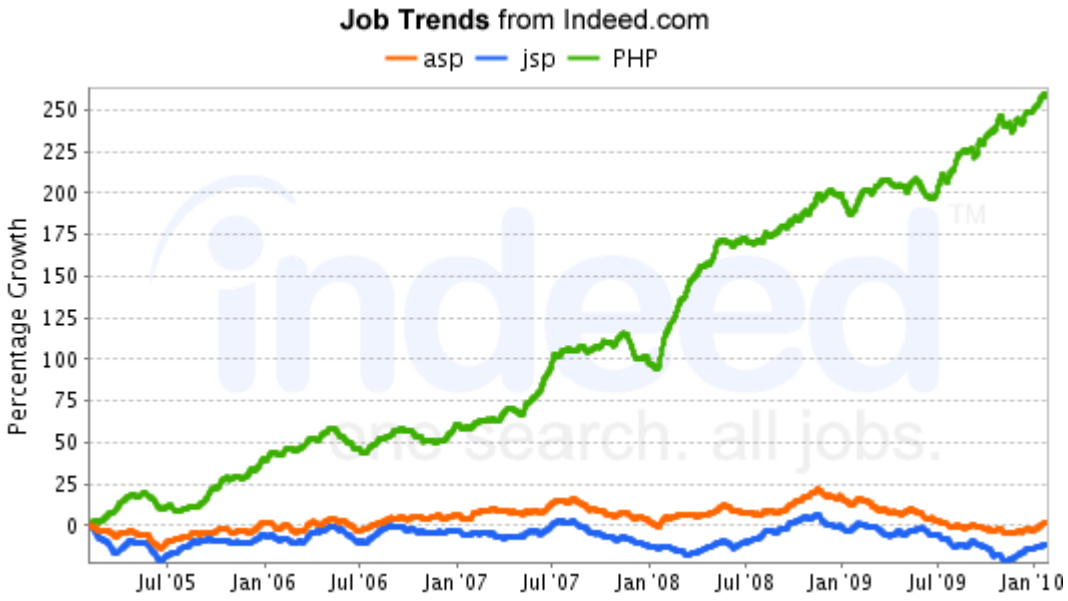
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Strict//EN" "http://www.w3.org/TR/xhtml/DTD/xhtml1-
strict.dtd">

<html xmlns=”http://www.w3.org/1999/xhtml”>
<!-- HTML comment -->
</html>
```

3.1.2. PHP

A PHP a legelterjedtebb webes programozási nyelv. A legtöbb internetes oldal és alkalmazás ebben a nyelvben van megírva. Szintaktikában nagyon hasonlít a Perl, illetve a C nyelvhez.

A nyelv Rasmus Lendorf nevéhez kötődik. A fejlődése kezdetén csak CGI-programok egy halmaza volt. Később kiegészítette ezt egy Form Interpreter (Űrlap Feldolgozó) alkalmazással, így a kettőből jött létre a PHP/FI. Az új változatot Lendorf már C nyelven írta és 1995-ben kiadta az első nyilvános változatát.

A C nyelvhez képest sokkal kevesebb megkötést alkalmaz. Gyengén típusos nyelvek közé tartozik. Viszonylag kis webes alkalmazásokat rövid idő alatt elkészíthetők vele. Amiért a PHP ennyire elterjedt az valószínűleg a fentebb felsoroltakon túl még az ingyenessége és a könnyen tanulhatósága miatt következett be.



1. ábra: Az asp, jsp és a php nyelvek az álláshirdetések beli változása[1]

Több verziója is létezik, melyek közül – a dolgozat elkészítése idején – a PHP 5-ös verziója a legújabb. Diplomamunkám elkészítése során is ezt alkalmazom.

Habár az alap OO már a PHP 3-mas és a PHP 4-es verziójában is jelen volt, de csak mint „primitív típus”; az igazi áttörést a PHP 5-ben érték el, amikor is teljesen újraírtak mindent, ami az objektumorientáltságra vonatkozott. Olyan új – egyébként az OO-ban alapvető – eszközökkel bővítették ki, mint a konstruktor, destruktor, kivételkezelés, illetve a változóknál meg lehet adni a privát, protected és

[1] http://www.indeed.com/jobtrends?q=asp%2C+jsp%2C+PHP&l=&relative=1

public mivoltát is. Megjelentek az absztrakt osztályok és metódusok, illetve a final osztályok és metódusok.

A kód kinézete:

```
/**
 * Többsoros megjegyzés
 */
class MyClass implements IBaseInterface
{

    private $field;

    public function __constructor($param)
    {
        $this->field = $param;
    }

    // Egysoros megjegyzés
    public function MyMethod()
    {
        return $this->field;
    }
}
```

3.1.3. CSS

Maga a szó egy mozaik szó, mely az angol Cascading Style Sheets szavakból származik. A CSS egy stílusleíró nyelv, mely XHTML, illetve HTML típusú strukturált dokumentumok, weblapok megjelenítését írja le. Elsődleges célként talán a modularitását hozhatjuk fel. Szétválasztja a dokumentum illetve a weblap megjelenítését magától a tartalomtól.

A CSS egyszerű szintaxissal rendelkezik, csak néhány angol nyelvű kulcsszót használ a stílusok tulajdonságaihoz. 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.

Elhelyezhetjük külön fájlban, esetleg a html dokumentum head részében deklarálva, vagy magánál a html elem style tulajdonságánál.

A kód kinézete:

```
body, html {
    background-color: #ECEFFF;
}

img {
    border-width: 0px;
}

.container {
    postition: absolute;
    width: 1000px;
    left: 50%;
    margin-left: -500px;
}
```

3.1.4. JavaScript

Ez egy objektum alapú szkript nyelv. A Netscape Communications mérnöke, Brendan Eich fejlesztette ki. Eredeti neve Mocha volt, később váltott LiveScript-re, végül pedig felvette a ma is ismert JavaScript nevet, mikor a szintaxisa egyre közelebb került a Sun Microsystems Java programozási nyelvéhez. Először az Európai informatikai és kommunikációs szabványosítási szövetsége (ECMA International[2]) szabványosította 1997-1999 között. A jelenleg is érvényes szabvány az ECMA-262 Edition 3, ami a JavaScript 1.5-nek felel meg. Ez a szabvány egyben ISO szabvány is.

A JavaScript lényegében a szerverekre ható nagy terhelést – a kliens oldal erőforrását kihasználva – hivatott csökkenteni a JavaScript. A JavaScript kód vagy a html fájlban vagy külön (jellemzően .js kiterjesztésű) szövegfájlban van.

A JavaScripthez manapság már több framework[3] is létezik, melyek közül én a MooTools[4]-t választottam.

[2] European Computer Manufacturers Association - European association for standardising information and communication systems
[3] framework = keretrendszer
[4] MooTools = My Object Oriented JavaScript Tools

A kód kinézete:

```
var Class = new Class({
    initialize: function() { },
    // more methods and functions..
});

Class.implement(new Events);

var myClass = new Class();
```

3.2. Adatbázis

Az interneten elérhető ingyenes adatbázis kezelő rendszerek közül választunk egyet:

DBMS	Sun - MySQL 5.3

3.2.1. Sun - MySQL

A MySQL egy többfelhasználós, többszálú, SQL-alapú relációs adatbázis-kezelő szerver.

A szoftver eredeti fejlesztője a svéd MySQL AB cég, amely kettős licenceléssel tette elérhetővé a MySQL-t; választható módon vagy a GPL, vagy egy kereskedelmi licenc érvényes a felhasználásra. 2008 januárjában a Sun felvásárolta 800 millió dollárért a céget. 2010. január 27.-én a Sun-t felvásárolta az Oracle Corporation, így a MySQL is Oracle tulajdonba került.

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.

3.3. Plugin-ok, Patch-ek és Template Engine-ek

A fentebb felsorolt tulajdonságok és jóságok ellenére a PHP-től kezdve a MySQL-nek is vannak kevésbé jó tulajdonságai. A következő néhány alpontban ezekre a rossz tulajdonságokra adok egy-egy lehetséges megoldást.

3.3.1. eAccelerator

A PHP azon rossz tulajdonságát hivatott kiküszöbölni, mely szerint minden php fájl meghívásakor a forrás kód, minden egyes alkalommal lefordul. Az eAccelerator egy teljesen ingyenes, nyílt forráskódú, PHP gyorsító és optimalizáló.

Az eAccelerator a PHP MMCache kiterjesztéséből származik. Bájtkódú gyorsítótárazást tesz lehetővé. Legtöbbször az osztott memóriába- vagy közvetlenül a lemezre cache-el; így segít a szerver terhelését enyhíteni.

3.3.2. MemCached

A memcached eredetileg a Danga Interactive által lett kifejlesztve a LiveJournal számára, de manapság már szinte minden nagyobb oldal használja. Legtöbbször azon weboldalak gyorsítására alkalmazzák, mely sokszor – esetleg állandóan – adatot kér le az adatbázisból; és ezen adatbázis objektumok gyorsítótárazását végzik a memóriában.

A memcached egy hatalmas hash tábla, amelyet akár több gép között is megoszthat – ezzel, tovább – enyhítve szerverek terhelését. Egy kulcs-érték párokból álló táblát hoz létre, melyben a kulcs indexelt és általa könnyen elérhető az érték. Ezt a táblát a RAM-ban tárolja, mely a lemezolvasáshoz képest többszörös gyorsaságú írást és olvasás tesz lehetővé.

3.3.3. vlibTemplate

A template motorok feladata, hogy elkülönítse a megjelenítést magától a feldolgozástól, esetleges egyéb működésektől. Jelen esetünkben tehát egy template engine-nek az a feladata hogy a PHP-t elkülönítse a HTML/XHTML-től.

Ez egy PHP-ban írt template engine. Alapvetően kettő részre oszthatjuk azon fájlokat, melyekkel tehát foglalkozni fogunk. Az egyik a „template” fájl a másik pedig a „php” fájl.

A template kód kinézete:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Strict//EN" "http://www.w3.org/TR/xhtml/DTD/xhtml1-
strict.dtd">

<html xmlns=”http://www.w3.org/1999/xhtml”>

<head>
 <title>
      {tmpl_var name=’title_szoveg’}
 </title>
</head>

<body>
 <div>
      {tmpl_var name=’body_szoveg’}
 </div>
 <div>
      {tmpl_if name=’if_felt’ op=’>’ value=’5’}
           <p>Igaz</p>
      {tmpl_else}
           <p>Hamis</p>
      {/tmpl_if}
 </div>
      <ul>
           {tmpl_loop name=’ciklus’ }
                <li>{tmpl_var name=’_0’}</li>
           {/tmpl_loop}
      </ul>
 </div>
</body>
</html>
```

Ezen template kód példán bemutatom, hogy hogy kell használni magát ezt a vLibTemplate engine-t:

```
require_once ('vlib/vlibTemplate.php');

$mytemplate = new vlibTemplate('templates/alap.html');

$mytemplate -> setVar ('title_szoveg', 'Welcome');
$mytemplate -> setVar ('body_szoveg', 'Hello World!');

$mytemplate -> setVar ('if_felt', 3);

$mytemplate -> newLoop ('ciklus');
for ($i = 0; $i<100; $i++);
{
   $mytemplate -> addRow (array($i));
}
$mytemplate -> addLoop ();

$mytemplate -> pparse ();
```

4. Felhasználók

Ezen webes alkalmazás elsősorban az egyetemen tanuló hallgatóknak és az ott tanító tanároknak, gyakorlatvezetőknek készült, hogy segítse mindkét felet a Programozás 1 és 2 tantárgyakhoz kapcsolódó házi feladatok beküldésében, illetve ellenőrzésében. Az alkalmazást felhasználói szempontból alapvetően három részre osztom. Vannak hallgatók, tanárok és adminisztrátorok. Ezen csoportok lesznek a program felhasználói.

4.1. Jogkörök

A fentebb felsorolt felhasználók nem egyenrangúak a programban. Habár az összes felhasználó a házi feladat beküldése miatt fogja használni ezen webes alkalmazás, de mindenkinek megvan a maga jogköre. A felhasználói csoportokat szét kell választanunk a különböző hozzáférési szintek alapján.

4.1.1. Hallgatók jogköre

A hallgatók alapvetően a feladatok olvasása és a programok feltöltése miatt fogják használni ezt a webes szoftvert. Továbbá lehetőséget kell teremteni a hallgatóknak a program tesztelésére is a végső beadás előtt. Azt is meg kell engedni a felhasználóknak, hogy megtekintsék a korábbi feladatokat valamint a hozzájuk tartozó eredményeket. A korábbi feladatoknál is lehetőséget kell teremteni a programjuk tesztelésére, de itt már az összes tesztesettel.

Egy hallgatónak ne legyen joga, feladatot kiírni, módosítani annak bármely részét. Továbbá ne tudjon teszteseteket létrehozni. Ne tudja tesztelni a programját a nem kifejezetten erre a célra létrehozott tesztesetekkel. Minden hallgató csak és kizárólag a saját eredményét láthatja, tehát nem engedhetjük meg, hogy a csoporttársai eredményeit megtekintse.

4.1.2. Tanárok jogköre

A tanárok, gyakorlatvezetők tudják olvasni az éppen aktuális feladatokat. Tudjanak kiírni új feladatot, és ahhoz vagy a meglévő aktuálisakhoz tudjanak új tesztesetek megadni, hibásakat pedig törölni. Minden tanár lássa a csoportja névsorát, a lezárt és ellenőrzött beadandóknak az eredményét személyenkénti lebontásban.

A gyakorlatvezetők ne tudják globálisan ellenőrizni a feladatokat az egész évfolyamra nézve. Csak az eredményt kapják meg.

4.1.3. Adminisztrátorok jogköre

Az adminisztrátoroknak két fontos feladata van. Az első a hallgatók regisztrálása, illetve importálás a neptunból. A második pedig az, hogy a tanárok által kiírt feladatokra beküldött programokat globálisan ellenőrizzék az összes tesztesetre.

5. Követelmények

Mielőtt hozzákezdünk megvalósítani egy rendszert, tisztába kell lennünk azzal, hogy pontosan mit is kell megvalósítanunk. Ismernünk kell a rendszerrel szemben támasztott elvárásokat. A követelmény egy rendszer szolgáltatásai és a rájuk vonatkozó szolgáltatások együttese.

5.1. Megvalósíthatósági tanulmány

A megvalósíthatósági tanulmány, terv egy olyan dokumentum, amely általános leírást tartalmaz az elkészítendő rendszerről. Ezen tanulmány alapján döntjük el, hogy egyáltalán, elindul-e a megvalósíthatósági folyamat, lesz-e második lépés. Alapvetően két kérdésre kell választ adnunk, egyrészt, hogy a kifejlesztendő rendszer támogatja-e az általános célkitűzéseket és igényeket. Másrészt pedig azt, hogy egyáltalán megvalósítható-e a vízióban elképzelt rendszer az adott idő- és költségkereten belül.

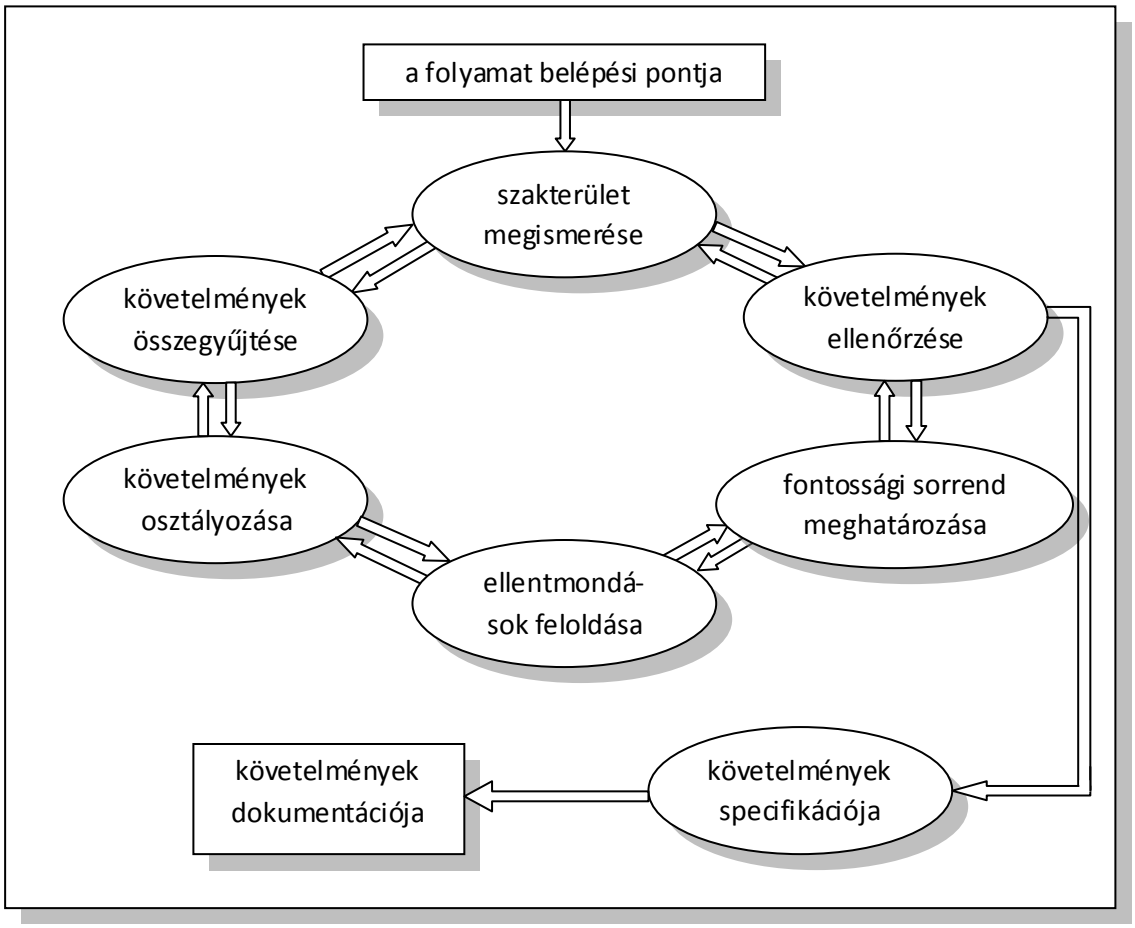
Az első lépés a megvalósíthatósági tanulmány elkészítéséhez, hogy megfelelő információkat kell beszereznünk. Fel kell kutatni, hogy kitől vagy kiktől szerezhető be az információ, és be is kell szerezni azokat. Legtöbbször ilyenkor a vízió megbízójával kell konzultálni.

A megvalósíthatósági tanulmány lezárásaként el kell dönteni, hogy a mit várok el a rendszertől és mit nem. A felhasználó és a fejlesztő ezen információk tudatában dönt, hogy elindítható-e a projekt.

5.2. Követelmények feltárása, elemzése

Ha a megvalósíthatósági elemzés arra a következtetésre jutunk, hogy elindulhat a projekt, akkor a következő lépés a követelmények feltárása és elemzése. A követelmények feltárása és elemzése a felhasználóktól, managerektől, rendszergazdáktól - együttes nevükön, kulcsfiguráktól - történő információ beszerzését és elemzését jelenti. Itt általában problémába szoktunk ütközni.

A kulcsfigurák általában nem tudják, hogy mit várnak el a rendszertől, legalábbis nem tudják megfogalmazni. Vagy nagyon triviális dolgokat kérnek, vagy megoldhatatlan dolgokat. A legnagyobb kihívást a kommunikáció jelenti a szakterület képviselője és a fejlesztő között, mivel mindketten a saját szakterületükön kialakult szakmai szlenggel dolgoznak. Ezt a problémát a fejlesztő hidalja át azzal, hogy megtanulja az adott szakterületi szlenget és terminológiát. A következő akadályt azt jelenti, hogy a különböző kulcsfigurák, különböző igényekkel rendelkeznek, melyek átlalában ellentmondásosak. Természetesen külső körülmények is befolyásolják a követelményeket, ilyen pl az üzlet-, vállalati-, illetve maga a nemzeti politika is. A fentebb leírt problémák esetlegesen plusz időt vesznek igénybe, de ezekre mind-mind tekintettel kell lennünk.



2. ábra: Követelmények feltárása és elemzése - folyamatábra

A követelménye feltárás egy iteratív folyamat, melyben az egyes lépések összefüggnek egymással. A szakterület megismerése, mint már

említettem a fejlesztő feladata. A követelmények összegyűjtésének lényeges momentuma, hogy azt a fejlesztő, illetve a megrendelő együttesen tegye. A követelmények osztályozása az a lépés, amely alapvetően befolyásolja a tervezést, illetve nagyban segít az ellentmondások eliminálásában, valamint a követelmények fontossági sorrendjének meghatározásában.

A megvalósíthatósági eredményeképpen megszületik a követelmény dokumentáció.

5.3. Alkalmazásom követelményei

Jelen esetben számomra könnyű volt a követelmények feltárása és elemzése, mivel egyszemélyben vagyok mind felhasználó és fejlesztő. Tudom, hogy mire kell ügyelni és figyelni a Programozás 1 és Programozás 2 házi feladatok alkalmazásánál. A feladatok és jogkörök tisztázása után még jó néhány irányelvet meg kell jelölni.

5.3.1. Licensz

Az alkalmazást csak és kizárólag ingyenes programok felhasználásával és ingyenes futtató környezetben kell elkészítenem. Úgy vélem, hogy a programnak is ingyenesnek és szabadon terjeszthetőnek kell lennie, azaz GNU-nak.

GNU General Public License (rövid neve GPL, magyarul: GNU Általános Nyilvános Licenc) egy általános célú nyílt forráskódú licenc, amelyet a Free Software Foundation (FSF) tervezett a GNU projekt programkódjaihoz.

5.3.2. Hozzáférhetőség - WAI

Gondolnunk kell hátrányos helyzetűekre is, mint pl. a gyengén látókra, akik speciális eszközökkel és böngészővel nézik a weboldalakat.

A World Wide Web konzorcium (W3C) megalkotta a WAI-t (Web Accessibility Initiative), amely egy kezdeményezés, hogy törekedjünk a

weboldalak minél könnyebb és egyszerűbb felépítésére, hogy a hátrányos helyzetűek is akadálymentesen használhatják azokat. A fogyatékkal élő emberek nem csak a weboldalak olvasásakor, de már a számítógép használatakor is sok akadályba ütköznek. Azzal, hogy a weboldalunkat egyszerűbbé és könnyebben hozzáférhetővé tesszük nem csak a hátrányos helyzetűek speciális eszközei számára lesz használhatóbb a weboldalunk, de a mobil eszközökkel is kompatibilisebb lesz, a kisebb erőforrásigény miatt.

5.3.3. Moduláció – WML, HTML

Az alkalmazást minél modulárisabban kell megvalósítani. Ez annyit tesz, hogy egy MVC (Model-View-Controller) struktúrát kell követni.

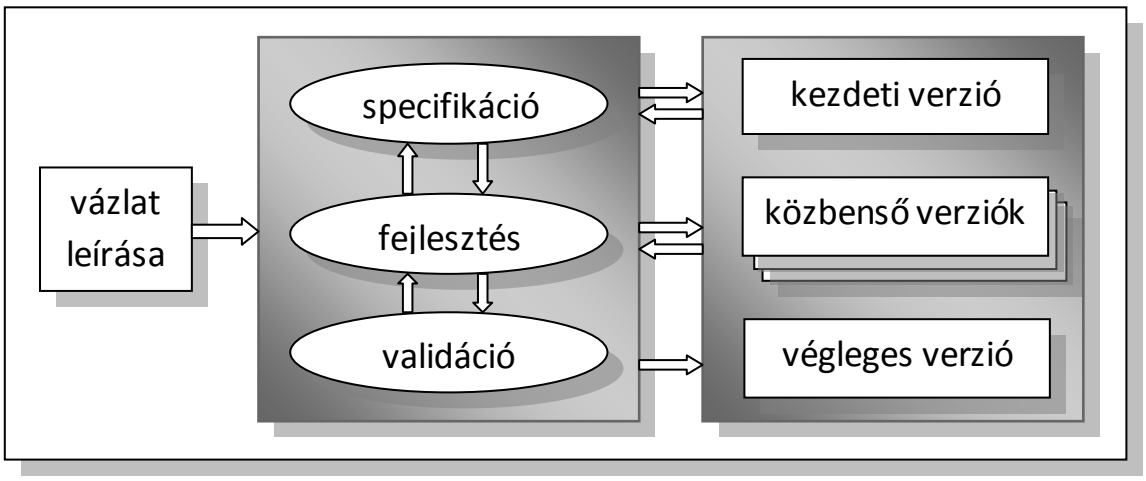
Az a tény hogy külön választottam magát a php-t és a megjelenítést már egy lehetőséget teremt arra, hogy néhány dolog megváltoztatásával mobil eszközökre is alkalmas megjelnítendő felültet tudok létrehozni. A modellt lényegében a Libary könyvtárakban található osztályok reprezentálják. A view részt pedig a Templates könyvtárban található template-ek valósítják meg. A Controllereket pedig a php fájlok.

6. Fejlesztés

Az alkalmazás megvalósítása során az evolúciós modellt alkalmaztam.

6.1. Az evolúciós modell

A vízesés modell mintegy tagadásaként alakult ki az 1980-as években. A kialakítása két ötlet köré épült. Az első alapötlet az volt, hogy ebben a modellben a vízesés modell linearitásával szemben a párhuzamosításra helyeződik a hangsúly. Megpróbáljuk párhuzamosítani az egyes lépéseket. A másik alapfelvetés pedig az volt, hogy egy nagyon korai implementációt készítünk, majd ebből kiindulva újabb-és-újabb implementációk során jutunk el a végső verzióhoz. Így tehát létrehozunk egy verziósorozatot.



3. ábra: Evolúciós fejlesztés modellje

A modellben kezdetben rendelkezésünkre áll a követelményrendszer, majd elkészítjük a követelmény specifikációt. Ezek után implementálunk és validálunk. Így kapunk egy kezdeti implementációt. Ezeket a lépéseket ismételve az különböző közbülső implementációkat kapunk, majd a végén megkapjuk a végleges változatot. A modellben ezeket nevezzük prototípusoknak. A gyors visszacsatolás a párhuzamosságban és a verziókezelésben rejlik.

A következő két alpontban ezen prototípusokat jellemzem.

6.1.1. A feltáró prototípus

A követelmények pontosításában és finomításában a felhasználó és a fejlesztő közösen vesz részt. Egy kezdetleges, működő verziót kap készhez először a felhasználó és ennek megfelelően tudja a követelményeit pontosítani. Ezen pontosítások figyelembevételével készítjük el a következő prototípust, ami már közelebb áll a felhasználó elképzeléseihez. Ezen lépések ismétléseivel létrehozunk egy prototípus sorozatot, amelyekbe egyre több felhasználói igény épül be, míg nem eljutunk a tényleges követelményeknek megfelelő verzióig.

Tehát az első verzió csak az egyértelmű követelményeket foglalja magába, amelyek világosak mind a felhasználó, mind a fejlesztő számára. A kevésbé ismert, komplexebb részek csak a későbbi prototípusokba kerülnek bele. Az így lépésről-lépésre létrejövő verziók egymásra épülnek.

6.1.2. Az eldobható prototípusok

Ebben az esetben a prototípussal a felhasználót segítjük a követelményrendszer minél pontosabb megfogalmazásában. Elkészítünk egy prototípust és azt átadjuk a felhasználónak, hogy eldöntse, hogy a rendszer mely részeit szükséges újradefiniálnunk. Létrejön egy újabb prototípus, amely közelebb áll a felhasználó igényeihez, így az előzőt eldobhatjuk.

Ezen eldobható prototípus alapú fejlesztési modellben tehát a felhasználót oly módon segítetjük, hogy a legkevésbé egyértelmű elvárásokat is már megvalósítjuk az első verzióban. Ezt használja fel a felhasználó a saját igényeinek pontosítása érdekében.

6.1.3. Az evolúciós modell értékelése

Mint minden modellnek az evolúciós modellnek is megvannak a maga előnyei és hátrányai.

Az előnyei közé sorolható, hogy a felhasználó viszonylag gyorsan kap egy prototípust, amelyet hamar ki tud próbálni, és ezáltal a saját igényeit, követelményeit folyamatosan tudja finomítani. Modulárisabbá válik az egész fejlesztési folyamat, nem kell teljes rendszereket, alrendszereket kipróbálni, hanem elég csak bizonyos kisebb rendszerelemeket. A rendszer fejlesztése során a funkcionalitás egyre bővül.

A rendszer nagy hátránya viszont, hogy a teljes fejlesztési folyamat nem látható. Kevésbé lesz strukturáltabb a rendszerünk. Több anyagi- és emberi erőforrásra van igény a folyamat során, pl. szükséges speciális ismeretek, szakemberek és esetlegesen speciális eszközök, mivel a modell központjában a gyors prototípusfejlesztés áll.

6.2. UML diagramok

Az UML[5] egy szabványos, általános célú modellező nyelv, melynek segítségével szöveges és grafikus modelleket készíthetünk.

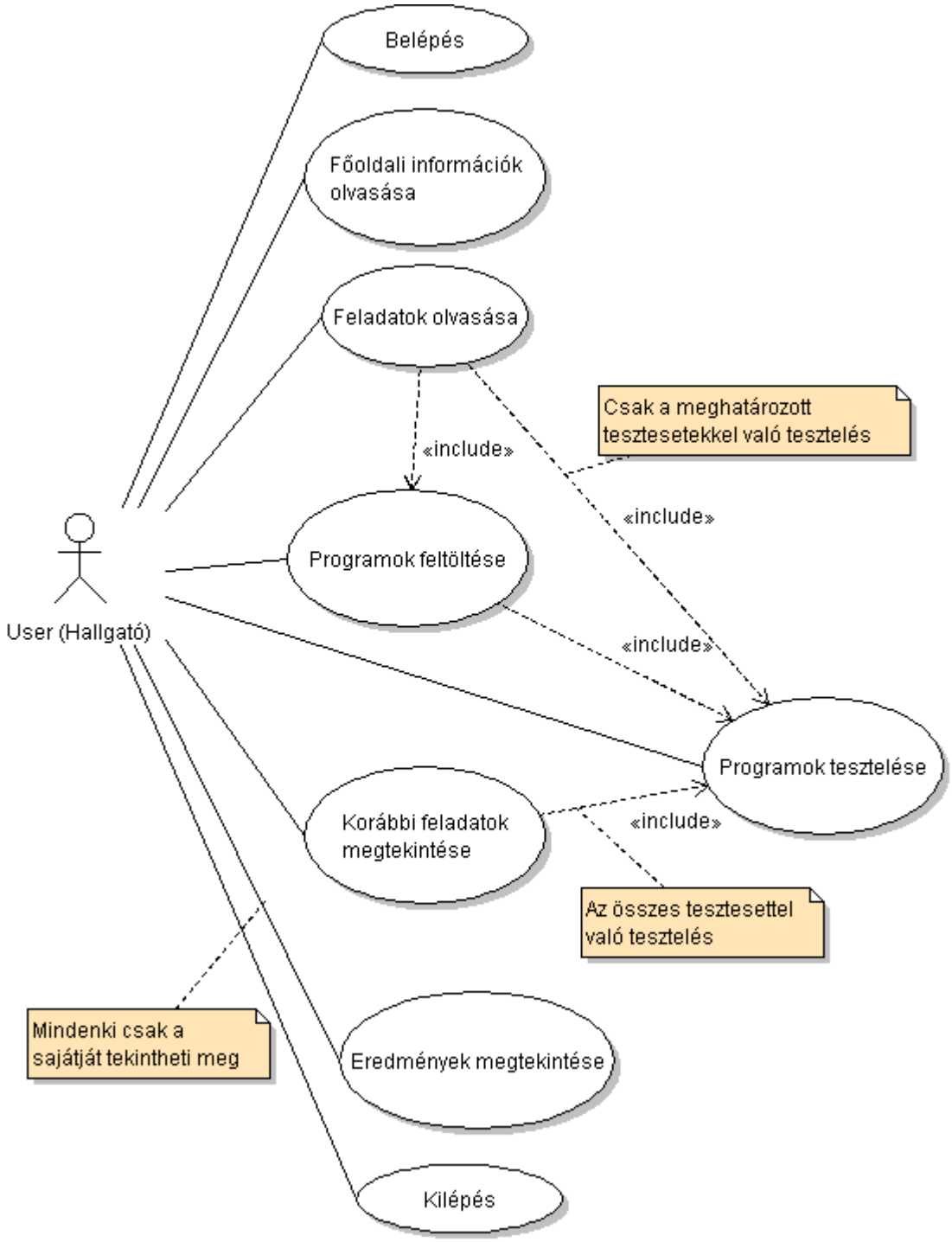
Modellezhetünk többek között rendszereket, szervezeteket, ezeknek a viselkedését, és egymáshoz viszonyított kölcsönhatását; továbbá különböző szereplőkről, azok egy-egy rendszerbeli viselkedéséről. Szokás még használni üzleti- tevékenységek és folyamatok modellezésénél is. Nem utolsó sorban pedig szoftvereknél és programoknál. Az UML az objektumorientált programozás szabványos specifikációs nyelve.

Bár az UML széles körben elfogadott és használt szabvány, gyakran kritizálják két ok miatt. Az egyik az, hogy túl nagy és túl bonyolult, mivel túl sok diagramot tartalmaz, melyeknek egy jó részét alig használják, jó része pedig redundáns. A másik ok pedig a pontatlan szemantika, mivel az UML szemantikát részben OCL-lel, részben angol nyelven, részben az UML-lel magával definiálják, és hiányzik a formális nyelveknél megszokott szigorú definíció.

[5] UML = Unified Modeling Language; magyarul: Egységes Modellező Nyelv

6.2.1. User használati eset diagram

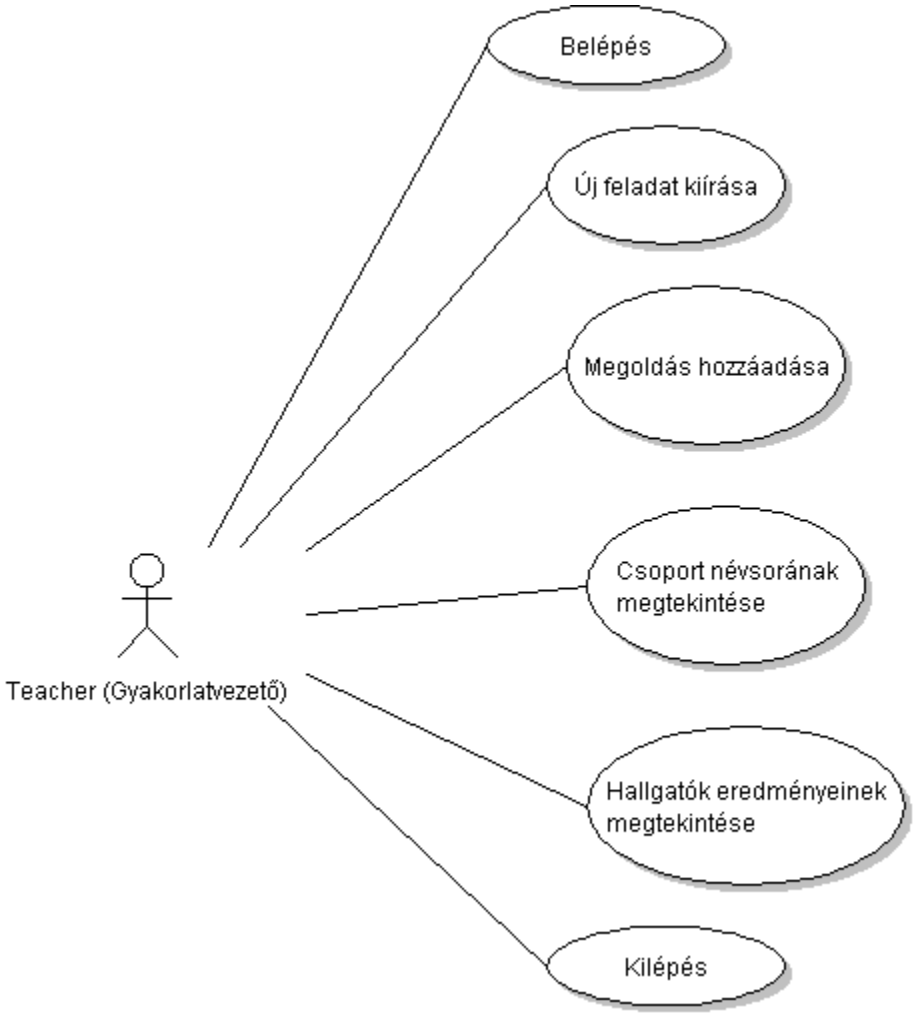
Az alábbi ábra a User, jelen esetben egy hallgató use case; azaz használati eset diagramját mutatja be.



4. ábra: User use case diagram

6.2.2. Teacher használati eset diagram

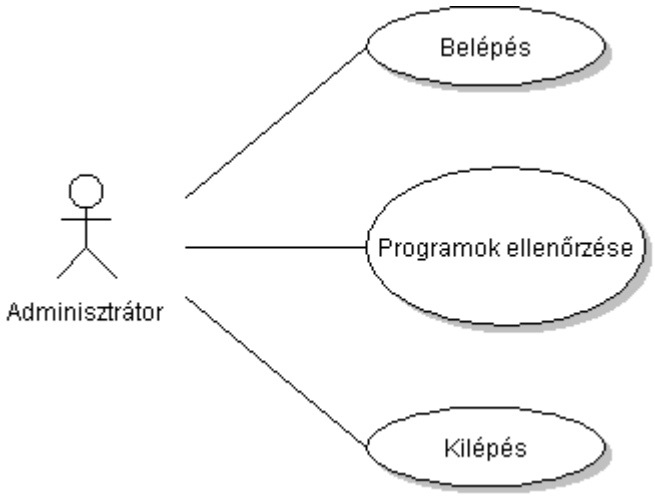
Az alábbi ábrán látható egy tanári user case, azaz használati eset diagram.



5. ábra: Teacher use case diagram

6.2.3. Admin használati eset diagram

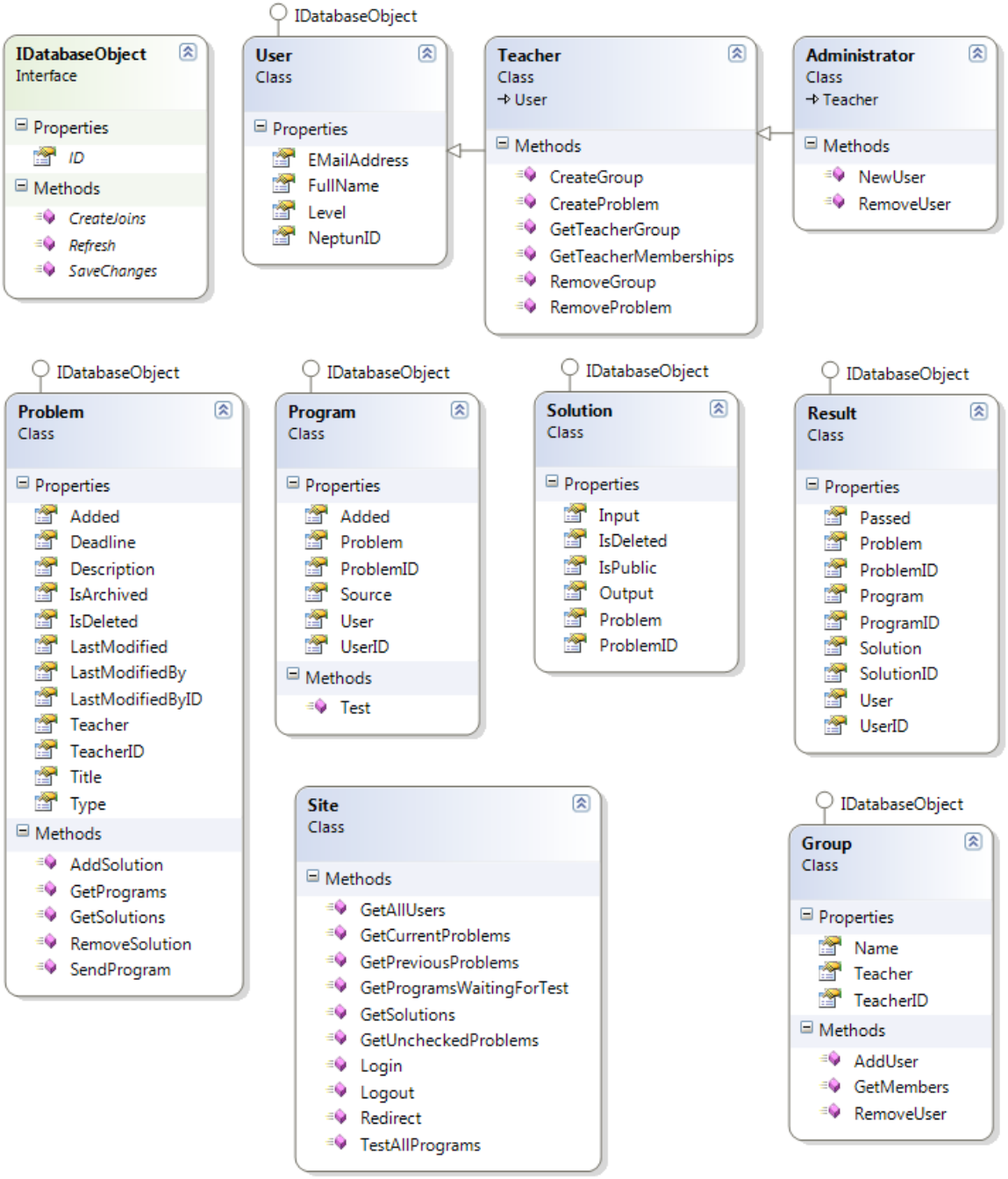
A következő ábrán pedig egy admin használati eset diagramja látható.



6. ábra: Administrator use case diagram

6.2.4. Osztály diagram

A következő ábrán a program osztály diagramját mutatom be, mely tartalmazza az egy interfacet, az összes osztályt és az öröklődési viszonyukat is.



7. ábra:Osztály diagram

6.3. PHP függvények és osztályok leírása

Az alábbi pontokban jellemzem a fejlesztés során az alkalmazást felépítő osztályokat és interface-t. Az osztályok jellemzése utána részletesen bemutatom a tulajdonságaikat és metódusaikat.

6.3.1. Az IDatabaseObject interface

Ez az interface lényegében egy üzleti objektumot valósít meg. Lényegében ez az alap modell.

Metódusai:
`Refresh`: Frissíti az objektumot az adatbázis alapján.
`SaveChanges`: Elmenti a változtatásokat az adatbázisban.
`CreateJoins`: Létrehozza a kapcsolódó objektumokat.

6.3.2. Az User osztály

Egy felhasználót reprezentál, amely felhasználó jelen esetünkben egy hallgató. Implementálja az `IDatabaseObject` interface-t.

Tulajdonságai:
`ID`: Az egyedi azonosítója.
`Level`: Szintje (Lehetséges szintek: `User`; `Teacher`; `Administrator`) Ez az osztály a `User` szintet fogja megkapni.
`EMailAddress`: A hallgató e-mail címe.
`FullName`: Teljes az adott felhasználónak neve.
`NeptunID`: Neptun azonosítója.

Metódusai:
`ChangePassword`: Ezen metódus megváltoztatja az adott hallgatóhoz tartozó jelszót.
`GetMemberships`: Visszaadja azokat a csoportokat amiknek tagja az adott felhasználó.
`GetProgramForProblem`: Az adott feladatra elkészített és legutoljára feltöltött programot adja vissza.
`GetProgramsWaitingForTest`: Visszaadja azokat a programokat, amik tesztelésre várnak.
`GetResults`: Visszaadja a `User` által elért eddigi eredményeket.

6.3.3. A Teacher osztály

Ez egy tanárt - jelen esetünkben gyakorlatvezetőt - reprezentál. Ezen osztálynak a `User` osztály az ősosztálya, abból származtatjuk.

Tulajdonságai:
`Level`: A tanár szintje (Lehetséges szintek: `User`; `Teacher`; `Administrator`). Ez az osztály a `Teacher` szintet fogja megkapni.

Metódusai:
`CreateGroup`: Létrehoz egy csoportot az aktuális tanár ID-jához.
`RemoveGroup`: Kitöröl egy adott a tanárhoz kapcsolodó csoportot.
`CreateProblem`: Létrehoz egy új feladatot.
`RemoveProblem`: Kitöröl egy feladatot.
`GetTeacherMemberships`: Visszaadja azokat a csoportokat, ahol ő tanít, gyakorlatot tart.

6.3.4. Az Administrator osztály

Ez az osztály egy adminisztrátort valósít meg. A `Teacher` osztályból származtatjuk.

Tulajdonságai:
`Level`: Az adminisztrátor szintje (Lehetséges szintek: `User`; `Teacher`; `Administrator`). Ez az osztály a `Administrator` szintet fogja megkapni.

Metódusai:
`NewUser`: Hozzáad egy felhasználót.
`RemoveUser`: Kitöröl egy felhasználót.

6.3.5. A Group osztály

Csoportokat reprezentáló osztály. Az `IDatabaseObject` interface-t implementálja.

Tulajdonságai:
`ID`: A csoport azonosítója.
`Name`: A csoport neve.
`Teacher` (objektum): A csoportot tanító tanár példánya.
`TeacherID`: A csoportot tanító tanár azonosítója.

Metódusai:
`AddUser`: Belerak egy felhasználót a csoportba.
`RemoveUser`: Kiveszi a megadott felhasználót a csoportból.
`GetMembers`: Az adott csoport tagjait adja vissza.

6.3.6. A Problem osztály

Ez az osztály valósítja meg a feladatokat. Implementálja az `IDatabaseObject` interface-t.

Tulajdonságai:
`ID`: A feladat azonosítója.
`Title`: A feladat címe
`Type`: A feladat milyen programozási nyelvvel kell megoldani. Egyelőre kétféle programozási nyelven lehet választani, vagy `c`-t vagy `java`-t.
`Description`: A feladat leírása.
`IsArchived`: Ennek az értéke `true` vagy `false`. Azt jelzi, hogy a feladat archivált és kiértékelt-e vagy sem.
`IsDeleted`: Jelzi, hogy az adott feladat (probléma) törölt-e vagy sem, mivel csak logikai törlés van.
`Deadline`: A másodpercre pontos határidőt tartalmazza, ameddig az adott feladatot be lehet küldeni.
`Added`: Az a dátum, amikor létre lett hozva a feladat.
`LastModifiedBy` (objektum): Annak a felhasználónak a példánya, aki utoljára módosította a feladatot.
`LastModifiedByID`: Az utoljára módosító felhasználni az egyedi azonosítója.
`LastModified`: Az utolsó módosítás időpontja.

`Teacher` (objektum): A feladatot létrehozó felhasználó (tanár) példánya.
`TeacherID`: A feladatot léterhozó tanár egyedi azonosítója.

Metódusai:
`SendProgram`: Az aktuális problémára egy megoldást (programot) add hozzá az adott felhasználótól.
`AddSolution`: Egy lehetséges tesztesetet ad hozzá: egy bementet és egy elvárt kimentet; továbbá azt, hogy tesztelésre használható-e az adott teszteset.
`RemoveSolution`: Kitöröl egy tesztesetet az adott problémára.
`GetPrograms`: A feladatra a felhasználók által elkészített megoldásokat adja vissza.
`GetSolutions`: Az összes tesztesetet adja vissza.

6.3.7. A Program osztály

A beküldött programokat reprezentálja. Az `IDatabaseObject` interface-t implementálja.

Tulajdonságai:
`ID`: A program egyedi azonosítója
`Problem` (objektum): Annak a feladatnak (problémának) a példánya, amelyre az adott programot be lett küldve.
`ProblemID`: A feladat egyedi azonosítója.
`User` (objektum): Annak a felhasználónak (hallgatónak) a példánya, aki beküldte a megoldást (programot)
`UserID`: A beküldő felhasználó azonosítója.
`Added`: Azon időpont, amikor a programot a felhasználó feltöltötte.
`Source`: Maga a forráskód.

Metódusa:
`Test`: Leteszteli, hogy a program jól működik-e.

6.3.8. A Result osztály

A program – összes esetre – való tesztelése után az eredményeket reprezentáló osztály. Implementálja az `IDatabaseObject` interface-t.

Tulajdonságai:
`ID`: Az eredmény azonosítója.
`Passed`: Azt jelzi, hogy az adott felhasználó, adott problémára adott megoldása (programja) sikeresen átesett-e a teszten.
`Problem` (objektum): A megoldáshoz tartozó feladat példánya.
`ProblemID`: A feladat azonosítója.
`User` (objektum): Annak a felhasználónak a példánya, akihez a megoldás tartozik.
`UserID`: A felhasználó azonosítója.
`Program` (objektum): Azon program példánya, amelynek az eredményét megkaptuk.
`ProgramID`: A program azonosítója.

Metódusai:
Ennek az osztálynak nincsenek metódusai.

6.3.9. A Solution osztály

Teszteseteket reprezentáló osztály.

Tulajdonságai:
`ID`: A teszteset azonosítója
`Problem` (objektum): A feladat példánya, amely feladathoz a tesztesetet adtuk.
`ProblemID`: A feladat (probléma) azonosítója.
`IsDeleted`: Jelzi, hogy a teszteset törölt-e vagy sem.
`IsPublic`: Azt mutatja meg hogy előtesztelésnél felhasználható-e az adott teszteset.
`Input`: Az adott tesztesethez tartozó bemenet.
`Output`: Az elvárt kimenet amit az adott bemenetnél a programnak produkálnia kell.

Metódusai:
Ennek az osztálynak nincsenek metódusai.

6.3.10. A Site osztály

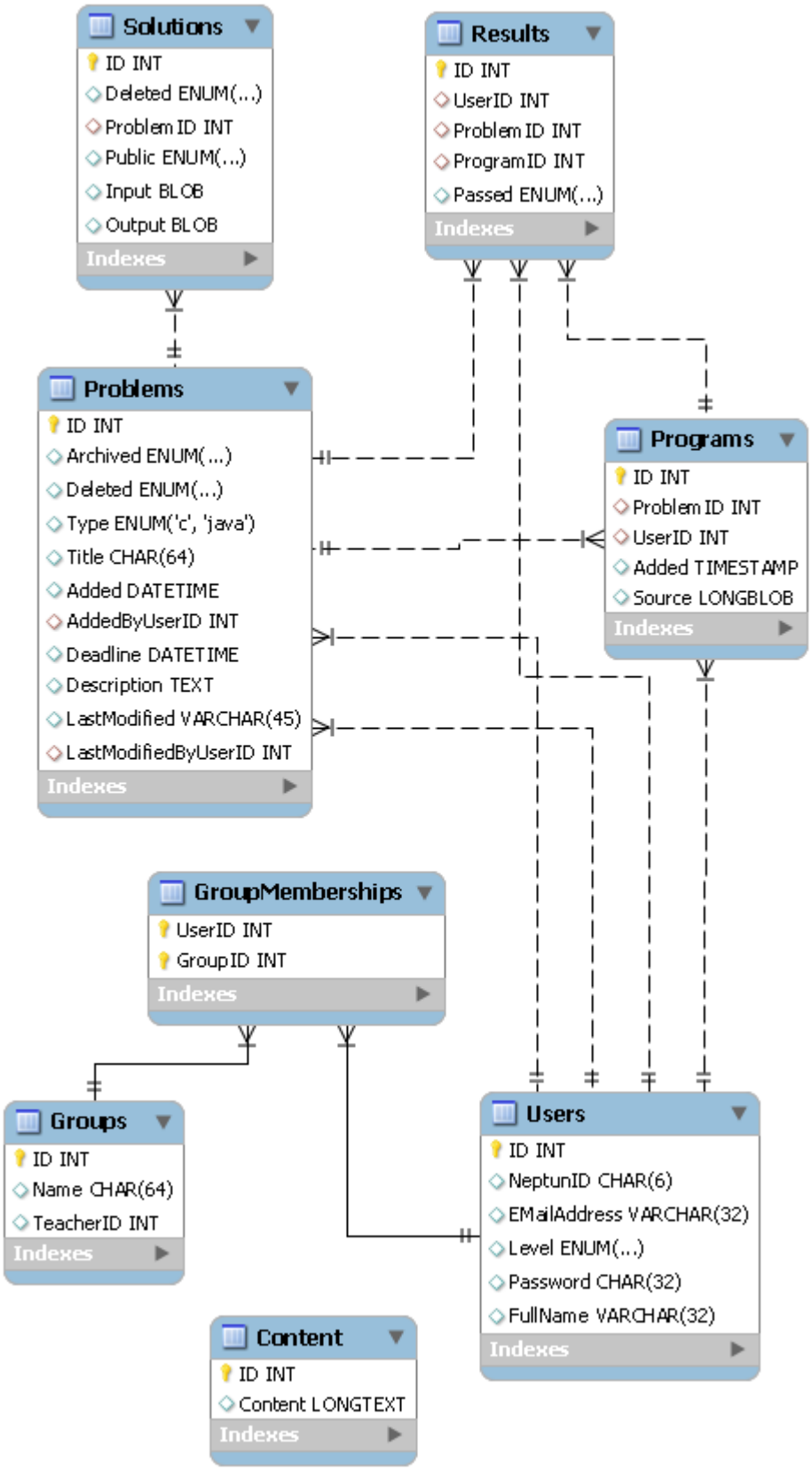
Magát a webalkalmazást kezelő osztály. Lényegében ez az osztály felel az oldal működéséért.

Tulajdonságai:
Ennek az osztálynak nincsenek tulajdonságai.

Metódusa:
`Redirect`: Átirányít a paraméterben megadott oldalra.
`Login`: Beléptet egy felhasználót az oldalra.
`Logout`: Kilépteti az aktuális felhasználót az oldalról.
`GetAllUsers`: Lekéri az összes felhasználót az adatbázisból.
`GetCurrentProblems`: Lekéri az aktív feladatokat, amelyek sem nem archiváltak, sem nem járt le a határidejük.
`GetPreviousProblems`: Lekéri az összes olyan feladatot, amelynek lejárt a beadási határideje és egyben ellenőrzöttek (archiváltak) is.
`GetUncheckedProblems`: Lekéri azon problémákat, feladatokat, amelyeknek lejárt az beadási határidejük, de még nem ellenőriztük le a beadott programokat, ezáltal nem is archiváltuk.
`GetProgramsWaitingForTest`: Minden programot visszaadja, amely tesztelésre vár..
`GetSolutions`: Az összes tesztesetet visszaadja az adott problémára.
`TestAllPrograms`: Egy adott problémára leteszteli az összes beérkezett programot.

6.4. Adatbázis

A program fejlesztése során a következő táblákat és kapcsolatokat használva valósítottam meg a végleges változatot.



8. ábra: Adatbázis táblák

6.4.1. A Users tábla

Column Name	Datatype	PK	NN	BIN	UN	ZF	AI	Default
ID	INT	☑	☑	☐	☑	☐	☑	
NeptunID	CHAR(6)	☐	☑	☐	☐	☐	☐	
EMailAddress	CHAR(64)	☐	☐	☐	☐	☐	☐	
Level	ENUM('User', 'Teache...	☐	☐	☐	☐	☐	☐	User
Password	CHAR(32)	☐	☐	☐	☐	☐	☐	
FullName	CHAR(64)	☐	☐	☐	☐	☐	☐	
		☐	☐	☐	☐	☐	☐	

9. ábra: A Users tábla felépítése

Ebben a táblában tároljuk a felhasználókat. Mind az adminisztrátorokat, mind a tanárokat és a hallgatókat is.

A tábla oszlopai:

ID: A felhasználó egyedi azonosítója. Azért használok számot karakter (NeptunID) helyett, mert gyorsabb indexelésnél.

NeptunID: A felhasználók neptun azonosítója.

Email: A felhasználó e-mailcíme.

Level: A felhasználó szintje; lehet: `User`; `Teacher`; `Administrator`.

Password: Titkosítva tároljuk a jelszót.

Fullname: Teljes név.

6.4.2. A Groups tábla

Column Name	Datatype	PK	NN	BIN	UN	ZF	AI	Default
ID	INT	☑	☑	☐	☑	☐	☑	
Name	CHAR(64)	☐	☐	☐	☐	☐	☐	
TeacherID	INT	☐	☐	☐	☑	☐	☐	
		☐	☐	☐	☐	☐	☐	

10. ábra: A Groups tábla felépítése

A csoportok tábla. Lényegében ebben a táblában tároljuk a csoportok nevét, és hogy melyik tanár tanítja az adott csoportot.

A tábla oszlopai:

ID: Minden csoport egyedi azonosítót kap.

Name: A csoport neve.

TeacherID: A csoportot tanító, a csoportban gyakorlatot tartó tanár ID-ja.

6.4.3. A GroupMemberships tábla

Column Name	Datatype	PK	NN	BIN	UN	ZF	AI	Default
UserID	INT	☑	☑	☐	☑	☐	☐	
GroupID	INT	☑	☑	☐	☑	☐	☐	
		☐	☐	☐	☐	☐	☐	

11. ábra: A GroupMemberships tábla felépítése

Ez egy szótártábla, amely megvalósítja a felhasználók és a csoportok közötti kapcsolatot.

A tábla oszlopai:

UserID: A felhasználó egyedi azonosítója.

GroupID: A csoport egyedi azonosítója.

6.4.4. A Problems tábla

Column Name	Datatype	PK	NN	BIN	UN	ZF	AI	Default
ID	INT	☑	☑	☐	☑	☐	☑	
Archived	ENUM('true', 'false')	☐	☐	☐	☐	☐	☐	false
Deleted	ENUM('true', 'false')	☐	☐	☐	☐	☐	☐	false
Type	ENUM('c', 'java')	☐	☐	☐	☐	☐	☐	c
Title	CHAR(64)	☐	☐	☐	☐	☐	☐	
Added	DATETIME	☐	☐	☐	☐	☐	☐	
AddedByUserID	INT	☐	☐	☐	☑	☐	☐	
Deadline	DATETIME	☐	☐	☐	☐	☐	☐	
Description	TEXT	☐	☐	☐	☐	☐	☐	
LastModified	VARCHAR(45)	☐	☐	☐	☐	☐	☐	'CURRENT_TIMESTA...
LastModifiedByUserID	INT	☐	☐	☐	☑	☐	☐	
		☐	☐	☐	☐	☐	☐	

12. ábra: A Problems tábla felépítése

A hallgatóknak a beküldendő házi feladatokat (problémákat) ebben a táblában tároljuk.

A tábla oszlopai:

ID: A feladat egyedi azonosítója.

Archived: Ez lényegében egy logikai azonosító annak eldöntésére, hogy leteszteltük-e az erre a feladatra beküldött programokat. Értéke lehet `true` vagy `false`.

Deleted: A problémát törölhetünk is, viszont akkor sem töröljük ki az adatbázisból, hanem csak átállítjuk ennek a logikai értékét. Értéke lehet `true` vagy `false`.

Type: A feladat típusa. Jelenleg két típus támogatott a rendszerben. Az egyik a c a másik pedig a java. Tehát itt megadhatjuk hogy az adott feladatra c vagy java programokat várunk.

Title: A feladat címe.

Added: Az az időpont amikor a feladatot hozzáadtuk.

AddedByUserID: Annak a felhasználónak az azonosítója, aki hozzáadta a feladatot.

Deadline: A határidő, ameddig a feladatra várjuk a megoldásokat.

Description: A feladat szövege. Ez egy hosszabb terjedelmű, magyarázó szöveg.

LastModified: Azon időpont, amikor a feladat szöveg utoljára módosítva lett.

LastModifiedByUserID: Annak a felhasználónak az azonosítója, aki utoljára módosította a feladatot.

6.4.5. A Solutions tábla

Column Name	Datatype	PK	NN	BIN	UN	ZF	AI	Default
ID	INT	☑	☑	☐	☑	☐	☑	
Deleted	ENUM('true', 'false')	☐	☐	☐	☐	☐	☐	false
ProblemID	INT	☐	☐	☐	☑	☐	☐	
Public	ENUM('true', 'false')	☐	☐	☐	☐	☐	☐	false
Input	BLOB	☐	☐	☐	☐	☐	☐	
Output	BLOB	☐	☐	☐	☐	☐	☐	
		☐	☐	☐	☐	☐	☐	

13. ábra: A Solutions tábla felépítése

Ebben a táblában tároljuk a teszteseteket, amelyekre fogjuk tesztelni a feladatokra beküldött programokat.

A tábla oszlopai:

ID: A teszteset egyedi azonosítója.
Deleted: Ha egy tesztesetet törölni szeretnénk, az nem jár a táblából való törlésével. A törlést itt is csak logikailag valósítjuk meg. Értéke lehet `true` vagy `false`.
ProblemID: Annak a feladatnak az azonosítója, amelyhez az adott teszteset tartozik.
Public: A tanárok kétféle tesztesetet hozhatnak létre. Amelyik a „próbálgatások” folyamán értékelődnek ki, és amelyek csak az igazi, éles tesztelés során. Erre szolgál ez a mező. Értéke lehet `true` vagy `false`.
Input: Ebben a blobban tároljuk ez hogy a tesztesetnek mi lesz a bemenete.
Output: Azt pedig hogy az adott bemenetre milyen kimenetet várunk el, ebben a blobban tároljuk.

6.4.6. A Programs tábla

Column Name	Datatype	PK	NN	BIN	UN	ZF	AI	Default
ID	INT	☑	☑	☐	☑	☐	☑	
ProblemID	INT	☐	☐	☐	☑	☐	☐	
UserID	INT	☐	☐	☐	☑	☐	☐	
Added	TIMESTAMP	☐	☐	☐	☐	☐	☐	CURRENT_TIMESTAMP
Source	LONGBLOB	☐	☐	☐	☐	☐	☐	
		☐	☐	☐	☐	☐	☐	

14. ábra: A Programs tábla felépítése

A beküldött programokat sem fájlként tároljuk el, hanem az adatbázisban. Erre szolgál ez a tábla.

A tábla oszlopai:

ID: A program egyedi azonosítója.
ProblemID: Annak a feladatnak az azonosítója, amelyhez a beküldött program tartozik.
UserID: Annak a felhasználónak az egyedi azonosítója, aki beküldte a programot.
Added: Az az időpont, amikor a programot feltöltötte az illető.

Source: A forráskódja a programnak.

6.4.7. A Results tábla

Column Name	Datatype	PK	NN	BIN	UN	ZF	AI	Default
ID	INT	☑	☑	☐	☑	☐	☑	
UserID	INT	☐	☐	☐	☑	☐	☐	
ProblemID	INT	☐	☐	☐	☑	☐	☐	
ProgramID	INT	☐	☐	☐	☑	☐	☐	
Passed	ENUM('true', 'false')	☐	☐	☐	☐	☐	☐	false
		☐	☐	☐	☐	☐	☐	

15. ábra: A Results tábla felépítése

Ebben a táblában lesznek tárolva a végleges tesztek eredményei. Ezek azoknak a teszteknek az eredményei, amelyekben mind a publikus, mint a nem publikus tesztesetekre végrehajtódtak.

A tábla oszlopai:

ID: Az eredmények egyedi azonosítója.
UserID: Annak a felhasználónak az egyedi azonosítója, akinek az eredményét tároljuk.
ProblemID: Annak a feladatnak az egyedi azonosítója, amelyre teszteltünk.
ProgramID: Annak a programnak az egyedi azonosítója, amit lefordítva és lefuttatva az összes tesztesetre teszteltünk.
Passed: Ha az összes teszteset eredménye pozitív kicsengéssel ér véget, akkor a passed értéke `true` lesz, ellenkező esetben pedig `false`.

6.4.8. A Content tábla

Column Name	Datatype	PK	NN	BIN	UN	ZF	AI	Default
ID	INT	☑	☑	☐	☑	☐	☑	
Content	LONGTEXT	☐	☐	☐	☐	☐	☐	
		☐	☐	☐	☐	☐	☐	

16. ábra: A Content tábla felépítése

Ebben a táblában tárolhatunk nagy mennyiségű szöveget. Például a főoldali információkat is itt tárolhatjuk.

A tábla oszlopai:

ID: Egyedi azonosító.
Content: Nagy mennyiségű szöveg tárolására.

7. Alkalmazás leírása

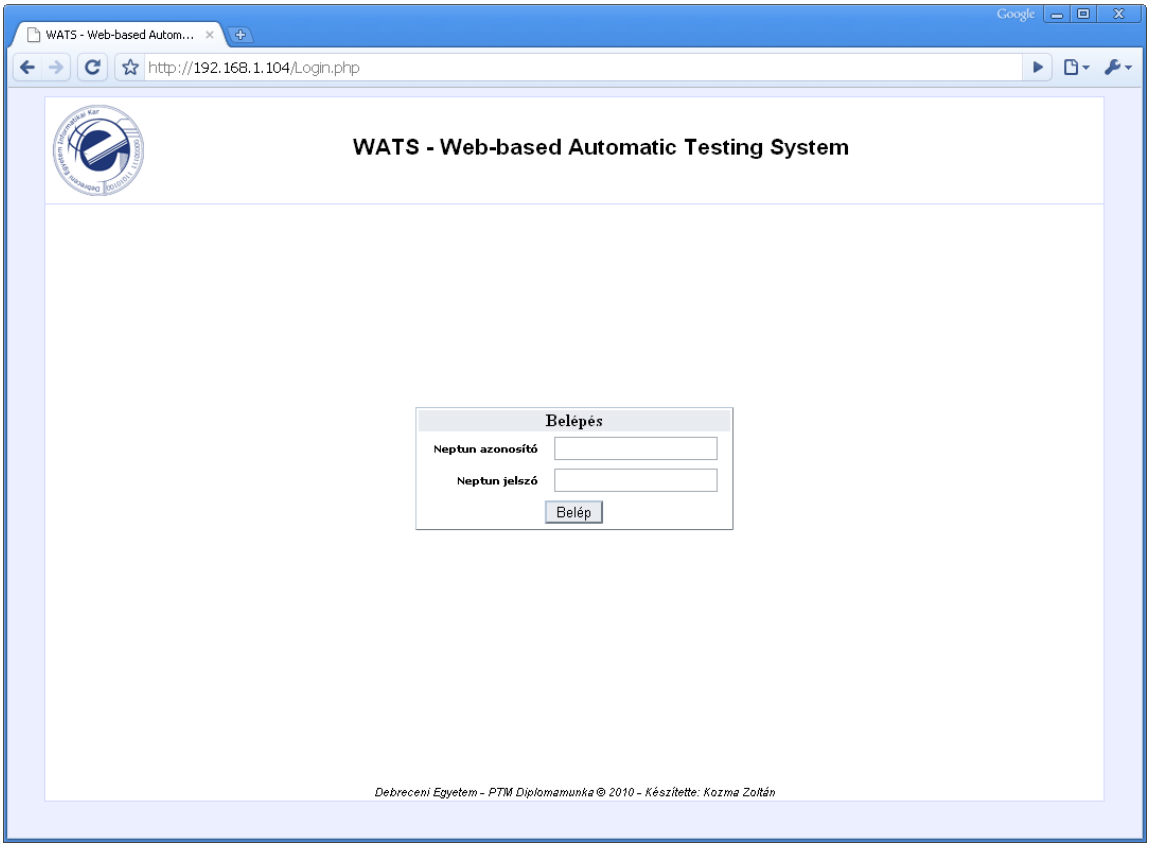
Az alkalmazás fejlesztése, tesztelése és futtatására egy otthoni – Debian Linux alapú operációs rendszerrel működő – asztali számítógépet használtam.

7.1. Az alkamazás bemutatása

A következő néhány pontban az webes alkalmazás fontosabb oldalait mutatom be.

7.1.1. Bejelentkező oldal

A bejelentkezési oldal egy egyszerű form-ot tartalmaz, mely elkéri a user, pl. hallgató neptun- azonosítóját és jelszavát és ezek segítségével megpróbál belépni az oldalra.

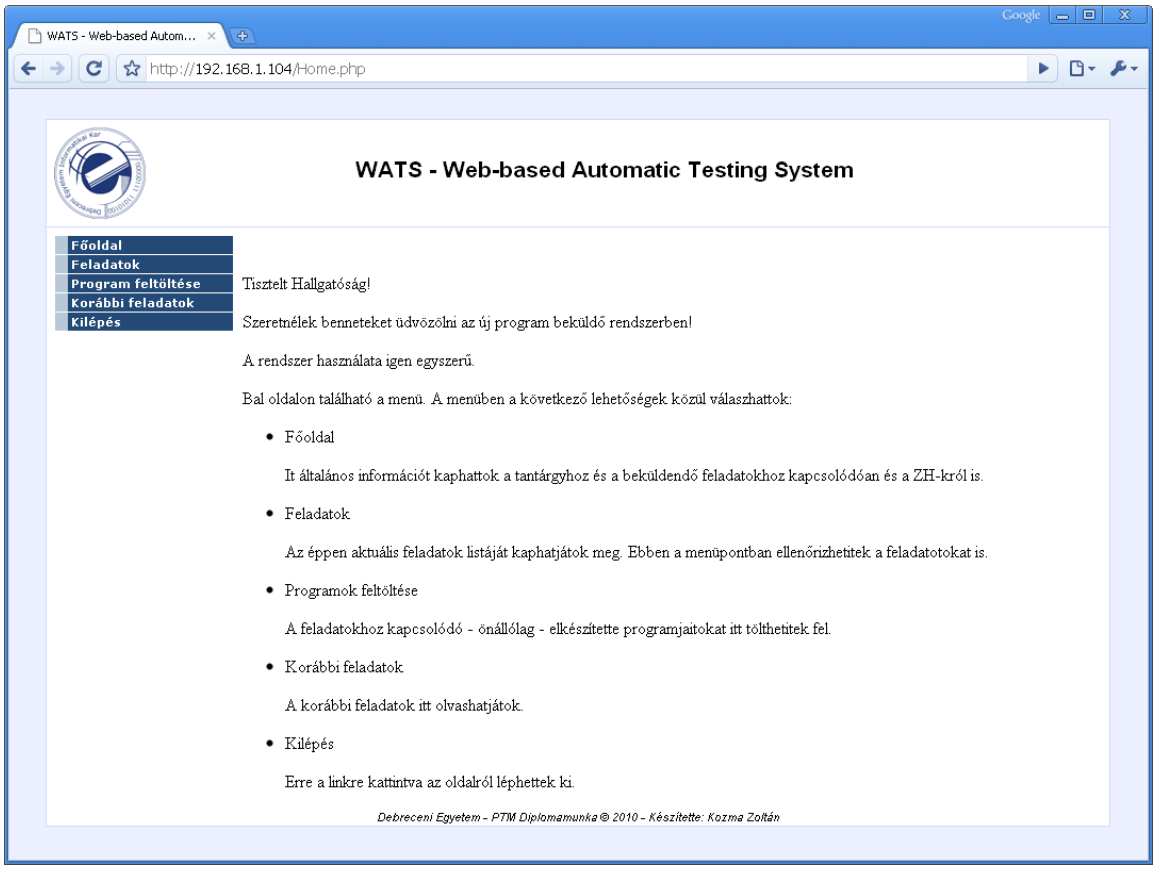


17. ábra: Bejelentkező képernyő

7.1.2. Főoldal

Sikeres bejelentkezés után a főoldal jelenik meg az autentikált felhasználóknak. Ezen az oldalon általános információt írhatunk ki az egyetem hallgatóságának mind az oldal mind a követelményekről.

Az oldalt csak adminisztrátori jogosultsággal lehet szerkeszteni.



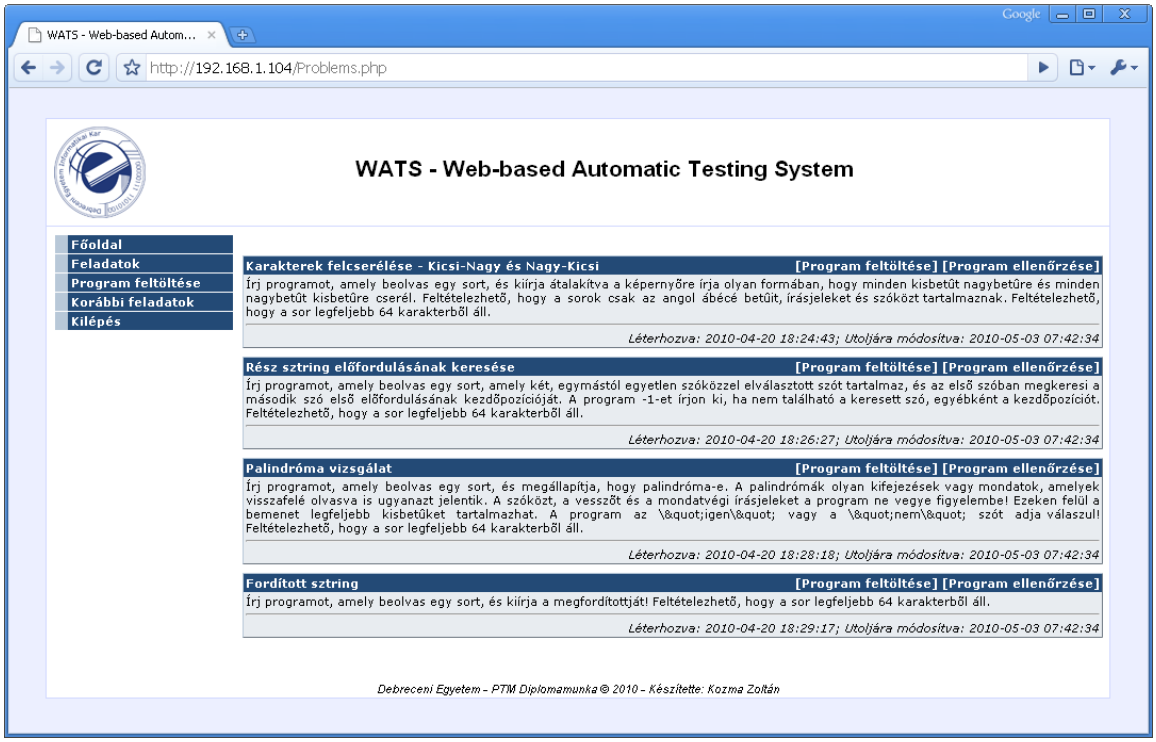
18. ábra: Főoldali kép

7.1.3. Feladatok

A feladatok oldalon az éppen aktuális, határidőn belüli feladatok listáját láthatjuk.

Egy-egy „dobozban” több információt mutatunk meg a hallgatóknak. A feladat- címe és szövege, a létrehozás- és az utolsó módosítás dátuma. A jobb felső sarokban két link is található. Az egyik a program feltöltése, mely segítségével rögtön a feltöltő oldalra jutunk. A

másik link pedig a program ellenőrzése, melynek segítségével a program tesztelési oldalra juthatunk.



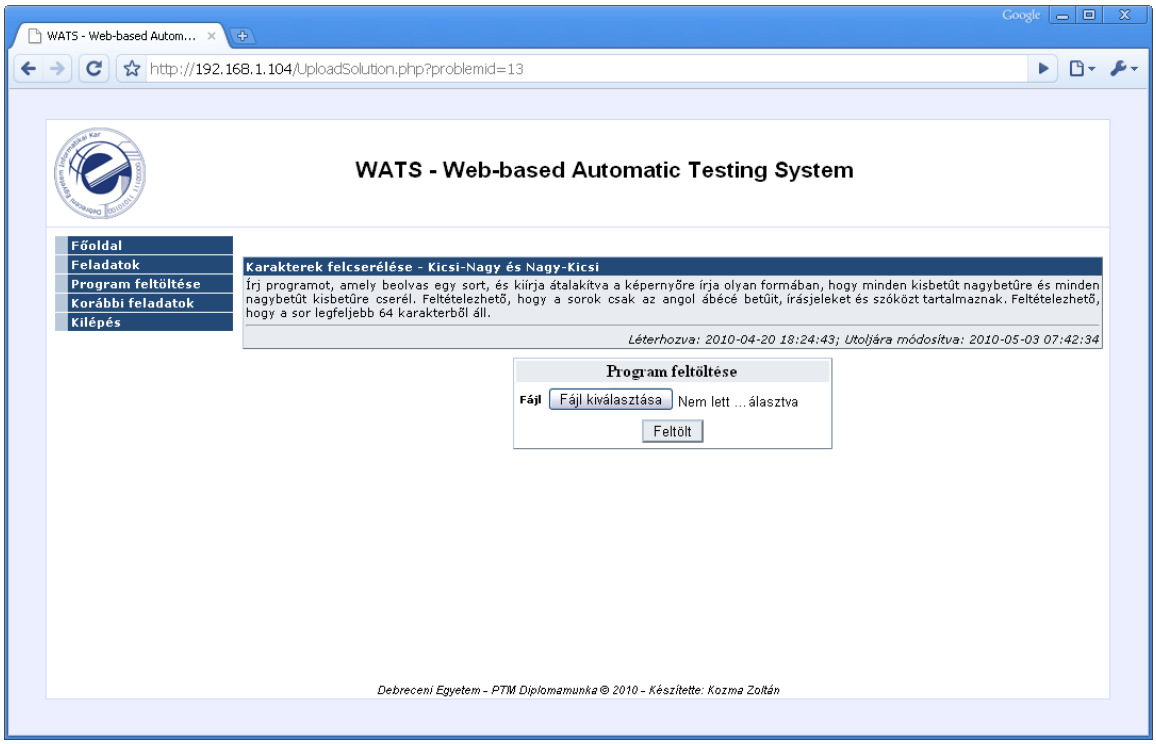
19. ábra: Beküldendő feladatok listája

7.1.4. Program feltöltése

A programok feltöltését vagy közvetlenül a feladatok listájában található linkkel érhetjük el, vagy a menüben található linkre kattintva kiválasztjuk, hogy melyik feladathoz szeretnénk feltölteni az elkészített programunkat.

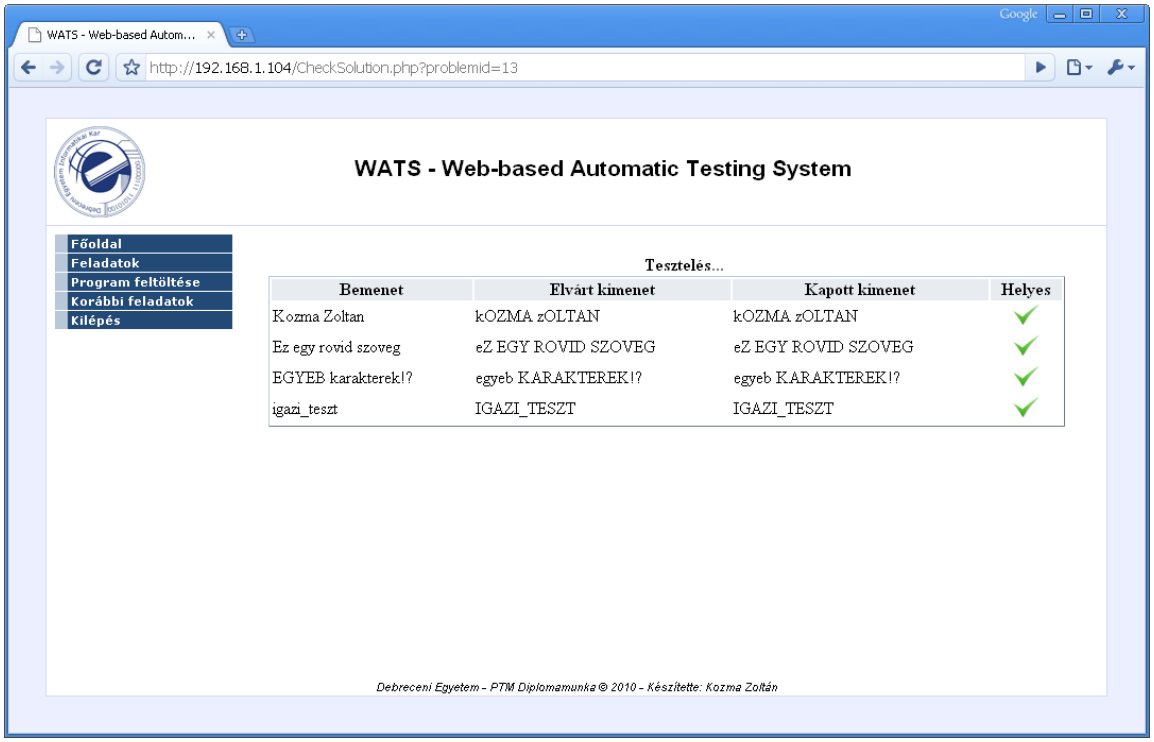
A következő oldalon megjelenik maga a feladat címe és szövege. A feladat-doboz után pedig egy külön egységként jelenik meg a form. Ezen form segítségével tölthetjük fel az elkészített programunkat.

Sikeres, illetve sikertelen feltöltés után egy szöveg jelenik meg az oldalon, amiben értesülünk az eredményességről, illetve az eredménytelenségről.



20. ábra: Program feltöltése

7.1.5. Program tesztelése



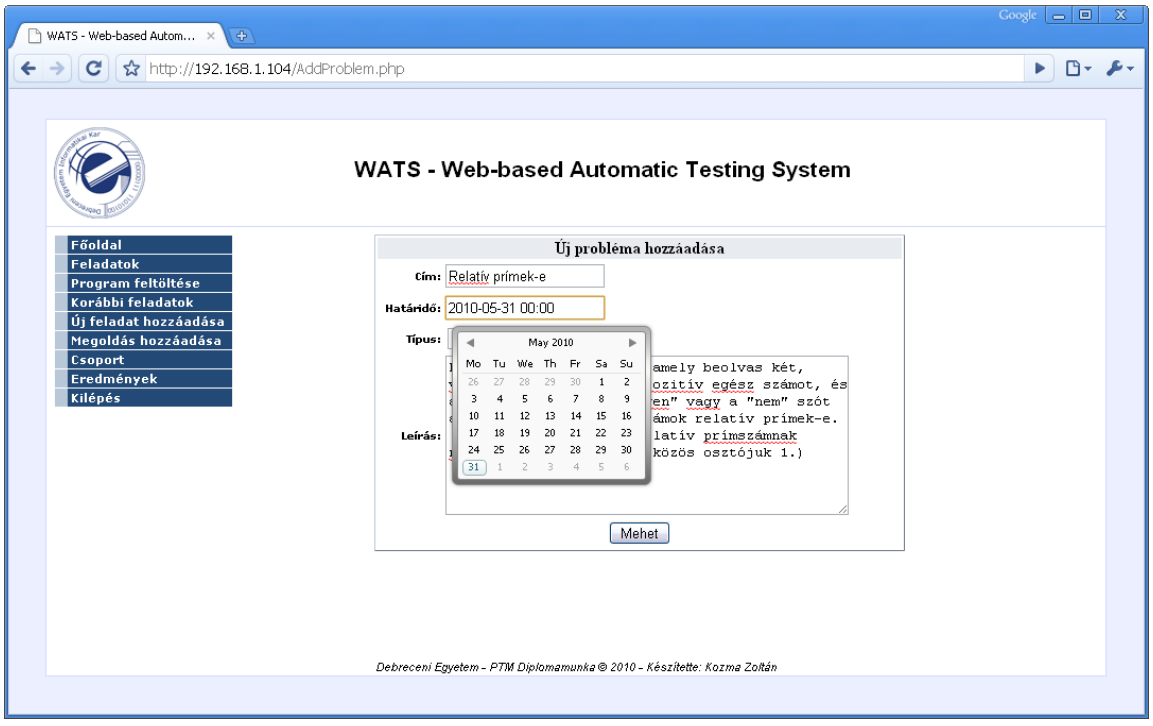
21. ábra: Program tesztelése publikus tesztesetekkel

A program tesztelése is webes felületen történik. A program fordítása után a tanárok és gyakorlatvezetők által publikusnak minősített tesztesetekkel teszteljük a programot.

Az oldalon egy táblázatos formában jelenik meg az eredmény. A táblázatban láthatjuk hogy milyen bemenetre lett tesztelve a program, mit várunk el az aktuális bemenet válaszaként és mi az aktuális kimenet, amit a program adott. Az elvárt és az aktuális kimenet összehasonlításából megkapjuk, hogy a programunk helyesen működik-e, avagy sem.

7.1.6. Új feladat hozzáadása

Új feladat hozzáadását csak tanári joggal rendelkező felhasználó végezheti el. A feladat hozzáadására rendelkezésünkre áll egy form.



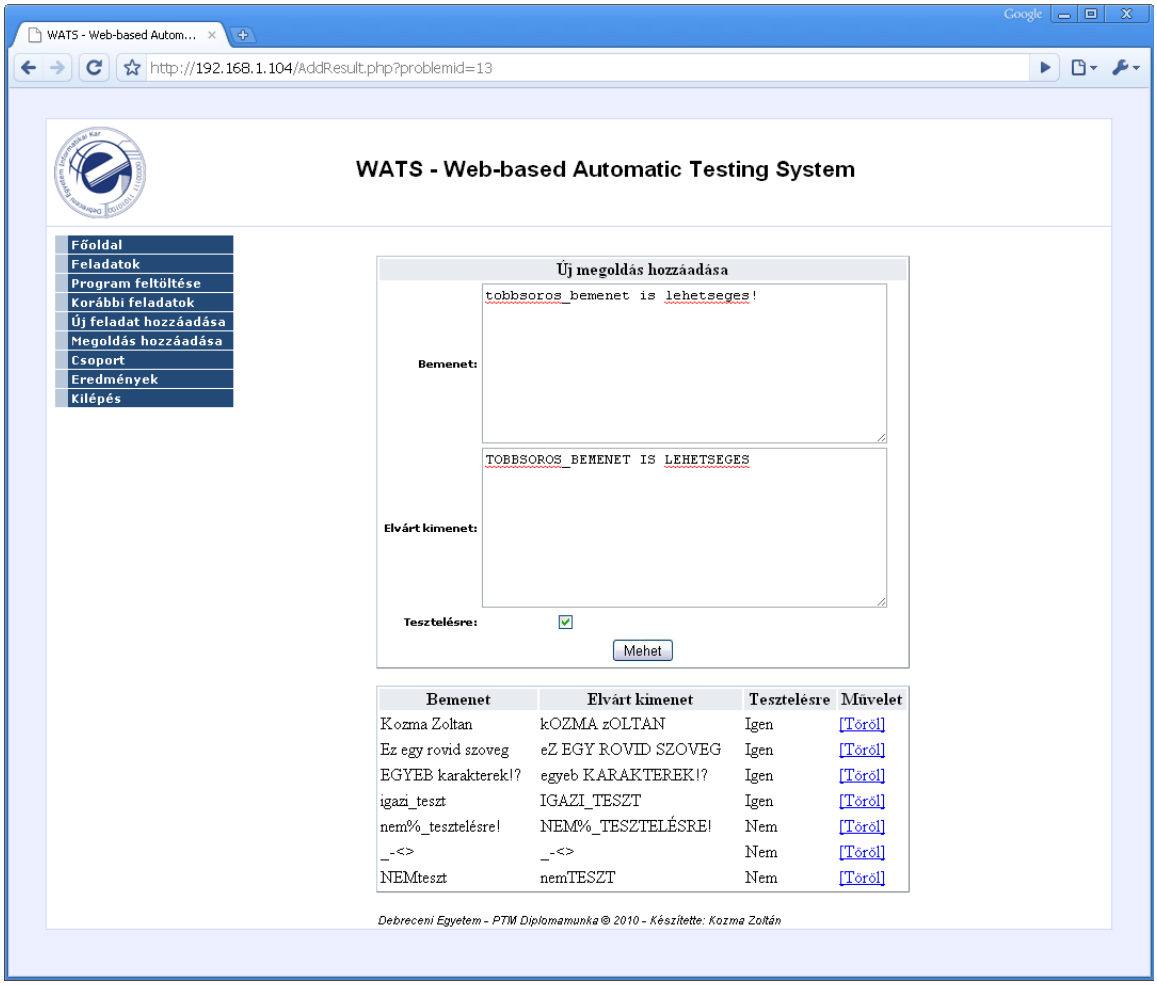
22. ábra: Új feladat hozzáadása.

Az űrlapon meg kell adni a feladat címét, ami egyértelműen beazonosítja a feladatot. A következő résznél meg kell adni azt a határidőt, ameddig várjuk a programokat. Itt percre pontosan megadhatjuk a határidőt. Majd egy select mező következik, ahol két

lehetőség közül választhatunk, `c` és `java`. Itt azt választjuk ki, hogy milyen programozási nyelven készített feladatokat várunk. A végén pedig egy leírást adhatunk, hogy hogyan is kell kinéznie a programnak, mit kell tudnia, mire kell ügyelni; egyszóval a feladattal (programmal) szembeni követeléseinek írjuk le.

7.1.7. Tesztesetek (megoldások) hozzáadása

A megoldások hozzáadását a tanári joggal rendelkező felhasználók végezhetik.



23. ábra: Tesztesetek hozzáadása.

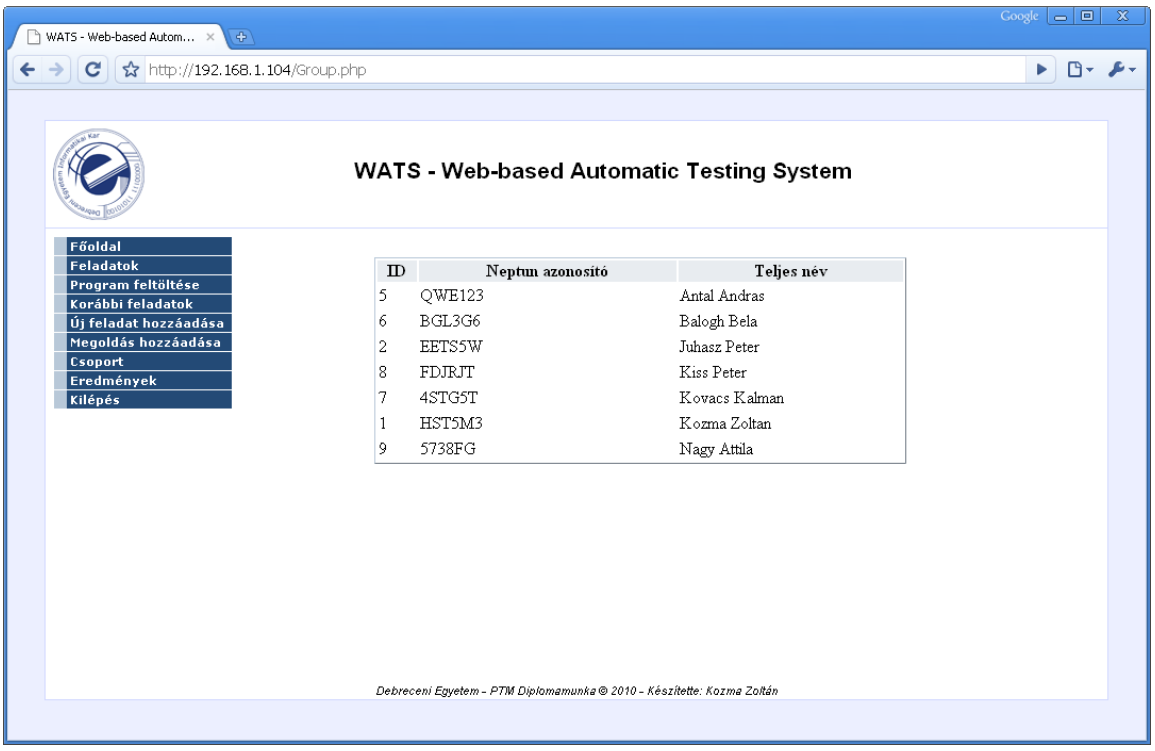
Ezen az oldalon egy űrlap, illetve az eddig hozzáadott tesztesetek listáját láthatjuk.

A formon meg kell adnunk egy bemenetet, illetve egy elvárt kimentet. A minden egyes, itt szereplő bemenetre le fogjuk tesztelni a programunkat és a kapott kimenetet összehasonlítjuk az elvárt kimenettel. Ha ezek meg fognak egyezni, akkor a programunk feltételezhetően jól működik, ha pedig nem, akkor rosszul. Továbbá megadhatjuk, hogy azt a teszt esetet amit éppen hozzá kívánunk adni engedjük-e, hogy a felhasználók (hallgatók) használják, mint próba teszteset.

Az űrlap alatt láthatjuk az eddigi tesztesetek listáját. Láthatjuk, hogy mi a bemenet, mi az elvárt kimenet, azt hogy a hallgatók használhatják-e tesztelésre, avagy sem. A sor végén egy `[Töröl]` műveletet láthatunk. Rákattintva törölhetjük az aktuális tesztesetet a listából.

7.1.8. Csoport

Minden tanár megnézheti hogy kik vannak az ő csoportjában. A tanárok csak és kizárólag a saját csoportjukat látják.

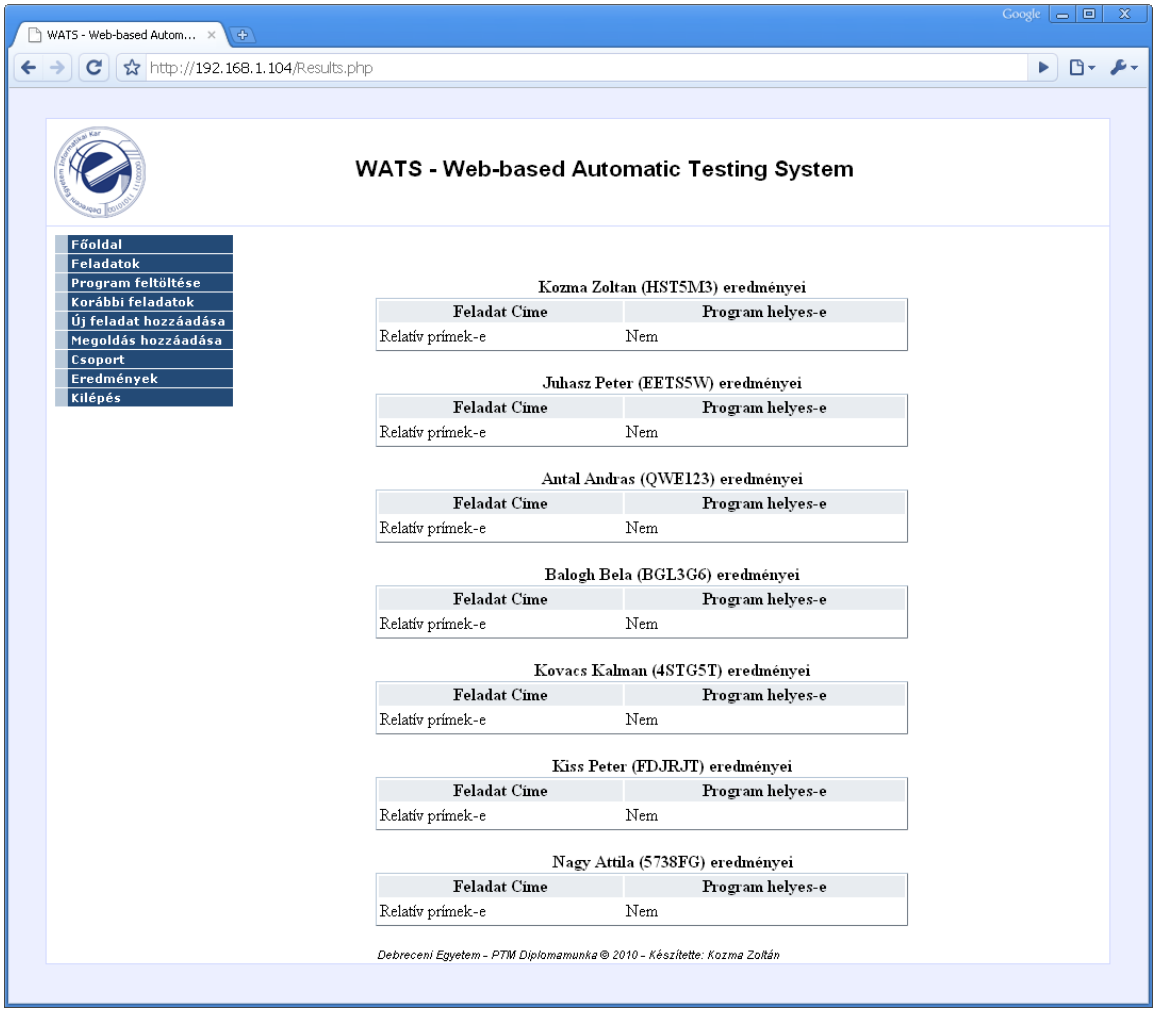


24. ábra: Csoport tagjainak listázása.

A listában megtekintheti a tanár a gyakorlati csoportjában tanuló hallgatók neptun azonosítóját és a teljes nevüket. Nem tudja a listát módosítani, mert erre csak az adminisztrátornak van jogosultsága.

7.1.9. Eredmények

Ha az adminisztrátor leellenőrizte a beküldött programokat, akkor az eredmények megjelennek minden tanár számára.



25. ábra: Csoport eredményeinek listázása.

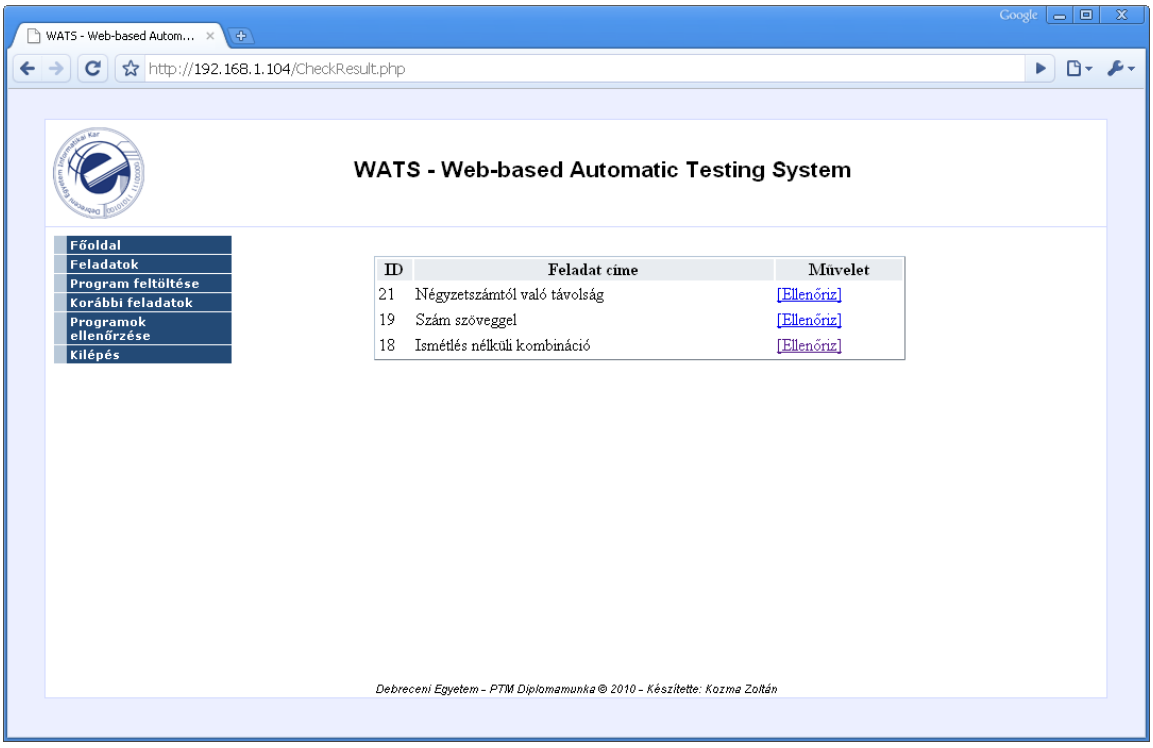
A csoport tagjaira lebontva jelennek meg az eredmények. Tehát minden gyakorlaton az aktuális gyakorlatvezetőnél levő hallgató sorban fel lesz sorolva, és a hallgató neve és neptun azonosítója alatt pedig az ellenőrzött összes feladat látható lesz.

Két oszlop található; az egyikben a feladat neve, a másikban, hogy az összes tesztesetre letesztelve mindig helyes eredményt adott-e a program.

7.1.10. Programok ellenőrzése

A programok ellenőrzését csak az adminisztrátor végezheti. Egy táblázat jelenik meg az oldalon, mely azokat a feladatokat tartalmazza, amelyeknek lejárt a beadási határidejük, de még nem ellenőrizte le a rájuk beérkezett programokat senki.

Az adminisztrátor egyszerűen az `[Ellenőriz]` linkre kattintva leellenőrizheti a rendszer segítségével a felhasználók (hallgatók) programjait.



26. ábra: Programok ellenőrzése oldal.

8. Összefoglalás

8.1. Eredeti célok

A diplomamunkám elkészítése előtt az volt a célom, hogy létrehozzak egy olyan rendszert, amely a hallgatóságot és a tantestületet is segíti és szorosan kapcsolódik a programozói hivatáshoz. Erre a legmegfelelőbbnek egy olyan webes alkalmazást találtam, amelynek nincs szüksége, sem flash-re, sem .netframework-re, sem JVM-re, teljesen független mindenféle keretrendszertől és csupán elég egy böngésző és egy internet kapcsolat, amely a mai világban az összes felhasználó gépén jelen van.

8.2. Elért eredmények

A diplomamunkám kezdetén támasztott elvárásokat sikeresen teljesítettem. Elkészítettem egy olyan webes alkalmazást, amely segíti az egyetemen a Programozás 1 és 2 tantárgyak házi feladatainak beadását, és ezáltal a tanárokat „tehermentesíti” azok beszedése, tárolása, ellenőrzése és tesztelése alól. A tervezés, fejlesztés, megvalósítás, tesztelés fázisain keresztül jutottam el a kész webes alkalmazásig. Megpróbáltam a teljes szoftverfejlesztési életciklust a tanultak alapján a gyakorlatban is megvalósítani. A teljes webes alkalmazás megalkotása során sok szakmai cikket és weboldalt olvastam, ezáltal számos új ismeretre tettem szert, amellyel sikeresen bővítettem az egyetemen megszerzett tudásomat.

8.2. További fejlesztési lehetőségek

Annak ellenére, hogy az alkalmazás már alkalmas a programok beküldésére, illetve tesztelésére, de szeretném még bővíteni a funkcionalitását, hogy képes legyen a plágiumellenőrzésre is.

Továbbá szeretnék létrehozni egy webes szerkesztő részleget is, hogy ne kelljen még feltölteni sem a programokat, hanem fent online

meg lehessen írni vagy éppen a feltöltöttet szerkeszteni, így biztonságosan lehet tárolni, nem veszne el semmi sem.

9. Ábrajegyzék

10. Tárgymutató

11. Irodalom- és program jegyzék

11.1. Felhasznált írott irodalom

- Juhász István – A rendszerfejlesztés technológiája c. óráján készült egyetemi jegyzetem.
- Uzonyi Nikoletta egyetemi jegyzetei
- Ian Sommerville – Szoftverrendszerek fejlesztése
- Peter Moulding – PHP Black Book

11.2. Felhasznált elektronikus irodalom

- http://hu.php.net/
- http://vlib.clausvb.de/
- http://dev.mysql.com/doc/refman/5.0/en/index.html
- http://mootools.net/docs/core
- http://en.wikipedia.org/wiki/PHP
- http://hu.wikipedia.org/wiki/CSS
- http://hu.wikipedia.org/wiki/JavaScript
- http://hu.wikipedia.org/wiki/MySQL
- http://en.wikipedia.org/wiki/Memcache
- http://en.wikipedia.org/wiki/Modularity_(programming)
- http://en.wikipedia.org/wiki/VlibTemplate
- http://hu.wikipedia.org/wiki/Unified_Modeling_Language
- http://en.wikipedia.org/wiki/Web_Accessibility_Initiative
- http://hu.wikipedia.org/wiki/GNU_General_Public_License
- http://www.monkeyphysics.com/mootools/script/2/datepicker
- http://www.w3.org/WAI/
- http://www.w3.org/TR/WCAG10/
- http://eaccelerator.net/
- http://www.memcached.org/
- http://www.webmaster-toolkit.com/css-menu-generator.shtml
- http://alexdp.free.fr/violetumleditor/page.php?id=en:home

11.3. Felhasznált fejlesztői környezetek és programok

- Debian Linux
  http://www.debian.org/
- Notepad++
  http://notepad-plus.sourceforge.net/hu/site.htm
- NetBeans 6.8 (PHP bundled)
  http://netbeans.org/
- Apache HTTP Server Project
  http://httpd.apache.org/
- MySQL
  http://www.mysql.com/
- MySQL Workbench
  http://wb.mysql.com/
- phpMyAdmin
  http://www.phpmyadmin.net/home_page/index.php
- Violet UML Editor
  http://alexdp.free.fr/violetumleditor/page.php?id=en:home