SZAKDOLGOZAT

Szegedi László

Debrecen

2010

Debreceni Egyetem
Informatika Kar

Webes alapú automatikus programtesztelő rendszer fejlesztése

Témavezető:
Kósa Márk
egyetemi tanársegéd

Készítette:
Szegedi László
programtervező informatikus

Debrecen
2010

Tartalomjegyzék

Köszönetnyilvánítás

Köszönettel tartozom Kósa Márk témavezetőmnek, aki lehetőséget biztosított munkám sikeres elvégzéséhez és dolgozatom megírásához azzal, hogy mindvégig útmutató tanácsokkal látott el.

1. Bevezetés

A Debrecen Egyetem Informatika Karán évek óta használnak olyan automatikus tesztelő rendszereket, amelyek jelentősen megkönnyítik a hallgatók, és oktatók dolgát. A legnagyobb szükség ezekre a Magas szintű programozási nyelvek 1 és Magas szintű programozási nyelvek 2 tárgynál van, mivel minden évben több száz hallgató próbálja megszerezni az e kurzusok elvégzéséhez szükséges aláírást.

Az aláírás megszerzésének követelménye a két zárthelyi dolgozat teljesítése, valamint három házi feladatsor elkészítése. Ezeknek a programoknak az oktatók által előállított teszteseteken hibátlanul kell működnie. A feladatok kiírása után a hallgatók rendelkezésére állnak kezdeti tesztesetek, amelyek támpontul szolgálnak a program megírásakor. A leadási határidő lejárta után az oktatók által megadott végleges tesztesetek használatával a rendszer automatikusan kiértékeli a programokat, és egy plágium-ellenőrző segítségével kiszűri az egymáshoz hasonló megoldásokat.

A hallgatók a rendszert a hallg.inf.unideb.hu címen érhetik el. Ekkor egy szerverre kapcsolódnak, amely GNU/Linux rendszert futtat. Miután bejelentkeztek különböző parancsok futtatásával érhetik el a funkciókat. Például:

- Megoldások tesztelése
- Megoldások beadása
- Programok beadásának ellenőrzése
- Eredmények megtekintése

Egyik probléma, hogy rengeteg időt elvehet ezeknek a parancsoknak az újbóli begépelése, vagy újra előkerítése az előzmények közül, valamint a hallgatóknak több alkalmazást is használniuk kell a házi feladatok megoldása során. Szükség van egy SSH kliensre a szerver eléréséhez. Ilyen program lehet például a PuTTY. Egy újabb alkalmazás segítségével tudják felmásolni a távoli szerverre a szükséges fájlokat. Ilyen program lehet például a WinSCP.

Az általam fejlesztett webes felület azon próbál meg segíteni, hogy ezek a műveletek minél kevesebb időt vegyenek igénybe, valamint megpróbálja a legszükségesebb funkciókat ellátni az előbb említett több program helyett.

2. Felhasznált technológiák

2.1. Fejlesztői környezet

Az alkalmazás fejlesztése Windows platformon, NetBeans IDE 6.8 integrált fejlesztői környezet és TomCat 6.0 webszerver használatával történt. A program futtatható más webszerver használatával is, a referencia problémák kiküszöbölése után. A programkód Java Enterprise Edition technológia használatával íródott.

2.2. Java

A Java egy objektum orientált programozási nyelv, amelyet a 90-es évek elején kezdett fejleszteni a Sun Microsystems. Szintaktikája a C-re és a C++-ra hasonlít. A Java fordító a programot egy bájtkódra fordítja, amelyet a Java Virtual Machine (JVM) interpretál. Ennek a fordítási folyamatnak köszönhető a hordozhatóság, vagyis a Javaban íródott programok hasonlóan fognak futni különböző hardvereken. Ez volt az egyik fő szempont a nyelv tervezésénél. További szempontok voltak még az objektum-orientáltság, függetlenség az operációs rendszertől.

2.2.1. Java EE technológia

Az évek során a Java nyelv és a hozzá kapcsolódó technológiák folyamatos fejlődést mutatnak, valamint használatuk egyre szélesebb körben vált elterjedtté. Ez tette szükségessé, hogy a Java platformnak három kiadása jelenjen meg. Ezek a Java Standard Edition (Java SE), Java Micro Edition (Java ME) és a Java Enterprise Edition (Java EE). A Java SE segítségével hagyományos desktop alkalmazások fejlesztését teszi lehetővé. Java ME segítségével a mobil eszközökre való fejlesztés történhet. A Java EE sok felhasználós, vállalati méretű rendszerek fejlesztését teszi lehetővé. Oly módon segíti a nagyméretű rendszerek fejlesztését, hogy gyakran felmerülő problémákra kínál megoldást azáltal, hogy egy futási környezet áll rendelkezésünkre, amely különböző szolgáltatásokat nyújt a felmerülő igények kielégítésére. A következő két problémára a JavaServlet és JavaServer Pages (JSP) technológiák nyújtottak számomra megoldást:

- Többszálúság: A programtesztelő rendszert egyidejűleg több felhasználó is elérheti, ezért fel kell készíteni több kérés párhuzamos kiszolgálására, hogy a felhasználó zavartalanul dolgozhasson.
- Távoli elérés: Mivel a rendszer webes felülettel rendelkezik, az adatokat a szerveroldalon tároljuk, viszont a megjelenítés a kliensoldalon történik a böngésző segítségével. A böngésző és a szerver közötti kommunikáció a HTTP protokollon keresztül történik.

2.2.2. N rétegű architektúra

Egy Java EE alkalmazás a futásakor egy N rétegű szoftver-architektúrát alkot. Ebben a rendszerben minden egyes réteg különböző feladatot lát el, és ennek elvégzése során igénybe veszi az alatta lévő rétegek szolgáltatásit. Az architektúrát felépítő retegek:

- Kliens réteg
- Webréteg
- Adatréteg
- Üzleti logikai réteg

Az adatréteg feladata az adatok perzisztens tárolása, és a rajtuk végezhető műveletek támogatása. Ezt a réteget leggyakrabban relációs adatbázis segítségével valósítják meg, de egyre inkább kezd elterjedni az objektumorientált adatbázis használata.

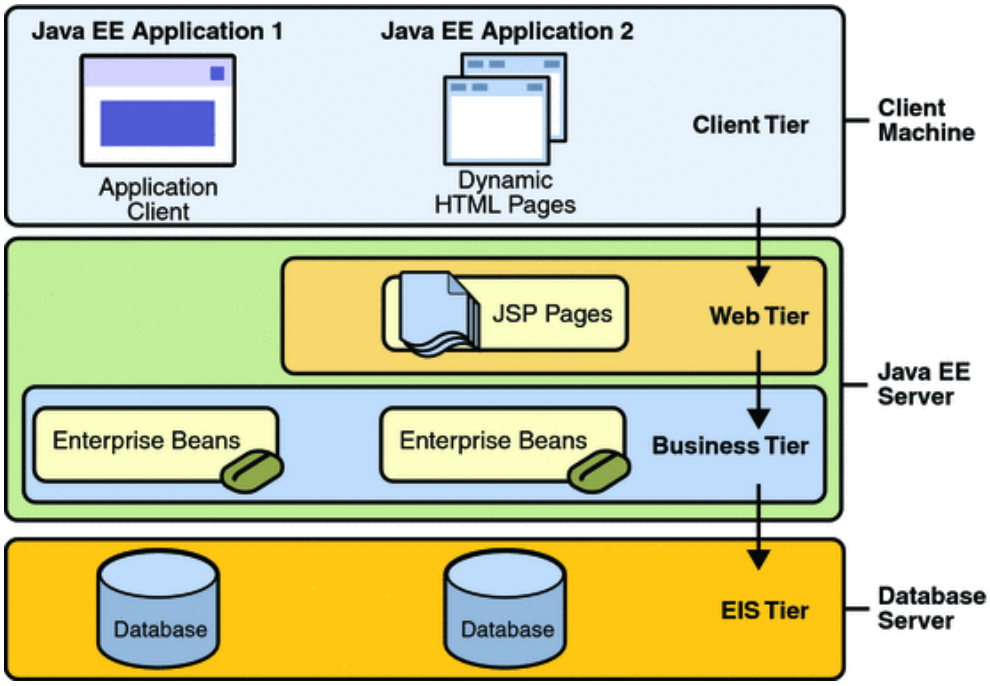
A perzisztens adattárolás lényege, hogy az adatokat a memóriában tároljuk ahelyett, hogy rögtön a relációs adatbázisba képeznénk őket. Bizonyos időközönként mentés készül a memória tartalmáról, és folyamatos esemény logolás történik, így egy esetleges rendszerhiba, vagy áramszünet esetén azonnal visszaállítható a memória eredeti állapota, és a munka zavartalanul folyhat tovább.

Az adatrétegre az üzleti logikai réteg épül, aminek feladata az alkalmazás igényeinek megfelelő funkcionalitásokat biztosítani úgy, hogy az üzleti szabályok figyelembevételével hívja meg az adatréteg szolgáltatásait.

A kliensréteg biztosítja az alkalmazás felhasználói felületét, vagyis meghívja a bekövetkezett felhasználói eseménynek megfelelő üzleti logikai funkciót. Beszélhetünk vastag kliensről, amely hagyományos desktop alkalmazást használ, valamint beszélhetünk

vékony kliensről, amely webes alkalmazás segítségével biztosítja a megjelenítést. Ilyen vékony kliens szerepel az automatikus tesztelőnél is.

Ha vékony klienst akarunk kiszolgálni, akkor szükségünk van egy webrétegre a kliens- és üzelti logikai réteg közé. Ez a réteg fogadja a HTTP-kéréseket, értelmezi azokat, meghívja a megfelelő üzleti funkciókat, majd a megfelelő HTTP-választ generálja.



1. ábra

Az 1. ábra két Java EE alkalmazást futtató környezetet szemléltet. Az első alkalmazás nem használ dinamikus oldalakat, csak egy asztali alkalmazást futtat, vagyis vastag kliens, és nincs szüksége a HTTP-kérések kiszolgálására. A második alkalmazás már egy vékony klienst futtat és igénybe veszi a webréteg szolgáltatásait. A fenti ábrán látható rétegek:

- Kliensréteg: ez a réteg az alkalmazást futtató számítógépen fut
- Webréteg: a Java EE alkalmazásszerveren fut
- Üzleti logikai réteg: a webréteghez hasonlóan, a Java EE alkalmazásszerveren fut
- Adatréteg vagy Nagyvállalati Információs Rendszer: az adatbázis szerveren fut

2.2.3. Java EE komponensek

A Java EE különböző komponensekből hozták létre, amelyek egymással folyamatosan kommunikálnak. Ezeknek a komponenseknek a funkciói be vannak építve a Java EE alkalmazásba származtatott osztályok segítségével.

A Java EE specifikáció a következő komponenseket definiálja:

- az alkalmazás kliensek azok a komponensek, amelyek a kliens gépen futnak
- a Java Servlet, JavaServer Faces (JSF) és a JavaServer Pages (JSP) olyan web komponensek, amik az alkalmazás szerveren futnak
- az Enterprise Java Beans (EJB) üzleti komponensek szintén az alkalmazás szerveren futnak

Ezeket a komponenseket Java nyelven írták, és a fordítás módja ugyanaz, mint bármely más Java nyelven megírt osztálynak. A különbség a Java EE komponensek és a többi Java nyelven megírt osztály között, hogy a komponensek beépíthetők a Java EE alkalmazásokba, mivel formailag megfelelnek a Java EE specifikációnak, és ezek a komponensek az alkalmazás szerverre lesznek telepítve, és itt futnak.

Kliens

A kliensek tehát két félék lehetnek. Beszélhetünk vastag és vékony kliensről. A vastagkliens egy asztali alkalmazást futtat, míg a vékony egy webes alkalmazást.

Vékony kliens, vagy web kliens

Két részből áll:

- A dinamikus web oldalak valamilyen leíró nyelven íródnak (például HTML, XML vagy PHP), ezeket a web rétegben futó web komponensek generálják.
- A böngésző jeleníti meg az alkalmazásszerver által küldött oldalakat.

Vastag kliens vagy alkalmazás kliens

A kliens gépen egy asztali alkalmazás fut. Legtöbbször egy grafikus felhasználói interfész (GUI) jelenik meg a felhasználó előtt, és ezt tudja manipulálni, de előfordul; hogy csak egy parancssori alkalmazás jelenik meg. Legtöbbször közvetlenül kommunikál az üzleti logikai réteggel, de ha szükséges, akkor nyithat egy HTTP kapcsolatot, hogy egy szervlet segítségével kommunikáljon a web réteggel. Ezek az alkalmazások legtöbbször Java nyelven

íródnak, de használható bármilyen másik nyelv is, amely együttműködik a Java EE alkalmazásszerverrel.

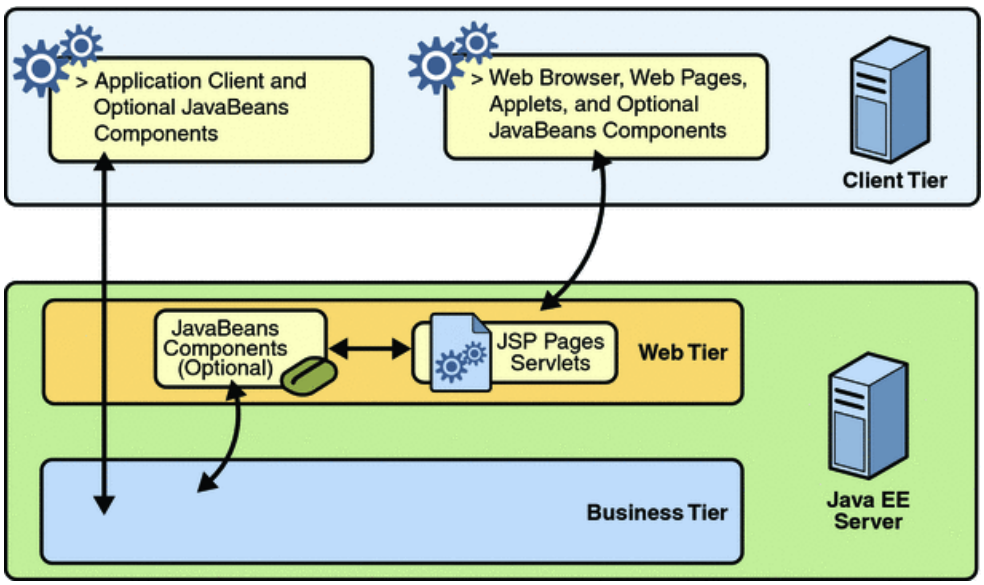
2.2.3.1. Java Beans komponens architektúra

A szerver- és a kliens réteg is tartalmazhat JavaBeans komponenst, ami irányítja az adat továbbítást a kliens és az alkalmazásszerver között, valamint a szerver komponens és az adatbázis között. Ezek a komponensek tartalmazzák az üzleti logikát, vagyis a kliensek felé felkínált metódusok implementációit. Kezelik az adatáramlást a vékony- vagy vastag kliens és az alkalmazásszerveren futó komponensek között, valamint az alkalmazásszerver és az adatbázis között.

A Java Beans komponensek állapottal rendelkeznek, amik a get és set metódusaival módosíthatók. Ezek a komponensek a metódusok használatával könnyen létrehozhatók és kezelhetők, azonban meg kell felelniük szigorú elnevezési és formatervezési követelményeknek.

A Java EE alkalmazásszerver kommunikációja

A 2. ábra azt mutatja, hogy a kliens réteg hogyan kommunikál. A vastag kliens kommunikálhat az alkalmazásszerveren futó üzleti réteggel közvetlenül, valamint a vékony kliens, ami egy böngészőben fut, a web rétegben futó szervlet vagy JSP oldalon keresztül kommunikál az üzleti réteggel.



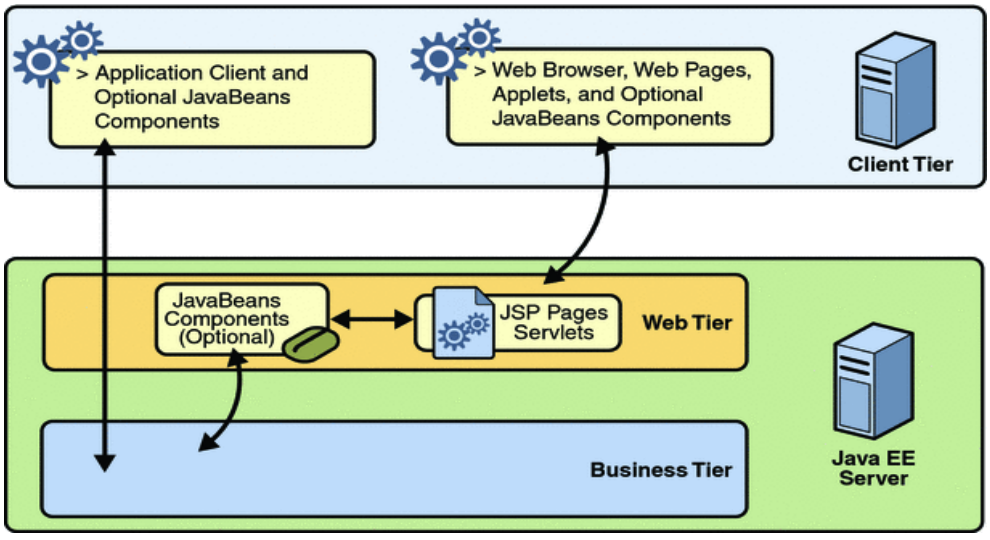
2. ábra

Egy Java EE alkalmazás vékony (böngésző alapú) klienst vagy vastag (alkalmazás alapú) klienst használ. El kell döntened, hogy melyik változatot használd, az alapján, hogy a feladatokat inkább a kliens-oldalon, közel a felhasználóhoz (vastag kliens) végezteted és így tehermentesíted a szervert vagy, hogy olyan sok funkcionalitást végeztetsz a szerveren, amennyit csak lehetséges (vékony kliens). Minél több feladatot hárítasz a serverre, annál könnyebb terjeszteni, telepíteni, és üzemeltetni az alkalmazást, jóllehet a több funkció kliens oldalon tartásával jobb felhasználói élményt lehet elérni.

Webkomponensek

Ezek a komponensek vagy szervletek, vagy pedig JSP vagy JSF technológia használatával létrehozott oldalak. A szervletek Java nyelven írt osztályok, amelyek dinamikusan feldolgozzák a kéréseket, és választ generálnak. A JSP oldalak szöveges dokumentumok, amelyek szervletekké fordulnak, és így szervletként hajtódnak végre. A JSP oldalak a statikus tartalom megjelenítésének természetesebb és egyszerűbb módjára adnak lehetőséget. A JavaServer Faces technológia a szervlet és JSP technológiára épül, és biztosít egy „user interface framework”-ot a felhasználói felület elkészítésére.

A statikus HTML lapok és appletek web komponensekbe vannak csomagolva az alkalmazásban, de nem tekinthetők web komponenseknek a Java EE specifikáció szerint. A szerver oldali „utility” osztályokat szintén web komponensekkel lehet egybe csomagolni, és a HTML lapokhoz hasonlóan szintén nem tekinthetők web komponenseknek. A következő ábra azt mutatja, a web réteg, a kliens réteghez hasonlóan tartalmazhat JavaBeans komponenst, ami kezeli a felhasználói kéréseket, majd továbbküldi az üzleti rétegnek.

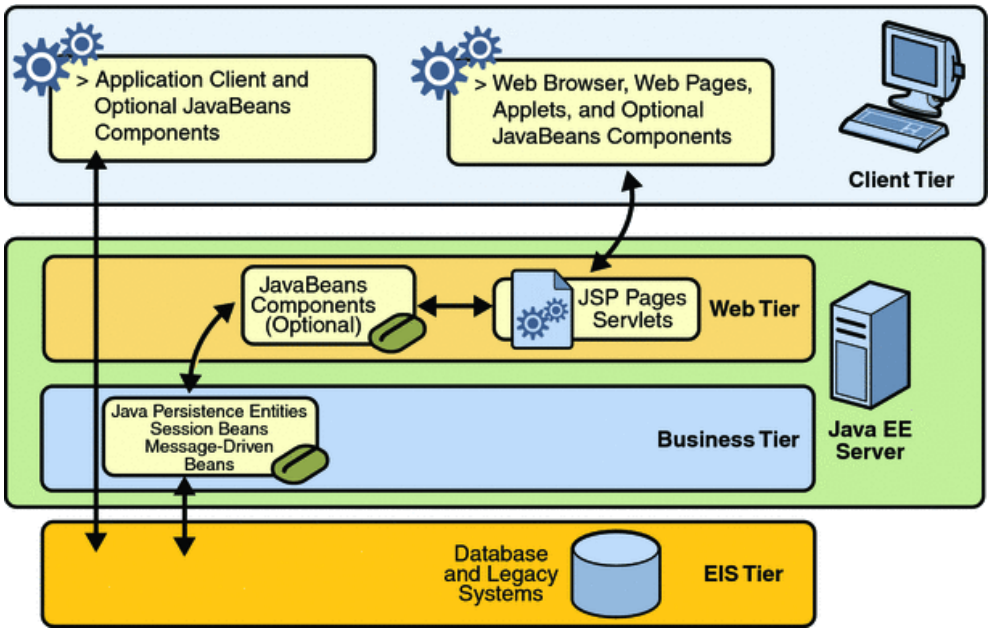


3. ábra

Üzleti komponensek

Az üzleti kód az a logika, ami megfelel az adott üzleti terület igényeinek, például egy banki, vagy kereskedelmi rendszerben az üzleti rétegben futó „enterprise bean” feladata lesz az átutalás elvégzése, aminek során ellenőrizni kell a jogosultságokat, majd az egyik számla egyenlegét csökkenteni, a másik számla egyenlegét pedig növelni kell.

A 4. ábra azt mutatja, hogy egy „enterprise bean” adatokat fogad a kliens programtól, ezeket ha szükséges feldolgozza, majd továbbküldi az adatrétegnek tárolásra, és ez az „enterprise bean” fog kommunikálni a klienssel.



4. ábra

Adatréteg

Az adatréteg olyan szoftvertámogatást nyújt (például adatbázis kezelő rendszer, vagy „legacy system”), amik az adatokon elvégezhető elemi műveletek elvégzéséért felelősek.

2.2.4. JavaEE webes technológiák

Az első szerveroldali webes technológia a szervlet technológia, majd ezt a Java Server Pages (JSP) követte.

2.2.4.1. Szervlet

Olyan Java osztály, amely a kérés-válasz modellre épül, legtöbbször HTTP kérések kiszolgálására szolgál. Segítségével dinamikus tartalmat generáló szerveroldali megoldásokat

fejleszthetünk. A szervletek a kliensektől a kéréseket és adatokat egy bemeneti csatornán kapják, valamint a kimenetet egy kimeneti csatornára írják. Ezeket paraméterként kapják meg. A szervletek mindig egy nagyobb alkalmazás részei. Ez a webalkalmazás, amely magába foglalja a szükséges erőforrásokat, más Java osztályokat, JSP oldalakat és különböző konfigurációs fájlokat (például web.xml). Egy webalkalmazás akármennyi szervletet tartalmazhat, beleértve a nullát is.

A szervletek fontos tulajdonsága, hogy háromféle életciklussal rendelkezhetnek. Inicializáció, kiszolgálás és eltávolítás. Ha kérés érkezik egy szervlet irányába, és még nincs példánya, akkor a webkonténer feladata lesz, hogy betöltse a szervlet osztályt és létrehozzon belőle egy példányt, majd meghívja az init metódusát, amelyben lefoglalhatjuk a szükséges erőforrásokat. A programozás során HTTP szervleteket használtam, amelynek lényege, hogy a kliens küld egy kérést, a szerver erre ad egy választ, és ez a kommunikáció a HTTP-protokollnak megfelelően történik. Az ilyen szervlet osztályokat a HttpServlet osztályból kell származtatni. Ezeknél a szervleteknél a kiszolgálás nem a service metódus, hanem a doGet() és doPost() metódusok segítségével történik, vagyis a webkonténer ezen metódusok valamelyikét hívja meg a kérés típusától függően. Ha egy szervlethez bizonyos időn belül nem érkezik kérés, akkor a webkonténer eltávolítja a memóriából, és meghívja a destroy metódusát, amelyben elengedhetjük a lefoglalt erőforrásokat.

Minden szervletnek valamely Servlet interfészt kell implementálnia és importálnia kell a javax.servlet, valamint javax.servlet.http csomagokat.

2.2.4.2. JSP

A JSP egy a szervleteket kiegészítő, de nem helyettesítő technológia. A szervletek a dinamikus tartalom előállítására alkalmasak, azonban egy HTML oldalnak csak egy kis része dinamikus, és ezeknek a statikus tartalmaknak az előállítása java kódba ágyazva történt a szervleteknél, ezért a kód nehezen átlátható, hosszú és megírása rendkívül körülményes. Ezt a problémát küszöbölik ki a JSP oldalak, amik tekinthetők a szervletek ellentétének is, ugyanis itt a statikus kódba ágyazzuk a dinamikus tartalmat. Egy JSP oldal tehát egy szöveges dokumentum, ami két részéből áll:

- statikus adatok (például HTML, XML kód)
- dinamikus adatokat a JSP elemek tartalmazzák

A JSP oldalak tehát nem választhatóak el a szervletektől, sőt szoros kapcsolatban állnak azokkal, ugyanis a JSP oldalak szervletekké fordulnak le. Egy adott JSP oldalra való első hivatkozáskor létrejön a neki megfelelő szervlet, majd a szervlet lefordítódik, és az így létrejövő .class fájl fog lefutni. Ez a folyamat a JSP oldal minden módosításánál újra és újra megtörténik. Amikor egy kérés fut be egy JSP oldalhoz, akkor a webkonténer ellenőrzi, hogy a JSP-hez tartozó szervlet régebbi-e, mint az oldal. Ha igen, akkor ehhez a szervlethez fog befutni a kérés, ha pedig nem, akkor az oldalt újra szervletté transzformálja, és ide fog befutni a kérés. Ezért egy JSP oldal első használatakor a felhasználói oldalon érezhető a lassabb munkamenet, ez kiküszöbölhető a JSP oldalak előfordításával, és ebben az esetben már az első kérés sem fog lassabban végrehajtódni.

A JSP elemek három félék lehetnek:

- akcióelemek
- direktívák
- szkriptelemek

A szkriptelemek és direktívák „<%” és „%>” párok között jelennek meg a kódban, míg az akcióelemek XML szintakszist követnek.

Szkriptelemek

A szkriptelemek segítenek abban, hogy az oldal dinamikus legyen. Itt helyezhetjük el például a Java kódunkat. Ma még csak Java kódot használhatunk, de a jövőbeni tervek szerint bármilyen programnyelv használható lesz. Három fajtája van: a kifejezés, szkriptlet és deklaráció.

A deklarációkban változók és metódusok deklarálhatók. Az itt deklarált metódusok nem szálbiztosak, ezért a fejlesztőnek kell gondoskodni róla. Általános alakjuk: <%! típus változónév=”érték” %>

A kifejezések tetszőleges Java utasítások lehetnek, az utasítás kiértékelődik és sztring formájában kerül a válaszba, ezért a kifejezés kiértékelésének sztringgel kell visszatérnie. Általános alakjuk: <%= kifejezés %>

A szkriptletek olyan kódrészeket tartalmaznak, amelyek belekerülnek a JSP oldalból generált szervletbe. Nem tartalmazhatnak metódus deklarációkat, és tartalmazhat statikus kódrészeket. Általános alakjuk: <% kódrész %>

Direktívák

Az oldal tulajdonságait állíthatjuk be. Három fajtája van a page, include és taglib direktíva. Általános alakjuk: <%@ direktívaneve [attributumnév = „attributumérték”] %>

Page direktíva

Mint a neve is mutatja az oldalra jellemző értékeket állíthatunk be segítségével. Fontosabb beállítható jellemzők:

- language: Itt adható meg, hogy a szkriptelemek milyen nyelven íródnak. Alapértelmezett a Java nyelv. Alakja: <%@ page language = ”java” %>
- import: Funkciója ugyanaz, mint a Java osztályokban, vagyis az itt megadott osztályok funkciói használhatóak a kódban. Vesszővel elválasztva több import is szerepelhet ugyanabban a page direktívában. Például: <%@ page import=”java.util.*” %>
- session: Itt adható meg, hogy az oldal használja-e a session objektumot. Alapértelmezett értéke „true”. Ha „false”-ra állítjuk, akkor elérjük, hogy az ide érkező kérések nem lesznek hatással a munkafolyamatra. Alakja: <%@ page session = ”false” %>
- errorpage: Az itt megadott JSP oldalra fogja megkapni a vezérlést, ha egy el nem kapott kivétel dobódik. Például: <%@ page errorpage = ”errorPage.jsp” %>

Include direktíva:

Az itt megadott fájlok a JSP oldal szervletté transzformálása előtt bemásolódnak az oldal tartalmába, vagyis az include direktíva segítségével megoldható például az, hogy egy JSP oldal egy másik JSP oldalt tartalmazzon. Vigyázni kell arra, hogy ha a tartalmazott JSP oldalt módosítjuk, de a tartalmazó oldalt nem, akkor a változtatás nem fog megjelenni a számunkra, mivel a tartalmazó JSP oldal régebbi marad, mint a hozzá tartozó szervlet. Példa az include direktívára: <%@ include file=”tartalmazott.jsp” %>

Taglib direktíva:

Felhasználó által definiált elemek használatára ad lehetőséget. Használatuk és szintaktikájuk hasonló az akcióelemekéhez. Nagy előnyük, hogy összetett üzleti logikát

rejthetünk egyszerű utasítások jelölések mögé, és ezzel javíthatjuk a kód átláthatóságát és olvashatóságát. JAR fájlban tárolva hordozhatóvá tehetők és újrafelhasználhatóak.

Akcióelemek

Az akcióelemek XML-szintaxist követik, tipikusan valamilyen akció végrehajtására szolgálnak. Az alapvető akcióelemeken kívül definiálhatunk saját akcióelemet is. Általános alakjuk: <jsp:elem attribútum=”érték” [attributum2=”érték2”]…/>

Ha az akcióelem törzzsel is rendelkezik, akkor a következőre módosul az általános alakja:

<jsp:elem attribútum=”érték” [attributum2=”érték2”]…>
</jsp:elem>

Gyakran használt elemek:

Include:

Segítségével egy JSP oldalba egy másik oldalból, vagy szervletből származó választ illeszthetünk be. A végrehajtás folytatása függ attól, hogy szervlet, JSP vagy pedig HTML-oldalt illesztettünk-e be. HTML esetén az oldal tartalma beszúródik a válaszba. Ha szervelet vagy JSP oldal található az include paraméterében, akkor azok kimenete szúródik be az oldalba. Alakja: <jsp:include page=”oldal” flush=”true/false” >

<jsp:param name=”név” value=”érték”/>
</jsp:include>

Forward:

Használatával a vezérlést egy JSP oldalnak vagy egy szervletnek adhatjuk át. Alakja:

<jsp:forward page=”oldal” >
<jsp:param name=”név” value=”érték”/>
</jsp:forward>

UseBean, getProperty és setProperty

A useBean segítségével egy JavaBean komponens tölthető be, míg a másik két elem segítségével ennek a beannek a tulajdonságai módosíthatóak. A JavaBeanek olyan osztályok, amelyek példányai jellemzőkkel rendelkeznek, és megfelelnek néhány követelménynek. Például minden beannek rendelkeznie kell egy paraméter nélküli konstruktorral, amely automatikusan meg fog hívódni, ha az adott nevű JavaBean még nem létezett. Általános alakja: <jsp:useBean id=”név” scope=”érték” [class=”osztály” / type=”osztály”]/>

A scope attribútum határozza meg a bean hatáskörét, vagyis hogy az adott nevű bean keresése hol történik. Értékei a következők lehetnek:

- page: A bean az adott JSP oldalon és az oldal által tartalmazott fájlokban érhető el.
- request: Minden JSP oldalról elérhető a bean, ami megkapta a kérést.
- session: Minden olyan JSP oldalról elérhető a bean, amely nem tiltotta le a session objektumon.
- application: Az alkalmazásban található összes JSP oldalról elérhető a bean.

3. Programtesztelő rendszer

Munkám során egy, az automatikus programtesztelő rendszert kiegészítő webes felületet hoztam létre, amely lehetőséget nyújt a hallgatónak, hogy a szerveren ne parancssorból futtatható parancsokkal, hanem egy weblapon a lehető legkevesebb gépeléssel végezze el a programok feltöltését, tesztelését és beadását, és a rendszer válaszát is ezen a webes felületen jeleníti meg.

3.1. Bejelentkezés

A tesztelő elindulásakor az index.jsp oldal fogadja felhasználót. Azt, hogy mi legyen a nyitó oldal, a web.xml konfigurációs fájlban állíthattam be.

```
<welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
```

web.xml:

A web.xml konfigurációs és telepítési információkat tartalmaz az alkalmazásban szereplő webes összetevőkkel kapcsolatban. Például szervlet és JSP definíciókat tartalmaz, valamint itt hozhatunk létre új paramétereket a szervlet osztályoknak, amelyeknek értéket is adhatunk. A web.xml fájlt a könyvtár hierarchiában a web könyvtáron belül a WEB-INF könyvtárban található.

index.jsp

Az index.jsp oldalba az include akcióelem segítségével a login.jsp oldal van beágyazva, így a vezérlés ide adódik át.

```
<jsp:include page="login.jsp"/>
```

login.jsp

A felhasználótól JSP oldal egy felhasználónév és jelszó bemenetet vár, és ezek megadása után a Belépés gombra kell kattintani. Ennek hatására a vezérlés a loginAction.jsp oldalnak adódik át.

login.jsp

loginAction.jsp

A megadott felhasználónév és jelszó segítségével próbál a hallg.inf.unideb.hu címen elérhető szerverhez csatlakozni. Ezt az ssh csomagban lévő sshConnect osztály connect metódusának felhasználásával teszi meg.

```
public SshConnect(String username, String password) throws SshException {
                this.username = username;
                this.password = password;
                session = new SshSession(this.hostname, this.username,
this.password);
                session.setShellPrompt(shellPrompt);
                session.addSshListener(this);
                session.connect();
}
```

Ha a kapcsolat létrehozása sikertelenül fejeződik be, akkor egy SshException kivétel dobódik, amely továbbítódik a hívó JSP oldalnak, a loginAction.jsp-nek, amely a kivétel hatására átadja a vezérlést a loginException.jsp oldalnak.

```
<%@page errorPage="loginException.jsp" %>
```

loginException.jsp

Ha a kapcsolat létrejött, akkor beállítja a username és ssh attribútumokat, majd továbbadja vezérlést a forward akcióelem segítségével a welcome.jsp-nek.

```
<% SshConnect sshconnect = new
SshConnect(request.getParameter("username"),
                request.getParameter("password"));
                if(sshconnect.connect){
                session.setAttribute("ssh", sshconnect);
                session.setAttribute("username", request.getParameter("username"));
%>
<jsp:forward page="welcome.jsp" />
<% }%>
```

3.2. sshConnect osztály

Az sshConnect osztály a com.jscape.inet.ssh csomagot használja. Az osztály hat osztályszintű változóval rendelkezik, amelyek segítségével a kapcsolat aktuális állapotát tároljuk. Itt tároljuk, hogy a szerver milyen címen érhető el, az aktuális felhasználónevet és jelszavat, valamint egy shellPrompt nevű változóban azt, hogy milyen promptra várunk a szervertől, valamint a connected változó segít eldönteni, hogy a kapcsolat él e. A legfontosabb szerepe a változók közül a session nevű és SshSession típusú változónak van, amelynek segítségével történik a parancs elküldése a szerver felé.

goToDir metódus

Ez a metódus a feladat számát tartalmazó sztringet feldarabolja a „/” jelek mentén, majd a feladatsor és a feladat számát egy tömbben tárolja. Ezek után először megvizsgáljuk, hogy a felhasználó gyökérkönyvtárában van-e a feladatsor számával megegyező nevű könyvtár. Ezt az isDir metódus segít eldönteni. Ha van ilyen könyvtár, akkor egy „cd könyvtárnév” parancsot küldünk a szervernek, és így ez a könyvtár lesz az aktuális. Ez után az isDir metódus segítségével megnézzük, hogy itt találunk-e a feladat számának megfelelő könyvtárat. Ha találtunk, akkor újra egy „cd” parancsot küldünk a szervernek, és ez lesz az aktuális könyvtár. Amennyiben valamelyik könyvtár nem létezett, akkor a goToDir metódus „false” értékkel tért vissza.

```
public boolean goToDir(String feladatszam) throws SshException {
                String cmd = "";
                String[] result = feladatszam.split("/");
                for (int x = 1; x < result.length; x++) {
                      cmd = "cd " + result[x];
                      if (!this.isDir(result[x])) {
                              return false;
                      }
                String answer = encoding(session.send(cmd));
                System.out.println(answer);
                }
                return true;
}
```

isDir metódus

Az isDir metódus tehát eldönti, hogy az aktuális könyvtárban a paraméterként kapott nevű könyvtár létezik e. Egy unix parancsot küld a szervernek, amely ls, grep és wc utasításokból áll. A parancs úgy működik, hogy az „ls” kilistázza az aktuális könyvtár tartalmát, majd a „grep” az előző parancs kimenetében keresi a keresendő könyvtárat, és visszaadja azoknak a nevét, a „wc –l” parancs ezeket a visszaadott sorokat számolja meg. Ezután egy feltételes utasítással megnézem, hogy az elküldött parancsra kapott válasz egyenlő-e nullával. Ha egyenlő, akkor nem létezik az aktuálisban a keresett könyvtár, ellenkező esetben létezik.

```
public boolean isDir(String dir) throws SshException {
		String cmd = "ls | grep " + dir + " | wc -l";
		String answer = encoding(session.send(cmd));
		System.out.println(answer);
		if (Integer.parseInt(answer.substring(0, 1)) != 0) {
			return true;
		}
		return false;
}
```

Teszt metódus

A Teszt metódus a feladat számát kapja meg sztringként egy paraméterben, hogy a feladat számának megfelelő könyvtárra pozícionáljon a goToDir metódus segítségével. Ezután egy „prog1-teszt” parancsot küld a szervernek a feladata számát adja paraméterül, és a kapott választ megfelelő karakterkódolásúra alakítja, majd visszatér ezzel a sztringgel.

```
public String Teszt(String feladatszam) throws SshException {
		if (this.goToDir(feladatszam)) {
			String cmd = "prog1-teszt " + feladatszam;
			String answer = encoding(session.send(cmd));
			System.out.println(answer);
			return answer;
		}
		return "Nincs ilyen könyvtár!";
}
```

Bead metódus

A Teszt metódussal szinte teljesen azonos felépítésű és működésű, a különbség mindössze annyi, hogy a szervernek elküldött unix parancs nem tesztelésre, hanem a feladat beadására vonatkozik.

```
public String Bead(String feladatszam) throws SshException {
                if (this.goToDir(feladatszam)) {
                String cmd = "prog1-bead " + feladatszam;
                String answer = encoding(session.send(cmd));
                System.out.println(answer);
                return answer;
                }
                return "Nincs ilyen könyvtár!";
}
```

Results metódus

A Results metódus a felhasználó eredményeit listázza ki. Feladata, hogy elküldjön egy „prog1-eredmeny” parancsot, és a válaszul kapott sztringgel térjen vissza.

```
public String Results() throws SshException {
                String cmd = "prog1-eredmeny";
                String answer = encoding(session.send(cmd));
                System.out.println(answer);
                return answer;
}
```

createDir metódus

Ennek a metódusnak a fájlok feltöltésénél van fontos feladata, ugyanis itt alakul ki a megfelelő könyvtár hierarchia, ami a teszteléshez és a beadáshoz nélkülözhetetlen.

A metódus lényege, hogy megkapja a feladat számát „pti/feladatsor/feladat” formában, és ezt a sztringet darabolja a „/” jel mentén, és a darabokat egy tömbben helyezi el, amely így a feladatsor, és a feladat számát külön fogja tartalmazni.

Ezek után ellenőrzi, hogy létezik-e már a felhasználó gyökérkönyvtárában a feladatsor számával megegyező nevű mappa. Ha létezik, akkor az lesz az aktuális könyvtár, ha nem létezik, akkor létrehozza azt.

Majd ellenőrzi, hogy az aktuális könyvtárban már van-e a feladat számával megegyező könyvtár. Ha van, akkor azt törli, majd újra üresen létrehozza. Ha nem létezik, akkor létrehozza.

A metódus utolsó sora arra szolgál, hogy újra a felhasználó gyökérkönyvtára legyen az aktuális könyvtár, a többi művelet zavartalan elvégzése érdekében.

```
public void createDir(String dir) throws SshException {
		String[] result = dir.split("/");
		if (!this.isDir(result[1])) {
			session.send("mkdir " + result[1]);
			session.send("cd " + result[1]);
			session.send("mkdir " + result[2]);
		} else {
			session.send("cd " + result[1]);
			if (!this.isDir(result[2])) {
				session.send("mkdir " + result[2]);
			} else {
			session.send("rm -r " + result[2]);
			session.send("mkdir " + result[2]);
			}
		}
		session.send("cd /home/"+username);
}
```

exit metódus

A rendszerből való kilépéskor használt metódus, amely egy „exit” metódust küld a szervernek. Ekkor van jelentősége az SshSession osztály sendNoWait metódusának, aminek hatására nem várunk a $ promptra, ha ezt tennénk, akkor egy végtelen ciklusba jutnánk.

3.2.1. Karakterkódolási probléma

Kezdetben a karakterkódolás csak az angolszász karakterek megfelelő megjelenítésével foglalkozott, azonban a számítástechnika egyre szélesebb körű elterjedésével megnőtt az igény, hogy más nyelvek speciális karakterei is megjeleníthetőek legyenek, ezért különböző kódolási formák, és szabványok jelentek meg. Az ASCII kódtábla tartalmazta az összes angol nyelvhez szükséges betűt, majd megjelent az úgynevezett Latin-1, vagyis az ISO-8859-1, ami már számos európai nyelv betűit tartalmazta, de nem volt alkalmas például az „ő” és „ű” betűk megjelenítésére. Később megjelent a Latin-2, vagyis az ISO-8859-2, aminek segítségével már a legtöbb közép-európai országban használatos betűk megjeleníthetőek, viszont nem ismeri több ország, például Görögország, vagy India speciális írásjeleit. Az Unicode és az UTF-8 kódolás viszont már egységesen támogatja ezeknek a karaktereknek a megjelenítését is. A problémát az jelenti a programozók számára, hogy ezeknek a kódolási eljárásoknak a karakterábrázolási technikája eltérő ugyanis, míg az UTF-8 változó hosszúságú kódolást használ, addig az ISO szabványok fix 8 biten ábrázolják a karktereket.

3.2.1.1. Karakterkódolás módszerek

Unicode

Különböző nyelvekben használatos írásjelek egységes kódolását leíró nemzetközi szabvány. A szabvány a karakter kódoláson túl még az osztályozással, megjelenítéssel és használattal is foglalkozik. Régebben az UTF-32 volt használatos, azonban ez minden karakterhez 4 bájtot használt, így nagy mértékű volt a pazarlás. Az UTF-8 a manapság használatos Unicode szabvány.

UTF-8 (8 bites Unicode átalakítási formátum)

Változó hosszúságú karakterkódolási eljárás, amely bármilyen Unicode karaktert képes ábrázolni, és kompatibilis a 7 bites ASCII szabvánnyal.

A kódolás lényege, hogy az ASCII kódtábla karaktereit az ASCII kódjukkal jelöli, a többi karakter kódját pedig feldarabolja, és ezeket a darabokat valamilyen vezérlőjel utáni több bájtban helyezi el, úgy a bájtok mindegyike 127 feletti értékkel rendelkezik, így nem keverhető össze ASCII kódtábla egyetlen karakterével sem. Ez biztosítja, hogy az UTF-8 kódolás lefelé kompatibilis legyen az ASCII kódolással, vagyis minden 7 bites ASCII

karakter kódja saját maga marad. Ha tévesen valamilyen más kódolást próbálunk használni például egy magyar szöveg esetén, akkor csak az ékezetes karakterek esetén érzékelhetünk problémát, mivel ezeket két egymás utáni bájtban tárolja, ezért például egy „Á” betű helyett két olvashatatlan értelmezhetetlen kapnánk.

Egy UTF-8-as kód bináris alakjában a 0-val kezdődő bájtok a 7 bites ASCII kódtábla karaktereit jelöli, ha 11-el kezdődik a bájt, akkor ott valamilyen speciális karakter lesz több egymás követő bájton tárolva. Ekkor a egyesek száma adja meg, hogy hány bájton lesz tárolva a karakter. A 10-el kezdődő bájtok pedig az ilyen több bájton ábrázolt karakterek folytatását jelölik.

Char	Code	Char	Code	Char	Code	Char	Code	Char	Code	Char	Code	Char	Code	Char	Code
Ā	100	Đ	110	Ę	118	Ķ	136	Ń	143	Ó	d3	Ś	15a	Ű	170
ā	101	đ	111	ę	119	ķ	137	ń	144	ó	f3	ś	15b	ű	171
Ă	102	Ď	10e	Ě	11a	Ĺ	139;	Ņ	145	Œ	152	Š	160	Ų	172
ă	103	ď	10f	ě	11b	Í	13a	ņ	146	œ	153	š	161	ų	173
Ą	104	Ē	112			Ļ	13b	Ň	147	ŕ	155	Ţ	162	Ŷ	178
ą	105	ē	113	Ģ	122	ļ	13c	ň	148	Ŗ	156	ţ	163	Ź	179
Ć	106	ĕ	115	ģ	123	Ľ	13d	Ō	14c	ŗ	157			ź	17a
ć	107	Ė	116	Ī	12a	ľ	13e	ō	14d	Ř	158	ť	165	Ż	17b
Č	10c	ė	117	ī	12b			Ő	150	ř	159			ż	17c
č	10d			Į	12e	Ł	141	ő	151	Ş	15e			Ž	17d
				į	12f	ł	142			ş	15f			ž	17e

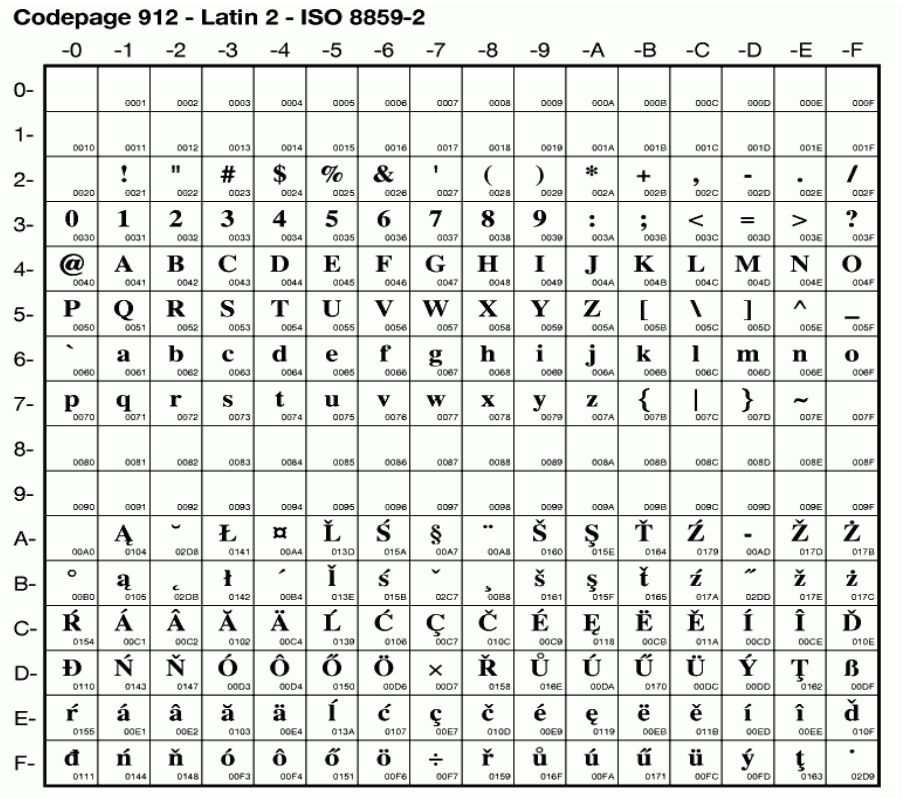
5. ábra: Az UTF-8 kódtábla egy részlete

ISO-8859

Más néven ISO/IEC 8859-es szabvány, amely 8 bites karakterkódolást használ. Ez a szabvány több egymástól független formában is megjelent, melyek mind más-más terület karaktereinek ábrázolására használhatók. A közép-európai nyelvek esetén az ISO 8859-2-es kódtábla használatos. Ez a szabvány az UTF-8-al ellentétben nem foglalkozik a karakterek osztályozásával, rendszerezésével, csak a kódolásra koncentrál. Azok a karakterek, amelyek nem részei egy széles körben kódkészletnek, vagy nem találhatóak meg a nemzeti nyelv leírásánál használt billentyűzeten, nem kerültek be a kódtáblába.

ISO-8859-2

Tartalmazza a latin ábécét használó Közép- és Kelet-európai nyelveket. Ilyen nyelvek a bosnyák, lengyel, horvát, cseh, szlovák, szlovén, román, szerb és a magyar. Ezen kívül még néhány nyugat európai nyelv leírására is alkalmas, mint például a finn. A legtöbb karaktertáblából kimaradt az „ő” és „ű” betű, azonban ez a szabvány támogatja ezeknek a karaktereknek a megjelenítését.



	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	000A	000B	000C	000D	000E	000F
1-	0010	0011	0012	0013	0014	0015	0016	0017	0018	0019	001A	001B	001C	001D	001E	001F
2-	0020	! 0021	" 0022	# 0023	$ 0024	% 0025	& 0026	' 0027	(0028	) 0029	* 002A	+ 002B	, 002C	- 002D	. 002E	/ 002F
3-	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	: 003A	; 003B	< 003C	= 003D	> 003E	? 003F
4-	@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
5-	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	\ 005C	] 005D	^ 005E	_ 005F
6-	` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F
7-	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	| 007C	} 007D	~ 007E	007F
8-	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	008A	008B	008C	008D	008E	008F
9-	0090	0091	0092	0093	0094	0095	0096	0097	0098	0099	009A	009B	009C	009D	009E	009F
A-	00A0	Ą 0104	˘ 02D8	Ł 0141	¤ 00A4	Ľ 013D	Ś 015A	§ 00A7	¨ 00A8	Š 0160	Ş 015E	Ť 0164	Ź 0179	- 00AD	Ž 017D	Ż 017B
B-	° 00B0	ą 0105	˛ 02DB	ł 0142	´ 00B4	ľ 013E	ś 015B	ˇ 02C7	¸ 00B8	š 0161	ş 015F	ť 0165	ź 017A	˝ 02DD	ž 017E	ż 017C
C-	Ŕ 0154	Á 00C1	Â 00C2	Ă 0102	Ä 00C4	Ĺ 0139	Ć 0106	Ç 00C7	Č 010C	É 00C9	Ę 0118	Ë 00CB	Ě 011A	Í 00CD	Î 00CE	Ď 010E
D-	Đ 0110	Ń 0143	Ň 0147	Ó 00D3	Ô 00D4	Ő 0150	Ö 00D6	× 00D7	Ř 0158	Ů 016E	Ú 00DA	Ű 0170	Ü 00DC	Ý 00DD	Ţ 0162	ß 00DF
E-	ŕ 0155	á 00E1	â 00E2	ă 0103	ä 00E4	ĺ 013A	ć 0107	ç 00E7	č 010D	é 00E9	ę 0119	ë 00EB	ě 011B	í 00ED	î 00EE	ď 010F
F-	đ 0111	ń 0144	ň 0148	ó 00F3	ô 00F4	ő 0151	ö 00F6	÷ 00F7	ř 0159	ů 016F	ú 00FA	ű 0171	ü 00FC	ý 00FD	ţ 0163	˙ 02D9

6. ábra: Az ISO 8859-2-es kódtábla egy részelete

encoding metódus

Ez a metódus az SshConnect osztályban található és minden, a hallg.inf.unideb.hu címen található szervertől kapott üzenetre meghívjuk. Ennek oka, hogy a szerveren beállított karakterkódolás UTF-8, így az ékezetes betűket hibásan jelenítené meg a böngészőnk.

A kapott üzenetet sztringként egy paraméterben kapja meg a metódus, amely átalakítja azt byte kóddá, majd ezt a kódot visszaalakítja ISO-8859-2 karakterkódolás használatával, amely már az ékezetes betűket a kívánt módon jeleníti meg.

```
public String encoding(String answer) {
		InputStreamReader reader = null;
		String s = "";
```

```
            try {
                InputStream bi = new ByteArrayInputStream(answer.getBytes());
                reader = new InputStreamReader(bi, "UTF-8");
                Reader in = new BufferedReader(reader);
                ArrayList<Byte> buffer = new ArrayList<Byte>();
                int ch;
                while ((ch = in.read()) > -1) {
                        buffer.add((byte) ch);
                }
                byte[] b = new byte[buffer.size()];
                for (int i = 0; i < buffer.size(); i++) {
                        b[i] = buffer.get(i);
                }
                s = new String(b, "ISO-8859-2");
                in.close();
            } catch (Exception ex) {
            } finally {
                return s;
            }
}
```

3.3. Az SshSession osztály

Az SshSession osztály a com.jscape.inet.ssh csomagban található. Ennek az osztálynak a használatával tudunk kényelmesen információt küldeni és fogadni a szervertől.

Az osztályt egy példányának előállításához három konstruktor áll rendelkezésünkre:

- Megadhatjuk a host címet, felhasználó nevet, jelszavat
- Az előzőek mellé megadhatunk még port számot is.
- Megadhatunk egyetlen SshParameters típusú paramétert, amely tartalmazza a kapcsolat létrehozásához szükséges összes információt.

Az osztály általam használt metódusai:

- addSshListener(SshListener listener): A listener maga az SshConnect osztály, aminek a feladata a kimenő és érkező adatok kezelése.
- connect(): Létrehozza a kapcsolatot a szerverrel. Megadhatunk neki paraméterként milliszekundumban egy időkorlátot is.
- disconnect(): Bontja a kapcsolatot a szerverrel.
- send(String command): A command paraméterben megadott parancsot küldi el a szervernek. Visszatérési értéke String, amiben a küldött parancsra adott válasz van.
- sendNoWait(String command): A send metódushoz hasonló, viszont ennek végrehajtása után nem várunk a promptra. Ezzel a metódussal adható ki például az „exit” parancs.
- setShellPrompt(String prompt)

3.4. A runScp és az ScpTest osztály

Ez a két osztály az scp csomagban helyezkedik el, segítségükkel történik meg a távoli szerverre a fájlok másolása.

A runScp osztály run metódusa két sztirnget kap paraméterként. Az első sztring a szerver host címét, a felhasználó nevét és jelszavát, valamint a célkönyvtárat tartalmazza a következő formában:

felhasználó:jelszó@host:/home/felahsználó/feladatsor/feladatszám.

A run metódus létrehoz egy ScpTest példányt, majd meghívja az ScpTest osztály Upload metódusát. Ha minden rendben zajlott, és nem történt semmi kivételes esemény, akkor a "A feladat feltöltése sikeresen befejeződött!" sztringgel tér vissza, ellenben egy "A feladat feltöltése közben hiba történt" sztring lesz a visszatérés értéke.

Az ScpTest osztály fogja elvégezni a tényleges másolást. Az osztály konstruktora meghívja szülőosztályának, a TestCase osztálynak a konstruktorát, majd a kapott paramétereket tárolja az osztályszintű változókban.

Az Upload metódus meghívja a createTask metódust, ami beállítja a kapcsolat portját és a host címet. Az Upload metódus ezek után beállítja a fájlok elérési útját, valamint a célkönyvtárat, és a kapcsolat további adatait, majd végrehajtja a másolást.

```
public class ScpTest extends TestCase {

        private File tempDir;
        private String sshHostUri;
        private int port=22;
        private String knownHosts;

        public ScpTest(String testname, String host, String file) {
                super(testname);
                this.sshHostUri=host;
                this.tempDir=new File(file);
        }

        public void Upload() throws IOException {
                Scp scp = createTask();
                FileSet fileset = new FileSet();
                fileset.setDir(tempDir);
                scp.addFileset(fileset);
                scp.setTodir(sshHostUri);
                scp.execute();
        }


        private Scp createTask() {
                Scp scp = new Scp();
                Project p = new Project();
                p.init();
                scp.setProject(p);
                if (knownHosts != null) {
                        scp.setKnownhosts(knownHosts);
                } else {
                        scp.setTrust(true);
```

```
                }
                scp.setPort(port);
                return scp;
            }
    }
```

3.5. Felhasználói felület

welcome.jsp

A sikeres bejelentkezés után tehát az welcome.jsp oldalhoz továbbítódik a vezérlés, ahol egy három részből álló képernyő fogadja a felhasználót.

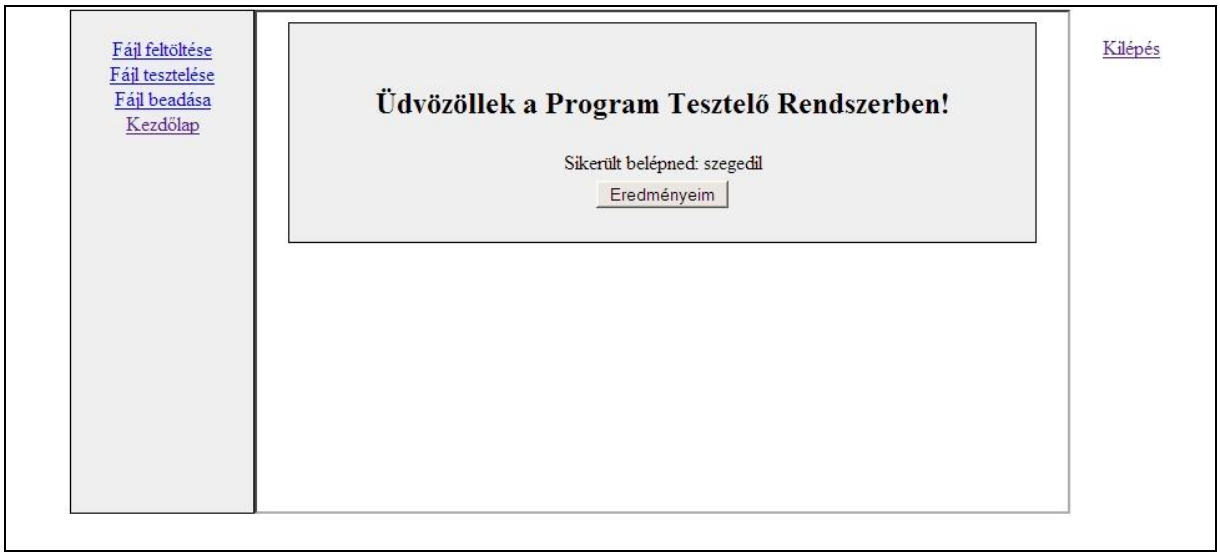
A képernyő bal oldalán van lehetőség kiválasztani, hogy programot feltölteni, tesztelni vagy pedig beadni akarunk, valamint egy hivatkozással visszatérhetünk később az oldal kezdeti állapotához.

A képernyő középső részén kezdetben a default.jsp oldal töltődik be. Ez egy üdvözlő képernyő, amelyen egy nyomógomb segítségével a felhasználónak lehetősége van lekérni az eddigi eredményeit. Ez a jsp oldal az üdvözlő szövegen kiírja a bejelentkezett felhasználó nevét is, ehhez a „username” attribútumra van szüksége.

```
<tr align="center"><td><h2>Üdvözöllek a Program Tesztelő Rendszerben!
</h2></td></tr>
<tr align="center"><td>Sikerült belépned: <%= session.getAttribute("username")%>
</td></tr>
```

A default.jsp helyén fognak megjelenni azok a lapok, amelyeknek a jobb oldalon elhelyezkedő hivatkozására rákattint a felhasználó.

A képernyő jobb oldalán a kilépés gombot találjuk.



welcome.jsp

3.5.1. Fájl feltöltése

Ha az ablak bal oldalán lévő „Fájl feltöltése” hivatkozásra kattintunk, akkor a vezérlés az upload.jsp oldalnak adódik át, és a böngésző középső részén megjelenő űrlapon a felhasználó a feladat számát és a feltöltendő feladat helyét adhatja meg.

A feladat számát „pti/feladatsor/feladatszám” formában kell megadni, a feltöltendő fájlok helyének megadásakor egy fájlkezelő ablak nyílik meg, ahol a main.java fájl elérési útját kell megadnunk, és a szükséges fájloknak ugyanabban a könyvtárban kell lenniük.

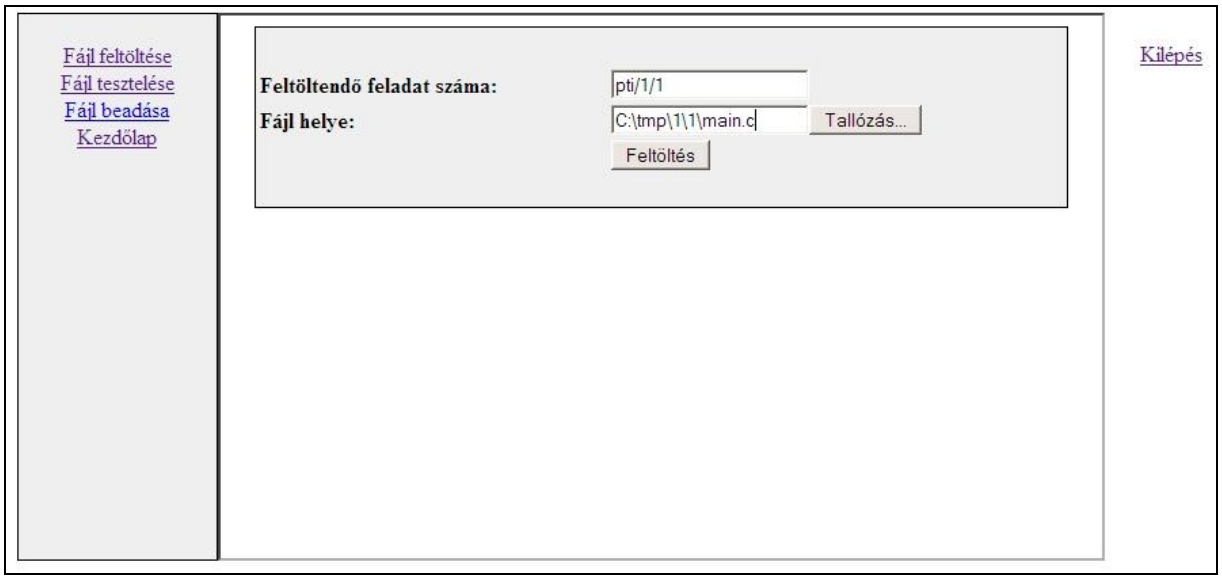
Ha ezeket a mezőket kitöltöttük, és rákattintottunk a „Feltöltés” gombra, akkor az uploadAction.jsp oldal jelenik meg, amelynek feladata a megfelelő paraméterek beállítása, a feltöltési könyvtár létrehozásához, és a feltöltéshez szükséges metódusok meghívása.

```
<%  SshConnect ssh = (SshConnect) session.getAttribute("ssh");
String psw = ssh.password;
String user = ssh.username;
String host = ssh.hostname;
String file = request.getParameter("file");
file = file.substring(0, file.lastIndexOf("\\"));
String[] result = request.getParameter("feladat_szam").toString().split("/");
String fsor = result[1];
String fszam = result[2];
ssh.createDir(request.getParameter("feladat_szam").toString());
```

```
String arg = user+":"+psw+"@"+host+":/home/"+user+"/"+fsor+"/"+fszam;
%>

<%= runScp.run(arg, file) %>
```

Ez az oldal a megadott elérési utat tartalmazó sztring végéről levágja a „\main.java” sztringet, majd a feladat számából meghatározza, hogy melyik feladatsornak, melyik feladatát szeretnénk feltölteni. Ezek után meghívja az SshConnection osztály createDir metódusát, majd a megfelelő paraméterrel meghívja a runScp osztály run metódusát, aminek kimenetét az oldalba generálni fogja.



upload.jsp

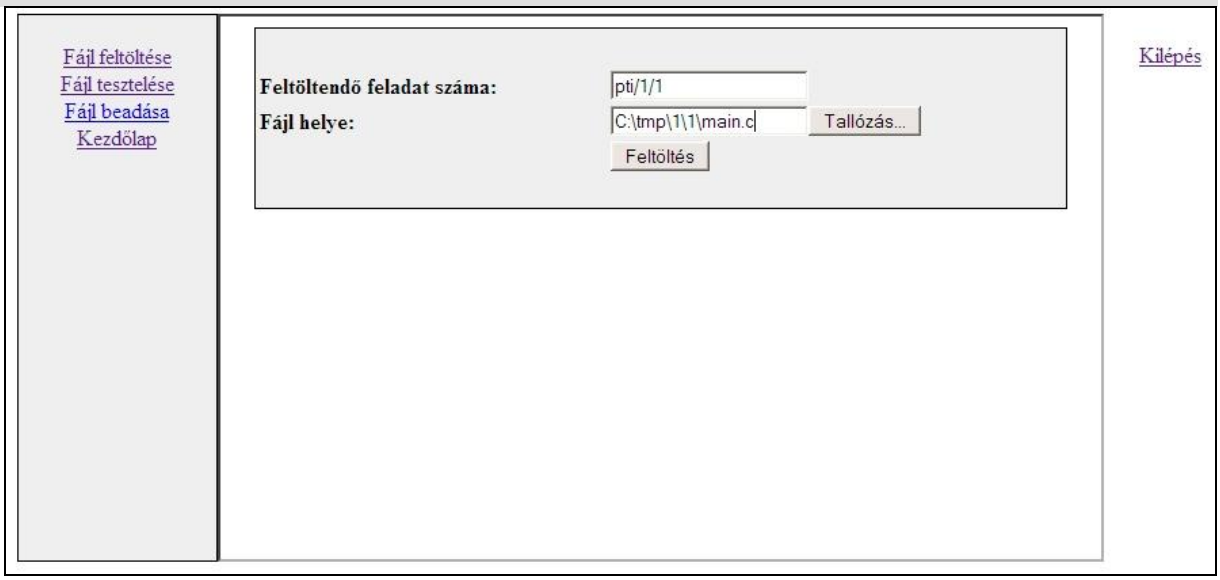
3.5.2. Feladat tesztelése

Amikor rákattintunk „Feladat tesztelése” hivatkozásra, akkor az ablak középső részén a teszt.jsp oldal fog megjelenni. Ez az oldal tartalmaz egy mezőt, ahova a tesztelendő feladat számát kell beírnunk (például pti/1/2), majd rá kell kattintanunk a „Tesztelés” gombra. Ekkor a vezérlést és a „feladat_szam” nevű attribútumot a „TesztSzervlet” osztály kapja meg. A szervlet az „SshConnection” osztály Teszt metódusát hívja meg, és a kapott sztringet fogja kiírni a képernyőre.

```
out.println(ssh.Teszt(request.getParameter("feladat_szam")));
```

Mivel ez az sshConnection osztálynak ez a Teszt metódusa meghívja a goToDir metódust, és ezáltal a megfelelő könyvtárba lép, ezért szükség van arra, hogy a tesztelés elvégzése, és az eredmény képernyőre írása után visszalépjünk a felhasználó gyökérkönyvtárába. Ennek elérése érdekében küldünk a szervernek egy olyan cd unix parancsot, amely a /home/felhasználó könyvtárat teszi aktuálissá:

```
ssh.session.send("cd /home/"+request.getSession().getAttribute("username"))
```



upload.jsp

3.5.3. Feladat beadása

Ha a bal oldali hivatkozások közül a Feladat beadására kattintunk, akkor a tesztelésnél ismertetett ablakhoz hasonló jelenik meg a képernyő középső részén. Itt is egy mezőt kell kitöltenünk, amelybe az előzőhöz hasonlóan a feladat számát kell beírnunk, majd a „Bead” gombra kell kattintani.

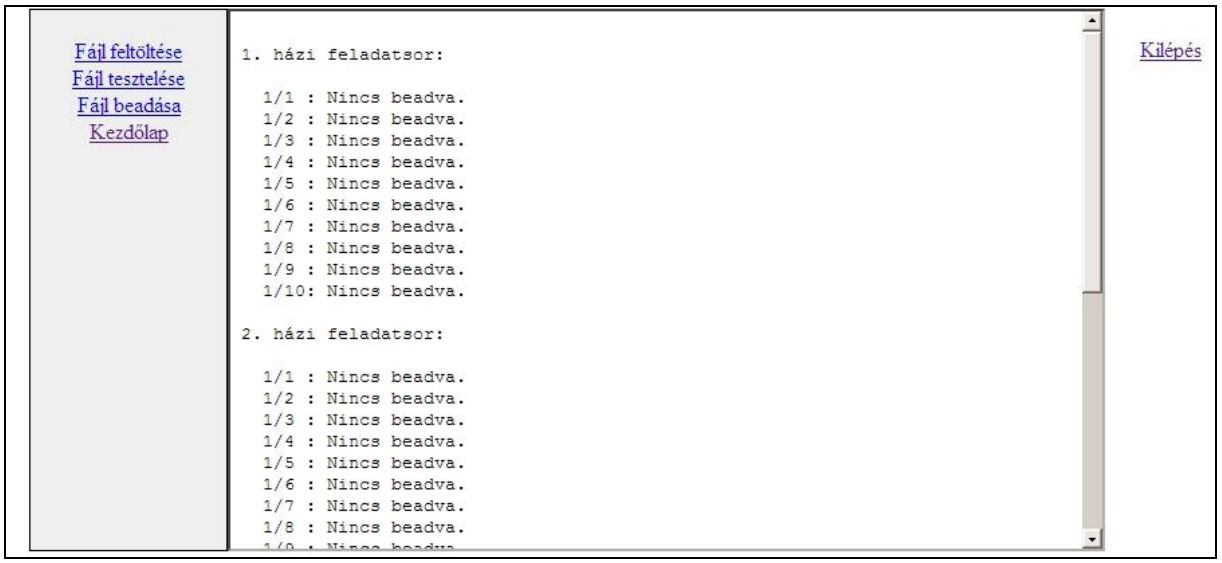
A feladat beadása a teszteléshez hasonlóan működik. Ugyanis a bead.jsp-n megjelenő mező kitöltése és a gombra kattintás utána a vezérlés a „servlet” csomagban található „BeadServlet” osztályhoz kerül át.

Ezután a szervletnek a doPost metódusa fog lefutni, amely az SshConnection osztály Bead metódusát fogja meghívni, és ennek paraméterként a feladat számát adja át. Ez a metódus az előzőleg már ismertetetett módon egy unix parancs elküldésével adja be a szükséges fájlokat. A Bead metódus egy sztringgel tér vissza, és ezt fogja a szervlet a kimenetre írni. A teszteléshez hasonlóan itt is szükség van arra, hogy visszalépjünk a felhasználó gyökérkönyvtárába, mivel a Bead metódus is meghívta a goToDir metódust.

bead.jsp

3.5.4. Eredmények lekérése

Az eredményeket az üdvözlő képernyőn lévő „Eredményeim” gombra kattintva kérhetjük le. Ekkor a default.jsp a servlet csomagban található ResultsServlet osztálynak adja át a vezérést. A szervlet doPost metódusa az SshConnection osztály Results metódudusát hívja meg, amely egy sztringgel fog visszatérni, és ezt fogja a ResultsServelt a képernyőre írni.



Eredmény lekérdezése és a ResultsServlet válasza

3.5.5. Kilépés

A képernyő bal oldalán lévő kilépés gombra kattintva a felhasználó kilép az oldalról és újra a bejelentkezési ablakkal találkozik. A vezérlést az exit.jsp kapja meg, amely meghívja az SshConnection osztály exit metódusát, majd az ssh és username attribútumok értékét „null”-ra állítja és ezután a login.jsp oldalt jeleníti meg.

4. Összegzés

Az általam fejlesztett projekt nem helyettesíti, csupán kiegészíti a jelenleg tökéletesen működő tesztelő rendszert. Ennek a rendszernek nehéz és körülményes kezelhetőségén kívántam javítani az által, hogy egy weblap formájában jelenítettem meg a tesztelő szolgáltatásait és üzeneteit.

A fejlesztés Java nyelven történt, ami mellett azért döntöttem, mert az Enterprise Edition megfelelő támogatást nyújt dinamikus tartalmú, egy időben több felhasználó igényeit kielégítő oldalak létrehozására, valamint több jól használható, előre megírt Java csomag is segítségemre volt a tesztelés, beadás és feltöltés funkciók megvalósításánál.

A jövőben praktikus lehet létrehozni a programnak egy olyan verzióját, amely a fájlfeltöltést, tesztelést és beadást is egy fájl-böngésző segítségével valósítja meg, így a felhasználó maga alakíthatná ki a számára tetsző könyvtárstruktúrát, a jelenlegi szigorú architektúrával ellentétben.

5. Irodalomjegyzék

– Imre Gábor – Szoftverfejlesztés Java EE platformon

– The Java EE 5 tutorial: http://java.sun.com/javaee/5/docs/tutorial/doc/

– SshSession Java API:
http://www.jscape.com/sshfactory/docs/javadoc/com/jscape/inet/ssh/SshSession.html

– Apache Ant Scp API:
http://cupi2.uniandes.edu.co/manualAnt/manual/api/org/apache/tools/ant/taskdefs/optional/ssh/Scp.html

– SshListener API:
http://www.jscape.com/sshfactory/docs/javadoc/com/jscape/inet/ssh/SshListener.html