Debreceni Egyetem
Informatikai Kar

EGY KONKRÉT BIOBANK ALKALMAZÁS MEGVALÓSÍTÁSA JAVA TECHNOLÓGIÁVAL

Témavezető:
Dr. habil. Hajdu András
egyetemi adjunktus

Készítette:
Nagy István Zoltán
Programtervező Informatikus Msc

Debrecen
2011

1. Tartalomjegyzék

1.Tartalomjegyzék....1
2.Bevezetés....2
3.Az alkalmazásról és a felhasznált technológiákról....5
3.1.A biobankokról....7
3.2.Alkalmazás architektúra....11
3.3.Java....15
3.4.Java Servlet és JavaServer Pages....18
3.4.1.Java Servlet....18
3.4.2.JavaServer Pages....20
3.5.Hibernate....23
3.6.PDF generálás....28
3.7.Appletek....31
3.8.XML technológiák....34
3.8.1.XML....34
3.8.2.DOM....37
3.8.3.XML technológiák az alkalmazásban....38
3.9.JavaScript....39
3.10.HTML és CSS....42
3.10.1.HTML....42
3.10.2.CSS....44
4.Összefoglalás....46
5.Irodalomjegyzék....48
6.Ábrajegyzék....53
7.Függelék....54
8.Köszönetnyilvánítás....55

2. Bevezetés

A dolgozatom témájául egy biobank alkalmazás megvalósítását, és az ahhoz használt technológiák bemutatását választottam. A döntés fő motivációja az volt, hogy valamilyen használható alkalmazást állítsak elő, amely az informatika eszközeit egy emberközeli területen hasznosítva segíti a terület szakembereit a mindennapi munkájuk elvégzésében, sőt a napi rutin mellett képes olyan többletfunkciók megvalósítására, amelyek a kutatás területén is alkalmazhatóak lehetnének. Ezért is esett a választásom az egészségügyi, illetve bioinformatikai területen alkalmazott szoftvertermékek ezen típusára. A dolgozat készítése során megvalósult biobank alkalmazás a genetikai eredetű betegségek diagnosztizálása során keletkező adatok tárolására és annak későbbi felhasználására ad egyfajta megoldást. Ez a terület azért is volt motiváló számomra, mert az elvégzett vizsgálatok alanyai gyermekek, és úgy érzem eléggé fontos lenne, ha a területen folyó kutatások eredményesebbek lehetnének, mert ha a jelenlegi betegeknek nem is feltétlenül lenne ettől jobb, de a jövő ilyen betegségben szenvedő gyermekeit tekintve talán komoly előrelépés történhetne. Sokat segített a terület megismerésében, illetve az ilyen rendszerek megvalósításához szükséges tapasztalat megszerzésében, hogy a mesterképzés megkezdésekor már Debrecen legjelentősebb bioinformatikai beállítottságú kutatás-fejlesztési vállalatának, az Astrid Research Kft.-nek erősíthettem a fejlesztői gárdáját. Az itt megszerzett tudást, amennyire csak a téma engedte, szerettem volna beépíteni a kifejlesztett alkalmazásba, hogy a funkcionalitás terén a piacon jelen levő professzionális szoftverek paramétereit sikerüljön megközelíteni, illetve ezzel párhuzamosan a dolgozatom értékét is növelni.

A dolgozat alapjául szolgáló alkalmazás létrehozásához használt eszközök, technológiák kiválasztása ekkor már szinte nem is volt kérdés, hiszen az eddigi tanulmányaim során leginkább a Java nyelvvel és a hozzá kapcsolódó technológiával kerültem kapcsolatba, így az alkalmazás legjelentősebb részéül szolgáló szerver oldali kódot a Java erre a célra használható megoldásainak egy részével a Java Servlet-ekkel, és JavaServer Pages (JSP) oldalakkal kezdtem el megvalósítani, mivel ezek tökéletesen elegendőnek bizonyultak a probléma megoldására, és ugyan valószínűleg egy webes keretrendszer, például Spring, Struts vagy Struts 2 használatával lerövidülhetett volna a fejlesztés ideje, de az alkalmazás programozásának kezdetekor az idő nem számított igazán fontosnak számomra. Egy

keretrendszer beépítése pedig nem csak előnyöket jelent, hanem kötöttségeket is, és nem biztos, hogy az eredményül kapott alkalmazás architektúra jobb lesz, mint a keretrendszer nélkül, például a keretrendszer nem megfelelő használata esetén. Emiatt úgy éreztem, hogy egy keretrendszer használata nem biztos, hogy elősegítené az alkalmazás létrehozását, illetve ezzel az egész dolgozat sikerességét. Jelenleg úgy érzem, hogy ez nem feltétlenül volt jó döntés, mert a jelenlegi alkalmazás még jobb lehetett volna mondjuk Struts 2 használatával, és az azóta szerzett tapasztalataim alapján ma már egészen biztosan egy ilyen keretrendszer használata mellett döntenék, mert úgy érzem, hogy jelenleg már nem lenne alapja a kezdeti aggodalmaimnak.

Az adatok tárolására természetesen valamilyen adatbázis-kezelő rendszer nyújtotta a legjobb és tulajdonképpen egyetlen alternatívát. Ebben a témakörben mindenképpen szerettem volna valamilyen lehetőleg open-source, vagy ingyenesen felhasználható terméket választani, így végül a MySQL használata mellett döntöttem, mivel úgy gondoltam, hogy mindenre képes, amire nekem szükségem van, így felesleges lett volna valami ennél komolyabb terméket beépíteni az alkalmazásba, illetve tudtam azt, hogy mivel Hibernate segítségével végzem az adatbázissal kapcsolatos műveleteket, ha nem használom ezeket a műveleteket az ajánlottól eltérően a programomban, akkor szükség esetén pár sor módosításával könnyedén lecserélhető lesz az adatbázis-kezelő rendszer a későbbiekben is.

Az alkalmazás futtatásához az Apache Tomcat servlet container-t választottam, mivel semmi olyan eszköz nem jelenik meg a programban, amely ennél komolyabb apparátust igényelt volna, illetve meggyőzően hatott az a tény is, hogy ez a Java Servlet és a JavaServer Pages specifikáció referencia implementációja, melynek a magját a Sun Microsystems szakemberei készítették, majd az Apache Software Foundation-nek adták át azt további gondozásra. Emellett pozitívan befolyásolta a választást az is, hogy open-source és tisztán Java HTTP webszerver implementációról van szó.

Mivel egy ilyen alkalmazás szinte elképzelhetetlen lenne az adatokból PDF dokumentumok létrehozása és letöltése nélkül, illetve a PDF közvetlen kinyomtatására szolgáló funkció nélkül, feltétlenül szükség volt valamilyen kliens oldali kódra is, hogy hozzáférhessek a nyomtatóhoz. A számos lehetőség közül a Java Applet használata tűnt a legcélszerűbbnek. A webalkalmazás és az applet kommunikációját XML technológiák segítségével előállított és feldolgozott üzenetekkel szerettem volna megvalósítani, mivel

korábbi dolgozatomban is XML-lel kapcsolatos alkalmazást hoztam létre, és megkedveltem ezeknek az eszközöknek a használatát. Persze sokat segített a döntés meghozásában az is, hogy az XML nagyon jól alkalmazható ilyen célokra, így szinte magától értetődő, hogy nem mást választottam.

Az alábbi lista a teljesség igénye nélkül szemlélteti a dolgozat és az alkalmazás létrehozásához felhasznált technológiákat, illetve szoftvertermékeket:

- Java
- HTML
- JavaScript
- JQuery
- CSS
- JavaServer Pages
- Java Servlet
- Hibernate
- MySQL
- XML
- XML Schema
- Ant
- Ivy
- JasperReports
- Apache Tomcat
- IntelliJ IDEA
- Fedora Linux
- LibreOffice

Ezek közül a legjelentősebbekkel a dolgozat további fejezeteiben bővebben is foglalkozom.

3. Az alkalmazásról és a felhasznált technológiákról

Ebben a szakaszban szeretném bővebben részletezni azt, hogy az alkalmazással kapcsolatban milyen megoldásokat használtam, illetve milyen döntéseket kellett meghoznom ahhoz, hogy az alkalmazás a jelenlegi formáját ölthesse. Emellett itt szeretném megragadni a lehetőséget, hogy röviden ismertessem az alkalmazás főbb funkcióit.

A rendszer egyik legfőbb feladata, hogy könnyen és gyorsan lehessen a beérkező betegek adatait rögzíteni. Erre persze csak a megfelelő jogosultságokkal rendelkező felhasználóknak van lehetősége, hogy a betegek személyes adatai ne kerülhessenek illetéktelen kezekbe. Amikor egy beteg vizsgálatra érkezik, az *új beteg* funkció használatával fel lehet venni a személyes adatait az adatbázisba, miközben a jogi korlátozások miatt alá kell íratni vele egy nyilatkozatot, amelyben kijelenti, hogy milyen céllal hajlandó a biobank rendelkezésére bocsátani az adatait. Ekkor több lehetősége van, természetesen akár az is, hogy nem hajlandó részt venni ebben, ami azt jelenti, hogy az adatait sem szabad felvinni a rendszerbe. A többi esetben pedig azt mondhatja meg, hogy kutatási, diagnosztikai, vagy mindkét céllal szeretné szerepeltetni adatait. A kutatók a személyes adatok nélkül, csak a releváns szakmai információkhoz férhetnek hozzá, a beteget tehát anonim módon kezelhetjük. A diagnosztikai esetben viszont a vizsgálatok dokumentálásához szükséges információikat is rögzítjük. Ebben a lépésben előfordulhat, hogy már létezik rekord a beteghez. Ez egyáltalán nem jelent problémát, mivel a beteg felvételére szolgáló űrlapon a *TAJ szám* vagy a *név* mezők kitöltésével a korábbi adatok felhasználásával automatikusan kitölthető az egész űrlap, természetesen csak teljes egyezés esetén. A beteg profiljának létrehozását követően lehetőségünk van a kért vizsgálatokhoz tartozó úgynevezett vizsgálatkérő lapok adatainak feltöltésére, amelyet szigorú ellenőrzés és intelligens kiegészítési megoldások segítenek. Egy vizsgálatkérő lap kitöltését követően azonnal ki is nyomtathatjuk azt a papír alapú dokumentáció segítése érdekében. A vizsgálatok elvégzését követően a labor munkatársai leleteket csatolhatnak a beteg profiljához, illetve a vizsgálatkérő lapokhoz. Ezt a lépést sablonok segítségével tehetik meg az illetékesek, hogy az egyes diagnózisokra jellemző általános információkat ne kelljen felvinniük minden alkalommal, csupán egyszer elmentik sablonként, majd következő alkalommal már csak a fent maradó, specifikus mezőket kell

kézzel kitölteniük, a többi mező értéke automatikusan átmásolódik a használt sablonból. Természetesen utólag lehet módosítani az automatikusan kitöltött mezőket is ha valamely korábbi sablon mondjuk csak egyetlen mezőjét tekintve nem megfelelő nekünk, akkor ne kelljen nélkülöznünk ezt a kényelmi funkciót. Fontos megemlíteni a sablonokkal kapcsolatban, hogy egy sablon mentésekor megadhatjuk azokat a feltételeket, amelyek teljesülése esetén a lelet hozzáadásakor egy vizsgálatkérő lapot, mint előzményt kiválasztva az adott lelet sablont meg kell jeleníteni, vagy el kell rejteni a felhasználó elől, ezzel is gyorsítva a munkáját. A lelet mentését követően itt is lehetőségünk van az azonnali nyomtatásra. A nyomtatással kapcsolatban természetesen teljesül az az elvárás, hogy a betegek adatait csupán egyszer kell megadni, és azok a beteg profiljából másolódnak át például a kinyomtatott vizsgálatkérő lap megfelelő rovatába. Ezek az információk a leletek és vizsgálatkérő lapok megjelenítésekor és szerkesztésekor is csak olvashatóak maradnak. Fontos funkció továbbá még a betegek visszakeresése is, amely történhet TAJ szám, név, vagy biobank azonosító alapján is, illetve a lista az alapvető eljárásoknak megfelelően több oldalra tördelhető, illetve bármely oszlop szerint rendezhető.

3.1. A biobankokról

„Biobanknak nevezzük azokat a mintagyűjteményeket, melyek mintái élőlények testéből származnak: ennek megfelelően léteznek emberi, állati, növényi, mikrobiális stb. mintagyűjtemények, biobankok. Más nézőpont, és így más megfogalmazás szerint a biobank genetikai mintát és az ehhez kapcsolódó genetikai és személyazonosító adatokat genetikai vizsgálat, illetve humángenetikai kutatás céljából tartalmazó mintagyűjtemény.”

(Mi a biobank?)

A biobank szoftverek olyan szoftverrendszerek, melyek tehát biológiai mintákhoz kapcsolódó adatok rögzítésére, visszakeresésére, illetve feldolgozására szolgálnak. A biobankok nem a bioinformatikai, hanem az orvosi informatikai rendszerek közé tartoznak, ahová az orvosi célú képfeldolgozási rendszerek is. A bioinformatikai alkalmazások csoportjába az NGS (Next Generation Sequencing), az oligo tervezés, a biostatisztika és az egyéni adatelemzés alkalmazásai tartoznak. A biobankok több csoportba sorolhatóak, mely csoportokat az alábbi lista ismerteti:

- Klinikai biobank
- Terápiás biobank
- Kutatási biobank
- Igazságügyi biobank
- Kevert típusú biobank

Ezen típusok jelentősen eltérnek, ám mindegyikre igaz az a fentebb már említett állítás, hogy biológiai minták „köré épülnek”. Az eltérés leginkább a felhasználás módjában fedezhető fel, mivel nem mindegy, hogy mondjuk egy-egy betegség kialakulásának okát, vagy a kezelés módját kívánjuk felfedezni a betegektől származó minták használatával, vagy akár mindkettőt is kitűzhetik a biobank építői célul. Fontos megjegyezni, hogy egy biobank megoldás létrehozása nagyon költséges vállalkozás, mivel a minták tárolása, rendszerezése, vizsgálata, a szoftver kifejlesztése, vagy megvásárlása, illetve az összegyűjtött adatok elemzése is tetemes költségekkel jár. A biobankokra is általánosan igaznak mondható, hogy a minták, illetve a bevont egyedek (mivel nem csak emberek lehetnek a minták tulajdonosai, bár a dolgozat szempontjából kétség kívül az emberi biobankok élveznek magasabb prioritást) számának növekedése segítheti a következtetések levonását. Ez abból adódik, hogy a

biobankok életciklusának utolsó fázisában az adatok elemzésére kerül a hangsúly, amelyet statisztikai és adatbányászati eszközökkel végezhetnek. A sok bevont egyed, illetve a tőlük származó adatok pontos, részletes elemzése komoly segítséget nyújthat a biobank adatait felhasználó személyeknek abban, hogy valamilyen következtetést lehessen levonni, valamilyen mintázatot lehessen felfedezni. Ilyen mintázat lehet mondjuk az, hogy akik több grillezett húst esznek, nagyobb valószínűséggel szenvedhetnek prosztatarákban.

(Grilled meat consumption and PhIP-DNA adducts in prostate carcinogenesis)

(Will Biobanking Change the World?)

Egy átlagosnak mondható biobank életciklusa során az alábbi tevékenységeket szokták végrehajtani:

- Biológiai minták mintavétele, összegyűjtése, feldolgozása, tárolása
- Klinikai adatok összegyűjtése - (Kórelőzmény, fizikális/műszeres vizsgálatok adatai, laboratóriumi adatok, képalkotó eljárások-képfeldolgozás)
- Környezeti-, szociális faktorok, életmód, táplálkozás, stb. felderítése
- Biokémiai vizsgálatok elvégzése
- Adatok elemzése adatbányászat, illetve biostatisztika segítségével

Egy biobankban nem csak betegek adataival találkozhatunk, hanem kutatási biobankok esetében általános, hogy úgy nevezett kontroll személyeket is be kell vonni, akik a vizsgált betegségeket tekintve nem érintettek, így segítik a normálistól való eltérések vizsgálatát, ahol ők, illetve pontosabban a tőlük származó minták képviselhetik a normális, egészségesnek mondható értékeket. A dolgozat témájául szolgáló biobank alkalmazás egy diagnosztikai és kutatási célú biobank részét képezi (illetve képezheti), amely a genetikai jellegű betegségekkel foglalkozik.

A biobankok közös jellemzője, hogy a biológiai minták illetve a hozzájuk tartozó adatok kezelése során körültekintően kell eljárni, a szigorú jogi és etikai szabályozás betartása mellett. A biobankokra vonatkozó szabályozást hazánkban a „2008. évi XXI. törvény a humángenetikai adatok védelméről, a humángenetikai vizsgálatok és kutatások, valamint a biobankok működésének szabályairól (2008. július 1-jén lépett hatályba)”, illetve a „26/2008.

EüM rendelet az egészségügyi szolgáltatások nyújtásához szükséges szakmai minimumfeltételekről” látják el. Emellett több nemzetközi ajánlás és iránymutatás is létezik, amelyeket érdemes figyelembe venni biobankok létrehozásakor:

- OECD Best Practice Guidelines for Biological Resource Centers (2007)
- NCI Best Practices for Biospecimen Resources (2007)
- ISBER. Best Practices for Repositories

Ezek a dokumentumok a jogi és etikai szabályozások mellett a biobankok menedzsmentjére, és a technikai részletekre vonatkoznak. A jogi kezelésről el kell mondani, hogy a biobank projektekbe bevont személyek esetében a biológiai mintáik felhasználásáról egy nyilatkozatot kell kitölteni, amelyben a minták tulajdonosai beleegyeznek a mintáik különböző célokra történő felhasználásába a nyilatkozatban foglalt feltételek mellett. Ez azért fontos, mert egy-egy biobankban nagyon érzékeny információk jelenhetnek meg a betegekről, amelyekkel egyikünk sem szeretné, ha visszaélhetnének.

(Törvényi háttér)

A biobankokat tekintve a biobank létrehozójának el kell döntenie, hogy egy általános vagy egy személyre szabott, egyedi megoldást szeretne választani. Ez a döntés tulajdonképpen ugyanaz, mint amit a vállalatok által használt bármely szoftver bevezetése előtt a vezetőknek meg kell hoznia. Az általános megoldások mellett szól az, hogy általában alacsonyabb áron elérhető „dobozos” szoftverekről beszélhetünk, míg az egyedi megoldásokat pontosan amiatt, hogy rajtunk kívül senki nem használja, most kell kifejleszteni részben vagy egészében, így viszonylag magas árakról beszélhetünk. Meg kell azonban jegyeznünk, hogy az általános eszközök használatánál a vállalat folyamatait kell a rendszerhez illeszteni, míg az egyedi esetben pontosan fordítva, így az utóbbinál nincs szükség kompromisszumokra, illetve egy általános adatmodellt alkalmazó termék körülményes bekonfigurálására, testreszabására. Emellett az egyedi termékben található jogosultságok is pontosan a mi igényeinknek megfelelőek, míg a „dobozos” szoftverek jogosultság kezelése nem minden esetben alakítható teljesen megfelelően. Az egyedileg megvalósított megoldásokról általában elmondható, hogy az ügyfél igényeit valószínűleg jobban kielégítik, mint az általános előre legyártott szoftverek, amely azért is fontos, mert egy

biobank létrehozásakor a biobank szoftver kiválasztásával vagy létrehozásával még csak a biobank egy részét tekinthetjük elkészültnek, mivel a minták tárolásának módja, a jogi háttér kidolgozása, a betegek illetve kontroll személyek kiválasztása egyenként is hasonló nehézségű feladatnak mutatkozik. Ez a dolgozat ezekkel a folyamatokkal a továbbiakban nem foglalkozik, mivel ezek nem szerepelnek a kitűzött célok között, csupán az alkalmazás előállításával kapcsolatos dolgokról olvashatunk a későbbi fejezetekben.

3.2. Alkalmazás architektúra

Az alkalmazás architektúráját tekintve több szempontból kell megvizsgálnunk a helyzetet. Mivel webalkalmazásról beszélünk, nem is kérdéses az, hogy kliens-szerver architektúráról van szó. A böngészőt tekintjük kliensnek, a webkiszolgáló, a webkonténer és az adatbázis-kezelő számára otthont adó gépet pedig szervernek. Azért a böngésző a kliens, mert a kliens-szerver architektúra esetén mindig a kliens kezdeményezi a kapcsolat létrejöttét, illetve a kliens az, amely nem osztja meg az erőforrásait, míg a szerver vár a kliensek kéréseire, hogy megválaszolhassa azokat, miközben az erőforrásai megosztottak. Ezen belül vékony kliensről beszélhetünk, amikor a böngészőt tekintjük, mivel a megjelenítésen kívül semmi más feladata nincs, az összes számítást, adattranszformációt a szerver végzi el. A vékony klienseknek köszönhetően az alkalmazás megjelenítéséhez nem szükséges olyan komoly hardverrel rendelkezzen a felhasználók számítógépe, viszont cserébe sokkal nagyobb terhelés tapasztalható a szerveren. Ez azonban nem jelent komoly problémát, mivel a kiszolgáló terhelése csökkenthető load-balancing technikák alkalmazásával, vagy akár azzal is, ha az adatbázis-kezelőt nem a webkiszolgálót futtató szerverre telepítjük. A kliens-szerver architektúra tehát lehetővé teszi ebben az esetben, hogy az alkalmazás futtatása a számítógépek többségén zökkenőmentesen kivitelezhető legyen, míg a kiszolgálók számának növelése, illetve a web- és adatbázis kiszolgálók külön hardveren futtatásával magas rendelkezésre állást, továbbá kielégítő mértékű skálázhatóságot eredményez. Továbbá a szerverek elosztottsága miatt könnyebbé válhat a karbantartás, esetleg a gépek cseréje, míg a kliensek kiszolgálása zavartalan maradhat. A szerverek általában kiválóan képesek arra, hogy a hozzáféréseket kontrollálják, és csak azon kliensek számára tegyék elérhetővé a kért erőforrásokat, amelyek tényleg jogosultak a hozzáférésre. A centralizált adattárolásnak köszönhetően pedig egyszerűvé válik az adatok frissítésének adminisztrálása például a P2P megoldásokhoz képest. A kliens-szerver architektúrának azonban éppen az a hátránya is, amiben az előnye rejlik, hogy a hálózatra épít, ugyanis a hálózat révén komoly veszélynek vannak kitéve az alkalmazás futtatásához használt kiszolgálók. Ez annak köszönhető, hogy ahogyan a kliensek bárhonnan elérhetik az internet segítségével az alkalmazást, úgy bárhonnan érkezhetnek támadások is, például DoS (Denial of Service) típusúak. Azt kell azonban mondanom, hogy ennek a fokozott sebezhetőségnek ellenére is megéri ezt az

architektúrát választani, mivel a tűzfalak és egyéb szoftveres vagy hardveres megoldások, amelyek a számítógépek védelmére hivatottak hatásosan csökkenthetik a veszély mértékét, illetve megfelelő körültekintéssel elérhető az is, hogy a kockázat elhanyagolható legyen. Szerencsére jelenleg több bevált megoldás is van a piacon a kliens-szerver architektúra megvalósítására, így ha valaki szeretne ilyen architektúrájú alkalmazást készíteni, könnyedén találhat követendő példákat.

(Client/Server Architectures)

(The Client Server Architecture)

A webalkalmazások készítői számára általában nem ismeretlenek az MVC (Model-View-Controller), illetve a Front-Controller minták, amelyek nagyon népszerű minták manapság. A Front-Controller a kérések centralizációját vallja, azaz egyetlen belépési ponton keresztül fogadja az érkező kéréseket, és további feldolgozás céljából delegálja azokat. Ez azzal jár, hogy a kérések során a minden kérést érintő feladatok, mint például az autentikáció, megoldása nagyban leegyszerűsödik.

A Front-Controller minta előnyei:

- Centralizált vezérlés: az egész alkalmazásra kiterjedő feladatok ellátása egyszerű
- Szálbiztosság: nem szükséges szálbiztosnak lennie a Front-Controller által hívott controller objektumoknak, mivel mindig új példányt hozunk létre. A modell elemeinek azonban továbbra is szálbiztosnak kell lennie.
- Konfigurálhatóság: csak a Front-Controllert szükséges a webszervernél konfigurálni, a feladatok szétosztása onnantól megoldott.

Hátrányként jelentkezik azonban:

- A teljesítmény megfontolás tárgyát kell képezze, mivel a Front-Controller nagyon leterhelt annak köszönhetően, hogy minden kérés rajta keresztül érkezik.
- A Front-Controller komplexitása nagyon nagy lehet ahhoz, hogy a feladatát el tudja látni.

(Martin Fowler: Front Controller)

(MSDN: Front Controller)

Az MVC architekturális minta pedig arra irányul, hogy szétválassza az adat, a vezérlés és a megjelenítés elemeit. Az MVC már a 80-as évek elején ismert volt, és ma is igen kedvelt minta a keretrendszerek tervezői között. MVC-t használ többek között a Struts, Spring, a

Tapestry, a Zend és a JSF is. Az MVC tipikusan az alábbiaknak megfelelően működik:

- A felhasználó interakcióba kerül a megjelenítési réteggel.
- A vezérélés a modell számára érthető akcióvá alakítja az interakciót.
- A vezérélés értesíti az akcióról a modellt ami általában a modell állapotának megváltozását eredményezi.
- A megjelenítés frissül automatikusan, esetleg a vezérélés jelzi számára, hogy frissítse magát. A frissítés során a megjelenítés a modell elemeit használja.
- A megjelenítési réteg vár további felhasználói interakciókra, hogy a folyamat újrakezdődhessen.

A vizsgált biobank alkalmazás megvalósításakor jó ötletnek tűnt az MVC és a Front-Controller minták egyfajta ötvözése, miszerint az adat a megjelenítés és a vezérlés jól elkülöníthető az alkalmazásban, és a vezérlés megvalósításához egy központosított megoldást alkalmaztam, amely tulajdonképpen Front-Controllerként osztja szét a kéréseket a vezérlés többi elemének. Az MVC minta elemeit a következőképpen választottam meg:

- Adat (Model): Entity Bean-ek és elemi műveletek implementálása, amelyek az adatokon értelmezhetőek.
- Megjelenítés (View): JSP oldalak, amelyek a felhasználói interakciókat fogadják, illetve az adatok megjelenítését végzik.
- Vezérlés (Controller): Java Servletek, amelyek a kéréseket leképezik az adatréteg számára értelmezhető formára, a magasabb szintű üzleti logika metódusainak hívásával.

(IBM: Model-View-Controller (MVC))

(Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller)

Az általam alkalmazott megoldás a Model 2 architektúrát valósítja meg, amelyet például az Apache Struts is használ. A Model 2 tekintetében meg kell jegyeznünk, hogy tulajdonképpen önmagában nem is MVC, mivel csak a megjelenítést és a vezérlést tekinti, az adat réteg megvalósítását a fejlesztőkre bízza. A Model 2 a JSP legelső verziójának megjelenésével egyidősnek tekinthető, mivel ez a minta a Sun Microsystems által 1998-ban kiadott két alkalmazási módszer egyike, amelyek arra szolgáltak, hogy a JSP hasznosításának módjait szemléltessék.

(Apache Struts)

(JavaServer Pages Access Model)

(Understanding JavaServer Pages Model 2 architecture)

Az alkalmazásomban a kérések beérkezésekor a megjelenítés fogadja a felhasználó interakcióit, aztán a vezérlési rétegnek továbbítva azokat, megtörténik a felhasználók hitelesítése, majd ha ez végbement, megkezdődhet a kérés feldolgozása. A vezérlés egy, a magasabb szintű üzleti logikát megvalósító, réteget használva végzi el a dolgát. Ez a réteg végzi az alacsonyabb szintű műveletek hívását, amelyek már tulajdonképpen elemi adatbázis műveletekig vezetnek, mint mondjuk egy entitás adatbázisból történő betöltése, törlése, frissítése, vagy új entitás adatbázisba történő beszúrása. Fontos azonban megjegyezni, hogy az egyes entitások közötti kapcsolatok kialakítását, az előbb felsorolt műveletek elvégzését, és a tranzakció-kezelést a magasabb szintű logikáért felelős réteg végzi. Az adatok módosításáról a vezérlés értesíti a megjelenítést, hogy az elvégezhesse az adatok felhasználásával a prezentációt.

A kliens-szerver megvalósításból fakadóan kijelenthetjük azt is, hogy több rétegű architektúráról van szó a megoldás egészét tekintve. A kliens lenne ebben a megközelítésben az első réteg, a webszerver és a dinamikus tartalomért felelős webalkalmazás a második és az adatbázis-kezelő a harmadik réteg.

(MSDN: Layered Application Guidelines)

3.3. Java

A Java egy erősen típusos, objektum-orientált programozási nyelv. Szintaxisa leginkább C és C++ nyelvekére hasonlít. A nyelv fejlesztését James Gosling, Mike Sheridan, és Patrick Naughton kezdte el, eredetileg Oak néven. Állítólag a névválasztás egy a James Gosling irodája előtt álló fának volt köszönhető, ám kiderült, hogy ez a név már foglalt. 1995-től már a ma is használt nevén illették a nyelvet. A Java nyelvet a Sun Microsystems fejlesztette, ám mára már az Oracle folytatja ezt a munkát a Sun Microsystems pár évvel ezelőtti felvásárlásának köszönhetően. A Java jövőjét tekintve éppen emiatt többen aggodalmaskodnak, mivel azt mondják, hogy nem tesz jó a Java-nak az, hogy az Oracle a jövőben szeretné a Java virtuális gépeket több változatban biztosítani, azaz lenne egy gyengébb minőségű ingyenes és egy jobb minőségű fizetős változat.

A Java egyik legjelentősebb jellemzője az újrafelhasználhatóság, amely a Java programok írásakor kulcs szerepet játszik. Ezt számos eszköz segíti, gondoljunk csak az öröklődésre, amelynek köszönhetően a szuperosztályban (más néven ősosztályban) megírt metódusok és adattagok (természetesen csak ha megfelelő láthatósággal rendelkeznek) elérhetőek az alosztályban (leszármazott) is. A másik kézenfekvő újrafelhasználási lehetőség, ha mondjuk előzőleg megírt osztályokat használunk egy Jar library-ból, amelyet több alkalmazásban is használhatunk. Úgy gondolom, hogy ez nagyban segíti a Java programozók mindennapi munkáját, mivel nincs szükség arra, hogy minden alapvető dolgot ők implementáljanak, inkább jól bevált, külön erre kifejlesztett eszközöket használhatnak egyszerűen egy-egy .jar állomány segítségével.

Fontos megemlíteni, hogy a Java még talán az újrafelhasználhatóságnál is jelentősebb, de mindenképpen ismertebb tulajdonsága a platformfüggetlenség. Ez már a kezdeti elképzelésekben is megjelent, amikor a nyelv fejlesztéséhez hozzáláttak a fentebb említett úriemberek. Feltehetően ez vezetett a Java mai napon tapasztalható széleskörű elterjedéséhez, népszerűségéhez. A platformfüggetlenség megvalósítása a korábban már érintőlegesen említett Java virtuális gép (JVM) segítségével történik. A megoldás lényege az, hogy a Java fordító (javac) a forráskódot nem a számítógép által közvetlenül értelmezhető gépi kódra fordítja le, hanem a Java bájtkódjára, amelyet aztán a minden platformon elérhető JVM adott platformon található implementációja tud értelmezni, és futtatni. Tehát a platformok közötti

különbségek elfedésére használják a JVM-et, amely így lehetővé teszi a Java forrásokból fordított bájtkód hordozhatóságát. Az egyetlen komolynak mondható probléma ezzel csupán az, hogy a natív kódokkal szemben valamennyi teljesítmény-csökkenés tapasztalható, például a C programok általában gyorsabban végzik el ugyanazt a feladatot a Java programhoz hasonlítva, ám meg kell jegyezni azt is, hogy a JVM implementációk között is vannak teljesítménybeli különbségek. Itt arra kell gondolni, hogy nem ugyanolyan gyors a Windows és a Linux operációs rendszerekre készített JVM, illetve érdemes megjegyezni az OpenJDK és az Oracle által fejlesztett társa közötti különbséget is. Szerencsére a teljesítménykülönbség az esetek egy részében elhanyagolható a Java és a C programok között, mivel a hardverek olyan léptékben fejlődtek az elmúlt évek során, hogy az eltérés ezredmásodpercekben mérhető lenne az esetek egy részében, így azt a felhasználók nem is veszik észre. Eközben viszont a Java kód megírása kevesebb emberi energiával elvégezhető, így a vállalatoknak kevesebb pénzbe kerül ugyanazon feladat elvégzése, ha a Java-t választják.

A számítógépek esetében egyes felhasználók számára akadály lehet az is , hogy a Java programok futtatásához a JVM-et külön telepíteni kell, mivel általában nincs előre telepítve ilyen virtuális gép a rendszerekre. Az utóbbi időben szerencsére ez a feladat már alig több egy kattintásnál. A Java elterjedését tekintve némi előrelépés figyelhető meg, mivel például a SunSpot már képes natívan értelmezni a Java bájtkódot, és biztosra vehetjük, hogy nem ez lesz az utolsó ilyen készülék. Ezzel szemben némi korlátozást jelent, hogy nem minden JVM képes az összes Java eszköz használatára, például a Java Mobile Edition csak az eszközkészlet töredékére épít. A Java terjedésének egyik legnagyobb előrelépésében azonban a Sun Microsystems fentebb már említett felvásárlása is közrejátszott, mivel az Oracle adatbázisaiban is egyre nagyobb szerepet kap a Java annak köszönhetően, hogy mára már az Oracle adatbázis-kezelőjébe beépített JVM segítségével a tárolt eljárásokat Java nyelven is megírhatjuk.

A Java támogatottságát tekintve is kitűnő, ami természetesen az elterjedtségéből is adódik. Az összes elemi adatszerkezetre létezik már Java implementáció, és ami a legjobb, hogy a Java beépített kollekcióinak használatával is meg lehet valósítani az egészen komplex feladatokat is. A bemenet és kimenet kezelése is egyszerűen megvalósítható a java.io csomag tartalmával. Hasonlóan egyszerűen meg tudjuk valósítani bonyolult XML dokumentumok DOM vagy SAX használatával történő feldolgozását is. A sort még folytathatnám, de azt

hiszem elegendő ennyi ahhoz, hogy belássuk, a Java használatával ez esetek legnagyobb részében nem kell magunknak alapvető eszközöket implementálnunk, elegendő csupán a meglevő eszközrendszer használatával vagy azok kiterjesztésével megoldani a feladatunkat, így a valódi problémára összpontosíthatunk a probléma megoldásához szükséges eszközök megteremtése helyett. Persze vannak olyan feladatok, amelyeket nem könnyű Java használatával megoldani, mint például a logikai programozást igénylőek vagy például operációs rendszert nem véletlenül nem szoktak Java-ban írni, de ez utóbbi például nem is volt célja a nyelv fejlesztőinek. A Java támogatottságát illetően nagyon nagy szerepe van a Java mögött álló közösségnek is, mint például az Apache Software Foundation, vagy az egyéni Java fejlesztők ezrei, akik készségesen segítenek egymásnak számos külön erre a célra létesített webportálon, fórumon, blogon. Itt meg kell jegyezni azt is, hogy általában a Java nyelven fejlesztett keretrendszerek, eszközkészletek dokumentáltsága is jó, illetve az esetek nagyon nagy százalékában kitűnő oktatóanyagok és információk érhetőek el ingyenesen, ezzel is segítve a fiatal, tapasztalatlan, de Java iránt érdeklődő fejlesztők tanulását.

Az eddig leírtak együttesen járultak hozzá ahhoz, hogy a Java nyelv használatával lássak neki, illetve valósítsam meg a dolgozat témájául szolgáló biobank alkalmazást.

3.4. *Java Servlet és JavaServer Pages*

3.4.1. Java Servlet

A Java Servlet, vagy röviden Servlet egy Java osztály a Java EE-ben, amelyet arra használnak, hogy HTTP kérésekre válaszoljon. Gyakran használják a HTTP Servlet jelentéssel a Servlet szót a Java fejlesztők. A Servleteket dinamikus tartalmak létrehozására használják, illetve képesek meg is jeleníteni ezeket az adatokat, mivel képesek például HTML vagy XML formátumban válaszolni a beérkező kérésekre. A Servletek nem tárolnak állapotot, a felhasználóktól érkező HTTP kérések összefűzéséhez, illetve a böngészés aktuális állapotának kezeléséhez a HTTP Session (munkamenet) technológia használatára van lehetőségünk. Ennek segítségével kezelhetjük a felhasználó adatait, illetve bejelentkezett státuszát. Egy ilyen session csak korlátozott időre áll rendelkezésünkre, mivel egy adott tétlenségi idő után a session invaliddá, használhatatlanná válik. A session azonosításához, illetve eléréséhez egy SessionID használata szükséges, amelyet HTTP Cookie vagy URL rewrite technológiák valamelyikével tudunk elkérni a klienstől. A Servlet osztály példányainak működése egyszerűen egy beérkező állapotmentes HTTP kérés megválaszolására irányul, azaz tipikus eset, hogy a beérkezett kérésre összeállítunk egy HTML dokumentumot, amelyet válaszként visszaadhatunk, vagy beállítunk bizonyos attribútumokat amelyeket egy JSP oldal fog felhasználni, amelyre a Servlet továbbítja a kérésünket. A Session segítségével tehát a HTTP protokoll állapotmentessége ellenére képesek vagyunk az aktuális felhasználók számára a korábbi interakciók figyelembevételével válaszolni az állapotmentes kérésekre, aminek köszönhetően tulajdonképpen böngészési állapotokat kezelünk. Ez egy mai webalkalmazás szempontjából teljességgel elengedhetetlen.

A Servletek életciklusa a példányosításunktól a destroy() metódusuk hívásáig tart, amely befejezteti a Servlet működését. A kérések kiszolgálása külön szálakon történik az init() metódus hívását követően, a service() metódus segítségével. A service() metódus megvizsgálja a beérkező kérést és a vizsgálat eredményétől függően továbbirányítja azokat HttpServlet esetében például a doPost() vagy doGet() metódusok valamelyikére. Ez a két metódus a GET illetve POST kérések feldolgozására szolgálnak. A GET kérések az URL használatával adnak át paramétereket a Servlet számára, míg a POST kérések a HTTP request

törzsében teszik meg ugyanezt. A Servletek használata során, ha a kérésre a Servlet állítja össze a válaszdokumentum HTML vagy XML kódját, akkor a Servlet doGet() vagy doPost() metódusát használva a response objektum getWriter() metódusát meghívva kapjuk eredményül azt a PrintWriter példányt, amely segítségével a válaszdokumentum teljes tartalmát könnyedén kiírhatjuk, míg ha bináris adatokat kell szolgáltatnunk válaszul, mondjuk egy kép letöltésének megoldásához írunk Servletet, amely a kliens számára a kép tartalmát rendelkezésre bocsátja, akkor a response objektum getOutputStream() metódusával célszerű inkább elkérni a Streamet, amelyre ki kell írnunk a kép tartalmát.

(Servlet Essentials)

(An Introduction to Java Servlets)

3.4.2. JavaServer Pages

A JavaServer Pages (JSP) és a Java Servlet mögött álló koncepciók teljesen eltérőek. Igaz az ugyan, hogy az alkalmazási terület, és a végcél, hogy HTTP kérésekre tudjunk válaszokat adni közös, azonban míg a Servletek használatakor Java kódba ágyazunk HTML vagy XML elemeket, amelyeket a fentebb már említett PrintWriterrel írunk ki a kliens számára, addig a JavaServer Pages ezt pont fordítva közelíti meg, és HTML elemek között ágyazhatunk be Java kódrészleteket. A JSP tulajdonképpen egyfajta válasz a nem Java alapú szerver oldali megoldásokra, mint például a PHP vagy az ASP, amelyet a Sun Microsystems 1999-ben adott. Ekkor ismerték fel azt, hogy a Java nem ad elegendő támogatást a webalkalmazások fejlesztéséhez.

A JSP oldalak két részből állnak: markup, illetve scriptlet elemekből. A Markup többnyire szabványos HTML vagy XML, míg a scriptlet elemek Java kódrészletek olyan blokkjai, amelyeket markup részletek szakíthatnak meg. A JSP fájl Java kódrészletei lefutnak a kérések alkalmával, és a hatásuknak köszönhetően áll elő a válaszdokumentum, amelyet a webszerver a kliens számára közvetít. A JSP fájlokat Java bájtkódra kell fordítani ahhoz, hogy használhassuk őket, de szerencsére ezt a fordítást csak egyszer kell elvégeznünk, illetve amikor változik a JSP tartalma.

A JSP szintaktika további XML szerű tageket (action) bocsát rendelkezésre a HTML tagjein felül, hogy beépített funkcionalitást érhessünk el, illetve lehetősége van a programozónak arra is, hogy úgy nevezett JavaServer Pages Standard Tag Library-t (röviden JSTL) hozzon létre, amelyek további funkcionalitást biztosítanak. Az Apache Software Foundation számos olyan projektet gondoz, amelyek ilyen hozzáadott tag library-ket tesznek elérhetővé. Ilyen például a Struts keretrendszer által használt struts tag library is. Az Oracle vagy régebben még Sun Microsystems által biztosított tag library-k közül kettőt említenék meg, amelyeket a dolgozat alapjául szolgáló alkalmazás létrehozásakor is felhasználtam. Az első a JSTL core néven hivatkozott, amely többek között egyszerű vezérlési szerkezeteket biztosít a programozók számára, mint például az if, forEach, choose tagek, de találhatunk benne például redirect vagy url taget is, melyet a navigáció során hasznosíthatunk például. Ezek nélkül az alapvető eszközök nélkül eléggé nehézkessé válna a dinamikus tartalmak JSP használatával történő megjelenítése. A második tag library a JSTL fmt, amelyet még meg

szeretnék említeni. Ez a tag library a manapság alapvető szerepet betöltő I18N (Internationalization) azaz a többnyelvűség eléréséhez szükséges eszközöket biztosítja. Egyszerűen megoldható a segítségével a ResourceBundle-ök használata, amelynek köszönhetően az alkalmazás összes feliratát egy központi helyen (egy properties kiterjesztésű fájl) tárolva az egész alkalmazás egy új nyelvre lefordítása is jelentősen leegyszerűsíthető.

A JSP specifikáció fejlődése 2006-ra elérte a 2.1-es verziót, amely a Java EE 5 részeként érhető el, míg a JSP 2.2 csak 2009 decemberére datálható. A JSP 2.0 újdonságainak köszönhetően egyszerűsödött a beágyazott beanekkel történő navigáció, a paraméterek gyorsabban és könnyebben kiírhatóak, illetve az igazán nagy újítás az Expression Language (röviden EL) megjelenése is sokat egyszerűsített a programozók dolgán. Az Expression Language egy olyan eszköz, amely a JSP-t igazán egyszerű szkript nyelvként használhatóvá teszi. Az EL és a JavaScript között több hasonlóság is megfigyelhető:

- implicit típuskonverziót használ a legtöbb esetben
- nem tesz különbséget a ' és ” jelek között
- az object.property és object[”property”] írásmódok jelentése között nincs különbség

Az EL használatával anélkül is képes lehet valaki egy JSP program megszerkesztésére, hogy Java programokat tudna írni, mivel csak bizonyos elemi dolgokat kell figyelembe vennie és ha megfelelően átadja az oldalra átirányító Servlet az adatokat, amelyeket a JSP oldallal szeretnénk megjeleníteni, akkor az eredmény egy tökéletesen működő oldal lehet. Az EL bármely statikus szövegben vagy olyan tagek attribútumaiban használható, amelyek képesek kifejezést elfogadni. Az EL segítségével könnyedén elérhetjük a request paraméterek értékeit vagy a használt beanek adattagjait, könnyen ellenőrizhetjük, hogy egy kifejezés üres-e, illetve logikai és hasonlító operátorokat is használhatunk. Ezeken túl egyszerűen érhetjük el a pageScope, requestScope, sessionScope és applicationScope változóit is.

A JSP a Servlet technológiával együtt alkalmazva kitűnő, egyszerűen használható és megbízható alternatívát nyújt a többi dinamikus tartalmak megjelenítésére használt technológiával szemben. A JSP-t a Servlet technológia kiegészítéseként tekinthetjük, ami lehetővé teszi platformfüggetlen web alkalmazások fejlesztését mindamellett, hogy a megjelenítési réteget elválasztjuk az üzleti logikától. Emellett a JSP kiválóan használható több keretrendszerrel együttesen is, mint például a Struts, Struts 2 vagy Spring. Ezek a keretrendszerek ki is kényszerítik többnyire azt, hogy a JSP oldalakat csupán a megjelenítésre

használjuk, mivel ha ettől eltérően szeretnénk cselekedni, akkor nemhogy megkönnyítenék a dolgunkat, hanem meg is nehezítenék. Az általam írt alkalmazásban mint azt korábban már említettem, ezen keretrendszerek használata nem, de a JSP oldalak és Servlet-ek közös használata megfigyelhető, illetve ügyeltem arra, hogy a JSP-t csak és kizárólag a prezentációs rétegben alkalmazzam.

(Expression Language)

3.5. Hibernate

A Hibernate egy object-relational mapping (ORM) eszközgyűjtemény, amely arra hivatott, hogy elősegítse Java objektumok perzisztenciájának relációs adatbázisban való megvalósítását. A perzisztencia tulajdonképpen azt jelentené ebben az esetben, hogy szeretnénk a Java objektumaink állapotát a JVM életciklusán túl is megőrizni. A Hibernate library mögött álló motiváció az volt, hogy az EJB2 perzisztencia megoldásának egy alternatíváját kínálja, amely jobb hatásfokkal, illetve könnyebben használható az EJB2-nél. A Hibernate projektet 2001-ben Gavin King alapította. A projekt jelenleg a Red Hat illetve a JBoss gondozásában érhető el és a 3-as verzió tekinthető elterjedtnek illetve stabilnak, míg a 4-es előzetes verziója is elérhető már a fejlesztők számára. A Hibernate már 2010 óta a Java Persistence API 2.0 specifikáció hivatalosan elismert implementációja.

Az ORM témának számos nehézsége van, amelyek abból adódnak, hogy az OO modell és a relációs modell között igen nagy különbségek vannak már alapjaikat tekintve is. Ezen különbségek, illetve az OO és RDBMS közötti leképezés nehézségei közül a következő szakaszban a teljesség igénye nélkül szeretnék ismertetni párat.

A nehézségek között meg kell említeni az adattípusok közötti eltéréseket, mint például a karakterláncok tárolásának módja. Az OO nyelvekben általában nem korlátozzuk a karakterláncok hosszát, csak a rendelkezésre álló memória szab határt neki, míg a relációs adatbázis-kezelők használatakor meg kell adnunk egy maximális hosszot, amely korlátozza a tárolható karakterek számát. Fontos továbbá még az is, hogy a karakterláncok kódolását is figyelembe kell venni a leképezés során, illetve egyes esetekben a relációs adatbázisban a végző whitespace karaktereket figyelmen kívül hagyjuk, míg mondjuk a Java esetében nem.

Probléma még az is, hogy az RDBMS-ben a rekordok azonosítására vagy azonosságuk vizsgálatára csak az elsődleges kulcs segítségével van lehetőségünk, míg az OO esetben, például Javaban különbséget teszünk az a == b és az a.equals(b) között (a és b objektumok). Ez mondjuk komoly gondot jelenthet ha egy halmazba szeretnénk beilleszteni két elemet, mivel RDBMS-ben két rekord különböző az elsődleges kulcsuk miatt, viszont Javaban már nem tudjuk ugyanezt a halmazt létrehozni, ha van legalább két olyan objektum, amelyek között fennáll az, hogy az a.equals(b) igaz.

Fontos megjegyeznünk, hogy az altípusok és öröklődés kérdésében is eltérés van, mivel

az OO nyelvek kezelik ezt a problémát, míg az RDBMS nem ad jól definiált, standardizált megoldást erre, bár igaz az, hogy pár RDBMS rendszer támogatja altípusok használatát. Ez a probléma az egyik legjelentősebb szerintem, mivel nem lehet igazán elegánsan kezelni a leszármaztatott objektumok állapotának tárolását. Itt meg kell említenem a szemcsézettség kérdését is, mivel előfordul, hogy egy objektumhierarchia letárolásához használt táblák száma többnyire kevesebb, mint az osztályok száma.

Az utolsó jelentős eltérés, amelyet mindenképpen szerettem volna megemlíteni, a tranzakciókezelés témakörébe tartozik, mégpedig arról van szó, hogy az OO nyelvekben egységnyinek tekintett műveletek sokkal kisebbek, mint az RDBMS tranzakciói. Az OO esetben tipikusan primitív típusú értékek beállításaként jelentkezik a legtöbb művelet, míg a relációs adatbázisokban a tranzakciók jóval több rekord értékeit módosíthatják. Az OO nyelvek általában nem rendelkeznek analógiával az izoláció vagy tartósság fogalmára, illetve az atomitás és a konzisztencia fogalmak is csak az előbb említett primitívekkel kapcsolatos műveletekre vonatkoznak.

(Hibernate: What is Object/Relational Mapping?)
(The Object-Relational Impedance Mismatch)
(The Vietnam of Computer Science)

A Hibernate használatával az OO és a relációs modell közötti oda-vissza leképezés könnyedén megvalósítható, csupán megfelelően fel kell készítenünk a library által nyújtott eszközök segítségével az objektumainkat. Erre két igen egyszerű módon van lehetőségünk: XML vagy Java annotációs konfigurációs megoldás használatával. Az XML használatával lehetőségünk van arra is, hogy megfelelően annotált Java Bean-ek vázát generáljuk le, amelyet a saját tartalmunkkal tölthetünk fel. Az annotációs megoldás számomra kellemesebbnek tűnik, mivel nem nyúl ki a Java eszközrendszeréből, és tiszta, világos, könnyen érthető módon tudjuk megadni azt, hogy milyen kapcsolatai vannak például az objektumoknak. Az annotációs megoldás hátránya viszont, hogy sajnos jelen pillanatban úgy tudom, hogy nem képes a Hibernate annotációs rendszere mindent megoldani, amit az XML konfigurációval meg lehet. Ez szerencsére a dolgozat során létrejött alkalmazáson túl mutató probléma, mivel nem volt szükségem olyan bonyolult kapcsolatrendszer használatára, amelyhez mindenképpen XML konfiguráció lett volna szükséges. Az annotációs konfiguráció másik hátránya, hogy csak a Java 5-ös vagy annál frissebb verzióival használható, bár

jelenleg, a 7-es Java megjelenésének küszöbén úgy érzem, hogy nem jelent ez akkora problémát.

A Hibernate konfigurációjakor elég sok mindent automatikusan is meg tud oldani a library, de ha szeretnénk azt kikényszeríteni, hogy a relációs séma a nekünk tetsző alakot öltse, szinte mindent meg tudunk kötni. Megadhatjuk az osztály annotálásakor a tábla nevét, amelybe menteni szeretnénk az osztály példányainak állapotát, minden adattagról megmondhatjuk, hogy perzisztens vagy tranziens-e, illetve milyen adattípust szeretnénk használni, mi legyen az oszlop neve, milyen néven hozza létre a Hibernate egy kapcsolat kapcsoló tábláját, illetve hogyan generáljuk az elsődleges kulcsot. A perzisztens illetve tranziens adattagok tekintetében egy perzisztensnek jelölt osztály valamennyi adattagja alapértelmezetten perzisztens lenne, ha mégis tranziensként szeretnénk kezelni, akkor azt jeleznünk kell a @Transient annotáció használatával. Ez persze nem teljesen igaz, mivel az annotációk használata esetén például ajánlott nem az adattagokat annotálni, hanem a hozzájuk tartozó gettereket, és előfordulhat, hogy nem teljesen egy az egyhez megfeleltetés áll fenn az adattagok és a getterek között. A kapcsolatok definiálásakor lehetőségünk van 1:1, 1:N (illetve N:1) és M:N kapcsolatok definiálására a @OneToOne, @OneToMany, @ManyToOne, @ManyToMany annotációk használatával. Ezen kívül itt megadhatjuk a fentebb már említett kapcsolótáblára vonatkozó adatokat, mint a tábla neve, vagy a kapcsoló oszlop neve, továbbá a @Cascade annotáció segítségével a kapcsolatra vonatkozó kaszkádolási beállításokat is azaz azt, hogy a kapcsolat egyik résztvevőjén bizonyos műveletek végrehajtása végett végbement változtatás esetén a másik oldali résztvevőre ez milyen hatással legyen. Erre nagyon jó példa mondjuk az az eset, amikor egy üzenet és az azt tartalmazó témakör kapcsolatát tekintjük egy fórum vagy blog esetében: ha az üzenetet például törölni szeretnénk, akkor nem szabad a témakört is törölni, viszont a témakör törlésekor az üzenet is haszontalanná válik, így a törlést arra is alkalmazni kell. A kapcsolatok konfigurálásánál döntenünk kell arról is, hogy a kapcsolt objektum mohó (eager) vagy lusta (lazy) stratégia alkalmazásával legyen betöltve. A mohó stratégia kikényszeríti az aktuális objektum betöltése esetén az ilyen módon megjelölt kapcsolatok objektumainak betöltését is, míg a lusta stratégiával megjelölt kapcsolatok esetében a kapcsolt objektumok betöltése nem történik meg, amíg ténylegesen nincs szükség erre, azaz amíg nem hivatkozunk a kapcsolt objektumra vagy objektumokra. Ebből persze számos probléma adódhat, mivel a lusta stratégia működésének feltétele, hogy a kapcsolt

objektumra való hivatkozás idején a Hibernate Session (munkamenet) nyitva legyen, amelyik a jelenlegi objektumot betöltötte. A kapcsolatok definiálásakor könnyedén készíthetünk kétirányú (bidirectional) illetve egyirányú (unidirectional) kapcsolatokat is. Ha kétirányú kapcsolatot szeretnénk használni két osztály példányai között, akkor csak annyit kell tennünk, hogy a megfelelő számossággal létrehozunk egy-egy annotációt mindkét oldalon arra vonatkozóan, hogy a két osztályból mely adattagok, illetve getterek segítségével szertnénk elérni a másikat, majd kijelöljük, hogy melyik oldal birtokolja a kapcsolatot (owning side), és a másik osztályban a birtokos eléréséhez használt kapcsolat létrehozására használt annotációnál megadjuk a mappedBy attribútum értékeként a birtokos azon adattagjának nevét, amelyiket a kapcsolat felépítéséhez használtuk. Ezen felül természetesen rengeteg lehetőségünk van még a Hibernate által legenerált adatbázis séma testreszabására, de mivel ez nem tartozik szorosan a témához, ezeket nem részletezem.

Az adatok mentéséhez, frissítéséhez, eléréséhez több megoldást is használhatunk. Az egyik lehetőségünk a Hibernate saját SQL szerű nyelvének, a HQL-nek (Hibernate Query Language) a használata, amellyel az SQL-ből megszokott utasításokhoz igen hasonló megoldásokkal manipulálhatjuk a perzisztens objektumainkat. A másik említésre méltó alternatíva a Criteria API használata, amely lehetővé teszi OO eszközök használatával ugyanezt. A Criteria API használata sokkal inkább barátságos lehet azoknak a programozóknak, akik nem szeretnének SQL vagy ahhoz hasonló eszközökhöz nyúlni, inkább azt szeretnék, hogy az adatok eltárolása során is olyan programozási eszközöket használhassanak, mint az alkalmazásuk többi részében. Ennek azonban természetesen ára van, mégpedig a teljesítmény lesz várhatóan kissé gyengébb, mint a HQL használata esetén. Ez a teljesítmény csökkenés kis rendszerek esetében abszolút elhanyagolható. Fontos megemlíteni azt is, hogy a Hibernate csak lehetőséget ad ezeknek az eszközöknek a használatára, de nem kötelező élni ezzel, mivel akár SQL utasításokat is használhatunk, ha nekünk az kényelmesebb. Természetesen nem csak SELECT utasításokra vagyunk képesek a Hibernate segítségével, például megfelelő beállítások mellett a rendszer automatikusan létrehozza vagy módosítja a tábláinkat, illetve lehetőségünk van tranzakciók indítására, véglegesítésére vagy épp visszagörgetésére is, amely a mai alkalmazások tekintetében szerintem alapvető elvárásnak számít.

A Hibernate használatának számos előnye van a JDBC kapcsolatokkal és a többi ismert

megoldással szemben. A nekem leginkább tetsző tulajdonságok között élen jár az a tény, hogy mivel HQL illetve Criteria API használatával kezeljük az adatainkat, így a konkrét RDBMS rendszer lecserélése pár sor módosításával megtörténhet. Itt persze említhetnénk ellenpéldákat, mert bizonyára vannak, de az elméleti lehetőségünk adott erre, és ha úgy használtuk a Hibernate eszközeit, ahogyan azt a fejlesztők kitalálták, akkor igen jó esélyünk van arra, hogy ez ténylegesen meg is valósuljon, és mondjuk MySQL-ről Postgres-re váltani kevesebb, mint 10 sor átírásával megoldható legyen. Persze nagyra értékelem azokat a funkciókat is, amelyek lehetővé teszik, hogy ne kelljen manuálisan felolvasni a táblák tartalmát egy-egy lekérdezés után, vagy mondjuk kézzel írni a táblák létrehozásához használt CREATE utasításokat, ezért is döntöttem amellett, hogy az alkalmazásomban a Hibernate előnyeit kihasználva fogom a perzisztencia réteget megvalósítani.

(Hibernate Reference Documentation: 2.2.2. Mapping simple properties)

(Hibernate Reference Documentation: 20.1.1. Working with lazy associations)

(Hibernate Reference Documentation: 15. HQL: The Hibernate Query Language)

(Hibernate Reference Documentation: 16. Criteria Queries)

(Hibernate Reference Documentation: 17. Native SQL)

(Hibernate Reference Documentation: 7. Association Mappings)

3.6. PDF generálás

A PDF (Portable Document Format) az Adobe Systems által készített formátum megnevezése, amelyet dokumentumok alkalmazásoktól, operációs rendszertől vagy hardvertől független megjelenítésére fejlesztettek ki. Ez azt jelenti, hogy függetlenül attól, hogy milyen környezetben készítettük az adott dokumentumot, minden környezetben ugyanúgy kell kinéznie, amikor valaki meg szeretné nyitni azt. A formátum eleinte zárt volt, de 2008 óta nyílt ISO standardként van jelen. A PDF dokumentumokkal a mindennapi élet során elég gyakran találkozhatunk, mivel nagyon kedvelt a webalkalmazások körében a PDF dokumentumok generálása, például az internetes áruházak kedvelt megoldása, hogy PDF-ben tölthetjük le a vásárláshoz tartozó számlát.

Az alkalmazásomban a betegek leleteihez illetve vizsgálataihoz tartozó adatlapok kapcsán alkalmaztam PDF dokumentumokat, mivel mind az adatlapok közvetlen nyomtatásakor, mind a letöltésük alkalmával létrejön egy-egy PDF dokumentum a megfelelő adatokkal, hogy a felhasználó által kért művelet elvégzése megkezdődhessen. Ebben a szakaszban ezeknek a PDF dokumentumoknak a létrehozására szeretnék koncentrálni.

A feladat elkezdése előtt nem volt tapasztalatom PDF dokumentumok létrehozásával kapcsolatosan, ezért olyan eszközök kereséséhez láttam, amelyekkel lehetőségem nyílik Java programból elérni a célomat, azaz valahogyan PDF dokumentumokat létrehozni vagy szerkeszteni. A keresésem eredményeként két olyan eszközt ismerhettem meg, amelyek tökéletesen megfelelőek voltak az alkalmazás szempontjából. Ezeket fogom ismertetni a következő sorokban.

Az első megoldás az iText library használata lehet, amely kitűnő referenciákkal rendelkezik, mivel az elégedett iText felhasználók között olyan vállalatokat találhatunk, mint a Google, a New York Times, az U.S. Department of Defense, a NASA, a Manning Publications Co. vagy éppen az Adobe, amely a ColdFusion megvalósításához használ iTextet. Úgy gondolom, hogy ez igazán elismerésre méltó lista, amelyen talán az Adobe megjelenése a legelismerőbb, mivel amint azt ismertettem már a korábbiakban, az egész PDF formátum kifejlesztése hozzájuk köthető, és ennek ellenére nem a saját megoldásukat választották, hanem az iTextet. Az iText előnyeinek listája az alábbiakat tartalmazza:

- Open Source

- A piacon található hasonló megoldások között a legjobban dokumentált (az iText in Action című könyv segít az eszközök használatában)
- Nagyon jó teljesítményű, a piac jelenleg legmegbízhatóbb ilyen eszköze, amely nagy adatmenyiség esetén is kitűnően használható
- A legtöbb PDF-fel kapcsolatos feladat megoldására használható.

Amikor megvizsgáltam, illetve kipróbáltam ezt a library-t akkor azt vettem észre, hogy nekem nem erre van szükségem, mivel nem szeretném a programomban kézzel összeállítani a PDF dokumentumok tartalmát, ugyanis az extra igények kielégítéséhez szükséges eszközök használata kissé körülményes néha. Azért is elvetettem az iText használatát, mert nekem nincs ennyi lehetőségre szükségem, nem szeretnék minden olyan lehetőséget kihasználni, amelyet az iText biztosít, és így feleslegesnek éreztem egy ilyen eszköz integrálását, mindamellett, hogy kitűnő minőségű eszközökről van szó.

A másik lehetséges megoldás, amelyet végül be is ágyaztam az alkalmazásomba, a JasperReports nevű volt. Ez egy olyan eszköz, amelyet nem kifejezetten PDF generálására, hanem inkább riportok készítésére fejlesztettek ki, és több formátumot támogat a PDF mellett, mint például a XML, HTML, CSV, XLS, RTF, TXT. Számomra azért tűnt jobb döntésnek ezt a megoldást választani, mert egyszerű sablonokat, úgy nevezett riportokat képes kezelni a JasperReports, amelyet a Netbeans alapú szerkesztőjével, az iReporttal könnyedén meg lehet szerkeszteni, mintha csak egy irodai szoftvercsomagról lenne szó, és csupán jelölni kell a dinamikus adatok helyét, illetve be kell állítanunk azt, hogy hogyan szeretnénk a dokumentumunkat tartalommal feltölteni. Rengeteg beállítással van lehetőségünk testre szabni a riportokat, illetve több adatforrásból is szerezhetjük adatainkat, legyen szó SQL, XML és az arra épülő technológiák, vagy akár Java Bean-ek használatáról. Az én esetemben XML technológiák támogatására volt szükség, mivel nem szerettem volna SQL adatforrást kezelni a PDF generálás során sem, ha már Hibernate mellett döntöttem a perzisztencia kérdésében, illetve nem szerettem volna a Java Beaneket sem használatba venni ez ügyben, mivel a nyomtatás egy másik részfeladatához már úgyis legeneráltam egy XML dokumentumot, amelyet itt is könnyedén használatba vehettem. Fontos volt számomra az is, hogy a betegekhez tartozó egyedi azonosítókat nyomtatáskor illetve PDF dokumentumok generálásakor a későbbi feldolgozás megkönnyítésének érdekében ne szöveggel hanem vonalkódok használatával tudjam elhelyezni a dokumentumokban, amely nagyon egyszerűen

megoldható volt a JasperReports használatával a beépített Barbecue nevű vonalkód generáló library-nak köszönhetően. Persze ennek a megoldásnak is vannak hátrányai, mint például az, hogy a megszerkesztett sablonokat, amelyek egy XML dokumentumba kerülnek mentésre (.jrxml fájlok), még külön le kell fordítani miután módosítottunk rajtuk az iReport segítségével, és csak így kapjuk meg a .jasper kiterjesztésű riport állományokat, amelyeket már használhatunk is. Ez a fordítás történhet az iReport beépített eszközével is, vagy megtehetjük Java programból is közvetlenül, ám én biztonságosabbnak éreztem azt, hogy manuálisan fordítom le szerkesztés után ezeket a riportokat, mert így hiba vagy figyelmeztetés esetén még korrigálhatok, míg ha futási időben történik valami rendellenes a legutóbbi szerkesztés hibássága miatt, a hiba javítása sokkal tovább tarthat.

3.7. Appletek

A Java Appletek olyan Java programok, amelyeket weboldalak HTML kódjába ágyazva helyeznek el a fejlesztők és a kliens számítógépén futnak le. Az appletek általános jellemzője, hogy nem a szokásos main() metódussal szoktuk elindítani őket, hanem a főosztályuk az Applet osztály egy leszármazottja kell legyen, és több metódus implementációjával reagálhatunk különböző eseményekre, mint például az init(), amely az applet betöltődésekor fut le, a start() az applet indításakor, a paint() akkor, ha a böngésző szeretné újrarajzolni részben vagy egészében az appletet, a stop() akkor, ha le szeretnénk állítani az appletet illetve a destroy() akkor, amikor a böngésző eltávolítja az appletet a munkaterületről.

(Java 5 Belépés a programozás világába 328-330. oldal)

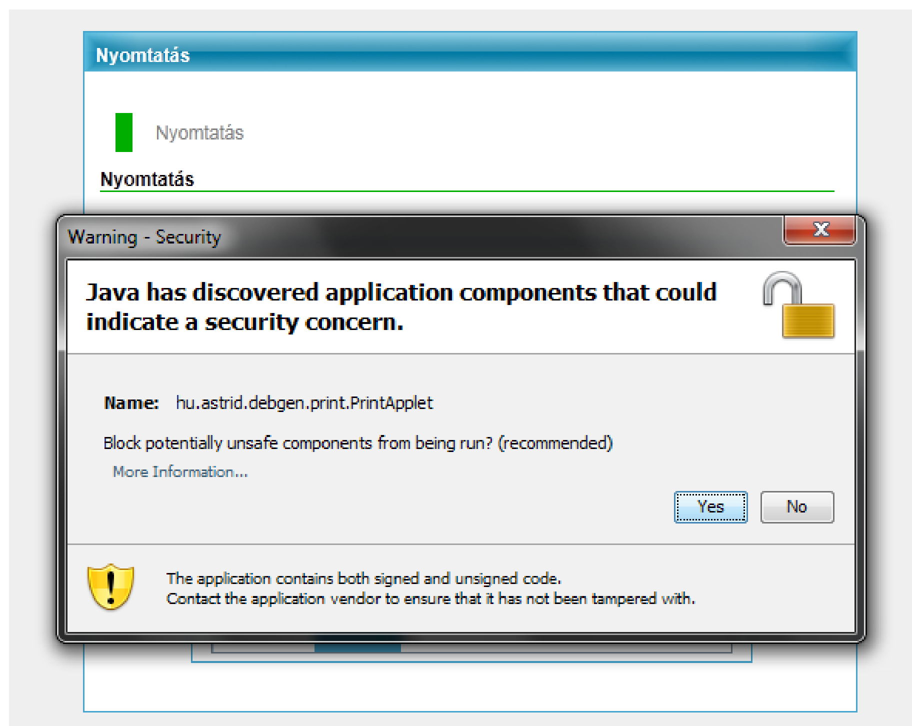
Az appletekkel kapcsolatosan tudni kell azt, is, hogy a biztonságosság fenntartása végett szigorúbb megkötéseknek kell megfelelniük, mint az egyéb Java alkalmazásoknak, mivel könnyedén lehetne őket különben kár okozására használni úgy, hogy az áldozatok nem is igazán tudnának tenni ellene, ha a beágyazó oldalra látogatnak. Ezek a megkötések az alábbiakban olvashatóak:

- Egy applet nem olvashat vagy írhat a kliens számítógépének fájlrendszerén egyetlen fájlt sem
- Az applet nem léphet kapcsolatba más számítógéppel, csak azzal, amelyikről letöltötte a kliens.
- Az applet nem férhet hozzá rendszer-információkhoz.
- Ha új ablakot nyit az applet, akkor a felhasználó egyértelműen értesítést kap arról, hogy nem biztonságos ablakról van szó.

(Java 5 Belépés a programozás világába 350-351. oldal)

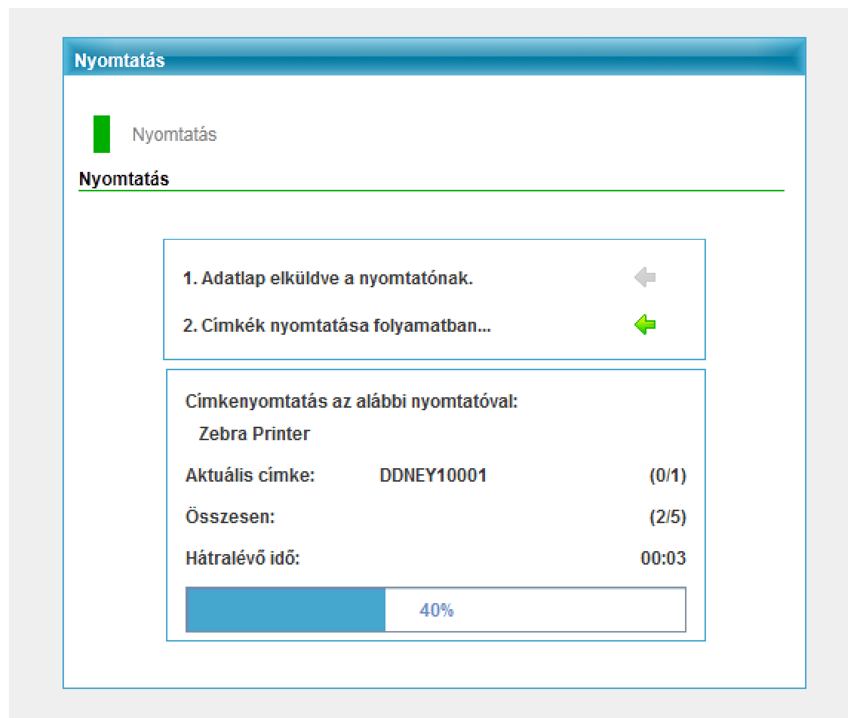
A fentebb említett megkötések enyhíthetőek abban az esetben, ha az applet készítője aláírja az appletet. Aláírt applet már képes például a fájlrendszer fájljait írni vagy olvasni, illetve ez szükséges ahhoz is, hogy például nyomtatni tudjunk a kliens nyomtatójával. Ez persze nem azt jelenti, hogy ha aláírunk egy appletet, akkor a megkötések egyáltalán nem érvényesülnek, csupán egy kérdés jelenik meg a felhasználó számítógépén az applet betöltésekor, amely az aláíró adatait ismerteti, illetve megerősítést kér az applet megbízhatóságával kapcsolatosan. Ha a felhasználó ekkor úgy dönt, hogy nem megbízható az

applet, akkor az applet futtatása el sem kezdődik. A biztonsági aggályokat felvető műveletek előtt további megerősítési kérések jelenhetnek meg. Például nyomtatásakor az alábbi kérdést láthatjuk:



Ábra 1: Biztonsági megerősítés nyomtatás előtt

Az alkalmazásomban a nyomtatás megvalósítására használtam Java Applet technológiát. Ebben a kérdésben is több lehetőség közül választhattam, mivel nem csak az appletek segítségével lehetséges a kliens gépén Java programot futtatni, hanem például a Java WebStart használatát is választhattam volna, viszont úgy éreztem, hogy egy HTML oldalakból álló, webes alkalmazáshoz inkább illik egy a környezetébe beágyazódó applet, mint egy WebStart-tal megvalósított alkalmazás. A kifejlesztett applet a paraméterül kapott URL-ek segítségével fér hozzá egy XML dokumentumhoz, amelyet vonalkódos címkék egy speciális címkenyomtatón való kinyomtatásához hasznosít, illetve egy PDF dokumentumhoz, amelyet a számítógéphez csatlakoztatott hagyományos nyomtatón próbál kinyomtatni a megadott példányszámban. A nyomtatások miatt nagy szükség volt az applet aláírására. Az appletem végleges formája a 2. ábrán tekinthető meg a beágyazó HTML oldal egyes részeivel együtt.



Ábra 2: A nyomtatáshoz használt applet

3.8. XML technológiák

Ebben a fejezetben az alkalmazás által használt XML technológiák közül ismertetek nagyon röviden kettőt, ám mivel a dolgozat nem XML témájú és nem szeretnék több oldalt szentelni ennek a területnek, a többi használt technológiát csupán megemlítem. Az összes technológia kifejtése ugyanis a dolgozat terjedelmét meghaladná, viszont mindenképpen szerettem volna megjegyezni, hogy XML technológiákat is használok, illetve röviden bemutatni az általam használtak közül kettőt. A témák kiválasztását illetően a választásom az XML nyelvre illetve a DOM-ra esett.

3.8.1.XML

Az XML szó az Extensible Markup Language, azaz a Kiterjeszthető Jelölő Nyelv rövidítése. Az XML és a HTML (HyperText Markup Language) közös őstől, az SGML-től (Standard Generalized Markup Language) származó nyelvek, amelynek köszönhetően egy XML és egy HTML dokumentum felépítése nagyon hasonlónak tűnhet első ránézésre, sőt az XHTML dokumentumok a kettőt ötvözik is, mivel szabványos XML dokumentumok, amelyek teljesen ugyanazt a célt szolgálják, mint a HTML dokumentumok. Az XML jelentősége azonban jóval nagyobb annál, mint hogy XHTML dokumentumok révén beszélhessünk róla, mivel az informatika bármely területén találkozhatunk XML dokumentumokkal, és azt hiszem bátran kijelenthetjük, hogy a számítógépet használók többsége akár anélkül is, hogy tudott volna erről, de használt már XML dokumentumokat. Ez annak köszönhető, hogy például az irodai szoftvercsomagok, mint például a LibreOffice vagy Microsoft Office, amelyekkel a felhasználók többsége találkozik, ma már elég sok XML formátumú dokumentumot használnak, vagy az internetes világban a technológiák nagyon nagy része használ XML-t, legyen szó RSS vagy Atom hírcsatornákról, SOAP alapú szoftverekről, időjárás előrejelzéseket megjelenítő minialkalmazásokról, Jabber (XMTP) protokollt használó azonnali üzenetküldő alkalmazásokról, vagy éppen XHTML alapú weboldalakról. Az előbbi példákból is látható, hogy az XML valóban nagyon elterjedt, így további példák helyett tekintsük meg, hogy mi is az XML az XML 10 pontjának segítségével.

1. Az XML célja az adatok strukturálása: Az XML nyelvet arra tervezték, hogy alkalmas legyen az általunk használt adatok jól definiált struktúrában történő leírására, amely

könnyedén használható az adatok gépi feldolgozására, miközben nagyon intuitív és ha szeretnénk emberként is egyszerűen és érthetően tudjuk kezelni a leírt információkat.

2. Az XML egy kicsit olyan, mint a HTML: Az XML és a HTML külsőleg tényleg nagyon hasonlónak mondható, mint ahogyan arra már fentebb is utaltam. Gondoljunk csak a címkékre (tag), amelyeket mindkét nyelvben < és > illetve </ és > jelek használatával határolunk. Ez nem a véletlen folytán van így, ugyanis az XML és a webböngészők fejlődése párhuzamosan segítette egymást. Az XML a hasonlóságok ellenére nem tévesztendő össze a HTML-lel mivel az XML definíciója jóval szigorúbb, kötöttebb, amely kötöttséghez hasonló követelmények HTML-lel kapcsolatosan nem léteznek, illetve fontos, hogy a címkék nevei XML-ben teljesen szabadak, nem tartozik hozzájuk előre meghatározott jelentés, míg a HTML-ben ennek pont az ellenkezője igaz.

3. Az XML szöveg, de nem olvasgatásra való: Ez a kijelentés kissé pontosításra szorulhat, mivel vannak leíró XML dokumentumok is, például a DocBook szabvány használatával létrehozott publikációk, amelyek XML formátumot használnak, szabad szemmel is könnyedén értelmezhetőek. El kell mondanunk azonban azt is, hogy a gépi feldolgozásra szánt XML dokumentumok legtöbbször nem értelmezhetőek egykönnyen emberek számára, mivel ezeket nem is emberi olvasgatásra szánták.

4. Az XML bőbeszédű, mert ilyennek tervezték: Mivel az XML szöveges formátumú, és a címkék neveit például kétszer is leírjuk ahhoz, hogy a mondjuk köztük található szöveges szekció tartalmát értelmezhessük, egyértelmű, hogy nem a legtömörebb ábrázolása ez az információnak, ám ez a szöveges formátum számos előnnyel jár, míg a terjedelem ma már hála a hatékony tömörítő eljárásoknak és az olcsó tárterületnek nem jelent hátrányt.

5. Az XML technológiák egy egész családja: Az XML használata nem csak az XML dokumentumok írásáról szól, hanem egy egész sor technológia épül rá, amelyek segítségével több problémát is meg tudunk oldani. Ilyen technológia például a DTD, az XML Schema, a Relax NG, a DOM, a SAX, az XLink, az XPath, az XQuery és még számos egyéb amelyeket most nem sorolnék fel.

6. Az XML új, de nem előzmények nélkül való: Mint azt már korábban említettem, az XML őse az SGML, amely 1986 óta ISO szabvány. Az XML fejlesztői az SGML legjobb részeit használták fel, majd azt kiegészítették a HTML-lel kapcsolatos tapasztalataikkal.

7. A HTML-től XML-en keresztül vezet az út az XHTML-ig: Az XHTML

dokumentumok a HTML dokumentumok utódai, amelyek már XML alapúak, így sokkal szigorúbb, kötöttebb formátumot használnak a HTML dokumentumokhoz képest, ugyanakkor a HTML címkéinek többségét az XML megkötéseinek megfelelőre transzformálva örökli az XHTML, így már egyértelmű jelentést is tudunk társítani az elemekhez.

8. Az XML moduláris: Az XML lehetőséget nyújt arra, hogy új dokumentum formátumokat hozhassunk létre régi formátumok alapján. Ehhez legnagyobb segítséget szerintem a névterek használata jelenthet.

9. Az XML az RDF és a Szemantikus Web alapja: Az RDF, a Resource Description Framework, azaz Erőforrásleíró Keretrendszer rövidítése, amely metaadatok leírásához használható eszközrendszer. Az RDF és a Szemantikus Web területekkel kapcsolatos kutatások napjaink és a közeljövő egyik nagy előrelépését jelenthetik, és valóban az XML áll a terület központjában.

10. Az XML licencmentes, platform-független és sokak által támogatott: Az XML elterjedéséhez és támogatottságához nagyban hozzájárult az, hogy licencmentességének köszönhetően bárki ingyenesen használhatja, és szöveges formátumának köszönhetően bármelyik platformon elérhető, szerkeszthető, és feldolgozható.

(XML in 10 points)

(XML 10 pontban)

3.8.2. DOM

A DOM (Document Object Model) egy nyelv és platformfüggetlen eszköz XML, HTML és XHTML dokumentumok elemeivel történő interakcióra. Ennél fogva kitűnő támogatást nyújt mind az XML dokumentumok felolvasására, mind módosításukra vagy létrehozásukra. A DOM nagy előnye a SAX (Simple API for XML) nevű alternatívával szemben, hogy nem csak olvasni, hanem módosítani is tudja az XML dokumentumokat, viszont ennek az előnynek is ára van, mivel ahhoz, hogy a DOM működni tudjon, az egész XML dokumentumot a memóriába kell olvasni és ott egy fát létrehozni belőle, hogy bármilyen műveletet végezhessünk rajta, míg a SAX csak végigolvassa a dokumentumot és eseményvezérelten végzi a feldolgozást. A DOM implementációja számos nyelven elérhető, így a Java programozóknak is.

A DOM feldolgozás első lépése az XML dokumentumból az elemzett dokumentum (parsed document) létrehozása, amelyet az elemző (parser) végez, majd ebből létrejön a DOM fa, amelyet a navigációkor használhatunk. A DOM fa úgynevezett csomópontokból (Node) áll. A DOM API két megközelítést alkalmaz a dokumentumok feldolgozása során. Az egyik objektum orientált módon, öröklődési hierarchia segítségével kezeli az elemeket, míg a másik egyértelműen azt vallja, hogy minden elem csomópont. A fa előnye a csomópontok gyors elérése, mivel mindegyik csomópont a memóriában található, és nem kell az XML-ből kikeresni őket. Ha a DOM-ot új XML dokumentumok létrehozása esetén használjuk, először egy dokumentum objektumot kell létrehoznunk, majd ehhez kell hozzáadnunk a megfelelő hierarchiában a csomópontokat. Amikor minden csomópont a helyére került, az egész dokumentum tartalmát könnyedén stream-re írhatjuk.

(What is the Document Object Model?)

3.8.3. XML technológiák az alkalmazásban

A dolgozat témájául szolgáló alkalmazásban is fontos szerepet játszanak az XML dokumentumok, ugyanis a vizsgálatkérő lapokból és leletekből generált PDF dokumentumokat, amelyeket letölthetnek a felhasználók, vagy a nyomtatás során használ a nyomtató applet, úgy hozom létre, hogy az érintett perzisztens objektumok felhasználásával készítek egy XML reprezentációt, amelyben az összes szükséges adat megtalálható a PDF generálás során használt sablon tartalommal való feltöltéséhez. Az XML dokumentumot a DOM segítségével hozom létre. A dokumentum felhasználásakor pedig XML Schema elleni validáció megy végbe, hogy biztos lehessek abban, megfelelő dokumentummal dolgozom a PDF generálás során. Az XML dokumentumot nyomtatás esetén nem távolítja el az alkalmazás akkor sem, ha már a PDF dokumentum létrejött, mivel a nyomtatást végző applet az XML fájlt is megkapja, hogy a címkenyomtatón nyomtatandó címkék száma és kódja a kért mennyiségnek megfelelő legyen. Az általam használt XML dokumentumok struktúráját úgy alakítottam ki, hogy emberi olvasásra is alkalmasak legyenek, ezért beszédes címkéket használtam, ami ugyan a hálózati kommunikáció szempontjából nagyobb terhet ró a kiszolgálóra, azonban a PDF generáláskor használt sablon szerkesztésekor, illetve egy esetleges hibakeresés alkalmával nem okoz gondot a bonyolult, és olvashatatlan fájlok szerkesztése vagy olvasása, továbbá a legenerálásra használt kód olvashatósága is sokat javul ennek köszönhetően. A PDF generálásakor használt sablonok az XML dokumentumok egyes elemeire az XPath használatával hivatkoznak.

3.9. JavaScript

A JavaScript egy a webes alkalmazások készítői között rendkívül elterjedt nyelv, amelyet többnyire kliens oldalon használnak, ám kiszolgálóoldalon is futtatható bizonyos körülmények között. A JavaScript használatát illetően meg kell említeni az asztali gadget-ek illetve widget-ek fejlesztését, valamint azt, hogy a PDF dokumentumokban is találkozhatunk JavaScripttel. A JavaScript az aktuális oldal kódjához képes hozzáférni és módosítani azt, ezzel késztetve a böngészőt a weblap megjelenítésének módosítására. A JavaScript képes kliensoldali változók, cookie-k értékének módosítására, átirányítások megvalósítására, az oldal teljes átformálására, illetve akár komolyabb számításokat is végezhetünk vele, vagy például nyomtatni is tudunk a segítségével, de az igazi erejét nem ebben látom.

A JavaScript kialakulása 1995-re tehető, amikor megjelent a Netscape 2.0-ás verziója. A JavaScript támogatását a következő évtől már az Internet Explorer 3-ba is beépítették, igaz a Microsoft jogi okokból JScript néven kezdett egy saját változat fejlesztésébe, amely többnyire a JavaScripttel egyező funkcionalitású volt, de tartalmazott pár újítást is, amelyek szinte kivétel nélkül kudarcnak bizonyultak. A következő nagy lépést az ECMAScript megjelenése jelentette 1997-ben, amelynek az első megvalósítása a JavaScript volt, de a JScript illetve ActionScript dialektusokat is meg kell említeni. Az ECMAScript 1998-óta már ISO szabvány is. Ma már a JavaScript segítségével igen komoly feladatokat oldanak meg a fejlesztők nap mint nap, és a webes fejlesztések többségében jelen van. A JavaScriptnek köszönhetően az alkalmazások készítői gazdag, igényes felületeket tudnak összeállítani. A JavaScript kombinálható szinte az összes webes technológiával és lehetővé teszi olyan feladatok megoldását is, mint például dinamikusan változó űrlapok megjelenítése, törlési megerősítések, vagy űrlapok tartalmának hálózati kommunikáció nélküli validálásának megvalósítása. Az egyik legjelentősebb JavaScriptre épülő megoldás azonban szerintem az AJAX (Asynchronous JavaScript and XML).

(JavaScript Language Overview)

(Netscape and Sun announce JavaScript, the open, cross-platform object scripting language for enterprise networks and the internet)

(JavaScript Zsebkönyv 1-3. oldal)

Az AJAX arra szolgál, hogy aszinkron kommunikációt létesíthessünk a szerverrel az

oldal újratöltése nélkül, majd a válasz segítségével frissítsük az oldal egyes részeit, vagy egészét. Az AJAX kialakulásához vezető rögös út azzal kezdődik, hogy 1990-es években az Internet Explorer 5 képes volt ilyen kérések használatára ActiveX és REST segítségével, majd később, amikor ezt felfedezték a Mozilla majd más böngészők fejlesztői, létrehozták a saját XMLHttpRequest implementációikat. Ez azonban még csak a kezdet volt, mivel az AJAX nagyon népszerű lett, és tulajdonképpen ennek volt köszönhető a JavaScript feltámadása is. A mai népszerű weboldalak üzemeltetői közül az egyik legjelentősebb AJAX felhasználó a Google. Az AJAX elnevezést illetően meg kell jegyezni, hogy tulajdonképpen nem is szükséges XML az AJAX megvalósításához. Az alkalmazásomban például JSON (JavaScript Object Notation) formátumban érkező válaszok segítségével valósítottam meg a JQuery AJAX megoldását használva a beteg TAJ szám alapján történő felismerését.

(Ajax: A New Approach to Web Applications)

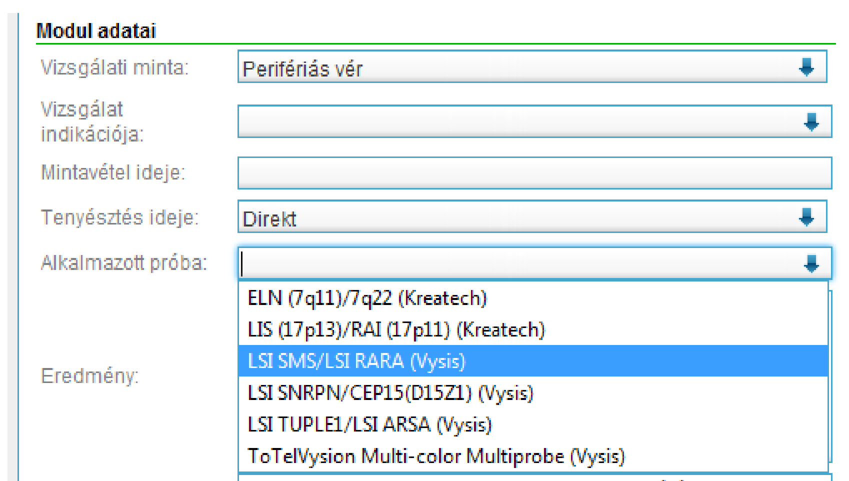
(JavaScript Zsebkönyv 219-221. oldal)

A JQuery egy olyan JavaScript eszközgyűjtemény, amely megkönnyíti a bonyolult felületek összeállítását, miközben feloldja a JavaScriptre egyébként jellemző böngészők közötti eltérésekből adódó problémák jelentős részét. Ennek köszönhető, hogy a jelenleg használt legnépszerűbb JavaScript library. A JQuery Project számos kisebb projektet tömörít, és tartozik hozzá egy JQuery UI nevű előregyártott felületelemeket tartalmazó gyűjtemény is, amelyben számos úgy nevezett widget megtalálható, amelyek egy-egy olyan problémára nyújtanak megoldást, amelyekkel a webfejlesztők rendszeresen szembe kell nézzenek. A JQuery UI elemeit egységes kinézettel láthatjuk el egy egyszerűen használható grafikus CSS generáló eszköz, az úgy nevezett ThemeRoller segítségével.

Az alkalmazásomban a fentebb már említett AJAX megoldáson túl a felhasználói munkamenet lejárta előtt történő figyelmeztetés megjelenítését, illetve a dátumválasztókat is JQuery segítségével valósítottam meg, ahogyan azt a 3. ábra is mutatja.

Ábra 3: JQuery dátumválasztó

Az űrlapok megjelenítésénél pedig szinte mindenhol használok JavaScript eszközöket, hogy a korábbi kérdésekre adott válaszok alapján el tudjam rejteni, illetve meg tudjam jeleníteni a nem releváns, illetve releváns részeit az űrlapoknak. Az űrlapokat tekintve még egy további igény is felmerül egyes helyeken, mégpedig az, hogy az egyes mezőkbe korábban beírt adatokra automatikusan ki lehessen egészíteni a jelenlegi szerkesztés során beírt szórészleteket. Ez igen hasznos lehet például keresés vagy olyan mezők esetén, ahol általában ugyanazon értékeket szokták megadni a felhasználók, de szükség lehet új elemek megadására is, emiatt nem lehet legördülő listát használni. Ennek a megoldására is a JavaScriptet illetve AJAX-os megoldást használtam. Erre a 4. ábrán láthatunk példát.



Ábra 4: Automatikus kiegészítés AJAX használatával

3.10. HTML és CSS

Az alkalmazás felhasználói felületének elemeit illetve azok kinézetét a HTML illetve a CSS technológiákkal valósítottam meg, ezekről szeretnék rövid leírást prezentálni ebben a fejezetben.

3.10.1. HTML

A HTML (HyperText Markup Language) egy leíró nyelv, amelyet arra használnak, hogy a benne definiált építőelemek (tag-ek, attribútumok) segítségével összeállítsanak a webböngészők által értelmezhető dokumentumokat. Ezen dokumentumok megjelenítése során a böngésző nem írja ki az elemek tartalmát a képernyőre, hanem értelmezi azokat, és a hozzájuk tartozó jelentést felhasználva jeleníti meg az oldal tartalmát. Ez böngészőtől függően eltérő lehet attól a kinézettől, amelyet a fejlesztő szeretett volna létrehozni, de közelítőleg egyező eredmény várható a korszerű böngészőktől.

A HTML története az 1980-as években kezdődik. Ekkor alkotta meg Tim Berners-Lee, fizikus a CERN számára a kutatók közötti dokumentum-megosztás megvalósítására. 1990-végére hozta létre a kliens és a szerver programot a HTML használatához. Ekkor még a HTML eszköztára igencsak szegényesnek mondható volt, mivel ekkor még csak 20 címkét tartalmazott a nyelv, melyek többsége (a hyperlink kivételével) az SGML-ből származott. Ezen címkék közül 13 még a HTML 4-es verziójában is megtalálható. Berners-Lee a HTML-t az SGML egy felhasználásának tekintette, a HTML publikálása 1993-ra tehető, amikor Berners-Lee és Dan Connolly kiadta a HTML vázlat verzióját, amely már egy SGML DTD-t (Document Type Definition) is tartalmazott a nyelvtan definiálására. A HTML dokumentumok validálására ma is DTD-t használnak. Az Internet Engineering Task Force (IETF) a publikációt követően HTML csoportot alapított és hamarosan kidolgozták a HTML 2.0-ás verzióját. A HTML gondozását 1996 óta a World Wide Web Consortium (W3C) végzi, illetve 2000 óta ISO szabványként beszélhetünk a nyelvről. A 4.01-es verzió 1999-ben jelent meg, amelyet még ma is sok alkalmazáshoz használnak. Napjainkban komoly szakmai érdeklődés fogadta, illetve fogadja a HTML5 megjelenését, illetve az új funkciókat, amelyek már beágyazott médialejátszást is lehetővé tesznek. Az egyetlen probléma, ami a HTML5 terjedését hátráltatja az, hogy a böngészők még nem teljesen támogatják a használatát, ezért

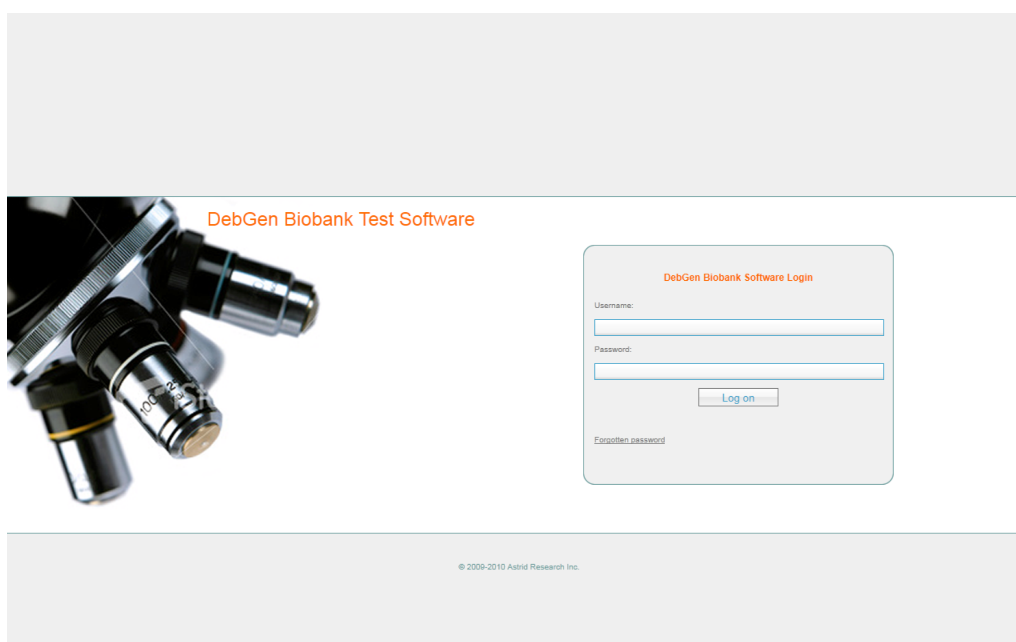
körülményes ilyen oldalakat karbantartani, illetve publikálni.

(Information Management: A Proposal)

(HTML Tags)

(HTML5 differences from HTML4: Miscellaneous)

Az alkalmazásomban megjelenő oldalak kivétel nélkül HTML nyelvűek, habár nem HTML hanem JSP fájlokat szerkesztettem az elkészítésük során, hogy az oldalak statikus vázát dinamikus tartalommal tudjam feltölteni, viszont a szerver oldali kód lefutása után HTML oldalak állnak elő, amelyeket a böngésző segítségével tudnak megjeleníteni a felhasználók. Az 5. ábra szemlélteti a bejelentkező felületet.



Ábra 5: Az alkalmazás bejelentkező felülete

3.10.2. CSS

A CSS szó a Cascading Style Sheets rövidítése. A CSS arra szolgál, hogy az általunk használt webes dokumentumok kinézetét könnyedén meg tudjuk adni. A CSS első verziója, a CSS level 1 1996 decemberében jelent meg W3C ajánlásként. A CSS ezen verziója még leginkább csak a szövegek betűtípusának, színének, hátterének, a szavak közötti távolságnak a szabályozására volt alkalmas. A CSS level 2, azaz a CSS második verziója már jelentős újításokat hozott 1998-ban, mint a z-index vagy a relatív vagy abszolút pozicionálás lehetősége, illetve a szövegek árnyéka. A CSS 2.1 volt a második verzió kiterjesztése, amelyet utoljára 2010 végén módosított a W3C, miközben folyik a munka a harmadik verzión. A CSS 2-vel ellentétben a CSS 3 nem egy nagy specifikáció, hanem több kisebb dokumentum, amelyeket moduloknak hívnak. A modularizációból fakadóan adódik, hogy a különböző modulok különböző státuszban állnak, de ma már több, mint 40 modul stabilnak mondható, és kiadás előtti státuszba került.

(What is CSS?)

(Cascading Style Sheets, level 1)

(Document Object Model (DOM) Level 2 Style Specification)

A CSS stílusok használatára három alapvető módon van lehetőségünk. Az első, az úgy nevezett szövegközi stílusok használata, amely a HTML vagy XHTML elemek style attribútumával tehető meg. Ekkor a style attribútum értékeként kell megadnunk az adott elemre vonatkozóan, hogy milyen tulajdonságait milyen értékekkel szeretnénk használni, például `style="display: block"`. A második lehetőség, a beágyazott stíluslapok használata a style tag segítségével. Ez úgy tehetjük meg, hogy a HTML dokumentum head elemén belül egy `<style type="text/css">` nyitó és egy `</style>` záró elem közé beírjuk a használni kívánt CSS stíluslap tartalmát. A harmadik módszer az, hogy egy külső CSS stíluslapot hozunk létre külön fájlban, majd ezt a link elem segítségével, vagy egy beágyazott stíluslapon belül a @import segítségével tölthetjük be azon az oldalon, ahol használni szeretnénk. A külső stíluslapok használata számos előnnyel járhat, például nem keveredik a HTML kód a megjelenítési információkkal, egyetlen CSS stíluslapon össze tudjuk gyűjteni az alkalmazásra vonatkozó összes stílus információt, illetve mivel a böngészők ezeket a külső stíluslapokat gyorsítótárakban tárolják, a weboldal betöltése is gyorsabb lesz,

mert kevesebb adatot kell letöltenünk.

Sajnos a CSS kapcsán is meg kell jegyeznünk azt, amit a JavaScript tárgyalásakor is említettem, hogy sajnos az egyes böngészők készítői kissé eltérően értelmezik a szabványt, illetve olyan is könnyen előfordulhat, hogy egy újabb verziós CSS-t egyáltalán nem támogat a böngészőnk, ezért érdemes több böngésző használatával tesztelnünk a webalkalmazásunkat, hogy a legnépszerűbb böngészők hiba nélkül meg tudjanak birkózni az alkalmazásunk megjelenítésével.

(CSS Zsebkönyv 1-5. oldal)

Az alkalmazásomban CSS 2.1-et használtam, és igyekeztem az Internet Explorer, Mozilla Firefox, Opera és Google Chrome böngészők által hibátlanul értelmezett oldalt létrehozni. A 6. ábrán azt szeretném bemutatni, hogy a HTML-lel foglalkozó szekció végén megtalálható 5. ábra által bemutatott bejelentkező képernyő CSS használata nélkül milyen képet mutatna.

DebGen Biobank Test Software

DebGen Biobank Software Login

Username:

Password:

Log on

Forgotten password

© 2009-2010 Astrid Research Inc.

Ábra 6: A bejelentkező felület CSS nélkül

4. Összefoglalás

Az alkalmazás használható lenne genetikai kutatáshoz, illetve diagnosztikához, így azt mondhatom, hogy az alkalmazás elkészítése során az általam kitűzött céloknak sikerült megfelelnem. Természetesen nem állítom, hogy tökéletes megoldásról beszélhetünk, mivel az eddigi tapasztalataim alapján azt kell mondjam, tökéletes szoftverek nem is léteznek. A technológiai megoldásokkal kapcsolatban mint azt a korábbi fejezetekben igyekeztem is érzékeltetni, minden általam kiválasztott eszköz szerves részét képezi az alkalmazásnak, így a kezdeti terveim erre vonatkozóan is teljesültek, nem volt szükség azok módosítására. A Java technológiák közül a JavaServer Pages, a Java Servlet, illetve a Java Applet technológiák elegendőnek bizonyultak a feladat megoldásához, így nem érzem hibának, hogy ezeket választottam, bár kétségkívül igaz az, hogy az általam készített Servlet-eket ma már biztosan a Struts 2 eszközeivel helyettesíteném, mert sokkal tisztább, egyszerűbb megoldásokkal lehet ugyanazt a hatást elérni, ami a későbbi karbantartás esetén is komoly előnyt jelenthet. Az alkalmazás többi részét azt hiszem ma is hasonló eszközökkel kezdeném el megvalósítani.

Az alkalmazás továbblépési lehetőségeit tekintve számos irányt látok, ezeket az alábbi szakaszokban szeretném taglalni.

Az első, és egyben legjelentősebb fejlődés az lenne, ha az alkalmazás és az alkalmazó intézmény egészségügyi információs rendszere között lenne egy jól definiált interfész, amely megkönnyítené a diagnosztikai jellegű adatok rögzítését, így a biobankban részt vevő betegek rendszeres vizsgálatakor nem róna plusz terhet a vizsgálatokat végző személyzetre, hogy az elvégzett vizsgálatok eredményeit a biobankban is rögzítsék, illetve az intézmény egészségügyi információs rendszerében is. Azt belátni, hogy mindkét helyen dokumentálni kell az eseteket nem nehéz, mivel a biobank szempontjából az adatok rögzítése esszenciális lehet, míg az egészségügyi információs rendszerbe az intézmény szabályai, illetve a betegen elvégzendő más vizsgálatok miatt van szükség, hogy a pontos kórelőzmény segítse a vizsgálatokat végző orvosok munkáját.

A második komoly továbblépési irányként azt látom, hogy egyfajta LIMS (Laboratory Information Management System) megoldást integráljunk az alkalmazásba, mert ez nagyban segítené a beérkezett minták nyomon követését.

Fejleszteni lehetne még a felhasználói felületen is, mivel ha több AJAX-os kérést

használnék, akkor gazdagabb, ugyanakkor kényelmesebben használható felhasználói felületen érhetnék el a felhasználók az alkalmazást, illetve a hálózati forgalom csökkenése is elérhető lenne megfelelően megtervezett, illetve megvalósított AJAX alapú UI használata mellett.

Az alkalmazás értéke növelhető lenne még akkor is, ha az adatbányászati technikák igényeit figyelembe véve alakítanánk ki az adatmodellt, hogy a biobank által tárolt adatok könnyedén elemezhetőek legyenek. Ez azonban elég komoly feladat lenne, mivel a jelenlegi OLTP adatbázis mellé célszerű lenne egy OLAP megoldás létrehozása, és az adatok aggregált kezelése, illetve bizonyos időközönként az OLTP és OLAP adatbázis közötti adatmozgatás automatikus vagy félautomatikus megvalósítása.

5. Irodalomjegyzék

- Dirk Louis - Peter Müller: Java 5 Belépés a programozás világába

  Panem Könyvkiadó 2006

  ISBN 963 545 4546
- Christian Wenz: JavaScript Zsebkönyv

  Kiskapu Kft. 2006

  ISBN 978 963 9637 22 1
- Eric A. Meyer

  Kiskapu Kft. 2006

  ISBN 963 9637 11 4
- Mi a biobank?

  http://www.biobanks.hu/index.php?option=com_content&view=article&id=48&Itemid=27&lang=hu

  2011-04-15 17:04
- Grilled meat consumption and PhIP-DNA adducts in prostate carcinogenesis

  http://www.ncbi.nlm.nih.gov/pubmed/17416774

  2011-04-15 18:12
- Will Biobanking Change the World?

  http://www.scientificcomputing.com/Articles-IN-Will-Biobanking-Change-the-World-061710.aspx

  2011-04-16 13:46
- Törvényi háttér

  http://www.biobanks.hu/index.php?option=com_content&view=article&id=50&Itemid=34&lang=hu

  2011-04-15 18:20

- Client/Server Architectures

  https://www.thedacs.com/databases/url/key/216

  2011-03-28 18:01
- The Client Server Architecture

  http://www.webdevelopersnotes.com/basics/client_server_architecture.php3

  2011-03-29 19:09
- Martin Fowler: Front Controller

  http://www.martinfowler.com/eaaCatalog/frontController.html

  2011-03-30 17:13
- MSDN: Front Controller

  http://msdn.microsoft.com/en-us/library/ff648617.aspx

  2011-03-30 19:30
- IBM: Model-View-Controller (MVC)

  http://www.ibm.com/developerworks/library/j-struts/#h3

  2011-03-30 17:34
- Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller

  http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html

  2011-03-30 18:02
- Apache Struts

  http://struts.apache.org/index.html#Welcome

  2011-03-30 20:03
- JavaServer Pages Access Model

  http://www.kirkdorffer.com/jspspecs/jsp092.html#model

  2011-03-30 21:21
- Understanding JavaServer Pages Model 2 architecture

  http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc.html

  2011-03-30 20:42
- MSDN: Layered Application Guidelines

  http://msdn.microsoft.com/en-us/library/ee658109.aspx

  2011-03-31 19:02

- Servlet Essentials

  http://www.novocode.com/doc/servlet-essentials/chapter1.html

  2011-03-24 19:03

- An Introduction to Java Servlets

  http://www.webdevelopersjournal.com/articles/intro_to_servlets.html

  2011-03-26 11:20

- Expression Language

  http://download.oracle.com/javaee/1.4/tutorial/doc/JSPIntro7.html

  2011-03-27 17:05

- Hibernate: What is Object/Relational Mapping?

  http://www.hibernate.org/about/orm

  2011-03-31 19:56

- The Object-Relational Impedance Mismatch

  http://www.agiledata.org/essays/impedanceMismatch.html

  2011-04-01 17:04

- The Vietnam of Computer Science

  http://blogs.tedneward.com/2006/06/26/The+Vietnam+Of+Computer+Science.aspx

  2011-04-01 18:23

- Hibernate Reference Documentation: 2.2.2. Mapping simple properties

  http://docs.jboss.org/hibernate/annotations/3.5/reference/en/html_single/#entity-mapping-property

  2011-04-02 12:14

- Hibernate Reference Documentation: 20.1.1. Working with lazy associations

  http://docs.jboss.org/hibernate/stable/core/reference/en/html/performance.html#performance-fetching-lazy

  2011-04-02 10:20

- Hibernate Reference Documentation: 15. HQL: The Hibernate Query Language

  http://docs.jboss.org/hibernate/stable/core/reference/en/html/queryhql.html

  2011-04-02 14:25

- Hibernate Reference Documentation: 16. Criteria Queries
  http://docs.jboss.org/hibernate/stable/core/reference/en/html/querycriteria.html
  2011-04-02 15:12
- Hibernate Reference Documentation: 17. Native SQL
  http://docs.jboss.org/hibernate/stable/core/reference/en/html/querysql.html
  2011-04-02 15:17
- Hibernate Reference Documentation: 7. Association Mappings
  http://docs.jboss.org/hibernate/stable/core/reference/en/html/associations.html#assoc-intro
  2011-04-01 22:05
- XML in 10 points
  http://www.w3.org/XML/1999/XML-in-10-points.html
  2011-04-04 17:32
- XML 10 pontban
  http://www.w3c.hu/forditasok/XML_10_pontban.html
  2011-04-04 18:13
- What is the Document Object Model?
  http://www.w3.org/DOM/#what
  2011-04-07 21:12
- JavaScript Language Overview
  http://jibbering.com/faq/#tips
  2011-04-09 11:38
- Netscape and Sun announce JavaScript, the open, cross-platform object scripting language for enterprise networks and the internet
  http://web.archive.org/web/20070916144913/http://wp.netscape.com/newsref/pr/newsrelease67.html
  2011-04-09 13:05
- Ajax: A New Approach to Web Applications
  http://www.adaptivepath.com/ideas/e000385
  2011-04-09 14:21

- Information Management: A Proposal
  http://www.w3.org/History/1989/proposal.html
  2011-04-10 16:47
- HTML Tags
  http://www.w3.org/History/19921103-hypertext/hypertext/WWW/MarkUp/Tags.html
  2011-04-10 17:32
- HTML5 differences from HTML4: Miscellaneous
  http://www.w3.org/TR/html5-diff/#syntax-misc
  2011-04-10 19:43
- What is CSS?
  http://www.w3.org/standards/webdesign/htmlcss#whatcss
  2011-04-16 18:20
- Cascading Style Sheets, level 1
  http://www.w3.org/TR/2008/REC-CSS1-20080411/
  2011-04-16 19:42
- Document Object Model (DOM) Level 2 Style Specification
  http://www.w3.org/TR/2000/REC-DOM-Level-2-Style-20001113/
  2011-04-16 20:57

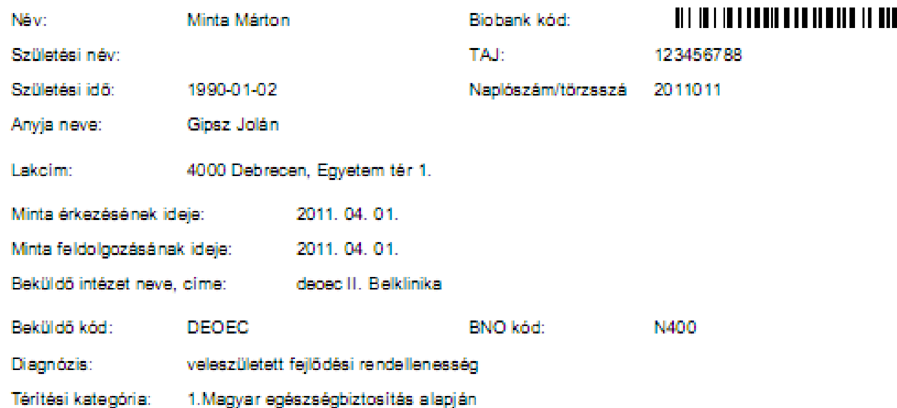
6. Ábrajegyzék

Ábra 1: Biztonsági megerősítés nyomtatás előtt..33
Ábra 2: A nyomtatáshoz használt applet...34
Ábra 3: JQuery dátumválasztó..41
Ábra 4: Automatikus kiegészítés AJAX használatával...42
Ábra 5: Az alkalmazás bejelentkező felülete..44
Ábra 6: A bejelentkező felület CSS nélkül...46

7. Függelék

Egy az alkalmazással készített vizsgálatkérő lap kicsinyített első oldala:

Veleszületett fejlődési rendellenesség vizsgálatkérő lap



Név:	Minta Márton	Biobank kód:	
Születési név:		TAJ:	123456788
Születési idő:	1990-01-02	Naplószám/törzsszá	2011011
Anyja neve:	Gipsz Jolán		
Lakcím:	4000 Debrecen, Egyetem tér 1.		
Minta érkezésének ideje:	2011. 04. 01.		
Minta feldolgozásának ideje:	2011. 04. 01.		
Beküldő intézet neve, címe:	deoec II. Belklinika		
Beküldő kód:	DEOEC	BNO kód:	N400
Diagnózis:	veleszületett fejlődési rendellenesség		
Térítési kategória:	1.Magyar egészségbiztosítás alapján		

Családi anamnézis: Nem ismert

Prenatalis anamnézis: Nem ismert

Betegre vonatkozó Nem ismert

Citogenetikai vizsgálat

Minta: min. 2-3 ml. Heparinnal alvadásgátolt perifériás vér (zöld kupakos cső)

DNS preparálás

Minta: min. 5-10 ml EDTA-val alvadásgátolt perifériás vér (lila kupakos cső)

Tárolt minta vizsgálatának kérésekor a mintavétel 2011. 04. 14.

Fragilis-X szindróma (mindkét vizsgálat elvégzése szükséges)

Citogenetikai vizsgálat

Minta: min. 2-3 ml. Heparinnal alvadásgátolt perifériás vér (zöld kupakos cső)

Molekuláris genetikai vizsgálat (Southern-blot)

Minta: min. 5-10 ml EDTA-val alvadásgátolt perifériás vér (lila kupakos cső)

Megjegyzés:

Vizsgálatot kérő orvos Dr. Kovács János

....

Orvos aláírása,

A nyomtatás időpontja: 2011. 04. 17. 05:07:07 Oldalszám: 1 / 3

Készült a DEBGEN Biobank Szoftverrel

8. Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani Dr. Hajdu Andrásnak és Uzonyi Bélának a dolgozattal kapcsolatos témavezetői teendők nagylelkű elvállalásáért, illetve az elmúlt időszakban kapott támogatásért. Köszönöm Kovács Zoltánnak, és az Astrid Research Kft.-nek, hogy a dolgozat létrehozását lehetővé tették. Emellett szeretném mindenkinek megköszönni a támogatást, illetve segítséget, akik bármilyen formában is hozzájárultak a dolgozat létrehozásához. Köszönöm továbbá az Informatikai Kar valamennyi oktatójának, hogy a dolgozat elkészítéséhez szükséges tudást áldozatos munkájuk révén megszerezhettem, mivel a dolgozat az ő segítségük nélkül nem jöhetett volna létre. Végül, de nem utolsó sorban, szeretnék köszönetet mondani az általam felhasznált technológiák, eszközök, megoldások kidolgozóinak, fejlesztőinek, hogy használhattam az általuk kifejlesztett nagyszerű termékeket.