SZAKDOLGOZAT

Szikora Zsolt

Debrecen
2009

Debreceni Egyetem
Informatikai Kar

Web alapú információs rendszer fejlesztése

Témavezető

Dr. Adamkó Attila
egyetemi tanársegéd

Készítette

Szikora Zsolt
programtervező informatikus Msc

Projekttárs
Záborski László
programtervező informatikus Msc

Debrecen
2009

Tartalomjegyzék

Bevezetés

Diplomamunkám témája: web alapú információs rendszer fejlesztése. Ez a téma több okból is felkeltette az érdeklődésemet.

- Legutóbbi munkahelyemen a követelményelemzés és -tervezés volt feladatom, de sajnos meglehetősen sajátos módon. A kusza érdekviszonyok miatt egy olyan projekthez kellett „követelményt terveznem", amely már az implementációs fázis végén járt. Így aztán mikor kiderült, hogy egy olyan terméket kell előállítanom, mely egy *még pontosításra váró valós megrendelői igényt* próbál kielégíteni, ez a szakdolgozati feladatkiírás rögtön szimpatikussá vált számomra.

- Az egyeztetések során kiderült, hogy a megvalósítandó terméknek nem kell a már meglévő ősrendszerhez kapcsolódnia. Igaz ugyan, hogy egy meglévő rendszerrel együttműködni mindig nagyobb kihívás, mint egy újat létrehozni – most viszont lehetőségem adódott a rendszert az *alapoktól kezdve saját koncepció szerint* kialakítani.

- A feladatot nem egyedül kellett megoldanom. A kitűzött célokat Záborski László[1] csoporttársammal együtt *team-munkában* igyekeztünk megvalósítani. Szakdolgozati munkánk ezáltal még közelebb kerülhetett a valódi szoftvertervezői tevékenységhez. Nagyon örültem, hogy a Debreceni Egyetem is felismerte a projekt munkák jelentőségét. A magányos kungfu harcosként küzdő szoftver guruk ideje leáldozott, a cégek többségének a biztonságos működés miatt jó csapatjátékosokra van szüksége.

- A feladat megoldása során a *Java Vállalati Környezet*[2] szolgáltatásait használtuk. A használt technológia és a hozzá kapcsolódó eszközrendszer ismerete a jövőben is minden bizonnyal hasznomra válik majd.

Vannak vélemények, – pl. [pg2003] – melyek szerint épp hogy non-mainstream technológiákat érdemes megismernünk. Az elit szemlélet tőlem sem áll távol, de sajnos még nem találkoztam olyan alternatív módszertannal, melynek feltétlen lelkes hívévé válhattam

1 http://zabo.hu/
2 Java Enterprise Edition (Java EE)

volna. És persze egy alternatív megközelítés sikerességéhez mindig szükségeltetik egy nagy adag szerencse és üzleti érzék is.

Megrendelői háttér

A webes információs rendszert a *Hajdú-Bihar Megyei Területi Gyermekvédelmi Szakszolgálat*[3] (a továbbiakban: *hbmtgysz*) számára feljesztettük ki.

A megrendelő elsődleges célja: nevelőszülők és a nevelt gyerekek személyes és egyéb adatainak nyilvántartása, az adatok lekérdezhetőségének lehetővé tétele. Jelenleg körülbelül 700 gyerek van elhelyezve kb. 250 nevelőcsaládnál. Ezek szerint nem kell sok adatot kezelnünk. Vigyáznunk kell azonban az elhamarkodott ítéletekkel, nehogy úgy járjunk, mint a mesebeli Bill Gates a [640kB] történetben.

A jelenlegi adatbázisba várhatóan csak a Hajdú-Bihar megyei szolgálathoz tartozók adatai fognak bekerülni, azonban már ezzel szemben is elvárás, hogy az adatok a cég bármely irodájából elérhetők legyenek. A jelenlegi információs rendszer legnagyobb hiányossága éppen az, hogy nem támogatja a konkurens elérést.

Ha sikerülne egy részterület támogatását megvalósítanunk, valószínűleg más megyék szolgálatainak is felkelthetnénk az érdeklődését a termék iránt. Ekkor azonban előbb-utóbb felvetődhet az egyes megyei tegyesz-ek közötti adatcsere eszméje, mely szerencsés esetben találkozhat a minisztériumi szintről időnként elő-előkerülő egységes adatkezelési igénnyel.

Mindezek alapján talán érthető, ha már az első verzió kialakításához is igyekszünk kihasználni a Java EE környezet szabványos eszközeit.

A szakszolgálatoknál jelenleg is használnak egy országosan is elterjedt számítógépes rendszert. A több egymásra épülő – részben opcionális – alkalmazásmodulból álló rendszer neve TEGYESZ. A TEGYESZ a szakterületi folyamatokat rendkívül kiterjedten támogatja.

Ez már az alkalmazáshoz tartozó nyolcvannyolc oldalas [ősTegyesz] felhasználói útmutatóból is kiviláglik. Van azonban vele egy „kis” gond: nem támogatja a hálózaton keresztül történő elérést. Pontosabban ez még megoldható lenne, de a több helyről történő *egyidejű* használatot

3 http://www.hbmtgysz.hu/

sajnos az alkalmazott adatbázis-kezelő rendszer nem támogatja.

Célkitűzés

A projekt célja természetesen elsősorban a megrendelő által elvárt követelményeknek való megfelelés. Másrészt célul tűztük ki, hogy olyan rendszerszerkezet alakítsunk ki, mely nem lesz gátja az esetleges későbbi evolúciónak. A fentiek biztosításához a Java EE menedzselt környezetet választottuk. Igyekeztünk a projektet ingyenesen használható, lehetőleg nyílt forráskódú szoftverek felhasználásával megvalósítani. Célunk volt továbbá a team munkából adódó feladatszervezési teendők meghatározásának és véghezvitelének gyakorlása is.

A projekt munka felosztása úgy történt, hogy a rendszertervezés és kialakítás minden részébe mindketten belekóstolhassunk. A kezdeti lépések megtétele után azonban kiderült, hogy célszerű valamiféle felelősségi kört meghatároznunk egymás – és természetesen ezzel egyidejűleg magunk – számára is.

Végül arra jutottunk, hogy én követelményelemzés és tervezés továbbá a koncepcionális rendszer szerkezet kialakítás mellett leginkább a perzisztálandó objektumokkal és menedzselésükkel bíbelődöm; Záborski László projekttársam pedig az EJB rétegre épülő felhasználói funkcionalitásokkal és az ezeket biztosító felületekkel foglalkozik majd.

Jelenlegi állapot

A projekt végrehajtása során felmerülő nehézségek ellenére sikerült összeállítanunk a teljes rendszertervet, és elkészítenünk a rendszer főbb elemeinek egy szűkített funkcionalitású, de alapvetően működőképes implementációját. Az eddig megvalósított alkalmazás már *minden*, a végső változatban használatos technológiát felhasznál.

A rendszert természetesen az újrafelhasználhatóság és skálázhatóság szem előtt tartásával alakítottuk ki. A munka során megismerkedtünk a Vállalati Környezet több technológiájával, és fel is használtuk ezeket a környezeti szolgáltatásokat. A fejlesztést a szakdolgozat leadásával sem hagyjuk abba, hiszen a korábban írtak szerint nekünk is határozottan érdekünk, hogy ez a projekt minél hamarabb kiteljesedhessen.

A szakdolgozat szerkezete

Szakdolgozatom ezt a bevezetést követően még további öt fő fejezetre tagolódik. Először röviden ismertetem a fejlesztéshez használt technológiákat. Ezt követően bemutatom a kialakított rendszert. A felhasználói jogkörök kezelésével és a rendszerbiztonsággal kapcsolatos ismertetést kiemeltem egy külön főfejezetbe. Ezt követően – a 4 . fejezetben – a rendszerfejlesztés tapasztalatairól írok. Az ötödik fejezet az alkalmazás telepítéséhez szükséges szoftver környezetet, továbbá a telepítés menetét ismerteti.

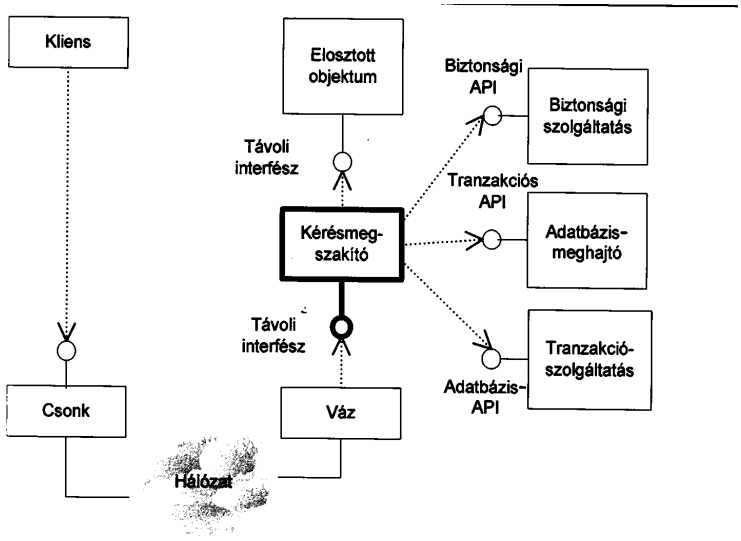
1 Felhasznált technológiák

Az alábbiakban megpróbálom röviden ismertetni a felhasznált technológiák lényegét.

1.1 Java EE, menedzselt környezet

A Java EE egymondatos meghatározása [ig-j2ee] szerint így hangzik: „architektúra *vállalati méretű* alkalmazások fejlesztésére, a Java nyelv és internetes technológiák felhasználásával".

A gyakorlatban a nagy méretű információs rendszerek fejlesztésekor rendszeresen felmerülnek bizonyos ismétlődő követelmények (pl. skálázhatóság, többszálúság, perzisztencia, …). Elvileg ezek bármelyike – vagy akár az összes követelmény is – kielégíthető lenne a több rétegű alkalmazásba integrált natív kódrészletekkel. A gyakorlatban azonban ez nagyon nehezen megvalósítható, már csak a követelmények sokrétűségéből adódó kódméret miatt is. Ezeknek az követelményeknek úgy próbálunk a rendelkezésre álló mindig szűkös fejlesztési időn belül megfelelni, hogy ezeknek a middleware[4] szolgáltatásoknak a megvalósításához elkészített API-kat használunk fel. Ezek az API-k ideális esetben valóban biztosítják is a szabványos funkcionalitás korrekt megvalósítását.



1. kép: Implicit middleware (forrás: [ig-j2ee] 2.1 ábra)

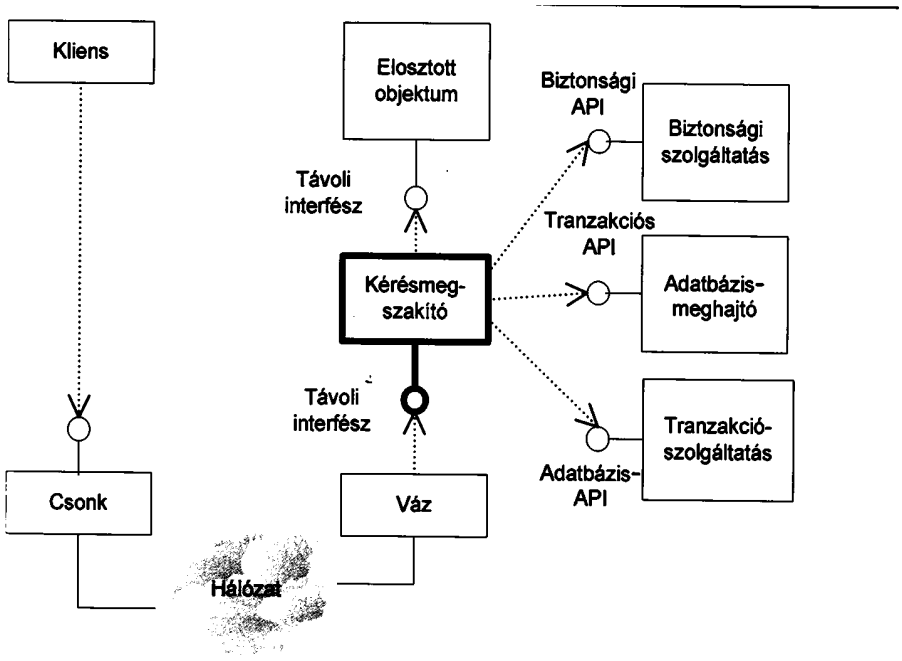
A Java világában ezek az API interfészek publikusak, és ennek köszönhetően több API-

4 Az elnevezés abból adódik, hogy ezeket a szolgáltatásokat általában a többrétegű alkalmazások középső rétegeiben valósítjuk meg.

implementáció közül is választhatunk. Ha a kódba nem helyeztünk el API implementáció specifikus részeket, akkor akár át is állhatunk másik megvalósításra, és programunk működőképességét ez nem fogja befolyásolni.

Egy újabb lépést tehetünk az újrafelhasználhatóság irányába, ha az általunk írt kódba nem helyezünk el explicit middleware API hívásokat. Az explicit API hívásokkal az az egyik gondunk, hogy a forráskód méretét felfújják. A másik probléma, hogy az így kialakított komponens rugalmatlanná válik: ha eladás után a komponens vevője másképp szeretné használni valamelyik hivatkozott API-t, akkor ezt csak a forráskód kiadásával tudjuk számára lehetővé tenni.

Ezt a kellemetlenséget küszöböli ki az explicit middleware használata. Az explicit middleware alkalmazásakor a forráskódban nem hivatkozunk a kért szolgáltatást nyújtó API-ra közvetlen módon. Ilyenkor egy külön leíró fájlban elég csupán deklarálnunk a kért szolgáltatásokat. Így mindenki jól jár. Nekünk a kódban elég csupán az üzleti logikára koncentrálnunk, miközben a komponens felhasználóját is megóvhatjuk a middleware szolgáltatások áthangolásakor egyébként elkerülhetetlen forráskóddal való találkozás okozta sokktól.



2. kép: kép: Explicit middleware (forrás: [ig-j2ee] 2.2 ábra)

Az implicit middleware az elosztott objektumhoz érkező kérés megszakításával valósítható meg. A kérésmegszakítót természetesen nem nekünk kell megírnunk, hanem a szolgáltatásleíró fájl alapján generálódik. Az implicit middleware használatának természetesen feltétele, hogy az alkalmazásunkat egy erre felkészített konténer futtassa. A Java EE szolgáltatásokat használó alkalmazások futtatására képes alkalmazásokat Java Alkalmazáskiszolgálóknak (Java Application Server) nevezzük.

Ezzel kapcsolatban talán érdemes megjegyeznem, hogy a széles körben elterjedt vélekedéssel ellentétben a népszerű Apache Tomcat[5] nem tekinthető teljes értékű Java EE kiszolgálónak. A projekt futtatói soha nem is jelölték ezt ki célként. A Tomcat fő célja Java szervletek és JSP-k futtatása. A félreértés talán abból adódhat, hogy a projekt a kitűzött célt el is éri, ráadásul igencsak versenyképes performancia paraméterekkel, így aztán számos egyszerűbb webalkalmazáshoz választják a Tomcat konténert. Erről a témáról [jw-tomcat] ír bővebben. Ezek után talán már nem is kell bővebben megmagyaráznom, hogy rendszerünkhöz miért választottunk inkább egy teljes értékű szerver implementációt, a GlassFish v2.1-et.

1.2 EJB

Az EnterPrise JavaBean-ek a vállalati alkalmazások alap építőelemei. A jelenlegi EJB 3.0 háromfajta EJB-t definiál.

- A **session bean**ek üzleti folyamatokat reprezentálnak. Metódusaiknak egy-egy használati esetet feleltetünk meg.
- Az **entity bean**ek az üzleti modell entitásait reprezentálják. A entity beanek a perzisztenciát nyújtó adatbázissal tartanak kapcsolatot, példányaik leginkább egy-egy adatbázisbeli rekord memóriabeli cache-ének tekinhetők.
- A **message diven bean**ek az aszinkron üzenetkezeléshez adnak kényelmes megoldást.

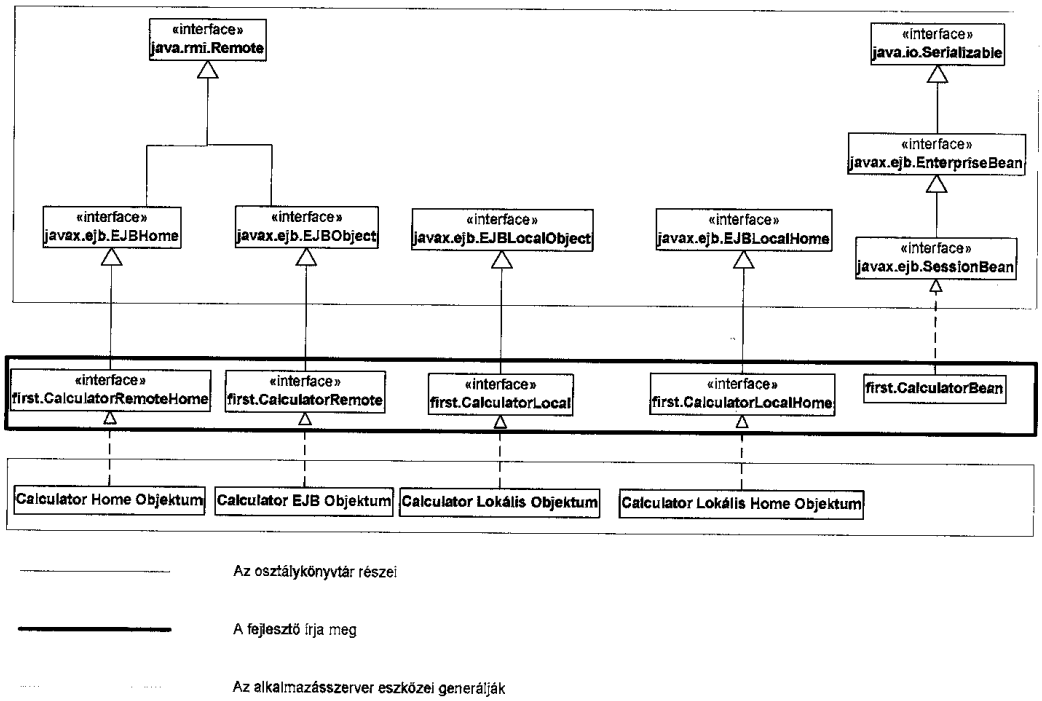
A J2EE 1.4-ben használt EJB 2.1 beanek felépítése és használata meglehetősen sok körültekintést igényel.

Jól szemlélteti a helyzet összetettségét a [ig-j2ee]-ben illusztrációként létrehozott egyszerű

5 http://tomcat.apache.org/

állapotmentes bean osztálydiagramja.

A programozónak „csupán” a vastagon bekeretezett öt fájlt, azaz az implementációs, bean osztályt és ahhoz négy különböző interfészt kell elkészítenie. A felső mezőben lévő interfészekre elegendő csupán hivatkozni a megfelelő fájlokban. A legalsó sor objektumait a konténer generálja. A fent említett öt fájlon túl természetesen el kell még készítenünk a `META.INF/ejb-jar.xml` telepítésleírót is és le kell gyártanunk a szintén elmaradhatatlan gyártóspecifikus fájlokat. Ezt az egészet fordítás után jar-ba csomagolva már létre is jön a kihelyezhető[6] bean.



3. kép: Egy állapotmenetes EJB 2.1 session bean osztálydiagramja (forrás [ig-j2ee] 2.5 ábra)

Ehhez képest az annotációkat és a függőség beszúrást[7] kiterjedten alkalmazó EJB3 kezdetben olyan egyszerűnek tűnt, mint egy álom. Egy idő – és persze sok elkeseredett hibakeresés – után azonban rá kellett jönnöm, hogy a mögöttes folyamatok részletes ismerete nélkül aligha fogok boldogulni. És persze a jövőben előfordulhat, hogy esetleg bele kell nyúlnom egy-egy J2EE alkalmazásba, így aztán a projekt során tettem egy kis kitérőt a J2EE világába is. Egy

6 A *deployment* szó fordításaként a továbbiakban igyekszem a *kihelyezést* használni.
7 dependency injection

idő után vége szakadt a keserves favágásnak, mert felfedeztem, hogy a fejlesztőrendszerek a J2EE technológiát is számos varázslóval és egyéb szolgáltatással támogatják.

1.3 JPA

A Java EE 5 platform részeként az EJB 3.0-val egyidejűleg bevezetett **J**ava **P**erzisztencia **A**PI keretrendszerrel a Sun célja elsősorban az volt, hogy ez az új API egyszerűsítse a perzisztenciát használó Java SE és Java EE alkalmazások fejlesztését. Másrészt szerette volna a teljes Java közösséget egyetlen standard perzisztencia API mögött tudni.[8]

A JPA megpróbálta a szabvány alkotásakor már elérhető különböző perzisztencia szolgáltatók[9], legjobb megoldásait egységes keretek – azaz pontosabban egy szabványos interfész-megvalósítások – közé szorítani úgy, hogy eközben meghagyta a gyártóspecifikus szolgáltatások igénybevételének lehetőségét is.

A JPA egységesíti az objektum/relációs leképezés (ORM: Object/Relational Mapping) megvalósítását. Természetesen lehetővé teszi, hogy a régebbi stílust követve az xml deszkriptorban megadott paraméterek alapján vezéreljük a perzisztálást. Lehetővé teszi ugyanakkor azt is, hogy a nekem is sokkal barátságosabbnak tűnő annotációk használatával lényegében POJO[10] osztályokat perzisztálhassunk. Az annotációk előnye az is, hogy az agilis módszerekkel szinte mindig együtt járó intenzív refactoring mellett sem téveszthetők szem elől, hiszen mindig az érintett osztály- illetve mező-deklarációk környékén maradnak.

1.4 JDBC

A Java DataBase Connectivity a JPA-hoz hasonlóan szintén egy middleware szolgáltatás, csak

8 Forrás: [jpa-faq]

9 Hibernate, TopLink, JDO, ...

10 **P**lain **O**ld **J**ava **O**bject: a hagyományos ős-java-objektumra utal. Egy POJO osztály rendkívül egyszerűen létrehozható.

- mezői privátak
- a mezőket az osztályon kívülről csak a mezőkhöz tartozó, szokásos CamelCase elnevezésű gettereken és szettereken keresztül lehet elérni

(a fenti két követelményt egy egészséges kóder magától is betartja, nem jelent tehát nagy kötöttséget)

- van alapértelmezett – azaz paraméterek nélküli – konstruktora

(ez ahhoz kell, hogy az osztály objektumait a JPA közbeékelődésével, reflectiont használva lehessen példányosítani)

persze alacsonyabb szinten. Remélem, nem voltam félreérthető: az alacsonyabb szó most nem a szolgáltatás színvonalára utal, csupán annyiról van szó, hogy amit a JDBC nyújt, az a szolgáltatás a rendszer-architektúra egy mélyebb rétegében helyezkedik el.

A JBDB API teszi lehetővé, hogy a Java nyelvű programok egy egységes, gyártófüggetlen felületen keresztül kommunikáljanak adatbázisok széles skálájával. Segítségével kapcsolódhatunk SQL adatbázisokhoz vagy más táblázatos adatforrásokhoz is, mint például a táblázatkezelők munkalapjai, vagy akár az egyszerű CSV fájlok. A JDBC technológia használatával valóban közel kerülhetünk a minden programozó álmát jelentő "Write Once, Run Anywhere" megvalósításához. Tudom jól, hogy a megismerhető világnak csak kis részletével kerültem kapcsolatba eddigi életem során, de talán az is jelent valamit, hogy nem jut eszembe egyetlen adatbázis-kezelő rendszer sem, aminek jdbc drivere ne lenne.

1.5 JNDI

A **J**ava **N**aming and **D**irectory **I**nterface is a Java Platform része. Lehetővé teszi, hogy a Java techológiát használó alkalmazások egy egységes interfészen keresztül érjék el a sokféle névtár és címtár (directory) szolgáltatást.

1.6 JTA

A **J**ava **T**ransaction **A**PI egy elosztott tranzakciós rendszerben érintettek (erőforrás menedzser, alkalmazásszerver és a tranzakciót használó alkalmazás) és egy tranzakciós menedzser közötti szabványos interfészeket határozza meg. A JTA-t minden Java EE alkalmazásszervernek implementálnia kell.

A JTA implementálása azért fontos, mert ezzel válik lehetővé az akár több különböző erőforrást, és adatbázist is érintő tranzakciós műveletek biztonságos végrehajtatása.

1.7 Hibernate[11]

A Hibernate valójában ennek a fejezetnek a kakukktojása, hiszen nem egy szabványos Java technológia, hanem a JPA egyik népszerű megvalósítása.

11 https://www.hibernate.org/

Sajnos nem találtam a különböző perzisztencia szolgáltatók elterjedtségéről hitelesnek tűnő adatokat, ám a különböző fórumokon olvasható vélemények alapján és az általam kipróbált többféle *googlefight küzdelem*[12] eredményeképpen is arra jutottam, hogy valószínűleg a Hibernate a legelterjedtebb termék a maga kategóriájában. Ez nem csak az LGPL szerinti licencelésének, de teljesítményének is köszönhető.

Remélem, hogy ezt nem csak a *Hibernate in Action* revideált kiadásaként Christian Bauer és Gavin King által megírt Java Persistence with Hibernate [jp-hbn] mondatja velem. A könyv egyébként csak először tűnt nagyon riasztónak – akkor is csupán vastagsága miatt... Hamarosan kiderült azonban, hogy egy nagyon is jól követhető, élvezetes olvasmány. A könyvben említett `.jar`-ok természetesen már nem használhatók, és sajnos a leírt `ant` scripteket is módosítanom kellet a hibernate jelenlegi verziójával való együttműködéshez, de így legalább a saját bőrömön is megtapasztalhattam, hogy két év milyen nagy idő az „O/R Mapping business”-ben.

A műben szerepelt néhány nagyon kifejező ábra, ami jól szemlélteti az eddig említett technológiák egymással való együttműködését. A bemutatandó három ábra mindegyike az adatbázis és a Hibernate közötti kapcsolatkezelésről szól.

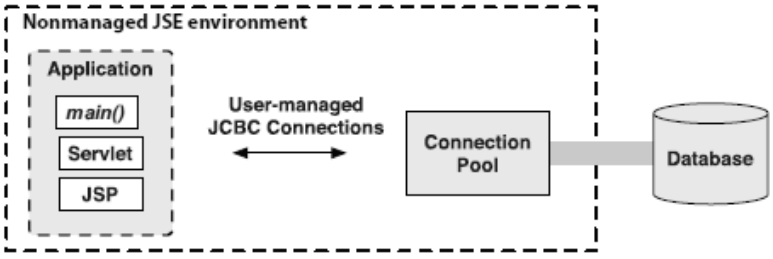


Figure 2.2 JDBC connection pooling in a nonmanaged environment

4. kép: JDBC kapcsolat pooling nemmenedzselt környezetben [jp-hbn]

Az 4. képen egy „majdnem” közvetlen adatbázis kapcsolatot kezelő alkalmazás látható. Azért csak majdnem, mert általában nem tanácsos minden esetben egy új kapcsolatot létrehozni, ha interakcióba szeretnénk lépni egy adatbázis szerverrel. A kapcsolatok kezeléshez a gyakorlatban inkább a kapcsolat poolingot szokás használni, hiszen egy új kapcsolat igénylése

12 pl. http://googlefight.com/index.php?word1=toplink+persistence&word2=hibernate+persistence

több okból is költséges lehet. Néhány DBMS akár egy teljesen új szerver oldali processz indítását is kezdeményezi ilyenkor. Ráadásul a tétlen kapcsolatok fenntartásának is vannak költségei. Egy pool menedzser gondoskodhat a már meglévő kapcsolatok optimális kihasználásáról, vagy akár le is csatlakozhat, ha már nincs szükség az összes meglévő kapcsolatra. A poolozás további előnye még, hogy a néhány jdbc driver számára szintén nagy költséggel létrehozható *előkészített kijelentéseket*[13] is megcache-elheti a valószínűleg hamarosan megismétlődő kérések közötti időszakban. Ilyen Poolinghoz hazsnálható pl. a C3P0 ingyenes és szabad kapcsolat-pool rendszer. Az alkalmazás tehát a pool-tól igényel JDBC kapcsolatot, majd végrehajtja az SQL kijelentéseket. A művelet befejeztével a kapcsolatot az alkalmazás bontja, ám az valójában nem bomlik le, csak visszakerül a pool menedzserhez.

Ha nem menedzselt környezetben alkalmazunk valamilyen perzisztencia szolgáltatást is, akkor az 5. ábrán látható rendszert kapjuk.

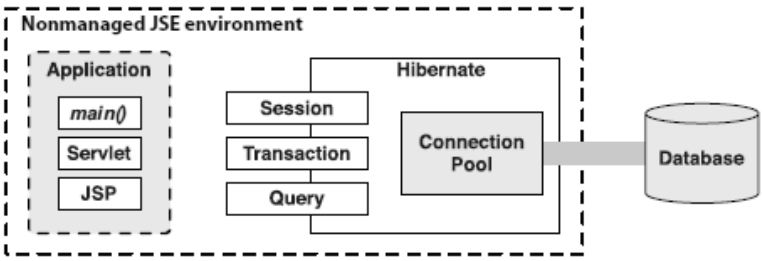


Figure 2.3 Hibernate with a connection pool in a nonmanaged environment

5. kép: Hibernate kapcsolat poollal nemmenedzselt környezetben[jp-hbn]

A kapcsolat pool szolgáltatásait az alakalmazás most már nem közvetlenül, hanem a JPA provider közvetítésével veszi igénybe. A pool hangolását ekkor a `hibernate.cfg.xml` vagy a `hibernate.properties` fájlok felhasználásával végezhetjük el. Az alkalmazás a Hibernate-hez fordulhat natív Hibernate API-n, vagy a szabványos JPA-n keresztül is kérésekkel. A JPA használata előnyös lehet, ha nem akarunk egy adott perzisztencia szolgáltatóhoz hozzáragadni. A natív Hibernate API használata akkor válik igazán igazán kifizetődővé, ha már kezdjük érezni a JPA lehetőségeinek korlátait.

13 A *prepared statement*-et próbálom meg így tolmácsolni magyar nyelven.

A harmadik – a szakdolgozati projektben általunk is választott – forgatókönyv szerinti perzisztálás menedzselt környezetben történik. Ilynekor rendszerint a *connection pool* kezelését is az alkalmazásszerver erőforrás-kezelője végzi. A perzisztencia szolgáltató egy JNDI névre való hivatkozással igényelheti az adatbázis kapcsolatot.

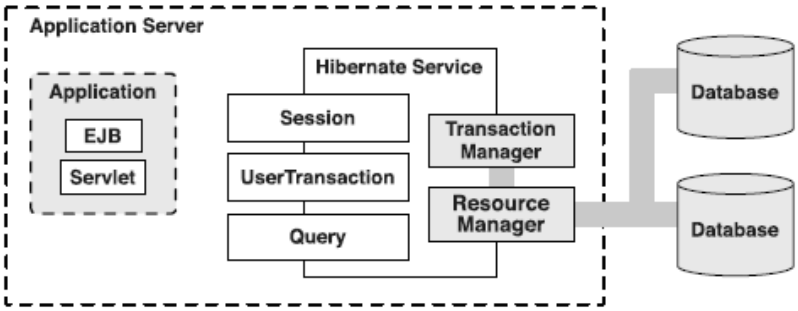


Figure 2.6 Hibernate in an environment with managed resources

6. kép: Hibernate menedzselt erőforrásokkel ellétott környezetben[jp-hbn]

Mindez persze megoldható alkalmazásszerver nélkül is. Léteznek önálló JTA providerek is a feladat megoldásához, és persze megfelelő kiegészítésekkel és kellő hangolással[14] akár a korábban említett Apache Tomcat is rávehető, hogy ilyen funkcionalitást nyújtson.

1.8 JSF

A **J**ava **S**erver **F**aces technológia egy szerver-oldali interfész komponens a Java alapú webalkalmazások felhasználói felületének kialakításához. A JSF főbb összetevői az alábbiak.

- Egy kifinomultan hangolható működésű API, mely nagy segítséget ad...
  - a felhasználói felület komponenseinek és azok állapotának reprezentálásához
  - az eseménykezeléshez
  - a szerver oldali validációhoz és az
  - adatkonverzióhoz,
  - az oldalak közötti navigációhoz
  - a tartalmak nemzetköziesítéséhez (I18N, L9N).
- A felhasználói felület komponenseinek kifejezéséhez és az egyes elemek szerver-oldali

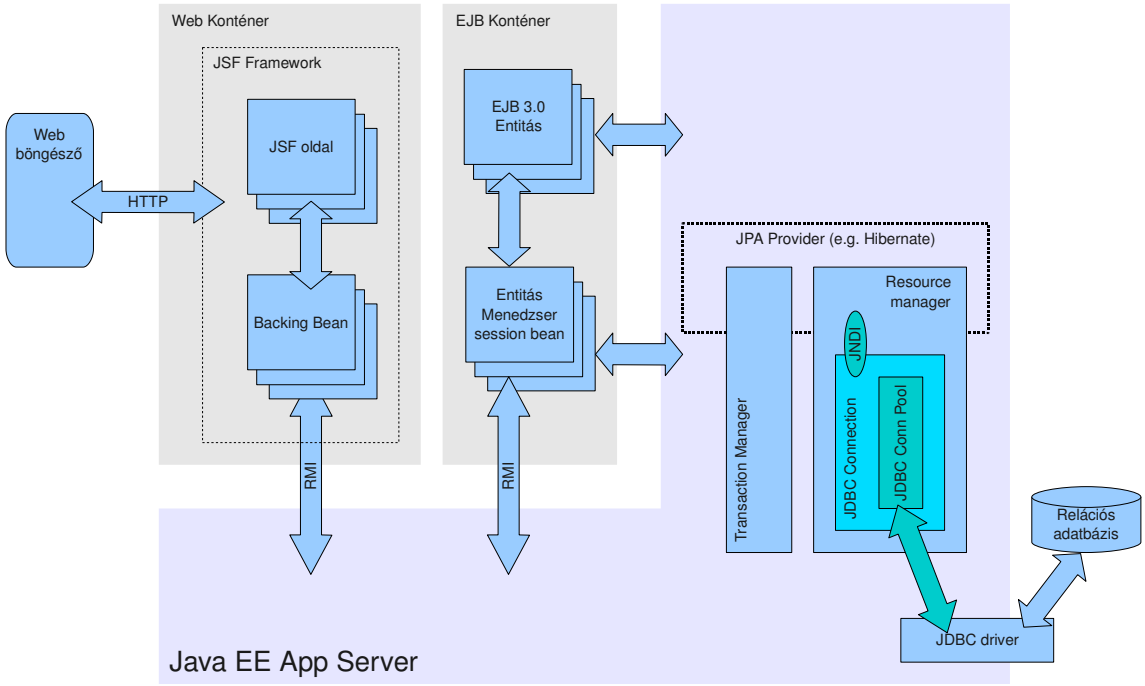
14 A dolog részleteiről lásd pl. [jp-hbn] 98. oldalának megjegyzéseit

objektumokhoz (menedzselt beanekhez) történő kapcsolódásához rendelkezésünkre bocsát továbbá két JavaServerPages (JSP) testreszabott címkekönyvtárt (azaz „magyarul” custom tag library-t)

A jól meghatározott programozási modell és a címkekönyvtárak nagy mértékben könnyítik a szerver oldali webalkalmazások készítőinek és karbantartóinak vállaira nehezedő terheket.

2 Alkalmazás szerkezet

1. A fejlesztés során végül az 7. ábrán szemléltetett próbált alkalmazásszerkezetet alakítottuk ki. Igyekeztünk az MVC mintát követni. A JSF keretrendszer `.jsp` oldalakból generálja a *View*-t Backing bean-ek felhasználásával. A *Controller* szerepét is betöltő backing beanek vezérlik a `faces-config.xml` által behangolt működésű `FacesServlet` nézetváltásait, és az entitásmenedzser beanek távoli interfészein keresztül irányítják a session beanek által kezelt entitásokat, melyek a *Modellt* alkotják.



7. kép: A kialakított alkalmazásszerkezet

A kialakított alkalmazásszerkezet előnyei

- Az alkalmazás JPA szolgáltatója egyszerűen lecserélhető, mivel a kódoláskor

elkerültem Hibernate-specifikus szolgáltatásokra való hivatkozásokat.

- A tranzakciókezelést az alkalmazásszerver JTA implementációjára bíztuk – ez által lehetőség lesz a későbbiekben akár arra is, hogy elosztott tranzakciókat kezeljünk.
- A perzisztencia réteg JNDI közvetítésével találja meg a hozzá tartozó adatbázis kapcsolatot, így annak elérhetősége rugalmasan szabályozható.
- Viszonylag könnyen átállhatunk másik adatbázis termékre. Ehhez csupán a megfelelő néven be kell regisztrálnunk a szerveren a megfelelő adatforrást, és természetesen át kell állítanunk a JPA provider SQL dialektusát is.
- Az JSF oldalon található menedzselt (backing) beanek az entitás menedzser session beaneket azok távoli interfészén keresztül érik el, így a szoftver módosítása nélkül az is megoldható, hogy a két oldalt két külön számítógépekre telepítsük. A terheléselosztást még jobbá tehetjük, ha magát az adatbázist is egy külön hoszton helyezzük el.

2.1 Adat réteg

Ami a konkrét adatréteget illeti, a projekt fejlesztése során én PostgreSQL-t használtam, projekttársam pedig MySQL-t.

Itt jegyzem meg, hogy az adatréteg kezelését valójában teljes egészében a JPA providerre bíztam. Magát az adatbázis sémát is a JPA szolgáltató generálja. Ehhez a `persistence.xml`-ben található perzisztenciaegység[15] leírásba kellett felvennem a megfelelő gyártóspecifikus propertyt:

```
<properties>
  <property name="hibernate.hbm2ddl.auto" value="update"/>
</properties>
```

bejegyzésnek köszönhetően a Hibernate minden egyes kihelyezéskor ellenőrzi az adatbázisban rendelkezésre álló adatbázis sémát, és ha nem találja benne a szükséges elemeket, akkor automatikusan legenerálja az ezek létrehozásához szükséges DDL utasításokat és azokat végre

15 A perzisztenciaegység állítja be mindazoknak az alkalmazás által egymáshoz vagy egy csoportba tartozónak ítélt osztályoknak a körét. Az egy perzisztenciaegységhez tartozó osztályok példányait a JPA mindig egyetlen adatbázisban helyezi el.

is hajtja.

2.2 Adat elérési réteg

Az adatbázis adatait a Java EE menedzselt környezet JDBC-n keresztül éri el.

2.3 Entitás réteg

Az entitás réteget a megfelelően annotált EJB 3.0 entitások alkotják. Példaképpen szeretném bemutatni a kategória-szerű törzsadatok szuperosztályaként létrehozott `Category` osztály egy részleltét. Azért pont ezzel az osztállyal szemléltetek, mert nagyon a szívemhez nőtt. Meglehetősen sokat küzdöttem vele, mire sikerült így kialakítanom. A dolog részleteiről bővebben a 4.4.3 fejezetben írok.

```
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS) // Hibernate *can* handle this
strategy :)
// using default table name
public abstract class Category implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    @Column(name = "thevalue", length = 64, unique=true, nullable = false) // value is a SQL-
99 keyword
    private String value;

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (id != null ? id.hashCode() : 0);
        return hash;
    }

    @Override
    public boolean equals(Object object) {
        // TODO: Warning - this method won't work in the case the id fields are not set
        if (!(object instanceof Category)) {
            return false;
        }
        Category other = (Category) object;
        if ((this.id == null && other.id != null) || (this.id != null &&
this.id.equals(other.id))) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return this.getClass().getCanonicalName() + "[id=" + id + ", value=" + value +
                "]";
    }
```

A használt annotációk `@RetentionPolicy`-je `RUNTIME`, ami lehetővé teszi, hogy a reflexiós API

segítségével a JPA provider megvizsgálja, meg van-e jelölve az adott annotációval egy mező, metódus vagy osztály.

A JPA hangolását a `persistence.xml` segítségével végezhetjük el. Ebben kell deklarálnunk a perzisztencia egységet, a használt tranzakció kezelőt és az adatforrást is. Itt adhatók meg a korábban említett provider-specifikus beállítások is.

```
<persistence-unit name="Tegyesz-ejbPU" transaction-type="JTA">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <jta-data-source>jdbc/tegyesz</jta-data-source>
</persistence-unit>
```

2.4 Session bean réteg

Az EJB 3.0 lehetővé teszi, hogy megfelelően annotált POJO osztályokat használjunk.

A session beanek az `EntityManager` interfészen keresztül férhetnek hozzá a perzisztencia kontextus szolgáltatásaihoz. Az `EntityManager` példányhoz mindig tartozik egy perzisztencia kontextus. Ez a perzisztencia kontextus az entitáspéldányok olyan halmaza melyben minden egyes perzisztens identitáshoz tartozik egy egyedi entitás példány.

A perzisztencia kontextuson belül az egyes entitáspéldányok és életciklusuk menedzselt.

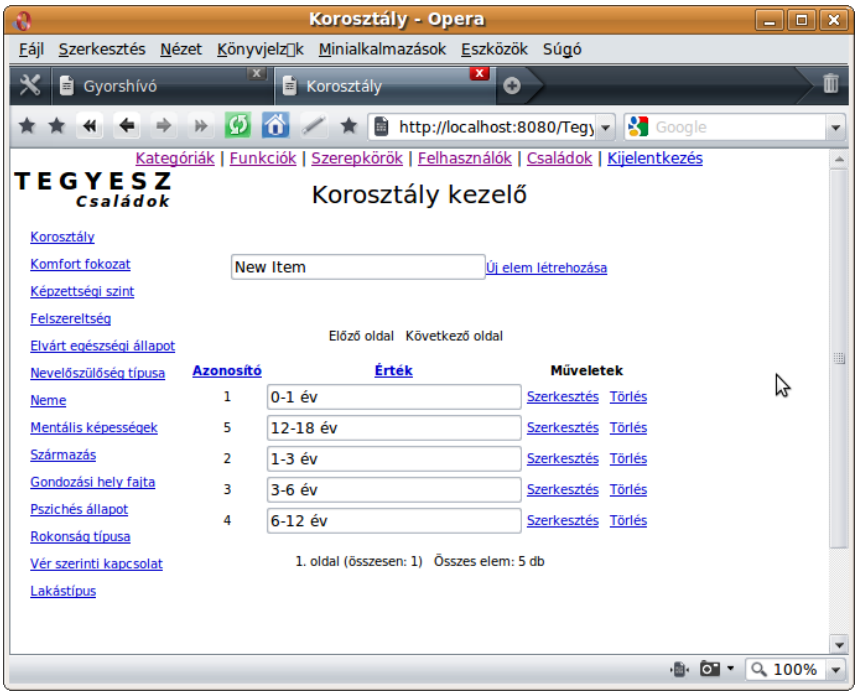
```
@Stateless
public class FunctionManager implements FunctionManagerRemote {

    @PersistenceContext
    private EntityManager em;

    public void create(Function function) {
        em.persist(function);
    }
```

A fenti forráskód részlet talán segíthet még világosabbá tenni az eddig leírtakat.

2.5 JSF réteg

A JSF réteg megvalósítása elvileg nem az én feladatom lett volna ebben a projektben, mégis meg kellett ismerkednem a technológiával. A technológia alapelveinek ismerete nélkül egyrészt nem lett volna esélyem rá, hogy részt vegyek a hitelesítéssel és biztonsággal kapcsolatos, az egész rendszer működését érintő koncepciók kidolgozásában. Másrészt nem tudtam volna kipróbálni, tesztelni az EJB réteg szolgáltatásait. Harmadrészt enélkül nem tudtam volna segíteni projekttársamat a JSP réteg implementációja közben adódó számos gond leküzdésében. A JSF réteg az ő munkájának gyümölcse, de reményeim szerint talán

nekem is volt néhány hasznos ötletem.



8. kép: Korosztályok kezelése

Erről a rétegről tehát Záborski László ír majd bővebben szakdolgozatában, ami engem illet én csupán néhány ehhez kapcsolódó élményemről fogok majd szólni a 4.5 fejezetben.

2.6 Backing bean-ek

Az egyes JSF oldalakhoz tartozó backing bean-ek szerepe a JSF laphoz tartozó nézet elemeinek tartalmi kezelése, továbbá az akcióelemek által kezdeményezett vezérlések megvalósítása a `faces-comfig.xml`-ben meghatározott lap-közlekedési szabályok szerint.

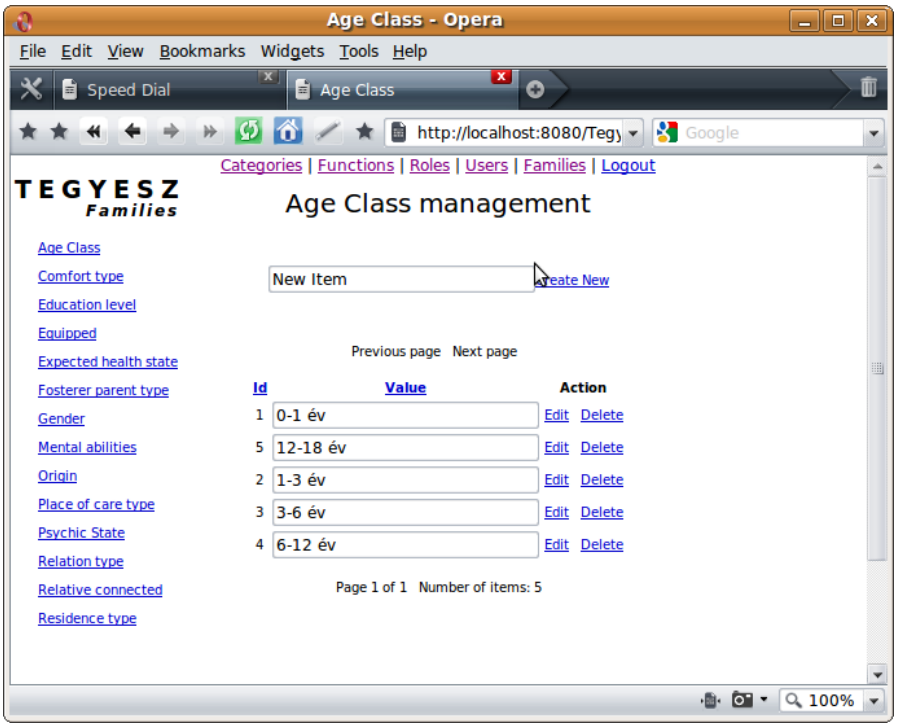
A projekt tartalmaz egy Bean-t a hitelesítés megvalósításához is. Többet azonban most nem szólok erről sem, nem szeretném elvenni projekttársam kenyerét.

2.7 L9N[16] réteg

A weblapon megjelenő üzenetek mindegyike lokalizált. Ez az alkalmazás a

16 L9N az angol nyelvterületen régóta előszeretettel alkalmazott rövidítési módszer a viszonylag hosszú szavak tömör megadására. A középső számérték a LocalizatioN szó középső betűinek darabszámára utal. Ehhez hasonló, épp a témába vágó rövidítés az i18n = InternationalizatioN. Egy másik, a témához kevésbé tartozó ilyen rövidítés: s9y = SerendipitY. Ez utóbbi egyébként az általam is használt blog-motor neve is egyben.

követelményrendszer sajátosságaiból adódóan várhatóan soha nem jelenik majd meg nemzetközi környezetben. Mégis amellett döntöttünk, hogy alkalmazni fogjuk a JSF kínálta L9N lehetőségeket.



9. kép: A korosztálykezelés oldal a böngésző alapértelmezett nyelvének angolra állításátkövetően

Projekttársam az angol és magyar nyelvhez tartozó `.properties` fájlokat készítette el. Sikerült elérnie, hogy még a validátoroktól érkező, az L9N által közvetlenül nem támogatott `FacesMesage`-ek is lokalizáltan jelenjenek meg.

2.8 Web kliens réteg

A JSF lapok `<div>`-szerkezetét és a hozzájuk tartozó `.css`-eket csoporttársam úgy készítette el, hogy minden jelenleg elterjedt böngészőben hasonlóan jelenjenek meg. Az alkalmazás tehát tetszőleges böngészővel használható.

3 Biztonság

A biztonság kezelését a Java EE konténer a JAAS segítségével támogatja. Az [ig-j2ee] forrásban olvasott 9. fejezet alapján azonban világossá vált számunkra, hogy az ott leírt rendszer nem hangolható elég rugalmasan.

3.1 Szerepkör alapú elérés kezelési módszer

A szerepkör alapú hozzáférés szabályozás lényege, hogy minden egyes felhasználó bizonyos szerepköröket kap. Minden egyes szerepkörhöz hozzárendelhetők bizonyos rendszerfunkciók. A felhasználók csak a szerepköreikhez tartozó rendszerfunkciókat érhetik el.

A fő rendszerfunkciók azonosításához a funkcióhoz tartozó JSP lappal társított Bean nevét használjuk. Az adott lapokról elérhető, és a laphoz társított `PageBean` osztály üzleti folyamatokat indító `Action` metódusainak neve azonosítja a fő funkción belüli alfunkciókat.

A felhasználói szerepkörök és rendszerfunkciók, valamint ezek alapértelmezett egymáshoz rendelését a 8.4 melléklet tartalmazza.

A felhasználói felület JSF komponenseinek `rendered` tulajdonságét felhasználva már eleve meg sem jelenítjük a felhasználó számára nem látható üzleti tartalmakat.

Ha esetleg – a felület alkotójának figyelmetlensége vagy bármi egyéb huncutság folytán – a felhasználónak mégis sikerülne egy `Action` metódust illetéktelenül elindítania, akkor azt nem tudja felhasználni, mert minden üzleti metódus meghívja a backing beanen belül az alábbi függvényt.

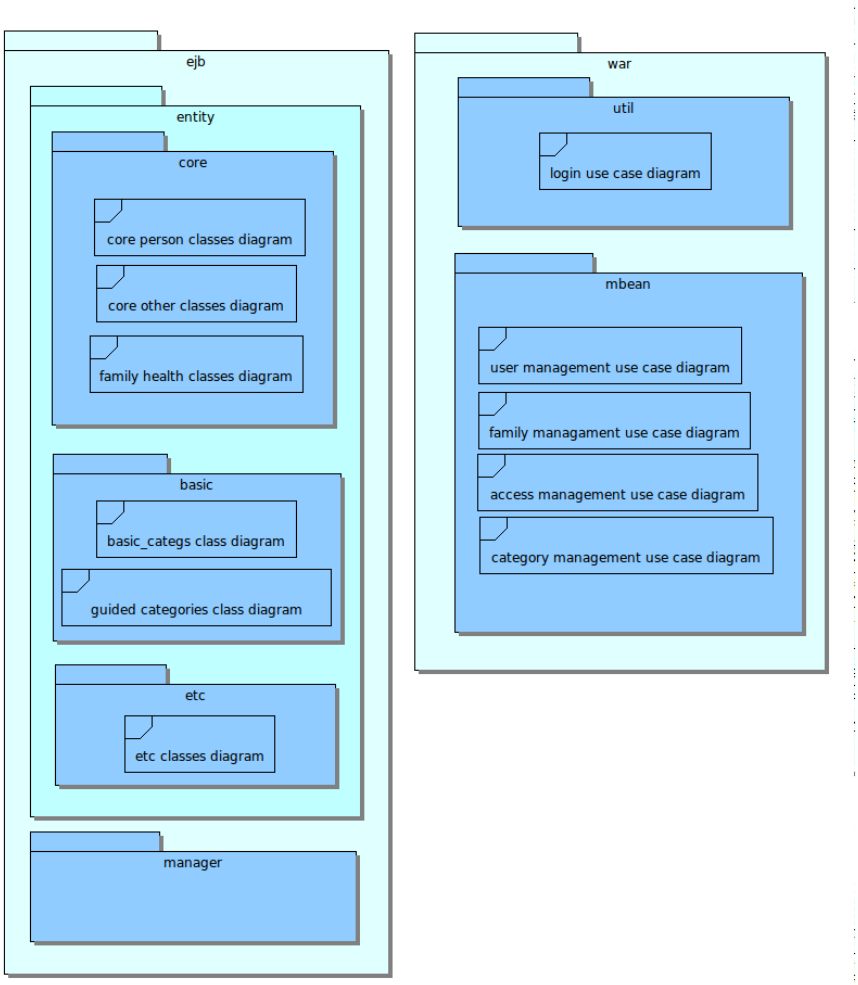
```
protected boolean isAllowed() {
        return Security.getCheckFunction(this.getClass().getSimpleName() + "." + new
Exception().getStackTrace()[1].getMethodName());
    }
```

Ez a függvény meghatározza saját osztályának egyszerű nevét, és ehhez hozzáfűzi a '.' szeparátort követően az őt hívó metódus nevét. Ezzel, mint paraméterrel hívja meg a tényleges ellenőrzést végző `public static boolean getCheckFunction(String funcName)`-et. Nekem különösen tetszik ebben, ahogy egy létrehozott, de el nem dobott kivételhez tartozó `StackTrace` segítségével meghatározza a hívó metódus nevét.

Ezt a szerintem rendkívül ötletes megoldást már nem tudom, pontosan hol láttam, csak arra emlékszem, hogy – az egyébként is rendkívül figyelemreméltó – [mindprod]-ról kiindulva jutottam el a forráshoz.

3.2 Csomagdiagram

A projekthez tartozó kódot a 10. ábrán ismertetett csomagokba szerveztük. Az ábrán nem szerepel ugyan, de természetesen minden egyes csomagot az alkalmazás `tegyesz` csomagjában helyeztünk el. A `tegyesz.ejb` csomagba kerültek a `Tegyesz` Java EE projekt `Tegyesz-ejb` alprojektjéhez tartozó elemek, a `tegyesz.war` pedig értelemszerűen az Enterprise projekhez tartozó másik, `Tegyesz-war` alprojekt elemeit tartalmazza.

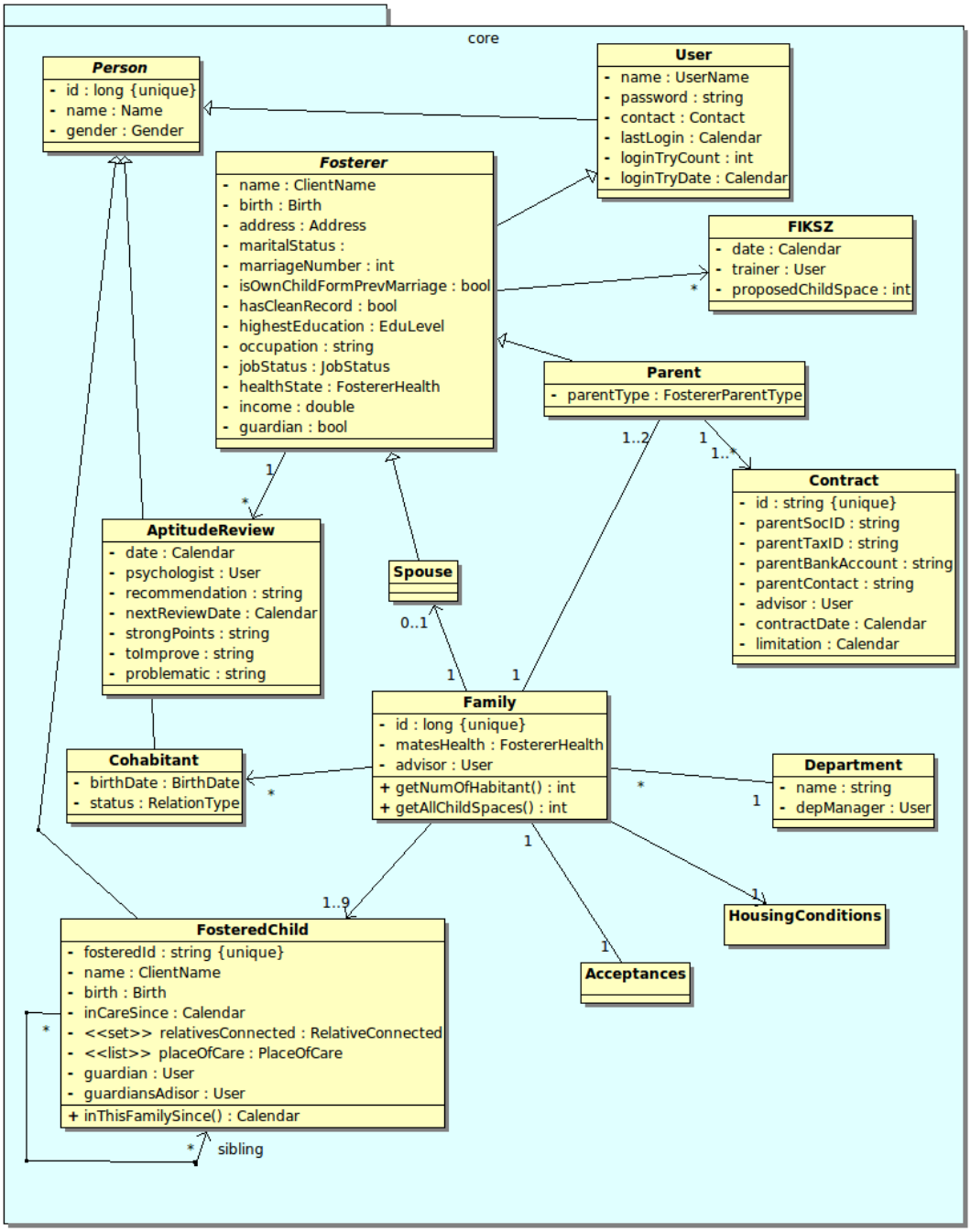


10. kép: Csomagdiagram

Az entitásosztályok a `tegyesz.ejb.entity` alcsomagjaiba kerültek, az őket menedzselő session beanek pedig a `tegyesz.ejb.manager`-be. A `war.mbean` csomag tartalmazza a JSF oldalakhoz tartozó menedzselt beanek implementációs osztályait, a `war.util`-ba pedig a biztonságkezelést és a lokalizációt kezelő segédosztályok kerültek.

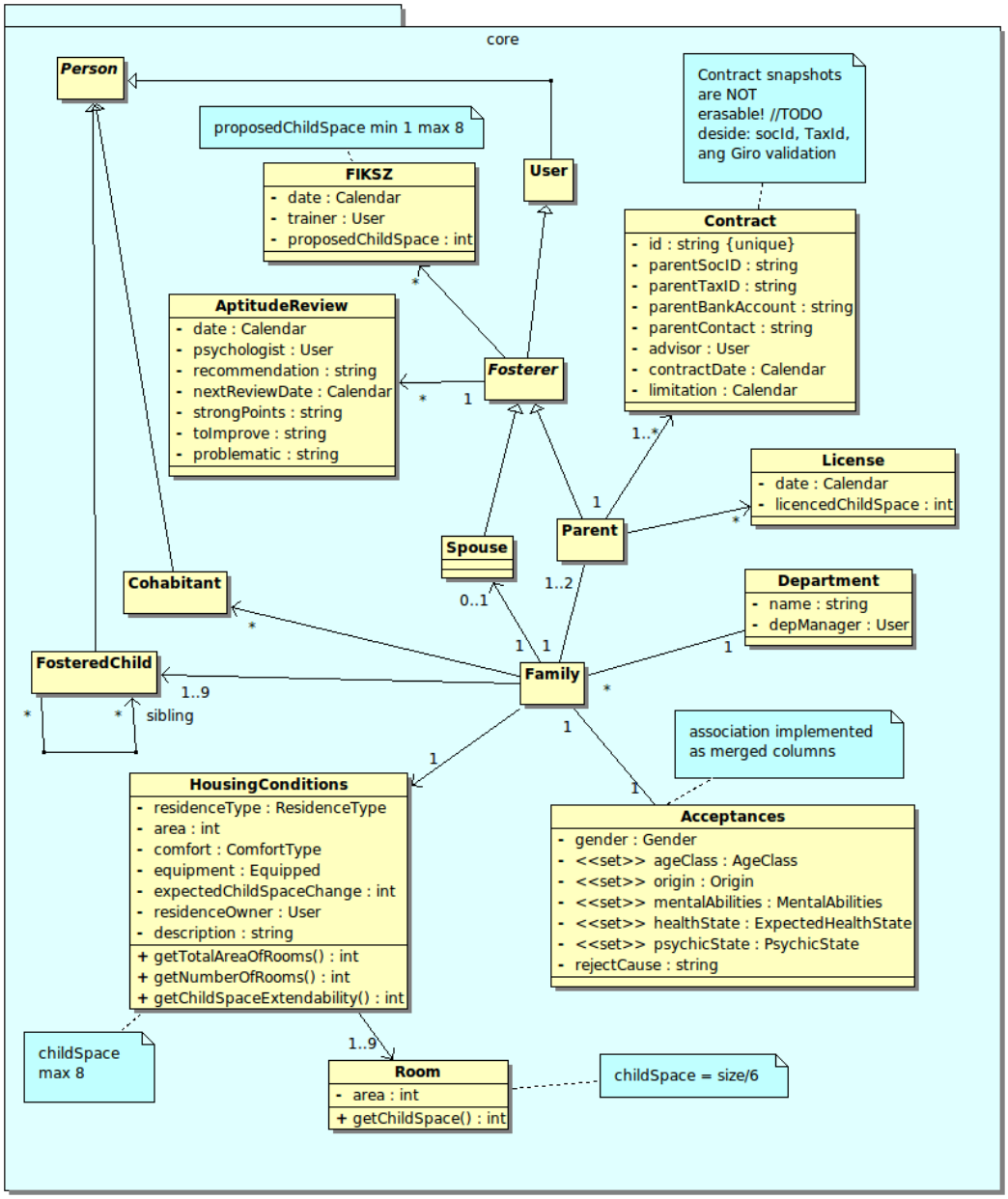
3.3 Osztálydiagramok

Az EJB projekt entitásainak kialakításában nagy segítségünkre voltak az osztálydiagramok. A következőkben a kialakított osztályszerkezetet mutatom be az UML eszközrendszerének felhasználásával.



11. kép: A személyeket reprezentáló központi osztályok

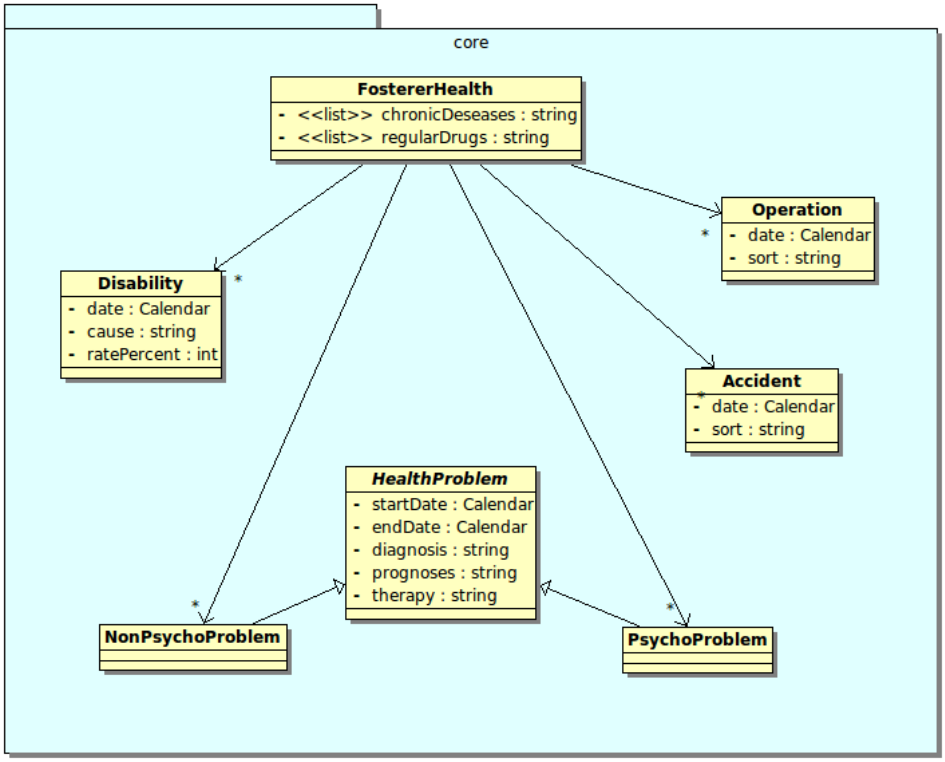
A `core` csomagba kerülő osztályok az alkalmazás magját reprezentálják. Az osztályok egymáshoz kapcsolódását sajnos nem tudom egyetlen ábrán szemléltetni, mert a megrendelő igényekből adódóan számos osztálytulajdonságot fel kellett vennünk. Az alkalmazás központi osztályainak egy olyan nézetét tartalmazza a 11. ábra, melyen a személyekhez tartozó osztályok mezői vannak kiemelve.



12. kép: További központi osztályok

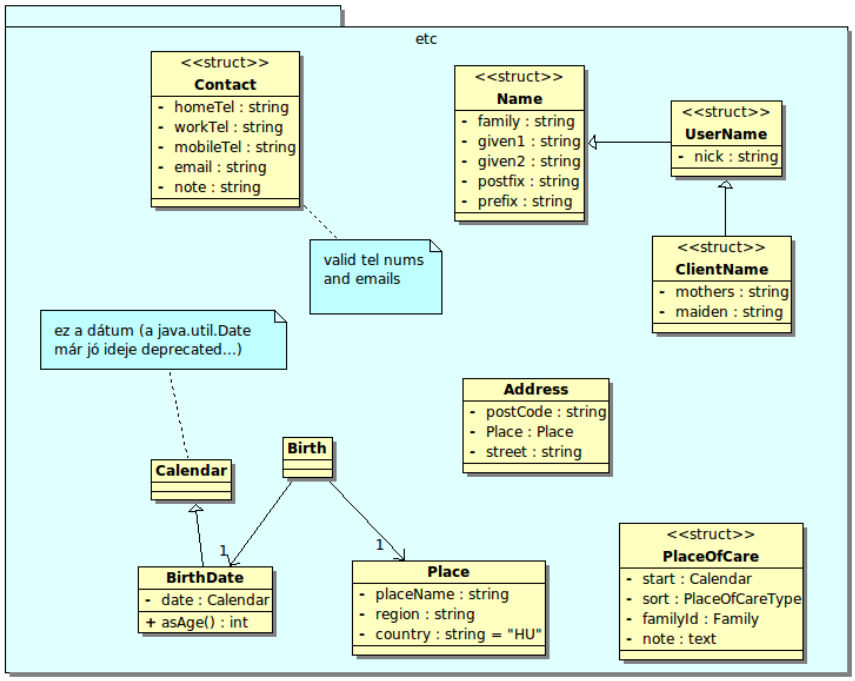
A személyekhez kapcsolódó további osztályokat szemlélteti a 12. ábra oly módon, hogy a

korábban kirészletezett személy-jellegzetességek helyett a kapcsolódó egyéb osztályok mezőire és érdekesebb metódusaira koncentrál. Külön diagramon részletezem a nevelőszülők egészségével kapcsolatos, még mindig a központi osztályokat tartalmazó csomagban implementált osztályokat.



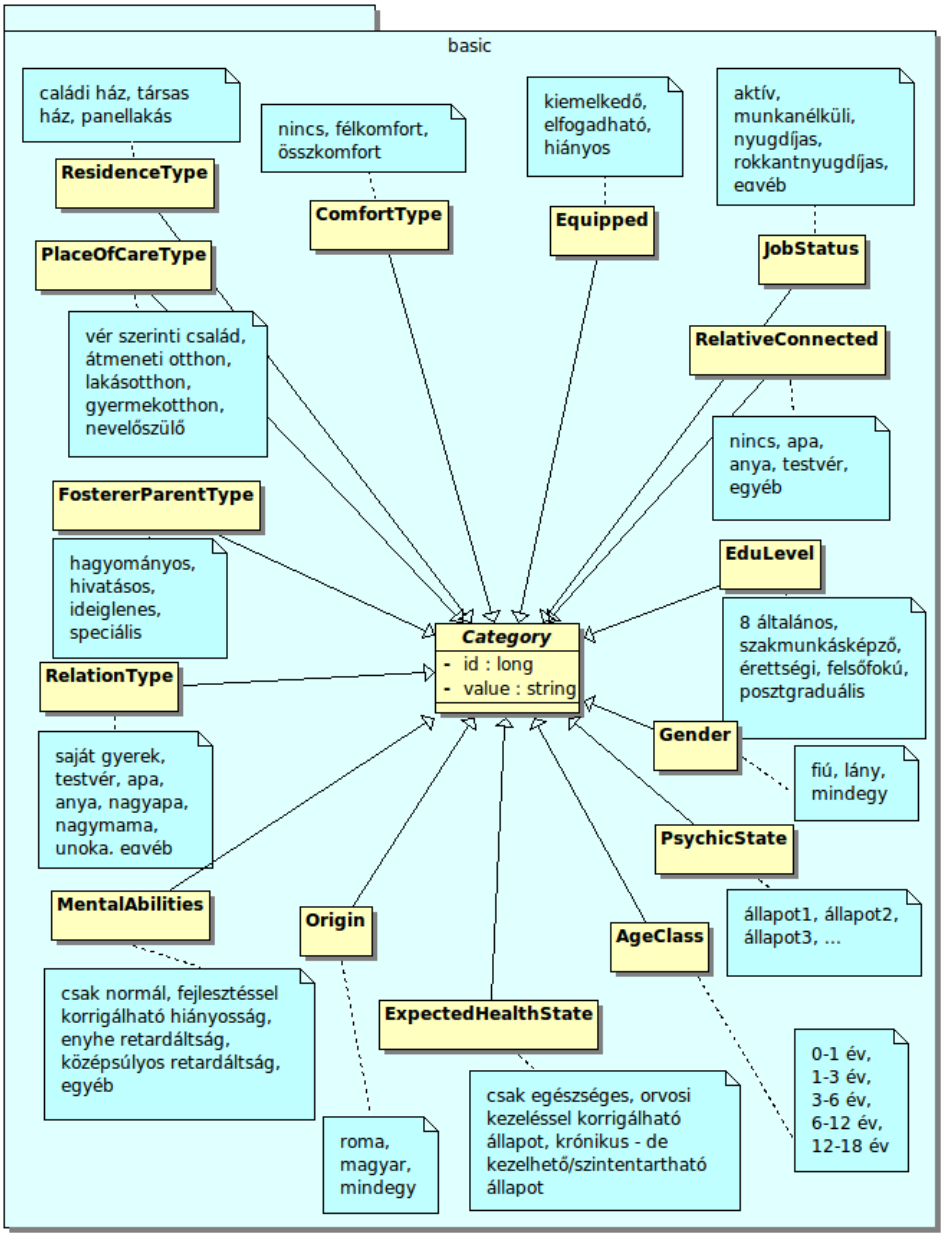
13. kép: Egészségügyi vonatkozású osztályok

Az etc csomagba kerültek az alaposztályokat kiegészítő további osztályok.



14. kép: Segédosztályok

Az alkalmazás törzsadatait alkotják a különféle kategóriafajták.

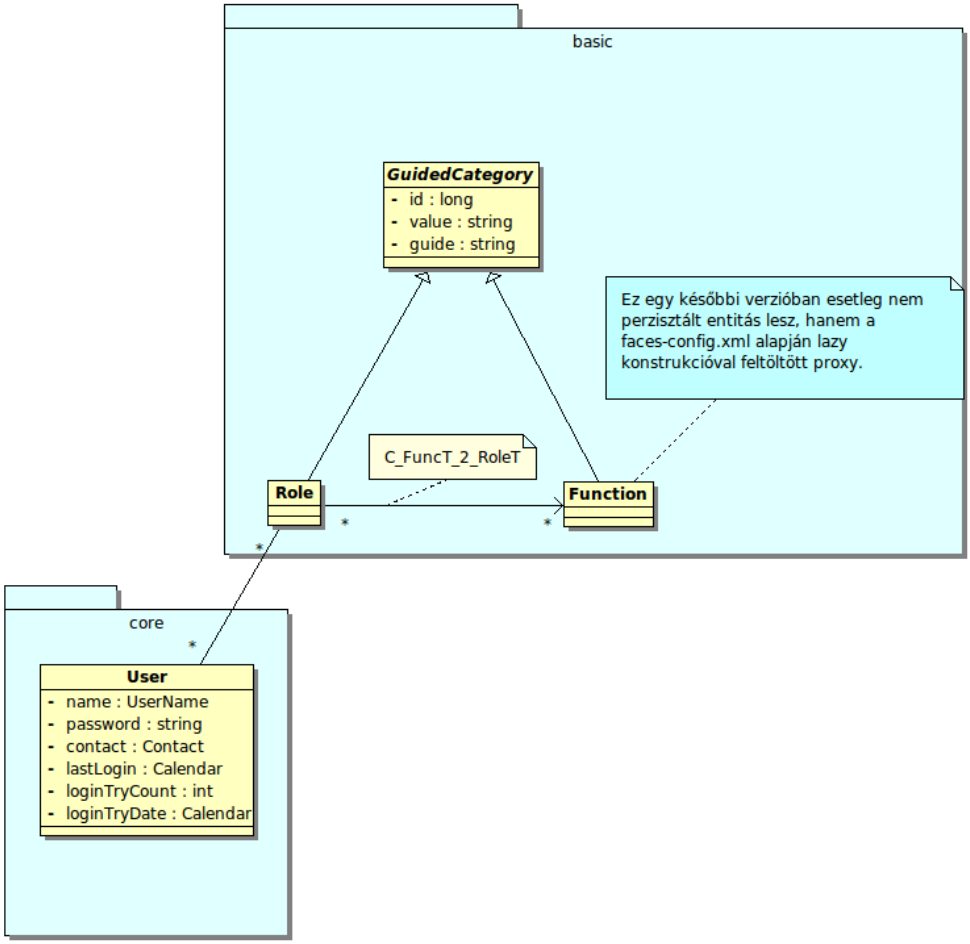


15. kép: Kategóriaosztály és leszármazottai

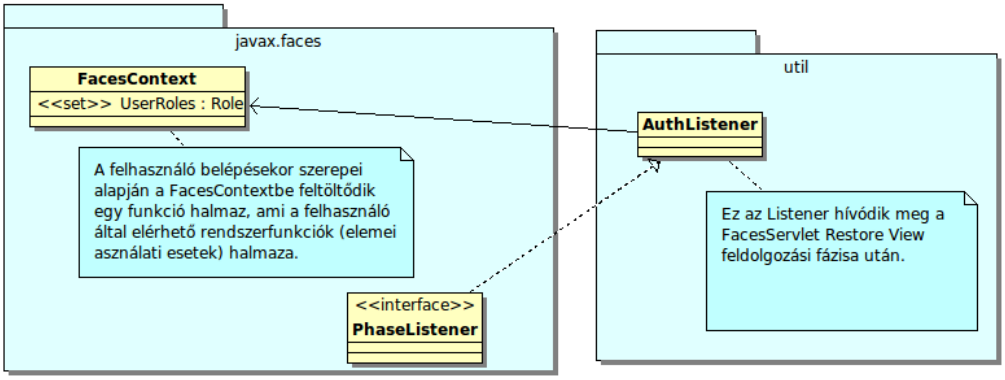
Az eddigi osztályok létrehozására a felhasználók által megfogalmazott alapvető funkcionális követelmények kielégítése miatt volt szükség. A felhasználók rendszerhez kapcsolódását szabályozó szerepkörök és rendszerfunkciók kezelése, valamint az elérés szabályozása miatt szükséges volt további osztályok bevezetése is. Ezeket az osztályokat ismertetem a most következő két ábrán.

A 16. ábrán a `tegyesz.ejb.basic` csomagba elhelyezett Role és Function osztályokat szemlélteti,

a rákövetkező pedig `a tegyesz.war.util` csomagban elhelyezkedő, eredeti terveink szerint a JSF hat állapotú kérésfeldolgozási mechanizmusába beépülő `AuthListener`-t mutatja be.



16. kép: Szerepkörök és rendszerfunkciók

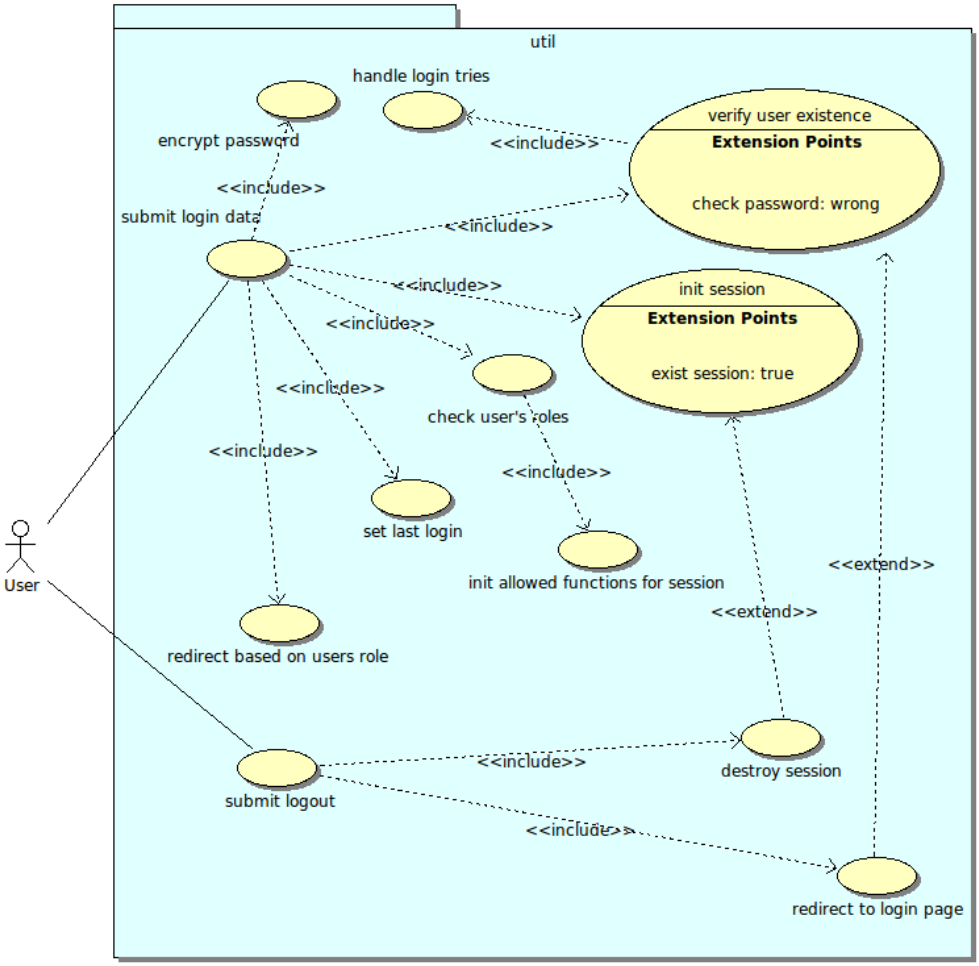


17. kép: AuthListener

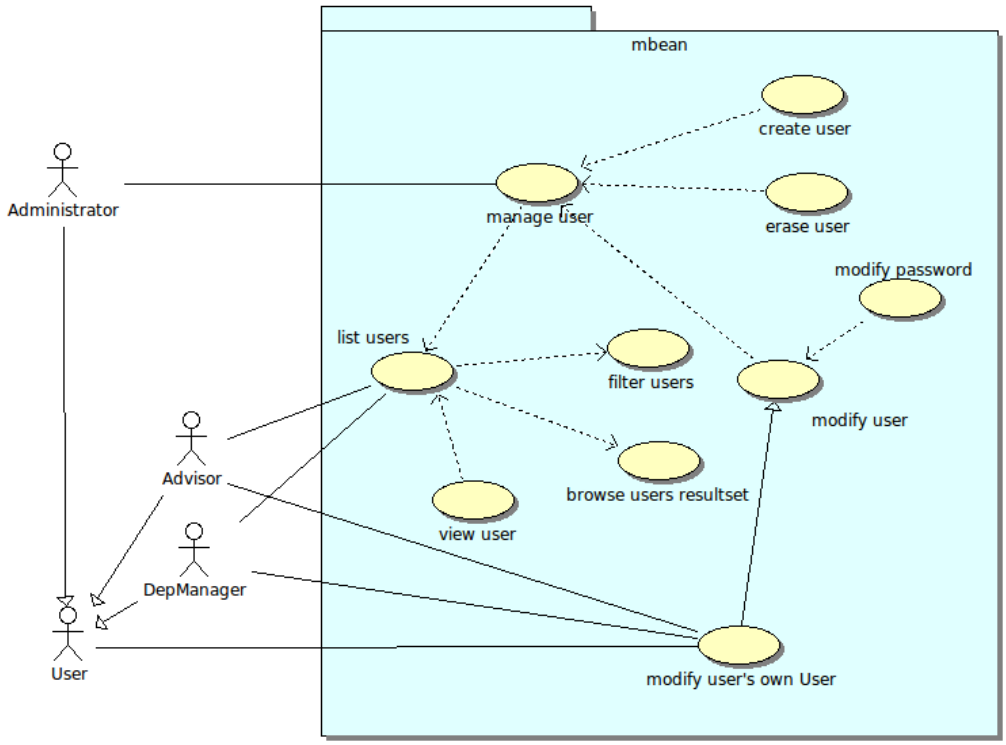
Az ábrán látható hitelesítési módot nem találtuk eléggé finoman hangolhatónak, a JSF technológiával való közelebbi megismerkedésünk során rájöttünk, hogy a Java Server Faces sajnos csak a laponkénti jogellenőrzéshez nyújt igazán kényelmes támogatást; ha a lapon szereplő komponensek működését is a felhasználói szerepkörtől függően szeretnénk vezérelni, akkor kénytelenek vagyunk a a 3 fejezetben leírtak szerinti módszert alkalmazni.

3.4 Felhasználói eset diagramok

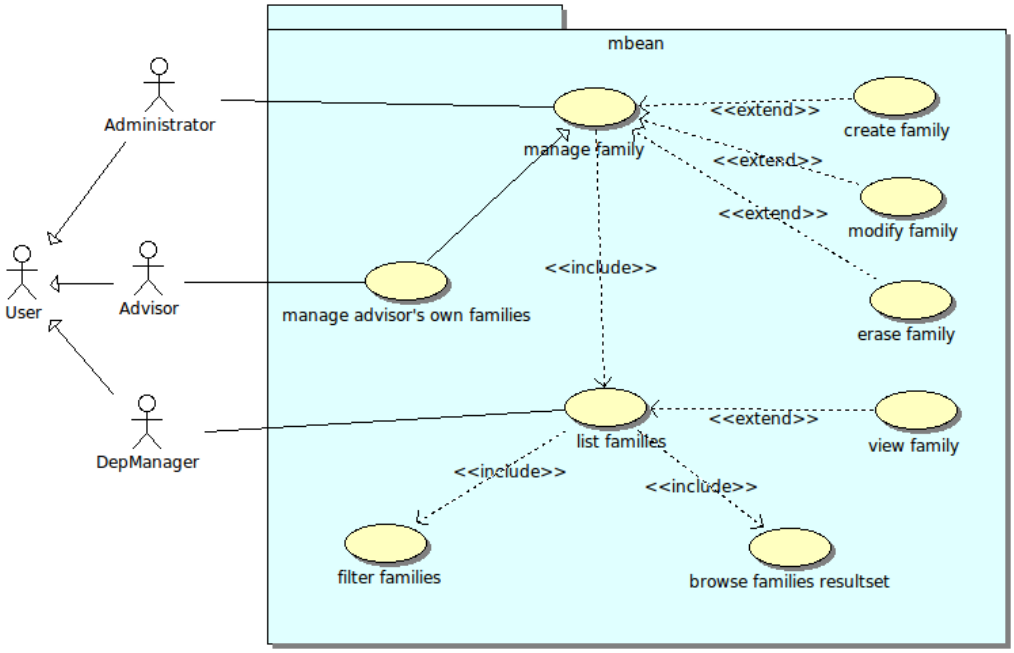
A felhasználói eset diagramok létrehozásakor igyekeztünk a különböző szerepkörű aktorokhoz tartozó felhasználói eseteket közös eset diagramban ábrázolni. Jó közelítéssel azt mondhatjuk, hogy az egyes esetdiagramoknak egy-egy JSF oldalt feleltetünk meg.



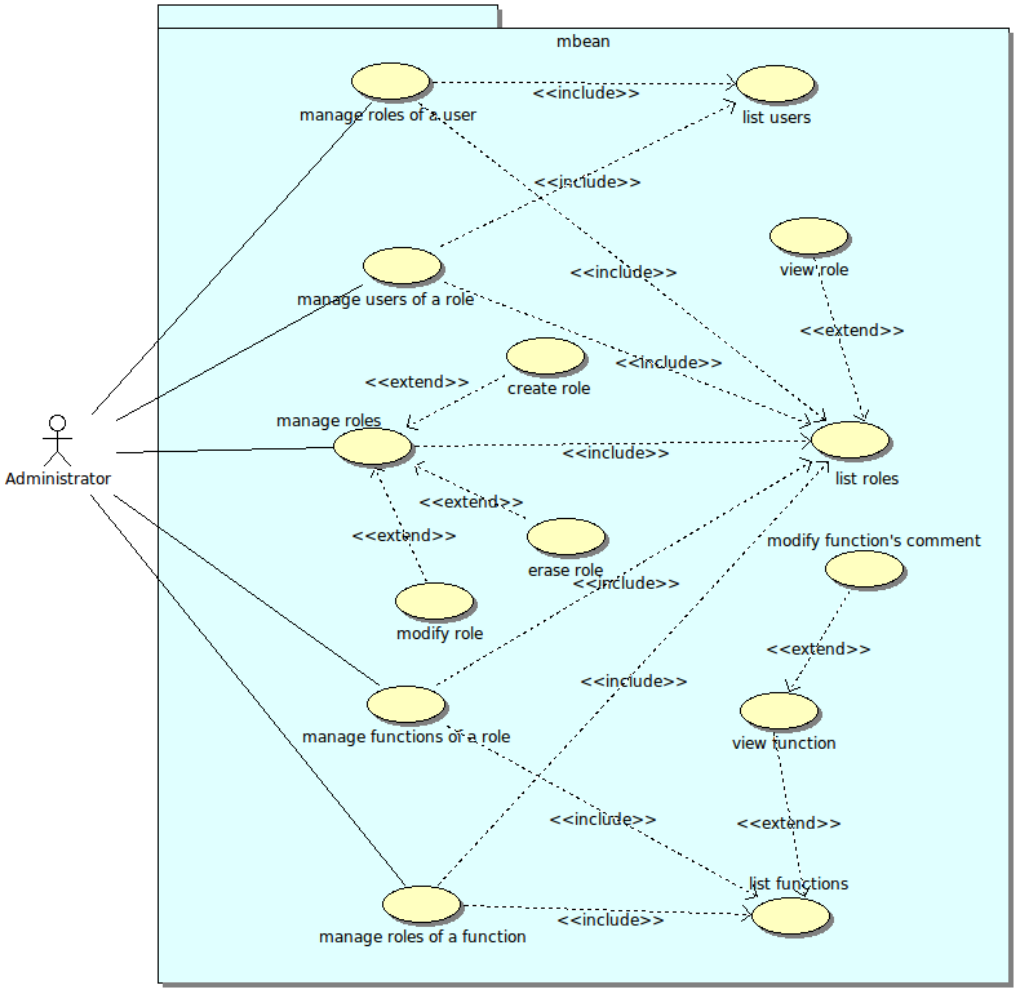
18. kép: Bejelentkezés használati eset diagram



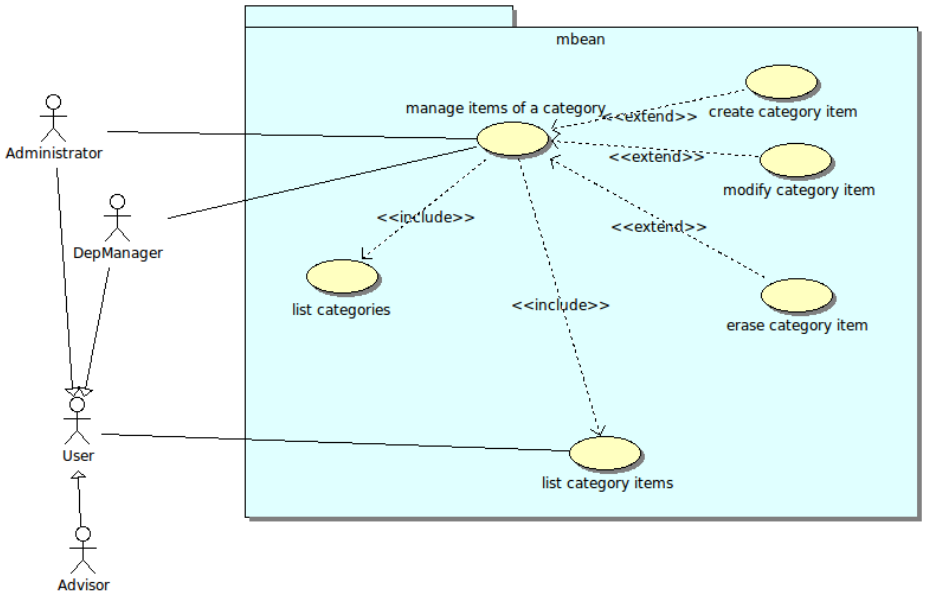
19. kép: Felhasználókezelés használati eset diagram



20. kép: Család kezelés használati eset diagram



21. kép: Hozzáférés kezelés használati eset diagram



22. kép: Kategória jellegű törzsadat kezelés használati eset diagram

4 Rendszerfejlesztési tapasztalatok

A rendszer fejlesztéséhez a rendelkezésre álló rövid idő és a megvalósítandó funkcionalitás viszonylagos egyszerűsége miatt is a vízesés modellt alkalmaztuk. A megrendelőkkel folytatott egyeztető találkozókon több körben próbáltuk meg felmérni a pontos igényeket, amik alapján elkészíthettünk egy többé-kevésbé véglegesnek tekinthető követelménylistát. Ez tartalmazta a funkcionális és nem funkcionális követelményeket egyaránt. A létrehozott követelményjegyzék pontos értelmezését segíti a projekt szakterületi fogalmait leíró fogalomszótár. Szakdolgozatom mellékletében bemutatok néhány, az egyeztetés során felhasznált dokumentumot.

A fentiekből kiindulva többszöri iterációval sikerült kialakítanom egy osztálydiagramot. Az osztályszerkezet kialakításakor igyekeztem csupán a szakterületi fogalmakra és folyamatokra koncentrálni, lehetőleg megfeledkezve a háttérben dolgozó relációs adattárolási elvből adódó korlátozásokról.

4.1 Követelmények pontosítása

Előző munkahelyemen sajnos saját bőrömön sikerült megtapasztalnom, mekkora káoszt okozhat egy projektben a nem kellő alapossággal végzett követelménytervezés. A szakdolgozati munkában ezért igyekeztem ezt a csapdát elkerülni, nagy hangsúlyt fordítva a szakterületi fogalmak és folyamatok lehető legpontosabb megértésére és rögzítésére. Szerencsére a megrendelői oldal kulcsszereplői is nagyon segítőkésznek mutatkoztak.

4.1.1 Megbeszélések és emlékeztetők

A kulcsfigurákkal történő egyeztetésekről a helyszínen minden esetben jegyzeteket készítettem. A kézi jegyzeteimet utólag egyeztettem projekttársammal is. Az így letisztázott és szövegszerkesztőben rögzített dokumentumot – a megbeszélés utólagos átgondolása során felvetődő újabb kérdésekkel kiegészítve – e-mailben elküldtem a megrendelői kulcsfiguráknak. Ők az emlékeztető egyes pontjait elfogadták vagy elutasították, illetve a feltett kérdésekre szintén e-mailben röviden válaszoltak. Minden újabb egyeztetést az előző megbeszélésen tisztázott dolgok ismételt pontosításával kezdtünk. Ennek a szigorú

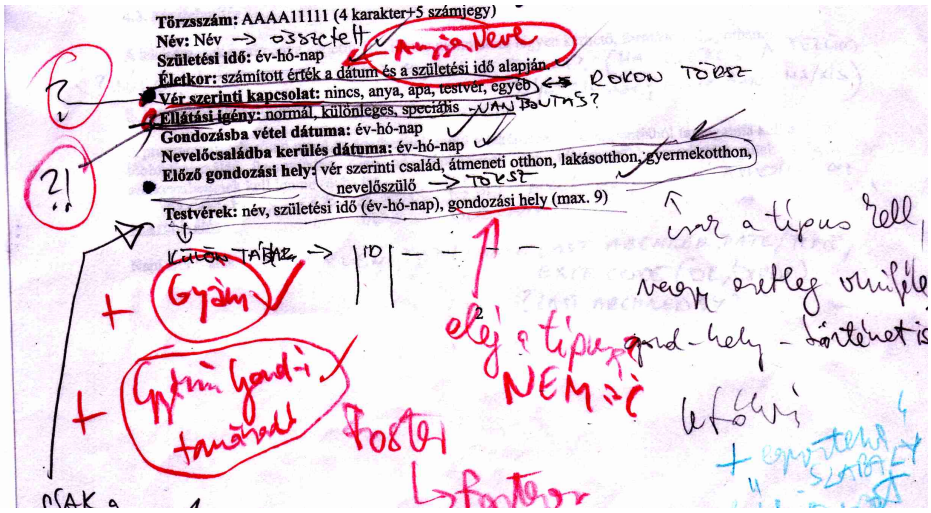
menetrendnek és a többszöri interjúknak köszönhetően sikerült elegendő információt összeszednünk a pontos szakterületi problémamodellezéshez. Természetesen csak nagyon kis szeletét ismerhettük meg a gyermekvédelmi munkának vagy a nevelőcsaládok működésének. Gyermekvédelmi szakemberré nem váltunk, de az egyeztetett és jóváhagyott követelményjegyzékből illetve és fogalomszótárból dolgozva már bízhattunk benne, hogy a modellezéskor nagy hibát már semmiképpen nem fogunk elkövetni.

4.1.2 Felhasznált eszközök

A követelménytervezés egyik leghatékonyabb eszközének egy ősi, általános célú hardver rendszer bizonyult: *színes tollak, filcek és ceruzák.*

Ez természetesen nem jelenti azt, hogy az integrált szoftverfejlesztő rendszerek kifinomult eszközrendszere haszontalan lenne. Ehhez a projekthez az Eclipse-be is beintegrálható *Visual Paradigm for UML* szoftvercsomagot próbáltam ki. Kényelmesen, különösebb tanulás nélkül intuitív módon is kezelhető felhasználói felületével megnyerte ugyan a tetszésemet ez a programcsomag, de az ősi hardverrel nem vetekedhet. Ha valakinek nagyon nagy rutinja van már a követelménytervezésben, akkor valószínűleg sok időt takarít meg azzal, hogy ilyen termékeket használ – és talán a termék meglehetősen borsos árát is kitermelheti a felhasználó ezzel az időmegtakarítással. A szoftvercsomag követelménytervezést támogató képességeit [vpumlrm] mutatja be nagyon impresszív módon.

A *Visual Paradigm for UML*-lel történő hosszas „játszadozás” után ismét visszatértem a gyökerekhez: elővettem az írószerszámokat, a papírra nyomtatott megbeszélés emlékeztetőket, saját jegyzeteimet, és használatba vettem *egy nagy asztalt* is. Ezen az *igazi clipboardon* szét tudtam teríteni a megrendelői oldal kulcsfiguráival történt megbeszéléseken megfogalmazottakról készített emlékeztetőket.



23. kép: Az igazi hardver clipboard egy elemének részlete

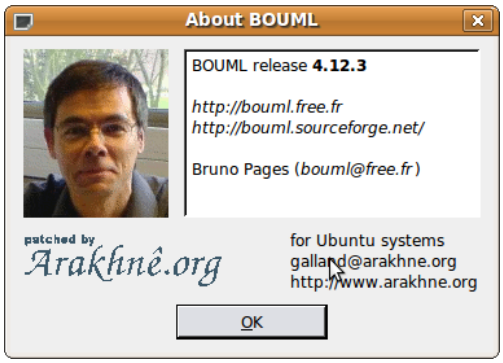
Így már korán sikerült észrevennem, ha a megbeszélésen rögzített követelmények ellentmondásba kerültek egy korábbi vagy későbbi megbeszélés állításaival. Annyira megtetszett ez a módszer, hogy magát a fogalomjegyzéket is először csak így, papíron állítottam össze. Később persze a letisztázott dokumentumokból az elektronikus változat is elkészült.

4.1.3 Eredmény

A követelménytervezés eredményeként létrejött a mellékletben is szereplő *szakterületi fogalomszótár* és a *követelményjegyzékek*.

4.2 Osztálydiagramok és felhasználói esetdiagramok kialakítása

Az osztálydiagramok kialakításához a BOUML-t használtam. A *donationware* szoftver nagy előnye, hogy viszonylag kevés egerészéssel, a billentyűzetet használva hatékonyan lehet benne létrehozni UML diagrammokat.



Igaz ugyan, hogy nem mindig úgy teljesíti az UML2 követelményeit, ahogy azt pl. [störrle] helyesnek gondolja, de nekem nagyon szimpatikusnak tűnt. A szerző arcát munkám során többször is megnéztem. A felhasználói eseteket is a BOUML-lel rajzoltuk meg. Kezdetben persze csak „cetliztünk”, az eredményeket csak [störrle] és különböző Internetes források hosszas tanulmányozása után kezdtük gépre vinni.

4.3 Alkalmazás szerkezet implementálása

Az implementálást Eclipse-szel[17] kezdtem, nagyon sok gondot okozott azonban az Eclipse és a Glassfish összehangolása. Hiába álltam át az EclipseCon alkalmából az időközben megjelent[18] *Glassfish Bundle for Eclipse*-re. Sajnos nem tudtam összehangolni az egymással elvileg „*flawlessly*” együttműködő komponenseket. Végül átálltam NetBeans-re. Főleg, hogy projekttársam is ezt javasolta.

Az Eclipse használata közben sikerült megszoknom a beépített szövegszerkesztőjének nagyon kellemes, szolgáltatásait és szinte korlátlan testre szabhatóságát. Véleményem szerint a NetBeans még csak meg se közelíti ezt a szintet. Viszont jobban együttműködött a GlassFish-sel. Végül aztán, [bigals]-nak köszönhetően még a projekthez tartozó automatikus kezdeti license-kommentet is sikerült rajta beállítanom.

4.4 EJB alprojekt implementációja

Az entitások implementálása során is számos meglepetéssel találkoztam.

Egy ilyen volt például, amikor egy teljes napon át próbáltam megküzdeni egy titokzatos

17 http://www.eclipse.org/
18 A hír: [gf4eclipse]

hibával, amit egy forráskódból már réges-régen kitörölt, ám a szerver emlékeiben nagyon is frissen élő, valami miatt a JNDI-ben is létezőként jegyzetten session bean okozott.

Másik emlékezetes kalandom az EJB3 szerinti helyi és távoli interfészekhez kötődik. Sajnos nem vettem komolyan a fejlesztőrendszer diszkrét *warning*ját, és ugyanazt a metódust a helyi és a távoli interfészen keresztül is publikáltam. Ezek után az alkalmazásom persze mindenféle titokzatos kivételeket dobált. Az egészben az volt a legmegtévesztőbb, hogy a dolog néha működni látszott.

4.4.1 Adatbázis réteg

Adatbázis motorként PostgreSQL 8.3 telepítését és alapvető behangolását [pginstall] útmutatása szerint végeztem el, és feltettem mellé a PGAdmin III-at is.

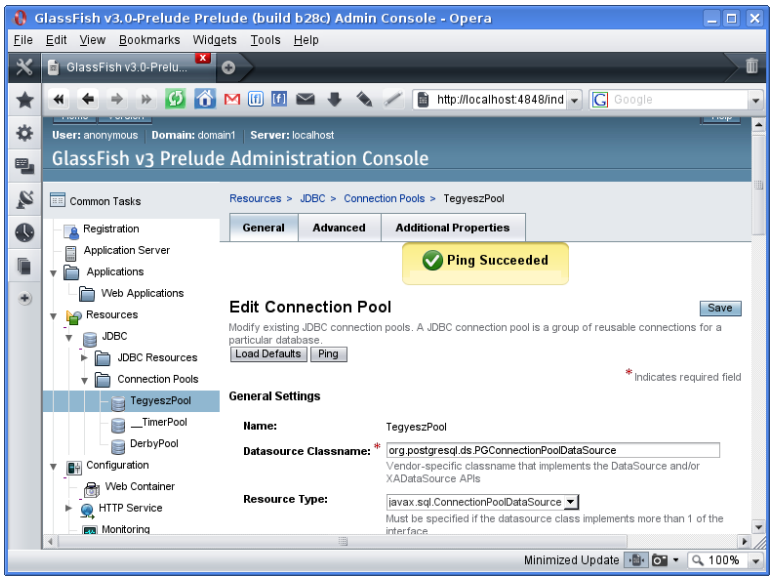
A PostgreSQL-lel meg voltam elégedve, soha nem hagyott cserben. Egyetlen alkalommal volt vele csupán gondom, de arról is én tehettem. Érdekes történet ez is.

Kísérletezgetéseim során perzisztencia szolgáltatóként a TopLink-et használva létrehoztam már jó néhány rekordot, örültem, hogy működik a dolog, majd visszaálltam Hibernate-re. Sokáig nem is volt ezzel semmi gond, mígnem egyszer elkezdte a GlassFish dobálni a rettenetes kivételeket. Én időközben már a forráskódot is átszerveztem kissé, ezért először magamra gyanakodtam. Valójában csak annyi történt, hogy a Toplink nyilván a saját szekvencia generátora szerinti id-ket adott a rekordoknak, és a Hibernate is a sajátját használta. A Toplikkel csak néhány táblát töltögettem fel, ezeket viszont történetesen olyan id-kkel, melyekhez épp akkor ért el a Hibernate szekvenciája is. Miután ezt felismertem, a Hibernate szekvenciájának aktuális érlékét kissé előrébb lökve azonnal ment is minden vidáman.

4.4.2 Connection Pool és Connection beállítása

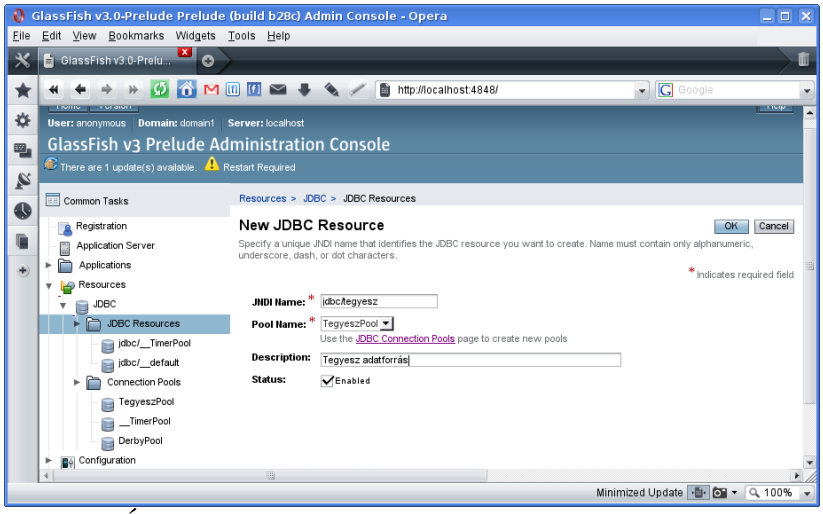
Mivel a JDBC kapcsolatot az alkalmazásszerver kezeli, a kapcsolódáshoz szükséges jar-t (az én estemben a `postgresql-8.3-604.jdbc4.jar`-t) elérhetővé kellett tennem a szerver által használt classpath-ból. Ennek legkézenfekvőbb módszerét választva bemásoltam a jdbc drivert a szerver lib könyvtárába.

Ezután – a szerver újraindítását követően – az admin konzolt felhasználva be kellett regisztrálnom egy erőforráskezelő által menedzselt pool-t a korábban már beállított adatbázis kezeléséhez. A beállítások egy alapszintű gyors ellenőrzését teszi lehetővé a beállított pool adatlapján elérhető *ping*.



24. kép: A Pool beállítás ellenőrzése

A pool beállítása után már csupán annyi volt hátra, hogy a JNDI-ben is regisztáljam őt JDBC adatforrásként. Ennek végrehajtását szemlélteti a következő ábra.



25. kép: Új JDBC adatforrás létrehozása

4.4.3 Entitások

Az entitások kialakítása is sok meglepetést tartogatott. Ennek érzékeltetésére szeretném most egyik élményemet részletesebben is elmesélni.

A 2.2 fejezetben említettem, hogy még visszatérek a `Category` osztályhoz. Ennek a példának az az érdekessége, hogy maga a `Category` soha nem példányosítható, így az eredeti koncepció szerint `InheritanceType.SINGLE_TABLE beállítása` volt. Ez szépen működött is, az adatbázisban egyetlen `Catgory` táblában tárolódtak a leszármazottak pédányai, miközben a JPA szépen kezelte a tábla diszkriminátor oszlopát is.

Ezzel azonban az volt a gondom, hogy ha a `value` oszlopra hitem szerint mindenképpen szükséges unique megszorítást már a `Category` osztályban megadtam, akkor ez természetesen az összes kategóriát átfogóan korlátozta. Tehát nem volt lehetőség két különböző kategória-leszármazott objektumban sem ugyanazt a `value`-stringet használnom. Ez azonban nem megengedhető, hiszen az az emberek működése más: hajlamosak vagyunk mindenféle kategóriaelem felsorolást egy „egyéb”-bel lezárni.

Azt viszont nem sikerült elérnem, hogy a fenti `InheritanceType.SINGLE_TABLE` beállítás mellett a leszármazottakban a diszkriminátor és a `thevalue` oszlop együttesen kapjon egy `unique` constraintet. Így aztán maradt az a megoldás, hogy minden egyes kategória-leszármazott külön táblában jelenik meg.

4.4.4 EJB3 Entitásmenedzser beanek

4.5 WAR alprojekt implementációja

A 2.5 fejezetben említett okok miatt bele kellett folynom az alkalmazás web szerver oldali részének implementálásába is.

A JSF technológiával való ismerkedésemben mindenekelőtt a [ig-j2ee] forrást használtam föl. A JSF-fel is úgy jártam, mint az EJB3 beanekkel. Mivel a *Java Server Faces* a JSP technológián alapul, működésének megértéséhez szükséges a JSP ismerete is. A *Java Server*

Pages használata viszont feltételezi *Java Servletek* világának ismeretét.

Ismét bebizonyosodott, hogy a programozásnak sincsenek királyi útjai: hogy igazán értsem, mi történik a háttérben, el kellett „játszadoznom” pár szervlettel és JSP lappal, mielőtt a JSF példákba belevágtam volna.

Különösen nagy segítséget és megvilágosodást jelentett számomra a [balusc-jsf] demó projekt.

Kezdetben volt olyan tervünk is, hogy a [jsf-matrix] JSF Matrix-on található AJAX-os komponensek valamelyikét használjuk, ettől azonban elriasztott minket a használatukkor generálódó kód ijesztő mennyisége és bonyolultsága. Maradtunk tehát az egyszerűbb alap komponenseknél.

5 Telepítés

Az Enterprise alkalmazások telepítése mindig igényel bizonyos előkészületeket. A telepítés megkezdése előtt a megfelelő szoftver környezetet kell kialakítani.

5.1 Előkészületek

1. **Alkalmazásszerver**

   A TeGyeSz Család program egy Java EE alkalmazásszerverre telepíthető föl. A fejlesztéshez a Sun támogatásával fejlesztett, nyílt forráskódú *GlassFish v2.1* alkalmazásszervert használtuk, de az alkalmazás futtatásához természetesen bármely más, a Java EE 5 specifikációt teljesítő alkalmazásszerver is megfelel. A glassFish szerverről további információk a [sun-gf] webhelyen érhetők el.

2. **Adatbázis-kezelő**

   A rendszer bármely elterjedtebb adatbázis-motort támogat. A fejlesztés során én PostrgreSQL adatbázissal dolgoztam, projekttársam pedig MySQL-t használt. A 4.4.1 fejezetben részletesen ismertetem a PostgreSQL telepítését és behangolását.

3. **Perzisztencia szolgáltató**

   Az alkalmazás `Tegyesz.war`-jában lévő `Tegyesz-ejb.jar META.INF` könyvtárában található `persistence.xml` telepítésleíró JPA Providerként a `Hibernate` -et használja, ezért ennek elérhetőségéről is gondoskodni kell. A `Hibernate` modulként a szerverre telepíthető egy kattintással, vagy akár úgy is, hogy a szükséges .jar-okat kézzel bemásoljuk az alkalmazásszerver lib könyvtárába. A jelenleg szükséges .jar [hb-jars] tartalmazza.

   A fejlesztés során kerültük a Hibernate-specifikus elemek használatát, ennek köszönhetően a `persistence.xml` megfelelő módosításával jó eséllyel használhatunk másik perzisztenciaszolgáltatót.

4. **JDBC driver**

   Az alkalmazott adatbázis-kezelőhöz tartozó JDBC drivert természetesen szintén be

kell másolnunk az alkalmazásszerver `lib` mappájába. További részletek olvashatók erről a 4.4.2 részben.

5. **Connection Pool, Connection, JNDI Binding**

   Az alkalmazás JTA alapú tranzakciókezelést használ, így létre kell hoznunk egy Connection Pool-t az adatbázisunk számára. Ennek felhasználásával definiálnunk kell egy Connectiont. A `Tegyesz-ejb.jar`-ban megadott alapértelmezés szerint az alkalmazás perzisztencia egysége a JTA által menedzselt adatforrást `jdbc/tegyesz` néven fogja keresni a JNDI névtárban. További részletek találhatók erről a 4.4.2 fejezetben. Természetesen akár a telepítésleíró is módosítható.

5.2 A telepítés folyamata

A telepítéshez az alkalmazást tartalmazó `Tegyesz.ear` fájlt kell feltölteni a szerverre. Ez történhet pl. a szerver menedzser webes felületéről. A kihelyezés (deployment) során az alkalmazásszerver bejegyzi a perzisztencia egység session beanjeit, a perzisztencia ellátó (persistence provider) automatikusan létrehozza a szoftver működéséhez szükséges adattáblákat, kapcsolatokat, megszorításokat és szekvenciákat az adatbázis alapértelmezett (`public`) sémájában, és persze a war alprojekt megfelelő elemei is alaphelyzetbe kerülnek. A folyamat részletei a szerver naplójában is megtekinthetők.

5.3 A rendszer üzembe helyezése

Sikeres telepítés után a rendszer adatbázisa még üres, azt fel kell tölteni az alapértelmezett törzsadatokkal. Ez az alaphelyzetbe állítás a `http://szerver:8080/Tegyesz-war/faces/init.jsp` oldal meglátogatásával tehető meg. A webcímben szereplő „`szerver`” természetesen behelyettesítendő a szerver tényleges nevével vagy IP számával.

Az alaphelyzetbe állítás egyetlen gombnyomással indítható, és nem igényel semmiféle előzetes hitelesítést. Azt követően azonban, hogy a rendszer alaphelyzetbe állítása megtörtént, már nincs mód ismételt alaphelyzetbe állításra. Alaphelyzetbe állításkor automatikusan létrejönnek az alapértelmezett törzsadatok, szerepkörök és a program verziónak megfelelő rendszerfunkciók, valamit a szerepkörök és rendszerfunkciók egy alapértelmezett egymáshoz

rendelése. Létrejön továbbá egy adminisztrátori szerepkörrel ellátott felhasználó.

A további felhasználók létrehozását, a működés testreszabását, a szerepkörök-funkciók összerendelését is a rendszer adminisztrátora végezheti el a `http://szerver:8080/Tegyesz-war/` webhely meglátogatását követően. Az alapértelmezett bejelentkezési név és jelszó egyaránt – idézőjelek nélküli – „`admin`”.

6 Összefoglalás

6.1 A projektről

A TEGYESZ CSALÁD projekt fejlesztése során sok olyan problémával találkoztam, melyek megoldásának ismerete minden bizonnyal a jövőben is hasznos lesz számomra. Alkalmam nyílt belekóstolni a Java Enterprise környezet világába. A látottak megerősítették bennem azt a meggyőződést, hogy a Java EE technológia eszközrendszerének hatékony kihasználása nagy felkészültséget igényel.

6.2 Adatforrásokról, segítségekről

A munka során sok forrást tekintettem át, megismertem sok fórumot, blogot, szakembert. A nekem leginkább tetsző helyeket meg is jelöltem magamnak[19], hogy a későbbiekben is visszatérhessek hozzájuk. A fejlesztésben leginkább [ig-j2ee] volt segítségemre. Ahhoz azonban, hogy az említett könyv valódi partnerévé válhassak, már korábban meg kellett ismernem a Java szemléletű programfejlesztés alapjait. Ebben sokat köszönhetek [vég-ij]-nek.

Végül, de nem utolsó sorban sok gondomat, bizonytalanságomat és kérdésemet sikerült eloszlatniuk csoporttársaimnak, és sokat köszönhetek konzulensem támogatásának is. Záborski László projekttársamat azért hagytam a felsorolás végére, hogy külön kiemelhessem: ő is nagyon sokat tett együttműködésével a projekt megvalósulása érdekében.

6.3 Jövőbeli tervektől

Ahogy a bevezetőben is említettem, a projekttel távlati terveink is vannak. Ez egyben azt is jelenti, hogy a későbbiekben is szeretnék kapcsolatban maradni Java technológiákkal. Bízom benne, hogy ezt a tervemet sikerül is megvalósítanom.

19 Könyvjelzőim többsége publikus, a http://delicious.com/szikora/ címen elérhető.

7 Irodalomjegyzék

[vég-ij] Vég Csaba, Instant Java / Java EE / SOA I. és II., Logos2000 2007

[störrle] Harald Störrle, UML 2 Unified Modelling Language, Panem 2007

[jp-hbn] Christian Bauer, Gavin King, Java Persistence with Hibernate, Manning 2007

[ig-j2ee] Imre Gábor(szerk is), Balogh Péter, Berényi Zsolt, Dévai István, Soós István, Tóthfalussy Balázs, Szoftverfejlesztés Java EE platformon, Szak kiadó 2007

[pg2003] Paul Graham: Beating the averages
http://www.paulgraham.com/avg.html

[hb-jars] Hibernate in GlassFish - Reloaded
http://blogs.sun.com/alexismp/entry/hibernate_in_glassfish_reloaded

[sun-gf] GlassFish - Open Source Application Server
https://glassfish.dev.java.net/

[jsf-matrix] AJAX JSF Matrix
http://www.jsfmatrix.net/

[balusc-jsf] The BalusC Code: Debug JSF lifecycle
http://balusc.blogspot.com/2006/09/debug-jsf-lifecycle.html

[pginstall] PostgreSQL - Community Ubuntu Documentation
https://help.ubuntu.com/community/PostgreSQL

[bigals] Big Al's Blog: Using the "License" functionality in NetBeans 6
http://bigallan.blogspot.com/2008/02/using-license-functionality-in-netbeans.html

[gf4eclipse] The Aquarium: Glassfish Bundle for Eclipse - 0.9.9
http://blogs.sun.com/theaquarium/entry/glassfish_bundle_for_eclipse_0

[vpumlrm] Visual Paradigm for UML Requirements Management
http://www.visual-paradigm.com/product/vpuml/demos/requirements/

[mindprod] Roedy Green: Canadian Mind Products Java & Internet Glossary
http://mindprod.com/jgloss/jgloss.html

[jpa-faq] Japa Persistence API FAQ
http://java.sun.com/javaee/overview/faq/persistence.jsp

[jw-tomcat] Javaworld: Is Tomcat enterprise ready?
http://www.javaworld.com/javaworld/jw-01-2008/jw-01-tomcat6.html

[640kB] Wikiquote Talk: Bill Gates 640K/1MB
http://en.wikiquote.org/wiki/Talk:Bill_Gates#640K.2F1MB

[ősTegyesz] Felhasználói kézikönyv a TEGYESZ információs rendszerhez, Belső kiadás ,2008

8 Mellékletek

8.1 A követelmények első megfogalmazásának egy oldala

Ingatlan alapterülete: m2
Szobák száma: db, max 9 → ~~for~~ Config tömb: MaxSzoba
Szobák alapterülete: az egyes szobák alapterülete m2-ben.
Férőhelyek szobánként: alapterület / 6 (max) Config tömb: Férőhely Formula
Összes férőhely alapterület szerint: az egyes szobák férőhelyeinek összege (számított érték)

TBO ← **Saját gyerekek:** név, nem, születési idő (max. 9)
Nevelt gyerekek: száma, név, nem, születési idő, törzsszám (AAAA11111) (max. 9),
TBO ← **Egyéb rokon:** név, nem, születési idő, rokonság, megjegyzés
↳ TÖRZS

A FIKSZ-tréner által javasolt férőhelyek száma: *n (1-8)* ← Config
Felülvizsgálat alapján javasolt férőhelyek száma: *n (1-8)*
Az aktuális működési engedélyben meghatározott férőhelyek száma: *n (1-8)*
Bővíthetőség: Összes férőhely alapterület szerint – Nevelt gyerekek száma (számított érték)
Változások: „csökkenés várható: *n*", „növekedés várható: *n*", „–"

Problémajelzés: szöveg, hosszabb
Erősségek: szöveg, hosszabb
Vállalások a nevelt gyermekekre vonatkozóan:
Nemük: fiú, lány, nem fontos
Életkoruk: 0-1 év, 1-3 év, 3-6 év, 6-12 év, 12-18 év (és/vagy) → TÖRZS 2 MEZŐ!
Származásuk: roma, csak magyar, mindegy, egyéb: „..." ? ← COMMENT
Mentális képesség: csak normál, fejlesztéssel korrigálható hiányosság, enyhe mentális retardáltság, középsúlyos mentális retardáltság, egyéb: „..." ?
Egészségi állapot: csak egészséges, orvosi kezeléssel korrigálható állapot, krónikus de kezelhető/szintentartható állapot
Pszichés állapot:
Kizáró/nem vállalt ok:

Nevelőszülő tanácsadó: név — USER_ID
Területi koordinátor: név — USER_ID

2.2. A nevelt gyerekek adattáblájában rögzítendő adatok:

Törzsszám: AAAA11111 (4 karakter+5 számjegy)
Név: Név → összefelt
Születési idő: év-hó-nap
Életkor: számított érték a dátum és a születési idő alapján.
Vér szerinti kapcsolat: nincs, anya, apa, testvér, egyéb ↔ ROKON TÖRZS
Ellátási igény: normál, különleges, speciális - VAN BOUTMS?
Gondozásba vétel dátuma: év-hó-nap
Nevelőcsaládba kerülés dátuma: év-hó-nap
Előző gondozási hely: vér szerinti család, átmeneti otthon, lakásotthon, gyermekotthon, nevelőszülő → TÖRZS
Testvérek: név, születési idő (év-hó-nap), gondozási hely (max. 9)
↓
KÜLÖN TÁBLÁBA → | ID | — — — —

8.2 Az első követelmény-egyeztetésről készült emlékeztető első oldala

Emlékeztető követelmény pontosításról

Tartalomjegyzék

1.1 Körülmények

Hely Hadú Bihar Megyei Területi Gyermekvédelmi Szolgálat, Debrecen
Dátum 2009-02-19
Jelen vannak

- Megrendelői oldalról
  - Ferenczi Beáta, igazgatóhelyettes, a nevelőszülői hálózat vezetője
  - Békési József, **sajnos az ő beosztását elfelejtettem megkérdezni...**
  - Erdei Sándor, igazgató
- Fejlesztői oldalról
  - Szikora Zsolt PTI MSc hallgató

1.2 Tisztázott, pontosított témák

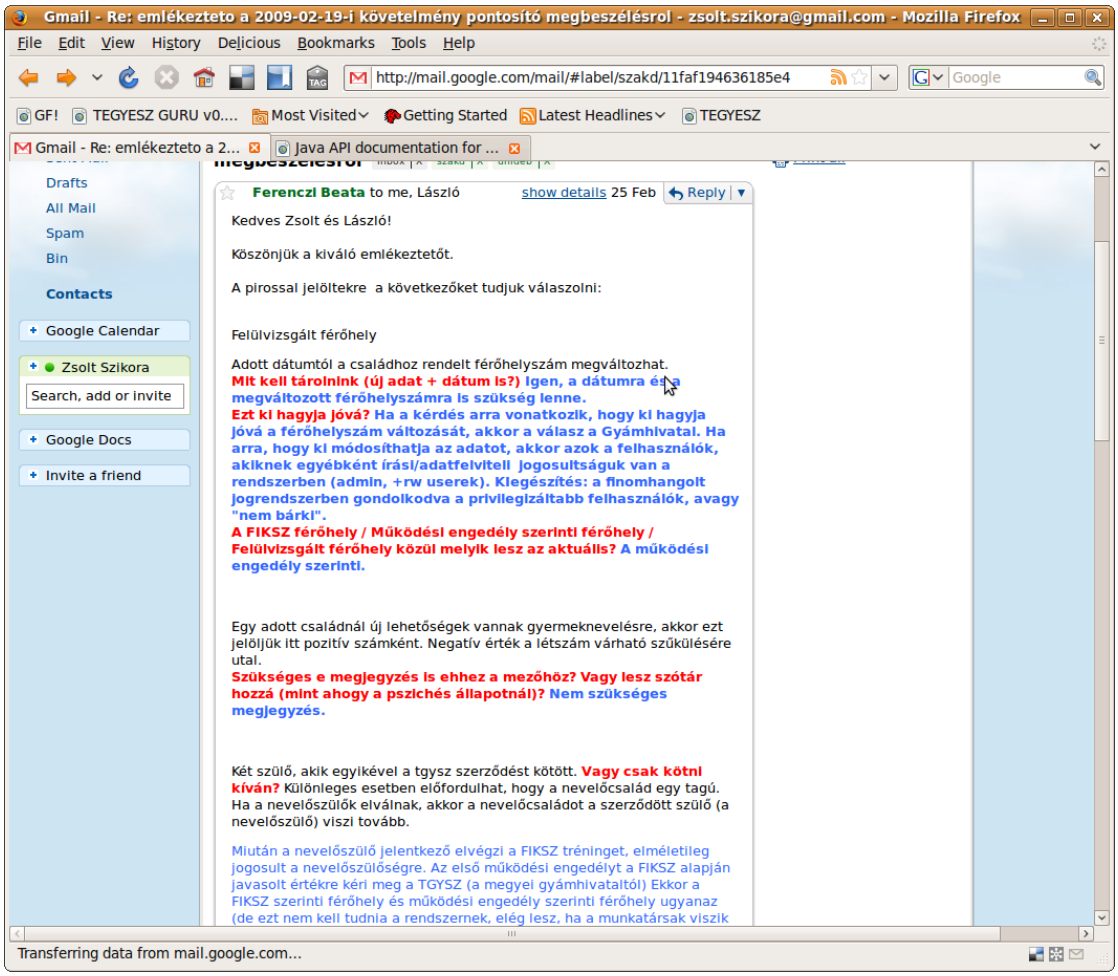
1.2.1 A rendszer működési környezete

- csak intraneten lesz elérhető
- Kliens-Szerver felépítés
  - Kliens: lehetőleg tetszőleges web böngésző
  - Szerver: MS Windows 2000; jelenleg egy csupán fájl kiszolgálóként működik.
  - Távlati cél: szerver – és a kliensek – átállítása valamilyen Linux disztribúcióra

1.2.2 Nevelőszülői hálózat nyilvántartás adatlap módosítása:

- *FIKSZ adatok (Dátum, Tréner, ...)*
  férj és feleség számára nem feltétlenül azonosak.
- *Felülvizsgálatok (Dátum, Pszichológus,...)*
  férj és eleség számára nem feltétlenül azonosak.

8.3 Egy emlékeztetőben feltett utólagos kérdésekre adott válasz egy részlete



8.4 Menürendszer, szerepkörök és rendszerfunkciók alapértelmezett kapcsolata

Szerepkörök						Funkció		Megjegyzés	Főfunkció azonosító	Alfunkció azonosító
unauthorized	user	administrator	advisor	department manager	Fő funkció	Alfunkció				
1					Bejelentkezés				LoginPageBean	doLoginAction
						Regisztráció		ebben a projektben szükségtelen	RegisterPageBean	doRegisterAction
1						Elfelejtett jelszó		egy következő verzióban	LostPasswordPageBean	doRetrievePassword
	1	1	1	1	Saját adatlap megtekintése				OwnDataPageBean	
	1	1	1	1		saját jelszó módosítás			OwnDataPageBean	modifyPasswordAction
	1	1	1	1		saját adat módosítás			OwnDataPageBean	modifyData.Action
1	1	1	1	1	Segítség				HelpPageBean	
1	1	1	1	1	Keresés általános keresés			egy következő verzióban (szerepkör szerint)	GeneralSearchBean	doSearchAction
1	1	1	1	1	Menütérkép			egy következő verzióban (szerepkör szerint)	MenuMapPageBean	getManuMapAction
		1		1	Kategória kezelő				CategoryPageBean	
		1		1		kategóriák listázása			CategoryPageBean	getCategoryListAction
		1		1		kategória elemek listázása			CategoryPageBean	getCategoryItemListAction
		1		1		kategória elem létrehozása			CategoryPageBean	createNewCategoryItemAction
		1		1		kategória elem módosítása			CategoryPageBean	modifyCategoryItemAction
		1		1		kategória elem törlése			CategoryPageBean	deleteCategoryItemAction
		1			Funkciók kezelése				FunctionPageBean	
		1				funkciók listázása			FunctionPageBean	getListAction
		1				módosítás		**csak** a megjegyzés módosítható	FunctionPageBean	modifyAction
		1				funkció - szerepkör összerendelés			FunctionPageBean	toRoleAction
		1			Szerepkörök kezelése				RolePageBean	
		1				szerepkörök listázása			RolePageBean	getListAction
		1				létrehozás			RolePageBean	createAction
		1				módosítás			RolePageBean	modifyAction
		1				törlés			RolePageBean	deleteAction
		1				szerepkör - funkció összerendelés			RolePageBean	toFunctionAction
		1				szerepkör - felhasználó összerendelés			RolePageBean	toUserAction
		1	1	1	Felhasználók kezelése				UserPageBean	
		1				felhasználók listázása			UserPageBean	getListAction
		1				létrehozás			UserPageBean	createAction
		1	1	1		egyedi rátekintő			UserPageBean	getDetailsAction
		1				adatok módosítás			UserPageBean	modifyDetailsAction
		1				jelszó módosítás			UserPageBean	modifyPasswordAction
		1				törlés			UserPageBean	deleteUserAction
		1				Felhasználó – szerepköreinek módosítása			UserPageBean	modifyRolesAction
		1	1	1	Család kezelés				FamilyPageBean	
		1		1		családok listázása			FamilyPageBean	getListAction
			1			tanácsadóhoz tartozó családok listája		ez egy szűkített lista	FamilyPageBean	getOwnListAction
		1	1			létrehozás			FamilyPageBean	createAction
		1	1	1		egyedi rátekintő			FamilyPageBean	getDetailsAction
		1	1			módosítás			FamilyPageBean	modifyAction
		1	1			törlés			FamilyPageBean	deleteAction