SZAKDOLGOZAT

ZÁBORSKI LÁSZLÓ

Debrecen

2009

DEBRECENI EGYETEM

INFORMATIKA KAR

WEBES INFORMÁCIÓS RENDSZER FEJLESZTÉSE

Témavezető:

DR. ADAMKÓ ATTILA

Egyetemi Tanársegéd

Készítette:

ZÁBORSKI LÁSZLÓ

Programtervező Informatikus MSC

Debrecen

2009

Tartalomjegyzék

Bevezetés

Kitűzött Cél

A szakdolgozat célja, hogy egy projektben készítsünk el egy Webes információs rendszert a Hajdú-Bihar Megyei Területi Gyermekvédelmi Szakszolgálat[1] [továbbiakban: *hbmtgysz*[2]] szakemberei részére. Másodlagos cél, hogy a címben megadott alkalmazás fejlesztői megismerkedjenek a projektben való munkával, és a technika állása szerinti modern alkalmazásfejlesztéssel, valamint technológiákkal.

Megrendelői Háttér

A *hbmtgysz* szakembereinél rendelkezésre áll jelenleg is egy, a szakszolgálatoknál országosan is elterjedt, hálózaton keresztül történő elérést és több felhasználó egyidejű munkáját nem támogató rendszer. Ennek a rendszernek a hatékonysága a mai technikákhoz képest rendkívül alacsony. A helyi Területi Gyermekvédelmi Szakszolgálatban [továbbiakban: TEGYESZ], ezen a szoftveren keresztül kezelnek jelenleg nagyjából 700 gyerek és 250 nevelő család személyes és egyéb adatait.

Távlati célnak feltűnt számunkra, hogyha a meglévő programot sikerül lecserélni egy olyan új programra, ami jó minőségben és jól használható módon működik, vagyis felhasználóbb baráttá válik, akkor akár az összes TEGYESZ munkáját meg tudnánk vele könnyíteni.

Aktuális állapot

A rendszert egy nagyon jól szabályozható jogrendszerrel együtt alakítottuk ki. A rendszerterv kialakítása után elkészítettünk egy a rendszer főbb elemeinek egy szűkített funkcionalitású, de működőképes implementációját. Ez az implementáció már minden a végső változatban használatos technológiát felhasznál.

A projekt szempontjából jelenleg a végrehajtás utolsó fázisánál tart, mivel a technológiák megismerése rengeteg időt elvett, ezért az átadásra, a kitelepített környezeten való tesztelés és ezzel együtt a projekt lezárására folyamatban van.

[1] Internetes elérhetőségük: http://www.hbmtgysz.hu

[2] hbmtgysz: Hajdú-Bihar Megyei Területi Gyermekvédelmi Szakszolgálat

A dolgozat felépítése

A dolgozatot öt fejezetből építettem fel. Az első fejezetben ismertetem a projektet. A második fejezetben bemutatom a fejlesztéshez alkalmazott szoftvereket, technológiákat és egy számomra fontos hardver elemet. A harmadik fejezetben bemutatom a tervezet és megvalósult szoftverfejlesztési folyamatot. A negyedik fejezetben részletezem a használati eseteket. Az ötödik fejezet egy kiemelt használati eset, a paraméterek kezelésének megvalósítását mutatja be.

1. A projektmunka

Projekt és Vállalatirányítási tanulmányaink[3]-ból merítve projektünk a Weiss és Wysocki [1994] féle elméletet követve öt szakaszra bontható, és megvalósulása koordinációs szervezeti formában történt.

1.1. A koordinációs szervezeti forma

1.1.1. Jellemzői

A projekt vezetője tulajdonképpen egy koordinátor, akinek csak információs, tanácsadói és tervezési jogköre van és feladatai elsősorban a határidők és költségkeretek betartására korlátozódnak

1.1.2. Előnyei

A projekttagok szokásos szervezeti egységüknél maradnak. Nem változik a vezető-munkatárs viszony, így nincsen szükség sem a munkaerő „kikérésére", sem reintegrációjára. Nagyon rugalmas a munkaerő felhasználása. Nagyobb lehetőség kínálkozik az adott egységnél a tapasztalatok cseréjére, hiszen maga az egység egyszerre több projekten is dolgozik

1.1.3. Hátrányai

Nem egyértelmű a projekttel kapcsolatos felelősség kérdése, hiszen a projekt koordinátora nem rendelkezik elegendő „súllyal". Az egyes szervezeti egységek közötti lassú a döntések meghozása a megegyezés időigényessége miatt , mely súlyos késedelmekhez vezethet és sok konfliktust eredményezhet, mivel a projektvezetőnek nincs hatásköre, erősen függ a szakirányú vezetők meghatározásaitól. A döntés előkészítés és a döntés elválasztásából sok konfliktus származhat. A projektkoordináció rengeteg idő- és erőráfordítást igényel

[3] Előadó: Dr. Rutkovszky Edéné

1.2. Projektben vállalt szerepkörök

Projekt koordinátor:

Dr. Adamkó Attila - Egyetemi Tanársegéd

Projekt tagok:

Szikora Zsolt DEIK-PTIMSC hallgató - szoftverfejlesztő[4]

Záborski László DEIK-PTIMSC hallgató - szoftverfejlesztő

1.3. Fázisai

1.3.1. Projekt kialakítás

Követelmények elemzése során, többször is kellett egyeztetni a megrendelővel, mire kialakult a végleges megvalósíthatósági tanulmány. Fontos megemlítenem, hogy az elemzések során meg kellett ismerkednünk egy TEGYESZ működési fázisaival, fogalmaival ahhoz, hogy pontos képet kapjunk az elkészítendő rendszer használati eseteiről. A **projekt célja** elsősorban a megrendelő [*hbmtgysz*] által elvárt követelményeknek való megfelelés. A **változatok kialakítása** indulásnál a technológia választás során volt. Például egy változat, amit később a koordinátor javaslatára elvetettünk, hogy XML alapú adatbázist használunk a projekt során.

1.3.2. Tervezés

Projekttársammal közösen úgy döntöttünk, hogy a későbbi újrafelhasználhatóság és bővíthetőség érdekében a projektet *JAVA Vállalati Környezet*[5]-ben készítjük el. A koordinátor javaslatára a használt technológiák: Hibernate, JPA, JSP, JSF. Ezen technológiák megismerése után tudtunk csak **feladatokat rögzíteni**, illetve **tevékenységeket azonosítani**, ezért ez a rész egy kissé nehézkesen haladt.

1.3.3. Szervezés

A projekt a *1.1 A koordinációs szervezeti forma* fejezetben leírtak alapján valósult meg, ahol Attila segédkezett nekünk, hogy munkánkat minél jobban el tudjuk látni. A **munka felosztása** szerint a **követelmény és a rendszer tervezésében** mindketten részt vettünk. A

[4] Információ róla: http://zsolt.szikora.info

[5] Java Enterprise Edition [Java EE]

követelmények pontosításánál projekttársam vállalta a fő szerepet, lévén több szabadidővel rendelkezett [még a projekt indulása előtt vált aktív munkakeresővé]. A rendszer tervezése során a jogrendszer és a felhasználó kezelési mód az én ötletem alapján került megtervezésre. Az implementálásnál pedig felhasználtuk a Java EE strukturális kialakítását, úgy, hogy Zsolt készíti el az EJB réteget, én pedig a WAR részt. Ekkor még bíztunk abban, hogy könnyen fogunk haladni, hiszen gontoltuk, hogy jól elkülöníthető ez a két rész egymástól az adott technológiáknál.

1.3.4. Végrehajtás

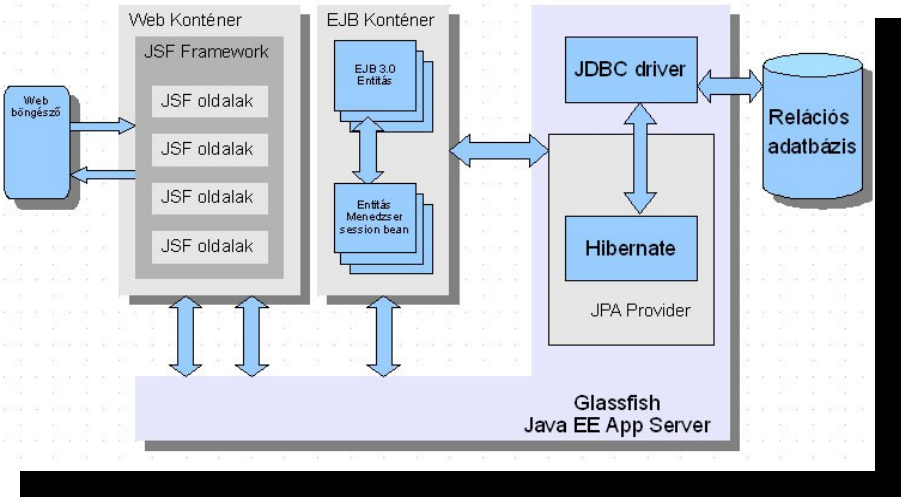
Ennél a fázisnál szembesültünk azzal, hogy, a munka felosztása nem volt tökéletes, mert a két fejlesztendő terület sajnos teljesen egymásra épül, így mindkettőnknek meg kellett ismerkedni a választott technológiákkal. Ezen kívül nagyban függtünk attól, hogy a projekttársunk hogyan haladt, tehát hiányzott a kritikus út megtervezése.
Mivel a *hbmtgysz* ajánlotta, hogy ha a projekt sikeresen zárul, akkor segít nekünk a többi TEGYESZ-nél való bemutatkozásban, ezért feltételezem, hogy akár hosszú távú szerződésre is számíthatunk, ezért a fejlesztési modellnek a spirál modellt választottuk, amiről később még részletesen írok.

1.3.5. Projektlezárás

Jelenleg az átadás még hátra van, de remélem azonban, hogy mire államvizsgázni megyünk, a projektet sikeresen lezárjuk.

2. A fejlesztés során használt eszközök, technológiák

2.1. Alkalmazás technológiai felépítése



1. ábra Alkalmazás felépítése

Java alkalmazásunk tehát GlassFish szerveren található, megjelenítésre JSF és JSP technológiákat használ. A JSF-k és JSP-k előállítanak Html 4.0 típusú oldalt, CSS grafikai elemekkel. Az adatbázis szerverrel [MySQL, Postgre] JDBC-n keresztül kommunikál egy JPA-val, melynek megvalósítására a Hibernate-t használtuk fel.

2.2. Fejlesztői, tesztelői környezet

A fejlesztéshez és a teszteléshez, ugyanúgy igyekeztünk ingyenesen használható szoftvereket használni, ahogyan maga az alkalmazás is ilyen környezetben fut majd. A szoftver elkészítéséhez a NetBeans IDE 6.5.1 verziójával éltem. Adatbázis kezelőnek a MySQL 5.0.20-nt-t választottam, de az alkalmazás adatbázisszervertől független. Ezt a függetlenséget bizonyítandó projekttársam a PostgreSQL-t használta fel az adatbázisának kezelésére. Alkalmazásunkat Microsoft Internet Explorer 7.0, Mozilla Firefox 3.0 és Opera böngészőkkel teszteltük.

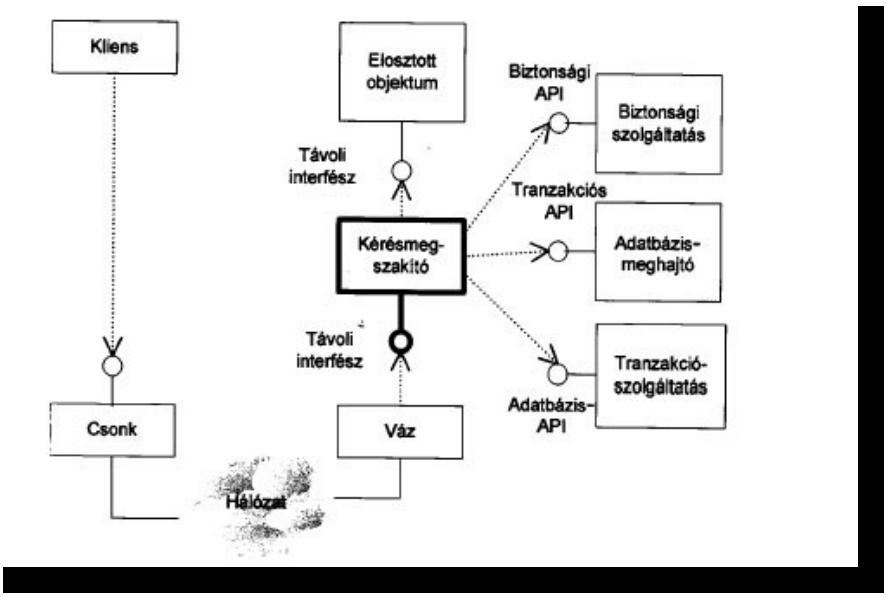
2.3. Java EE, Glassfish szerver

A Java nyelv [1991] és a hozzá kapcsolódó technológiák folyamatos fejlődése miatt a Java-t három különböző kiadásra [edition] osztották. A Java Standard Edition [Java SE] hagyományos desktop alkalmazások és appletek fejlesztését teszi lehetővé, a Java Micro Edition [Java ME] segítségével mobil eszközökre készíthetünk alkalmazásokat, és végül Java

Enterprise Edition [Java EE], mely elosztott, sok felhasználóval rendelkező, vállalati méretű szoftverrendszerek fejlesztéséhez nyújt támogatást.

A Java EE egymondatos meghatározása Imre Gábor könyve szerint: „Architektúra vállalati méretű alkalmazások fejlesztésére, a Java nyelv és internetes technológiák felhasználásával.” Egy ilyen környezetre való fejlesztésnél a következő szempontokat kell figyelembe venni:

- Perzisztencia: A rendszer működéséhez szükséges adatok tárolása perzisztens módon történik, általában relációs adatbázisban.
- Többszálúság: Egyidejűleg több felhasználó is elérhesse az elkészült alkalmazást.
- Tranzakciókezelés: Az adatok konzisztenciájának figyelése.
- Távoli elérés: A kapcsolódási protokollok egységes kezelése.
- Névszolgáltatás használata
- Skálázhatóság: A rendszer akár több hardverre szétszedésének figyelembevétele
- Magas rendelkezésre állás: Klaszterezés használata
- Aszinkron üzenetküldés, amennyiben arra szükség van.
- Biztonság: Jogosultság kezelés és adat biztonság
- Monitorozás: Jól átlátható statisztikák készítése



2. ábra Implicit middleware[6]

Ezeket a szolgáltatásokat a Java EE támogatja különböző technológiákkal, az általunk használtakat ezekből külön feltüntetem a 2.8 Technológiák részben. Ezeket az előre elkészített „modulokat” middleware-szolgáltatásnak hívják. Ezen szolgáltatásokat igyekeztünk

[6] Imre Gábor: Szoftverfejlesztés Java platformon 13. oldal 2.2. ábra

alkalmazásunknál megvalósítani, ha lehetséges volt, akkor szabványos elemeket [API-kat] használva. A kétféle középréteg megvalósítás közül az implicit típusúra (2. ábra) tettük a hangsúlyt, hogy könnyebben lehessen a későbbiek során továbbfejleszteni, módosítani.
A Java EE szolgáltatásokat használó alkalmazások futtatására képes alkalmazásokat Java Alkalmazás kiszolgálóknak [Java Application Server] nevezzük, ennek egy megvalósítását választottuk alkalmazásunk futtató környezetére, a GlassFish-t, annak is a 2.1-s verzióját.

2.4. MySQL szerver

A MySQL[7] egy több felhasználós és több szálú SQL-alapú relációs adatbázis-kezelő szerver. A szoftver fejlesztője volt a svéd MySQL AB cég, amely kettős licencezéssel tette elérhetővé a MySQL-t; választható módon vagy a GPL, vagy egy kereskedelmi licenc érvényes a felhasználásra. 2008. januárjában a Sun felvásárolta a céget. A sors iróniája, hogy 2009. áprilisában a Sun-t felvásárolta az Oracle, ezzel szerintem az adatbázisok területén monopol helyzetbe kerülhet. Személy szerint 2002 óta használok MySQL szervereket webes alkalmazások készítésére.

2.5. PostGre szerver

Az egyik legjobb szabadforrás kódú adatbázis kezelő rendszer. Amikor döntenem kellet anno, hogy MySQL vagy Postgre[8] vonalon fejlesszek, én a MySQL-t választottam, mert könnyen telepíthető volt Microsoftos környezetre. Ez a különbség idővel eltűnt a két szerver közül. Ma pedig már úgy látom, hogy lényegtelen az adatbázis kezelő szerver típusa, ha léteznek jó kezelő eszközök és interfészek hozzájuk.

2.6. Szoftverek

A NetBeans egy integrált fejlesztői környezet, amely a Java nyelven alapul, jelenleg a 6.5.1-es verziója aktuális. A BOUML[9]-t a sourceforge.net-en találtam, amelyben a különböző UML diagramokat készítettem.

[7] Bővebben: http://www.mysql.com

[8] Bővebben: http://www.postgresql.org

[9] Bővebben: http://bouml.sourceforge.net/authors.html

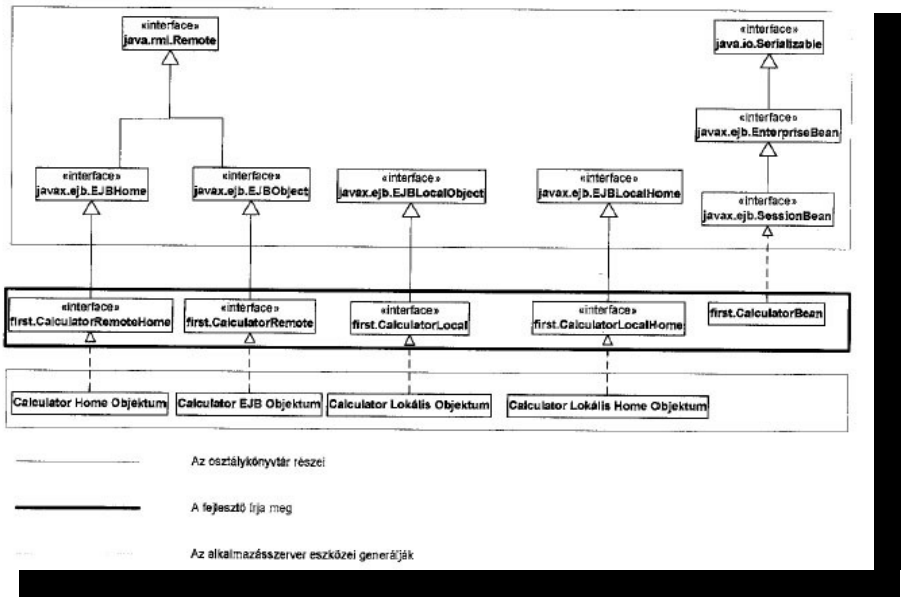
2.7. Hardverek

A fejlesztési időt lerövidítette, és a programozást nagyban megkönnyítette, a számítógépemben lévő videokártya vezérlő, mely kétfejes, és teljesen ki is tudtam használni úgy, hogy két monitort raktam rá.

2.8. Technológiák

2.8.1. EJB 3.0

Enterprise Java Bean [röviden EJB] egy összetett middleware-szolgáltatás, ami támogatja a többszálúságot a perzisztencia és tranzakció kezelést, a távoli elérést és az aszinkron üzenetküldést.



3. ábra Egy egyszerű EJB[10]

Egy EJB-komponens fizikailag egy jar fájlként jelenik meg, amelyben találhatók az alábbi alkotó elemek:

- Enterprise Bean osztály
- EJB-objektum
- Távoli interfész (kimaradhat, ha van lokális interfész)
- Home interfész (kimaradhat, ha van lokális interfész)
- Lokális interfész (kimaradhat, ha van távoli interfész)
- Lokális home interfész (kimaradhat, ha van home interfész)

[10] Imre Gábor: Szoftverfejlesztés Java platformon 18. oldal 2.5. ábra

- Telepítés leíró (Kötelezően META-INF/ejb-jar.xml)
- Gyártó specifikus fájlok

Az EJB technológia hátránya, hogy a fejlesztők által bonyolult volt, mert nehézkes volt a sok fájl karbantartása [minimum 5 fájl, három Java és két XML]. Mivel a bean osztálynak kötelező volt implementálni a MessageDrivenBean interfészt, ezért bizonyos callback metódusokat akkor is meg kellett valósítani, ha arra nem volt szükség. Egy EJB metódus hívása több sornyi kódot jelentet. Ezen kívül redundás volt a kód, mivel a Data Transfer Object-eket definiálni kellett, így az entity beanek perzisztens attribútumait meg kellett ismételni és a teszteléshez is szükség volt egy plusz EJB konténerre.
Az 3. ábra-n is jól látható, hogy bár a nevében egyszerű, azért nem voltak könnyen átláthatóak az első EJB-k. Ennek egyszerűsítését célozta meg az EJB 3 technológia, amely háromfajta EJB-t definiál, ezek az entity, session és message diven beanek. Bővebben erről projekttársam ír, mivel az ő feladat volt az EJB konténer kialakítása.

2.8.2. JPA, JDBC

Java Persistence API [továbbiakban: JPA] és a Java DataBase Connectivity [továbbiakban: JDBC] middleware-k a perzisztencia kezelést hivatottak megvalósítani. Mindkét middleware a Java 5-s verziójában sokat egyszerűsödött. Bővebben erről projekttársam ír, mivel az ő feladat volt az EJB konténer kialakítása.

2.8.3. Hibernate, Toplink

Alapvetően a Hibernate[11], Toplink technológiák az objektum orientált és a relációs szemléletmód közti kommunikációt hivatott megoldani. Erre azért van szükség, hogy Java objektumainkat tetszőleges adatbázisba menthessük. Bővebben erről projekttársam ír, mivel az ő feladat volt az EJB konténer kialakítása.

2.8.4. Szervlet

A Java szervlet olyan Java nyelven írt alkalmazás, amelyek a szerver oldalon tárolódnak, és a szerveren futnak. Legfrissebb változata a Java EE 5 részeként a Servlet 2.5. Szervlet működésének életciklusa három fázisra bontható:

- Inicializálás

[11] Bővebben: https://www.hibernate.org/

Kérés érkezése esetén, ha még nincs példányosítva a szervlet, a webkonténer betölti a szervlet osztályt, létrehoz belőle egy példányt, majd meghívja az init() metódust. Ekkor végrehajtható az inicializáció, a kiszolgáláshoz szükséges erőforrás lefoglalása.

- Kiszolgálás

Minden kérés érkezése esetén a szervlet service() metódusa hívódik meg, amely egy ServletRequest és egy ServletResponse típusú objektumot kap paraméterül.

- Eltávolítás

Ha egy bizonyos ideig nem érkezik kérés a szervlethez, akkor a webkonténer úgy dönthet, hogy eltávolítja a memóriából a szervletet, ehhez először meghívja a destroy() metódusát, így felszabadul az erőforrás.

2.8.5. JSP

A Java Server Pages [továbbiakban: JSP] a szerveleteket kiegészítő (és nem kiváltó) szerver oldali technológia, ami a szervletek gyengeségeit hivatott kiküszöbölni. A JSP-k előnye a szervletekkel szemben, hogy a Java kódba ágyazott HTML jelölő elem keveredését elkerülve, egyszerű szövegfájlként, statikus jelölőelemekkel képesek dinamikus tartalmat generálni. A dinamikus tartalmak generálása a technológiára jellemző, ún. JSP-elemek segítségével történnek. A JSP oldalak szervletté fordulnak le, és a JSP végrehajtásakor tulajdonképpen a hozzátartozó szervlet doXXX() metódusa fut le. A jspInit(), jspDestroy() pedig ugyanazokat a feladatokat látja el, mint a hagyományos szervleteknél az init() és destroy() metódusok.

JSP oldalaknál egy kérés érkezése esetén, az egy speciális, web konténer által biztosított szervlethez kerül, ami megvizsgálja, hogy a JSP oldal újabb-e, mint a belőle generált szervlet. Ha újabb, akkor a JSP oldalt (HttpServlet) szervletté transzformálja, az eredményt lefordítja, és annak továbbítódik a kérés

A JSP elemeknek három típusa van.

- Szkript elemek

Ezekben megírt Java kódot tartalmazni fogja az oldalból generált szervlet osztály. Megkülönböztetünk három féle szkriptlet elemet, melyeknek megadási formátumai:

  - Deklaráció

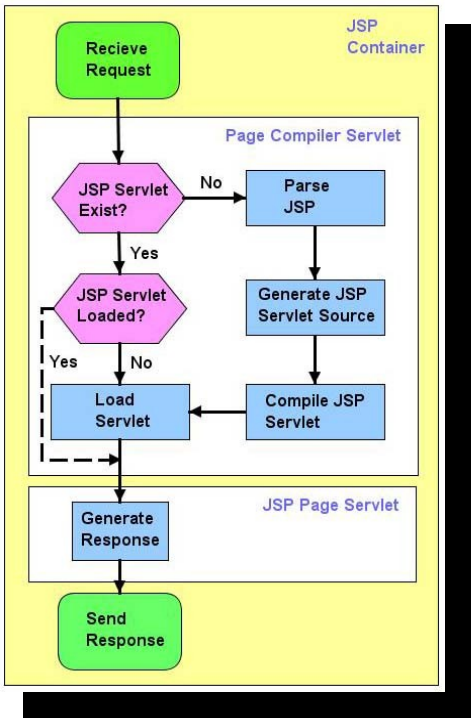
Szintaxisa: ***<%! deklaráció %>***. Az így deklarált változó vagy metódus az oldalból generált szervlet osztályba deklarációként kerül.

  - Kifejezés

Szintaxisa: ***<%= kifejezés %>***. A kifejezés tetszőleges Java utasítás lehet, ami a JSP-oldal végrehajtása során kiértékelődik és az érték sztinggé konvertálva bekerül a válaszba.

- Szkriptlet

Szintaxisa: ***<% kód %>***. A beírt programkód a generált szervlet doXXX() metódusába kerül bele, ezért nem tartalmazhat osztály vagy metódus információt. Keverhetőek statikus tartalommal.



4. ábra JSP működési folyamat.

- Direktívák

A JSP-oldal fordítását és végrehajtását szabályozzák, ezek a konténernek küldött üzenetek. Általános szintaxisa: ***<%@ direktívanév {attributúmNév = „érték”}* %>***. Háromféle direktívát különböztetünk meg.

- Include direktíva

Szintaxisa: ***<%@ include file = „relative URL” %>***. Hatására a relatív URL-en elérhető fájl tartalma statikusan bemásolódik a JSP oldalba.

- Taglib direktíva

Szintaxisa: ***<%@ taglib file = „tag könyvtár URI-ja” prefix=”prefix” %>***. Ez a direktíva deklarálja, hogy a JSP-oldal a felhasználó által definiált elemeket használ.

- Page direktíva

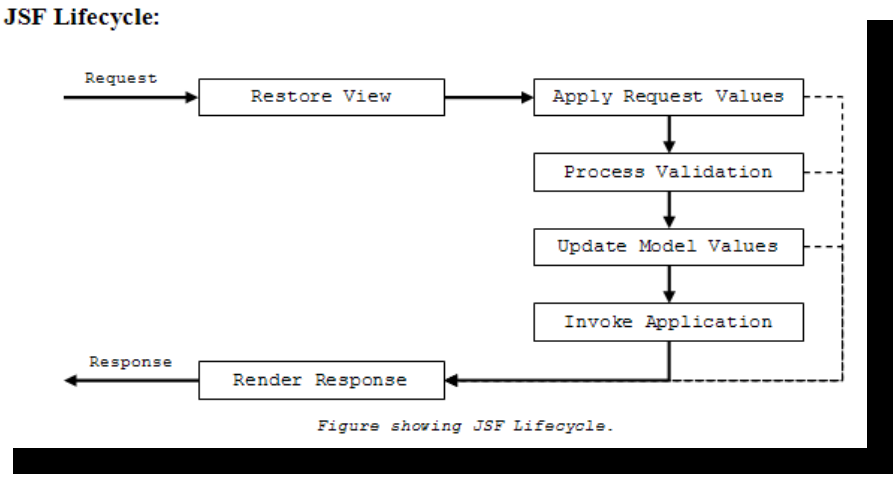
Szintaxisa: ***<%@ page jellemző= „érték” %>***. Segítségével az egész oldalra érvényes jellemzők értékeit állíthatjuk be.

- Akcióelemek

A generált szervletben JavaBeans komponensek vagy Servlet API metódusainak meghívásaként jelennek meg.

2.8.6. JSF

A Java Server Faces [továbbiakban: JSF] technológia az egyik leginkább elterjedt webes keretrendszer, köszönhetően annak, hogy épít a JSP technológiára és MVC tervezési minta alapján dolgozik, valamint számos Java EE - alkalmazás szerver és fejlesztői környezet tartalmazza. A specifikációjának két elterjedt változata van, de jelenleg a 1.2-t használjuk (JSR-252).



5. ábra JSF életciklusa

A JSF minden kérésfeldolgozásnál egy állapotgépet használ, ami vezérli a folyamatot és 6 fő állapotot tartalmaz.

- A nézet visszaállítása (Restore Wiew)

A keretrendszer első lépésben eldönti, hogy olyan oldalra került-e, amelynek állapotát már elmentette-e már [továbbiakban: postback]. Ha postback, akkor ebben a fázisban a JSF visszaállítja az egyes komponensek állapotát. Ha nem postback, akkor a keretrendszer létrehozza a komponensfa gyökerét, majd a JSF kérésfeldolgozási állapotgépét a nézet generálása állapotba kényszeríti.

- A kérés paramétereinek rögzítése (Apply Request Values)

Ebben a fázisban minden input komponens ellenőrzi, hogy nincs-e hozzárendelve paraméter a kérésben. Ha van, akkor azt az UIInput ősosztály submittedValue tagváltozójában tárolja el.

- Validációk futtatása (Process Validation)

Ekkor a ViewRoot-tól kezdve mélységi bejárással történik a komponensfában található komponensek validálása. Ebben a fázisban csak azok a komponensek kerülnek validálásra, amelyekre igaz, hogy:

  - a felhasználó által küldött űrlapban szerepelnek
  - a kérésben megtalálható a hozzájuk tartozó paraméter
  - a rendered attributuma true
  - az immediate attributuma false

Validálási folyamat az, hogy először a kapott paramétert, ha van (ha nincs, és required, akkor hiba!) konvertálni kell a megfelelő Java-típusra, egy JSF konverter segítségével (lehet egyedit is megadni). Ezután a komponenshez tartozó validátor(oka)t le kell futtatni. Amennyiben több validátor van, akkor addig fut a validálás, amíg egy hibát nem jelez, vagy be nem fejezi. Ha hiba van, akkor a JSF elhelyez egy hibaüzenetet a komponens üzenetei között és a nézet generálása részre ugrik az állapotgép. Ha a validálás sikeres, akkor a konvertált és validált értéket beírja a komponens value tagváltozójába.

- A modell aktualizálása (Update model Values)

Ebben a lépésben a konvertált és validált értékek beírása megtörténik a modellbe.

- Alkalmazás meghívása (Invoke Application)

Ekkor az események továbbítása a komponenseken keresztül az összes regisztrált osztály felé megtörténik. Egyetlen kivétel van, éspedig az immediate attributum igaz esetén az alkalmazás hívása nem itt, hanem az első fázisban történik.

- Nézet generálása (Render Response)

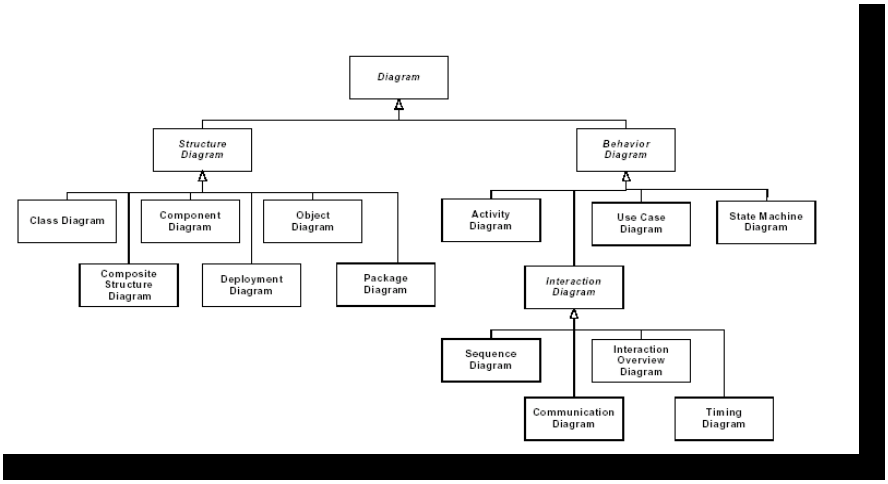
Itt a felépített komponensfát bejárva a beállított renderkit segítségével a rendszer legenerálja a szükséges kimenetet.

2.8.7. Log4j

Nyílt forráskódú keretrendszer 1999-ben jelent meg először. Gyors elterjedését a rendkívül egyszerű implementálhatóság és XML fájllal konfigurálhatóság csak segítette. Nekem korábbi Java alkalmazásfejlesztéseimnél már bevált, úgyhogy itt is javasoltam csoporttársamnak, hogy ezt használjuk loggolásra.

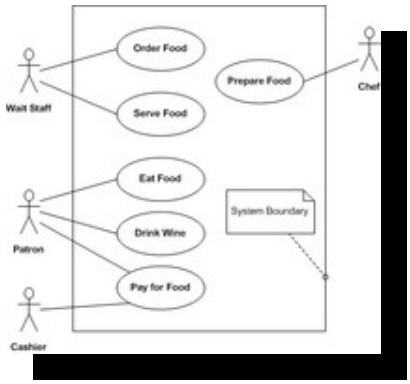
2.8.8. UML

A Unified Modeling Language [továbbiakban: UML] az objektumorientált programozás szabványos specifikációs nyelve. Az UML grafikus jelöléseket használ rendszerek absztrakt modelljének leírására. A szabvány UML jelölést használó diagramokat UML modelleknek nevezzük. 1994-ben fejlesztették ki, Grady Booch, Jim Rumbaugh és Ivar Jacobson közreműködésével. 1997-ben az Object Modeling Designs [továbbiakban: OMD] saját szabványává teszi az UML1.1-est. 2002-től az UML 1.5-be már olyan eszközök kerülnek bele, amelyek lehetővé teszik, hogy abból a modellből, amit UML által formalizálunk, abból kódot lehet generálni. Jelenleg, 2005-től az UML2-t használjuk, diagram felépítése:



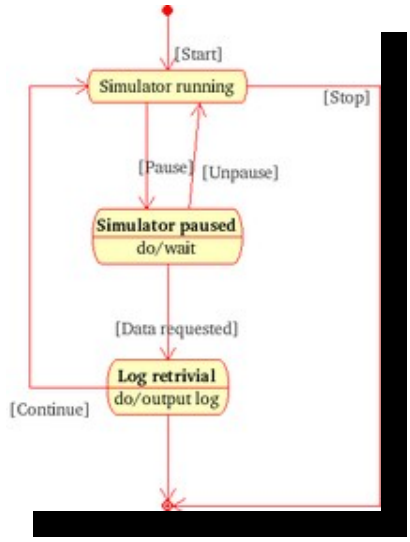
6. ábra UML diagramok térképe

Az osztálydiagram egy statikus modell. A rendszerben használt osztályokat mutatja azok attribútumaival együtt. Az osztálydiagram tartalmazza továbbá az osztály szintű kapcsolatokat. A komponensdiagram a rendszer fizikai komponenseit és az azok közötti függőségeket mutatja. Az összetett struktúradiagram az osztályok belső szerkezetét mutatja és azt, hogy az adott szerkezet milyen kollaborációkat tesz lehetővé. A telepítési diagramok a rendszerimplementációhoz használt hardvert, a hardverre telepített szoftverkomponenseket és azok viszonyát hivatottak reprezentálni.



7. ábra Használati eset diagram

Az objektumdiagramok pillanatfelvételek a rendszer állapotáról. Osztályok példányait és kapcsolatait jeleníti meg. Az objektumdiagram konkrétabb az osztálydiagramnál, mert objektumok példányainak a kapcsolatát írja le objektumosztályok kapcsolata helyett. A csomagdiagram azt mutatja, hogy hogyan szerveződnek a szoftverelemek csomagokba illetve hogyan viszonyulnak ezek a csomagok egymáshoz. A viselkedési diagramok azt írják le, hogy minek kell történnie a modellezett rendszerben. Az aktivitásdiagramok a munkafolyamatot modellezik.



8. ábra Állapotgép diagram

Az állapotgép diagramok a rendszer lehetséges állapotait és az azok közötti átmeneteket mutatják állapotgépes ábrázolással. A use case diagramok fogalmazzák meg a rendszer használati eseteit. Az interakciós diagramok fogalmazzák meg a rendszerelemek közötti kommunikációt.

2.8.9. HTML 4.0

A HyperText Markup Language [továbbiakban: HTML] statikus weboldalkészítés alapjául szolgáló nyelvezet. Egy olyan nyelv melyet a web, a hiperszöveges adatbázis oldalainak leírásaihoz használ fel. Jelölőnyelv. A HTML az SGML (Standard Generalized Markup Language) séma továbbfejlesztéseként került kialakításra. A szöveges információk mellett képek és más multimédiás tartalmak oldalakba illesztésének, valamint az oldalak közötti hiperkapcsok kialakításának lehetővé tevője. A HTML dokumentum kiterjesztése htm vagy html lehet. A file formázóutasításokat, parancsokat tartalmaz [továbbiakban: TAG], melyek befolyásolják a későbbiekben a kliens oldalon megjelenő web-oldal kinézetét.

A HTML-t 1990-óta használják a weben a honlapok megjelenítéséhez. A HTML 1.0-nál alakult ki az alapvető szerkezet, a nyitó és záró címek „***<html></html>***”, a fejléc, és a törzs, azaz a „***<body></body>***”. Ezeknek, az elemeknek a zárócímkéje nem elhanyagolható a mai napig sem. Ekkor már lehetőség volt dokumentum típus deklarációra (DTD), és létre lehetett hozni bekezdéseket, hivatkozásokat, fejléceket, felsorolást, amit mind a mai napig használunk is.

HTML 2.0-tól különül el a böngésző által szabályozott és a dokumentum szerkezetében létrehozott formázás. Új (formázási) lehetőségeket építettek a nyelvbe, pl.: „***<code>***”, „***<EM>***”, „***<b>***”, „***<i>***” és a „***<TT>***”. Ezután rögtön bevezetésre kerültek az űrlapok, az űrlapon belül a többsoros szövegbevitel, és a kiválasztható opciók. Ez a verzió már szabvány lett, a későbbi fejlesztések alapjául is szolgált.

A HTML 3.2 verziót 1996-ban fogadta el a W3C. Ekkortól van lehetőségünk Java alkalmazások beágyazására, és scripteket is innentől használhatunk. Ekkor jelent meg a "style" elem is. Ebben a verzióban már lehetőség volt táblázatok készítésre, a betűtípust beállítani.

A HTML 4. 0, amit a szakdolgozatban is használok, 12 éve 1997 nyarán tette hivatalos ajánlássá a W3C, megfelel az ISO 8879 előírásainak. Ez már igazából egy SGML alkalmazás. Fejlesztésekor szem előtt tartották a csökkent képességűek érdekeit, a nemzetközi karakter készleteket és a jobbról balra olvasás támogatását. Tovább fejlesztették az űrlapok és táblázatok használhatóságát, a keretek [frame-ek] használata hivatalossá vált.

Ezután új irányt vett a HTML fejlődése, kiterjesztették úgy a HTML 4.0-t, hogy bevonták az XML-t a nyelvbe, így létrejött a XHTML1.0.

2.8.10. CSS 2.0

A Cascading Style Sheets [továbbiakban: CSS] teljes és totális kontrollt nyújt a hipertext típusú dokumentumok kinézetéhez. Ez egy olyan leírónyelv, melynek segítségével különböző stíluslapokat hozhatunk létre és ágyazhatunk be HTML honlapjainkba. Ezek a stíluslapok befolyásolják az oldal megjelenését: meghatározhatjuk velük, hogy az egyes HTML elemek mekkora méretben, hol és milyen színnel és stílusban jelenjenek meg a felületen. A CSS stíluslap lehet külön fájlban [lásd: ./design/default.css], lehet az aktuális fájlban, és lehet közvetlenül egy tag elem attribútuma [lásd: style="…"] ként is elhelyezni. Abban az esetben, ha a stíluslap külön file-ban van, akkor arra hivatkozni kell az adott hipertext ***<head>***fejlécében***</head>***. Szintén a head-be tesszük a stíluslapot, ha az aktuális file-ba akarjuk helyezni az aktuális file-ra vonatkozó teljes formázást. Azonban tehetjük közvetlenül az elembe [például: *** ***Ez egy bekezdés***! ***] is. Most hogy már tudjuk, hogy hova lehet helyezni stíluslapokat, fontos hogy tudjuk, mindezeknek van prioritása is, melyek a felsorolásnál csökkenő sorrendben vannak elhelyezve.

1. A közvetlenül attribútumként megadott stílusok.
2. A head részben definiált stíluslapok.
3. A külső stíluslapok.
4. A böngésző alap beállításai.

A stíluslap minden eleme kijelölőből [selector] és meghatározásból [declaration] áll.

Általános meghatározás:

```
p {
   font-size:Arial, Helvetica, sans-serif;
   background-color:#FFDEAD;
   color:#000080;
}
```

9. ábra CSS kódrészlet tag formázására

A hátterét, betűtípusát, valamint ezek színeit határozza meg *** ***közötti résznek*** ***. Ez a típusú utalás a dokumentum összes *** *** elemére vonatkozik.

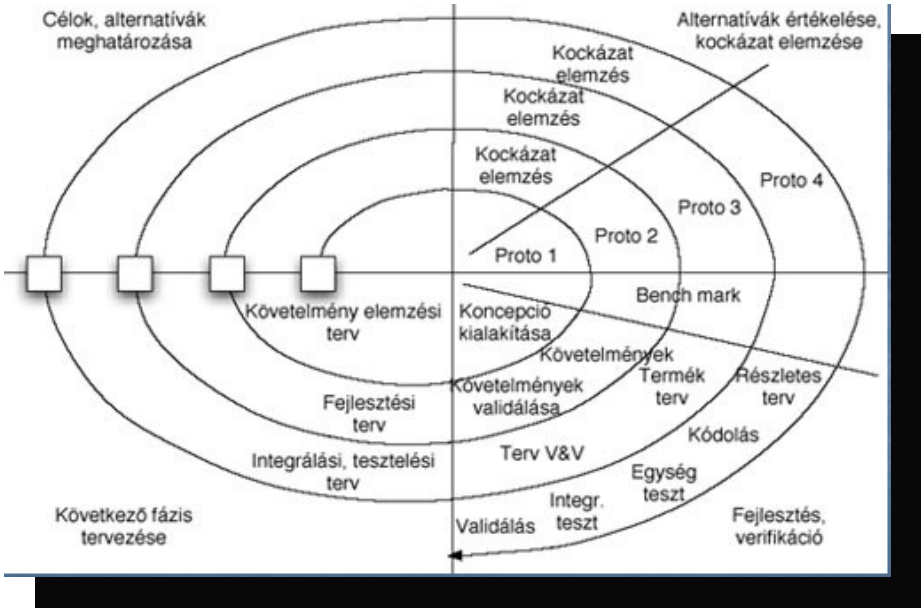
```
#header {
    position:absolute;
    top:0px;
    left:0px;
}
```

10. ábra CSS kódrészlet jelölő bemutatására

Így lehet azonosító kijelölőket is készíteni.

3. Tervezett és megvalósult szoftverfejlesztési folyamat

3.1. Tervezett modell a Spirál



11. ábra Spirál model

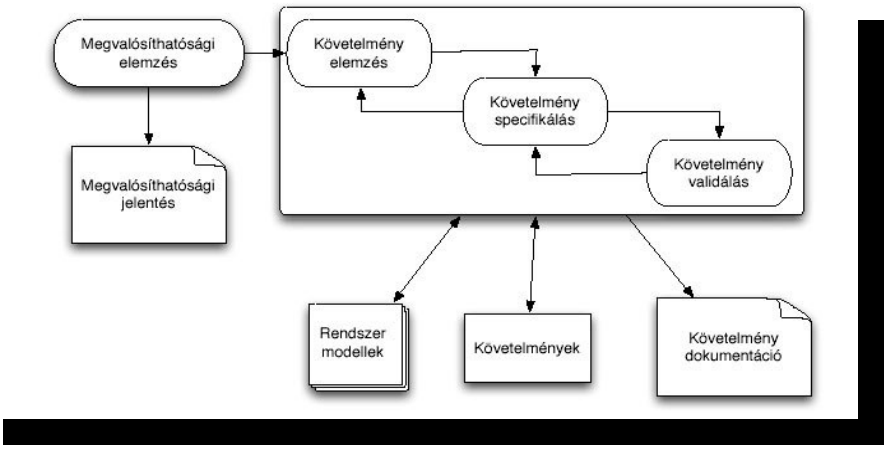
Miután átnéztük a követelményeket, úgy döntöttünk, hogy a szoftverfejlesztés a spirál modellt fogja követni. Ez a modell Barry Boehm professzor nevéhez fűződik, aki a Harwardon végzett. Újdonság a korábban kialakult modellekhez képest a kockázat kezelés. A modell szerint a fejlesztés iterációs lépések sorozata, ahol az egyes iterációkban kitűzött célok folyamatosan fejlődnek és valamennyi fázis ciklikusan változik. Ha a modellt ábrázoljuk, akkor megkapjuk a spirálhoz hasonlatos alakzatot. Fontos, hogy minden rész megoldását, részmegoldását ki kell értékelni és elemezni kell az adott megoldás kockázatát. Ha a kockázat kisebb, mint a várható haszon, akkor következhet a következő ciklus.

Elképzeléseinkben az szerepelt, hogy első fázisban megvalósítjuk a felhasználó kezelést a jogosultság rendszert és a családok adatait, majd később kiterjesztjük a rendszert különböző statisztikai modulokkal, illetve bővülne még más területek Nevelőszülői adatbázisával.

3.2. Megvalósult modell

A modell, amit megvalósítottunk a vízesés modellre emlékeztet és az alábbi fázisokon ment végig.

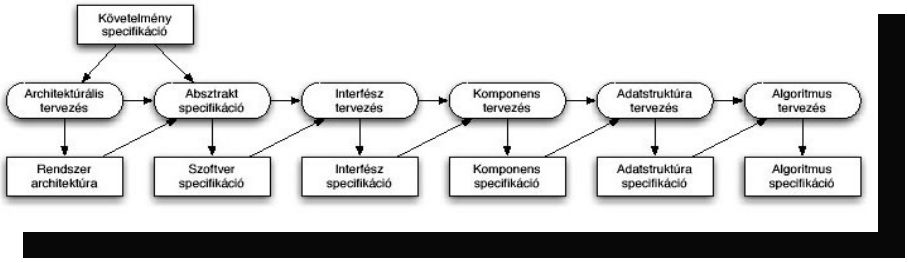
3.2.1. Követelmény meghatározás



12. ábra Követelmény meghatározás

Definíció szerint a követelmény meghatározás része a szolgáltatások meghatározása. A működési és fejlesztési környezet megismerése. A követelmény meghatározási folyamat szakaszai a megvalósíthatósági elemzés, követelmények elemzése, spacifikálása és végül validálása. A *4.1 Követelmények specifikálása* résznél látható, hogy több levélváltás és személyes találkozó is volt a megrendelővel mire kialakult a végleges követelmény specifikáció.

3.2.2. Szoftver tervezés



13. ábra Szoftver tervezés folyamata

A specifikációt megvalósító szoftver struktúra megtervezése. Mivel a szoftver (JAVA) adott volt, ezen a ponton a használati eseteket vettük át és készítettünk megfelelő diagramokat. Ezzel modellezve a szoftver működését.

3.2.3. Implementáció

A szoftver struktúra futtatható szoftverré alakítása. Ez a rész volt a legnehezebb, hiszen rengeteg számunkra új technológiával kellett megismerkednünk.

3.2.4. Szoftver validálás

Ennek a fázisnak feladata, hogy ellenőrizze, a szoftver megfelel a specifikációjának, azaz a szoftver megfelel-e a megrendelő igényeinek.

3.2.5. Tesztelés, Elfogadási teszt

Egyik legfontosabb rész ez, hiszen feltár(hat)ja az esetleges szoftver hibákat. Amikor a hibákat javítottuk következhet a szoftver és működési környezetének beüzemelése a megrendelőnél és a szoftver átadása.

4. A szoftverfejlesztés

4.1. Követelmények specifikálása

A szoftverfejlesztési modellt követve kértünk specifikációt a TEGYESZ vezetőitől. Első körben scannelt fájlként e-mailben kaptuk meg (a függelékben található Követelmények meghatározása dokumentumot. Ezt átnézve készítettünk egy szó/fogalomtárat ebből a függelékben lehet látni részletet (50. ábra *Részlet a fogalomtárból*). Majd több körös megbeszélések végén elkészítettük a saját specifikációnkat.

4.2. Tervezés

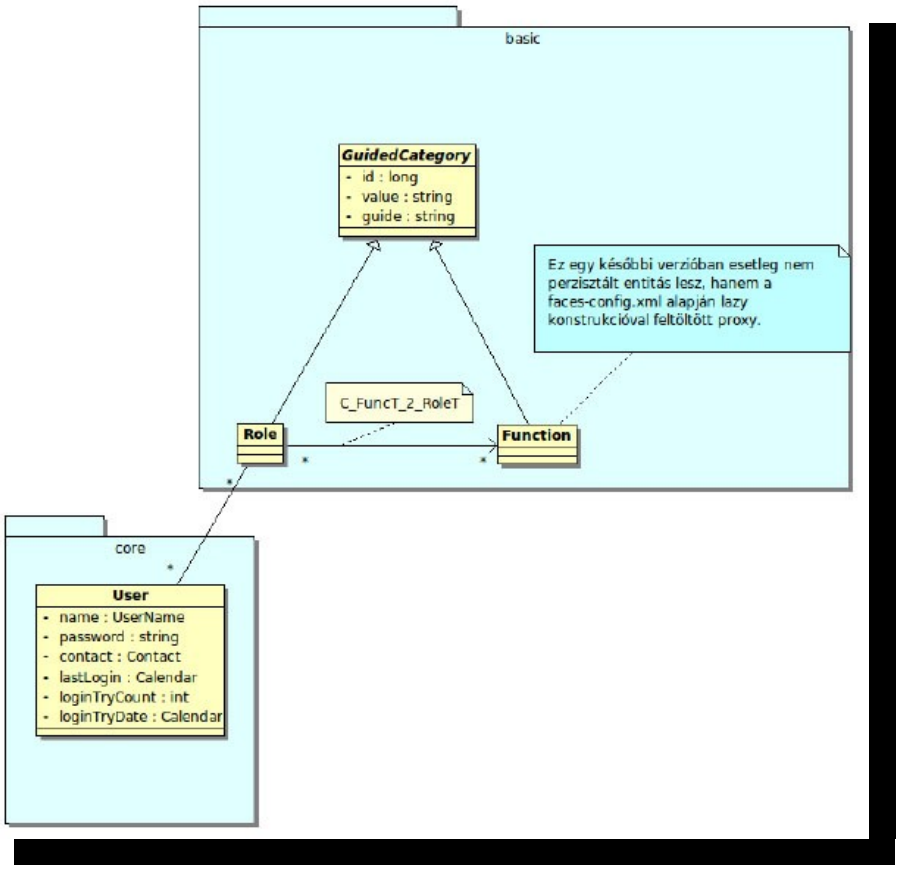
Ebben a fázisban elkészítettük a tárolandó adatokról és a használati esetekről BOUML-ben a diagramokat. Kialakult a csomagdiagram, a projektet TEGYESZ-nek neveztük el, ez két fő részre bontottuk, követve a Java EE Vállalati környezeti ajánlást, EJB és WAR csomagra, értelemszerűen létrejött a *tegyesz.ejb* és *tegyesz.war* csomag.
Az entitások a *tegyesz.ejb.entity* csomagba kerültek, a session bean-ek a *.tegyesz.ejb.manager* csomagba, a *tegyesz.ejb.core* csomagba működéshez szorosan kapcsolódó osztályok. Az EJB

3.0-hoz még tartozó message diven beanek nem készültek, mivel első körben egyetlen szerveren fog futni az alkalmazás, és nem fog kommunikálni más szerverekkel.

A war csomag felépítése *tegyesz.war.mbean* csomagban találhatóak a JSF oldalakhoz tartozó menedzselt beanek. Speciális utilok, melyek közvetlenül nem kapcsolódnak az üzleti logikához, hanem egyéb funkciót látnak el, pl. nemzetköziesítés vagy jogosultság figyelés a *tegyesz.war.util* csomagban kaptak helyet. Ezen kívül a war csomagban találhatóak a JSF fájlok, jsp kiterjesztéssel és egy /design alkönyvtárban a grafikai elemek, főként css.

4.3. Osztály diagramok

4.3.1. Törzsadatok - entity

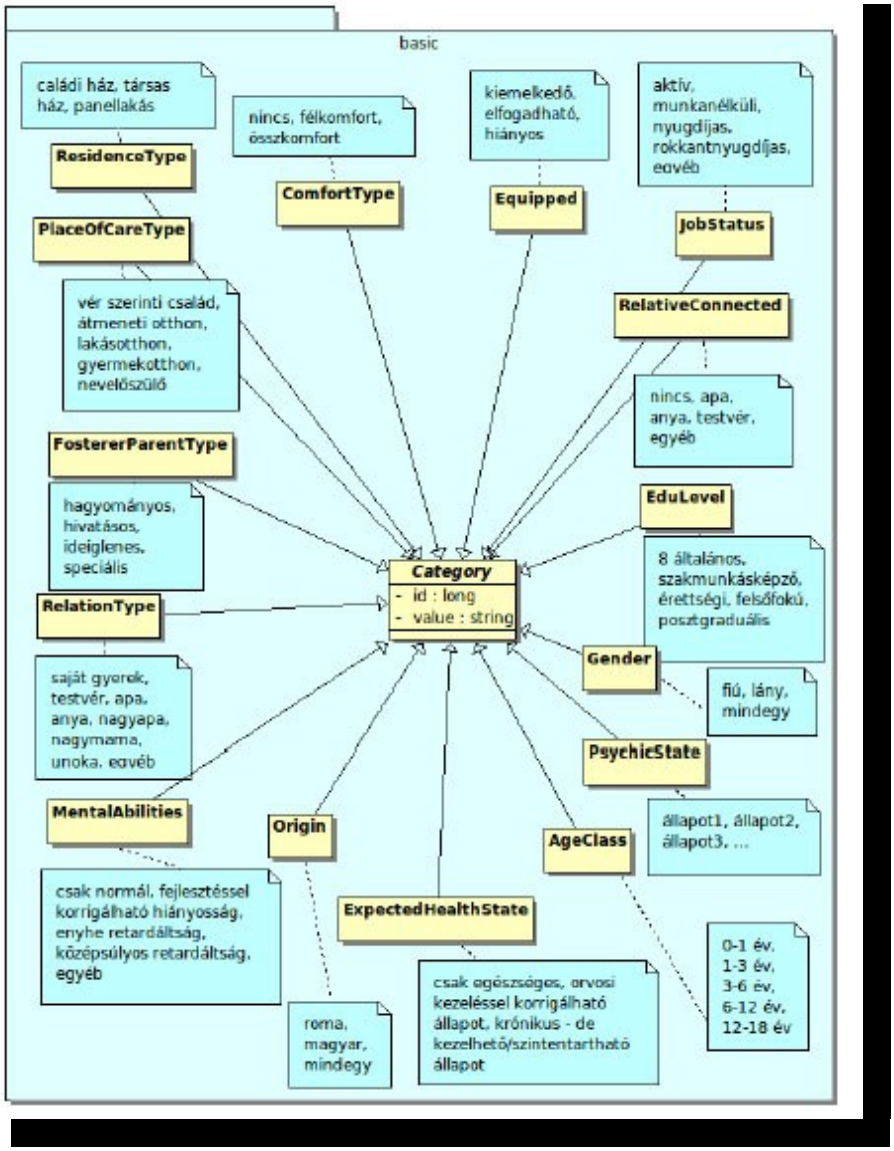


14. ábra Jogosultság kezelés osztály diagramja

Először a kategóriákat mutatom be, hiszen ezek a törzsadatok.

A 15. ábra Törzsadatok osztálydiagramja érdekessége, hogy minden osztály egyetlen absztrakt osztály, a category leszármazottja. A Category osztály egy ID-t és Value-t tartalmaz. Implementáláskor ezzel sokat küzdöttünk, hogy hogyan lehetne úgy perzisztálni, hogy ezeket az osztályokat egyetlen táblába képezze le a hibernate és a unique megmegszorítás a value és

az osztálynév legyen. Sajnos, ezt nem sikerült megoldanunk, így külön táblákba tároljuk az adatokat.



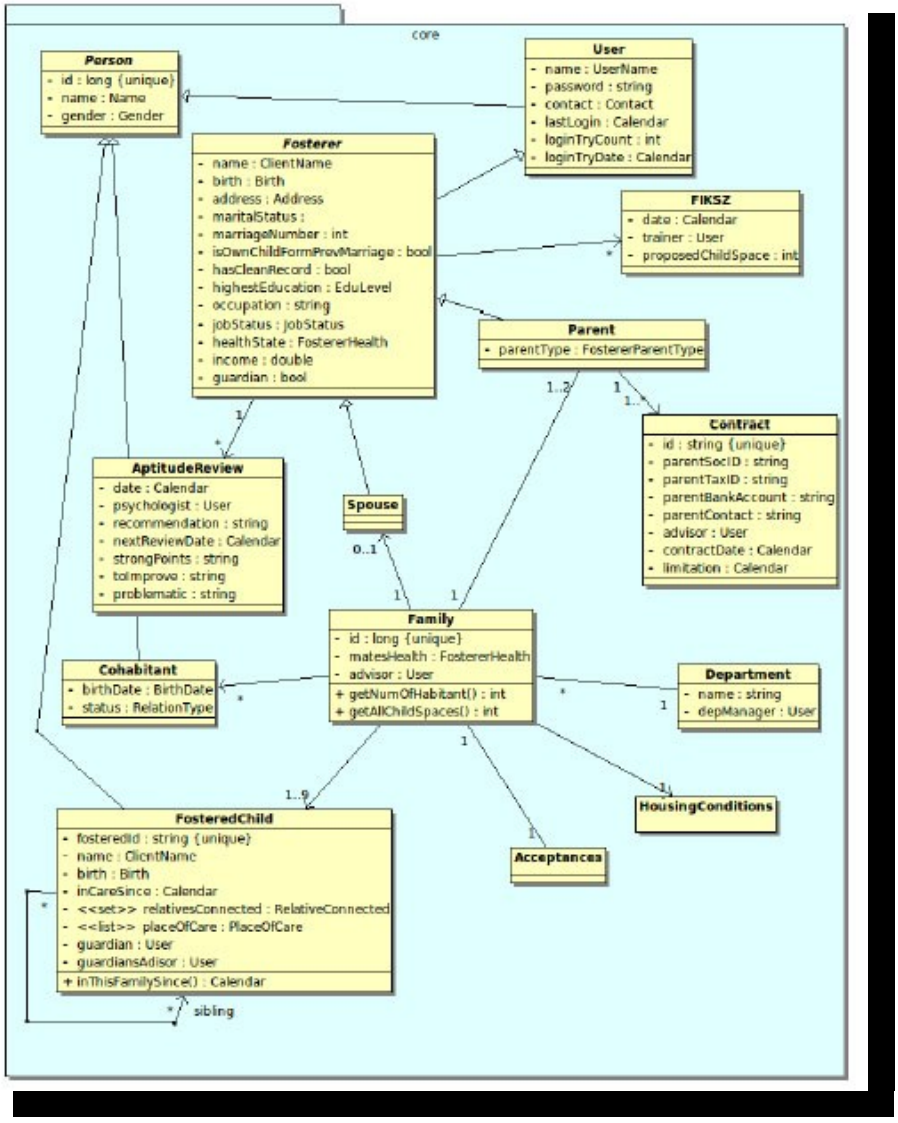
15. ábra Törzsadatok osztálydiagramja

Igazából még törzsadatnak számít, de a rendszer működésétől elválaszthatatlan a jogosultságrendszer kezeléséhez szükséges szolgáló osztályok. Éppen ezért ezeket külön csomagba raktuk, bár utólag a csomagnév választás nem volt szerencsés. Ennek a csomagnak a neve *tegyesz.ejb.basic*. Felesleges volt őket külön rakni, hiszen egy későbbi verzióban tervben van, hogy a rendszer automatikusan legenereálja a functions osztályhoz tartozó objektumokat. Így a fejlesztőnek arra sem kell figyelnie, hogy beregisztrálja a megjelenítéshez és szabályzáshoz szükséges metódusokat ennek az osztálynak leképezett

táblájába. A 14. ábra Jogosultság kezelés osztály diagramja-ban látható user osztály nem része a csomagnak, csak a kapcsolódás megértése miatt szerepel az ábrán.

4.3.2. A működési magot kiszolgáló osztályok - core csomag

Ebbe a csomagba az alkalmazás magját képező osztályok kerültek. Sokáig próbáltuk egy oldalon feltüntetni a teljes diagramot, de aztán ezt feladtuk, mert a BOUML alkalmazás kereteit feszegettük ezzel. Így a jobb átláthatóság kedvéért három diagramon készült el a teljes terv.



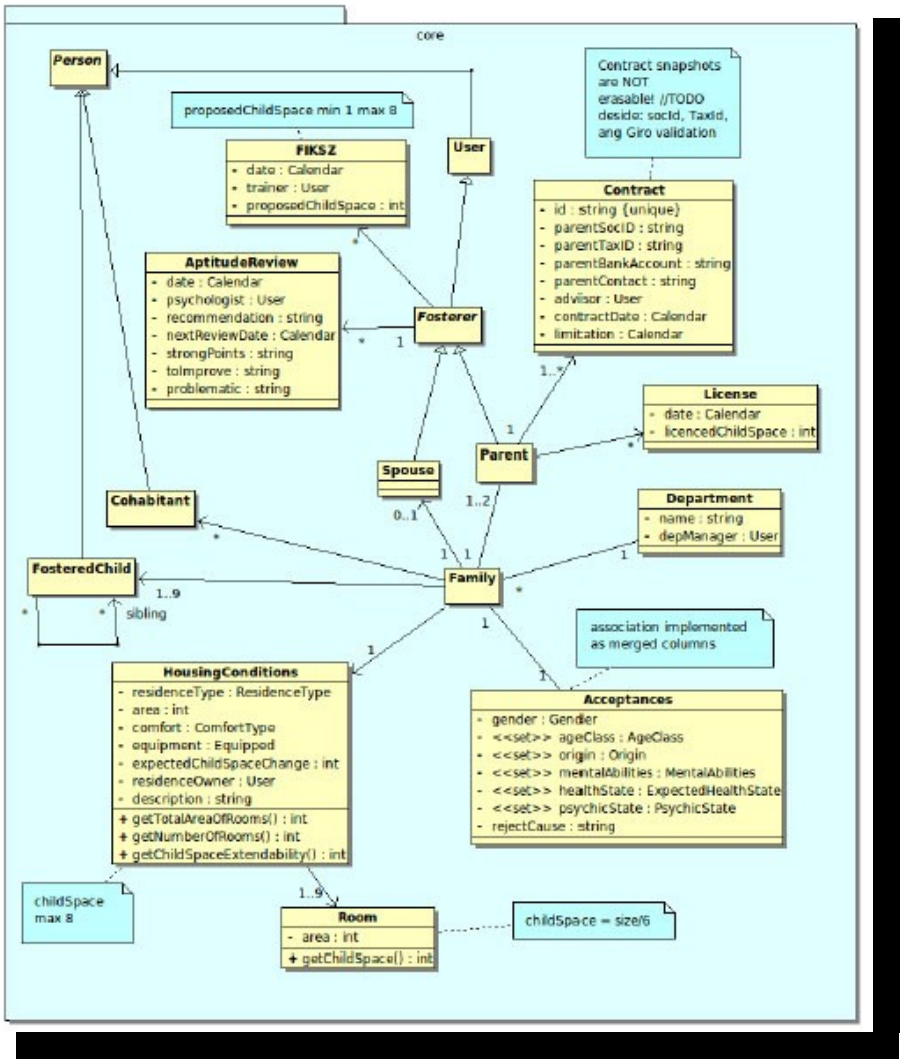
16. ábra core csomag első osztálydiagram

A 16. ábra core csomag első osztálydiagram-on a személyekhez tartozó osztályok láthatóak teljes egészében. Jól látható, hogy a person és a fosterer osztályok absztrakt osztályok. Az is

érzékelhető, hogy a rendszer elő van készítve arra is akár, hogy ne csak a tanácsadó és a területi vezető használhassa a rendszert, hanem a nevelőcsaládok is elérhessék. Természetesen ez a jogosultsági beállításoktól is függ.

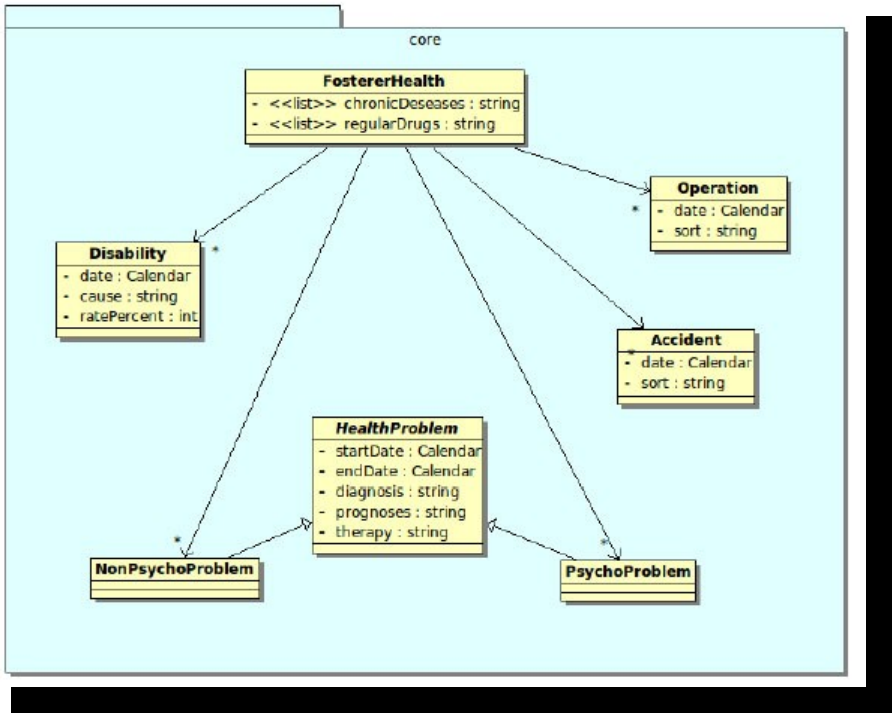
A family osztály két metódusa szolgálja azt a célt, hogy előre meghatározott szabályok alapján számolt érték azonnal rendelkezésre álljon. getNumOfHabitant() megadja a családban együttélők számát, getAllChildSpace() megadja a családnál elhelyezhető gyermekek számát.

A többi osztály az (egészségügy kivételével) a 17. ábra kapcsolódó osztálydiagram-on látható. Ezen az ábrán jelöltük az integritási megszorításokat, amelyekre különösen oda kell figyelni. A room osztályhoz tartozó getChildSpace() metódus megadja egy adott szobában elhelyezhető gyermekek számát. A HausingCondition metódusai getTotalAreaOfRooms() a teljes lakás területét, a geNumberOfRooms() a szobák számát míg a ChildSpaceExtendibility() megadja a lakásban elhelyezhető gyermekek számát.



17. ábra kapcsolódó osztálydiagram

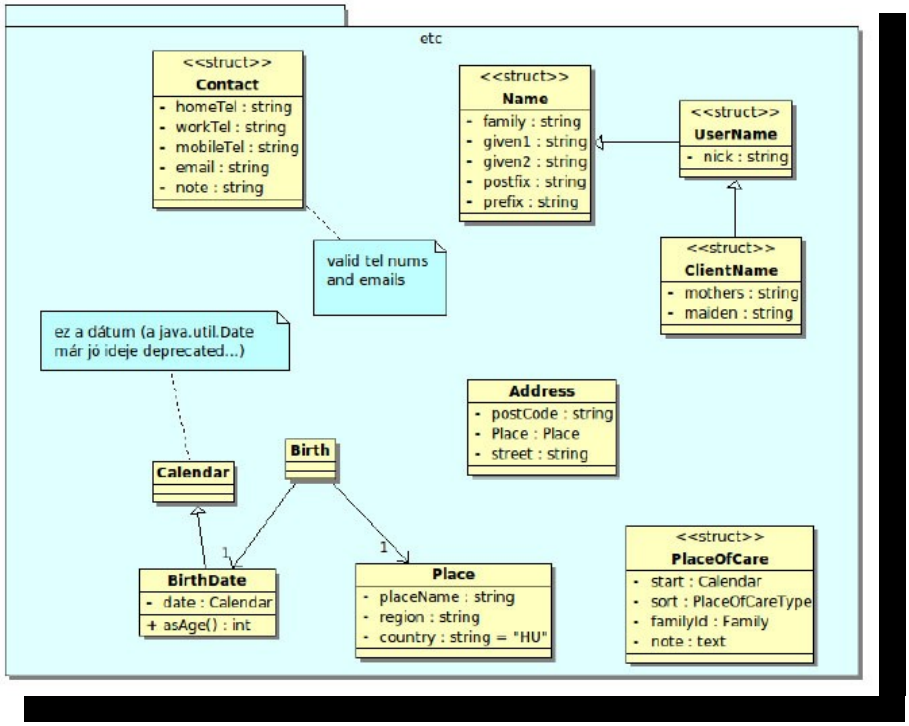
Egy részt külön is kiemeltünk, mivel komoly fejtörést okozott a TEGYESZ vezetőinek, hogy ezen adatok közül mihez van joguk, hogy tároljanak. Ez pedig a nevelőszülők egészségi állapotával foglalkozó osztálydiagram (18. ábra). Ezen a diagramon a fostererHealth és a HealthProblem osztályok absztraktak. A accident és a diability kórtörténeti képet is tükröz, ezért ezek 1-n kapcsolatban vannak a fostererHealth osztállyal.



18. ábra egészségügy

4.3.3. Segédosztályok – etc csomag

Ide szorultak ki azok az osztályok, amik nem alap osztályok, de nem is a működés szerves részéhez tartoznak. Látható, hogy pl. a contact osztályt lehetet volna még bontani, hiszen a homeTel, workTel és a mobileTel azonos típusú tartalmak. A projekttársam elvetette a javaslatomat, hogy ez legyen, pedig akkor bármennyi telefonszámot fel lehetet volna venni. Ebből is látszik, hogy mennyire aprólékosan igyekeztük tervezni a rendszert. A birthDate asAge() metódusa a pontos életkort adja vissza az aktuális dátum alapján.



19. ábra Segéd osztályok

4.4. Használati esetek

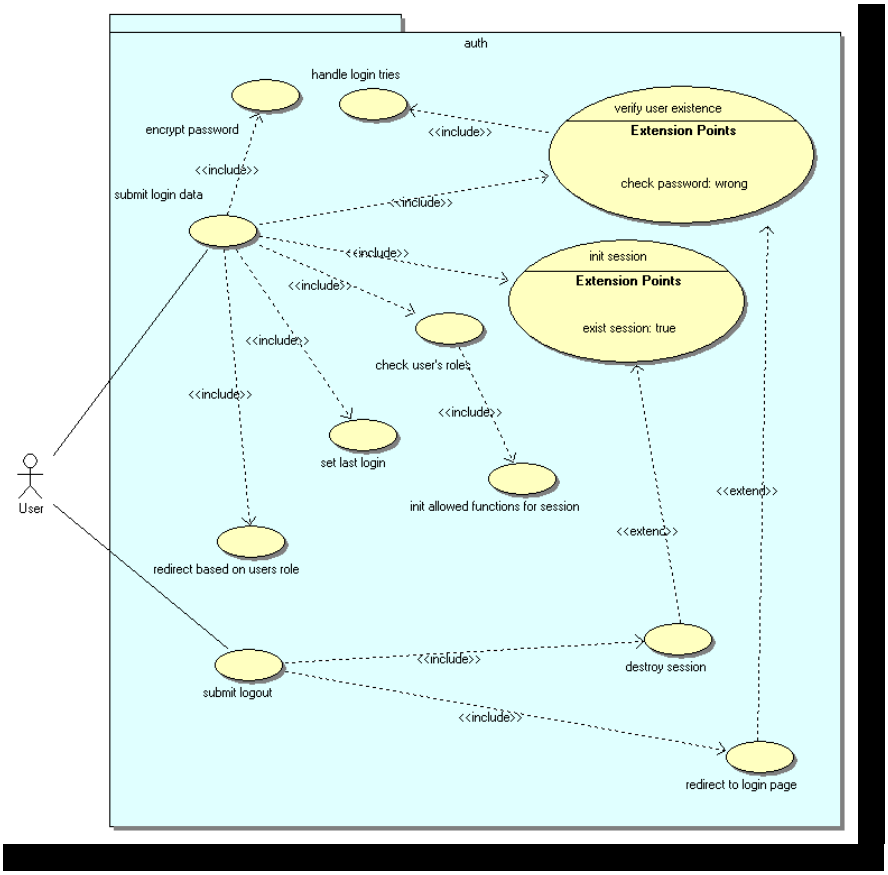
A rendszer tervezése során a használati eseteknél igyekeztük az üzleti logikát úgy tervezni, hogy az egyes használati esetek egy-egy JSF oldalon jelenjenek meg.

4.4.1. Autentikáció

A belépési folyamat úgy lett tervezve, hogy minden user, akinek joga van használni a rendszert, ugyanazokon a eseményeken keresztül tud eljutni az induló képernyőhöz.

A belépési oldalon (az üdvözlő szöveget, esetleg híreket elolvasva) meg kell, hogy adja felhasználói nevét és jelszavát a felhasználónak, majd postolja a szervernek. Ezután a jelszó kódolása [encrypt password] történik, hiszen kódolva tároljuk ezt az adatot, majd ellenőrizzük a felhasználó érvényességét [verify user existence]. Ezen a ponton lehet egy nem mindig lefutó esemény, mégpedig az, ha hibás a jelszó, ezt exteinson pontként jelöljük az ábrán. Ekkor megszüntetjük a sessiont, ha van, és megjelenítjük a belépési képernyőt. Ezen kívül, hogy törési próbálkozások ellen védjük a rendszert, terveztünk egy olyan részt, ami a belépési próbálkozásokat tartja nyilván [handle login tries]. Működési elv az, hogy a sikertelen próbálkozáskor, ha ez az első, akkor beírja a próbálkozást a felhasználóhoz, mint infó. Ha adott időn belül [5 perc] több hibás próbálkozás [3 db] van, akkor letiltja a felhasználót egy

időre [1 óra], azaz nem engedi be a rendszerbe, még akkor sem, ha eltalálja a jelszót. Ezt finomítandó később lehet belekódolni ebbe a részbe egy képre írt szöveges ellenőrzést, de idő és erőforrás hiányában ezt most inkább elhagytuk.
Ezután újra próbálkozhat a belépéssel. Ha sikeres a jelszó megadás, akkor 0-za a rendszer a belépés próbálkozást nyilvántartó mezőket.

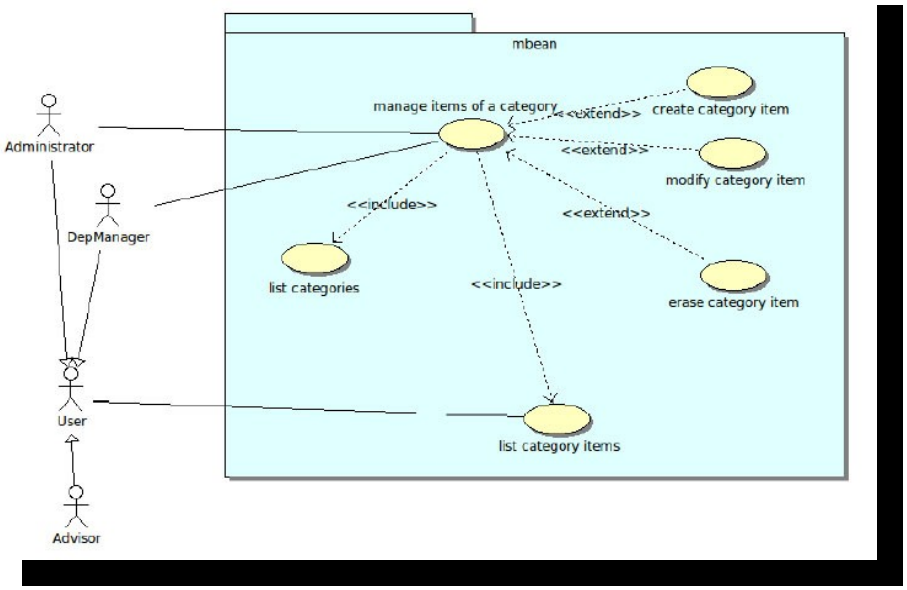


20. ábra felhasználó be és kiléptetése

Ha sikeres a belépés, akkor indít a rendszer egy session-t a felhasználónak. Itt azt vizsgáljuk, hogy létezik e már élő session, ha igen, akkor megszünteti a korábbit, és újat indít.
Ezután ellenőrizzük a felhasználó jogait, és beállítjuk, hogy mit jeleníthet meg a program részére. Majd beállítjuk az utolsó bejelentkezés dátumát az aktuális időre. Ezzel végeztünk is az ellenőrzési folyamattal. Utolsó lépésként redirect-tel eljuttatjuk a felhasználót az induló oldalra.
A másik folyamatsor rövidebb, ez pedig a kilépés. Ha a felhasználó erre a vonalra kerül, megszüntetjük a session-jét, és visszairányítjuk a login oldalra.

4.4.2. Kategória kezelés

Ez a use case a törzsadatok kezelésével foglalkozik. Szerepkörök szerint új törzsadat elemet csak az adminisztrátor és a területi vezető vihet fel, a többi felhasználó csak listázhatja azokat. Terveink szerint a törzsadatokat, mivel egy absztrakt osztályból a kategóriából épülnek fel, ezért egy táblában tároltuk volna, melynek egy attribútuma a törzsadat típusa lett volna. [Azonban, mint korábban említettem egy speciális unique megkötés miatt ezt nem tudtuk megvalósítani.] Így a kezelő felületet is úgy terveztük, hogy ezeket együtt, egységesen lehessen kezelni, mint később ez a megoldásban látszani fog, ezt sikerült is megvalósítanom.
A teljes folyamat terv szerint úgy néz ki, hogy az oldalra érkezve a felhasználó (esetünkben ugye az adminisztrátor vagy a területi vezető) listázza a törzsadatok típusait.



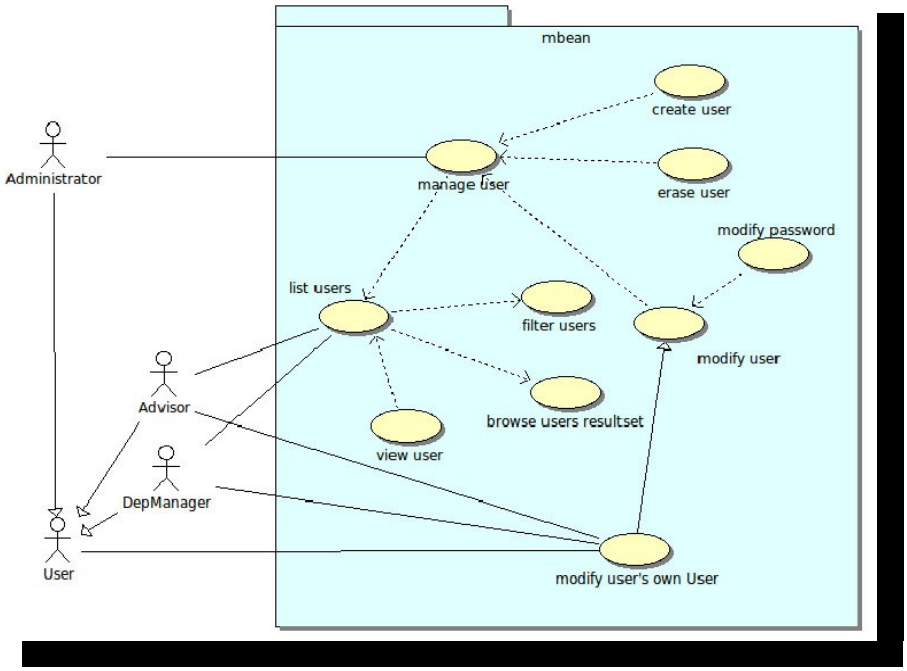
21. ábra Törzsadatok kezelése

Ezek közül választva megjelennek az adott törzsadat típus elemei, listázódnak. Ezután az adott felhasználó kezelheti azokat úgy, hogy módosít, töröl és létrehozhat kategória elemet. Mivel a rendszerben ezekhez az elemekhez rendeltünk id-t, amit tárolunk az űrlapok kitöltésénél, a megnevezését (value) át lehet írni, a kapcsolat akkor is megmarad. Törlésnél figyelni kell arra, hogy van e hivatkozás más attribútumokban az adott elemre (megszorítások kezelése).
Ez a volt a terv, és az implementálását ennek a résznek bemutatom a következő fejezetben.

4.4.3. Felhasználó kezelés

Ebben a részben a felhasználókra vonatkozó általános adatok kezelését terveztük, kihagyva a jogosultság kezelését, amit külön területként találtunk ki, és *4.4.5 Jogosultság kezelés*. pontban ismertetek.

Szerepkörök szerint az adminisztrátor tudja az összes felhasználó adatát kezelni, míg a tanácsadó és a területi vezető csak listázásra és rátekintésre jogosult. Ellenben az összes felhasználó láthatja, és módosíthatja a saját adatait.



22. ábra felhasználó kezelés

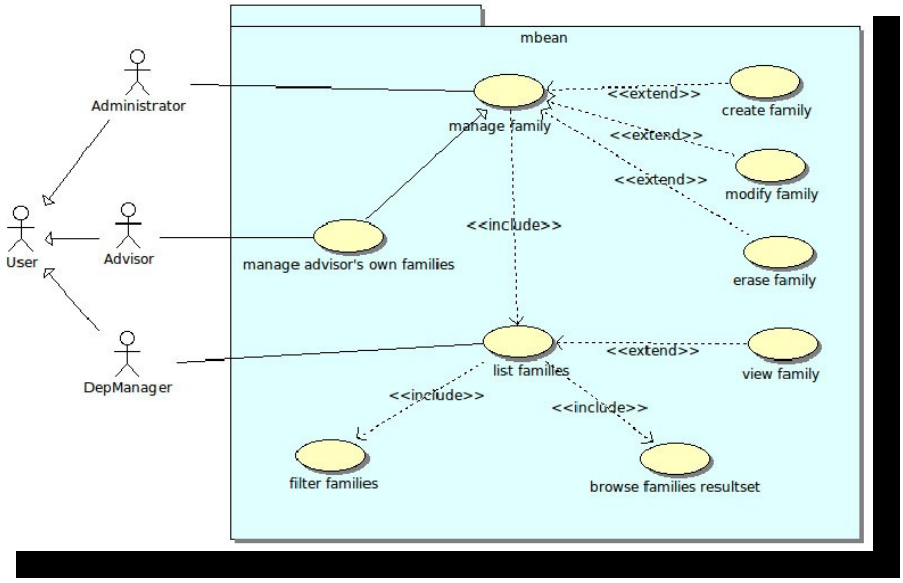
Felhasználói eset szerint az oldalra belépéskor az adminisztrátor, tanácsadó és a területi vezető tudja listázni a felhasználókat, és egyesével a részletesebb adatokat megnézni. Listázásnál a későbbi több adattal számolva beépítettünk lapozást, illetve terveztünk szűrést is a felületre.

Az adminisztrátor ezen a felületen még tud létrehozni és törölni is felhasználókat. Törlésnél természetesen figyelni kell a megszorításokat, ami az osztály diagramból jól látható. Ezen kívül az adminisztrátor az egyetlen személy, aki jogosult bármelyik felhasználó jelszavának módosítására.

Másik folyamaton látható, hogy a felhasználók általában tudják a saját adataikat megtekinteni, és módosítani. Természetesen vannak kapcsolódó adatok amiket, mint pl. a szerződés adatait nem módosíthatják. Ezen a felületen keresztül változtathatják jelszavukat is. Azért emeltük ki ezt a folyamatot külön, mivel ezt speciálisan kell kezelni.

4.4.4. Nevelőcsalád kezelés

Szerepkörök szerint ide [az első verzióban] csak a dedikált felhasználók juthatnak el. Az adminisztrátor az általános adatok kezeléséért felelős, míg a tanácsadó a saját családjaiért. A területi vezető pedig statisztikák készítéséhez tud a rendszerből listázni, lekérdezni adatokat.



23. ábra család kezelés

Folyamatok szerint a dedikált felhasználók egy családok rátekintő listát látnak az induló képernyőn. A tanácsadó ugyan ezen a felületen egy előre szűkített listát fog látni, azokat a családokat, amelyekhez ő van felvéve, mint kapcsolattartó.

A területi vezető csak szűkíteni tudja listát, hogy könnyebb legyen a keresés, és esetleg bele tud menni az egyes listaelemekbe, családok adataiba, de pusztán csak rátekintő jelleggel.

Az adminisztrátor és a tanácsadó tud módosítani, törölni az általa kezelhető családokon. Az adminisztrátor képes létrehozni bármelyik tanácsadóhoz családot, de a tanácsadó csak saját magához vihet fel új családot. Törlésnél természetesen a megszorításokat figyelni kell.

4.4.5. Jogosultság kezelés

Ez a rész számomra a legkedvesebb, mivel én javasoltam, hogy ilyen módon állíthatóak legyenek a jogosultságok a rendszerben. Ezzel együtt ennek a résznek az implementálásával dolgoztam meg a legjobban, mert ha ténylegesen a JSF XML-es összekapcsolási lehetőségét szerettem volna szabványosan használni, gyakorlatilag pont ez a fajta jogosultság rendszer kezelése nehezen kivitelezhető vele. A kitalált jogrendszer kezeli azt, hogy egy oldalon belül meglegyen az összes folyamathoz tartozó menüpont, maximum nem lesz élő egy adott felhasználónál, vagy ha mégis sikerül elindítania, hibaüzenetet jelenít meg a rendszer neki,

hogy nincs joga az adott folyamathoz. A JSF-nél viszont az összes szerepkörhöz az adott oldalakat le kellene külön-külön gyártani, ami esetünkben, ha az adott oldalon 5 féle szerepkör van definiálva, akkor oldal darabszám (nézet) az 5.-en. Ami ha csak a használati esetek számát figyelembe véve, ami 5, az 5 az 5.-en azaz 3125 oldal. Lássuk be, hogy ez kissé nehézkes lenne és sok ideig tartana elkészíteni, viszont az biztos, hogy jól mutatna a faces-xml-ünk. Ráadásul a terveink szerint az adminisztrátor bármikor tervezhet egy új jogcsoportokkal ellátott felhasználót, amihez vagy dinamikusan legyártjuk a JSF-eket, vagy szólnak nekünk, hogy készítsük el. Ez nem járható út, úgyhogy más módszert kellett alkalmazni.

Az általános elv a jogosultság kezeléshez az volt, hogy minden folyamathoz rendelünk jogosultságot, ezeket a jogosultságokat utána a rendszer adminisztrátora tetszőleges csoportokba rendezi. Ezeket a csoportosított jogokat pedig kiosztja a felhasználóknak.

Így, például ha a belépési folyamatot vesszük figyelembe, ha a belépési folyamatot nem rendeljük hozzá egy jogcsoporthoz, és a jogcsoportot egy felhasználóhoz, akkor hiába jó a felhasználónév és a jelszó az adott felhasználó mégsem tud semmit kezdeni a rendszerünkkel.

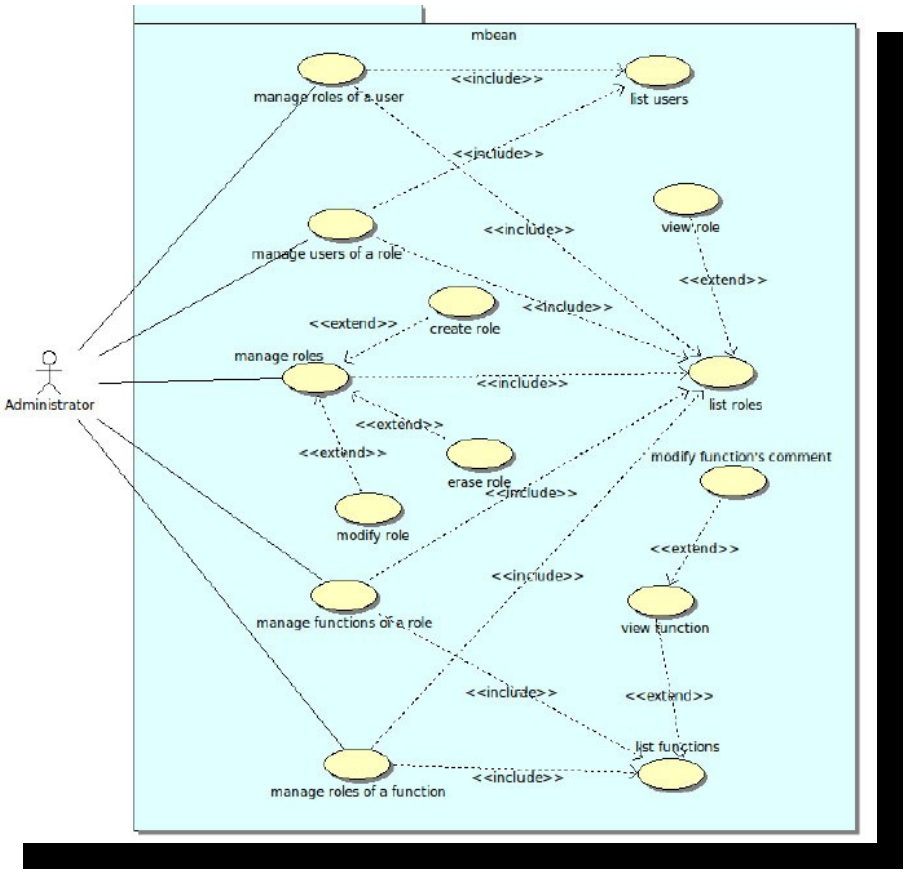
Hosszú távú elképzelés az, hogy a folyamatokhoz adott jogokat a rendszer automatikusan felolvassa a „kódból”, így az adminisztrátornak csak ezek csoportba rendezését és a felhasználókhoz rendelését kell ésszerűen megoldani.

Ezt a menüpontot csak és kizárólag az adminisztrátornak van joga kezelni. Projekttársam kérésére oda-vissza átjárható a rendszer, úgyhogy nem csak az elengedhetetlenül szükséges használati eseteket írtuk meg, ezek a function [joglista elem] jogcsoporthoz [role] adása és a jogcsoport [role] felhasználóhoz adása. Hanem ennek ellenkező irányú bejárását is, ezek a jogcsoport joglista elemhez adása és a felhasználó jogcsoporthoz adása.

Első használati eset ezek közül a jogcsoport kezelés [manage roles], ahol listázzuk a már meglévő jogcsoportokat. Ezen a felületen tud létrehozni, módosítani és törölni jogcsoportot az erre jogosult felhasználó. Illetve, ha kiválaszt egy jogcsoportot, akkor megnézheti, módosíthatja a hozzá rendelt jogok listáját és felhasználók listáját.

Másik felhasználói folyamat, hogy a funkciókat listázva rátekinthet ezekre, terv szerint módosíthatja a commentjét ezeknek. Mivel a value ezeknek ilyeneket fog tartalmazni, hogy *OwnDataPageBean.modifyPasswordAction*, *(*ami ugye a felhasználó saját adatai rátekintő rész jelszó módosítását engedélyezi,) gondoltuk, hogy jó lenne, ha lenne mellette egy leírás,

hogy ez mit takar. Ezen kívül az adott funkciónál megnézheti, hogy mely jogokhoz van rendelve, és ezeket módosíthatja.



24. ábra Jogosultság kezelés

Harmadik folyamat listázza a felhasználókat és egyre rákattintva állíthatjuk a hozzá rendelt jogcsoport listát. Implementálás során jöttünk rá, hogy ez a rész áthelyezhető a felhasználó kezelés részeként

4.5. Látványtervezés, Menüsor, ergonómia

Miután elkészítettük a használati eseteket, átgondolva a menüt, arra döbbentem rá, hogy gyakorlatilag a használati esetek belépési pontjai a menüsor egyes egyedei. A menüsorhoz készítettem egy excel táblát, hogy implementálásnál láthassam, hogy mely pontokat kell még megvalósítani. Ez a táblázat hely hiányában a függelékben található a Menürendszer, Szerepkörök és rendszerfunkciók alapértelmezett kapcsolata részben.

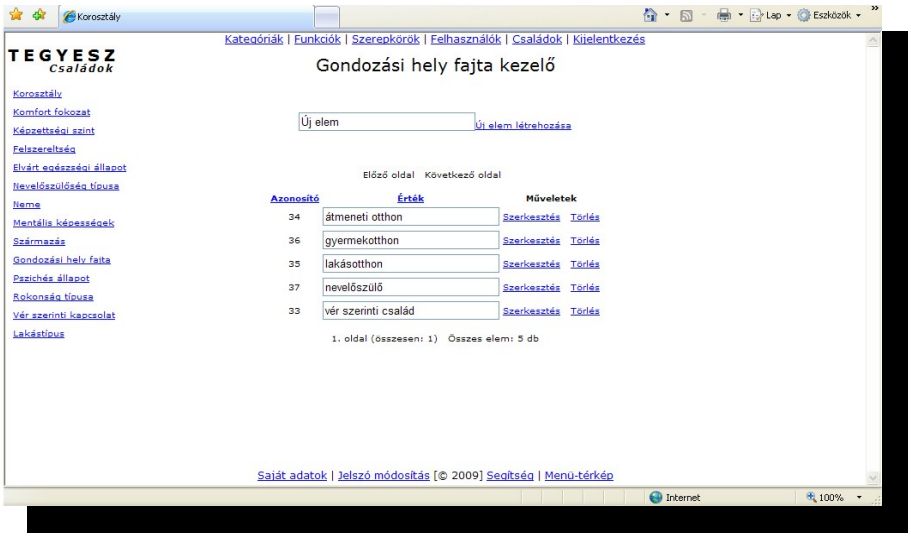
Webes alkalmazásoknál főként kétféle menüsort terveznek a rendszereknek. Az egyik módszer a képernyő bal oldalán, esetlegesen további almenüket layer-en lenyitó menü, míg a másik a képernyő tetején elérhető fő menüpontok, és abból lenyíló almenük. A rendszerünket úgy terveztem, hogy a fő menüsor a képernyő tetején fixen legyenek, míg az összes

felhasználó által elérhető menüpontok, mint pl. a saját adatlap megtekintése, jelszó módosítás, menütérkép a képernyő alján legyenek elérhetőek. A keretet úgy állítottam be, hogy hasznos információk esetén ne kelljen az oldalon görgetni, így a képernyő alja és teteje fix keretbe foglalja a rendszert. Egyes menüpontoknál, mint pl. a Kategóriakezelésnél megjelenik almenüsor is, ezeket a képernyő bal oldalára pozícionáltam, és amíg a felhasználó nem vált másik fő menüpontra, kint is van folyamatosan. Így a rendszer két szempontból is szerencsés, egyik, hogy nem kell bonyolult javascriptes és/vagy ajaxos kódokat használnunk a megjelenítéshez, másik, hogy jól átlátható a kezelői felület. A rendszerünkhöz nem terveztünk látványtervet, grafikai elemeket, mert ez egyrészt a szakdolgozathoz kapcsolódó munkának nem volt célja, másrészt én tapasztalatom az, hogy nem tudok jól rajzolni, másról lemásolni dizájnt meg nem szerettem volna. Nem illet volna a szakdolgozat szellemiségéhez. A kódban igyekeztem a WEB2 technológiának megfelelő html elemeket használni, és egységesen kezelni, és ezekhez osztályokat rendelve kis mértékben dizájnolni. Így később is könnyen lehet majd grafikai elemeket hozzárendelni a rendszerhez.

25. ábra Bejelentkező képernyő magyar nyelven

Mint az ábrán is látható, a bal felső sarokba terveztem egy logót, és a képernyő tetejére egy üdvözlést, ahol bejelentkezés után a főmenü is meg fog jelenni. Ez alatt látható a üdvözlő szöveg, amit egyenlőre még nem a végleges adattal van feltöltve. Két fontos vezérlés látható az oldalon, de telepítés után az inicializálás vezérlést megszüntetjük. A bejelentkezés az autentikációs folyamatot indítja el, postolva a felhasználónév jelszó párost a rendszernek. Hogy a belépés utáni vezérléshez szoktassuk a felhasználót, már nem bejelentkezett módban is vannak az alsó menüsorba tervezett elemek. Ezek pedig a regisztráció és elfelejtett jelszó

(első verziónál nem élő funkciók). Adott oldalra interaktív segítségnyújtó oldal és végül a menütérkép, ami bejelentkezés nélkül elég szegényes tartalommal rendelkezik.



26. ábra Egy kategória elem lista

Bejelentkezés utáni képernyő tagoltsága jól megfigyelhető a 26. ábra Egy kategória elem lista esetén. Főmenü vezérlés, ami fixen megjelenik fent látható. Alul a saját adatok kezelése, jelszó módosítás, ami a regisztráció és a elfelejtett jelszó menüpont helyett jelenik meg szintén fixen a bejelentkezés után, illetve a segítség és a menütérkép, ami minden felhasználótípusnál megjelenik. Baloldalon a helyi almenü, középen pedig a tartalom.

4.6.JSF Framework-ök

Felmerül a kérdés, hogyha már JSF –et használunk az oldalak megjelenítésére, miért nem alkalmaztunk framework-öt az oldalak grafikai tervezéséhez, implementációjához. Bevallom, hogy elég rendesen belekóstoltam ebbe a részbe, átnézve 3-4 framework-öt is. Ezek között volt az Icefaces, Visual Web JSF amik amúgy alapból telepednek a Netbeanssel, vagy a JBoss Richfaces amit az AJAX JSF Mátrixról[12] töltöttem le. Csodáltam a fejlesztőket, milyen jól átgondolt elemeket találtak ki, és fejlesztettek le. Teszteltem is, melyiket tudnám a legjobban felhasználni. Végül azonban a generált forrás alapján úgy döntöttem az előző fejezetben ismertetett okok miatt, és mivel ez az első ilyen alkalmazásom, inkább megmaradok a standard és jól olvasható HTML kódnál, ezért maradtam a ’fapados’ verziónál.

[12] Bővebben: http://www.jsfmatrix.net

4.7.Jogosultságkezelés implementálása

A Jogosultság kezelés részben vázolt probléma megoldásán sokat rágódtam. A végén úgy döntöttünk, hogy ezt nem lehet szabványos automatizmusokkal lefejleszteni, hanem két helyen is kezelni kell. Egyrészt a menedzselt bean-nél kell ellenőrizni azt, hogy az adott metódust ki hívta meg. Másrészt a JSF oldalakon a vezérlések kirakásánál a rendered opcióba be kell írni ezt az ellenőrzést. Erre mutatok példát a következő fejezetben.

Azt, hogy a folyamat vezérelt jogellenőrző rendszernek mely metódusokra kell ellenőriznie a függelékben található menürendszer alapján előre definiáltuk. A *tegyesz.war.util* csomagban található security bean kezeli a jogosultságokat a getCheckFunction() metódussal. Ez a metódus egyetlen paramétert vár, mégpedig a metódus nevét. Először az volt a gondolatom, hogy a hívott metódusnál meg tudom találni a hívó osztály mely metódusából hívták, de átgondolva a JSF generálási folyamatot, ez nem volt járható út. Így ezzel a gondunk az volt, hogyan lehetne nem kézzel beírni a metódus nevet, mivel először én kézzel adtam meg, hirtelen más ötletem nem lévén.

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

27. ábra az isAllowed() metódus

Projekttársam addig őrlődött a problémán, amíg a végén rájött a megoldásra. Nem direktben hívjuk az ellenőrző metódust, hanem minden egyes osztályba, ahol erre szükség van, elkészítünk egy isAllowed() metódust, és ezen keresztül történik meg a hívás. Így biztos, hogy tudni fogjuk a metódus pontos nevét. Így már könnyebben ment az ellenőrzés implementálása, és sokkal inkább automatikus volt.

5. Konkrét használati eset bemutatása

Sokat őrlődtem, hogy melyik használati esetet válasszam bemutatásra, végül a törzsadatok kezelését választottam, mivel ez egy képernyőn meg lett valósítva. Első lépésben bemutatom a felhasználói oldal generálásához és kezeléséhez használt fájlrendszer felépítését. Majd folytatom azzal, hogy bemutatom az osztályban használt változókat. Ezután a 4.4.2 Kategória kezelés fejezetben talált használati eset megvalósítását részletezem a JSF és a menedzselt bean metódusain keresztül. Majd pár megoldást részletezek, például a JSF kódban milyen JSP elemeket voltam kénytelen használni, mert nem találtam rá más megoldást, illetve hogy hogyan tudtam megoldani a teljes rendszer nemzetköziesítését.

5.1.Strukturális felépítése

Ennek a felületnek a implementálásához összesen 4 Java osztályt és egy JSF kódsort kellett felhasználjam. Az osztályok legfontosabb természetesen a *tegyesz.war.mbean* csomagban megtalálható *categoryPageBean*, ami JSF vezérelt bean és bemutatom a következő részben a működését. A nemzetköziesítéshez szükség volt az általános *tegyesz.war.util.MsgL9n* és az oldal specifikus *tegyesz.war.mbean.TranslatedCategoryName* osztályra. Ezen kívül a korábban említett *tegyesz.war.util.security* jogosultság kezelő osztályra.

5.2.Kategória kezelés használati eset folyamatai

```
public class CategoryPageBean {

    @EJB
    private CategoryManagerRemote categoryManager;
    private static final String categNames[] = {
        "AgeClass", "ComfortType", "EduLevel", "Equipped", "ExpectedHealthState",
        "FostererParentType", "Gender", "MentalAbilities", "Origin",
        "PlaceOfCareType", "PsychicState", "RelationType", "RelativeConnected",
        "ResidenceType"
    };
    private String categName = "AgeClass";
    private Category categoryItem;
    private UIData categoryItemTable;
    static private final int itemsPerPage = 8;
    private int firstItem = 0;
    private String pagingDirection;
    private String sortForAttribute = "value";
    private String sortingDirection = "asc";
    private int itemCount = -1;
```

28. ábra Változók deklarálása

Első lépésként categoryPageBean osztályban használt változókat gyorsan átnézzük, a könnyebb érthetőség kedvéért. Mint az ábrán is látható, próbáltam beszédes változó és metódus neveket adni a kódomban. A categNames tömbben tárolom a törzsadatok entitás neveit. A categName, az első törzsadat, amit én önkényesen kiválasztottam, hogy ezen menüpont választása esetén az elemei listázódnak. Az itemsPerPage meghatározza, hogy egy oldalon hány elem jelenik meg, ezt későbbi fejlesztésnél lehetne egy rendszer környezeti objektum elemként használni, amit a felhasználó konfigurálhat magának. Igaz, ekkor lehet, hogy az ergonómia romlik…

A firstItem és a pagingDirection a lapozás implementálásához kellet, a firstItem azaz elem, amelyiktől a listázás kezdődik. A sortForAttribute és a sortingDirection az elemek rendezésének iomplementálásához kellet. Az attribute enum is lehetne, hiszen csak ID és

value értékeket vehet fel, mint ahogy a sortingDirection értékei az asc. és a desc. Az itemCount az elemek számát adja meg.

A categoryManager osztály-t néhány helyen én is módosítottam, mivel a rendezésre és lapozásra a projekttársam rendszere nem volt felkészítve, és úgy ítéltem meg, hogy ezekre szükség van.

Kiemeltem két metódust a categoryPageBean osztályból a getCategoryListAction, ezek implementálását mutatom most be.

5.2.1. Törzsadatok listázása (getCategoryListAction)

A kategóriák listázása metódus, számomra komoly kihívás volt, mivel nekem csak angol rövidített néven van meg categNames tömbben az adott elem, és ebből kellet egy nemzetköziesített listaelem megjelenítést készítenem. Hosszas töprengés után létrehoztam egy technikai osztályt, amit elneveztem TranslatedCategoryName osztálynak, ennek az osztálynak csak annyi volt a lényege, hogy volt két változója, az egyik az osztálynevet tárolja, ami a rendszerben entitásnév, a másik értéke pedig a nemzetköziesített nevet, ez a segéd osztály a függelékben megtalálható. Ezután már nem volt más dolgom, mint példányosítani annyi elemet, ahány categNames volt, és megadnom értéknek a native és a MsgL9n osztállyal lefordított nevet az objektumnak, majd hozzáadnom ahhoz listához az objektumot, amit megjelenítek a JSF-el.

```
public List getCategoryListAction() {
    List list = new ArrayList();

    if (isAllowed()) {
        for (String nativeItemName : categNames) {
            TranslatedCategoryName tcn = new TranslatedCategoryName();
            tcn.setNativeCategoryName(nativeItemName);
            tcn.setValue(MsgL9n.getMessage(nativeItemName));
            list.add(tcn);
        }
    }
    return list;
}
```

29. ábra getCategoryListAction

Ezen kívül persze, mivel itt jogosultság kezelés szükséges, használom a korábban már említett isAllowed() metódushívást, ami engedélyezi a lista összeállítását.

```
<h:form id="List">
    <div id="ListForm" class="ListForm">
        <h:dataTable value="#{CategoryPageBean.categoryListAction}" var="item">
            <h:column>
                <h:commandLink action="#{CategoryPageBean.changeCategory}"
                               value="#{item.value}">
                    <f:setPropertyActionListener value="#{item.nativeCategoryName}"
                                                 target="#{CategoryPageBean.categName}"/>
                </h:commandLink>
            </h:column>
        </h:dataTable>
    </div>
</h:form>
```

30. ábra Kategória elemek listázása JSF kódrészlet

A kódban látható [30. ábra Kategória elemek listázása JSF kódrészlet], hogy egy adattáblát rendelek ehhez a metódushoz. A táblának egyetlen oszlopa van, aminek elemeiben egy vezérlőelem látható [*<h:commandLink*].

```
public String changeCategory() {
    initCategory();
    setItemCount(categoryManager.getCount(categoryItem.getClass()));
    return null;
}
```

31. ábra Törzstábla cserélése - changeCategory()

A vezérlőelem a changeCategory() metódust hívja , miután beállítja az osztály categName változó értékét az adott entitás nevére. A menedzseltBean-ben inicializálunk, és beállítjuk az választott törzsadat elemszámát az ItemCount változóba.

```
private void initCategory() {
    categoryItem = getNewCategObject();
    categoryItem.setValue(MsgL9n.getMessage("NewItem"));
}
```

32. ábra Törzsadatok osztály inicializálása - initCategory()

5.2.2. Törzsadat elemek listázása [getCategoryListAction]

Ennek a metódusnak az elkészítése igazából nem volt nehéz feladat, viszont a JSF-ben való implementálása a lapozás és a rendezés miatt koránt sem volt egyszerű.

```
public List getCategoryItemListAction() {
    List list = null;
    if (isAllowed()) {
        int fromIndex = firstItem;
        int toIndex = itemsPerPage;
        if (toIndex >= getItemCount()) {
            toIndex = getItemCount();
        }
        list = categoryManager.findEntities(toIndex, fromIndex,
                sortForAttribute + " " + sortingDirection, categoryItem.getClass());
    }
    return list;
}
```

33. ábra Törzsadat elemek listázása - getCategoryListAction()

Mint az ábrán is látható, hasonlóan az előző részhez, itt is egy listát kellett összeállítani a JSF-nek. Ebben a metódusban a Category menedzserben általam implementált metódusát kell felparaméterezve meghívni. A paraméterek sorban: megjelenítendő elemek száma maximum,

hányadik elemtől kell megjeleníteni az elemeket, az adatok melyik attribútum szerint és hogyan vannak rendezve, melyik törzsadatról van szó. A categoryManager a nekünk kellő listát fogja visszaadni. Ezt a részt azért nem részletezem, mert bár én kódoltam, ennek dokumentálása a projekttársam feladatai közé tartozik.

A lapozás JSF kódja a 34. ábra Lapozás Implementálása oldalon látható. Itt már használom a rendered opciót arra, hogyha adott irányba nem kell lapozni, akkor csak *<h:outputText* elemként jeleníttetem meg az adott elemet. Erre azért volt szükség, hogy a lapozást kezelő elemek ergonómiailag kényelmesen mindig ugyanott legyenek megtalálhatóak.

```
<h:form id="ListItems">
    <div id="ListItemsMenu" class="ListItemsMenu">
        <h:outputText value="#{msg.Previouspage}"
                      rendered="#{CategoryPageBean.firstItem==0  && CategoryPageBean.itemCount>0}"/>
        <h:commandLink value="#{msg.Previouspage}" action="#{CategoryPageBean.page}"
                      rendered="#{CategoryPageBean.firstItem>0  && CategoryPageBean.itemCount>0}">
            <f:setPropertyActionListener value="prev" target="#{CategoryPageBean.pagingDirection}"/>
        </h:commandLink>
         
        <h:outputText value="#{msg.Nextpage}"
                      rendered="#{CategoryPageBean.firstItem +CategoryPageBean.itemsPerPage
                                  >=CategoryPageBean.itemCount
                                  && CategoryPageBean.itemCount>0}"/>
        <h:commandLink value="#{msg.Nextpage}" action="#{CategoryPageBean.page}"
                      rendered="#{CategoryPageBean.firstItem+CategoryPageBean.itemsPerPage
                                  <CategoryPageBean.itemCount
                                  && CategoryPageBean.itemCount>0}">
            <f:setPropertyActionListener value="next" target="#{CategoryPageBean.pagingDirection}"/>
        </h:commandLink>
         
    </div>
```

34. ábra Lapozás Implementálása

Mint az ábrán is látható, lapozáskor a page() metódust hívja meg a rendszer miután a pagingDirection értékét átállítja. A page metódus semmi mást nem tesz, csak aszerint, hogy a pagedirection mire van állítva, hozzáadja vagy elveszi az egy lapon megjelenő elemek számát a firstitemhez. Így a következő nézetben már a frissített lista lesz.

```
public String page() {
    if (getPagingDirection().equals("prev") && firstItem >= itemsPerPage) {
        setFirstItem(firstItem - itemsPerPage);
    } else if (getPagingDirection().equals("next") && firstItem + itemsPerPage <= getItemCount()) {
        setFirstItem(firstItem + itemsPerPage);
    }
    return null;
}
```

35. ábra lapozás vezérlés - page()

A lapozás után következik a kódban a törzsadat elemek listázása [36. ábra a Törzsadat elem lista implementálása], ami mint az előző példa mutatta egy adattáblán keresztül jelenítem meg a törzselem adatokat. Viszont itt nem egy oszlopból álló, hanem rögtön három oszlopos táblázatot generálok. Amiből kettő az törzselem attribútumai (id, value) és egy plusz az akciók megjelenítésére van (módosítás, törlés).

Ennél a táblánál már használom a JSF *<h:datatable*-ra vonatkozó *<h:facet* szintaxisát. Ez azt a célt szolgálja, hogy HTML oldal generálódásakor a generáló készítsen a táblán a fej- vagy lábléc sort, attól függően, hogy name attribútumnál header vagy footer van megadva értéknek.

Én a fejléc sort választottam, és ebbe a sorba helyeztem el a rendezést. Ezt úgy készítettem el, hogy *<h:commandlink* vezérlő elemként hivatkozok az adott attribútum nevére, meghívva az applysort() metódust, úgy, hogy előtte még beállítom a sortForAttribute változó értékét értelemszerűen id vagy value értékre.

Az applysort() metódus [37. ábra Rendezés vezérlése - applySort()] mindössze annyi, hogy a sortingDirection értékét váltja, plusz nullára állítja a firstItem értéket. Így a következő nézetben már az új rendezés szerint jelenik meg a tartalom az első elemtől kezdve.

A facet után látható *<h:outputtext* és a *<h:inputtext* a tényleges adatok értékét írja ki.

```
<h:dataTable value="#{CategoryPageBean.categoryItemListAction}" var="item"
             binding="#{CategoryPageBean.categoryItemTable}"
             rendered="#{CategoryPageBean.itemCount>0}">
    <h:column>
        <f:facet name="header">
            <h:commandLink value="#{msg.Id}" action="#{CategoryPageBean.applySort}">
                <f:setPropertyActionListener value="id" target="#{CategoryPageBean.sortForAttribute}" />
            </h:commandLink>
        </f:facet>
        <h:outputText id="ItemId" value="#{item.id}"/>
    </h:column>
    <h:column>
        <f:facet name="header">
            <h:commandLink value="#{msg.Value}" action="#{CategoryPageBean.applySort}">
                <f:setPropertyActionListener value="value" target="#{CategoryPageBean.sortForAttribute}" />
            </h:commandLink>
        </f:facet>
        <h:inputText value="#{item.value}">
            <f:validateLength maximum="16"/>
        </h:inputText>
    </h:column>
    <h:column>
        <f:facet name="header">
            <h:outputText value="#{msg.Action}"/>
        </f:facet>
        <h:commandLink action="#{CategoryPageBean.modifyCategoryItemAction}" value="#{msg.Edit}"/>
         
        <h:commandLink action="#{CategoryPageBean.deleteCategoryItemAction}" value="#{msg.Delete}"/>
    </h:column>
</h:dataTable>
```

36. ábra a Törzsadat elem lista implementálása

A harmadik oszlop elemei két vezérlést valósítanak meg, az egyik a módosítás, amit a *5.2.4 Törzsadat elemek módosítása (modifyCategoryItemAction)* fejezetben tárgyalok a másik pedig az adott elem törlése, amit a *5.2.5 Törzsadat elemek törlése (deleteCategoryItemAction)* fejezetben tárgyalok.

```
    public String applySort() {
        if (sortingDirection.equalsIgnoreCase("asc")) {
            setSortingDirection("desc");
        } else {
            setSortingDirection("asc");
        }
        firstItem = 0;
        return null;
    }
```

37. ábra Rendezés vezérlése - applySort()

5.2.3. Törzsadat elemek létrehozása (createNewItemAction)

```
<h:form id="NewItem">
    <div id="NewItem" class="NewItem">
        <h:inputText id="NewItem" value="#{CategoryPageBean.categoryItem.value}">
            <f:validateLength maximum="64"/>
        </h:inputText>
        <h:commandLink action="#{CategoryPageBean.createNewCategoryItemAction}" value="#{msg.CreateNew}" />
    </div>
</h:form>
```

38. ábra Törzsadat elem létrehozása

Ez egy viszonylag rövid kódrészlet. Feladata, hogy új kategória elemeket hozzon létre. Erre készítettem egy NewItem formot, amiben egy *<h:inputtext* box-ot adok meg a JSF-ben, ami várja az új elem értékét. Itt használom a JSF validálási lehetőségét az *<f:validatelength* opciót, ami azt ellenőrzi, hogy az adott inputbox-ba beírt szöveg nem lehet hosszabb, mint a maximum attribútumban megadott érték. Amennyiben mégis az, egy standard hibaüzenetet jelenít meg a rendszer, és nem hajtja végre a létrehozást. A *5.4 Nemzetköziesítés* fejezetben leírom, hogy ezt az üzenetet is sikerült lefordítatnom a rendszerrel magyar nyelvre. Ha a felhasználó az inputbox-ot helyesen kitöltve rákattint a vezérlésre, akkor a createNewItemAction() metódus fog lefutni.

```
public String createNewCategoryItemAction() {
    if (isAllowed()) {
        try {
            categoryManager.create(categoryItem);
            setItemCount(categoryManager.getCount(categoryItem.getClass()));
        } catch (Exception e) {
            MsgL9n.putFacesErrorMessage("UserErrorMessage1", e.getMessage());
        }
    }
    return null;
}
```

39. ábra Törzselem létrehozása - createNewCategoryItemAction()

Ez a metódus, először is vizsgálja a felhasználó jogosultságát, majd megpróbálja beilleszteni az EJB konténert felhasználva az adott elemet a törzsadat elemek közé. Ha sikerül, listázásnál már a helyén jelenik meg. Ha nem, akkor egy általunk megfogalmazott üzenet fog megjelenni pirossal az oldal alján. A rendszer az id értéket automatikusan rendeli egy elemhez, de erről projekttársam feladata, hogy írjon.

5.2.4. Törzsadat elemek módosítása (modifyCategoryItemAction)

Listázáskor, mint láthattuk [36. ábra a Törzsadat elem lista implementálása] az elemeket soronként jelenítem mega felhasználónak, és a value értéket *<h:inputext* boxba helyezem el. Mindezt azért, hogy a módosítást implementálhassam az oldalba. A tábla akciók oszlopában az a *<h:commandlink* vezérli, amelyben a *CategoryPageBean modifyCategoryItemAction*() metódusát hívjuk meg.

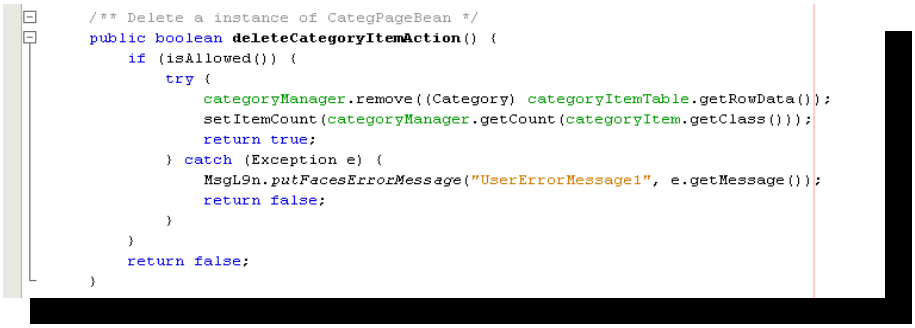
```
/** Modifies a instance of CategPageBean */
public boolean modifyCategoryItemAction() {
    if (isAllowed()) {
        try {
            categoryManager.edit((Category) categoryItemTable.getRowData());
            return true;
        } catch (Exception e) {
            MsgL9n.putFacesErrorMessage("UserErrorMessage1", e.getMessage());
            return false;
        }
    }
    return false;
}
```

40. ábra Törzsadat elem módosítása - modifyCategoryItemAction

Itt is a jogosultság kezelés az első lépés, majd megpróbáljuk végrehajtani a módosítást az adott elemen. Ha valamiért, pl. unique megszorítás miatt nem sikerül, itt is egy általunk generált hibaüzenetet kap a felhasználó.

5.2.5. Törzsadat elemek törlése (deleteCategoryItemAction)

Listázáskor, mint láthattuk [36. ábra a Törzsadat elem lista implementálása] az elemeket soronként jelenítem mega felhasználónak. Ha a felhasználó törölni akar egy elemet, akkor a tábla akciók oszlopában az a *<h:commandlink* vezérli, amelyben a *CategoryPageBean deleteCategoryItemAction*() metódusát hívjuk meg.



```
/** Delete a instance of CategPageBean */
public boolean deleteCategoryItemAction() {
    if (isAllowed()) {
        try {
            categoryManager.remove((Category) categoryItemTable.getRowData());
            setItemCount(categoryManager.getCount(categoryItem.getClass()));
            return true;
        } catch (Exception e) {
            MsgL9n.putFacesErrorMessage("UserErrorMessage1", e.getMessage());
            return false;
        }
    }
    return false;
}
```

41. ábra Törzsadat elem törlése - deleteCategoryItemAction

A metódus a jogosultság vizsgálat után megpróbálja az elemet törölni. Ha sikertelen megszorítási okok miatt, akkor a rendszer erről tájékoztatja egy általunk megfogalmazott hibaüzenetben a felhasználót a lap alján.

5.3. Beágyazott JSP kódok

A kódolás során többször is úgy éreztem, hogy a JSF határait elértem, és máshogy egy adott dolgot nem tudok megvalósítani, csak úgy, ha beágyazott JSP kódot használok. Többször is használtam a szkript elemet, azok közül is a kifejezést ebben a kódban. Egyszer a title nevének lokalizált megadásához, másodszor pedig a dizájnkönyvtár HTML kódban abszolút, de a JSF kódban mégis relatív megadásához.

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title><%= new tegyesz.war.mbean.CategoryPageBean().getTranslatedClassName()%></title>
        <link REL="STYLESHEET" TYPE="text/css"
        HREF="<%=request.getContextPath()%>/design/CategoryManagement.css">
        <jsp:include page="/head.jsp" />
    </head>
    <body>
        <f:view>
```

42. ábra beágyazott JSP kódok

Ezen kívül a page direktívát egy fájl include-olására. A fájlban egyébként csak az általános dizájnra vonatkozó css hívása található, és ez az include egyébként az összes megjelenítő oldalon szerepel.

5.4. Nemzetköziesítés

Már csak egyetlen téma maradt, ami számomra fontos, hiszen sokat gondolkodtam ennek megoldásán. Ez pedig az L9n, azaz a localization megoldása a rendszerünkben. Bár erre nem biztos, hogy szükség lesz is valaha, viszont kihívásnak tekintettem ezt megvalósítani.

Három lépcsője volt ennek a folyamatnak. Az első és egyben legegyszerűbb, a JSF oldalon megadott szövegek nemzetköziesítése.

```
    <application>
        <locale-config>
            <default-locale>en</default-locale>
            <supported-locale>hu</supported-locale>
        </locale-config>
        <message-bundle>tegyesz.war.properties.Messages</message-bundle>
    </application>
```

43. ábra faces-config.xml részlet

Ehhez először a faces-config.xml-ben meg kellett adnom a szükséges beállításokat, majd a JSF kódban az *<f:loadBundle* taget felhasználva basename attribútumban megadva a teljesen kvalifikált elérési úttat.

```
<f:view>
    <f:loadBundle basename="tegyesz.war.properties.Messages" var="msg"></f:loadBundle>
    <h:form id="Top">
        <div class="Top">
            <a href=""><h:outputText value="#{msg.Categories}"/></a>
```

44. ábra a loadbundle és egy hivatkozás a JSF-ben

Majd megjelenítéskor a loadbundle var attribútumának értékével hivatkozva fordíttattam a rendszerrel le a különböző kifejezéseket. Természetesen a hivatkozott properties fájlokban meg kellett tenni a fordítást.

```
Users=Felhasználók
Value=Érték
javax.faces.validator.LengthValidator.MAXIMUM= Validációs hiba történt.
                                                A várt szöveg karakterszáma több, mint {0}.
```

45. ábra properties_hu fájl részlet

A következő lépés a 45. ábra properties_hu fájl részlet-ben is látható paraméterezett és beépített hivatkozások generálása volt. Végül az utolsó lépés a rendszer által generált hibaüzenetek nemzetköziesítése volt. Erre készítettem el a *tegyesz.war.util* csomagban található az MsgL9n osztályt, aminek feladata volt, egyrészt, hogy a kódból kezelje a nemzetköziesítést (ezt a *getMessage(String msgNativeTitle)* metódus valósította meg), másrészt a JSF message üzenetei közé fel tudjak venni elemeket (ezt a *putFacesErrorMessage*() metódus valósította meg).

Összefoglalás

A TEGYESZ családok projekt tervezése, fejlesztése közben megismert szoftverek és technológiák rengeteg új ismerettel bővítették eszközkészletemet. Bízom benne, hogy a későbbiek során hasznomra lesz a most megismert Java Enterprise környezet és a hozzá kapcsolódó (JSF, JSP, JPA, Hibernate) technológiák. Kialakult véleményem a projekt során, hogy érdemes megismerkedni az új technológiákkal, mert sok esetben rengeteget segíthetnek a gyorsabb, jobb és magasabb minőséggel ellátott szoftverek könnyebb kialakításában.

A technológiák megismerése mellett bővült a tapasztalatom Java fejlesztés területén, és csak remélhetem, hogy a sok problémán, amin sikerült átküzdenem magam, hosszú távon a segítségemre lesz.

Ezeken kívül célja volt még a szakdolgozatnak, hogy megismertessen minket a projektmunka előnyeivel, hátrányaival. Szerintem sikerült megtapasztalnunk mindkettőnknek, hogy ketten jó projekttervezés mellett hatékonyabban tudunk egy adott feladatot kidolgozni, megoldani.

A projekt, a feladat, és a problémák megoldása során sok mindennek kellett utána nézni, több szakmai fórumba beleolvasni. Nagyon sokat segített két könyv a megoldások keresésében, ezek Vég Csaba könyve Instant Java / Java EE / SOA I. és II. és Imre Gábor, Balogh Péter, Berényi Zsolt, Dévai István, Soós István, Tóthfalussy Balázs közös könyve a Szoftverfejlesztés Java EE platformon. Ezt a két könyvet ebben a sorrendben csak ajánlani tudom mindenkinek, aki meg akar ismerkedni ezekkel a technológiákkal.

Jövőbeli terveinkről dolgozatom során már írtam, említettem, hogy reméljük az elkészített szoftverrel sikerül elérnünk azt, hogy a Területi Gyermekvédelmi Szakszolgálatok a későbbiekben megrendelőinké váljanak. Ez egyben azt is jelenti, hogy bízunk a most megszerzett tudásunkban, és a megismert technológiákat a jövőben is hasznosítani szeretnénk. Végül pedig remélem, sikerült egy maradandó szoftvert alkotni, ami hasznára válik a *hbmtgysz* szakemberei számára is.

Irodalomjegyzék

[CSS-en] CSS Tutorial (2008.05.20)
http://www.w3schools.com/css/

[CSS1] CSS Példák (2009.04.20)
http://www.csszengarden.com/tr/magyar/

[CSS2] Bevezetés a CSS alapjaiba (2009.04.20)
http://prog.hu/cikkek/907/Bevezetes+a+CSS+alapjaiba.html

[HTML] A Html története (2008.05.20)
http://www.standard-team.hu/frame.html

[Java] Vég Csaba: *Instant Java / Java EE / SOA I. és II.*
Logos2000 kiadó 2007

[Java-en] Java in action (2009.02.17)
http://www.java.com/en/java_in_action/

[JSP, JSF] Imre Gábor (szerk.), Balogh Péter, Berényi Zsolt, Dévai István, Soós István, Tóthfalussy Balázs: *Szoftverfejlesztés Java EE platformon*
Szak kiadó 2007

[JSF-en] A first look at JavaServer Faces (2009.03.17)
http://www.javaworld.com/javaworld/jw-11-2002/jw-1129-jsf.html

[JSF fw] JSF AJAX Component Library Feature Matrix 2009.02.23
http://www.jsfmatrix.net

[model1] A szoftverfejlesztési módszertan (2009.04.16)
http://zeus.nyf.hu/~akos/progmodsz/szoftverfejlesztes.ppt

[model2] Szoftver életciklus modellek (2009.04.16)
http://jpnet.hu/files/sze.hu/~szoftvertechnologia/pdf/lifecyc.pdf

[MySQL] MySQL hivatalos oldala (2009.04.29)
http://www.mysql.com/

[Netbeans] NetBeans Developer 2.1 - Útmutató a használathoz (2009.02.20)
http://www.inf.bme.hu/ooret/1999osz/DevelopmentTools/Netbeans/netbeans.html

[PostGre] PostGreSQL hivatalos oldala (2009.04.29)
http://www.postgresql.org/

[Sun] Sun hivatalos oldala (2009.04.28)
http://www.sun.com/

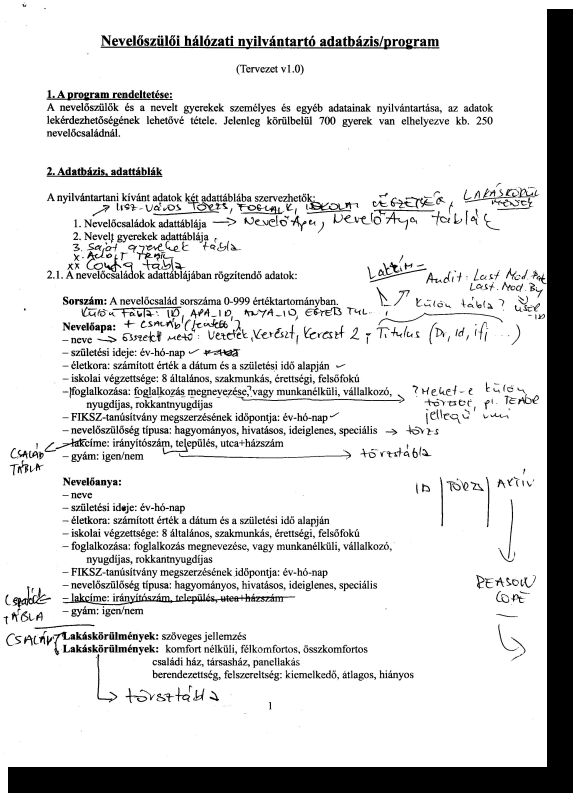
[Szervlet] Finta Annamária - Java Szervletek (2009.03.12)
http://www.codexonline.hu/CodeX1/alap/webpr/webpro/jase/jse.htm

[UML] Unified Modelling Language (2009.02.15)
http://www.geocities.com/SiliconValley/Network/1582/uml.htm

Függelék

Követelmények meghatározása TEGYESZ

1. oldal



Nevelőszülői hálózati nyilvántartó adatbázis/program

(Tervezet v1.0)

1. A program rendeltetése:
A nevelőszülők és a nevelt gyerekek személyes és egyéb adatainak nyilvántartása, az adatok lekérdezhetőségének lehetővé tétele. Jelenleg körülbelül 700 gyerek van elhelyezve kb. 250 nevelőcsaládnál.

2. Adatbázis, adattáblák

A nyilvántartani kívánt adatok két adattáblába szervezhetők:

1. Nevelőcsaládok adattáblája
2. Nevelt gyerekek adattáblája

2.1. A nevelőcsaládok adattáblájában rögzítendő adatok:

Sorszám: A nevelőcsalád sorszáma 0-999 értéktartományban.

Nevelőapa:
– neve
– születési ideje: év-hó-nap
– életkora: számított érték a dátum és a születési idő alapján
– iskolai végzettsége: 8 általános, szakmunkás, érettségi, felsőfokú
– foglalkozása: foglalkozás megnevezése, vagy munkanélküli, vállalkozó, nyugdíjas, rokkantnyugdíjas
– FIKSZ-tanúsítvány megszerzésének időpontja: év-hó-nap
– nevelőszülőség típusa: hagyományos, hivatásos, ideiglenes, speciális
– lakcíme: irányítószám, település, utca+házszám
– gyám: igen/nem

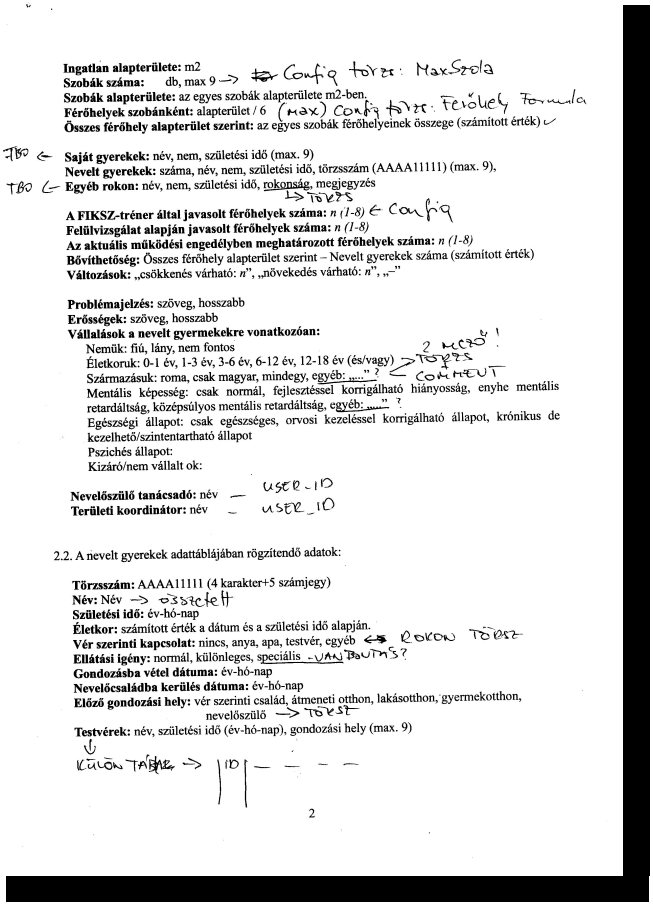
Nevelőanya:
– neve
– születési ideje: év-hó-nap
– életkora: számított érték a dátum és a születési idő alapján
– iskolai végzettsége: 8 általános, szakmunkás, érettségi, felsőfokú
– foglalkozása: foglalkozás megnevezése, vagy munkanélküli, vállalkozó, nyugdíjas, rokkantnyugdíjas
– FIKSZ-tanúsítvány megszerzésének időpontja: év-hó-nap
– nevelőszülőség típusa: hagyományos, hivatásos, ideiglenes, speciális
– lakcíme: irányítószám, település, utca+házszám
– gyám: igen/nem

Lakáskörülmények: szöveges jellemzés
Lakáskörülmények: komfort nélküli, félkomfortos, összkomfortos
családi ház, társasház, panellakás
berendezettség, felszereltség: kiemelkedő, átlagos, hiányos

1

46. ábra Követelmények meghatározása 1. oldal

2. oldal



Ingatlan alapterülete: m2
Szobák száma: db, max 9 → ~~fix~~ Config törzs: MaxSzoba
Szobák alapterülete: az egyes szobák alapterülete m2-ben
Férőhelyek szobánként: alapterület / 6 (max) Config törzs: 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

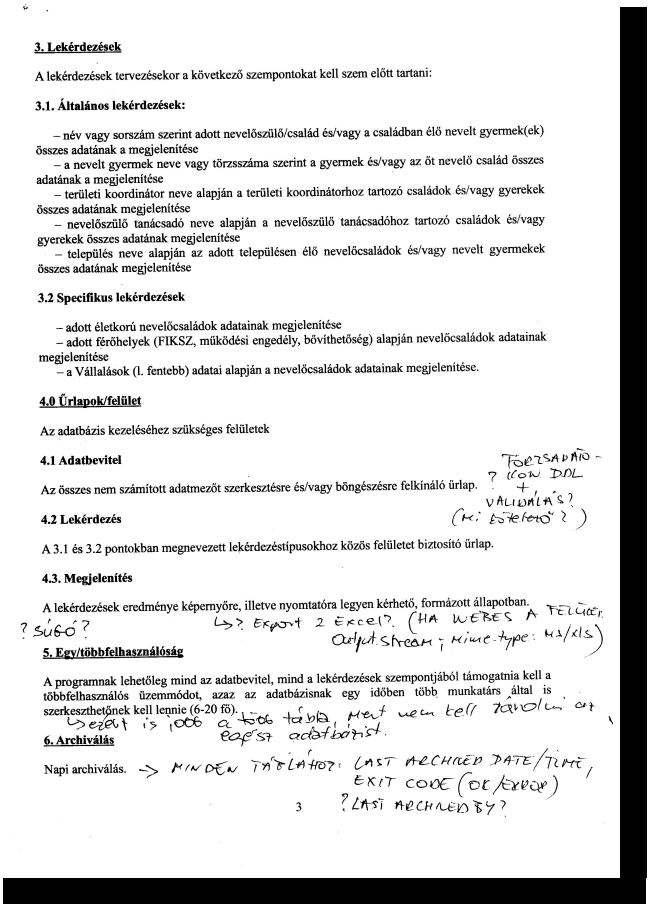
2.2. A nevelt gyerekek adattáblájában rögzítendő adatok:

Törzsszám: AAAA11111 (4 karakter+5 számjegy)
Név: Név → összefelett
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/BAUTM3?
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 TÁBLA → ID | — — — —

2

47. ábra Követelmények meghatározása 2. oldal

3. oldal



3. Lekérdezések

A lekérdezések tervezésekor a következő szempontokat kell szem előtt tartani:

3.1. Általános lekérdezések:

– név vagy sorszám szerint adott nevelőszülő/család és/vagy a családban élő nevelt gyermek(ek) összes adatának a megjelenítése
– a nevelt gyermek neve vagy törzsszáma szerint a gyermek és/vagy az őt nevelő család összes adatának a megjelenítése
– területi koordinátor neve alapján a területi koordinátorhoz tartozó családok és/vagy gyerekek összes adatának megjelenítése
– nevelőszülő tanácsadó neve alapján a nevelőszülő tanácsadóhoz tartozó családok és/vagy gyerekek összes adatának megjelenítése
– település neve alapján az adott településen élő nevelőcsaládok és/vagy nevelt gyermekek összes adatának megjelenítése

3.2 Specifikus lekérdezések

– adott életkorú nevelőcsaládok adatainak megjelenítése
– adott férőhelyek (FIKSZ, működési engedély, bővíthetőség) alapján nevelőcsaládok adatainak megjelenítése
– a Vállalások (l. fentebb) adatai alapján a nevelőcsaládok adatainak megjelenítése.

4.0 Űrlapok/felület

Az adatbázis kezeléséhez szükséges felületek

4.1 Adatbevitel

TÖRZSADATOK - ? ICON DDL + VALIDÁLÁS? (MI ESTEKOR??)

Az összes nem számított adatmezőt szerkesztésre és/vagy böngészésre felkínáló űrlap.

4.2 Lekérdezés

A 3.1 és 3.2 pontokban megnevezett lekérdezéstípusokhoz közös felületet biztosító űrlap.

4.3. Megjelenítés

A lekérdezések eredménye képernyőre, illetve nyomtatóra legyen kérhető, formázott állapotban.

? SÚGÓ? ↳? EXPORT 2 EXCEL? (HA WEBES A FELÜLET: Output Stream; Mime-type: MS/XLS)

5. Egy/többfelhasználóság

A programnak lehetőleg mind az adatbevitel, mind a lekérdezések szempontjából támogatnia kell a többfelhasználós üzemmódot, azaz az adatbázisnak egy időben több munkatárs által is szerkeszthetőnek kell lennie (6-20 fő).

↳ezért is 1000 a több tábla mert nem kell tárolni ott egész adatbázist.

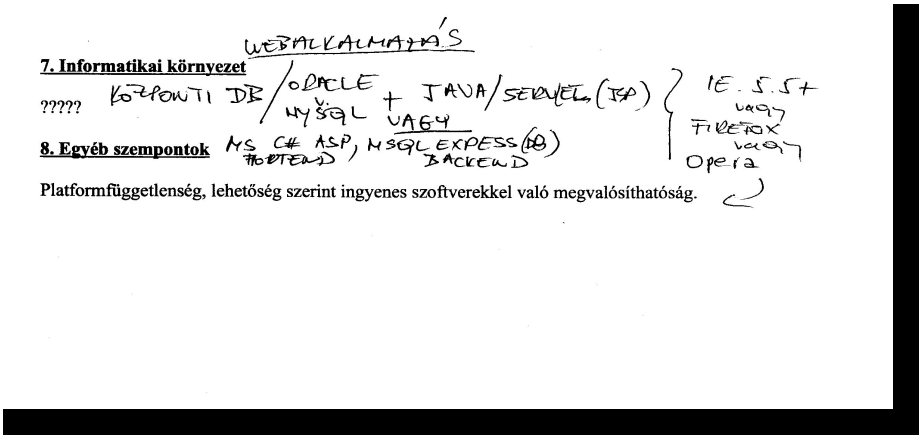
6. Archiválás

Napi archiválás. -> MINDEN TÁBLÁHOZ: LAST ARCHIVED DATE/TIME, EXIT CODE (OK/ERROR)

3 ? LAST ARCHIVED BY?

48. ábra Követelmények meghatározása 3. oldal

4. oldal



49. ábra Követelmények meghatározása 4. oldal

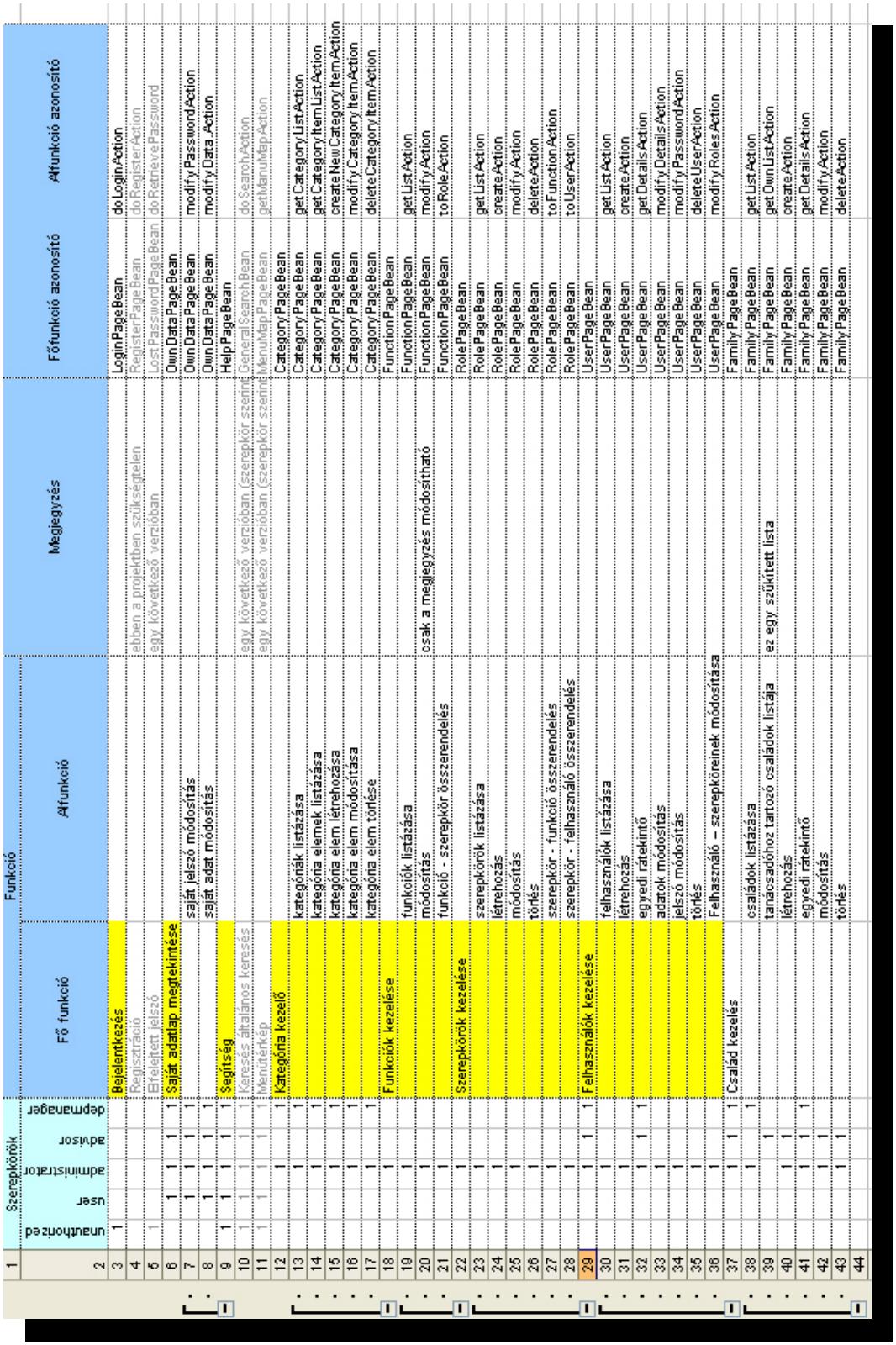
Fogalomtár

1.2.4 Fogalmak értelmezése

Név	Értelmezés
FIKSZ	FIKSZ = Felelősség-Információ-Komptencia-Szülőknek. Ez egy két szakaszból álló tanfolyam, leendő nevelőszülőknek. • Első szakasz: 28 órás döntés-előkészítő tréning. Az első tanfolyamrész után a tréner javaslatot tesz, hogy a nevelőszülők alkalmasak-e feladatukra. • Második szakasz: Elméleti ismeretek. Itt a leendő nevelőszülők jogi, egészségügyi,pedagógiai és pszichológiai ismereteket szeretnek. A tanfolyamról a nevelőszülők FIKSZ tanúsítványt kapnak. A tanúsítvány nélkül nem helyezhetnek ki gyermeket hozzájuk. A FIKSZ tanúsítványt csak egyszer szerzik meg a szülők.
Nevelőszülő	Aki nevelt gyermekekkel foglalkozik. Van FIKSZ tanúsítványa, a tegyesz szerződött partnere. A nevelőcsaládból a férj vagy feleség. (Akivel a tegyesz szerződést köt.)
Gyámhatóság	Az Északalföldi Regionális Államigazgatási Hivatal Gyámhatósága. A nevelőszülőkkel kapcsolatos eljárások engedélyezésében van szerepük.
Működési engedély	A nevelőcsaládban nevelhető gyerekek számát a FIKSZ tanúsítvány létrejöttekor meghatározzák. Ez azonban csupán javaslatként kerül a gyámhatóság elé. A tényleges engedélyezett létszámot ők határozzák meg egy működési engedéllyel.
Fiksz férőhely	A FIKSZ tanúsítványban a család számára javasolt férőhelyszám
Működési engedély szerinti férőhely	Amit a gyámhatóság jóváhagyott. A működési engedély szerinti férőhely jelzi a minkenkori aktuális férőhelyszámot. Ez megegyezik a dátuma szerint legfrissebb – tehát éppen aktuális – „Felülvizsgált férőhely" értékkel. Jól látom?

50. ábra Részlet a fogalomtárból

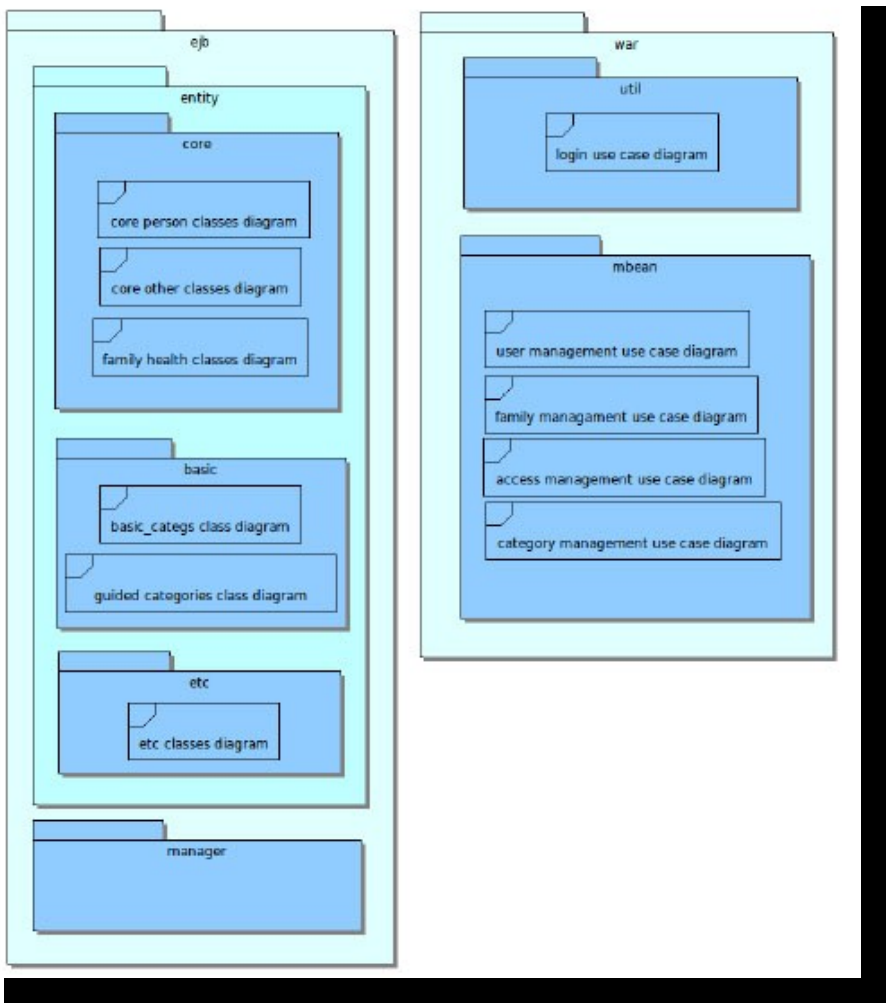
Menürendszer, Szerepkörök és rendszerfunkciók alapértelmezett kapcsolata



	Szerepkörök					Funkció		Megjegyzés	Főfunkció azonosító	Afunkció azonosító
	unauthorized	user	administrator	advisor	depmanager	Fő funkció	Afunkció			
3	1					Bejelentkezés			LoginPageBean	doLoginAction
4								ebben a projektben szükségtelen	RegisterPageBean	doRegisterAction
5	1					Regisztráció			RegisterPageBean	doRegisterAction
6		1	1	1	1	Elfelejtett jelszó		egy következő verzióban	LostPasswordPageBean	doRetrievePassword
7		1	1	1	1	Saját adatlap megtekintése			OwnDataPageBean	
8							saját jelszó módosítás		OwnDataPageBean	modifyPasswordAction
9		1	1	1	1		saját adat módosítás		OwnDataPageBean	modifyData Action
10		1	1	1	1	Segítség			HelpPageBean	
11		1	1	1	1	Keresés általános keresés		egy következő verzióban (szerepkör szerint)	GeneralSearchBean	doSearchAction
12		1	1	1	1	Menünkép		egy következő verzióban (szerepkör szerint)	MenuMapPageBean	getManuMapAction
13				1	1	Kategóna kezelő			CategoryPageBean	
14				1	1		kategóniák listázása		CategoryPageBean	getCategoryListAction
15				1	1		kategóna elemek listázása		CategoryPageBean	getCategoryItemListAction
16				1	1		kategóna elem létrehozása		CategoryPageBean	createNewCategoryItemAction
17				1	1		kategóna elem módosítása		CategoryPageBean	modifyCategoryItemAction
18				1	1		kategóna elem törlése		CategoryPageBean	deleteCategoryItemAction
19				1		Funkciók kezelése			FunctionPageBean	
20				1			funkciók listázása		FunctionPageBean	getListAction
21				1			módosítás	csak a megjegyzés módosítható	FunctionPageBean	modifyAction
22				1			funkció - szerepkör összerendelés		FunctionPageBean	toRoleAction
23				1		Szerepkörök kezelése			RolePageBean	
24				1			szerepkörök listázása		RolePageBean	getListAction
25				1			létrehozás		RolePageBean	createAction
26				1			módosítás		RolePageBean	modifyAction
27				1			törlés		RolePageBean	deleteAction
28				1			szerepkör - funkció összerendelés		RolePageBean	toFunctionAction
29				1			szerepkör - felhasználó összerendelés		RolePageBean	toUserAction
30				1	1	Felhasználók kezelése			UserPageBean	
31				1			felhasználók listázása		UserPageBean	getListAction
32				1			létrehozás		UserPageBean	createAction
33				1	1		egyedi rátekintő		UserPageBean	getDetailsAction
34				1			adatok módosítás		UserPageBean	modifyDetailsAction
35				1			jelszó módosítás		UserPageBean	modifyPasswordAction
36				1			törlés		UserPageBean	deleteUserAction
37				1	1		Felhasználó – szerepköreinek módosítása		UserPageBean	modifyRolesAction
38				1	1	Család kezelés			FamilyPageBean	
39				1			családok listázása		FamilyPageBean	getListAction
40				1	1		tanácsadóhoz tartozó családok listája	ez egy szűkített lista	FamilyPageBean	getOwnListAction
41				1	1		létrehozás		FamilyPageBean	createAction
42				1	1		egyedi rátekintő		FamilyPageBean	getDetailsAction
43				1	1		módosítás		FamilyPageBean	modifyAction
44				1	1		törlés		FamilyPageBean	deleteAction

51. ábra Szerepkörök

Csomagdiagram



52. ábra Csomagdiagram

TranslatedCategoryName osztály

Category.java | CategoryPageBean.java * | TranslatedCategoryName.java

```
package tegyesz.war.mbean;

/**
 *
 * @author Záborski László <http://www.zabo.hu>
 */
public class TranslatedCategoryName {
    private String nativeCategoryName;
    private String value;

    public TranslatedCategoryName() {
    }

    public String getNativeCategoryName() {
        return nativeCategoryName;
    }

    public void setNativeCategoryName(String nativeCategoryName) {
        this.nativeCategoryName = nativeCategoryName;
    }

    public String getValue() {
        return value;
    }

    public void setValue(String value) {
        this.value = value;
    }

}
```

22:36 INS

53. ábra TranslatedCategoryBean osztály

MsgL9n osztály

```
public class MsgL9n {

    protected static ClassLoader getCurrentClassLoader(Object defaultObject) {

        ClassLoader loader = Thread.currentThread().getContextClassLoader();

        if (loader == null) {
            loader = defaultObject.getClass().getClassLoader();
        }

        return loader;
    }
    public static String getMessageResourceString(String bundleName, String key, Object params[], Locale locale)
    public static String getMessage(String msgNativeTitle) {
        FacesContext context = FacesContext.getCurrentInstance();
        String text = getMessageResourceString(context.getApplication().getMessageBundle(),
                msgNativeTitle, null, context.getViewRoot().getLocale());
        return text;
    }
    public static void putFacesErrorMessage(String msgNativeTitle, String msgDetail) {
        FacesMessage msgFaces = new FacesMessage();
        msgFaces.setSeverity(FacesMessage.SEVERITY_ERROR);
        msgFaces.setSummary(getMessage(msgNativeTitle));
        msgFaces.setDetail(msgDetail);
        addMessage(msgFaces);
    }
    protected static void addMessage(FacesMessage message) {...}
    protected static void addMessage(String message) {...}
}
```

54. ábra MsgL9n metódusai 1. oldal

```
    public static String getMessageResourceString(String bundleName, String key, Object params[], Locale locale)

        String text = null;

        ResourceBundle bundle =
                ResourceBundle.getBundle(bundleName, locale,
                getCurrentClassLoader(params));

        try {
            text = bundle.getString(key);
        } catch (MissingResourceException e) {
            text = "?? key " + key + " not found ??";
        }

        if (params != null) {
            MessageFormat mf = new MessageFormat(text, locale);
            text = mf.format(params, new StringBuffer(), null).toString();
        }

        return text;
    }
    public static String getMessage(String msgNativeTitle) {...}
    public static void putFacesErrorMessage(String msgNativeTitle, String msgDetail) {...}
    protected static void addMessage(FacesMessage message) {
        FacesContext.getCurrentInstance().addMessage(null, message);
    }
    protected static void addMessage(String message) {
        FacesMessage msg = new FacesMessage(message);
        FacesContext.getCurrentInstance().addMessage(null, msg);
    }
```

55. ábra MsgL9n metódusai 2. oldal

Köszönetnyilvánítás

Mindenekelőtt szeretnénk megköszönni konzulensünknek, Dr. Adamkó Attilának, hogy nem hagyta, hogy az adott feladatot, melyet ki választottunk évfolyamtársammal [Szikora Zsolt] a lehető legkönnyebb módon elkészíteni, felhasználva az egyetem előtt szerzett tudásunkat. Ő erőltette, és motivált bennünket, hogy a fent megnevezett technológiákat megismerjük, és használjuk. Hálás vagyok neki, mert a szoftverfejlesztés egy másik módszerét ismerhettem meg ezáltal.

Köszönöm a projekttársamnak, Szikora Zsoltnak, a folyamatos együtt működést, a követelmények pontosításában és az EJB konténer elkészítésében való részvételt.

Köszönöm még a Debreceni Egyetem tanárainak, hogy tovább fejlesztették, aktualizálták szakmai tudásomat, ezáltal is versenyképesebbé téve engem az üzleti világban.

Utoljára, de nem utolsó sorban szeretném megköszönni Családomnak, hogy elviselték a hosszú, munka utáni éjszakázásokat, amikor készítettem ezt a feladatot. Külön kiemelném testvéremet, Záborski Éva-t, aki nagyon sokkal támogatott abban, hogy ez a dolgozat ilyen formában elkészüljön.