Szakdolgozat

Ésik Gábor András

Debrecen
2010

Debreceni Egyetem

Informatikai Kar

Szociális intézmény nyilvántartási rendszere

Témavezető
Vágner Anikó Szilvia
egyetemi tanársegéd

Készítette:
Ésik Gábor András
programtervező informatikus(BSc.)

Debrecen
2010

Tartalomjegyzék

I. Bevezetés

Szakdolgozatom témája egy a mindennapi életből vett probléma megoldása az egyetemi tanulmányaim alatt szerzett tudás alapján. A Java nyelvvel több tantárgy keretein belül is foglalkoztam, így nem meglepő, hogy erre a nyelvre esett a választásom, hiszen ez egy olyan nyelv, ami az informatika több rétegében is képviselteti magát, megtalálható az asztali, mobil, üzleti és webes alkalmazások világában. A probléma pedig egy szociális központ nyilvántartásának elektronikus megvalósítása.

Édesanyám munkahelye egy olyan intézmény, melynek a nyilvántartása komplex és bonyolult. Úgy gondoltam, hogy megpróbálom a nyilvántartásuknak egy részét elkészíteni. A teljes nyilvántartás és funkcionalitás megvalósítása jelentősen túlmutatott volna a szakdolgozat keretein. Így jómagam a személyi nyilvántartás megvalósítását tűztem ki célul, amely magában foglalja a szociális központ dolgozóinak és ellátottainak a kezelését, valamint néhány olyan funkciót is elkészítettem, mely jelentősen megkönnyíti az adminisztrációs munkát.

Az elhatározás akkor alakult ki bennem, amikor láttam, hogy egy ilyen összetettebb intézmény esetén milyen sok munkával jár az adatok rendszerezése és karbantartása, valamint egy ellenőrzéskor az adatok visszakeresése és egyeztetése. Magamban elképzeltem, hogy mennyivel egyszerűbb lenne a munkájuk, ha ezeket az adatokat pár kattintással elő tudnák keresni, és pár gomb megnyomása után minden szükséges információ megjelenne előttük. Mivel ez egy szociális központ, ahol a dolgozók a rászorult emberekkel foglalkoznak, az a meglátásom, hogy ha az alkalmazottak minél kevesebb időt töltenek papírmunkával, akkor annál több időt tudnak azzal tölteni, hogy a segítséget igénylő emberek életén valamelyest segítsenek. Véleményem szerinte ez egy értékelhető és nemes cél.

A nyilvántartást egy Java alapú webes alkalmazás fogja kezelni, azért döntöttem a web alkalmazás mellett, mert a mai viszonyok szerint ez a legkeresettebb alkalmazás típus, valamint én is nagy előnyének tartom, hogy hálózat és egy böngésző segítségével szinte bárhonnan elérhető az adott alkalmazás. Továbbá az architektúra lehetővé teszi - hogy idővel, ha az alkalmazás bővülne és az adatok mérete is úgy kívánja – a különböző rétegek külön szerveren való elhelyezését anélkül, hogy az alkalmazáson jelentős módosításokat kellene végezni.

Az alkalmazásban számos olyan technológiát fogok használni melyeket a Java kifejezetten az üzleti rendszerek készítéséhez készített. Az üzleti logikát EJB3-al fogom megvalósítani, az adatok adatbázisba történő leképezését JPA fogja végezni, valamint a megjelenítést és a vezérlést GWT-vel valósítanám meg, amely a Google által készített AJAX framework, ami napjainkban szintén nagy népszerűségnek örvend.

Célom a dolgozatommal az, hogy átfogóbb képet kapjak egy alkalmazás teljes fejlesztési menetéről, valamint, hogy elsajátítsam a gyakorlatban is az itt használt technológiák együttes alkalmazhatóságát.

II. Az adatbázis réteg

a. Igényfelmérés és az adatbázis tervezése

Dolgozatom tárgyát képező intézmény egy létező szociális központ egyszerűsített változata. Célom hogy egy alkalmazás elkészítése, amely képes megfelelni az intézmény nyilvántartási elvárásainak és ezen túlmenően egyszerű és gyors kezelhetőségével megkönnyítheti azon dolgozók munkáját, akik egyébként a papír alapú nyilvántartással végeznék munkájukat. Édesanyám segítségével betekintést nyerhettem a munkahelyének felépítésébe és szerkezetébe. Ennek köszönhetően rálátást kaptam egy ilyen összetett intézmény nyilvántartásának működésébe.

A szociális központ különböző részlegek üzemeltetésével igyekszik segíteni a szociálisan arra rászorultakon. Az intézmény az alábbi részlegekkel rendelkezik:

- Idősek Otthona
- Idősek Klubja
- Házi segítségnyújtás
- Támogató szolgálat
- Fogyatékosok Nappali Klubja
- Szociális Étkeztetés

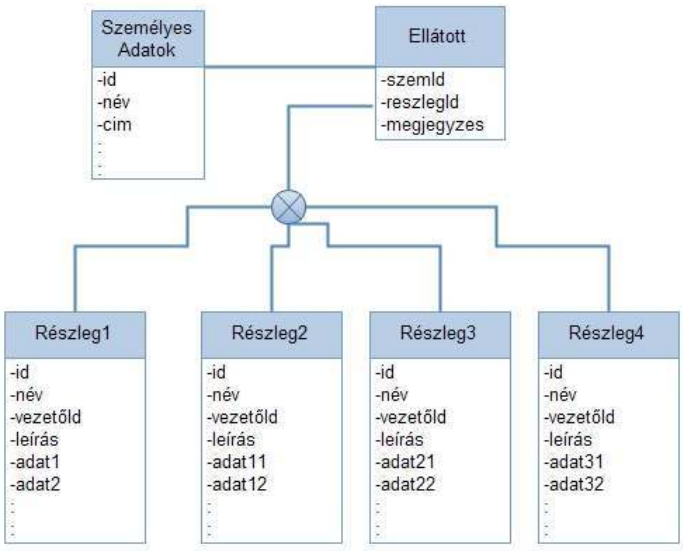
Az elsődleges igény az, hogy a különböz részlegek dolgozóit és ellátottait egyszerűen kezelni tudják, ezalatt értem az adatok bevitelét és az esetleges későbbi előkeresést. Ez a feladat elsőre nem is tűnik olyan nehéznek, azonban amikor hozzáláttam a tervezéshez rögtön szembe találtam magam néhány problémával.

A tervezés első lépése az volt, hogy valamelyest megismerkedtem az intézmény felépítésével, szerkezetével és ezeket leírtam modellek segítségével. A személyi nyilvántartás elkészítéséhez fel kellett mérni, hogy a személyek milyen jellemzőkkel, rendelkezhetnek, és ezeket hogyan lehetne csoportosítani. Alapvetően kettő fő csoportot különböztetek meg, a dolgozók és az ellátottak. Ezt a két típus jelentősen elkülöníthető a róluk eltárolt adatok alapján. Az alkalmazottakról a személyes adatokon kívül a képesítésükről és a beosztásukról tárolunk még információt, míg az ellátottakról attól függően, hogy melyik részleghez tartoznak más és más adatokat tárolhatunk.

A csoportosítás első lépéseként vettem azon adatokat melyek minden egyénnél ugyanúgy tárolásra kerülnek, ezek a személyes adatok. Terveim szerint ezek a személyes adatokat egy külön táblában, entitásban tároltam és a speciális tulajdonságok hivatkozhatnak

ezekre az értékekre. A dolgozók és ellátottak külön egységként való kezelése is kézenfekvő megoldásnak tűnt. A foglalkoztatott személyek rögzített adatai egységesnek mondhatók, míg a szolgáltatásokat igénybe vevők nyilvántartott adatai szolgáltatási területenként eltérőek. Ezen probléma megoldására többféle megközelítés is felvetődött bennem, elsődleges feladatom volt, hogy megvizsgáljam e lehetőségek hatékonyságát és megvalósíthatóságát.

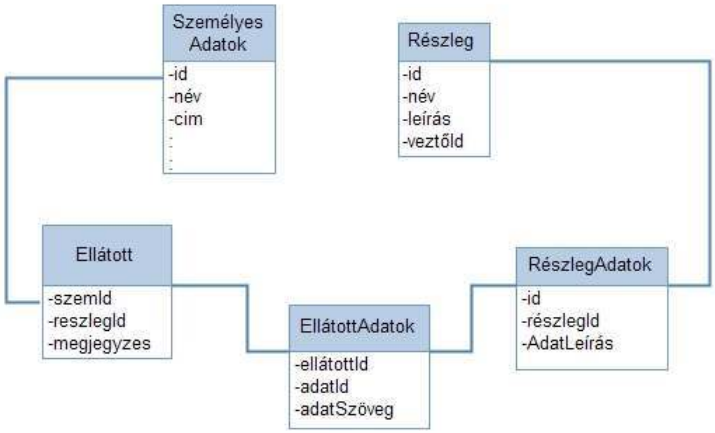
A nyilvántartás modellezésére három lehetőség vetődött fel bennem, ezek megvizsgálása után döntöttem a végleges modellről. A fő problémám az volt, hogy a különböző részlegeken nyilvántartott ellátottakról más és más adatokat tárolhatunk. A legegyszerűbb megvalósításnak az tűnt, ha minden egyes részleget egy külön entitás osztály reprezentál, és ekkor a hierarchia leképezését az „egy tábla egy osztályhierarchia" megvalósítás esetén kapunk egy nagy és összetett táblát, ami esetlegesen tartalmazhat null elemeket. Azonban ha nem ezt a leképezési stratégiát választjuk, akkor az a helyzet áll elő, hogy minden egyes részleg külön táblában fog helyet kapni, és minden egyes tábla tartalmazni fogja a saját részleghez szükséges adatokat. Már látszik, hogy nem ez a leg optimálisabb megoldás, mivel egy nagyon nagy és hézagos táblát, vagy pedig nagyon sok táblát fog eredményezni.



1. ábra Az első stratégia, külön táblák esetén

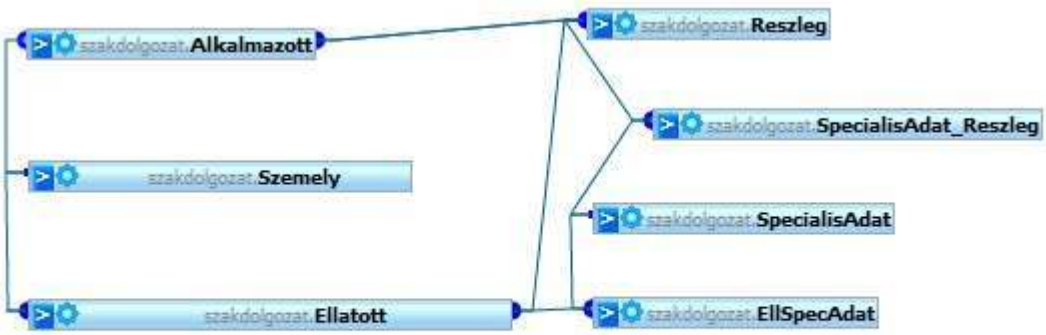
A második megoldási lehetőség egy olyan leképezés, amelynél a részlegekről nyilvántartott információkat mind eltárolom egy táblában, továbbá az ellátottaknak is lesz egy külön táblája, minimum egy id és részleg_id mezővel. Ezek után, létrehozva egy kapcsolótáblát ahol összekapcsolom az ellátottakat és az adott részlegnek megfelelően a róluk

rögzített adatokat. Itt persze szükség lehet arra, hogy nyilvántartsuk azt is, hogy mely részlegről milyen adatokat tartunk nyilván. Erre két megoldás áll rendelkezésünkre, vagy minden nyilvántartott adatról eltároljuk azt, hogy mely részleghez tartozik, vagy ha több részleghez is tartozhat ugyanazon adat, akkor érdemes ehhez is egy külön kapcsolótáblát létrehozni.



2. ábra A minden adat kapcsolótáblában stratégia

A harmadik lehetőség az előző két módszer ötvözése. Megpróbáltam minél több közös rögzítendő adatot összegyűjteni és belerakni az ellátott osztályba, valamint létrehoztam egy speciális_adatok táblát, ahol rögzítem azon adatokat, amelyek oly módon speciálisak, hogy minden részlegnél különbözőek lehetnek, esetleg csak egy-egy részlegnél kerül rögzítésre. Elsődleges célom ennél a módszernél az volt, hogy úgy alakítsam ki az entitás osztályokat, amikből majd adatbázis táblák generálódnak, hogy azok jól reprezentálják a szükséges részlegeket, valamint ha a későbbieken új részlegek kerülnek bevezetésre, akkor azok felvehetőek lesznek a rendszer minimális módosítása mellett.



3. ábra A harmadik megolás (részletesen a dolgozat végén)

b. A perzisztencia megvalósításának eszköze JPA

Az adatok adatbázisban történő tárolását a JPA (Java Persistence API) segítségével valósítottam meg. A JPA a Java által szabványosított perzisztencia keretrendszer. A perzisztencia kezelés elsődleges feladata hogy a programozó számára leegyszerűsítse a programban használt objektumok adatbázisban történő tárolását. Mivel napjainkban a relációs adatbázisok vannak elterjedve, ezért az objektum orientált szemléletet valahogy egyeztetni kell a relációs személettel. Ezt nevezik objektum-relációs leképezésnek, aminek a megvalósítására alkották meg a különféle perzisztencia providereket, mint például a Hibernate vagy a TopLink valamint az Enterprise Java Beans Entity is e célt szolgáljta. A különböző eszközök sokszor eltérően kezeltek bizonyos dolgokat, ezt elégelte meg a Java amikor úgy döntött hogy szabványosítja a perzisztencia kezelést, összeszedte az eddigi megoldások előnyeit és létrehoztak egy egységesített keretrendszert, így született meg a JPA.

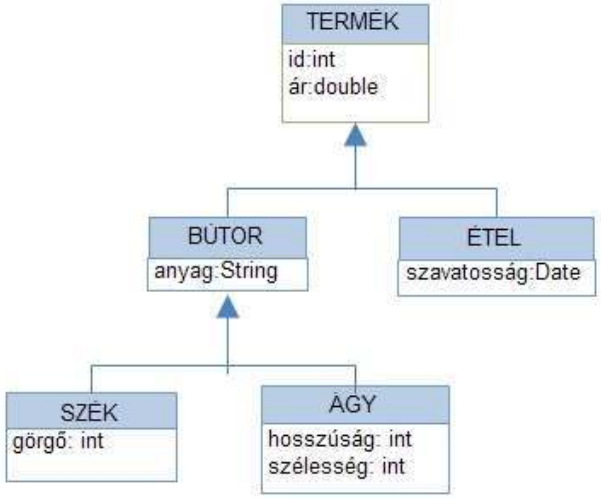
Az EJB3 bevezetésekor a JPA is leegyszerűsödött olyan szinten, hogy azt már egyszerűen, annotációk segítségével használni lehetett. JPA esetén a perzisztens objektumok egyszerű POJO (Plain Old Java Object) osztályok példányai, így használhatók JAVA SE környezetben is. A keretrendszer az adatbázisban tárolni kívánt objektumok elnevezésére az entitást (entity) használja. Egy egyszerű Java-osztályt könnyedén, annotációk felhasználásával EJB3-as entitássá tudjuk alakítani. Egy entitás osztály egy speciális Java Bean, minden adattagja rendelkezik lekérdező és beállító metódussal. Az osztály deklarációja előtt használnunk kell az @Entity annotációt, amit a javax.persistance csomagból importálunk be, ez alapján tudja a rendszer, hogy ez az osztály egy adatbázisba is leképezendő osztály. Az elsődleges kulcsot az @Id annotáció segítségével definiálhatjuk, abban az esetben, ha automatikusan generált id-ról van szó, akkor be tudjuk állítani hogy ezt majd az adatbázis kezelje a @GeneratedValue segítségével. Lényegében annotációk segítségével minden olyan dolgot be tudunk állítani amit egy adatbázis táblán is be lehet.

A keretrendszer megoldást kínál az entitások örökítésére, amelyet az eddigi megoldások nem tökéletesen vagy egyáltalán nem kezeltek le. Mivel az objektum orientált világban az öröklődés egy igazán erőteljes eszköz, sok előnyünk származik abból, hogy ezeket egy az egyben át tudja vinni a rendszer relációs sémára, a programozó közreműködése nélkül. Ezáltal az EJB3-as entitások tetszőleges hierarchiája megvalósítható, melyet az entitások POJO jellegéből adódóan egyszerűen az extends kulcsszót használva leszármazott osztályokat tudunk létrehozni. A már megszokott öröklődéshez képest itt egyedül a hierarchia

adatbázisba történő leképezése jelent újdonságot, erre több lehetőség is rendelkezésünkre áll, melyet a megfelelő annotáció segítségével a perzisztencia kezelő meg is valósít számunkra. Az öröklési hierarchia leképezéséhez három stratégia áll a rendelkezésünkre:

- Egy tábla egy osztályhierarchia (InheritanceType.SINGLE_TABLE)
- Külön gyermekosztályonként (InheritanceType.SINGLE_TABLE)
- Egy tábla egy konkrét gyermekosztályhoz (InheritanceType.SINGLE_TABLE)

A leképezési stratégia kiválasztásához annyit kell tennünk, hogy az osztálydefiníció elején az @Inheritance annotáció után zárójelben megadjuk a stratégia típusát.



4.Az öröklést bemutatásul szolgáló osztályhierarchia

Az első stratégia előnye a hatékony működés, ami abból adódik, hogy az osztályhierarchia egy táblába fog kerülni, így az adatbázisban egy táblában fog szerepelni az ősosztály és az összes leszármazottja. Ezzel elkerülhetők a különféle tábla összekapcsolások, amik akár lassíthatják is a működést. Ezzel szemben a módszer hátránya is ebből fog adódni, mivel a táblának sok oszlopa lesz és az osztályok különbségei miatt elég sok null érték lesz majd az adatbázistáblában, így elég nagy lesz a táblánk mérete és a szerkezete elég lyukas lesz. Felmerül azonban egy fontos kérdés, hogyan fogjuk tudni eldönteni, hogy az éppen kiolvasott sor milyen osztálynak feleltethető meg. Mivel egy közös táblában vannak így a tábla neve csak az ősosztályt tudja egyértelműen azonosítani.

Ennek a problémának a megoldására bevezettek egy plusz oszlopot a táblában, ami a típus információt fogja hordozni minden egyes sor esetén, ezt nevezik diszkriminátornak. Ennek segítségével már minden sorról meg tudjuk mondani pontosan, hogy melyik osztálynak feleltethető meg.

ID	ÁR	ANYAG	GÖRGŐK	HOSSZÚSÁG	SZÉLESSÉG	SZAVATOSSÁG	TÍPUS
1	500	MŰANYAG	0	NULL	NULL	NULL	SZÉK
2	1000	FA	NULL	200	140	NULL	ÁGY

A második stratégia, mint ahogy a nevéből is látszik – külön gyermekosztály – megpróbálja kiküszöbölni az első módszer hibáját, mégpedig úgy hogy minden osztályt külön táblában helyez el, így minden táblában csak a tényleg fontos adatok lesznek tárolva, ezzel eléri azt, hogy nem lesznek null értékek tárolva. A táblák közötti kapcsolatot az elsődleges kulcs biztosítja, ugyanis a gyermekosztályokban az elsődleges kulcs egyben idegen kulcsként is funkcionál, ennek segítségével a leszármazott osztály összekapcsolható az ősosztályával. Ebből adódik a módszer hátránya is, ha az osztály hierarchia túl nagy, akkor az adatok előállításához sok táblát kell összekapcsolni, ami könnyen rossz teljesítményhez vezethet.

TERMÉK tábla

ID	ÁR	TÍPUS
1	500	SZÉK
2	1000	ÁGY

BÚTOR tábla

ID	ANYAG
1	MŰANYAG
2	FA

ÉTEL tábla

ID	SZAVATOSSÁG

SZÉK tábla

ID	GÖRGŐK
1	0

ÁGY tábla

ID	HOSSZÚSÁG	SZÉLESSÉG
2	200	140

A harmadik stratégia elve az, hogy minden gyermekosztályhoz külön táblát készít, amely tartalmazza az adott osztály összes attribútumát, így nincs szükség a táblák összekapcsolására. Így ez a megoldás hasonlóan hatékony, mint az első stratégia, viszont nem zárja ki a nem nullázható oszlopok készítését. Ezzel az eljárással viszont elég nehéz megoldani a polimorfizmus támogatását. Ugyan használható ez a megoldás, de nem minden perzisztencia provider lesz képes kezelni ezt a fajta leképezést.

TERMÉK tábla	
ID	ÁR

BÚTOR tábla		
ID	ÁR	ANYAG

ÉTEL tábla		
ID	ÁR	SZAVATOSSÁG

SZÉK tábla			
ID	ÁR	ANYAG	GÖRGŐK
1	500	MŰANYAG	0

ÁGY tábla				
ID	ÁR	ANYAG	HOSSZÚSÁG	SZÉLESSÉG
2	1000	FA	200	140

A kapcsolatok kezelésére is ad eszközt a kezünkbe a JPA. Tetszőleges relációs adatbázis kapcsolat definiálható entitások között, amelyek majd ezeknek megfelelően fognak leképződni az adatbázis táblák szintjére. Az egyedtípusok maximális előfordulásának megfelelően beszélhetünk:

- @OneToOne (egy az egyhez)
- @OneToMany (egy a sokhoz)
- @ManyToOne (több az egyhez)
- @ManyToMany (sok a sokhoz)

típusú kapcsolatokról. A kapcsolat lehet egyirányú vagy kétirányú, az első esetben csak az egyik entitásból elérhető a másik, míg az utóbbi esetben mind a két entitás elérheti egymást. Egyirányú esetben a tulajdonos osztály egyértelmű, a kétirányú kapcsolat alkalmazásakor az alábbi szabályok közül választhatunk:

- Egy-egy kapcsolat esetén a tulajdonos az az oldal aki az idegen kulcsot tartalmazza
- Egy-több esetén a több oldal a tulajdonos (aki a @ManyToOne-t tartalmazza)
- Több-több választásakor egy un. kapcsolótábla tartalmazza az idegen kulcsokat, így a tulajdonos bármelyik oldal lehet

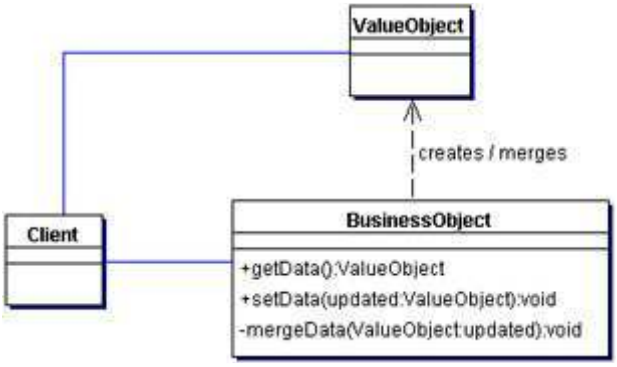
Az entitás osztályokban a több oldalon az adott elemhez kapcsolódó entitásokat valamilyen kollekcióban tároljuk, amelyet egyszerűen csak úgy kezelünk, mint bármely más Java –beli kollekciót. A kapcsolatok kezelésekor fontos kérdés, az, hogy amikor az entitások a memóriába kerülnek, mi történjen a kapcsolódó entitásokkal. Erre a keretrendszer két megoldást szolgáltat, ez a két megoldás a mohó (EAGER) és a lusta (LAZY) betöltés. A

mohó megoldás esetén az entitáshoz kapcsolódó elemek is rögtön betöltődnek a memóriába, míg a lusta megoldás esetén csak akkor töltődnek be, ha ténylegesen szükség van rájuk. Az első eset előnye hogy egyetlen lekérdezés kell hozzá, hátránya azonban az, hogy felesleges memória használattal járhat. Az utóbbi megoldás előnye pedig pont fordított, nem jár felesleges memóriafoglalással, viszont utólagos lekérdezés szükséges a kollekciók feltöltéséhez.

c. Kapcsolat az üzleti réteg felé

Az EJB3-as entitás osztályaink, számunkra megfelelő módon reprezentálják a szükséges adatokat, viszont az nem előnyös, ha ezek az entitás osztályok utaznak majd a hálózaton, az ügyfél és az üzleti logika valamint az adatbázis között. Ez a megoldás bizonyos problémákat vet fel, ugyanis az entitás osztályok olyan információkat is tartalmaznak, amiket nem feltétlenül előnyös, ha bárki, aki a programot használja, hozzáfér ezekhez az információkhoz. Továbbá bizonyos esetekben nem szükséges minden adatbázisban tárolt adatot a felhasználók elé tárni. Ezen problémák megoldására fejlesztették ki a Transfer Object J2EE Patterns-t, melynek célja az entitás osztályok adattagjait egységbe foglalják. Ezek speciális ValueObject –ek, vagy más néven DTO (Data Transfer Object)-k. Ezek egyszerű java osztályok adattagokkal és getter, setter metódusokkal.

A következő lépés az alkalmazás fejlesztése során, hogy a meglévő entitás osztályoknak megfelelően minden egyes osztályhoz hozzunk létre value osztályokat, mert az üzleti logika felépítésénél szükségünk lesz ezekre az állományokra. Az üzleti modell a program felhasználóinak minden esetben csak az érték objektumokat szolgáltatják eredményül, valamint a felhasználóktól ilyen típusú paramétereket vár. Az üzleti réteg valójában az adatbázissal kapcsolatban lévő entitás osztályok segítségével operál, azonban a kliensek felé csak az érték osztályokkal tartják a kapcsolatot és ez a kliens oldal felől is így van. Ennek köszönhetően az entitás osztályok szerkezete, ezáltal a konkrét adatbázis megvalósítás teljes mértékben elrejtésre kerül, ami egy összetettebb és komolyabb alkalmazásban szükséges elvárás már manapság.



5. ábra ValueObject

Ennek a kezelésére létrehoztam egy külön vo csomagot a program struktúrában, majd egy speciális Converter osztályt definiáltam, két statikus metódussal, melyek elvégzi a számomra szükséges konverziókat. A program további részében ahol egy entitáson végzek, műveletet aztán az eredményét közölni szeretném, a felhasználóval egyszerűen csak meghívom a Converter.entity2vo(Entitás e) metódust és már el is végzi az átkódolást. Ha a későbbiekben változás történik az entitás osztályokban, akkor elég csak a Converter osztályt módosítani és az alkalmazás többi része észre sem veszi, hogy módosítás történt. A polimorfizmusnak köszönhetően elég egyetlen entity2vo metódust definiálnom és megjegyeznem, mialatt annyi számú ilyen metódus van, ahány entitás osztályom van. Ez jócskán megkönnyíti a fejlesztést, ugyanis nem kell attól félni rossz konvertáló metódust hívok meg, mivel csak ez az egy van, és attól függően, hogy milyen típusú paramétert kap fog lefutni a megfelelő konverziós metódus.

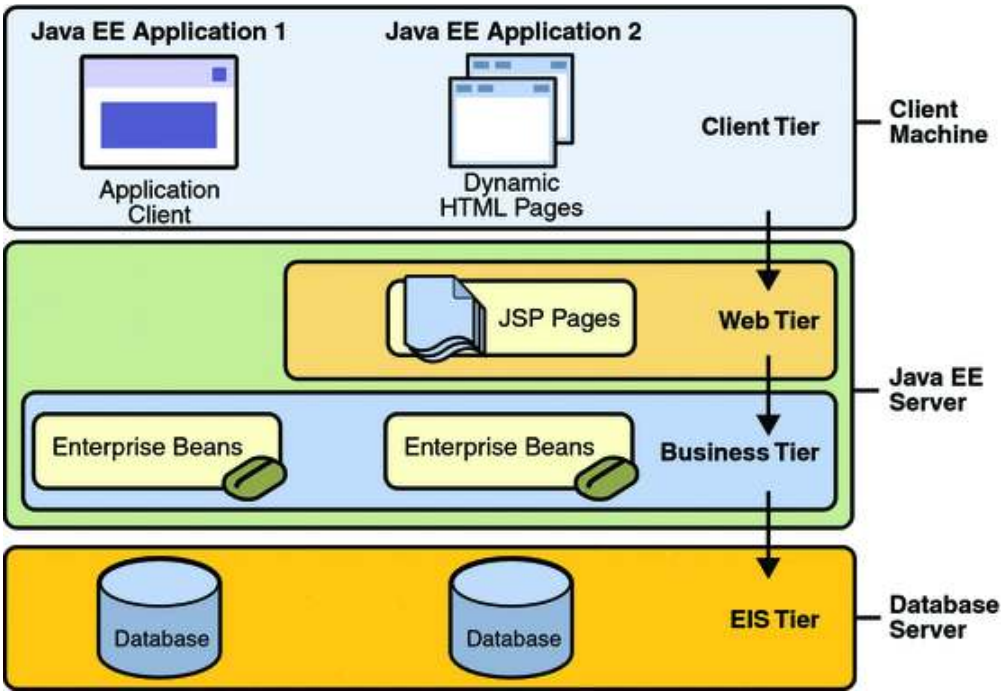
```
package vo;
import entity.*;
public final class Converter {
	/*
	 * Ez az osztály fogja végezni a konverziót az entitás és a vo osztályok között
	*/
	public static SzemelyVO entity2vo(Szemely sz){
		SzemelyVO szvo = new SzemelyVO();

		szvo.setId(sz.getId());
		szvo.setNev(sz.getNev());
		……
		return szvo;
	}
	public static ReszlegVO entity2vo(Reszleg r){
		ReszlegVO rvo = new ReszlegVO();

		rvo.setId(r.getId());
		rvo.setNev(r.getNev());
		…..
		return rvo;
	}
}
```

III. Az üzleti réteg megvalósítása

a. JEE

Az alkalmazás tervezésénél követtem az MVC tervezési minta követelményeit. Ez a minta azt vallja, hogy az alkalmazást osszuk fel rétegekre, legyen külön Model-View-Controller réteg, és mindegyik csak a saját feladatával foglalkozzon. Ha egy alkalmazás középpontjában az adatkezelés áll, és fontosak a velük végzett műveletek akkor célszerű elkülöníteni egymástól az adat és adatkezelési valamint a megjelenítési folyamatokat. Részemről az adatkezelés szabványos Java EE eszközökkel fogom megvalósítani, hiszen ezek az eszközök direkt e célokra lettek kifejlesztve.

A java erősségét az Enterprise (vállalati) kiadás képezi. A SUN J2EE néven rendszerezte azon programkönyvtárakat és keretrendszereket, amelyek elősegíthetik és megkönnyíthetik egy cég működését. A fő előny a szétválasztott rétegek – megjelenés, vezérlés, adatbázis – amelyek előre meghatározott interfészek segítségével kommunikálnak egymással.



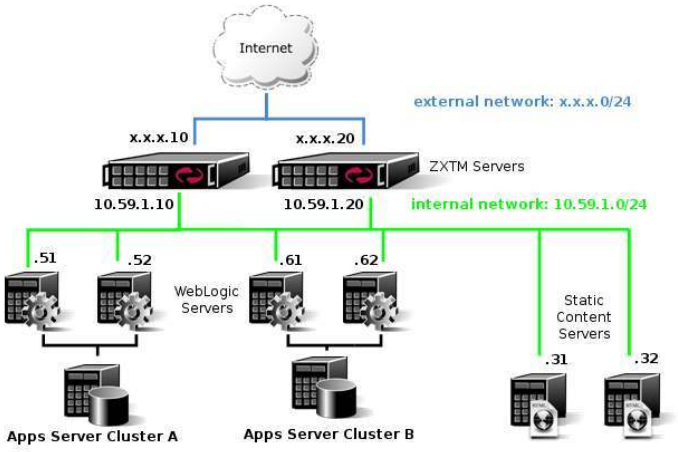
6 Java EE alkalmazás rétegei

A JEE fő előnye a Standard Java-val szemben, hogy nagyban gondolkodik, valamint megkönnyíti a nagyban gondolkodást. Egy Standard Java program esetén általában a programot egy ember használja egy gépen egy időben és erre a célra ez tökéletesen meg is felel, azonban ha egy alkalmazást egy időben több száz, esetleg több ezer - vagy még több –

ember szeretne használni akkor ez a verzió már nem nyújt kellő támogatást nekünk. A JEE azonban gondol ezekre az igényekre és megpróbálja maximálisan kielégíteni ezen speciális igényeinket. Itt az elsődleges cél a magas rendelkezésre állás, a több gépen való futtatás lehetősége, több adatbázis egyidejű használata valamint a szoftverkomponensek elosztott elérése. Az Enterprise környezet működéséhez nem elég a JRE sőt a JDK sem, szükség van egy speciális eszközre, amit úgy hívnak, hogy alkalmazás szerver, amely képes futtatni az Enterprise alkalmazásainkat. Alkalmazás szerverek:

- Glassfish (SUN)
- JBoss (Red Hat)
- WebSphere(IBM)
- WebLogic (ORACLE)

Az alkalmazás szerver (Application Server = AS) feladata, hogy lehetővé tegyék az elkészített üzleti alkalmazások futtatását. Ez elsőre nem tűnik nagy feladatnak, azonban ha létrehozunk egy több gépből álló fürtöt, amire feltelepítjük az alkalmazásszervert és ezeket összekapcsoljuk, akkor képesek lesznek egy rendszerként működni, majd erre az összetett rendszerre feltelepítjük az alkalmazásunkat, mely ezek után képes lesz minden gépen futni, így már nem fog gondot okozni a megnövekedett felhasználószám, sőt esetleg egy gép kiesése sem fogja felfüggeszteni az alkalmazás működését. Az alkalmazás szerver ilyen fürtözött rendszerben is képes több adatbázist is elérni, a megfelelően beállított adatforrások segítségével. Az egyszerű adminisztrációs felület segítségével az adminisztrátorok könnyedén karban tudják tartani még az ilyen összetett rendszereket is.



7. ábra Fürtözött alkalmazás szerver

b. EJB

Az Enterprise Java Beanek (EJB-k) nagy előnye, hogy szabványos felülettel rendelkeznek, és különféle szolgáltatásokat képesek nyújtani, ezáltal nagyon hasznos komponensek készíthetőek a felhasználásukkal, melyeket később akár több rendszer is képes lesz használni. Az általam használt verzió az EJB3, a 2-es verzióhoz képest nagyon leegyszerűsítették a használatát, kevesebb interfészt kell készítenünk és implementálnunk valamint xml-fájlok készítése alól is felment minket az új verzió, pontosabban:

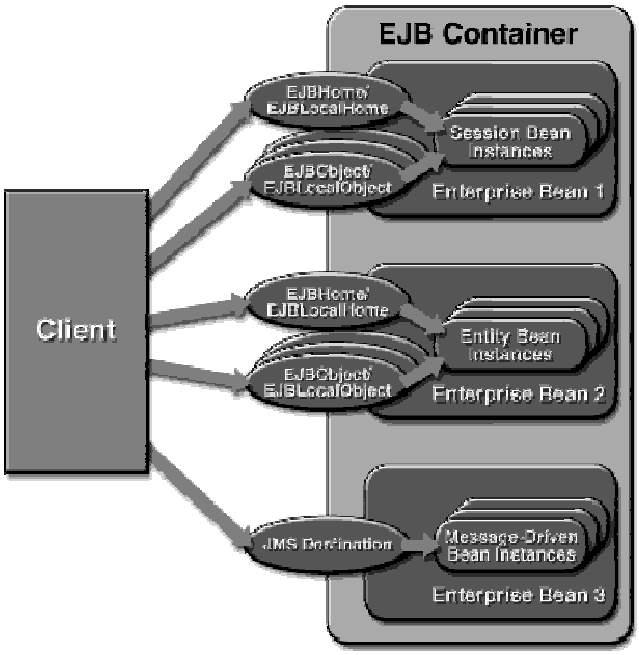
- A sok fájl kezelés minimum 3 java 2 xml.
- A bean osztályoknak kötelezően implementálni kellett a javax.ejbSession/Entiti/MessageDrivenBean interfészeket, így bizonyos metódusokat akkor is meg kellett valósítani amikor azok lényegi funkcionalitást nem is valósítottak meg.
- Az üzleti metódusok meghívása előtt több sornyi kódot kellett írni. Meg kellett keresni JNDI-API hívásokon keresztül a szükséges home interfészt, majd meghívni ennek a create() metódusát, ezek után létrejön az üzleti objektumunk aminek így már meghívhatjuk a szükséges metódusait.
- Specifikáció miatt a használatához szükséges volt DTO-k létrehozása, melyek némi redundanciát jelenthetnek.
- Az EJB-k futtatásához EJB-konténerre van szükség, így a tesztelése elég komplikált lehet. Tesztelhetünk lokális interfészen amihez szükséges egy másik komponens, szervlet vagy JSP.

Az EJB3 mindezekre megoldást nyújt nekünk, mindamellett hogy a technológiából származó előnyöket (rugalmas, deklaratív, skálázható) továbbra is megtartja. Az egyszerűsítések kulcsa a Java5-ben bevezetésre kerülő annotációk. A szabvány definiálta a különböző annotációkat, aztán az alkalmazásszerver gyártók feladata volt megvalósítani ezen annotációk feldolgozását, értelmezését. Az annotációk használata mellett továbbra is megmaradt a lehetőségünk hogy bizonyos konfigurációs beállításokat xml telepítés leírókban is megadhatunk.

Az EJB-k három fő részre oszthatóak Entity, Session, Message Beanek. Ezek mind más és más tulajdonságokkal, előnyökkel rendelkeznek, ebből adódóan más a felhasználási területük is. Az Entity bean az üzleti modell megvalósítására alkalmas, azaz entitásokat reprezentálnak. Az entitás a perzisztenciáért felelős, gyakorlatilag a relációs adatbázis egy

sorát képes egy objektumként a memóriában tárolni. Az alkalmazásom is entity bean-ekkel valósítja meg a perzisztencia kezelését.

A session bean –ek a legegyszerűbb EJB-típus. Üzleti folyamatokat reprezentálnak, metódusaik alkotják az üzleti alkalmazás által nyújtott szolgáltatások összességét. Egy Sesion Bean rendelkezhet állapottal vagy lehet állapotmentes, ennek megfelelően beszélhetünk Stateless és Statefull Session Bean-ekről. Az állapotmentes bean csak metódusokkal rendelkezik, adattagokkal nem, így egy a kliens számára lényegtelen hogy a példányfarmban lévő bean-ek közül melyikkel kommunikál éppen, mivel számára mind ugyan olyan. Ezzel szemben az állapottal rendelkező bean-nek már vannak adattagjai így ha egy kliens egy bean-t használ a példányfarmról akkor ő csak azt az egyet használhatja mivel az ő számára ez az egy tartalmazza a szükséges adattagokat.

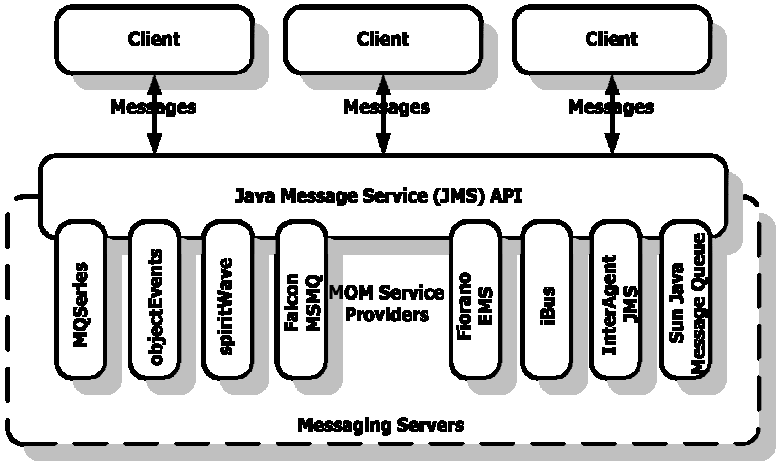


8. ábra Enterprise Bean példányfarmok

Az ábrán jól látható hogy egy adott Bean-ből az alkalmazás szerver több példányt is létrehoz az úgynevezett példányfarmján és ezeket használja fel az érkező kérések kiszolgálásra. Az állapotmentes Sesssion Bean előnye itt jól látszik, ugyanis ezek folyamatosan tudjkák kiszolgálni a kéréseket, míg adott darabszámú állapottal rendelkező bean csak adott darabszámú kérést tud egyszerre kiszolgálni.

A harmadik típus a Message-Driven Bean, azaz üzenetvezérlet/eseményvezérelt bean. Feladata, hogy valamilyen üzenetre reagáljon aszinkron módon, a forrás általában egy JMS

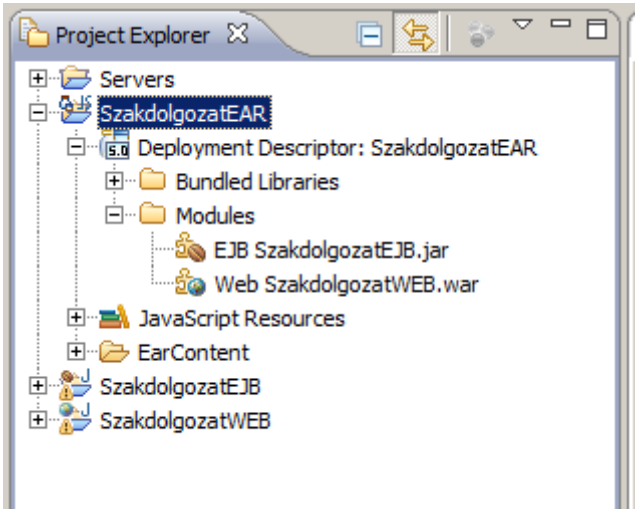
(Java Message Service) üzenet. A JMS-ről most elég annyit tudnunk, hogy úgy működik, mint egy levelezőlista: a JMS-ben topic-nak hívott témákra akármilyen szerkezetű adatot lehet küldeni, amelyek sorba állnak, majd a témára feliratkozott alkalmazások megkapják őket. Eseményvezérelt bean lehet feliratkozva egy témára, ekkor a JMS üzenetek hatására aktivizálódik. A bean életciklusát a konténer és a beérkező üzenetek vezérlik. Üzenet érkezésekor a konténer szerez egy szabad bean-t és odaadja az üzenetet. Az eseményvezérelt bean nagyon hasonlít az állapotnélküli session bean-re. Az ilyen üzenetvezérlet megoldásokat az olyan alkalmazások használják ahol nem megengedhető, hogy az egyes üzenetek elvesszenek, vagy valamilyen hiba miatt ne kerüljenek feldolgozásra. Például egy közösségi oldal számára nem szükséges ezt a technológiát használni, hiszen nem jelent nagy problémát, ha egy kép nem jelenik meg első kattintásra, viszont nagyon is jó megoldás ez a banki rendszerek megvalósítására. Ha indítunk egy tranzakciót, az nem biztos, hogy azonnal végre tud hajtódni – már nincs feldolgozási idő, hétvége van – azonban nagyon fontos hogy a művelet el legyen végezve. Ilyenkor elküldjük a kérésünket az átutalásról egy üzenet formájában, amely a csatornára kerülést követően le is mentődhet adatbázisba, a biztonság érdekében, aztán ha a bank hétfőn kinyit, elkezdi feldolgozni az üzeneteket, amelyek a csatornán vannak. Ez a módszer elég erőforrás igényes és nem minden esetben gyors, viszont megbízható és biztonságos, az elküldött üzenet mindaddig a csatornán marad, amíg a vevő felől nem érkezett sikeres feldolgozás jel, így ha valami hiba történik, újra tudja küldeni. Látható hogy ennek a technológiának is megvannak az előnyei és a hátrányai is, az adott feladat határozza meg hogy szükséges e ilyen szintű biztonság, avagy elfogadható ha egyes kérések néha nem érnek célba, és ilyenkor megkérjük a felhasználót hogy frissítse az oldalt, vagy kattintson újra a linkre.



9. ábra JMS Architecture

c. Alkalmazásom üzleti rétege

Alkalmazásom fejlesztéséhez az Eclipse fejlesztői környezetet használtam, mivel nagyon sok plugin van hozzá, ami segíti a fejlesztést. Először létrehoztam egy EJB projektet, majd egy Dynamic Web projektet. Már ebből is látszik, hogy a megjelenítést és a modellkezelést két külön projekt fogja kezelni. Ezek után létrehoztam egy EAR (Enterprise ARchive) projektet, aminek az lesz a szerepe, hogy magában foglalja az előbb létrehozott két projektemet. Lényegében ez az ear egy egyszerű archív állomány, mely annyit csinál hogy készít egy jar állományt az ejb projektből, egy war állományt a web projektből és ezeket egységbe zárja, az így megkapott ear-t egyszerűen fel lehet tölteni az alkalmazásszerverre és onnan már futtatható is.

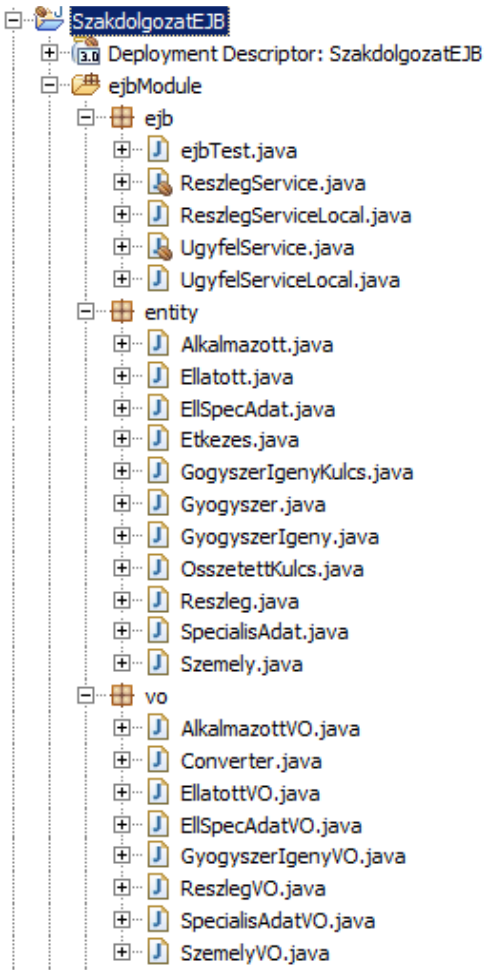


10. ábra EAR projekt struktúra

Az EJB projektem 3 csomagot tartalmaz – ejb, entity, vo - ezek csoportosítják a különböző feladatokat megvalósító osztálydefiníciókat.

- ejb csomag: Itt találhatóak az üzleti folyamatokat leíró interfészek, és az azokat megvalósító service osztályok.
- entity csomag: Itt kaptak helyet az entitás osztályok, amelyek az adatbázis reprezentációt, és a táblák közötti kapcsolatokat definiálják.
- vo csomag: Ez a csomag tartalmazza az entitások szállító osztályait (vo), valamint az ezek konvertálását elvégző Converter osztályt.

Az ejb csomagban azért vannak külön Service és SercieLocal állományok, mert a Local végű állományok csak interfészek, amelyek leírják, hogy az adott Service milyen metódusokkal kell, hogy rendelkezzen, és a Service osztály pedig implementálja a ServiceLocal interfészt és megvalósítja az ott definiált metódusokat. Az interfészeknek később még nagy szerepük lesz, ugyanis a web projekt már csak az interfészeken keresztül fog kommunikálni az üzleti réteggel.



11. ábra EJB Prjojekt Struktura

Próbáltam minden osztálynak „beszédes” nevet adni, hogy ha a későbbiekben hiba következik be, akkor már ránézésre tudjam, hogy hol kell keresni. Ilyen rétegelt architektúrájú alkalmazások fejlesztésénél a hibakeresés és a hibajavítás is jóval összetettebb dolog, mint egy egyszerű alkalmazásnál. Esetemben a ReszlegServiceLocal.java írja le a részlegekhez kapcsolódó metódusok specifikációját, míg az UgyfelServiceLocal.java pedig az intézmény „ügyfeleihez” (alkalmazott, ellátott) kapcsolódó folyamatokat definiálják.

A fentebb említett fájlokban specifikált metódusokat a ReszlegService és az UgyfelService osztályok implementálják, a tényleges megvalósítás tehát itt történik meg.

```
package ejb;

@Local
public interface ReszlegServiceLocal {

	Reszleg getReszlegByID(String id);
	void addReszleg(ReszlegVO reszleg);
	void updateReszleg(ReszlegVO reszleg);
	Collection<EllatottVO> getEllatottakByReszlegId(String id);
	Collection<AlkalmazottVO> getAlkalmazottakByReszlegId(String id);
}
```

A fenti kódrészlet mutatja, be hogyan is néz ki egy ilyen üzleti logikát leíró interfész. A @Local annotáció azt mondja meg az alkalmazás szervernek, hogy aki ezt az interfészt implementálja az egy helyi (lokális) Bean lesz, ami azt jelenti, hogy ahonnan hívni fogják ezeket a metódusokat – web réteg – az ugyanazon az alkalmazás szerveren lesz tárolva – futatva, mint az üzleti réteg. Nagyobb rendszereknél ahol ezek a rétegek már külön gépekre kerülnek, ott természetesen ez a megoldás nem állja meg a helyét, ott már a @Remote annotációval ellátott interfészeket kiterjesztő üzleti logikát készítenek, aztán az alkalmazás szerver elvégzi a beállításokat és a szolgáltatások távolról – más gépekről – is hívhatóvá vállnak.

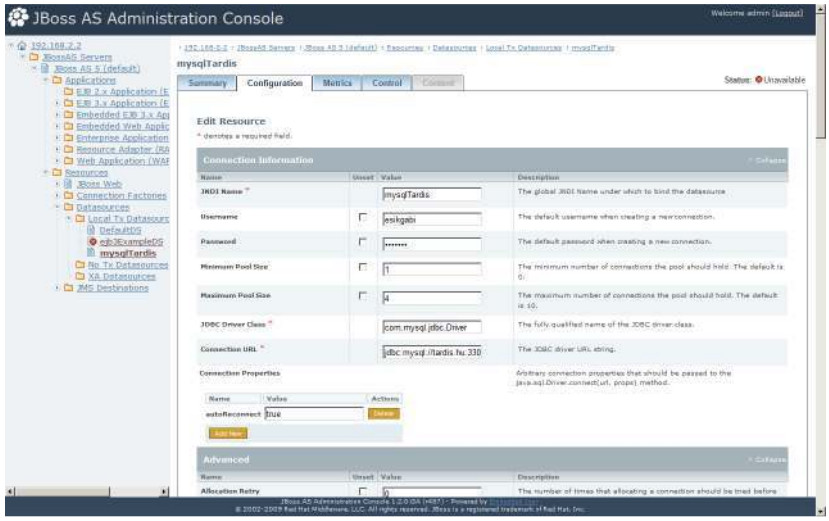
```
package ejb;
/**  Session Bean implementation class ReszlegService  */

@Stateless
public class ReszlegService implements ReszlegServiceLocal {
    // -- perzisztenica manager
	@PersistenceContext
	EntityManager em;

   // -- szükséges belső metódus
	private Alkalmazott getAlkalmazottById(String id) {
		return em.find(Alkalmazott.class,Integer.parseInt(id));
	}
  // -- hozzáadó üzleti metódus --
	@Override
	public void addReszleg(ReszlegVO reszleg) {
		Reszleg uj = new Reszleg();
		uj.setNev(reszleg.getNev());
		uj.setLeiras(reszleg.getLeiras());
		em.persist(uj);
	}
}
```

A fenti kódrészlet egy állapotmentes session bean segítségével megvalósított üzleti logikát szemléletet, az állapotmentességet a @Stateless annotáció jelzi az alkalmazás szervernek. A megvalósítandó metódusok listáját pedig az implementált interfész határozza

meg. Itt történik meg a web réteg felől érkező kérések megvalósítása adatbázis szinten, ha meg kell valamit jeleníteni a felületen, akkor itt felszedjük az adatbázisból, és vo-k formájában elküldjük a web rétegnek, ami majd meg fogja jeleníteni a felhasználónak az adatokat. Az osztálydefiníció után rögtön definiáljuk az adatbázist vezérlő egységünket, amelyet az EntityManager valósít meg. Itt is használunk egy @PersistenceContext annotációt, ami az adatbázis kontextust fogja definiálni az alkalmazás szervernek. A régebbi verziókban itt jó pár sort kellett begépelnünk mire megkaptunk egy ilyen EntityManager-t, de az annotációk bevezetése óta, már csak ez a két sor szükséges mind ehhez. Persze hogy ez így működjön meg kell szerkesztenünk pár konfigurációs xml fájlt. Először is a persistence.xml fájlban kell beállítanunk a tranzakció típusát, az adatforrást (data-source) és a használni kívánt adatbázis dialógust.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns=http://java.sun.com/xml/ns/persistence
xsi:schemaLocation=http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd version="1.0">
<persistence-unit name="Szakdolgozat" transaction-type="JTA">
  <jta-data-source>java:/mysqlTardis</jta-data-source>
    <properties>
     <property name="hibernate.dialect"
            value="org.hibernate.dialect.MySQL5InnoDBDialect"/>
     <property name="hibernate.hbm2ddl.auto" value="update"/>
    </properties>
</persistence-unit>
</persistence>
```

Az alkalmazás szerver adminisztrációs felületén pedig be kell állítanunk az adatbázisunk tényleges elérését, az adatbázis típusát, a használni kívánt jdb drivert emellett lehetőség van jó néhány speciális beállításra is.



12. ábra Alkalmazás szerver data-source beállítás

IV. Vezérlési és megjelenítési réteg

a. Új technológiák, új igények

Ahogy fejlődnek a webes technológiák úgy egyre szebb és gyorsabb webes alkalmazásokat lehet készíteni és ezek már inkább hasonlítanak egy asztali alkalmazáshoz, mint egy honlaphoz. A felhasználók találkoznak ilyen alkalmazásokkal és megtetszik nekik és elvárják, hogy ezentúl minél több ilyen legyen azok között, amit rendszeresen használ. Egy honlap esetén, ahogy navigálunk, az elemek között mindig egy újabb oldalra jutunk, elkerüljük a kérést a web szervernek, az pedig újra leküldi nekünk a teljes html oldalt, ha ez a folyamat elég gyorsan bekövetkezik, akkor csak annyit veszünk észre, hogy villan egyet képernyő és már ott is az új oldal. Ez a jelenség egy hírportálnál vagy egy iskolai honlapnál teljesen elfogadható. Azonban egy webes alkalmazás esetén, amit egy nagyvállalatnál a dolgozók nap mint nap használnak órákon keresztül ez már könnyedén hátránnyá válhat. A webes alkalmazások elterjedésének fő oka a rétegelt architektúrák bevezetésekor megjelenő vékony-kliens architektúra. Ez annyit jelent, hogy az üzleti logikát nagy szerverekkel végeztetik a cég egy külön helységében ahol ezek működési körülményei biztosítottak, és a felhasználó gépén csupán adatok jelennek meg, ott semmilyen számítás, üzleti művelet nem történik. Ennek egyik előnye, hogy a felhasználók gépeinek nem kell nagy erőforrásokkal rendelkezni, mindössze egy böngészőt legyen képes futtatni és azon keresztül ő mindent el tud végezni, épp ezért ez az eszköz lehet egy laptop, vagy egy PDA, de akár egy modernebb mobiltelefon.

13. ábra Vékony-kliens architektura

A vékony-klienses megjelenésével a vastag klienses megoldás kiszorult a piacról, viszont az igény megmaradt az iránt hogy a képernyőn a felhasználó egy ablakot lásson és azon belül a komponensek változzanak csupán. A kezdeti időkben ennek a megvalósítása nem volt egyszerű, hiszen a web egyet jelentett a html-el és ekkor még annak is örültek, ha egy html oldal tartalmát dinamikusan fel tudták tölteni, ezt php, jsp, servlet technológiákkal tették. Az a megoldás hogy egy html oldal tartalma a teljes dokumentum újratöltése – szerverről lekérés – nélkül megváltozzon még kezdetben szinte elképzelhetetlen volt. Szerencsére voltak, akik hamar rájöttek, hogy erre igen is nagy igény van és elkezdték megalkotni az ezt megvalósító technológiákat.

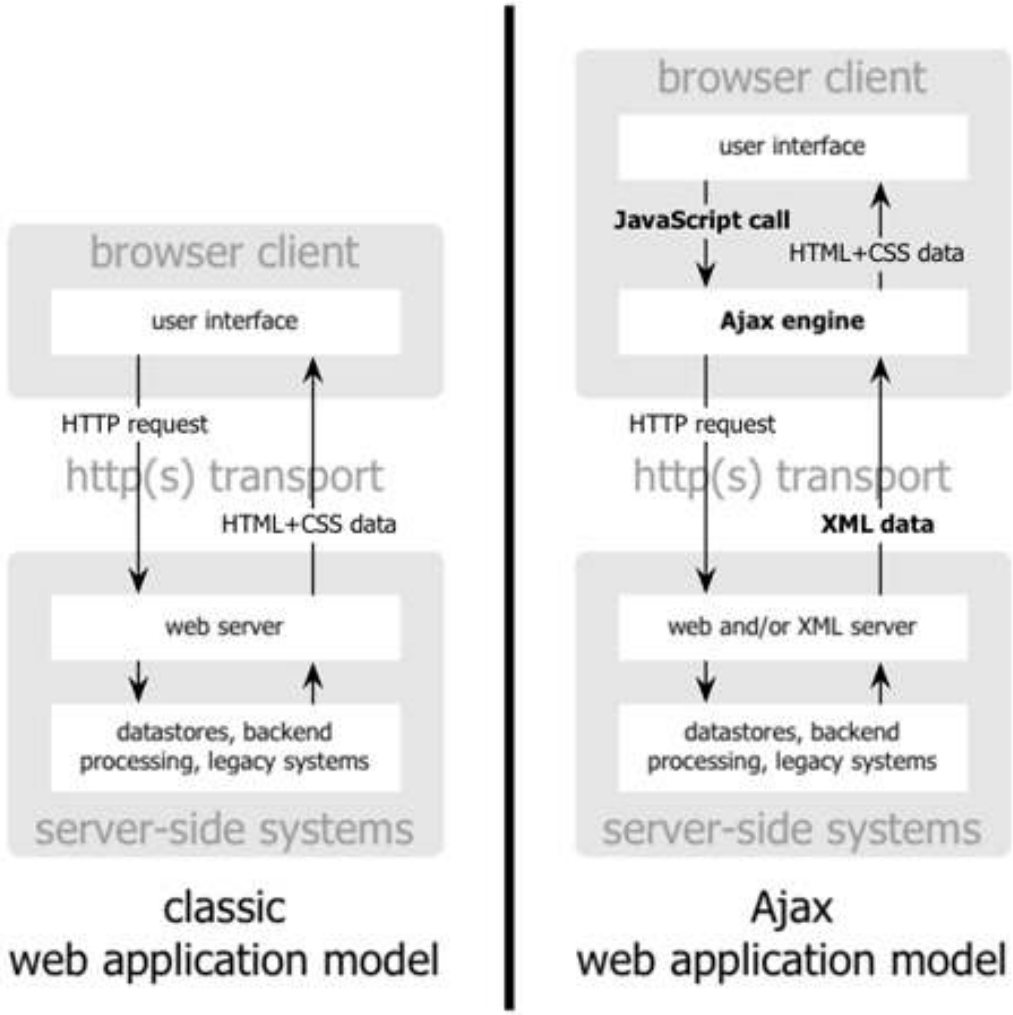
b. AJAX

Az AJAX (Asynchronus JavaScript and XML) interaktív webalkalmazások készítéshez létrehozott fejlesztési technika. A weboldal úgy cserél adatot a szerverrel, hogy az oldalt eközben nem kell újratölteni, ez nagyban növeli a honlap interaktivitását és sebességét, és ezáltal a felhasználó sokkal kellemesebbnek érzi a használatát. Az technológia a következő technikákat ötvözi:

- XHTML, HTML és CSS, ezek a tartalom leírására és formázásra szolgálnak
- DOM (Document Object Model) script nyelvekkel kezeli a dinamikus megjelenést és a már megjelenített adatokkal való együttműködést
- XMLHttpRequest adatok aszinkron kezelésére szolgál a kliens és a szerver között
- XML, XSLT az adatok továbbítása a kliens és a szerver között ilyen formátumban történik
- Ezek vezérlése és összekapcsolása JavaScript-el történik

A legfőbb előnye ennek az eszköznek a felhasználói élmény fokozása. Az Ajax-ot használó oldalak viselkedése már inkább az asztali alkalmazásokéhoz hasonlít mintsem a honlapokéhoz. Itt már lehetőség van arra, hogy csak az oldal bizonyos részei frissüljenek, amikor azoknak a tartalma valamiért megváltozik. Sokan vélik úgy, hogy az ajax segítéségével még nagyobb körben fognak elterjedni a webes alkalmazások olyan területeken is amelyeken eddig elsősorban az asztali alkalmazások domináltak.

Az Ajax gyorsaságának és dinamizmusának a titka abban rejlik, hogy a html oldalakkal szemben, itt nem utazik a hálózaton a teljes html kód. Ajax esetén a kliens és a szerver között csak a lényeges adatok mozognak, így a szerver válaszideje és a kliens megjelenítési ideje egyaránt csökken, mivel a kisebb mennyiségű adat hamarabb jut el egyik pontból a másikig. A szervertől kapott adatokból JavaScript segítségével fog HTML kód generálódni, ami jóval gyorsabb, mint ha egy nagy válaszidejű szerver generálná és küldené el nekünk. Abban az esetben fordulhat elő, hogy ez a megoldás lassabb mintha a szerverről töltenénk le, ha a kliens gépe nagyon kis teljesítményű és lassan képes generálni a tartalmat. Ebben a helyzetben viszont ez nem fogja befolyásolni a többi klienst, aki az oldalt, böngészi, míg egy lassú szerver estén, ha valami miatt belassul a szerver, akkor azt a lassulást bizony az összes kliens észlelni fogja.



14. ábra A klasszikus és az AJAX modell

Az Ajax elterjedésével a programozók egyre inkább rákényszerültek a JavaScript nagyobb arányú használatára. Ezzel egyre több programkód kerülhetett le kliens oldalra tehermentesítve ezzel a kiszolgálói oldalt, hiszen csak szükség esetén kell a szerverhez fordulni, mivel a kliens oldalon is tárolhatunk állapot információkat, melyeket eddig csak a szerver oldalról tudtunk lekérni. Ezen felül a jobb használhatóság többnyire jobb teljesítményt is jelent. Az Ajax segítségével készült alkalmazásoknak nem nagyon akad vetélytársuk, gyorsabban fejleszthetőek, közzétehetőek és használhatóak, mint más módszerrel készítettek, beleértve a régi asztali alkalmazásokat is. Ahogy egyre többen kezdtek fejleszteni Ajax alkalmazásokat nőni kezdett a verseny, és a programok bonyolultsága is.

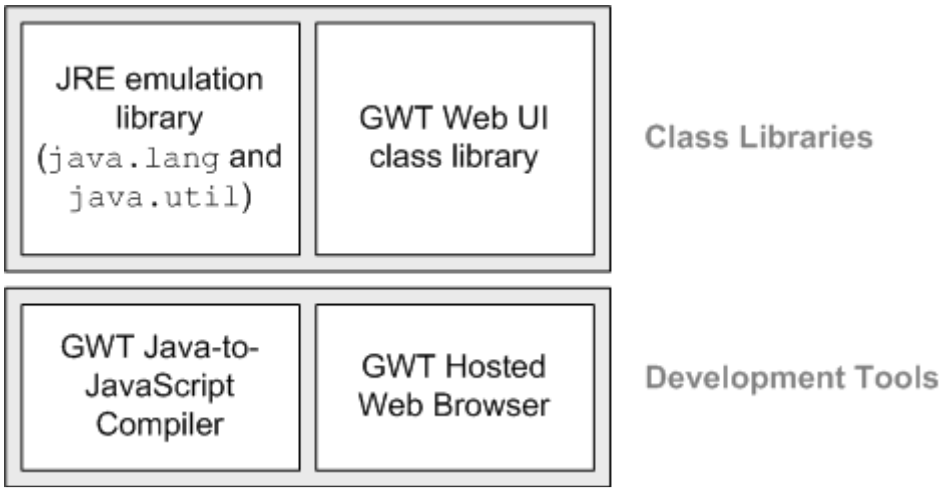
Az Ajax nem egy olyan technológia, amit önmagában nagy méretű alkalmazások fejlesztésére lehetne használni, nem álltak még kezdetben rendelkezésre olyan eszközök melyek segítették volna a nagy programok készítését JavaScripttel. A kezdeti keretrendszerek és programkönyvtárak némileg megkönnyítették a program készítést, de ettől függetlenül az Ajaxot még mindig olyasmire használták, amire a tervezői soha nem szánták. A probléma az volt a vastag kliensekhez hasonlóan megpróbálták az üzleti logikát itt is a kliens rétegben megvalósítani, de természetesen ennek ésszerű megvalósítására nem alkalmas az Ajax. Mindezek mellett a JavaScript hiányában van azon nyelvi eszközöknek, amelyek alkalmassá tennék a nyelvet az összetett programok készítésre, erre példa az objektumközpontú szerkezet, valamint a hibák fordítási időben történő elfogásának lehetősége. Az Ajax alkalmazások sikerét és korlátai felismerve néhány cég úgy döntött, hogy lépéseket tesz és alternatívát kínál az Ajax-szal szemben a gazdag webes élmény elérése érdekében. Az ilyen technológiákat, valamint az Ajax segítségével készített alkalmazásokat hívják gazdag internetes szolgáltatásoknak, RIA (Rich Internet Application). Alternatívák:

- Az Adobe Flash és a Flex. A Flash a legsikeresebb böngészőbővítmény napjainkban, a rendszerek 98%-án megtalálhatjuk. Sikere elsősorban az általa lejátszott SWF fájlok kis letöltési méretében rejlik. Nagyon szép dinamikus tartalmakat képes megjeleníteni.
- A Microsoft Silverlight. A Microsoft megelégelte a futásidejű Java-környezetet a böngészőjében, és eltávolította azt és helyére beépítette az ActiveX objektumok beágyazásának képességét. Mivel ez nem terjedte el kellőképp így létrehozta saját böngészőbővítményét, amely a .Net keretrendszerre épül.
- A Java FX. A Java Appletket készül kiváltani, és reagálás a Microsoft Silverlight-jára.

c. Google Web Toolkit (GWT)

i. Architektúra

A GWT a Google átlal készített AJAX fejlesztői eszköztár, az első verziót 2006. májusában adták ki. A Google felismerte az Ajax erősségét és úgy gondolta készít egy olyan toolkit-et melynek segítségével egyszerűen és hatékonyan fogunk tudni Ajax-os JavaScript kódokat készíteni, mindezt a már jól ismert Java nyelven keresztül. GWT-s alkalmazás készítésekor Java osztályokat írunk, és használunk fel, amikor elkészült a program azt a rendszer lefordítja JavaScript-é, úgy hogy közben optimalizálja is azt.



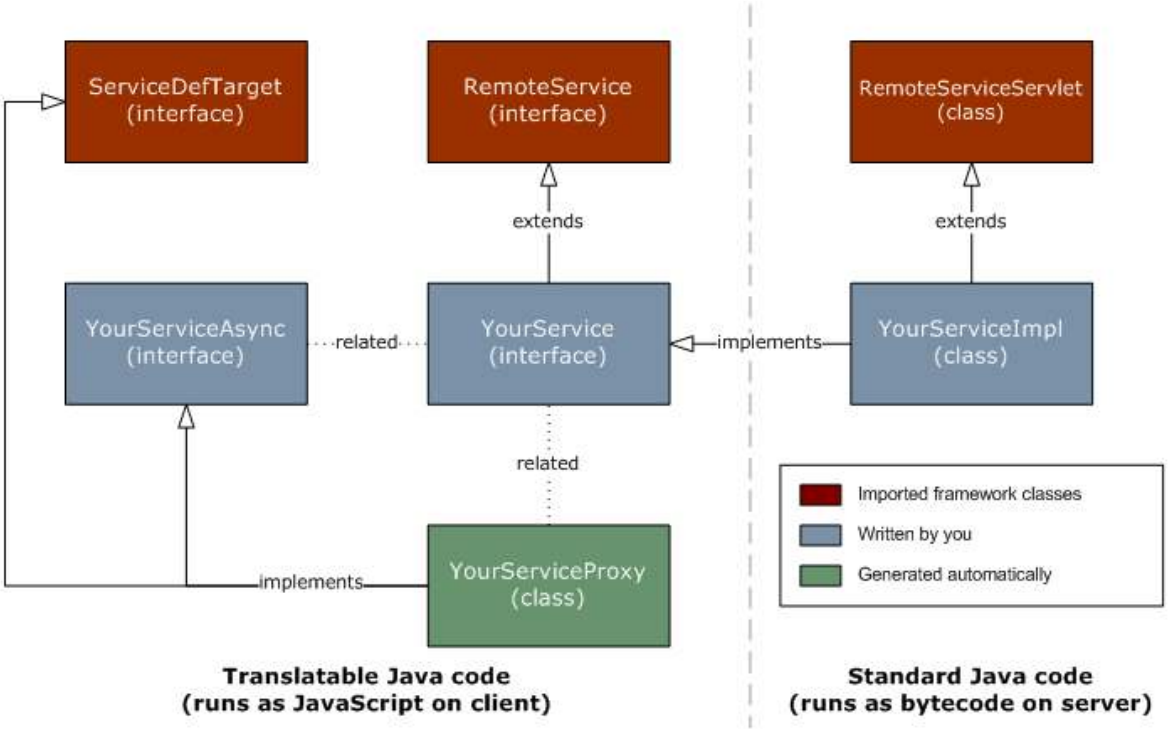
15. ábra Az architektúra részei

- Az emulált JRE könyvtár tartalmazza a legnépszerűbb java osztályok JavaScript implementációit. Az alkalmazás fejlesztésekor ügyelnünk kell arra, hogy kliens oldalon – ami JavaScriptté fog fordulni – csak olyan Java eszközöket használjunk, amelyek szerepelnek az emulált osztálykönyvtárban, különben fordítási hibát kapunk.
- A GWT Web UI osztálykönyvtár tartalmazza azon osztályokat és interfészeket melyek segítségével kialakíthatjuk a felhasználói felületünket. Ide tartoznak a beépített gombok, feliratok, szövegbeviteli elemek, táblázatok és még további elemek, amelyek előfordulhatnak a html világában. A GWT fordításkor ezen elemeket úgy fordítja le, hogy minden böngészőn ugyanúgy megjelenjen.
- A JavaScript fordító végzi el a Java osztályok JavaScriptté történő fordítását.
- A HostedBrowser lehetővé teszi, hogy a fejlesztés során kipróbálhassunk a kódunkat JavaScriptre fordítás nélkül, ilyenkor az alkalmazás Java-ként fut és a JVM interpretálja azt, ez nagyon hasznos hibakeresés során.

ii. Távoli eljáráshívás (GWT-RPC)

A GWT-ben rendelkezésünkre áll egy távoli eljáráshívás (RPC) könyvtár, melynek segítségével lehetőség nyílik arra, hogy a böngésző aszinkron módon kommunikáljon a kiszolgáló szerverrel. Az RPC könyvtár két részre osztható:

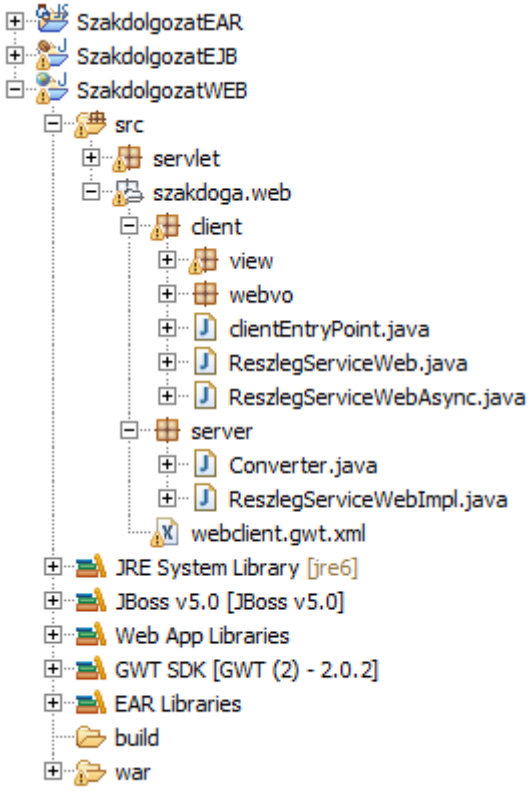
- a com.google.gwt.user.client.rpc csomag az ügyféloldali RPC támogatásért felelős
- a com.google.gwt.server.rpc csomag pedig a kiszolgáló oldali támogatásért



16. ábra GWT RPC komponensei

Az ábra jól szemlélteti az RPC megvalósításához szükséges komponenseket. Elsőre talán soknak tűnhet, de ha megnézzük, hogy a hét elem közül nekünk mindössze hármat kell elkészítenünk, a többit a rendszer elvégzi, akkor már jóval egyszerűbbnek tűnhet a dolog. A RemoteService felület egy olyan GWT-RPC felület, amely arra utasítja a GWT fordítóját, hogy e felülethez hozzon létre egy olyan kódot, ami képes lesz majd adatfolyammá alakításra. Lényegében ezen interfészen kell definiálnunk a szükséges metódusainkat, amit majd a szerver oldalon implementálni fogunk a RemoteServiceServlet–ből kiterjesztett osztályunkban. Az aszinkron interfész egyszerűen generálható a már megírt interfészünkből, ez annyival lesz más, hogy minden metódus void visszatérésű lesz, és kapnak még pluszba egy AsyncCallback paramétert, amiben majd a válasz üzenet fog visszajönni a szervertől.

A GWT megkövetel egy bizonyos csomagszerkezetet. A projektünkben szerepelnie kell egy client és egy server csomagnak, ezeken kívül és belül természetesen tetszőleges számú és nevű csomagot elhelyezhetünk, de ennek a kettőnek kötelező ilyen néven és itt elhelyezkednie. A client csomagnak tartalmazni kell egy osztályt, ami implementálja az EntryPoint interfészt, ez az osztály lesz az alkalmazás központja. Ide kerülhetnek a RemoteService osztályaink, amik mellé létrehozzuk az aszinkron (Async) változataikat is. A server csomagnak kell tartalmaznia a RemoteSercie-t implementáló RemoteServlet osztályokat, melyeket rendre Impl szóra végződnek.



17. ábra GWT csomag hierarchia

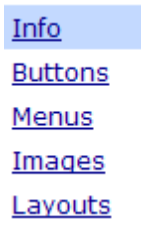
A client könyvtár gyökerében szerepel még egy gwt.xml konfigurációs fájl. Ennek az állománynak a segítségével tudjuk elvégezni a speciális GWT-s beállításokat. Itt adhatjuk meg a projektünkből kifordításra kerülő modulunk nevét, megadhatjuk az irányításért felelős EntryPoint-unkat, importálhatunk már elkészített komponenseket, megadhatjuk, hogy a fordító milyen böngészőkre, milyen nyelvekre készítse majd el a JavaScript kódot. Ezen fix könyvtárak és fájlok megkötését kivéve, teljes szabadságot élvezhetünk mind a könyvtárnevek és struktúrájuk valamint a fájlneveket illetően.

iii. Felületi elemek (GWT Widget Library)

A GWT biztosít számunkra jó néhány felületi eszközt, amelyeket használhatunk az alkalmazás felületének elkészítéshez. Lehetőségünk van természetesen saját elemeket is készítenünk, legegyszerűbb ha egy már meglévő elemből örökítjük a sajátunkat, így az már rendelkezni fog az ősének minden tulajdonságával, és nekünk már csak a saját igényeinket kielégítő kiegészítéseket kell implementálnunk.

A felhasználói felület elemei az alábbi csoportokba sorolhatóak:

- Statikus vezérlők

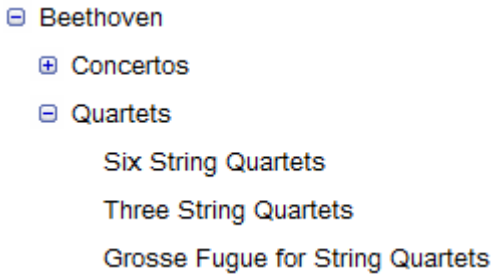


18. ábra Hiperhivatkozás

- Űrlap vezérlők

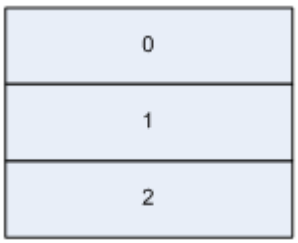
19. ábra Gombok

- Összetett vezérlők



20. ábra Fa struktúra

- Egyszerű/Összetett elrendezőpanelek



22. ábra Vízszintes panel elrendezés

- Egyszerű/Összetett tárolópanelek

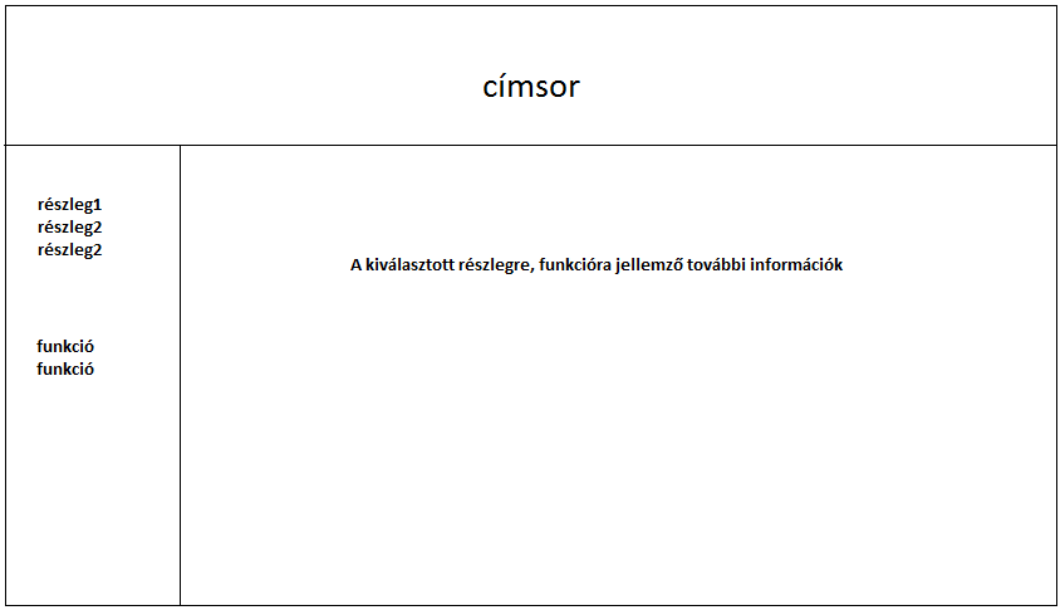
21. ábra Felugró ablak

- Esemény és szolgáltatásfelületek

d. Alkalmazásom megjelenítései rétege

i. Felület

Az alkalmazás felületének tervezésekor összegyűjtöttem az elkészített funkciókat, és csoportosítottam őket. Azt próbáltam szem előtt tartani, hogy a felület és a navigáció minél egyszerűbb legyen, ezáltal könnyű kezelhetőséget és egyszerű navigációt biztosítson a felhasználók számára. Úgy döntöttem, hogy a műveleteket elsősorban a részlegek szerint csoportosítom, ez annyit jelent, hogy az első feladata a felhasználónak kiválasztani azt a részleget, amellyel valamilyen műveleteket akar végezni. Ezt a kiválasztást egy bal oldali menüsorból végezheti majd el a felhasználó, ennek hatására a képernyő közepén fognak megjelenni a kiválasztott részlegre vonatkozó adatok.



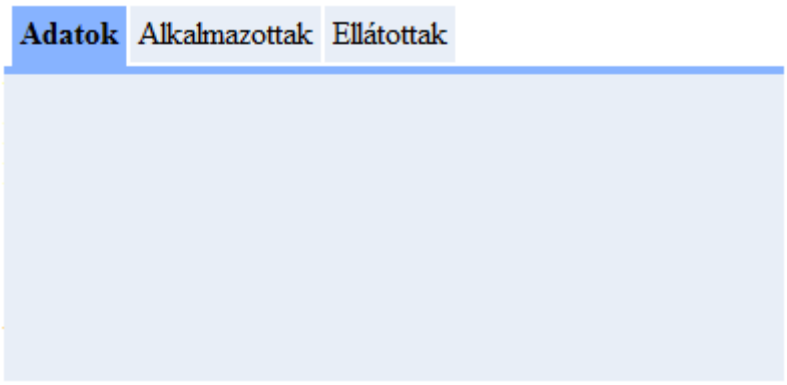
23. ábra Kezdeti terv

Ezen tervezet megvalósítható egyszerű felületi elemekkel, a címsor formázott szöveg, a bal oldali menüsor hiperhivatkozások a középső felirat szintén szimpla szöveg. A következő feladat a részlegek adatainak a rendezett megjelenítése, valamint ezen adatok szerkeszthetőségének megvalósítása.

Ismét elgondolkoztam azon, hogy mi alapján lehetne csoportosítani a megjelenő információkat, és ezen adat csoportokat külön egységekben megjeleníteni. Egy adott részleg információit alapvetően 3 csoportra lehet osztani:

- a részleg adati
- a részleg dolgozói
- a részleg ellátottai

A GWT TabPenel felületeleme tökéletesen megfelel számomra, ezzel az eszközzel egyszerűen és elegánsan el tudom szeparálni az adott részleg három adathalmazát.



24. ábra Részleg adatok TabPanel

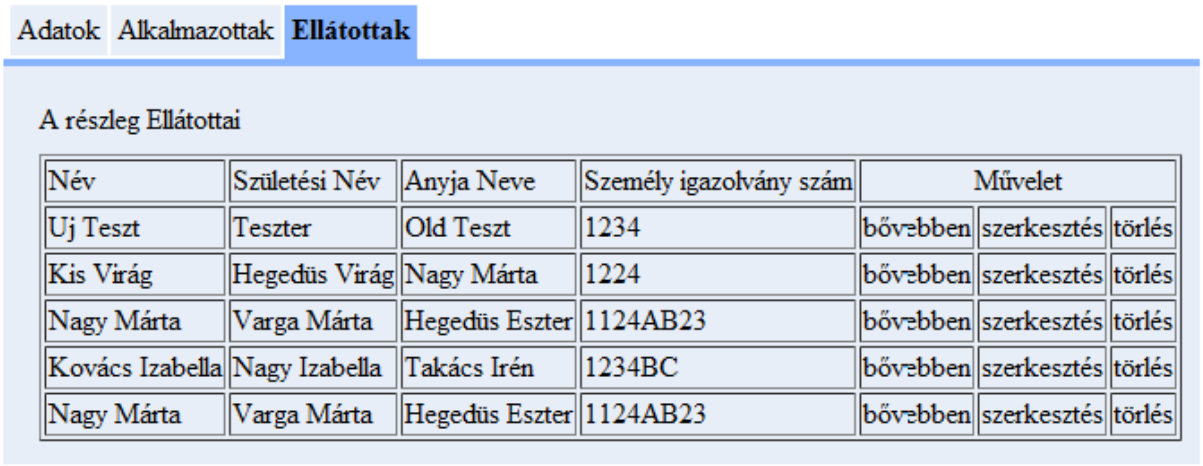
Az általam kiválasztott elrendezőpanel (TabPanel) több szempontból is megfelelő választásnak bizonyult. Egyértelműen megmutatja, hogy mely lehetőségek közül választhatunk és azt is jelzi, hogy aktuálisan mely adatoknál járunk, a navigálás is egyszerű és látványos. A vezérlés megvalósításának a terhét is leveszi a vállamról, mivel nem nekem kell figyelni a felhasználó interakcióit és azokat lekezelnem, ezt mind tudja magától, sőt még látványos is mivel az animáció bekapcsolásával a tabok közötti váltáskor a középső terület mérete dinamikusan változik a benne foglalt adatok méretétől függően. A felület ezen részének kódszintű megvalósítása:

```
final TabPanel tp = new TabPanel();
        tp.getDeckPanel().setAnimationEnabled(true);
        tp.setStyleName("gwt-TabPanel");

        tp.setWidth("90%");
        tp.setHeight("90%");

        tp.clear();
        tp.add(new ReszlegAdatView(historyToken),"Adatok");
        tp.add(new ReszlegAlkalmazottView(historyToken),"Alkalmazottak");
        tp.add(new ReszlegEllatottView(historyToken),"Ellátottak");
        tp.selectTab(0);
```

Ebből a kódrészletből is látszik, hogy egy ilyen komplex felületi elem létrehozása és beállítása milyen egyszerű feladat GWT segítségével. Létrehozunk TabPanel példányt tp néven, ezt követően beállítjuk rá az animációt, a stílusát valamint a méreteit, ezek után letöröljük a felületét, majd létrehozzuk a szükséges Tab-okat – Adatok, Alkalmazottak, Ellátottak -, majd végül beállítjuk, hogy a megjelenéskor a kiválasztott panel az első legyen, mivel az indexelés 0-tól kezdődik, azért szerepel a forrásban 0.

A következő megoldandó feladat az adatbázisból kinyert adatok megjelenítése. Erre a legalkalmasabb elrendezési forma a táblázatos megjelenítés, minden egyes személy – legyen akár alkalmazott vagy ellátott – egy táblázat egy sorában fog megjelenni. A problémám az, hogy nem lehet egységes megjelenítést létrehozni, mert a különböző részlegen lévő emberek más adatokkal rendelkezhetnek, valamint a rögzített adatok nagyságától függően a táblázat mérete ingadozhat, és ez a megjelenítésre lenne negatív hatással.

Et a problémát úgy oldottam meg, hogy készítettem egy egyszerűsített táblázatot, amely minimális adatokat tartalmaz az adott személyről, de minimum annyit hogy biztosan meg lehessen különböztetni egymástól két egyént, ez a táblázat alkalmazható minden egyes egyedre, mivel cellák tartalma a személyes adatok, mint név, születési név, anyja neve, személyi igazolvány száma. Természetesen lehetőséget kell biztosítani arra, hogy mindenkiről meg lehessen tekinteni az összes adatát, valamint azokat módosítani is lehessen, ezért a táblázat utolsó oszlopaiban elhelyezek egy-egy műveletet, amire rákattintva az adott művelet megvalósítható.



25. ábra Egyszerűsített táblázat

Az ábrán látható táblázat előállításáért az EllatottTablaView osztály felelős, itt történik meg az adott részleg ellátottainak lekérése, majd miután az adatok visszaérkeztek a szervertől, létrejön a táblázat beállítódnak a tulajdonsága aztán a táblázat sorai feltöltődnek értékekkel. Ezen feladatok elvégzése inkább mondható egyszerűnek, mint azok, amelyeket ezek után fogok megvalósítani. Figyelnem kellett azt, hogy a táblázat művelet oszlopai közül melyikre kattint a felhasználó és ezt melyik sorban teszi, a további események ugyanis ezek függvényében fognak bekövetkezni.

A táblázat megjelenítéséért és sorainak adattal való feltöltéséért felelős kódrészlet:

```
final FlexTable ft = new FlexTable();
	ft.setBorderWidth(1);
	ft.setWidget(0, 0, new HTML("Név"));
	ft.setWidget(0, 1, new HTML("Születési Név"));
	ft.setWidget(0, 2, new HTML("Anyja Neve"));
	ft.setWidget(0, 3, new HTML("Személy igazolvány szám"));
	cellFormatter.setColSpan(0, 4, 3);
	cellFormatter.setHorizontalAlignment(0, 4,HasHorizontalAlignment.ALIGN_CENTER);
	ft.setWidget(0, 4, new HTML("Művelet"));

	for(EllatottWebVO e: ellatottak){
		ft.setText(ft.getRowCount(), 0, e.getNev());
		ft.setText(ft.getRowCount()-1, 1, e.getSzuletesiNev());
		ft.setText(ft.getRowCount()-1, 2, e.getAnyjaNeve());
		ft.setText(ft.getRowCount()-1, 3, e.getSzemelyiIgazolvanySzam());
		ft.setText(ft.getRowCount()-1, 4, "bővebben");
		ft.setText(ft.getRowCount()-1, 5, "szerkesztés");
		ft.setText(ft.getRowCount()-1, 6, "törlés");
	}
```

A fenti kódrészlet egy dinamikus táblázat (FlexTable) példányosításával indul, ezek után beállítjuk a keretét, majd az első sor celláit felcímkézem, hogy jelezze, hogy az adott oszlopban milyen értékek szerepelnek. Ezek után további formázásra kerül sor, cella összevonás, majd egy for ciklus segítségével végigmegyek az ellátottak adatait tartalmazó kollekción és annak az értékeivel feltöltöm a táblázat sorait. Megfigyelhetjük, hogy ezt a felületi elemet is könnyedén sikerült megvalósítanunk GWT-s eszközök segítségével.

Következő feladatom a műveletek figyelés és megvalósítása. Szerencsémre a GWT az események kezelésére nagyon jó eszközöket kínál. Azon felületi elemekhez, melyekhez esemény köthető rendelhetünk úgynevezett esemény figyelőket (ClickHandler). Ezzel az eszközzel az egész táblázatomon tudom figyelni a kattintásokat. Az eseményfigyelőhöz létrehozhatok egy Cell objektumot, melyet a táblázatra történő klikkelés fog definiálni. Ennek a segítségével azt is meg tudom mondani, hogy melyik cellára klikkelt a felhasználó, ebből pedig kinyerhető hogy melyik műveletet szeretné végrehajtani.

```
ft.addClickHandler(new ClickHandler() {
@Override
public void onClick(ClickEvent arg0) {
	final Cell cell = ft.getCellForEvent(arg0);
	//--bővített nézet
	if(cell.getRowIndex()>0 && cell.getCellIndex()==4){ ……  }
	//--módosítás
	if(cell.getRowIndex()>0 && cell.getCellIndex()==5){ ……  }
	//--törlés
	if(cell.getRowIndex()>0 && cell.getCellIndex()==6){  ……  }
 }
});
```

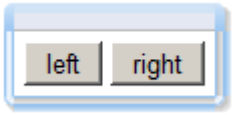
A soron következő feladat az, hogy valamilyen formában megjelenítsem a különböző nézeteket (bővített, módosító, hozzáadó, törlő). A vezérlés egyszerűségét szem előtt tartva, ezeket a nézeteket egy felugró (PopUp) ablakban jelenítettem meg. Nem szerettem volna minden egyes nézethez külön felugró ablakot készíteni, ezért létrehoztam egy általánosat, amit igény szerint be lehet állítani a tartalmához megfelelően. Erre a feladatra tökéletesen megfelelt számomra a GWT által biztosított DialogBox felületelem, így a saját felugró ablakomat ebből az osztályból származtattam, ezáltal minden szükséges tulajdonságot örökölt az ősétől, valamint én még kiegészítettem a számomra szükséges elemekkel.

```
package szakdoga.web.client.view;

import com.google.gwt.user.client.ui.*;

public class MyPopupPanel extends DialogBox{

	private VerticalPanel dialogPanel = new VerticalPanel();
	private HorizontalPanel buttonPanel = new HorizontalPanel();

	private Button leftButton = new Button("left");
	private Button rightButton = new Button("right");

    public MyPopupPanel(){
	buttonPanel.setWidth("100%");
	buttonPanel.add(leftButton);
	buttonPanel.add(rightButton);

	buttonPanel.setCellHorizontalAlignment(leftButton, HasHorizontalAlignment.ALIGN_LEFT);
	buttonPanel.setCellHorizontalAlignment(rightButton, HasHorizontalAlignment.ALIGN_RIGHT);

	dialogPanel.add(buttonPanel);
	this.add(dialogPanel);

	this.setGlassEnabled(true);
	this.setAnimationEnabled(true);
   }

   //--- az adattagok getter és setter metódusai
}
```

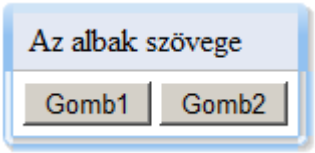
Ezzel az általánosan megírt felugró ablaknak a segítségével az összes nézetemet képes vagyok megjeleníteni, mindössze annyit kell tennem, hogy elkérem az objektumnak a getDialogPanel() metódusával a dialogPanel-t majd erre ráteszem a megjelenítendő nézetet. A felületen található még két gomb, amelyeket szintén getter metódusok segítségével elkérve beállítható a rajtuk megjelenő szöveg, valamint a megnyomásuk hatására bekövetkező események is, sőt ha esetleg valamelyik gombra nem lenne szükség, akkor egyszerűen meghívom a .hide() metódusát és az adott gomb rejtett lesz.

A fenti kódrészlet szemléltetéséhez bemutatnám a felületi elemet működés közben is, kezdve a példányosítástól az egyéb felületi elemek felviteléig. Az első lépés az, hogy példányosítok egy MyPopupPanel objektumot, amit ezek után tetszés szerint konfigurálhatok.

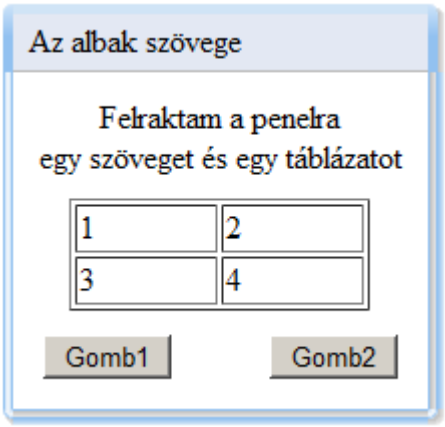
```
MyPopupPanel tmp = new MyPopupPanel();
 tmp.show();
```




```
 MyPopupPanel tmp = new MyPopupPanel();
            tmp.setText("Az albak szövege");
            tmp.getLeftButton().setText("Gomb1");
            tmp.getRightButton().setText("Gomb2");
            tmp.show();
```




```
Grid g = new Grid(2,2);
            g.setText(0, 0, "1");
            g.setText(0, 1, "2");
            g.setText(1, 0, "3");
            g.setText(1, 1, "4");
            g.setWidth("150");
            g.setBorderWidth(1);

HTML szoveg = new HTML("Felraktam a penelra<br>
                        egy szöveget és egy táblázatot"),

 MyPopupPanel tmp = new MyPopupPanel();
            tmp.setText("Az albak szövege");
            tmp.getLeftButton().setText("Gomb1");
            tmp.getRightButton().setText("Gomb2");
            tmp.getDialogPanel().insert(szoveg, 0);
            tmp.getDialogPanel().insert(g, 1);
            tmp.getDialogPanel().setSpacing(10);

            tmp.show();
```



A fenti példák jól illusztrálja mennyivel egyszerűbb egy ilyen általános és teljesen testre szabható eszközzel dolgozni, mint minden egyes célra egy új rendszer által nyújtott DialogPanelel. Ezen elemnek köszönhetően jóval lerövidült a fejlesztési idő, és az esetleges felületi módosításokat elég volt egyetlen egy helyen módosítani, és az összes elemen végrehajtódtak a változások, ez hatalmas segítséget jelent a kinézet tervezése során. Ezek után már csak annyi teendőm volt, hogy létrehoztam a különféle nézeteket, melyek majd részletesen megjelenítik az adott egyedekről a szükséges információkat, valamint a felületen elhelyezett gombokhoz hozzárendelni a megfelelő funkcionalitást elvégző szerverhívásokat.

ii. Kapcsolódás az üzleti logikához

A felület funkcionalitását a GWT által kínált távoli eljáráshívásokkal valósítottam meg, amelyek az üzleti réteg EJB szolgáltatásait fogják igénybe venni. Első lépésként létrehoztam a saját távoli szolgáltatást leíró interfészemet, melyet ReszlegServiceWeb-nek neveztem el és természetesen a com.google.gwt.user.client.rpc.RemoteService osztály leszármazottja, és ezen interfészen felsoroltam milyen szolgáltatásokra lesz szükségem a felhasználói felületen.

```
@RemoteServiceRelativePath("reszleg")
public interface ReszlegServiceWeb extends RemoteService {
	ArrayList<String> getReszlegNames();
	ReszlegWebVO getReszlegByName(String name);
	….
	void setReszlegVezeto(ReszlegWebVO reszleg, AlkalmazottWebVO alkalmazott);
	void updateReszleg(ReszlegWebVO reszleg);
	ArrayList<SpecialisAdatWebVO> getAllSpecAdat();
	void addNewEllatott(EllatottWebVO uj);
	void delEllatott(String id);
}
```

Ezt követően a fejlesztői eszközöm (eclipse) felismerte, hogy ez egy távoli szolgáltatást leíró interfész, és felajánlotta, hogy elkészíti ennek az aszinkron változatát, amely abban fog különbözni, hogy a metódusok mind void visszatérésűek, valamint a paraméterezésbe még be fog kerülni egy adattag.

```
public interface ReszlegServiceWebAsync {

	void getReszlegNames(AsyncCallback<ArrayList<String>> callback);

	void getReszlegByName(String name, AsyncCallback<ReszlegWebVO> callback);
	 ……..
	void updateReszleg(ReszlegWebVO reszleg, AsyncCallback<Void> callback);

	void getAllSpecAdat(AsyncCallback<ArrayList<SpecialisAdatWebVO>> callback);
	void delEllatott(String id, AsyncCallback<Void> callback);
}
```

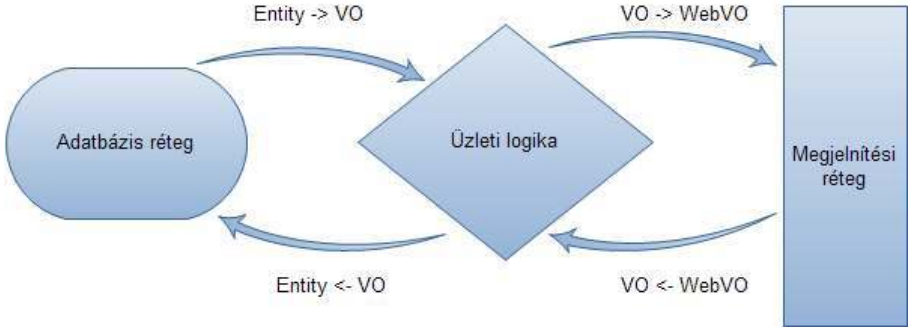
Rendre az interfész neve utal arra az interfészre, amiből generálódott, annyi különbséggel hogy Async végződést kap, így ránézésre látszik, hogy ez a ReszlegServiceWeb interfésznek az aszinkron változata. A paraméterlistákba bekerülő AsyncCallback típusú adattag felügyeli, hogy mikor érkezett vissza a kérdésre a válasz a szerver oldalról, és hogy sikeresen megtörtént az eljárás hívás vagy esetleg valamilyen hiba történt, és ennek megfelelően kell ezt a két lehetőséget külön megvalósítanunk.

Az elvárt szolgáltatások definiálása ezen interfészek elkészítésével meg is történt, most már csak meg kell őket valósítani. Ezeket a metódusokat a ReszlegServiceWebImpl osztályom valósítja, meg ami a com.google.gwt.user.server.rpc.RemoteServiceServlet osztály leszármazottja és implementálja a ReszlegServiceWeb interfészt.

```
public class ReszlegServiceWebImpl extends RemoteServiceServlet implements  ReszlegServiceWeb {

	@EJB ReszlegServiceLocal rsl;
	@EJB UgyfelServiceLocal usl;

	@Override
	public ArrayList<AlkalmazottWebVO> getReszlegAlkalmazott(String id) {
		ArrayList<AlkalmazottWebVO> alkalmazottak = new ArrayList<AlkalmazottWebVO>();

		for(AlkalmazottVO avo: rsl.getAlkalmazottakByReszlegId(id))
			alkalmazottak.add(Converter.vo2web(avo));
	 return alkalmazottak;
	}
	…..
	@Override
	 public void delEllatott(String id) {
		usl.delEllatott(id);
	 }
}
```

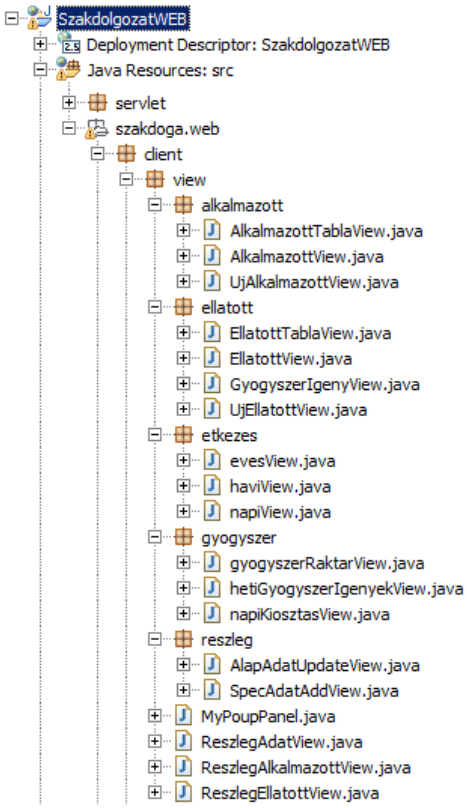
Az EJB3-nak köszönhetően az EJB-s szolgáltatásaink használhatóvá tételéhez csupán egy-egy sorra van szükségünk, a megfelelő @EJB annotáció után megadjuk a szolgáltatásokat definiáló interfészeket majd egy lokális változónevet írunk utánuk. Ezek után a változónkon keresztül már igénybe vehetjük az üzleti réteg által nyújtott valamennyi publikus szolgáltatást.

A forráskódban feltűnik az üzleti réteg és az adatbázis réteg közötti szállító objektumokon kívül (VO) egy újabb szállító objektum, ez pedig a WebVO. Ez az objektum hasonló szerepet lát el, mint az előző, annyi különbséggel hogy ez a megjelenítés kliens oldali részéről szállítja az adatokat a web réteg szerver oldalához és vissza. Mivel a GWT kliens oldali részre JavaScript-é fordul le, és a JRE csak egy része van implementálva JavaSriptre, így a web kliens oldali részén csak meghatározott típusú attribútummal rendelkező osztályok szerepelhetnek. Ezért is van szükség külön még a WebVO osztályokra, ezek konvertálást szintén egy Converter osztály végzi itt a web projekt server csomagjában.



iii. Az adatok megjelenítése a felületen

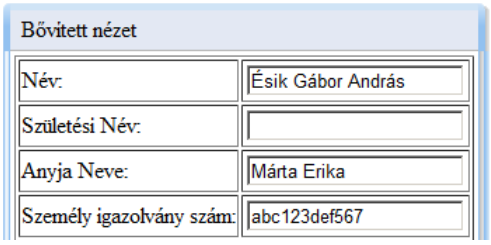
Az előzőekben bemutattam az alkalmazásom felületi felépítését, valamint az üzleti logikával kiépített kapcsolatát. A felületi elemek és a szolgáltatást nyújtó metódusok megvannak, az elvégzendő feladat az szervertől érkező információk megfelelő megjelenítése a felületen. A GWT ajánlása az, hogy foglaljuk a nagyobb megjelenítési egységeket úgynevezett nézet (view) osztályokba. Létrehoztam egy külön view csomagot így az alkalmazásnak ez egy külön egysége, amely csak a megjelenítéssel kapcsolatos állományokat tartalmazza. Ezen a csomagon belül szintén létrehoztam három alcsomagot, melyek a különböző egyedek megjelenítéséért felelős osztályokat tartalmazzák.



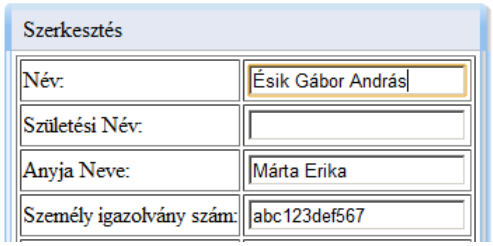
27. ábra View csomag szerkezet

A csomagszerkezetből jól megfigyelhető, hogy külön osztály jeleníteni meg számunkra az alkalmazottak és ellátottak táblázatos felsorolását, és mindkét entitáshoz van egy saját egyed megjelenítő nézet. A részleg csomagban kerültek a részleg entitás megjelenítéséhez szükséges view osztályok. A csomag gyökerében szerepel még négy fájl, ezek közül három megjelenítő szerepet lát el, méghozzá úgy hogy összefoglalja a nevében szereplő entitások speciális nézeteit, a kimaradt fájl a saját felugró ablakomat leíró osztály, melynek a bemutatása fentebb már megtörtént csak azért került ebbe a csomagstruktúrába, mivel ez is szorosan kötődik a megjelenítéshez.

A TableView nevű osztályok az adatok táblázatos megjelenítésért felelnek, ezek bemutatása már a felületi résznél megtörtént. Itt az egyedek megjelenítését megvalósító nézetek bemutatására fordítanék nagyobb figyelmet. Ezen osztályok elsődleges feladata, hogy az üzleti réteg segítségével adatbázisból felszedett objektum példányok szükséges adattagjait megjelenítse a felhasználó számára. Itt fontos megjegyeznem, hogy az alkalmazás megkívánja a megjelenítést és a módosítás lehetőségét is, e két felület szinte teljesen megegyezik, mindössze az információk szerkeszthetősége a különbség közöttük. A rendelkezésemre álló eszközök segítségével, könnyű szerrel meg tudtam ezt úgy valósítani, hogy ugyanazon állomány végezze el a megjelenítő és a szerkesztő nézet elkészítését. Az adattagok értékei TextBox elemekbe kerültek bele, és a felületi elem rendelkezik egy olyan metódussal, amelynek segítségével beállíthatjuk, hogy az adott elemet lehet e szerkeszteni, vagy csak olvasható.



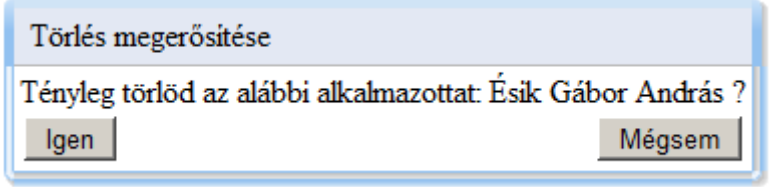
29. ábra Csak olvasható nézet



28. ábra Szerkeszthető nézet

A két felületi elem majdnem teljesen megegyezik, ahogyan ezt a képeken is látjuk. A képek készítésekor az egérrel ugyanazon a helyen álltam, az olvasható felületen nem történt semmilyen esemény, míg az írható felületen megjelent a kurzor és a texboxt kerete is elszíneződött, ezek az események jelzik a felhasználónak, hogy itt lehetősége van az adatok megváltoztatására.

A törlés feliratra kattintva megjelenik egy felugró ablak, amely megerősítést kér a felhasználótól, hogy valóban törölni kívánja e az adott elemet. Ezt szintén a saját PopupPanel eszközöm segítségével valósítottam meg, melynek értékül adtam a törölni kívánt elem nevét, hogy a felhasználó megfelelő tájékoztatást kapjon arról, hogy mit is szeretne törölni.



30. ábra Törlés megerősítése

A megjelenítési elemek kinézetének bemutatása után, bemutatom, hogy miként töltök fel információval egy-egy ilyen felületet.

```
public class AlkalmazottView extends Composite {

	private boolean csakOlvashato;
	private VerticalPanel vp = new VerticalPanel();
	private AlkalmazottWebVO alk = new AlkalmazottWebVO();
	private FlexTable ft = new FlexTable();

public AlkalmazottView(){	initWidget(vp);	vp.add(ft);	}

	public void init(){
		TextBox nev = new TextBox();
		nev.setReadOnly(csakOlvashato);
		nev.setText(alk.getNev());

		TextBox szulNev = new TextBox();
		szulNev.setReadOnly(csakOlvashato);
		szulNev.setText(alk.getSzuletesiNev());

		ft.setWidget(0, 0, new HTML("Név: "));		ft.setWidget(0, 1, nev);
		ft.setWidget(1, 0, new HTML("Születési Név:"));	ft.setWidget(1, 1, szulNev);
	}

	public void setCsakOlvashato(boolean csakOlvashato) { this.csakOlvashato = csakOlvashato;  }

	public void setAlk(AlkalmazottWebVO alk) {this.alk = alk;}
}
```

A fenti kódrészlet egy alkalmazott adatait képes megjeleníteni. Példányosításkor inicializálódik maga a felületi elem, a megírt setter metódussal beállítom a megjelenítendő AlkalmazottWebWO-t ami az adatokat tartalmazza, majd beállítom a csakOlvasható változót ami meghatározza hogy a létrehozott objektum szerkeszthető lesz, avagy csak olvasható. A beállítások után az objektum már rendelkezik minden szükséges információval, ezek után elvégezhetjük a megjelenítést az init() metódus segítségével, ami a megjelenített textboxok feltöltődnek értékkel.

```
mpp.setText("Bővített nézet");
  mpp.getLeftButton().setVisible(false);
	mpp.getRightButton().setText("Bezár");

	AlkalmazottView alkView = new AlkalmazottView();
	alkView.setCsakOlvashato(true);
	alkView.setAlk(alkalmazottak.get(cell.getRowIndex()-1));
	alkView.init();

   mpp.getDialogPanel().insert(alkView, 0);
mpp.center();  mpp.show();
```

A nézetek által megjelenített adatokat a távoli metódushívások szolgáltatják számunkra. Ezen a hívások aszinkron módon kommunikálnak a szerverrel, így előfordulhat az is hogy nem érkezik meg a válasz közvetlenül a gomb lenyomását követően, mi a gomb segítségével csak elküldjük a megfelelő kérést és amint van rá válasz az visszajön hozzánk. Ennek a típusú kommunikációnak ott van számottevő szerep ahol nagy számítási igényű üzleti folyamatok szükségesek, és egy ilyet a felhasználó elindíthat és nem szükséges megvárnia a folyamat végét, hanem dolgozhat tovább, és ha elkészült a válasz, akkor dolgozhat tovább azzal. Természetesen, ha nem jól kezeljük, ez lehet hátrány is ezért ügyelni kell arra, hogy megfelelő környezetben használjuk, vagy megfelelően alakítsuk ki hozzá a környezetet. Például ha egy adott információ nélkül a felhasználó nem mehet tovább, akkor tegyünk ki a képernyőre egy animációt és írjuk ki, hogy adatok lekérése folyamatban, ezzel tájékoztatjuk a klienst, hogy várja meg az adatokat és addig semmi mást ne csináljon.

```
public class ReszlegEllatottView extends Composite{

    private final ReszlegServiceWebAsync reszlegServiceWeb = GWT.create(ReszlegServiceWeb.class);

    public ReszlegEllatottView(String reszlegNev){
      initWidget(vp);
      reszlegServiceWeb.getReszlegEllatott(arg0.getId(), new AsyncCallback<ArrayList<EllatottWebVO>>() {
                    @Override
                    public void onFailure(Throwable arg0) {
                    Window.alert(arg0.getLocalizedMessage());
                    }

                    @Override
                    public void onSuccess(ArrayList<EllatottWebVO> arg0) {
                    vp.add(new EllatottTablaView(arg0,reszleg.getId().toString(),reszleg.getNev()));
                    vp.setWidth("100%");
                    vp.setSpacing(15);
                    }
          });
      }
}
```

A fenti kódrészleg bemutatja egy távoli eljáráshívás meghívását. Az osztály definiálásakor létrehoztam egy reszlegServiceWeb adattagot ami tartalmazza a ReszlegServiceWebAsync interfész által nyújtott szolgáltatásokat. A példányosítás a GWT.create() metódussal végezhető el. Ezek után meghívtam a getReszlegEllatott() metódust, melynek megadtam annak a részlegnek az id-ját amelyiknek kíváncsi voltam az ellátottaira, valamint kapott még egy AsyncCallback<ArrayList<EllatottWebVO>> paramétert, amely majd aszinkron módon adja vissza nekünk az ellátottak listáját.

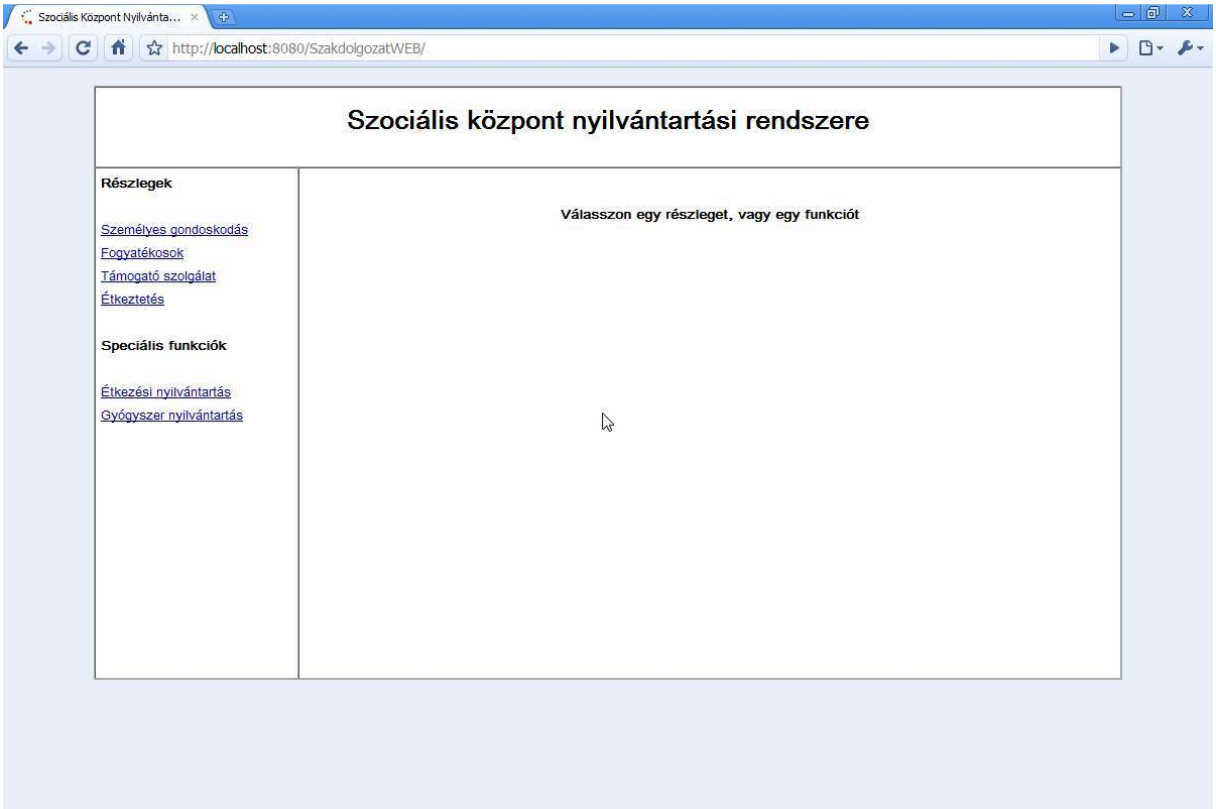
Ezek után meg kell valósítanunk két metódust – onFailure, onSuccess – ezek segítségével leírhatjuk, hogy milyen események történjenek, ha az eljáráshívás valamilyen hibával zárult, vagy sikeresen végrehajtódott. A szemléltetett kódrészlet esetén hiba bekövetkeztekor egy felugró ablakban megjelenítem a keletkezett hiba szövegét, fejlesztés során számomra ez elegendő információ, hogy megállapítsam a hiba helyét esetleg okát, később természetesen ide kerülhet a felhasználó számára is értelmezhető hibaüzenet. Sikeres végrehajtás esetén pedig az arg0-ba kaptam meg a felhasználók listáját, amit átadtam egy EllatottTablaView() konstruktorának mely a lista adatit táblázatos formában megjelenítette számomra.

Ezzel a példával végére értem a megjelenítési réteg elemeinek bemutatásának, kezdtem a felületi elrendezéssel, majd a felületen használt megjelenítő elemekkel aztán a saját készítésű elemekkel folytattam, ezt követően az adatok megjelentésének a különféle felületi eszközeit szemléltettem majd az üzleti logikával való kapcsolat kiépítése után bemutattam, hogy miként is jutnak ezek az elemek a megjeleníteni kívánt adatokhoz.

V. A rendszer felhasználói kézikönyve

Ebben a fejezetben szeretném bemutatni az elkészült alkalmazás által nyújtott szolgáltatások kezelését, az egyes funkciók elérését és működtetését. Ezen leírásnak köszönhetően egy átlagos felhasználó is képes lesz az alkalmazás használatára.
Az alkalmazás böngészőn keresztül érhető el, melynek köszönhetően nem szükséges a felhasználók számára erős konfigurációjuk számítógép, a böngésző futtatásához kevesebb erőforrás szükséges, mint egy komolyabb asztali alkalmazáshoz.

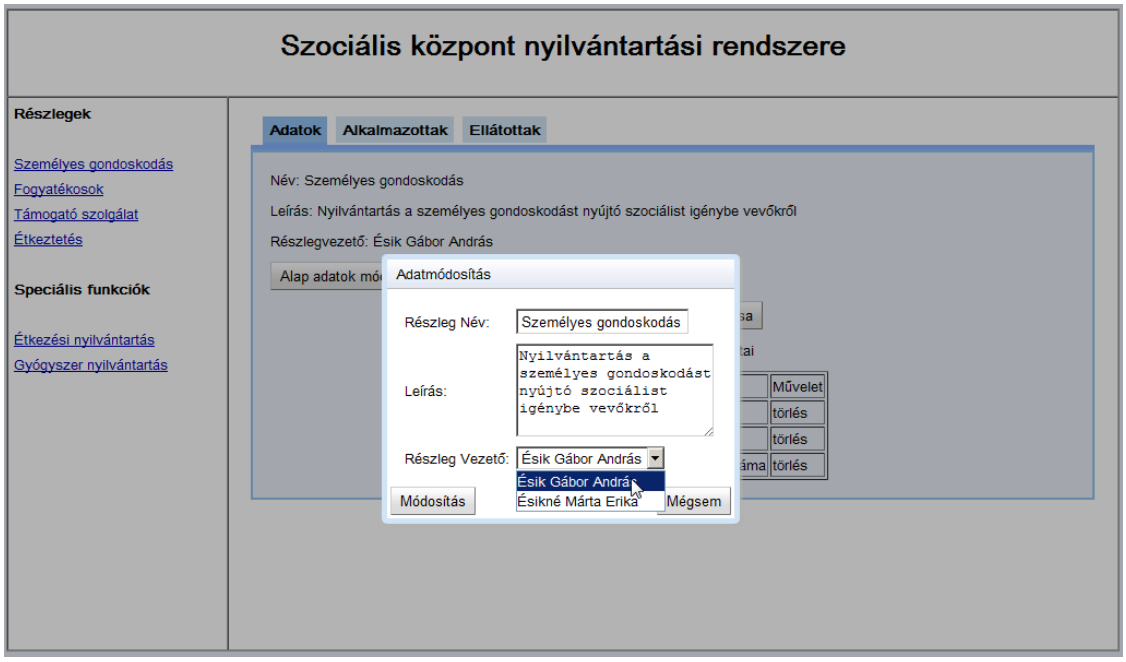
A rendszert elindítva, megjelenik a böngésző képernyőjében a három részre osztott felület, ahol a bal oldalon felsorolt részlegek közül kiválaszthatjuk ki azt, amelynek az adatit meg akarjuk nézni vagy változtatni, valamint használhatjuk a rendszer egyéb nyilvántartási funkcióit is.



A továbbiakban, a képernyőképekben már csak közvetlenül az alkalmazást fogom szerepeltetni, a böngészőt nem, mivel elég sok részt elfoglal, és miatta kevésbé láthatok az alkalmazás felületi elemei.

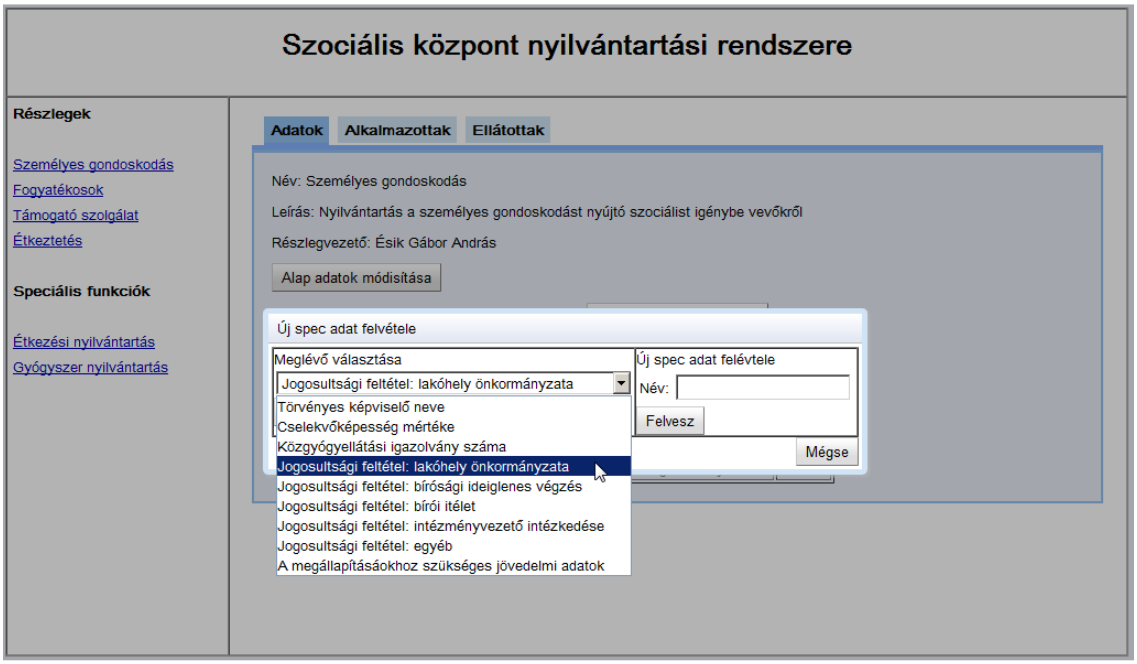
A megfelelő részleget kiválasztva a képernyő közepén megjelennek a számunka szükséges információk az adott részlegről. A megjelenő panel három füllel rendelkezik, alapból a választott részleg adatit tartalmazó fül aktív, így az ott megjelenő információk jelennek meg.

A panel első részében találhatóak a részleg minimális (alap) információi, melyek módosítása az „Alap adatok módosítása” gomb segítségével végezhető el.

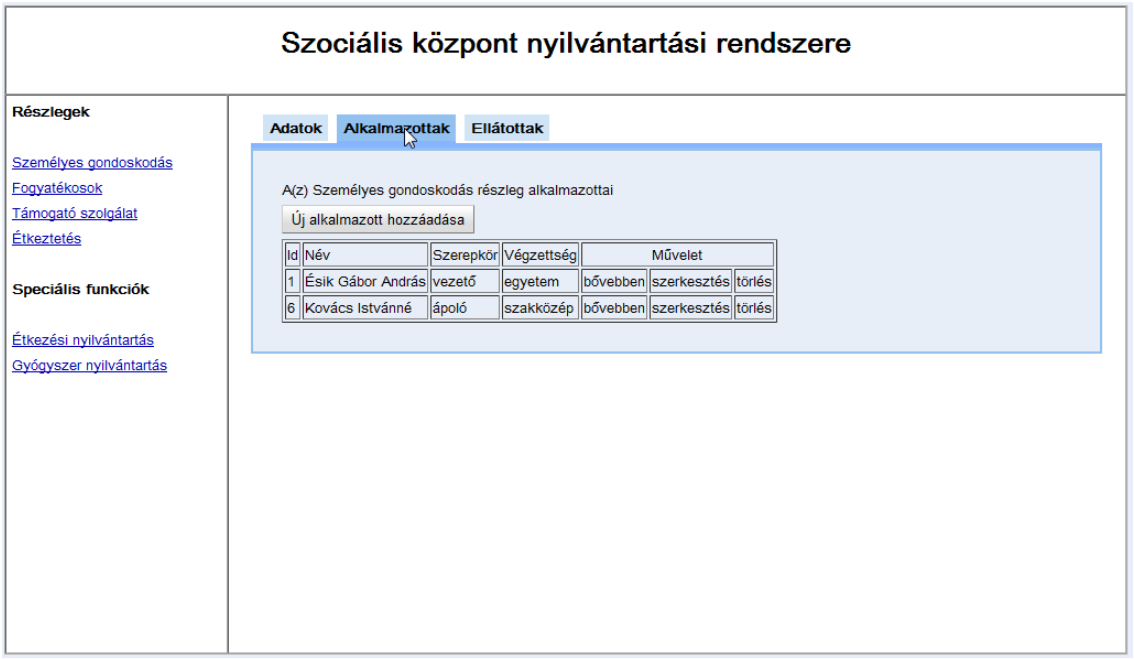


A részleg vezetőt egy legördülő menüből választhatjuk ki, ahol a megfelelő képesítéssel rendelkező alkalmazottak közül válogathatunk.

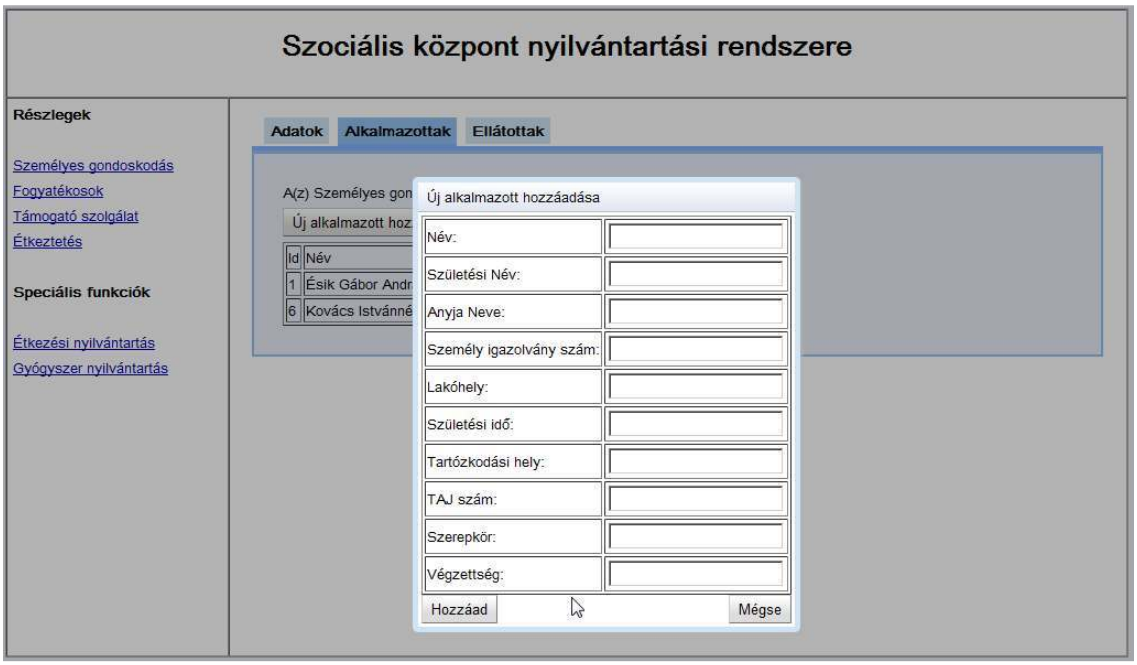
A részleghez új speciális adatot az „Új speciális adat hozzáadása” gomb segítségével tudunk adni. Itt két lehetőség közül választhatunk, vagy egy már más részleg által használt speciális adatot választunk ki a legördülő menüből, vagy létrehozunk egy teljesen újat.



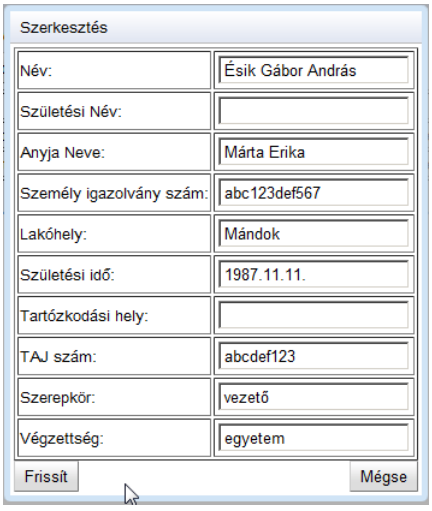
Az „Alkalmazottak” fülre áttérve megjelenítésre kerülnek a kiválasztott részlegen dolgozó alkalmazottak egy táblázatos formában, valamint itt található még egy „Új alkalmazottak hozzáadása” gomb melynek segítségével lehetőségünk van új alkalmazottat hozzáadni az adott részleghez.

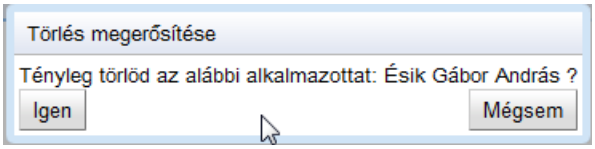


A meglévő alkalmazottak adatit lehetőségünk van csak megnézni a „bővebben” cella segítségével, míg szerkeszteni az adatokat a „szerkesztés” szöveggel tudjuk, valamint a „törlés” feliratra kattintva eltávolíthatjuk az adott alkalmazottat a részlegről. A szerkesztő és a bővített nézet elemek kinézetre azonosak, mindössze a gombok feliratában és a szövegmezők szerkeszthetőségében térnek el, valamint nagyon hasonló az új alkalmazott hozzáadó panel is csak itt természetesen minden szövegmező üres, és a műveletet a „felvétel” gombbal véglegesíthetjük.



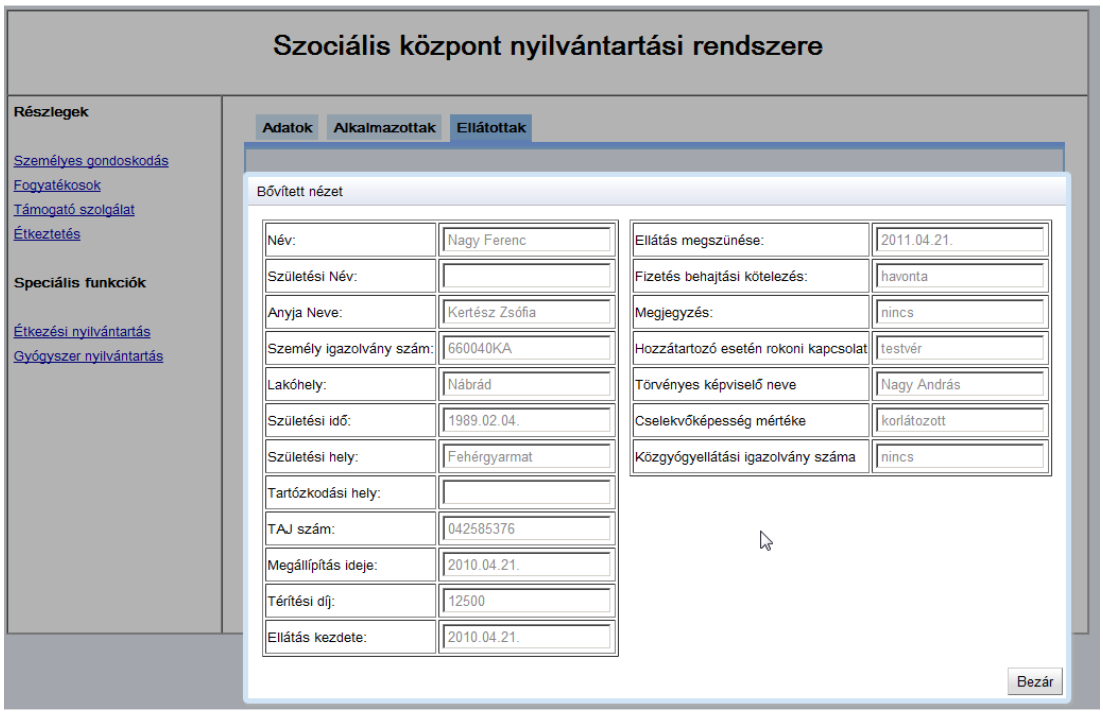
A szerkesztő nézet felülete és a törlés megerősítését kérő ablakokat az alábbi ábrák szemléltetik. A módosító nézet segítségével a rendszerben lévő adatok módosítása elvégezhető, a törlés előtt pedig egy ablak felugrik, hogy megbizonyosodjon róla a rendszer, hogy valóban törölni akarunk e, és az adott alkalmazottat akarjuk e törölni.



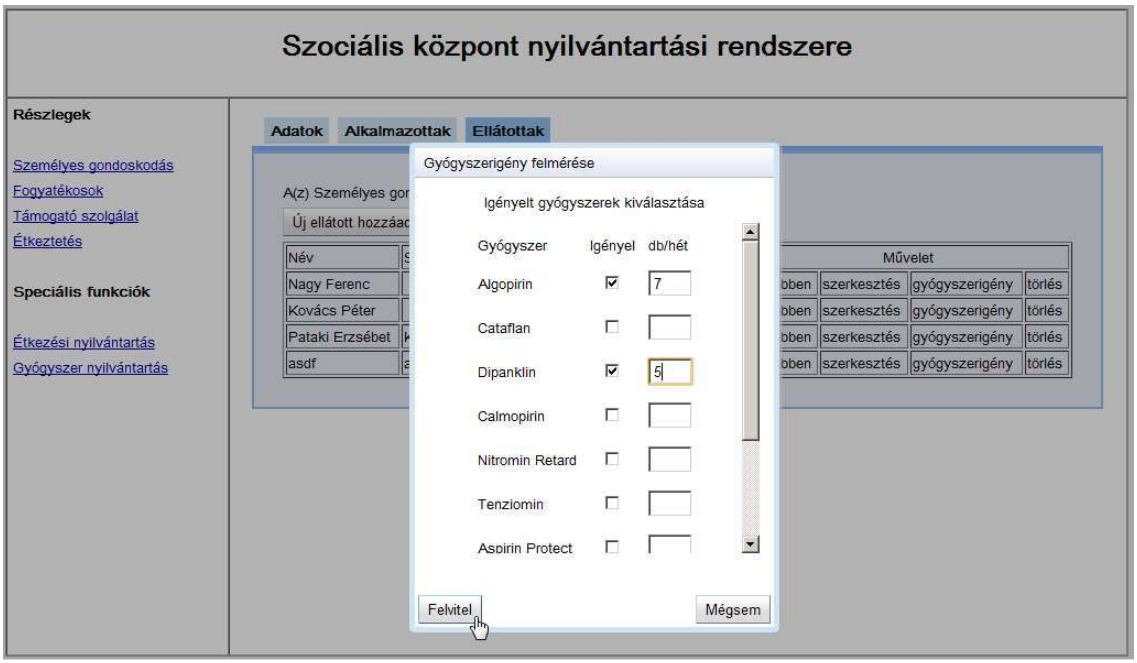


Az „Ellátottak” fül segítségével eljuthatunk a választott részleg által ellátott emberekhez. A felületen táblázatos formában láthatóak az ügyfelek adatainak egy része, melyek alapján szinte egyértelműen azonosítani lehet őket. Ezen felül, ha több információra is szükségünk van egy-egy ellátottról, akkor azt a „bővebben” szövegre kattintva el is érhetjük. Az ellátottakhoz hasonló módon működnek ezen a panelen is a különböző funkciók.

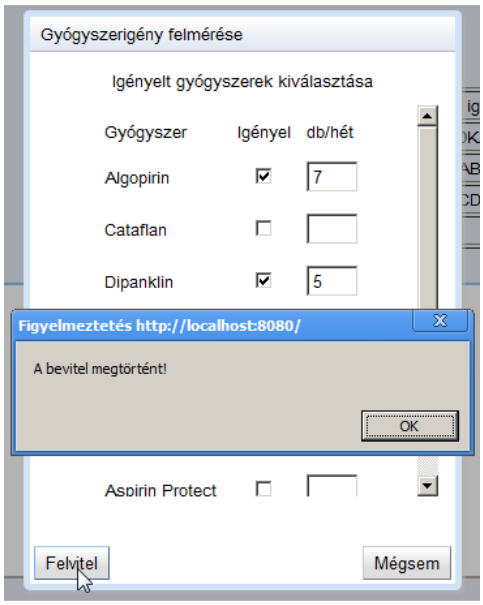
A különböző nézetek (bővített, szerkesztő, új) itt jóval több információt tartalmaznak, mivel az ellátottakról több adatot kell tárolni, mint a dolgozókról. A felület elrendezése és mérete más csupán, a gombok és a funkcióik megegyeznek az ellátottaknál tárgyaltakkal.

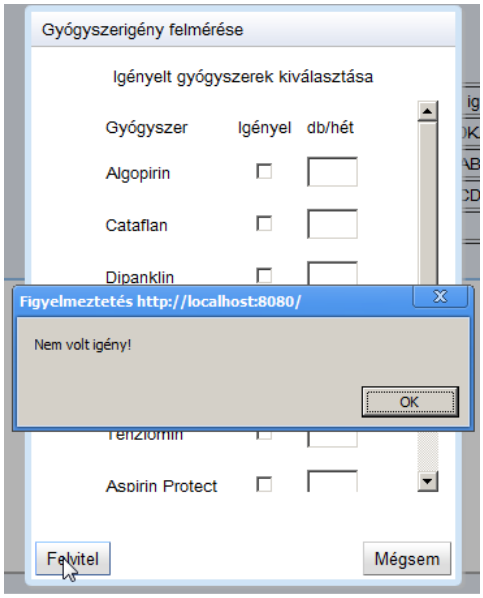


Az alkalmazottak táblázat műveletin túl még bekerült az ellátottakhoz egy új művelet, ami a gyógyszerigény felmérésére szolgál. Az intézmény ellátottainak többségének napi rendszerességgel van szüksége különféle gyógyszerekre és ezen igényeket lehet felvinni ezzel az opcióval, ennek segítségével majd összesítés készíthető a gyógyszerigényekről, ami majd segít a gyógyszerraktár karbantartásában.

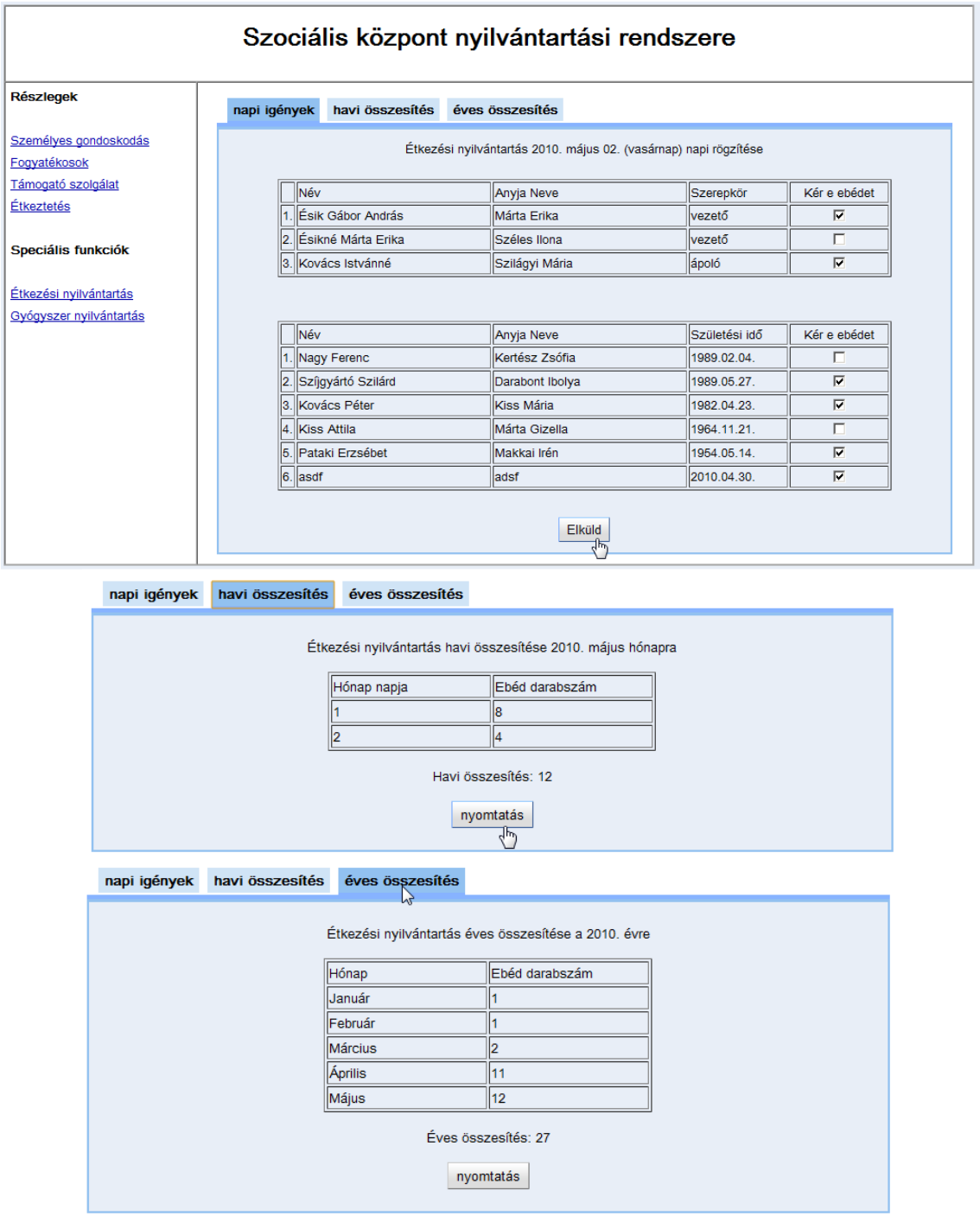


A sikeres adatbevitelt egy felugró ablak fogja jelezni nekünk, abban az esetben ha nem töltöttünk ki egy mezőt sem és úgy próbálnánk meg felvinni az űrlap tartalmát, azt az üzenetet kapjuk hogy nem volt igény, így nincs mit eltárolni. Sikeres igény esetén az OK gombra kattintva a felmérés űrlap bezárul.



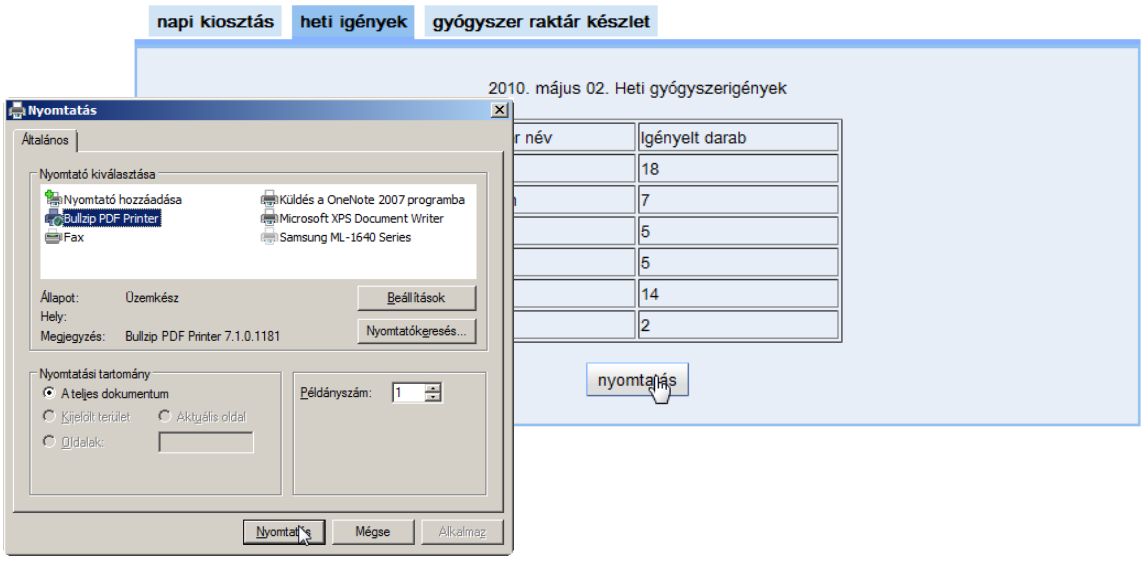


A speciális funkciók címszó alatt lévő menüpontokkal az intézmény két fontos nyilvántartásának kezelésére van lehetőségünk. Ezek közül az első az étkezések adminisztrálásáért felel, lehetőséget ad a napi igények felmérésére, és ebből havi majd éves összesítést is készít, melyeket ki is lehet nyomtatni.

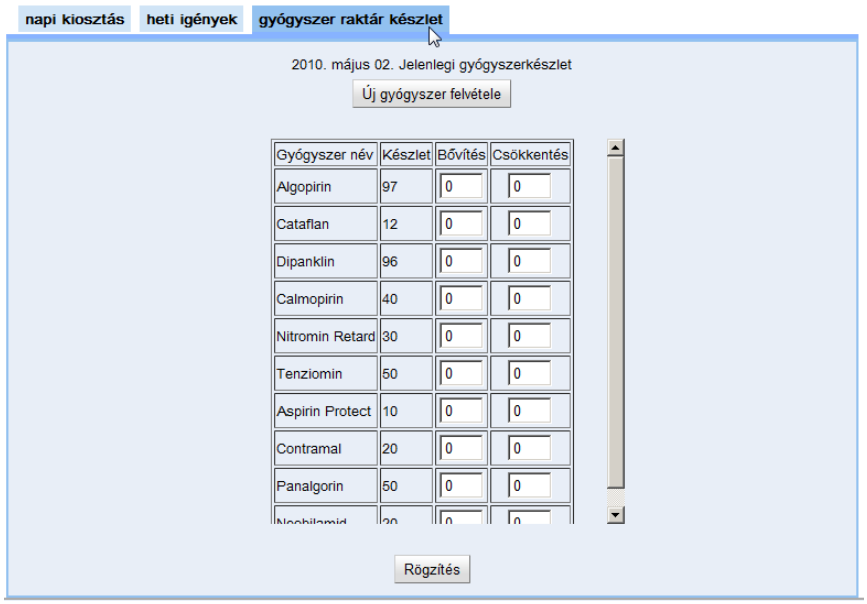


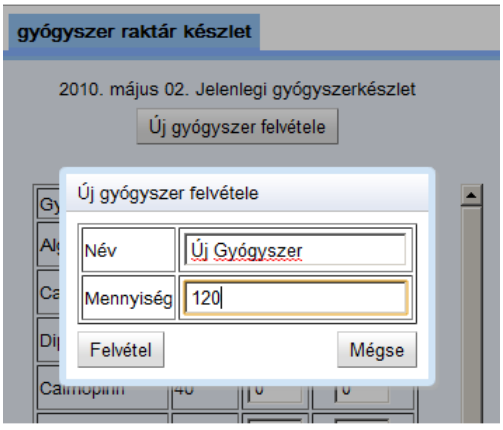
A nyomtatásra kattintva felugrik az operációs rendszerünk nyomtatóválasztó képernyője és kiválaszthatjuk a nyomtatót. A papírra csak az összesítés panel tartalma fog rákerülni (cím, táblázat, összesítés).

A gyógyszerek nyilvántartásáért felelős funkció keretében, feldolgozhatjuk az adott nap kiosztott gyógyszermennyiséget, a felületen megjelennek a rendszerben szereplő gyógyszerek és felvihetjük, hogy melyikből mennyi került kiosztásra. A kiosztott mennyiségek a gyógyszerraktárból le fognak vonódni, és az a raktár készlet fülön azonnal láthatjuk. A középső panelen az ellátottak által igényelt gyógyszermennyiségek összege olvasható, nyomtatható gyógyszerek szerint csoportosítva. Ezen adatok alapján egyszerűen nyomon követhetőek az igények, és a raktárkészlet állapota valamint elvégezhető a készlet módosítása.



A raktár panelen, kezelni tudjuk a központ gyógyszerkészletét, lehetőség van új gyógyszer felvitelére, valamint a meglévők mennyiségének módosítására, bővítés szükséges akkor amikor vásárlás történik, csökkentés akkor következhet be ha esetleg lejárt a szavatossága az adott gyógyszernek, esetleg valami külső hatás miatt fogyaszthatatlanná vált. Az itt elvégzett mennyiségi módosítások, azonnal mentődnek az adatbázisba, és a táblázat értékei is módosulnak.





Az adatok felvitele csak abban az esetben történik, ha ténylegesen volt módosulás, ha nem történt érték beírás, és úgy próbálnánk rögzíteni, a rendszer figyelmeztet hogy „Nem volt módosítás”.

VI. Összegzés

A szakdolgozatom témájául szolgáló alkalmazás fejlesztése során megtapasztalhattam, milyen összetett és körültekintést igénylő feladat egy ilyen program elkészítése. Először fel kell mérnünk az igényeket, hogy pontosan mire is szeretné használni a megrendelő a szoftvert. Sok esetben ezt a megrendelő sem tudja elég pontosan, és ebből adódnak majd a későbbi problémák, amikor a már elkészített alkalmazást kell bővíteni különféle funkciókkal, mert később derült ki, hogy jó lenne, ha még azt is meg tudná valósítani. Az ilyen konfliktusok elkerülése érdekében szokták előre lefektetni az igényeket a szerződésben, és az utólagos funkciókat már plusz költségért fejlesztés címen bele lehet rakni a programba.

Azt követően, miután kellőképp specifikálva volt az alkalmazás feladatköre, azt kellett eldönteni, hogy a megvalósítás során milyen technológiákat és tervezési mintákat alkalmazzak. Ezeknél a választásoknál fontos, hogy a programozó tisztában legyen a lehetőségekkel, ismerjen több nyelvet, technológiát, mivel csak így van lehetőség a választásra. Ezért is választottam a Java nyelvet, hiszen nagyon sok megoldási lehetőséget kínál bizonyos feladatokra, bővelkedik keretrendszerben, van külön vállalati eszközkészlete, és multi platformos, valamint a fejlesztő eszközök közül is lehet válogatni és mindezeket teljesen ingyen használhatjuk.

A megvalósításhoz használni kívánt eszközök kiválasztása után kezdődhet a tervezés. Nagyon fontos, hogy a kódolást mindig tervezés előzze meg, ahogyan több tanáromtól is hallottam tanulmányaim során: „a programírás papírral és ceruzával kezdődik”. Ennek megfelelően én is papírral kezdtem el a fejlesztést. Első körben rétegekre osztottam az alkalmazást, majd ezekbe besoroltam az ott megvalósítandó feladatokat. Egy-egy réteg megvalósítása közben többször is előfordult, hogy vissza kellett nyúlnom a papírhoz és a ceruzához, az adatbázis leképezésnél elég sok papírt felhasználtam, de jó pár oldalt kitettek a felületi tervek is. A fejlesztés során sokat könnyítettek a munkámon, hogy ezek a tervek el voltak már készítve.

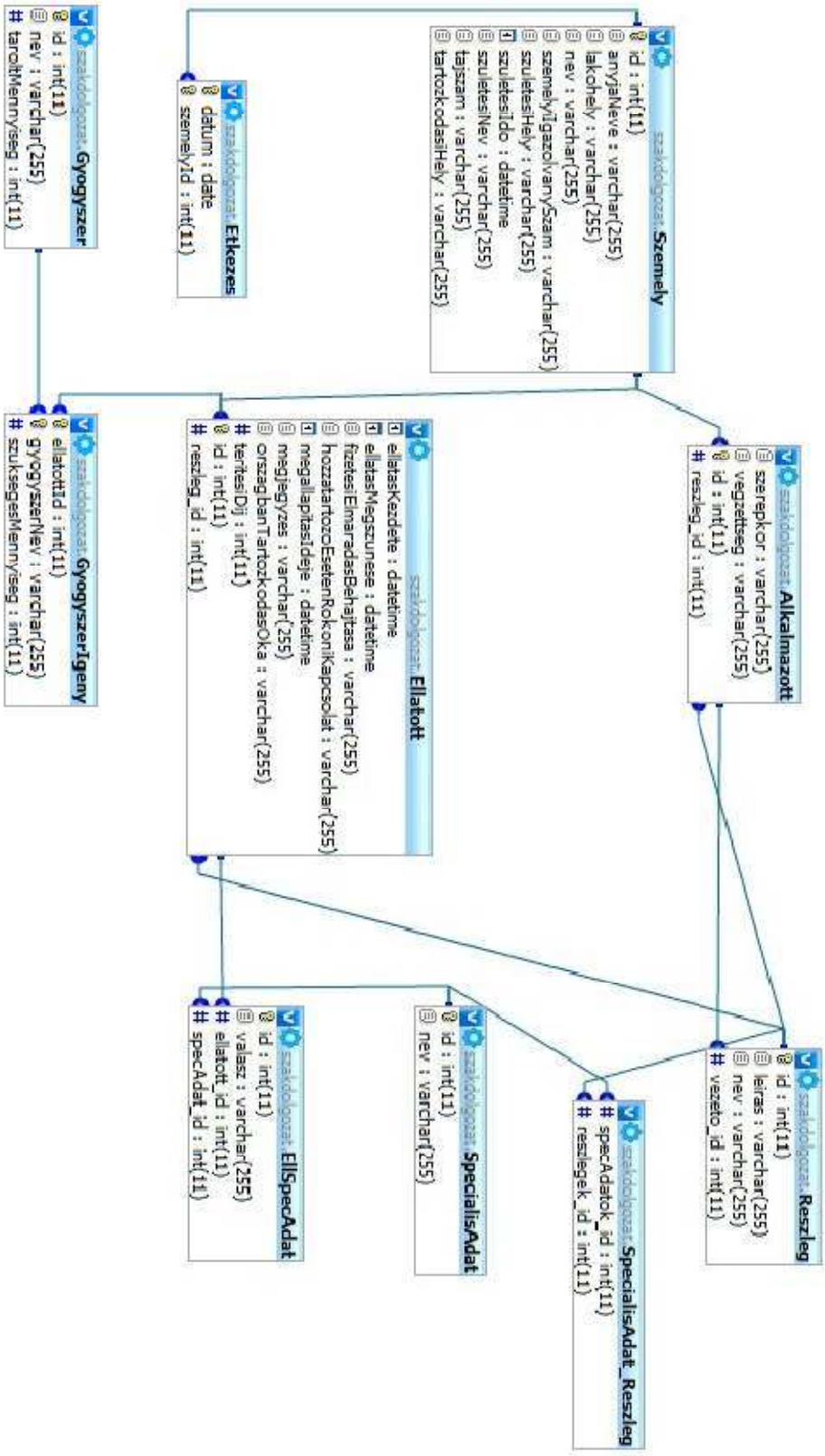
A dolgozatom fő célja az volt, hogy bemutassam egy alkalmazás elkészítését a ma rendelkezésünkre álló technológiák és módszerek segítségével, és szemléltessem, hogy ezeket miként lehet egymással összehangolva működésre bírni. Elég sok technikai és programozási problémával szembesültem a fejlesztés során, és megértem, hogy miért szükséges több ember egy összetettebb alkalmazás elkészítéséhez, hiszen egy ember könnyedén el tud veszni ennyi réteg és technológia hálójában.

Irodalomjegyzék

- Imre Gábor, Szoftverfejlesztés Java EE platformon, SZAK Kiadó, 2007
- Nyékyné Gaizler Judit: Java2: Útikalauz programozóknak 5.0 (1-2. kötet) ELTE TTK Hallgatói Alapítvány Budapest, 2008
- Ryan Dewsbury, Google Web Toolkit alkalmazások Kiskapu Kiadó 2008
- Vég Csaba: Instant Java/Java EE/SOA I - II. Logos2000 Kiadó, 2007
- http://code.google.com/intl/hu-HU/webtoolkit/
- http://gwt.google.com/samples/Showcase/Showcase.html
- http://hu.wikipedia.org/wiki/Ajax_(programozás)
- http://java.sun.com/javaee/5/docs/tutorial/doc/
- http://www.javaforum.hu/
- http://www.jboss.com/
- http://www.mysql.com/

Függelék

Az entitásokból generált adatbázis táblák, és a közöttük lévő kapcsolatok



A heti gyógyszerigények egy képbe nyomtatva

2010. május 02. Heti gyógyszerigények

Gyógyszer név	Igényelt darab
Algopirin	18
Calmopirin	7
Cataflan	5
Dipanklin	5
Még1Uj	14
Uj2	2

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani mindenkinek, akinek tanítása, munkája, segítsége hozzájárult ahhoz, hogy ez a dolgozat elkészülhessen.

Köszönettel tartozom:

Dr. Juhász István Tanár Úrnak a több év alatt megosztott ismeretért és tudásért;

Vágner Anikó Szilvia Tanárnőnek a témavezetőként nyújtott segítségéért;

Oláh Jánosné Intézményvezetőnek, aki betekintést engedett az intézmény működésébe, és biztosította számomra a szükséges nyilvántartási dokumentumokat.