Implementing a Java EE based information system as a student self-study task

Cs. Csizmadia

MÁV Szolgáltató Központ ZRT, Könyves Kálmán avenue 54-60, Budapest 1086, Hungary, jabbaco@gmail.com

Abstract

One of the highest levels of self-study is the problem solving, which primarily means solving a practical task using existing knowledge. During my studies, this development was the most complex learning process, since the knowledge, which had already been acquired, was not enough to solve the problem, moreover I had to take several problems. I had a challenging job to develop an information system for the agents of various insurance companies, since their previous administrative and client support methods include a lot of imperfection. The use of their technology is difficult, and it involves numerous possibilities for error, and is not up to date at all. Therefore, the exact method relies on the personal inventiveness and preparedness of the agents, which has an effect on the achievement of the agents and on the exterior opinion about the company. Thus, it also affects the competitiveness. In the process of acquiring knowledge, it has become essential to organize learning and manage time efficiently.

Keywords: *self-study; information-system; agency*

Java EE alapú információs rendszer megvalósítása önálló tanulói feladatmegoldás során

Csizmadia Cs.

MÁV Szolgáltató Központ ZRT, Könyves Kálmán krt 54-60, Budapest, 1086, Magyarország, jabbaco@gmail.com

Absztrakt

Az önálló tanulás egyik legmagasabb szintje a problémamegoldás, amely során elsősorban a már meglévő ismereteket felhasználva oldunk meg egy-egy gyakorlati feladatot. A képzésem során ez a fejlesztés valósult a legösszetettebb tanulási folyamatnak, mivel a problémamegoldáshoz nem volt elegendő a már megszerzett ismeret, több problémaszituációval is szembe kellett néznem. Kihívást jelentő feladatom volt egy információs rendszer fejlesztése különféle biztosítási cégek ügynökei részére, mivel azok adatkezelési, ügyféltámogatási módszerei esetenként kívánnivalót hagynak maguk után. Technológiájuk használata sokszor nehézkes, számos hibalehetőséget is magában hordoz, és így cseppet sem korszerű. Ezáltal a pontos metodika az ügynökök egyéni leleményességén, felkészültségén múlik, amely a biztosítási cég teljesítményére, külső megítélésére és ezekből következően a versenyképességére is negatív hatást gyakorol. Az ismeretszerzés során elengedhetetlenné vált a tanulás megszervezése és az idővel való hatékony gazdálkodás is.

Kulcsszavak: önálló tanulás; információs rendszer; ügynök

1. Bevezető

Az önálló hatékony tanulás útján megvalósított információs rendszer fejlesztés számos lehetőséget rejt a tanórán kívüli nem formális tanulás környezetben (Váczy, 2012). A nem

formális tanulás során a tanórákon túlmutató új ismeretekre lehet szert tenni, amelyek feldolgozhatók és beépíthetők a tanulási stratégiába, másrészt a szakmai kompetenciák fejlődéséhez is több szempontból hozzájárul. (Sagitova, 2014) (Barbosa et al, 2017) A nem formális tanulási módszerek fontos alapot adnak az életen át tartó tanulás megalapozása tekintetében is (Kővári, 2019). Mindezek mellett korábbi tanulmányok és tapasztalatok is segítették az önálló feladatkidolgozás során szükségessé vált problémák feltérképezését, megoldását.

1.1. Probléma ismertetése

Egy biztosítási cég ügynökei az eredményes munkásságuk érdekében számos nyilvántartást kénytelenek naprakészen vezetni. Nem elég csupán az ügyfeleik adatait tárolni, hanem az azokkal kapcsolatos kisebb-nagyobb teendőket is szinkronban kell tartaniuk. Mindez a gyakorlatban 2-3 táblázat és/vagy határidőnapló használatát jelenti elektronikus, vagy papíralapú formában. A bennük lévő adatok egy része így természetszerűen redundáns, ezért ezek állandó naprakészen tartása kényelmetlen feladatot jelent, és számos hibalehetőséget is magában hordoz. Ráadásul eme módszer korunk kívánalmainak sem tud eleget tenni, hiszen keresésre, összesített adatok lekérdezésére csak nagyon korlátozott mértékben használható. Miközben sem navigációs lehetőséget, sem közösségi hálózatokhoz való csatlakozást nem biztosít. Így az adatok nyilvántartása és kezelése az ügynökök egyéni leleményességén, felkészültségén múlik, amely így végső soron a biztosítási cég teljesítményére, versenyképességére is hatással van.

A felvázolt helyzet megoldására egy szoftverfejlesztő cég vállalkozott, más feladatok elvégzése mellett és viszonylag nagyobb határidővel. Személyes kapcsolatom révén én részt vehettem a munkálatok során, pontosabban a fenti probléma megoldását teljes egészében magamra vállaltam. Cserébe a cégvezetés hasznos információkkal látott el a biztosítási ügynökök ügymenetéről, valamint a biztosítók lehetséges elvárásairól. Így ilyen összefüggésben e szoftverfejlesztő cég lényegében a megrendelőmnek tekinthető, ezért a későbbiekben így is hivatkozom rájuk.

1.2. Elérendő cél kitűzése

A cél elérése érdekében számos feladatot kell sikerrel teljesíteni. Előbb magát a problémát kell behatóbban tanulmányozni, majd a lehetséges megrendelők ezzel kapcsolatos igényeit kell feltárni. Ezt követően biztosítani kell – a formálódó elképzelések alapján – a szükséges

adatok tárolását, megjelenítését és a velük kapcsolatos különféle szolgáltatásokat, funkciókat is. Előbb csupán gondolati síkon, majd később a fizikai valóságban is. Lényeges, hogy az információk közlésén és igény szerinti kezelésén túl az adatellenőrzést és a felhasználói azonosítást is meg kell oldanom. Majd a folyamat végeztével egy mindenre kiterjedő tesztelést kell foganatosítanom.

1.3. Távlati cél megfogalmazása

Jelen fejlesztés távlati célja a biztosítói adatbázisokhoz való laza kapcsolódás megteremtése a szolgáltatást igénybe vevő ügynökök beazonosítása érdekében. Erre legkorábban viszont csak egy sikeres tesztidőszakot követően kerülhet sor. Ezért a mostani fejlesztés során még egy teljesen független szoftver kialakítására kell törekedni.

2. Követelmény megfogalmazása

- A készítendő szoftver szolgáltatásait egy webhelyen keresztül lehessen elérni, amely a kliensként használt informatikai eszközök webböngészőjében jelenjen meg.
- A szolgáltatások igénybevételének előfeltétele a sikeres autentikáció.
- Az ügyfelekkel kapcsolatos beviteli adatokat egy vagy több weboldalon keresztül lehessen megadni.
- A bevitt adatok egy adatbázis-kezelő rendszerben kerüljenek eltárolásra.
- A szolgáltatások igénybevételekor a válasz is a weboldalon jelenjen meg, amelynek tartalmaznia kell az adatbázis-kezelő által tárolt adatok egy később meghatározott részét.
- A megjelenő adatokat lehessen rendezni, valamint utólag módosítani, törölni.
- A beviteli adatok megadása lehetőleg egyszerűen és gyorsan történjen.
- A webhely legyen áttekinthető, mégis tartalmazza a legfontosabb adatokat, biztosítsa a fontosabb funkciókat.
- A szoftver teremtse meg a közösségi hálózatokhoz, navigációs rendszerekhez és a biztosítói adatbázishoz való későbbi, további fejlesztést igénylő csatlakozás alapvető feltételeit.

3. Specifikáció

Az alkalmazással szemben támasztott alapvető elvárások megismerését követően sor került a program specifikációjára (elvárt szolgáltatások pontosítása, részletezése), amelynél

igyekeztem szöveges leírással kiegészített, formáknak alávetett módszert alkalmazni UML (Unified Modeling Language – egységes modellező nyelv) nyelven megfogalmazott diagramok segítségével. Hiszen ezáltal érhető el a leginkább a könnyű áttekinthetőség, és általa a félreértések elkerülése, valamint a tesztelési lehetőségek biztosítása. (Sándor S, László V, 2003) Továbbá Katona J et al. 2014 és 2016 munkákban is megjelenik az UML alkalmazása, mint hatékony szoftvertervezést támogató eszköz.

3.1. Adattárolási igények áttekintése

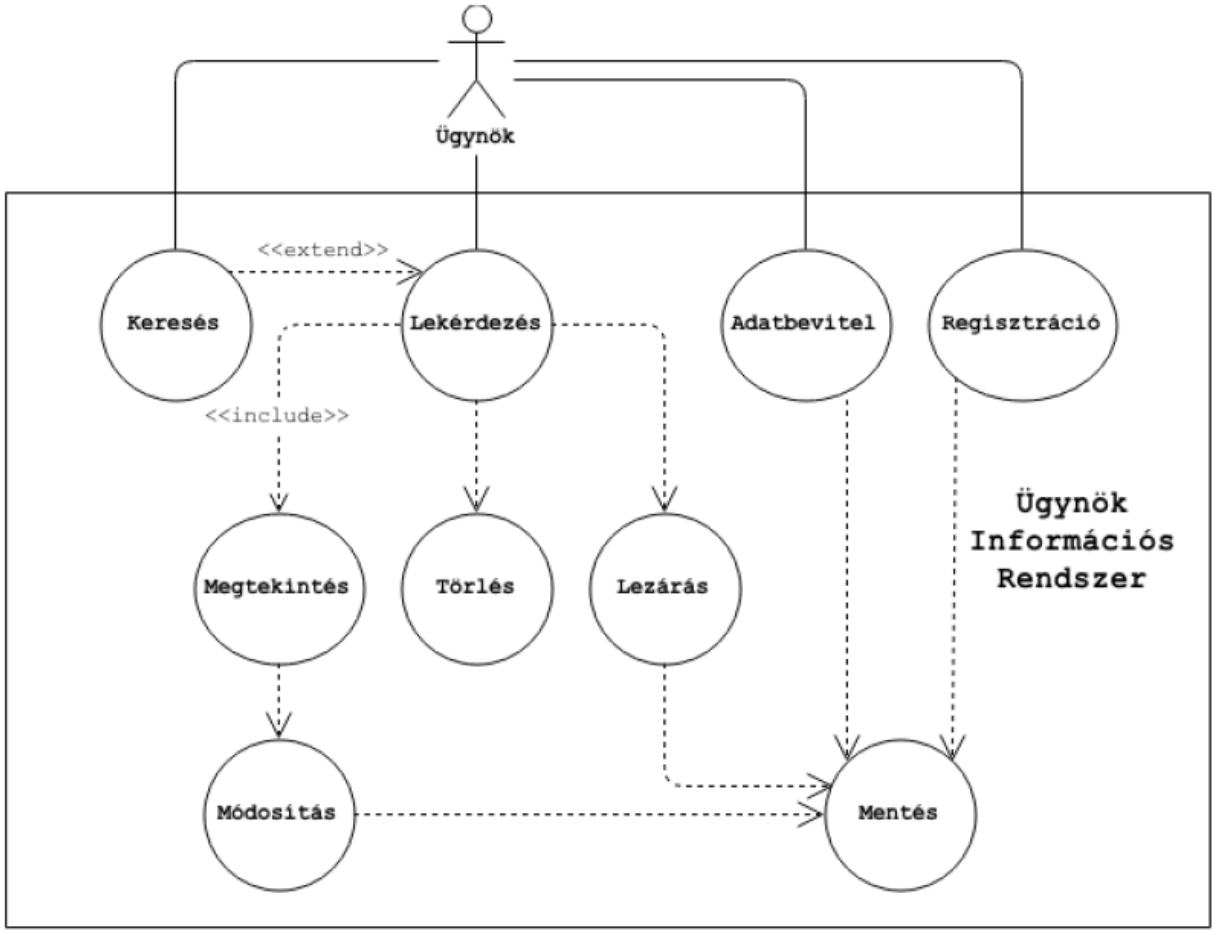
A specifikáció első lépéseként a tárolandó adatokat kellett áttekinteni, melyeket rövid idő alatt 5db egyedhalmazra választottunk szét. Majd ezt követően feltérképeztük a közöttük fennálló kapcsolatrendszert is. Egy-egy egyedhalmazon a modellezni kívánt világ közös tulajdonságokkal rendelkező egyedeinek halmazát értjük, melyek akkor vannak egymással kapcsolatban, ha van olyan, a feladat szempontjából fontos rendezőelv, amely az egyedhalmazok egyedeit egymáshoz rendeli. (Antal B, 2012)

Mindebből a további egyeztetési és tervezési munka érdekében egy egyszerűsített egyed-kapcsolat modellt készítettem, mellyel szemléletes módon, könnyen érthető formában tudtam ábrázolni a készítendő adatbázist.

3.2. Szolgáltatások megfogalmazása

- Egy újabb konzultáció során sikerült megfogalmazni a tervezendő alkalmazás alapvető szolgáltatásait, amelyet egy használati eset diagram segítségével rögzítettünk (1. ábra).
- Lekérdezés: Az adatbázisban tárolt adatok lekérdezése és megjelenítése az adott weboldalon, listák formájában. Külön lekérdezés szükséges az ügyféladatok, a hozzájuk kapcsolódó ügyadatok és az ügyekhez kapcsolódó részfeladatok részére.
- Keresés: Szelektív lekérdezés az adatbázisban szereplő ügyfelek közül. A kereséshez szükséges bemenő paraméter az ügyfél neve, illetve a nevének töredékszavai.
- Adatbevitel: Új adatok felvitele az adatbázisba. Külön beviteli lehetőség szükséges az ügyféladatok, az ügyek és a részfeladatokhoz részére. Az adatbevitel során fontos szempont a gyorsaság és az egyszerűség, ezért csak a legfontosabb adatok bevitele legyen kötelező, és ahol megoldható, ott legyen lehetőség alapértelmezett értékek megadására is.

- Regisztrálás: A készülő webalkalmazás szolgáltatásait igénybe venni kívánó biztosítási ügynökök regisztrálhatják saját magukat egy regisztrációs űrlap kitöltése által.
- Megtekintés: A weboldalon lévő listák egyes elemeinek adatait jeleníti meg részletesebben. A lekérdezések során alapértelmezetten mindig az első rekord adatai kerüljenek kifejtésre.
- Törlés: Törli az adott lista egyes eleméhez tartozó rekordot az adatbázisból.
- Lezárás: A listák egyes elemeinek státuszát „Zárolt”-ra állítja. A lekérdezések eredményeiben a Zárolt rekordok alapértelmezetten ne jelenjenek meg.
- Módosítás: Elvégzi a lista éppen megtekintett rekordjához tartozó adatok módosítását, illetve új adatokkal való kiegészítését.
- Mentés: Új adatok felvitele vagy a régiek módosítása után menti azokat az adatbázisba.



1. ábra: Elvárt szolgáltatások

4. Felhasznált technológiák

A további tervezés érdekében ki kellett választani a felhasználandó technológiákat, valamint át kellett tekinteni az általuk kínált lehetőségek körét. Mivel a tervezendő szoftver végső produktuma egy vagy több weboldal megjelenítése, ezért az elemzést is itt volt érdemes elkezdeni.

A weblapok forráskódját XHTML (Extensible Hypertext Markup Language – kiterjesztett hiperszöveges jelölőnyelv) formátumban, HTML5 nyelven terveztem megírni, míg a formázást és elrendezést CSS (Cascading Style Sheets – egymásba ágyazott stíluslapok) alkalmazásával kívántam biztosítani.

A legfontosabb döntés azonban a használandó programozási nyelvekre vonatkozott, a megfelelő nyelv kiválasztása ugyanis egy hozzá illő fejlesztői környezettel együtt jelentősen lerövidítheti a forráskód elkészítésére fordított időt, ráadásul a végeredmény minőségére, hatékonyságára is hatással lehet.

A követelményrendszer ismételt áttanulmányozása során arra a következtetésre jutottam, hogy a készítendő alkalmazásnak a perzisztencia (adatok állandósága), a többszálúság (adatok párhuzamos elérése) és a távoli elérhetőség kritériumainak feltétlen eleget kell tennie, ugyanakkor ezek megvalósításának pontos részleteivel már nem kívánok foglalkozni. Ezért én a Java Enterprise Edition (Java vállalati verziója, röviden Java EE), mint programozási platform mellett döntöttem. Ez ugyanis moduláris szoftverkomponensek segítségével, széleskörűen támogatja a többrétegű, elosztott alkalmazások készítését, miközben a fejlesztő számára indifferens részleteket jótékonyan elrejti. (Péter B. et al. 2007), (Jendrock et al, 2014).

A perzisztencia biztosítását egy relációs adatbázissal kívántam megoldani, amit egy Oracle adatbázis-kezelő szoftverrel terveztem menedzselni. Eközben a weblapokhoz tartozó funkcionalitás megteremtését JavaServer Faces (röviden JSF) technológia használatával gondoltam, amely egy a Java EE platform részét képező, komponens-orientált webprogramozási keretrendszer. (Péter B. et al. 2007), (Jendrock et al, 2014).

5. Programtervezés

A megrendelői elvárások és a technológiai lehetőségek megismerése után sor kerülhetett a szoftverfejlesztés következő fázisára, vagyis a megvalósítás (implementálás) terveinek egyre részletesebb kidolgozására.

5.1. Tervek felvázolása

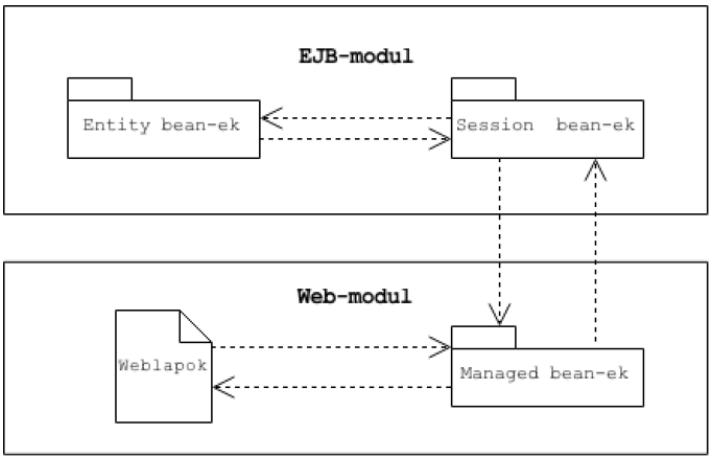
A konkrét tervezet kialakításának első lépéseként a szerkezeti elemeket igyekeztem meghatározni, vagyis a szükséges komponensek áttekintésével egy statikai modellt készíteni.

A szoftver vázát egy EJB- és egy webkonténer alkotja, amelyek strukturális szempontból modulnak is nevezhetők; az első egy jar, míg az utóbbi egy war kiterjesztésű állományként szerepel az alkalmazásszerverek fájlrendszerében.

Mindkét modul további részegységeket tartalmaz. Itt találhatók az üzleti logikai réteg entity és session bean-jeinek, valamint a webréteg managed bean-jeinek csomagjai (package) is. Illetve a webmodulba kerül egy web nevű könyvtár is, amelyben a kliensréteg által megjelenített weblapok forráskódjai, azok stíluslapjai és az esetleges szkriptfájlai kerülnek elhelyezésre. Mindezen komponensek a szolgáltatások biztosítása érdekében természetesen egymás közötti kommunikációt folytatnak. (2. ábra)

Érdemes megfigyelni, hogy az entity bean-ek és a weblapok közvetlenül sosem kerülnek kapcsolatba. Így bármelyikük továbbfejlesztése, esetleges cseréje a másik működőképességére nincs hatással (a réteges architektúrák elvének megfelelően).

A tervből szándékosan maradt ki a message-driven bean-ek csomagja, ugyanis jelen specifikációban aszinkron üzenetkezelésre még nem lesz szükség.



2. ábra: Alkotóelemek diagramja

5.2. Tervek finomítása

A szerkezeti elemek felvázolása után a pontos részletek kidolgozásába fogtam a programterv későbbi implementációjának megkönnyítése, valamint a koncepcionális hibák és hiányosságok felderítése érdekében.

5.2.1. Adatbázis tervezése

A specifikáció során már sikerült meghatározni, hogy a kitűzött feladat megvalósításához milyen adattárolási igényeknek kell megfelelni. Azonban az összetett vagy többértékű attribútumok tárolása, kezelése még átgondolást igényelt.

A megrendelő eredeti elképzelése szerint az ügyfelek és az ügynökök állandó lakcímei, levelezési címei vagy cégek esetén a székhelyük adatai egy `String` típusú (szöveges) állományban kerültek volna eltárolásra. Azonban a későbbi módosíthatóság, valamint a keresési, lekérdezési lehetőségek támogatása érdekében célszerűbbnek tartottam ezen adatokat részekre bontva, egy-egy külön attribútumba helyezni. Ráadásul e címekből egy ügyfél vagy ügynök esetén egyszerre több is előfordulhat. Így ezen adatok tárolására egy újabb adattáblát (Címek) vezettem be, melynek elsődleges kulcsával kívántam biztosítani a kapcsolatot az Ügyfél, illetve az Ügynök táblával.

Azonban a név attribútum esetén ezt a megoldást eltúlzottnak találtam. Ezért a többértékű keresztnév használata helyett egyszerűen bevezettem az Ügyfél és Ügynök táblába egy újabb attribútumot, a középső nevet. Illetve hasonlóképp jártam el a többértékű telefonszám esetén is.

Döntésemet jelentős részben az is indokolta, hogy a követelmények között az egyszerűsített adatbevitel kiemelt fontossággal szerepelt. Ezért a másodlagos adatok megadását (cím, név, telefonszám) csupán lehetőségként kell biztosítanom. De a használatukra valószínűleg majd csak ritkán kerül sor.

Az adatok áttanulmányozása során arra is felfigyeltem, hogy az ügyfelek és a jogi személyiségű ügyfelek (cégek) kapcsolattartó személyei lényegében csak az ügynökhöz való viszonyukban különböznek egymástól. Természetes, hogy egy kapcsolattartó esetén elegendő volna a nevének és elérhetőségének tárolása is. Azonban a későbbiek során – más ügyből kifolyólag – még belőle is lehet egy újabb ügyfél. Így a majdani (redundáns) adatrögzítés lerövidíthető volna, ha a kapcsolattartók adatai is eleve az Ügyfél táblában kerülnének letárolásra. Ilyen értelemben tehát az ügynökök egy egységes felületen keresztül tudnák igazgatni a már meglévő és a még csak kilátásban lévő ügyfeleik adatait. Ezáltal a szoftver az ügyfélkörük bővítéséhez is jelentős segítséget nyújthatna.

Ezen gondolatmenetet még tovább folytatva felmerült, hogy a cégek esetén nem csak a kapcsolattartók adatait lenne célszerű eltárolni. Hanem lehetőséget kéne biztosítani a megismert (egyéb) munkatársak felvételére is. Egyrészt a fent említett, praktikussági okok

miatt. Másrészt a hatékonyabb marketing kedvéért. A legjobb reklám ugyanis szájhagyomány útján terjed. Ezért amennyiben az alkalmazás a meglévő és leendő ügyfelek munkatársi viszonyait is tudná szemléltetni, akkor a biztosítási ügynökök sokkal célirányosabban tudnák a klientúrájukat bővíteni.

Az adatokat egészen más szemszögből vizsgálva feltűnt, hogy az adatbevitel során sokszor egy előre definiált listából kéne az egyes elemeket kiválasztani. Ezért az alacsonyabb tárhelyigény érdekében célszerűnek tartottam a listák elemei helyett csupán az elemek kezdőbetűit tárolni (az egyértelmű azonosíthatóság elvének betartása mellett).

5.2.2. Szolgáltatások összehangolása

A követelmények részletes kidolgozásával világossá vált, hogy az alkalmazásnak milyen alapvető szolgáltatásokat kell nyújtania a felhasználói felé. Ezek döntő többségénél azonban nem egymástól elszeparált funkciók megvalósításáról lesz szó, hanem egymással összedolgozó, tevékenységükben összehangolt szolgáltatásokról. Éppen ezért érdemesnek tartottam megvizsgálni az együttműködésük mikéntjét is.

5.2.3. Adatok megjelenítési terve

Véleményem szerint úgy lehet a lehető legtöbb, hasznos információt elhelyezni a szoftver kimenetéül szolgáló weblapon (az egyszerűség és átláthatóság követelményeinek betartása mellett), ha az adatok megjelenítése egy hierarchikus rendszerben egymásra épül.

5.2.4. Navigációs terv

Az adatok megjelenítésével kapcsolatos egyeztetés során újabb nézőpontok kerültek napvilágra. Ugyanis az ügyfélközpontú információközlésen túl a napi feladatok könnyed áttekintése is praktikus volna. Illetve a későbbiekben, a már rendeltetésszerűen használt alkalmazás a biztosítási ügynökök teljesítményéről is szolgáltathatna összesített adatokat. Ebből következően újabb igények kerültek megfogalmazásra, és így a követelmények leírása is az alábbiak szerint bővült:

- A lekérdezési szisztéma alapvetően háromféle séma szerint épüljön fel:
    1. Ügyfelek és a hozzá kapcsolódó adatok megjelenítése.
    2. Aktuális feladatok és az azokhoz kapcsolódó ügyféladatok szemléltetése.
    3. Ügynökök és a hozzájuk tartozó tevékenységek adatainak lekérdezése.

- Be lehessen állítani, hogy a három séma közül melyik legyen alapértelmezetten az adott ügynök számára.

Az igény kielégítése érdekében felkerül a weboldalra egy navigációs panel, amin keresztül elérhető lesz a háromféle szisztéma, vagyis az Ügyfél, a Napi és a további fejlesztést igénylő Vezetői nézet.

6. Implementáció

6.1. Perzisztencia biztosítása

A technológiai háttér feltérképezése, majd a tervek felvázolása, pontosítása után elérkezett az idő mindezen ismeretanyag konkrét formába öntésére, melynél az eddig bevált módszer szerint igyekeztem az alapoktól, a tárolandó puszta adatoktól kezdve kiindulni.

Ennek értelmében előbb SQL (Structured Query Language – strukturált lekérdező) nyelven megfogalmazott konfigurációs fájlokat készítettem, amelyek futtatása által az adatbázisban létrejöttek a táblák, azok attribútumai, megszorításai. Illetve helyenként alapértelmezett (default) értékeket is megadtam egy-egy tulajdonság kapcsán, hogy az adatrögzítést végző ügynökök munkáját ezzel is elősegítsem.

6.1.1. Vállalati Entity bean-ek létrehozása

Mivel lényegében véve az entitiy bean-ek szavatolják a kapcsolatot a szigorúan vett adatréteg és a funkcionalitásért felelős üzleti logikai réteg között, ezért munkálkodásomat feltétlen ezek megteremtésével kellett folytatnom.

Minden egyedhalmaz számára egy-egy osztályt hoztam létre, amelyben deklaráltam a szükséges tagváltozókat, és az azok elérését biztosító getter – setter metódusokat.

Például a `@Table`-lel meghatároztam, hogy ez az osztály az adatbázis mely táblára képződjön le az alábbiak szerint:

```
@Entity
@Table(name = "CLIENT")
public class Client implements Serializable {
```

6.2. Funkcionalitás életre keltése

6.2.1. Managed bean-ek teremtése

Bár az adatok minden információ-rendszer esetén látszólag kiemelt jelentőséggel bírnak, valójában a rajtuk végzett műveletek és a tálalásuk módja érdemel igazán figyelmet. A puszta adat ugyanis különböző kontextusba helyezve más-más jelentéssel kaphat.

Az efféle műveleteket pedig (többek közt) a managed bean-ekben elhelyezett metódusok végzik. Ezért következő lépcsőként ennek megalkotásába fogtam. Vagyis létrehoztam egy menedzselt osztályt a webkonténeren belül, amely a vezérlő és modell feladatkörét egyaránt el fogja látni.

Annotációk segítségével a bean-t elneveztem, ezen a néven regisztráltam, majd a hatókörét nézet típusúra állítottam. Innentől fogva ez a konténer által menedzseltnek tekinthető.

```
@ManagedBean(name = "managedClient")
@ViewScoped
public class ManagedClient implements Serializable{
```

6.2.2. Metódusok megalkotása

A vezérlő logika vázának megteremtése után még azt fel kellett tölteni tartalommal, vagyis az egyes szolgáltatáscsokrokhoz tartozó metódusokat meg kellett valósítani. Ennek folyamatát az Ügyfél-nézethez tartozó, az ügyféllista és űrlap menedzselését végző bean működésének bemutatásával prezentálom. Az aktuális ügyfelet lekérdező metódus az alábbi módon néz ki:

```
public List<Client> listClients() throws IOException {
clients = clientFacade.findAll(filter);
if (clients != null && !clients.isEmpty()&& client == null)
{ client = clients.get(0);
clientFacade.setClient(client);
} return clients; }
```

6.2.3. Session Bean-ek felépítése

A managed bean-ek megalkotása során számtalan alkalommal hagyatkoztam az entitások adatbázisból lekért adataira. Csakhogy ezek eléréséhez, életciklusaik kezeléséhez a session bean-ek használata szükséges. Ennek értelmében entitásonként létrehoztam egy-egy ilyen „köztes” osztályt az alábbiak szerint:

```
@Stateless
public class AddressFacade extends AbstractFacade<Address> {
@PersistenceContext(unitName = "AIS-PU")
private EntityManager em;
```

6.2.4. Lekérdezések végrehajtása

A menedzselt bean-ekben már létrehoztam azon lekérdező metódusokat, amelyek a tagváltozókba mentve biztosítják a megjelenítési réteg számára szükséges adatok elérését. Ők maguk a session bean-ek közvetítésével férnek hozzá az entitások attribútumaihoz. Csakhogy a session bean-ek már a Java Persistence Query Language (a Java perzisztens lekérdezőnyelve, röviden JPQL) nyelven fogalmazzák meg a kéréseiket.

6.3. Felület megalkotása

Miután megbirkóztam a szoftver lelkét képező üzleti logikai réteg legfőbb problémáival, a felhasználó szempontjából legfontosabb rész, a megjelenítendő weboldalak felépítésébe kezdtem. Ám minden építkezés az alapozással kezdődik, ezért előbb a weblapok kezelésére hivatott szervletet kellett beállítanom.

6.3.1. Szervlet beállítása

A kéréseket kiszolgáló szervlet és az adott weboldal közötti kapcsolatot a `web.xml`-nek nevezett konfigurációs fájl biztosítja. Vagyis a webszerver az ebben foglaltak alapján találja meg „az adott URL-hez tartozó szervletet”. Ezért az alábbiak szerint regisztráltam a `Faces Servlet` nevű szervletet, majd beállítottam, hogy az `xhtml` kiterjesztésű állományaimat ő dolgozza fel.

6.3.2. Adatok konvertálása és validálása

A szoftver működésének elengedhetetlen feltétele, hogy a tárolandó ügyfelek, ügyek és feladatok adatait az ügynökök adhassák meg a weblapokon elhelyezett, beviteli komponensek segítségével. Csakhogy ezen attribútumok nem csupán szöveges információt tartalmazhatnak, mivel sok esetben számok, dátumok tárolására is szükség volna. Viszont az adatbevitelre szolgáló mezők mindenképp szöveges (String) típusú adatokat továbbítanak a webszerver felé. Ezért e probléma megoldása érdekében a JSF beépített konvertereit célszerű alkalmazni.

6.3.3. Belső navigáció

Mivel a többféle lekérdezési szisztéma érdekében a webes felületet kétféle nézetre osztottam, így e weblapok közötti navigáció is kardinális kérdéssé lépett elő. Nem csupán statikus navigálási lehetőséget kellett biztosítanom a felhasználók számára (másik oldalra mutató linkek), hanem a funkcionalitásba ágyazott, dinamikus hivatkozásokat is célszerű volt elhelyezni (gombok).

6.3.4. Utolsó simítások

Bármilyen jól is működtek már ekkor a szoftver által biztosított szolgáltatások, az i-re a pontot a külcsínnek kellett feltennie. Ennek érdekében HTML címkékhez, CSS stíluslapokhoz és egy aprócska JavaScript program használatához folyamodtam. Lubbers, P., Albers, B., Salim, F., & Pye, T. (2011)., Frain, B. (2012)

Lényeges ugyanis, hogy a JSF címkék sokasága közt az egyszerű HTML címkék is használhatóak. Sőt, valójában a JSF nyelvi elemei vannak HTML címkékbe ágyazva, mivel a kliens csak ez utóbbiakat képes megjeleníteni. Így én fieldset címkét használtam az űrlapok keretbe foglalására. CSS alkalmazásával pedig (többek közt) sikerült más-más színnel jelölve megkülönböztetnem a listákban szereplő páros és páratlan sorokat. Miközben a JavaScript fájl segítségével a listák rendezési lehetőségeiről gondoskodtam.

7. A program tesztelése

A tesztelési fázis első, rám eső felét részben már sikerült lefolytatnom. Természetesen már a kód fejlesztése során is teszteltem a program működését, de szisztematikus viszgálatot még csak most végeztem.

Ezáltal többek közt teszteltem az Ügyfél nézet alábbi funkcióit:

- Új ügyfél felvételét, már meglévő megtekintését, módosítását, illetve törlését.
- Új ügy felvételét, már meglévő megtekintését, módosítását, lezárását, illetve törlését.
- A feladatlistában szereplő egyedek lezárását és törlését.
- Ügyfél, illetve ügy választása esetén a „kapcsolt” adatok megtekintését.
- Egy-egy lista minden adatának törlése esetén a rendszer válaszát.

Majd megvizsgáltam a Napi nézet alábbi szolgáltatásait:

- Új ügy felvételét, már meglévő megtekintését, módosítását, illetve törlését.

- Új feladat felvételét, már meglévő megtekintését, módosítását, illetve törlését.
- Napi feladat kiválasztása esetén a „kapcsolt” adatok megtekintését.
- A megtekintett ügyfél adatainak módosítását.
- Egy-egy lista minden adatának törlése esetén a rendszer válaszát.

Végezetül ellenőriztem az új ügynök regisztrációs lehetőségeit, és a nézetek közötti navigációt.

A teszteredmények kiértékelése után (és közben) javítottam a felmerülő hibákat, amelyek nagyrészt figyelmetlenségből fakadtak. A megoldásuk struktúrális változással nem járt.

8. Összegzés

A nem formális tanulás során elvégzett alkalmazásfejlesztési feladat eredményes lett, sikerült a felmerült problémákat megoldani, a tesztelési eredmények is pozitívok lettek. A tanulás ezen nem formális módja hatékonyan kiegészíti az iskolai keretek között tanultakat, azok alkalmazása terén további ismeretek elsajátítását alapozza meg.

A feladat során egy ügyfélkapcsolat kezelő rendszer kialakítsa volt a cél biztosítási cégek ügynökei számára. A feladat a valós életből jött probléma megoldására adott lehetőséget, mely további előnye volt az önálló feladatnak. Összességében a bemutatott fejlesztéshez hasonló feladatok ötleteket adhatnak más diákok, oktatók számára is az önálló feladatok alkalmazásának minél szélesebb körben alkalmazottá tétele érdekében.

Irodalomjegyzék

Sagitova, R. (2014). Students’ self-education: learning to learn across the lifespan. *Procedia-Social and Behavioral Sciences, 152,* 272-277.

Barbosa, J., Silva, A., Ferreira, M. A., & Severo, M. (2017). The impact of students and curriculum on self-study during clinical training in medical school: a multilevel approach. *BMC medical education, 17*(1), 9.

Péter B., et al. (2007). *Szoftverfejlesztés Java EE platformon*. Szak Kiadó Kft.

Antal B. (2012). *Az adatbázis-kezelés alapjai.* Buza Antal, Dunaújváros, 2012.

Sándor, S., & László, V. (2003). *Szoftvertechnológia és UML*. ELTE Eötvös K.

Kővári A. (2019). A felnőttoktatás 4.0 és az az ipar 4.0 kihívásai az életen át tartó tanulásban. PEDACTA, 9(1), 9–16.

Katona J et al. (2017). Examine the effect of different web-based media on human brain waves. In *2017 8th IEEE International Conference on Cognitive Infocommunications (CogInfoCom)*. 407-412

Lubbers, P., Albers, B., Salim, F., & Pye, T. (2011). *Pro HTML5 programming* (pp. 107-133). New York, NY, USA:: Apress.

Katona J et al (2014). Control of incoming calls by a windows phone based brain computer interface. In *2014 IEEE 15th International Symposium on Computational Intelligence and Informatics (CINTI)*. 121-125.

Jendrock, E., Cervera-Navarro, R., Evans, I., Haase, K., & Markito, W. (2014). *The java ee 7 tutorial* (Vol. 1). Addison-Wesley Professional.

Frain, B. (2012). *Responsive web design with HTML5 and CSS3*. Packt Publishing Ltd.

Váczy Zsuzsa (2012). Fejlesztési lehetőségek nem formális tanulási környezetben. Könyv és Nevelés. 2012(1).