Szakdolgozat

Docsa Tamás

Debrecen

2011

Debreceni Egyetem
Informatikai Kar
Információ Technológia Tanszék

Szerződés nyilvántartó rendszer fejlesztése

Témavezető:
Kollár Lajos
Egyetemi Tanársegéd

Készítette:
Docsa Tamás
Programtervező informatikus

Debrecen
2011

Tartalomjegyzék

Tartalomjegyzék 3
1. Bevezetés 4
2. Alkalmazott technológiák 6
2.1. Java 6
2.2. GlassFish 7
2.3. Apache Ant 7
2.4. Web Service 8
2.5. PostgreSQL 8
3. A szoftver felépítése 9
3.1. Adatbázis struktúra 10
3.2. Szerveroldali alkalmazás 14
3.3. Kliens alkalmazás 21
3.3.1. Szerződés fogalma 21
3.3.2. Architektúra 22
3.3.3. Többnyelvűség 25
3.3.4. Felhasználói felület 26
4. Fejlesztési lehetőségek 40
5. Irodalomjegyzék 42
6. Függelék 43
6.1. Rövidítések 43

1. Bevezetés

A mai modern világban egyre nagyobb teret hódítanak az elektronikus médiák és egyre inkább háttérbe szorítják a papír alapú termékeket. A nyomtatott sajtó helyét szép lassan átveszik az internetes hírportálok, a könyvek jelentős része pedig már elektronikus könyv formában is elérhető. Azonban a hivatalos dokumentumok, különös képen a szerződések, a mai napig ellenállnak ennek a tendenciának, és az esetek jelentős részében továbbra is nyomtatott vagy kézzel írott formában keletkeznek. Ezek kezelése és nyilvántartása – főleg egy nagyobb cég esetében – nem kis erőfeszítéseket igényel. Gondoljunk bele, például mennyivel könnyebb lenne egy adatbázisban, egyetlen gombnyomással megkeresni a minket érdeklő dokumentumot, mint egy akár több ezer iratot tartalmazó irattárban.

Szakdolgozatom célja megoldást nyújtani a papír alapú dokumentumok, azon belül is első sorban a szerződések nehézkes nyilvántartásának problémájára. Az általam készített szoftver egy felhasználóbarát grafikus felületen keresztül teszi lehetővé a szerződések kezelését. A nyilvántartó rendszer nem csupán a hétköznapi tevékenységeket (nyilvántartásba vétel, keresés)

teszi gyorsabbá és egyszerűbbé, hanem egyéb funkciókkal igyekszik megkönnyíteni és rugalmasabbá tenni a szerződésekben érintett felek munkáját. Lehetőségünk van például dátumhoz kötött értesítések megadására, hogy biztosan ne felejtkezzünk meg egy fontos határidőről sem, illetve, ezáltal értesülhetünk egy szerződés érvényességének kezdetéről és végéről. Lehetőségünk van tetszőleges típusú fájlok csatolására a szerződésekhez, így például digitális fényképeket vagy elektronikus dokumentumokat tudunk tárolni. Mivel a szerződések általában meglehetősen érzékeny, esetleg titkos információkat tartalmaznak, nagy hangsúlyt kell fektetni a biztonságra. Emiatt a rendszerbe kizárólag érvényes felhasználói névvel és a hozzá tartozó jelszóval lehet belépni, továbbá nagyon fontos, hogy egy szerződést csak az arra jogosult felhasználók láthassanak.

Tehát összefoglalva három fő szempont: a kényelem, a teljesítmény és a biztonság maximális figyelembe vételével terveztem és implementáltam a témám címében szereplő szerződés nyilvántartó rendszert.

2. Alkalmazott technológiák

A rendszer implementálásához a Java programozási nyelvet és több, a Java nyelvhez kötődő technológiát használtam fel, melyeket az alábbiakban részletesen bemutatok. Jelen dokumentum méretbeli korlátai nem engedik meg, hogy minden technológiát részletekbe menőkig bemutassak, így csak azon tulajdonságaikat ismertetem, amelyek miatt rájuk esett a választásom a szakdolgozatom készítéséhez.

2.1. Java

A Java egy objektum orientált programozási nyelv, melyet a Sun Microsystems fejlesztett ki a 90-es évek elején, és napjainkban is folyamatosan fejlődik. A Java-ban írt programot a fordító bytecode-ra fordítja. Mivel a bytecode platform-független, ezáltal a Java program tökéletesen megfelel a hordozhatóság követelményeinek. A bytecode futtatását a JVM (Java Virtual Machine – Java Virtuális Gép) végzi az aktuális operációs rendszeren.

A Java nyelvnek 3 fő típusa van: Java SE (Standard Edition – alap változat), Java EE (Enterprise Edition – nagyvállalati változat) és Java ME (Micro Edition – mobil eszközökre szánt változat). Ezek közül én a Java SE és Java EE változatokat használtam fel a rendszer implementálásához. [1]

Java Standard Edition: A Java nyelv standard változata tökéletesen alkalmas úgynevezett asztali alkalmazások (Desktop Application) készítésére. Objektumok széles palettáját biztosítja többek között a grafikus felhasználói felületek implementálásához, és támogatja a többnyelvű szoftverek egyszerű megírását. A Java SE-t a szerződés nyilvántartó rendszer kliens oldali részének elkészítéséhez használtam fel.

Java Enterprise Edition: A Java nyelv nagyvállalati változata egy széles körben elterjedt szerveroldali programozási platform. A standard változathoz képes kibővített programkönyvtárat (API) tartalmaz. Az alkalmazásszervereken (Application Server) futó moduláris szoftverkomponensek elősegítik a hibatűrő, többrétegű, elosztott alkalmazások készítését. [2]

Néhány fontosabb tulajdonsága, melyek kimondottan alkalmassá teszik arra, hogy a szerződés nyilvántartó rendszer alapjául szolgáljon:

- EJB (Enterprise JavaBean) komponensek futtatása menedzselt környezetben.
- Adatbázis hozzáférés menedzselt környezetből perzisztencia rétegen keresztül JDBC (Java Database Connectivity) segítségével.
- Teljes körű Web Service támogatás.

2.2. GlassFish

A GlassFish egy Java EE specifikációnak megfelelő alkalmazásszerver implementáció, amit a Sun Microsystems fejleszt. Ismeretes még Sun Java System Application Server néven is. Alapjául a Sun által kifejlesztett szoftver és az Oracle TopLink nevű perzisztencia-kezelő rendszere (JPA) szolgál, valamint a web tartalmak kezelésére szolgáló servlet container, amit az Apache Tomcat-ből származtattak, majd kiegészítették a Grizzly nevű komponenssel a skálázhatóság és nagyobb sebesség érdekében.[6]

2.3. Apache Ant

Az Apache Ant egy nyílt forráskódú szoftver, melyet az Apache Software Foundation fejlesztett ki. Segítségével automatizálhatjuk a java nyelven írt programok build folyamatát. A build folyamat leírását egy *build.xml* nevű fájl segítségével tudjuk definiálni, amelynek nevéből is látszik, hogy XML struktúrát alkalmaz.[3]

A szerződés nyilvántartó rendszer esetében a következő feladatokat végzi az Ant build rendszer:

- Forrás állományok fordítása.
- JAR (Java Archive) csomagok elkészítése.
- Deploy: szerveroldali komponensek telepítése az alkalmazásszerverben
- Adatbázis kapcsolat beállítása az alkalmazásszerverben

2.4. Web Service

A web service magyarul web szolgáltatás alkalmazások hálózaton keresztüli adatcseréjét valósítja meg HTTP protokollon keresztül. A kommunikáció alapjául XML dokumentumok szolgálnak. A web szolgáltatás rendszerint kétszereplős, van egy kiszolgáló (Provider) és egy igénylő (Requestor) fél. Egy kiszolgáló természetesen több különböző igénylőtől érkező igényt is ki tud szolgálni. Egy web szolgáltatást egyértelműen definiál az azt leíró WSDL (Web Service Descriptor Language) állomány, amely szintén XML formátumú. Bár a kommunikáció XML formátumú üzenetek továbbításán alapszik, a technológia segítségével lehetőségünk van akár bináris objektumok továbbítására is. [4]

2.5. PostgreSQL

A PostgreSQL vagy röviden Postgres egy nyílt forráskódú, ingyenes, relációs adatbázis-kezelő rendszer. Első verziója 1985-ben látott napvilágot a Kaliforniai Berkeley Egyetem gondozásában.

Tudását tekintve megközelíti a nagynevű adatbázis-kezelő rendszereket (Oracle, MSSQL) ám azoknál jóval kisebb, rugalmasabb, könnyebben kezelhető és erősen támogatott a

JDBC-n keresztüli hozzáférés, ami miatt különösen jó választás lehet közepes méretű Java alkalmazások esetén.[5]

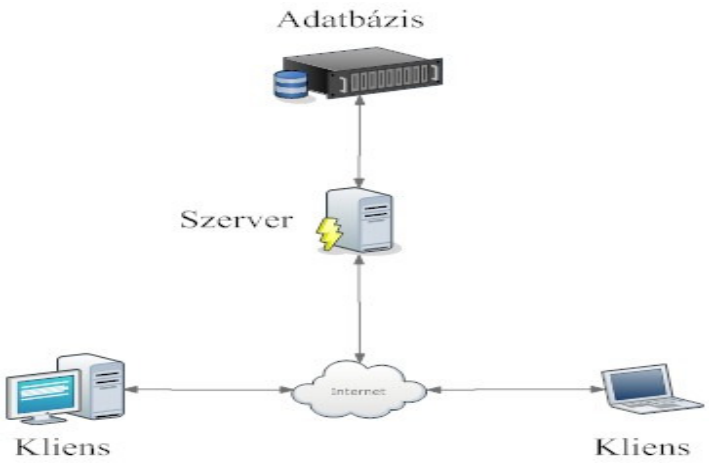
3. A szoftver felépítése

A szerződés nyilvántartó rendszer alapvetően egy szerver-kliens alapú rendszer. Felépítését tekintve megfelel a 3 rétegű MVC (Model View Control) architektúrának.

A **Model** réteget a PostgreSQL relációs adatbázis-kezelő rendszer alkotja. Ez a réteg kizárólag az adatok tárolásával foglalkozik.

A **Control** réteget a szerveroldali komponensek alkotják. Ide tartoznak az EJB komponensek, a web szolgáltatások és az adatbázis kapcsolatot biztosító alrendszer, melyen az alkalmazás szerver szolgáltat. Minden üzleti logika kizárólag ebben a rétegben található. A kapcsolatot az Model réteg felé JDBC kapcsolat biztosítja.

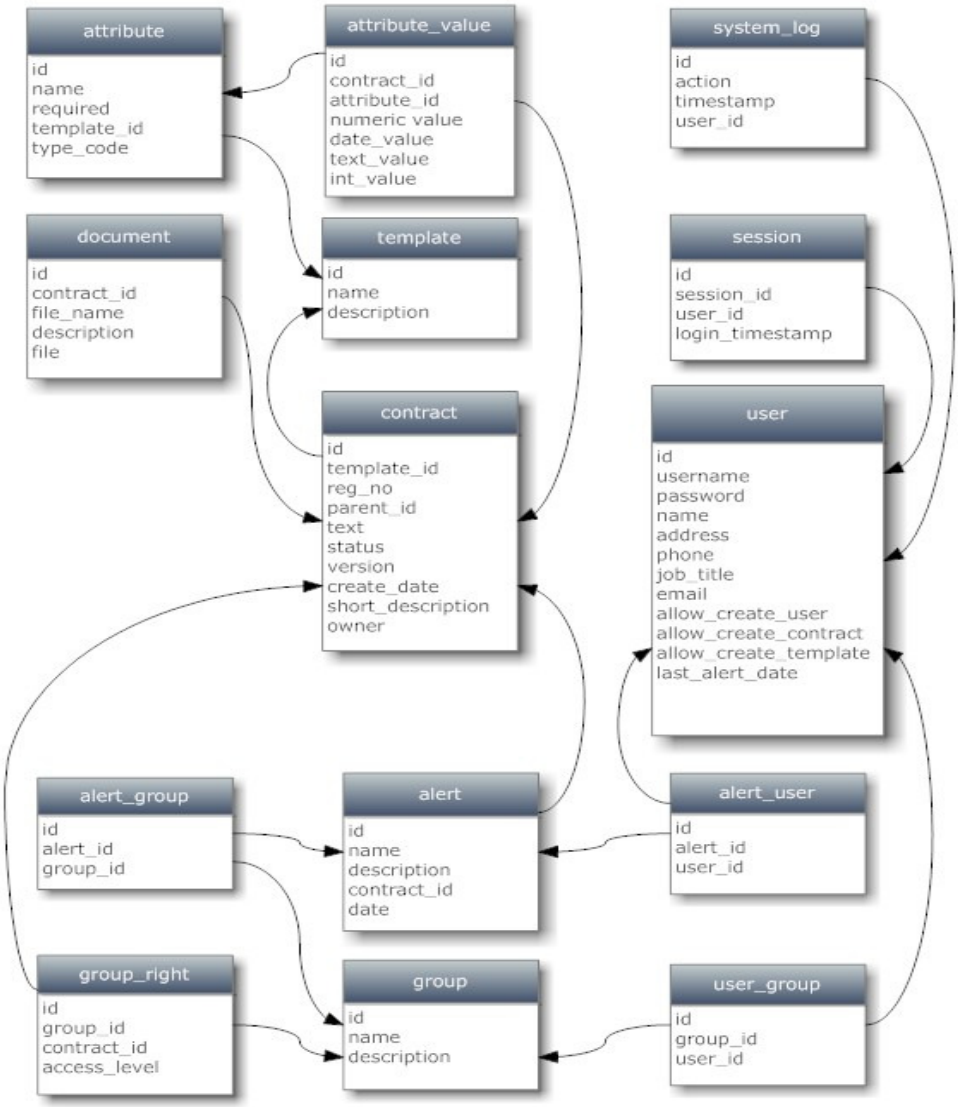
A **View** réteg maga a kliens szoftver. Ez egy úgynevezett vastag kliens, azzal a kitétellel, hogy nem tartalmaz üzleti logikát. Feladata kizárólag a felhasználói igények továbbítása a Control réteg felé, és az onnan érkező információk vizuális megjelenítése a felhasználó számára.



1. ábra: Magas szintű architektúra

3.1. Adatbázis struktúra

A következő fejezetben részletekbe menően ismertetni fogom az alkalmazás adatbázis rétegének felépítését.



2. ábra: Adatbázis architektúra

Az adatbázis felépítése úgy lett megtervezve, hogy a lehető legnagyobb rugalmasságot biztosítsa a rendszer számára. Szinte minden, a program által kezelt, szerződésekhez köthető entitás dinamikus szerkezettel rendelkezik, és adatbázisban tárolódik. Legjobb példa erre, a szerződés sablonok (*template*), melyek teljesen egyéni típusú és számú attribútum listával rendelkeznek. A felhasználók tetszőlegesen csoportokba szervezhetőek, egy felhasználó több csoportnak is tagja lehet.

Az adatbázis minden táblája rendelkezik elsődleges kulccsal, amit minden esetben az *id* nevű oszlop tartalmaz. Az azonosítók értéke szekvenciából generálódik, minden táblának saját szekvenciája van definiálva erre a célra. A táblák közötti kapcsolatot külső kulcsok definiálják, ezzel is biztosítva az adatbázis konzisztenciájának megőrzését.

- **contract** tábla: Az adatbázis legfontosabb táblája, ebbe kerülnek az adatbázisban tárol szerződések elsődleges adatai: szeződés száma (*reg_no*), rövid leírás (*short_description*), a szerződés szöveges tartalma (*text*), státusza (*status*), létrehozásának dátuma (*create_date*), létrehozója (*owner*) és típusának azonosítója (*template_id*). Továbbá tartalmaz két oszlopot, amelyek a rendszer jelenlegi verziójában nem kerültek felhasználásra, csupán azért kerülte bele a tábla-struktúrába, hogy a szoftver jövőbeni fejlesztése esetén az adatbázis modellhez már ne kelljen hozzányúlni. Név szerint ezek a *parent_id* és a *version*. A tervek szerint a *parent_id* használatával a szerződések hierarchikus struktúrába lesznek szervezhetőek, így lehetőség lesz szülő-gyermek viszony leírására, amellyel például egy szerződésmódosítást lehet felvenni az eredeti szerződés alá. A *version* oszlop segítségével verziózásra volna lehetőség. Használatával lehetőség nyílna a változások követésére, és a korábbi verziók visszakeresésére.

- **template** tábla: A szerződés sablonok tárolására szolgál. Maga a tábla csak az azonosítót (*id*), a sablon nevét (*name*) és leírását (*description*) tartalmazza, a hozzá tartozó attribútumok az *attribute* táblában találhatóak.

- **attribute** tábla: Itt kerülnek tárolásra a sablonokhoz (*template*) tartozó attribútumok, azaz egy adott szerződés tulajdonságai. Ilyen tulajdonságok lehetnek például a szerződő felek nevei, határidők, helyszínek, stb.

- **attribute_value** tábla: Egy konkrét szerződés példány esetén ebbe a táblába kerülnek az attribútumok értékei. A rendszer összesen négy különböző típusú attribútumot kezel, ezek név szerint az egész szám (*int*), tört szám (*numeric*), szöveg (*text*) és dátum (*date*). Minden típus számára külön oszlop található a táblában, adott típusú attribútum érték esetén az érték a megfelelő oszlopba kerül, míg a rekord többi érték-oszlopa *NULL* értéket tartalmaz.

- **document** tábla: Ebben a táblában a szerződésekhez csatolt bináris állományok kerülnek tárolásra, például egy papír alapú szerződés digitalizált változata, fényképek, hanganyagok, szöveges dokumentumok, stb.

- **alert** tábla: A szerződésekhez definiált értesítések tárolására szolgáló tábla. Tartalmazza az értesítés nevét (*name*), leírását (*description*), a szerződés azonosítóját (*contract_id*) és az értesítés dátumát (*date*).

- **alert_group** tábla: Kapcsoló tábla, az *alert* és a *group* táblák közötti *n:m* kapcsolatot valósítja meg. Lényegében a tábla tartalma mutatja meg, hogy egy értesítést mely felhasználói csoportoknak kell megjeleníteni.

- **alert_user** tábla: Kapcsoló tábla, az *alert* és a *user* táblák közötti *n:m* kapcsolatot valósítja meg. A tábla tartalma mutatja meg, hogy egy értesítést mely felhasználóknak kell megjeleníteni.

- **group** tábla: Felhasználói csoportokat reprezentáló tábla. Tartalmazza a csoportok nevét (*name*) és leírását (*description*), a csoportba tartozó felhasználók egy kapcsolótábla segítségével vannak azonosítva.

- **group_right** tábla: *n:m* kapcsolatot leíró kapcsoló tábla a *group* és a *contract* táblák között, megmutatja, hogy egy szerződéshez melyik felhasználói csoport

tagjainak vannak jogosultságai. A tábla *access_level* nevű oszlopa pedig megmondja, hogy milyen típusú jogosultságról van szó. A rendszer jelenleg írási és olvasási jogot kezel, de ez a tárolási mód lehetővé teszi, hogy a szoftver későbbi fejlesztése során újabb jogosultsági szinteket vezessünk be, az adatbázis struktúra módosítása nélkül.

- **user_group** tábla: Kapcsoló tábla, a *group* és a *user* táblák közötti *n:m* kapcsolat megvalósítását szolgálja. Megmutatja, hogy melyik felhasználói csoportokba melyik felhasználók tartoznak és fordítva.

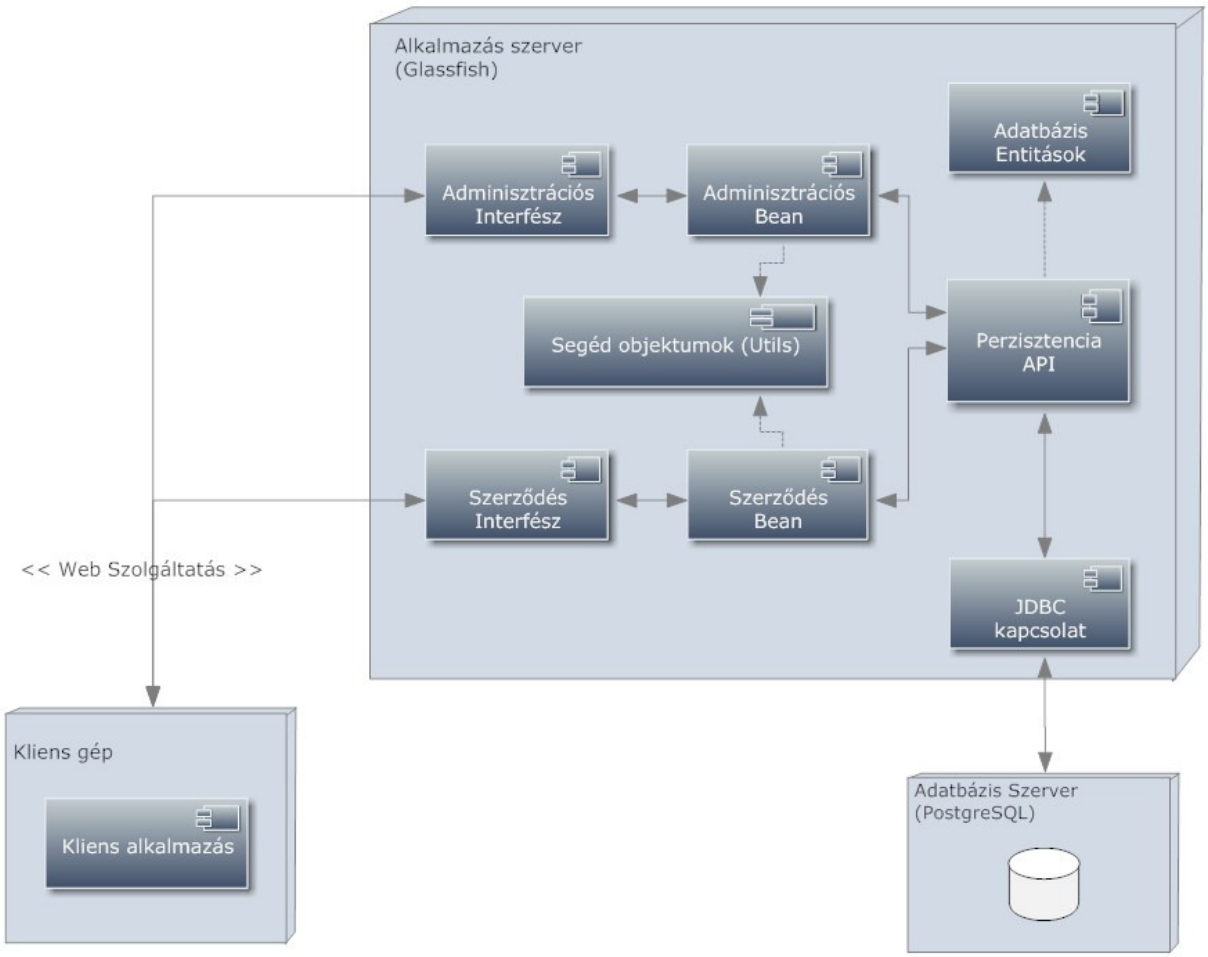
- **user** tábla: Szintén egy nagyon fontos tábla, a felhasználók adatai kerülnek benne tárolásra. A szerződések után a felhasználók a második legfontosabb entitások a rendszer működésének szempontjából. Tartalmazza a belépéshez szükséges felhasználói nevet (*username*), a jelszó titkosított változatát (*password*), a felhasználó személyes adatait (*name, address, phone, job_title, email*), a felhasználó értesítéseinek utolsó dátumát. Továbbá tartalmaz különböző adminisztratív jogosultságokat, amelyek megmutatják, hogy a felhasználónak van-e hozzáférése a felhasználó menedzsmenthez (*allow_create_user*), szerződés létrehozásához (*allow_create_contract*) és a sablonok kezeléséhez (*allow_create_template*).

- **session** tábla: Munkament információkat tároló tábla. A felhasználó bejelentkezésekor egy egyedi munkamenet azonosító generálódik (*session_id*), a felhasználó a rendszer használata során ezzel azonosítja magát.

- **system_log** tábla: Ebben a táblában tárolódnak a rendszernapló bejegyzései. Minden felhasználói tevékenységről automatikusan generálódik egy bejegyzés a naplóba.

3.2. Szerveroldali alkalmazás

Ahogy azt már említettem, a rendszer szerver oldali része egy Java Enterprise Edition alapokon nyugvó alkalmazás, amely Java alkalmazás szerver alatt fut. Én a fejlesztés során a Sun Microsystems által készített Glassfish nevű implementációt használtam, azonban bármely más gyártó termékén gond nélkül kell futnia, amely kompatibilis a Java EE specifikációval.



3. ábra: Szerver oldali komponensek

A szerver oldalon elhelyezkedő komponensek három fő rétegre oszthatóak:

1. **Kliens kapcsolati** réteg: Idetartozik az a két osztály, amelyeknek a metódusait a kliensalkalmazás web szolgáltatásokon keresztül tud igénybe venni. Minden egyes metódusnak meg van feleltetve a web szolgáltatás egy-egy művelete (web service operation). Az így előálló két web szolgáltatást az őket leíró WSDL dokumentum egyértelműen definiálja. A kliens és a szerver között szinkron kommunikáció folyik, azaz a kliens mindig megvárja, hogy a kezdeményezett művelet végrehajtódjon, és az így előállt eredmény birtokában tér vissza.

2. **Üzleti logikai** réteg: Ezt a réteget állapot nélküli (stateless) session bean-ek alkotják, melyek az alkalmazás hátterében megbúvó tényleges üzleti logikáért felelősek. Az, hogy állapotmentesek, annyit jelent, hogy a Bean-eknek nincs belső állapotuk, azaz nem tárolnak példány szintű adattagokban olyan információkat, amelyekre több művelet végrehajtása közben is szükség lenne, kizárólag a paraméterben kapott, és adatbázisból kinyert adatokon operálnak. Ide futnak be a klienstől érkező felhasználói igények, és itt történik az adatbázisból származó adatok feldolgozása.

3. **Adatbázis-kapcsolati** réteg: Az adatbázis eléréséhez szükséges komponenseket az alkalmazás szerver bocsátja a rendelkezésünkre. Az üzleti logikát megvalósító osztályok a Java perzisztencia API-n (JPA) keresztül férnek hozzá az adatbázisban tárolt adatokhoz. A JPA feladata, hogy az adatbázis műveleteket tényleges SQL utasításokká alakítsa, majd az eredmény képen előálló adatokat Java osztályokba „csomagolja”. A perzisztencia réteg JDBC kapcsolaton keresztül kapcsolódik az adatbázishoz.

A kliens kapcsolati réteg és az üzleti logikai réteg komponensei egy további szempont szerint is elválasztásra kerültek. Ez a szempont nem más, mint hogy az általa kezelt üzleti folyamatok adminisztratív jellegűek-e vagy a szerződések kezelésével kapcsolatosak. Ez a fajta csoportosítás több előnnyel is jár. Egyrészt, így sokkal átláthatóbbá válik a komponensek

feladatköre, ami nagyban elősegíti a karbantarthatóságot. Másrészt mivel az így elkülönített szoftverrészek teljesen különálló web szolgáltatás felülettel rendelkeznek, lehetőségünk van akár arra is, hogy a különböző feladatokat ellátó szoftverrészek két, fizikailag is különálló szerveren fussanak. Az alrendszerek fizikai elkülönítése hatékonyabbá teszi a szoftver működését a terhelés elosztása révén, illetve biztonságosabbá is teszi a rendszert, mert ha a két szerver közül bármelyik meghibásodik, akkor a másik gép azonnal átveheti annak helyét.

Most pedig felsorolás és rövid leírás szintjén bemutatom a két alrendszer által kezelt műveleteket, melyeket a kliens web szolgáltatás formájában tud elérni.

Az **Adminisztrációs Interfész** műveletei:

- **authenticate**(): a szerver azonosítja a felhasználót, és elküldi az egyedi munkamenet azonosítót (session id).
- **logOut**(): a rendszer kilépteti a felhasználót, és érvényteleníti a munkamenetet.
- **changePassword**(): felhasználó megváltoztathatja a bejelentkezéshez használt jelszavát.
- **getGroupList**(): felhasználói csoportok listájának lekérése
- **addGroup**(): új felhasználói csoport létrehozása
- **editGroup**(): meglévő felhasználói csoport módosítása
- **removeGroup**(): felhasználói csoport törlése
- **getUserList**(): felhasználók listájának lekérése
- **addUser**(): új felhasználó létrehozása
- **editUser**(): meglévő felhasználó adatainak módosítása
- **removeUser**(): felhasználó törlése
- **addUserToGroup**(): egy felhasználó hozzárendelése felhasználói csoporthoz

- **removeUserFromGroup**(): felhasználó eltávolítása egy felhasználói csoportból
- **getRights**(): adminisztrációs jogosultságok listájának lekérése
- **getSystemLogList**(): rendszer napló tartalmának lekérdezése

A **Szerződés Interfész** műveletei:

- **getTemplateList**(): sablonok listájának lekérdezése
- **addTemplate**(): új sablon létrehozása
- **editTemplate**(): meglévő sablon módosítása
- **removeTemplate**(): sablon törlése
- **addContract**(): új szerződés létrehozása
- **editContract**(): meglévő szerződés adatainak módosítása
- **removeContract**(): szerződés törlése
- **fileUpload**(): új állomány csatolása egy szerződéshez
- **editDocument**(): meglévő csatolt állomány adatainak szerkesztése
- **removeDocument**(): csatolt állomány törlése
- **downloadDocument**(): csatolt állomány bináris tartalmának letöltése a kliensre
- **searchContractByTemplate**(): szerződés keresése sablon alapján
- **searchContractByAttribute**(): szerződés keresése attribútum értékek alapján
- **getAlerts**(): olvasatlan értesítések listájának lekérdezése
- **dismissAlerts**(): értesítések olvasottként megjelölése

A beérkezett felhasználói igényeket a web szolgáltatások osztályai egyszerűen továbbítják a megfelelő EJB-nek. Az EJB példányok létrehozását az alkalmazás szerver EJB Container nevű alrendszere végzi automatikusan. Az így létrejött példányokhoz az @EJB annotáció segítségével lehet hozzáférni.

```
@EJB
private ContractHandlerLocal ejbRef;

@WebMethod(operationName = "editContract")
public boolean editContract(
            @WebParam(name = "sessionID")    String sessionID,
            @WebParam(name = "contract")     ClientContract contract
            ) throws CmanUnsuccessfulOperationException {
    return ejbRef.editContract(sessionID, contract);
}
```

4. ábra: Hozzáférés EJB példányhoz és Web Szolgáltatás művelet példa

Mindkét web szolgáltatás minden egyes művelete tartalmaz egy úgynevezett *SessionID* nevű paramétert, ami nem más, mint az egyedileg generált munkamenet azonosító. Ennek az azonosítónak segítségével azonosítható a műveletet kezdeményező kliens. Amikor a Bean függvényei meghívódnak, legelőször a munkamenet azonosító érvényességének ellenőrzése történik meg. Ezt az ellenőrzést egy úgynevezett Interceptor objektum végzi, melyet az osztálydefiníció előtt elhelyezett *@Interceptors(osztálynév_lista)* annotáció segítségével tudunk működésbe helyezni.

```
/**
 * Szerződésekkel kapcsolatos üzleti logikát implementáló osztály
 * @author Docsa Tamás
 */
@Stateless
@Interceptors({com.ccsoft.cman.util.security.SessionChecker.class})
public class ContractHandlerBean implements ContractHandlerLocal {
```

5. ábra: Interceptors annotáció alkalmazása

Amikor a Bean valamely metódusa meghívódik, akkor először az interceptor objectum megfelelően annotált ellenőrző függvénye eldönti, hogy a paraméterben kapott munkamenet azonosító érvényes-e.

```
@AroundInvoke
private Object check(InvocationContext invCtx) throws Exception {
    Object[] paramList = invCtx.getParameters();
    LoginService.checkSession((String) paramList[0], em);

    return invCtx.proceed();
}
```

6. ábra: Az interceptor objektum ellenőrző függvénye

Az interceptor-nak köszönhetően az azonosító vizsgálata automatikusan történik a Bean bármely függvénye esetén, nincs szükség semmilyen plusz kódra azokban. Amennyiben az ellenőrzés úgy találja, hogy az azonosító érvényes munkamenetet takar, akkor a vezérlés tovább adódik arra a metódusra, amelyet eredetileg meghívtak. Azonban ha az azonosító érvénytelen, akkor a kért művelet végrehajtása nem kezdődik el. Ennek az ellenőrzésnek segítségével meg tudjuk akadályozni, hogy illetéktelen személyek a kliens program megkerülésével jogosulatlanul férjenek hozzá a rendszerben tárolt adatokhoz.

A szerver oldali alkalmazás a végrehajtott műveletekről minden esetben egy bejegyzést készít a rendszer naplóba. Ezáltal másodpercre pontosan nyomon követhető, hogy ki, mikor és mit csinált a rendszerben. A rendszer napló bejegyzései a kliens program felületén megfelelő jogosultság mellett tetszőleges időszakra visszamenőleg megtekinthetőek. Ez a napló kizárólag a felhasználói tevékenységek ellenőrizhetővé tételét szolgálja, a rendszer működésére vonatkozó technikai adatokat nem tartalmaz. Erre a célra a szoftver a java.util.logging csomag Logger osztályát alkalmazza.

```
if (clientContract.id != -1) {
    entityContract = (Contract) em.createNamedQuery("Contract.findById").setParame
    User u = Util.getUserBySessionID(sessionID, em);
    Util.createLog(em, u, "Updating contract: "+clientContract.contNum);
    logger.info("Updating contract: ID = " + entityContract.getId());
} else {
    entityContract = new com.ccsoft.cman.entities.Contract();
    User u = Util.getUserBySessionID(sessionID, em);
    entityContract.setOwner(u);
    Util.createLog(em, u, "Creating new contract");
    logger.info("Inserting contract");
}
```

7. ábra: Példa naplózásra

Ezt a beépített naplózási szolgáltatást az alkalmazás szerver automatikusan konfigurálja, és tartalmát egyenesen a szerver naplóba irányítja. Ide már csak kizárólag hibakeresési szándékkal létrehozott, úgynevezett debug információk kerülnek. Ennek a naplónak a tartalma megtekinthető a Glassfish adminisztrációs felületén.

3.3. Kliens alkalmazás

3.3.1. Szerződés fogalma

Mindenekelőtt érdemes tisztázni, hogy mi is az a fogalom, amit a rendszer a szerződés szó alatt ért. Számunkra a szerződés egy összetett adathalmaz, amely rendelkezik egy egyedi azonosítóval, ez a szerződés szám, továbbá minden szerződés rendelkezik egy szöveges résszel és egy rövid leírása. A szöveges rész tartalmazza a szerződés teljes szöveganyagát, amely a papír alapú formán is olvasható. A rövid leírás egy olyan tömör kivonat, ami alapján azonosítani tudjuk az adott szerződést anélkül, hogy végig kellene olvasni annak teljes szövegét. A szerződések rendelkeznek úgynevezett attribútumokkal. Ezek a szerződést jellemző kulcsfontosságú információkat tartalmaznak. Lényegében véve, az attribútumok azok típussal ellátott név-érték párok. A rendszer négy különböző attribútum típust különböztet meg, ezek a szöveg, egész szám, tört szám és a dátum. Minden szerződésnek van egy sablonja. A sablon egy előre definiált attribútum gyűjtemény. Technikai szempontból maga a szerződés nem is tartalmaz attribútumokat, hanem a sablonja mondja meg, milyen attribútumokkal rendelkezik, és a szerződés csak a megfelelő attribútumok értékeit tartalmazza. Egy sablonnal általában egy jól definiált szerződés típust tudunk megadni. Például legyen egy „Munkaszerződés” nevű sablonunk. Ennek az attribútumai lehetnek többek között: munkáltató, munkavállaló, betöltött pozíció, munkaidő, fizetés, stb. Az így definiált sablonok, egy nagyon jól használható keresési lehetőséget adnak a felhasználó kezébe. Az előbbi példánál maradva, lehetőségünk van kikeresni az összes szerződést, ahol a munkáltató egy adott cég, vagy az egy adott pozíciót betöltő személyek szerződéseit.

Az eddig felsoroltak a szerződés azonosítására és tartalmára vonatkozó információkat tartalmaztak. Ezeken kívül lehetőségünk van fájlokat csatolni a szerződésekhez. Ez lehet a dokumentum lapolvasóval digitalizált változata, fénykép, hanganyag, vagy bármi más, ami valamilyen módon kapcsolódik a szerződésünkhöz.

A szerződésekben található esetleges bizalmas információk védelmét szolgálja a jogosultsági rendszer. Amikor létrehozunk egy szerződést, meg kell mondanunk, hogy mely felhasználói csoportok tagjainak és milyen típusú jogosultságok biztosítunk a szerződéshez. Ez lehet olvasási, és írási. Amennyiben egy felhasználót olvasási joggal ruházunk fel, ebben az esetben ő képes lesz megnyitni a szerződést, és elolvasni annak tartalmát, de módosításra nem lesz lehetősége. Ezzel szemben, ha írási jogot kap rá, akkor módosításokat tud végezni az adott szerződésen. Azt, hogy milyen mértékű változtatás engedélyezett a szerződésen, azt a szerződés egy további jellemzője, az állapota határozza meg. Egy szerződés állapota lehet nyitott, vagy zárt. Ha egy szerződés állapota nyitott, azzal azt jelezzük, hogy a szerződés még nem végleges ezért a szerződés számon és a sablonon kívül bármit módosíthatunk. Zárt állapotú szerződés esetén azt feltételezzük, hogy az már elérte végleges formáját és tartalmát, azért nem módosíthatunk semmit, aminek köze lenne a tényleges tartalomhoz, úgymint rövid leírás, szöveges rész, és attribútum értékek. Azonban a járulékos információk: csatolt dokumentumok, jogosultságok és értesítések továbbra is változtathatóak.

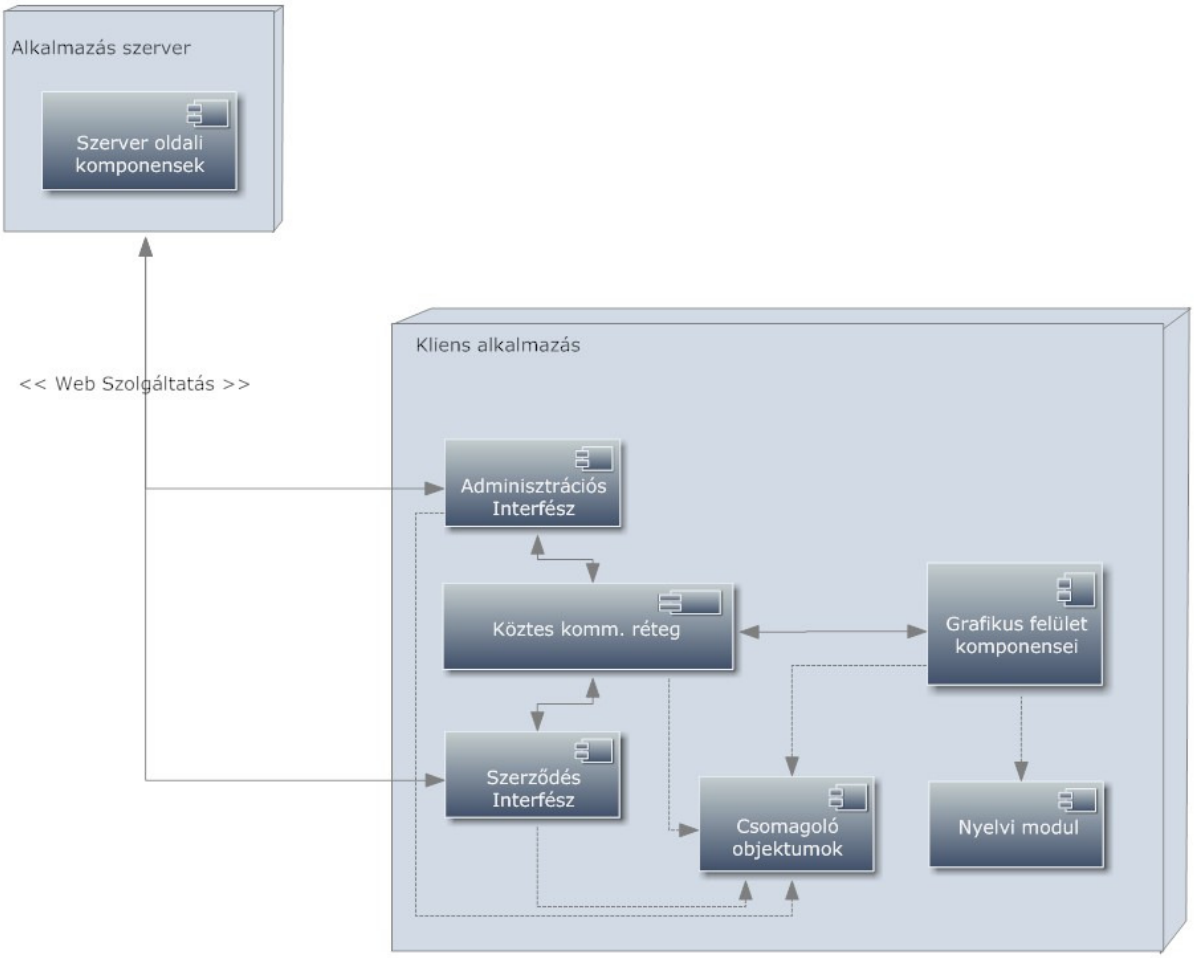
Egy további, ám nem kevésbé fontos tulajdonságról nem esett még szó, és ez az értesítések hozzárendelése. Az értesítések lényegében emlékeztető üzenetek, amelyekkel felhívhatjuk saját, vagy más felhasználók figyelmét bizonyos, a szerződés szempontjából fontos időpontokra. Ilyen lehet például a szerződés érvényességének kezdete és vége. Amikor hozzárendelünk egy értesítést a szerződésünkhöz, megadhatjuk, hogy kik azok a felhasználók, esetleg felhasználói csoportok, akiket értesíteni szeretnénk, melyik napra legyen beütemezve az értesítés, és milyen üzenet jelenjen meg az értesítendő felhasználóknak.

3.3.2. Architektúra

A teljes rendszert nézve, a kliensalkalmazás a prezentációs réteget alkotja, melynek alapvető célja a rendszerben tárolt adatok megjelenítése és felhasználói igények továbbítása a szerver oldali logikai réteg felé. Azonban, ha mint különálló programegységet nézzük,

elmondható róla, hogy önmaga is egy háromrétegű architektúra mentén épül fel. Az alkalmazás rétegei:

- Grafikai réteg
- Köztes kommunikációs réteg
- Szerver-kapcsolati réteg



8. ábra: Kliens architektúra

A **szerver-kapcsolati réteg** feladata – ahogy azt neve is mutatja – a szerverrel való kapcsolattartás. Ahogy azt már a szerver oldali alkalmazásnál láthattuk, a web szolgáltatások két jól elkülönített csoportra vannak osztva. Egyik csoportban az adminisztratív funkciók, a másikban pedig a szerződésekkel kapcsolatos műveletek találhatóak. Ennek megfelelően a szerver-kapcsolati réteget két komponens alkotja: az adminisztrációs interfész, és a szerződés interfész. Ez a két komponens lényegében függvények gyűjteménye, a függvények által lefedett funkcionalitás pedig egy az egyben megegyezik a komponens mögött álló web szolgáltatás műveleteivel. Amikor a felsőbb rétegben elhelyezkedő komponens igénybe szeretne venni valamilyen szerver oldali szolgáltatást, nincs más dolga, mint meghívni funkcionalitásért felelős komponens megfelelő metódusát. Ekkor a szerver-kapcsolati komponens először konvertálja paraméterül kapott adatokat olyan formátumra, amelyek már elküldhetőek a web szolgáltatáson keresztül. A szervertől érkező válasz alapján úgynevezett csomagoló objektumokat hoz létre, és ezzel válaszolja meg felsőbb szintről érkező kérdést.

A **köztes kommunikációs réteg** összekapcsolja az adminisztrációs és a szerződés interfész funkcionalitását, és egy egységes felületet biztosít a legfelső szintet alkotó grafikai komponensek felé. Ebben a rétegben történik a munkamenet kezelés, itt tárolódik a bejelentkezéskor kapott egyedi munkamenet azonosító. Továbbá a réteg fontos része az úgynevezett *cache*-elési funkció. Ennek lényege, hogy a leggyakrabban használt lekérdező műveleteket (felhasználók listája, felhasználói csoportok listája, szerződés sablonok listája) az első kéréskor végrehajtja, az eredményt memóriában tárolja, és az ezután érkező kéréseket ebből az átmeneti tárból válaszolja meg. Ezáltal lerövidül a komponensek inicializációs ideje, és jelentős terhet veszünk le a szerverről. A különböző grafikai komponensek figyelőként (*listener*) regisztrálhatják magukat azokhoz az ideiglenes tárban elhelyezett objektumokhoz, amelyek az ő működésükhöz szükségesek, ekkor, ha valamely művelet eredménye képen változás következik be az átmeneti tárban, akkor az így beregisztrált figyelő komponensek értesítést kapnak a bekövetkezett változásról. Ennek köszönhetően a felületen látható adatok nem lesznek elavultak vagy érvénytelenek.

A **grafikai réteget** a tényleges megjelenítést szolgáló komponensek. A felhasználói felület elkészítéséhez a javax.swing csomag osztályait használtam fel. Az alap grafikai elemeken

kívül több saját komponenst is használok felületen, melyeket valamely swing-es osztályból származtattam. Logikailag még ehhez a réteghez sorolható, a többnyelvűséget támogató komponens.

3.3.3. Többnyelvűség

A kliens szoftver fontos tulajdonsága, hogy támogatja a többnyelvű felhasználást. A rendszer értelmez egy darab aktuális nyelvet, és egy alapértelmezett nyelvet. Az alapértelmezett nyelv az angol. Az aktuális nyelvet a felhasználó a bejelentkezéskor tudja kiválasztani, bejelentkezést követően a nyelv változtatására nincs lehetőség. Szöveges elemek megjelenítésekor a grafikai komponens elkéri a nyelvi modultól az oda tartozó szöveg megfelelő fordítását. Amennyiben az aktuális nyelven elérhető a szöveg, akkor a nyelvi modul azzal megválaszolja a kérést, ellenkező esetben az alapértelmezett nyelven vett fordítást adja válaszul. Amennyiben egyik nyelven sem elérhető a kért szöveg, akkor alapértelmezés szerint az "<Unknown>" szöveg kerül megjelenítésre.

Az egyes nyelvek szerinti programfordítások nyelvi fájlokban vannak tárolva. Minden nyelv külön fájlban van tárolva. Ezek a nyelvi fájlok úgynevezett *property* fájlok. A property fájlok egyszerű szöveges fájlok, amelyek soronként kulcs-érték párt tartalmaznak *kulcs=érték* formában. Esetünkben a kulcs egy azonosító, amely egyértelműen azonosítja a program valamely szöveges elemét, az érték pedig az adott nyelv szerinti fordítás. A nyelvi fájlok nevének szerkezete: Language_<*nyelviKód*>.property ahol a nyelvi kód a szabványos két karakteres jelölése a nyelvnek (például magyar:hu, angol:en, német: de).

```
CONTRACT_ATTRIBUTE_VALUE_ERROR_FLOAT=Hibás formátum, tört szám megadása kötelező!
CONTRACT_INFO=Info
CONTRACT_INFO_OWNER=Tulajdonos
CONTRACT_INFO_CREATE=Létrehozva
CONTRACT_SAVE_SUCCESFULL=A szerződés mentése sikeres.
CONTRACT_SAVE_WARNING_TITLE=Folytatja?
```

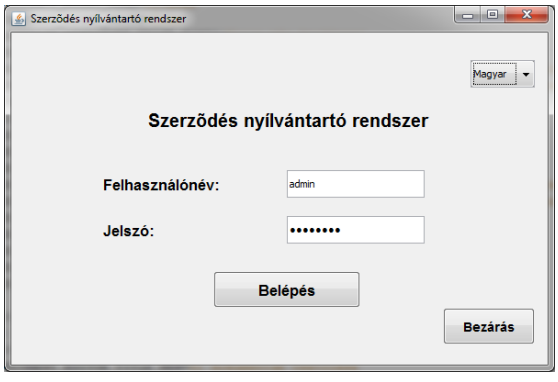
9. ábra: Magyar nyelvi fájl részlet

```
CONTRACT_ATTRIBUTE_VALUE_ERROR_FLOAT=Wrong format, float number required!
CONTRACT_INFO=Info
CONTRACT_INFO_OWNER=Owner
CONTRACT_INFO_CREATE=Created
CONTRACT_SAVE_SUCCESFULL=Contract saved successfully.
CONTRACT_SAVE_WARNING_TITLE=Continue?
```

10. ábra: Angol nyelvi fájl részlet

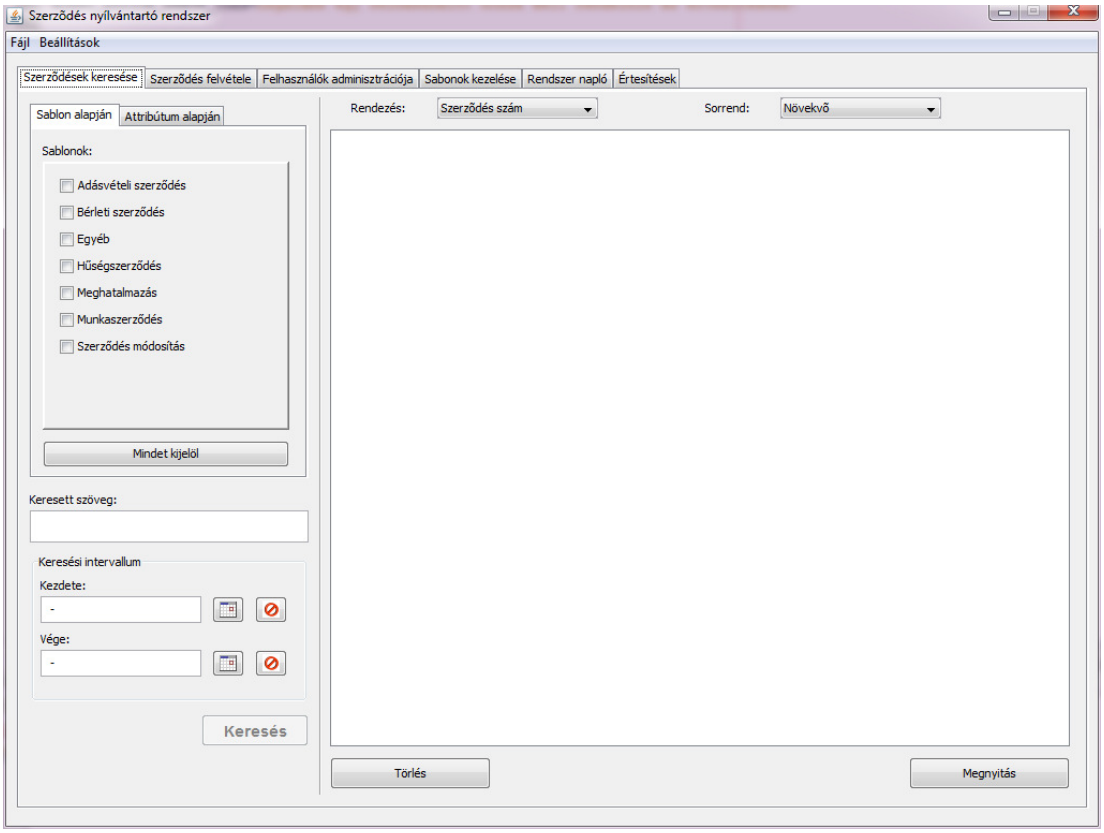
3.3.4. Felhasználói felület

A program indításakor először a bejelentkező képernyő fogadja a felhasználót. Itt kell megadni a bejelentkezéshez szükséges felhasználói nevet és jelszót. Továbbá itt van lehetőségünk kiválasztani, milyen nyelven szeretnénk használni a rendszert.



11. ábra: Bejelentkező képernyő

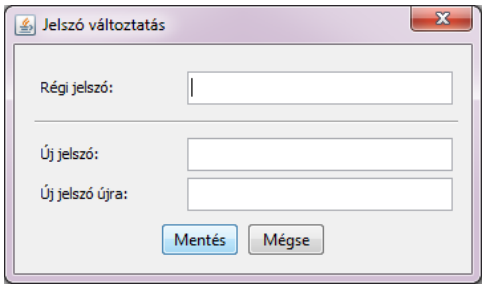
A belépés gombra kattintás után a rendszer belépteti a felhasználót, és sikeres belépés esetén megnyílik a program fő képernyője.



12. ábra: Fő képernyő

A fő képernyő 1024x768 felbontásra van optimalizálva, így a viszonylag régebbi monitorokon is kényelmesen elférnek a megjelenő információk. Az ablak legtetején egy menü sor található. Két fő menü van, a Fájl és a Beállítások. A fájl menüből tudjuk elérni a kilépés funkciót. A program bezárása kizárólag ezen a menüponton keresztül lehetséges. Ez azért fontos, hogy a program szabályosan ki tudja léptetni a felhasználót a rendszerből. A beállítások menüben találjuk a jelszó változtatás menüpontot, aminek segítségével a felhasználó megváltoztathatja a

belépéshez szükséges jelszavát. A jelszó megváltoztatásához mindenképpen szükséges a régi jelszó megadása.



13. ábra: Jelszóváltoztatás ablak

Közvetlenül a menü sor alatt láthatóak a program által nyújtott funkcionalitásokat reprezentáló fülek. A főablakban mindig az éppen kiválasztott fülnek megfelelő tartalom látható. Összesen legfeljebb 6 fül, azaz 6 fő funkciót kínál a felület a felhasználónak, azonban ez lehet kevesebb is, a felhasználó jogosultságaitól függően. A jogosultságok, amiket a felhasználók birtokolhatnak, a következőek:

- Új szerződés felvétele
- Sablonok kezelése
- Felhasználók adminisztrációja

Az **Új szerződés felvétele** jogosultság birtokában a felhasználó számára elérhetővé válik a *Szerződés felvétele* fül és ezáltal a lehetőség, hogy új szerződéseket hozzon létre a rendszerben.

A **Sablonok kezelése** joggal rendelkező felhasználók láthatják a vele megegyező nevű fület, ahol szerződés sablonok létrehozása, módosítása és törlése lehetséges.

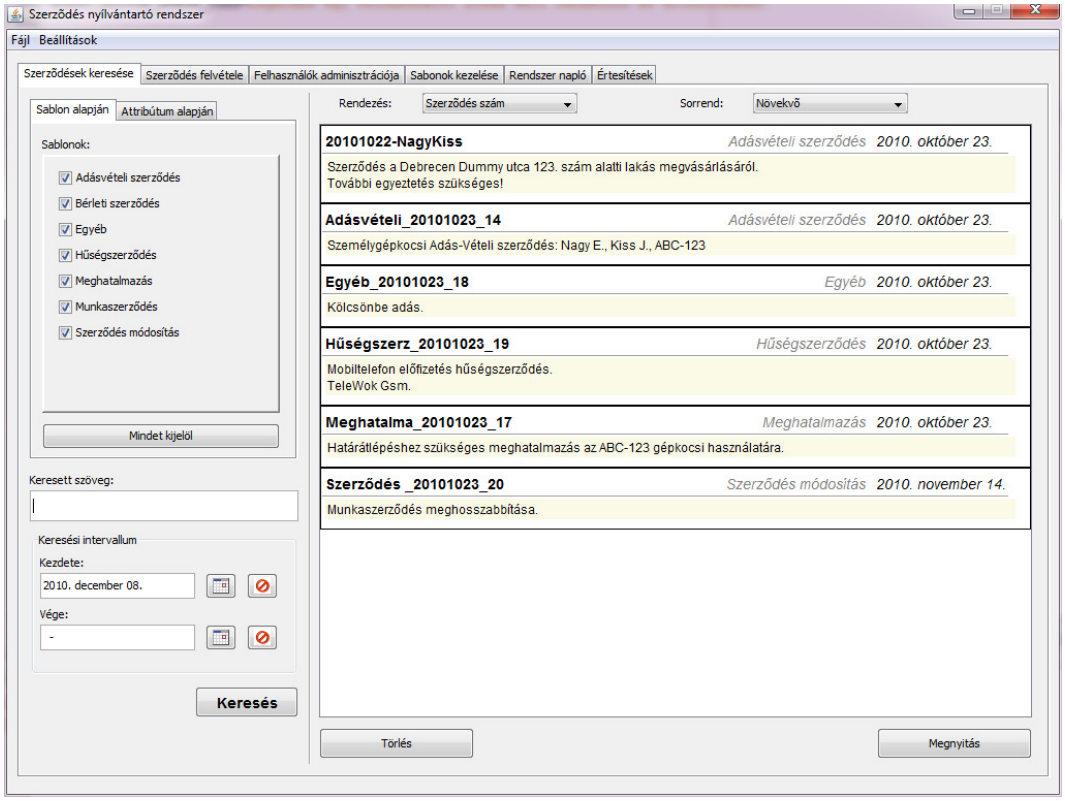
A harmadik jogosultság, a **Felhasználók adminisztrációja**, ami talán az egyik legfontosabb privilégium, tipikusan rendszeradminisztrátorok számára. Ennek birtokában elérhetővé vállnak a *Felhasználók adminisztrációja* és a *Rendszer napló* fülek.

Az eddig említett négy fülön kívül két további funkció is van még, melyek minden felhasználó számára alanyi jogon elérhetőek, ezek a *Szerződések keresése* és az *Értesítések*. Mivel a szerződéseket egy másik, dokumentum szintű hozzáférési rendszer védi, ezért ezt semmilyen biztonsági problémát nem jelent.

Az alábbiakban részletesen is bemutatom a felsorolt füleken keresztül elérhető funkcionalitásokat.

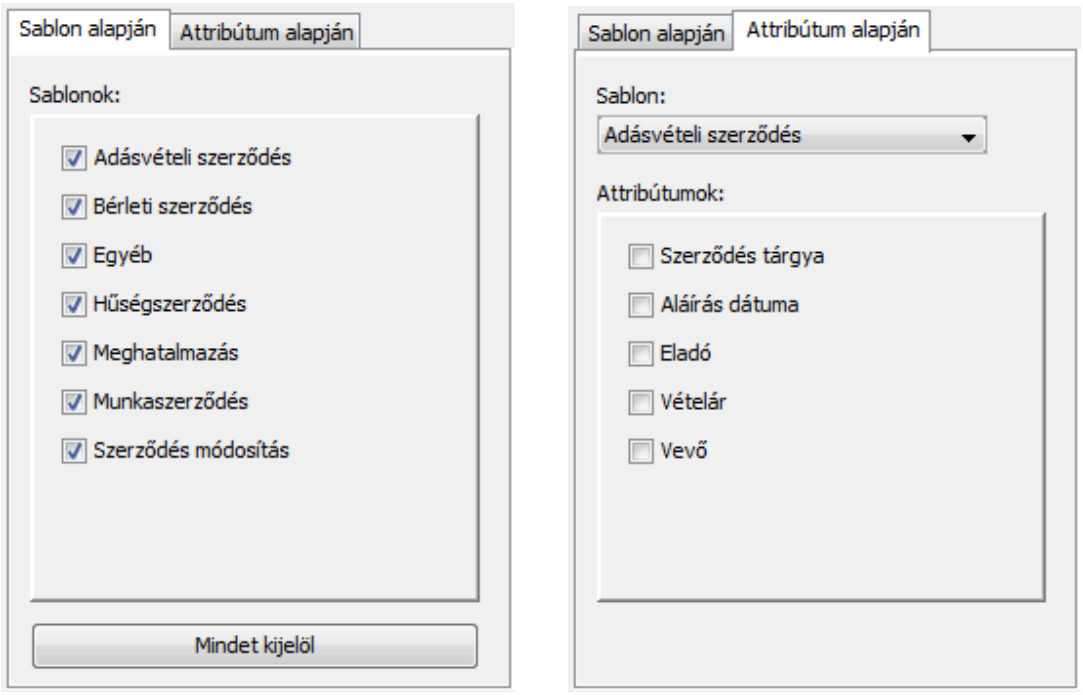
3.3.4.1. Szerződések keresése

Ez a program egyik legalapvetőbb funkciója. Segítségével visszakereshetjük és megnyithatjuk a rendszerben tárolt dokumentumokat.



14. ábra: Keresés képernyő

A képernyő két fő részből áll, baloldalon található a keresési kritériumokat tartalmazó panel, jobboldalon pedig a viszonylag nagyobb helyet elfoglaló eredmény lista. Alapvetően kétfajta keresési lehetőséget nyújt a rendszer: sablon alapú keresést és attribútum alapú keresést. Sablon alapú keresés esetén meg kell jelölnünk, mely sablon(ok)hoz tartozó dokumentumokat szeretnénk keresni. Legalább egy sablon megjelölése kötelező. Attribútum alapú keresés segítségével lehetőségünk van valamely sablon egy vagy több konkrét attribútumának tartalma alapján keresni.

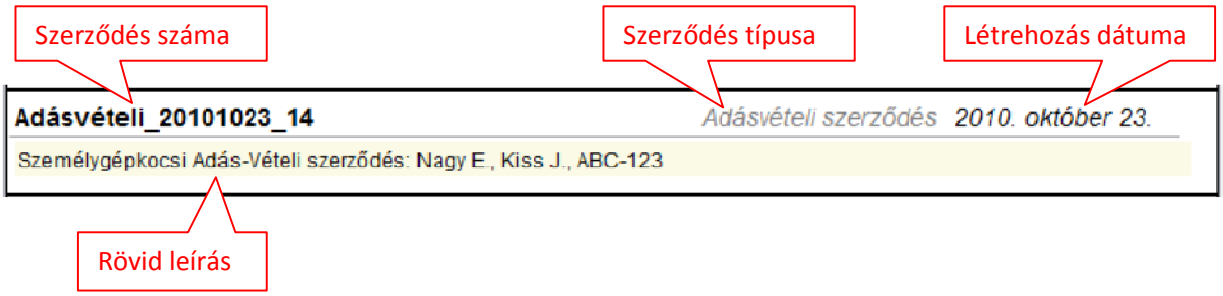


15. ábra: Sablon alapú és attribútum alapú keresés

Mindkét esetben lehetőségünk van megadni a keresett szöveget, valamint két dátumot, amelyek segítségével egy keresési időintervallumot tudunk meghatározni. Amennyiben a keresési intervallum alsó vagy felső határát üresen hagyjuk, úgy kereshetünk egy adott időpont előtt vagy után létrejött dokumentumok között. Ha egyik dátumot sem töltjük ki, akkor rendszer nem veszi

figyelembe a létrehozási dátumot a keresés során. A keresés eredményében kizárólag olyan szerződések fognak megjelenni, amelyekhez legalább olvasási jogosultsággal rendelkezünk.

Az keresés eredményeként a jobb oldali találati listában jelennek meg azok a szerződések, amelyek megfelelnek a keresési kritériumoknak. A listában a szerződéseknek egy összefoglalója látható, ez tartalmazza a szerződés számát, típusát, létrehozási dátumát és rövid leírását.



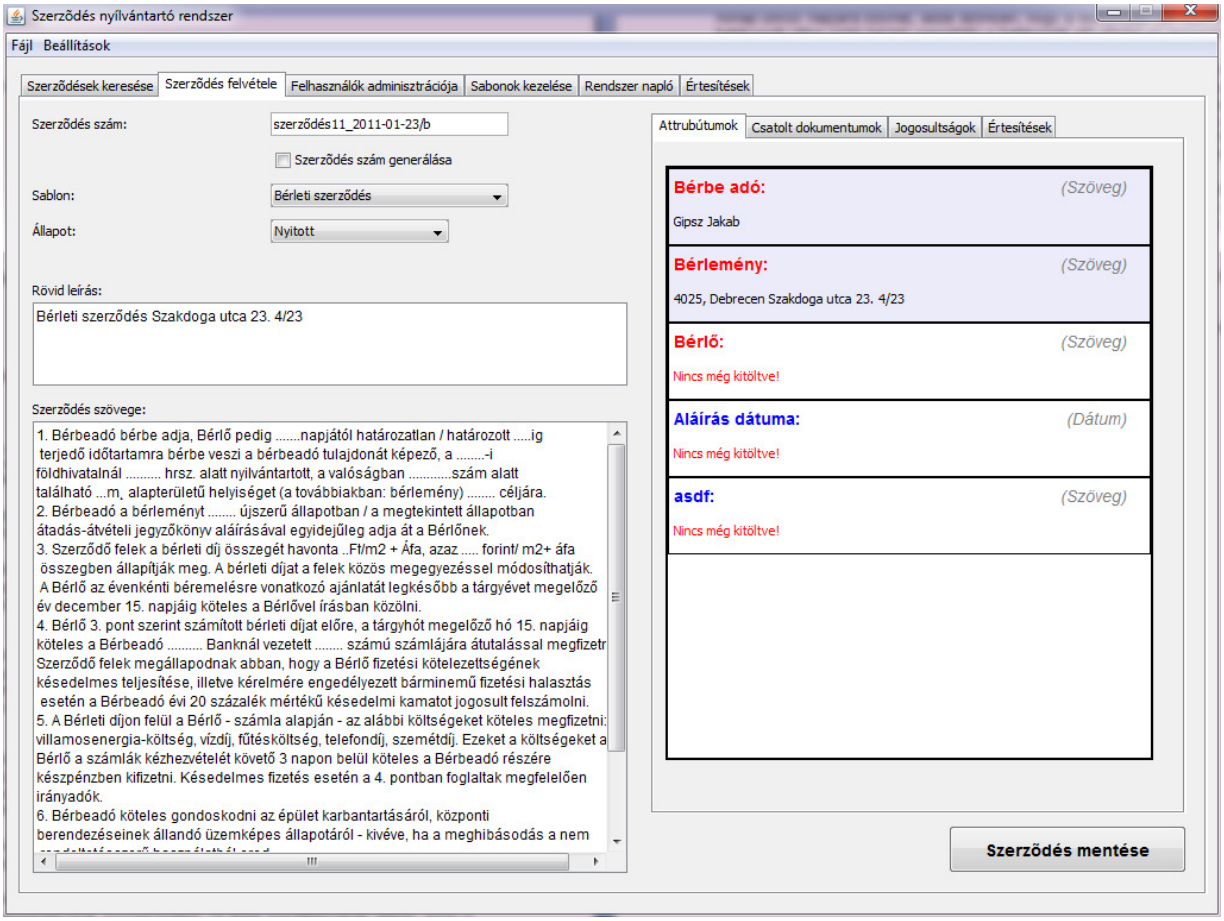
16. ábra: Eredménylista elemének felépítése

Az eredménylista elemei rendezhetőek a szerződés száma, típusa és létrehozási ideje szerint növekvő illetve csökkenő sorrendben. Ez a lista felett elhelyezett kettő darab legördülő menü segítségével állíthatjuk be.

Amennyiben az eredmények között megtaláltuk az általunk keresett dokumentumot, dupla kattintással vagy kijelölés után a *Megnyitás* gomb használatával megnyithatjuk – jogosultságunktól függően – szerkesztésre vagy olvasásra. A kijelölt szerződést a *Törlés* gomb segítségével törölhetjük a rendszerből. A törlés végleges, és visszavonhatatlan. A dokumentum megnyitásakor egy új ablak jelenik meg, melynek tartalma azonos a *Szerződés felvétele* panellel, ennek felépítését a következő fejezet fogja bemutatni részletesen.

3.3.4.2. Szerződés felvétele

A második fül alatt érhető el a szerződések felvételét szolgáló képernyő, amely csak azok számára látható, akik rendelkeznek az ehhez szükséges jogosultságokkal.

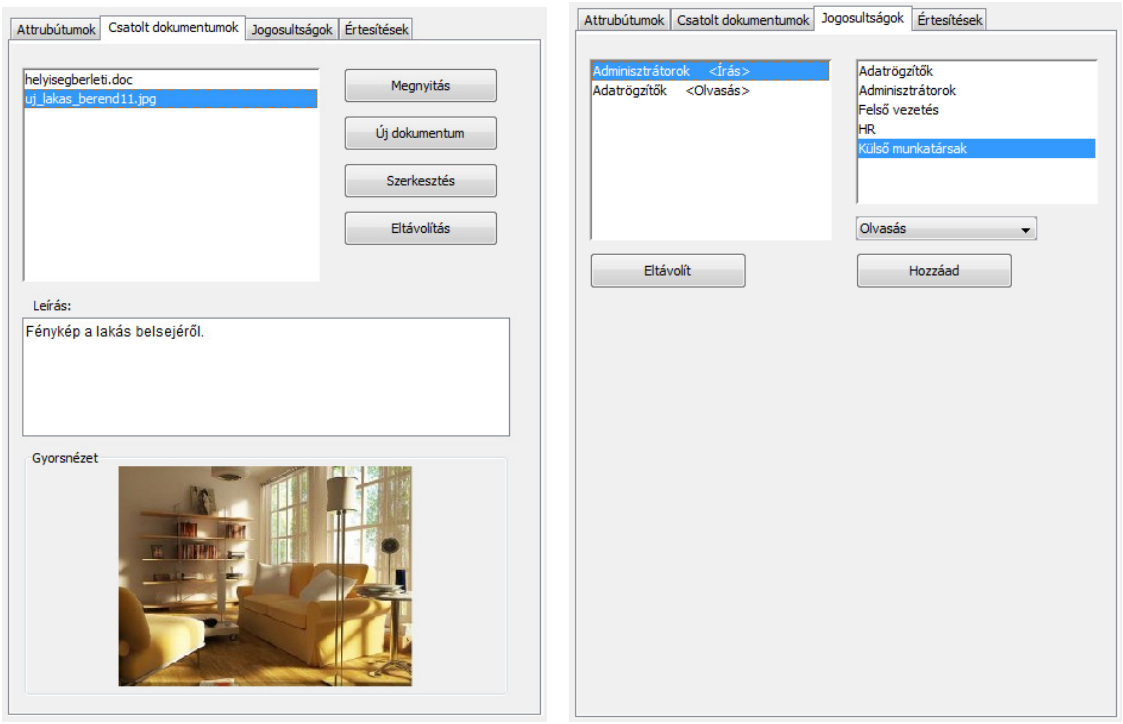


17. ábra: Szerződés felvétele képernyő

A szerkesztő felület bal oldalán a legfontosabb adatokat tudjuk beállítani. A jobb oldali rész egy fülekkel ellátott panel. Az első fülön tudjuk beállítani a szerződés attribútumait. A kötelezően kitöltendő adatok neve piros betűvel jelenik meg, míg az opcionálisak kékkel. A már

kitöltött attribútumok sora halványkék színű hátteret kap, ezzel is elősegítve az átláthatóságot. A második fülön tudunk csatolt dokumentumokat hozzáadni a szerződéshez.

A panel felső részén látható a csatolt dokumentumok listája. A mellette elhelyezett gombok segítségével tudunk új fájlt csatolni, meglévőt szerkeszteni (leírását), eltávolítani és megnyitni. Középen a kijelölt dokumentum leírása olvasható, alul pedig egy gyorsnézet foglal helyet, ami képek esetén egy kicsinyített méretű változatot, néhány gyakran használt dokumentum formátum (doc, xls, pdf, ppt) esetén egy, a fájl típusára utaló ikont, egyéb esetekben pedig egy kérdőjelet mutat.



18. ábra: Dokumentum csatolása és jogosultságok hozzárendelése

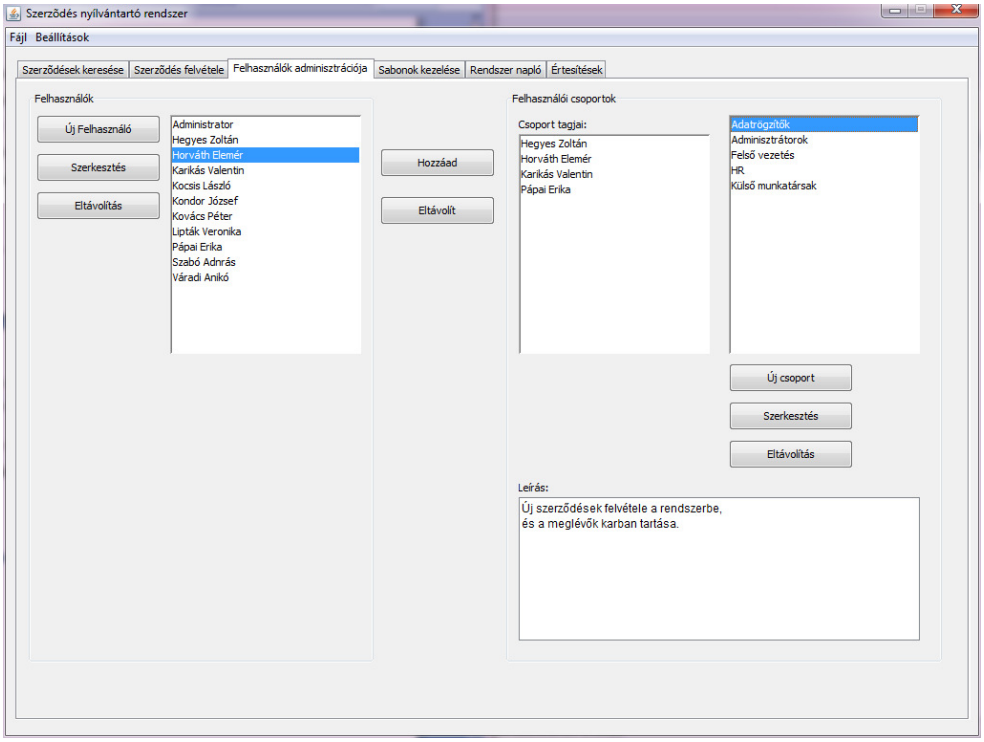
A harmadik fülön jogosultságokat tudunk hozzárendelni a szerződéshez. A bal oldali listában látható a kiosztott jogosultságok, a jobb oldaliban pedig a rendszerben regisztrált felhasználói csoportok, amelyekhez írási vagy olvasási jogot adhatunk a szerződésünkhöz.

A negyedik fülön van lehetőségünk értesítéseket definiálni. Új értesítés létrehozásakor meg kell adnunk, hogy mikor, kiknek, és milyen üzenettel jelenjen meg az értesítés.

Amennyiben nem új szerződést készülünk létrehozni, hanem egy meglévőt szerkesztünk, akkor az eddig felsoroltak mellett egy ötödik, információs fület is láthatunk, innen megtudhatjuk, hogy ki és mikor hozta létre a szerződést.

3.3.4.3. Felhasználók adminisztrációja

Ezt a képernyőt csak megfelelő adminisztrátori privilégium birtokában érhetjük el. Itt tudjuk létrehoz és naprakészen tartani a felhasználók és felhasználói csoportok adatait, illetve új felhasználók regisztrációja is itt történik.



19. ábra: Felhasználók adminisztrációja képernyő

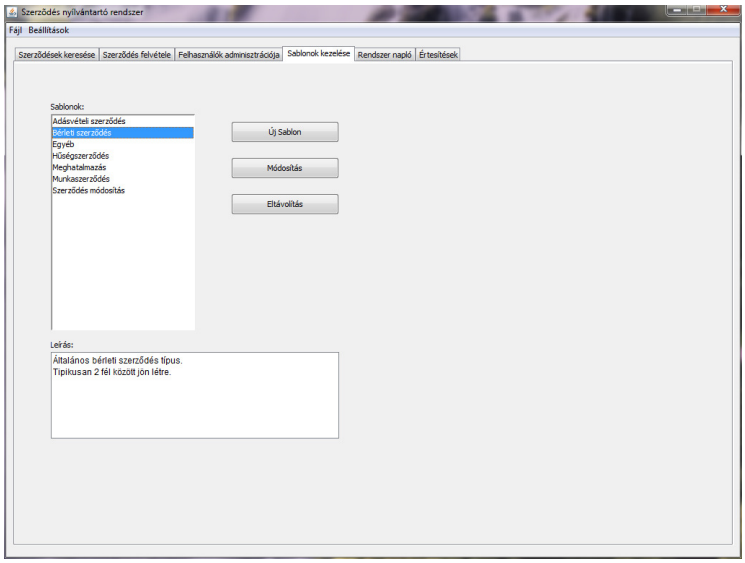
A képernyő bal oldalán a felhasználók listája és az ehhez kötődő létrehozás, szerkesztés és eltávolítás funkciók. A felhasználókról nyilvántartott adatok:

- Felhasználónév (ezzel tud bejelentkezni)
- Jelszó
- Név
- Cím
- Munkakör
- Telefonszám
- E-mail cím
- Jogosultságok

A képernyő jobb oldalán találhatóak a rendszerben létrehozott felhasználói csoportok valamint az egy adott csoportba tartozó felhasználók. A képernyő közepén elhelyezett *Hozzáad* gomb megnyomásával tudjuk a baloldalon kijelölt egy vagy több felhasználót a jobboldalon kijelölt csoporthoz hozzáadni. Az *Eltávolít* gomb segítségével tudunk egy felhasználót eltávolítani a csoportból.

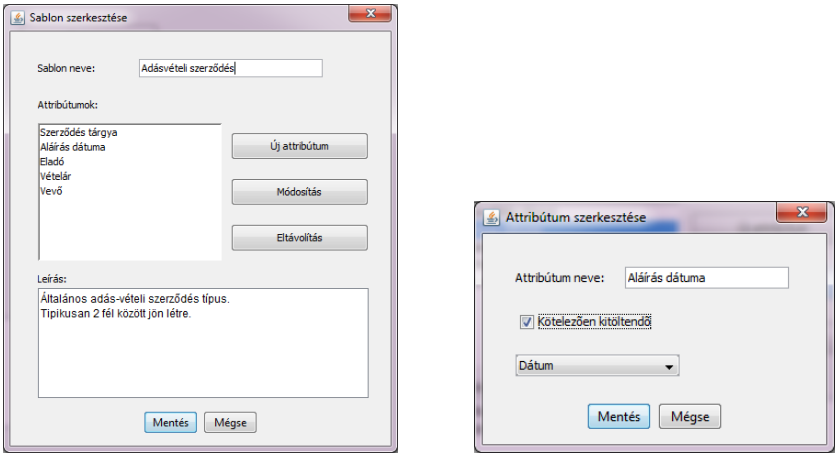
3.3.4.4. Sablonok kezelése

A *Sablonok kezelése* képernyőn a rendszerben tárolt szerződés sablonok karbantartását tudjuk elvégezni. A felületet csak azok a felhasználók érhetik el, akik az ehhez szükséges jogosultságokkal rendelkeznek. A baloldalon látható lista tartalmazza a már meglévő sablonjainkat. A lista melletti gombok segítségével új sablon létrehozására, meglévő módosítására és törlésére van lehetőségünk.



20. ábra: Sablonok kezelése

Ahogy azt már korábban leírtam, a sablonokat attribútumok alkotják. Ezeket az attribútumokat, valamint a sablon nevét és leírását tudjuk beállítani a sablon-szerkesztő ablakban. Az attribútumok listája mellett a már megszokott gombok találhatóak, melyek segítségével a sablon attribútumait tudjuk létrehozni, módosítani vagy törölni.

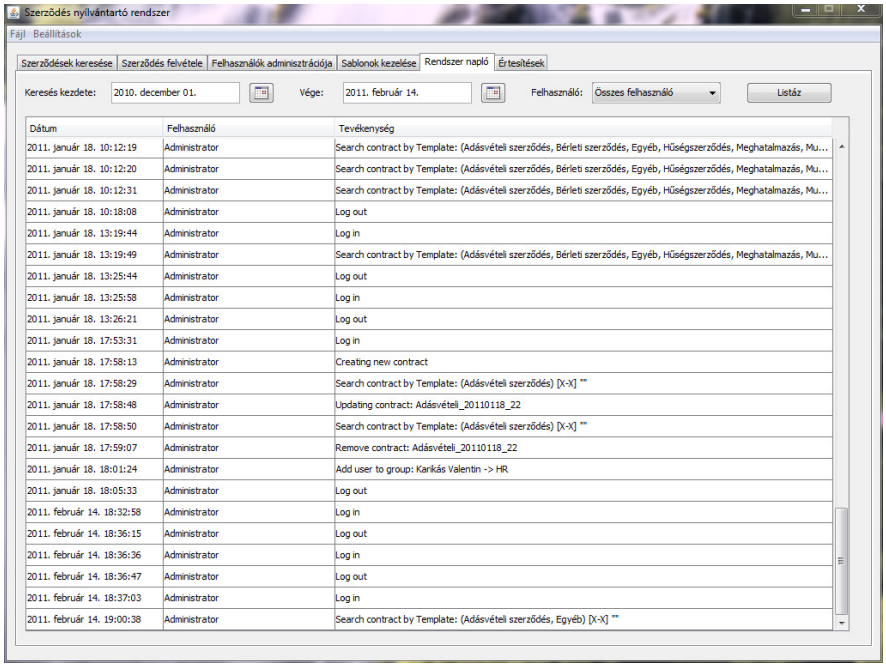


21. ábra: Sablonok és attribútumok szerkesztése

Az attribútum-szerkesztő felületen állíthatjuk be az attribútum nevét, típusát, valamint a *Kötelezően kitöltendő* beállítás bejelölésével tudjuk biztosítani, hogy szerződés létrehozásakor az adott attribútum kötelezően ki legyen töltve.

3.3.4.5. Rendszernapló

A *Felhasználók adminisztrációja* mellett a *Rendszernapló* a másik olyan képernyő, amit kizárólag az adminisztrátorok láthatnak. Segítségével lehetőségünk van kilistázni a rendszernapló tartalmát egy adott időszakra szűrve. A kiválasztott idő intervallumon belül kérhetjük az összes felhasználó bejegyzéseit, vagy szűkíthetjük a keresést egy adott személy tevékenységeire.

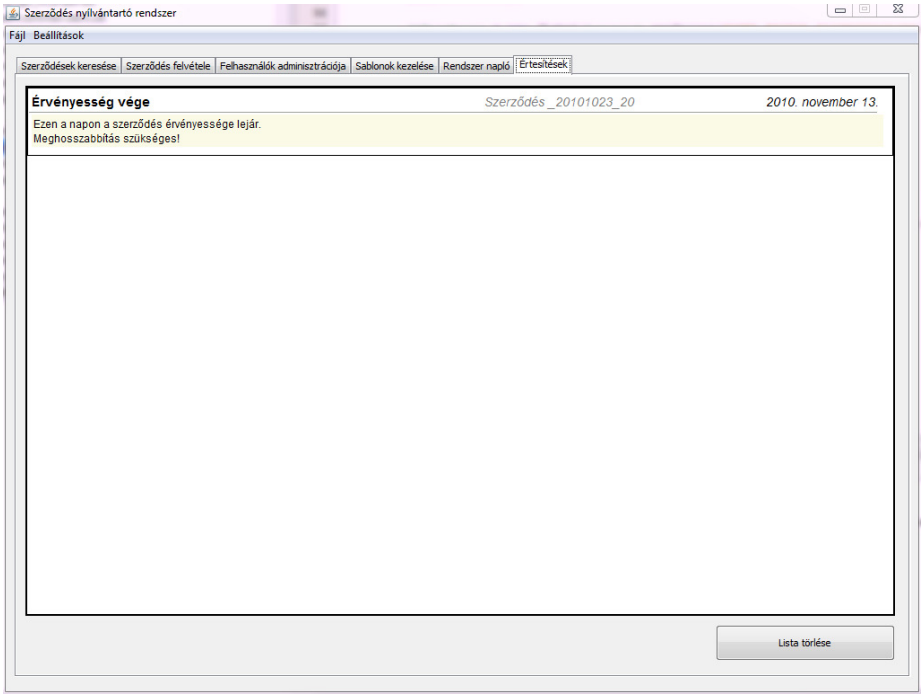


22. ábra: Rendszernapló

A naplóban minden egyes felhasználói interakcióról bejegyzés készül, így könnyen nyomon követhető, hogy melyik felhasználó mivel foglalkozott, valamint hiba esetén egyszerűen visszakereshető a probléma okozója.

3.3.4.6. Értesítések

Utolsó helyen láthatjuk az *Értesítések* képernyőt. A felhasználó bejelentkezés után itt azonnal láthatja a neki vagy csoportjának szóló, esedékessé vált értesítéseket. Az értesítések listája összetett elemeket tartalmaz, amelyből megtudhatjuk az értesítés címét, szövegét, az értesítést tartalmazó szerződés azonosítóját és az esedékesség dátumát.



23. ábra: Értesítések

A *Lista törlése* gomb megnyomásával a felhasználó jelzi, hogy tudomásul vette az értesítéseket, és törli a listát. A törölt értesítések nem visszaállíthatóak, és csak az ezután aktuálissá váló értesítések fognak legközelebb megjelenni a listában.

Amennyiben a felhasználónak vannak aktuális értesítései, az *Értesítések* fül piros színű, azonban egy ismert Java bug miatt ez néhány operációs rendszer alatt nem működik. [7]

4. Fejlesztési lehetőségek

Bár a rendszer jelenlegi állapotában is tökéletesen használható, számos további lehetőséget rejt magában, amelyek implementálásával a program használata a mindennapi gyakorlat során még rugalmasabbá, gördülékenyebbé válhat, vagy éppen a valós életben előforduló komplexebb problémák kezelését is lehetővé teheti.

Az egyik leg kézenfekvőbb ilyen lehetőség a rendszerben nyilvántartott szerződések hierarchiába szervezésének lehetővé tétele. Valós életbeli helyzetekben nem ritka, hogy egy szerződéshez később kiegészítéseket vagy szerződésmódosításokat csatolnak. Természetes igény tehát, hogy ezeket szeretnénk egy helyen, az eredeti dokumentumhoz csatolva látni. Ahogy azt már korábban, az adatbázis struktúráját bemutató fejezetben megemlítettem, az ennek a funkcionalitásnak a megvalósíthatóságát szolgáló oszlop már jelen van az adatbázis megfelelő táblájában. Név szerinte ez a *contract* tábla *parent_id* oszlopa. Ide tárolhatjuk annak a szerződésnek az azonosítóját, amely alá az új dokumentumot csatolni szeretnénk. Így szükség szerinti mélységű fa struktúra hozható létre, melyben a gyökér elem esetén a *parent_id* értéke *NULL*. Azonban a felhasználói felület módosítása a strukturált dokumentumok megjelenítéséhez már egy több időt és alapos tervezést igénylő feladat.

Következő lehetséges funkcionalitás a verzió-kezelés. Lényege, hogy egy szerződés módosításakor a régi verziót nem egyszerűen felül írjuk, hanem az új változatot megnövelt verzió számmal újra letároljuk. Így lehetőségünk nyílik nyomon követni, hogy egy adott szerződésen ki, mikor és pontosan mit módosított.

Az előző két fejlesztési lehetőség még a viszonylag egyszerűen megvalósítható, azonban a most következő már jóval elrugaszkodottabb gondolatokat tartalmaz. A rendszer használatának egyik fő nehézsége, a papír alapú szerződések szövegének kézzel való begépelése. Nagyban növelné a program használatának hatékonyságát, ha ezt a terhet levehetnénk a felhasználó válláról, és automatizálhatnánk. Ehhez természetesen szükségünk lenne valamilyen

karakterfelismerő szoftver integrálására, amelyet egy erre a területre szakosodott fejlesztő cégtől kellene beszerezni. Ideális esetben a felhasználónak nem lenne más dolga, mint behelyezni a papíron íródott szerződést a lapolvasóba, megnyomni a felületen egy gombot, és a képernyőn megjelenő szövegen legfeljebb már csak kisebb-nagyobb korrekciókat végezni. És ha már a lapolvasó használatánál tartunk, még egy további apró kényelmi szolgáltatás lehet, hogy a szerződéshez csatolni kívánt képállományokat akár közvetlenül a digitalizáló eszközből lehetne kinyerni.

Véleményem szerinte az ebben a fejezetben ismertetett fejlesztési lehetőségek mindegyike jelentősen hatékonyabbá tehetné a szoftver gyakorlati használhatóságát, ám sajnos az ezek kivitelezéséhez szükséges idő nagysága miatt nem jutottak tovább a tervező asztalnál. Bízom benne, hogy a nem túl távoli jövőben, lehetőségem lesz a megvalósításukra.

5. Irodalomjegyzék

[1] Java (programozási nyelv):
http://www.oracle.com/technetwork/java/javase/documentation/index.html

[2] Java Enterprise Edition:
http://www.oracle.com/technetwork/java/javaee/documentation/index.html

[3] Apache Ant:
http://ant.apache.org/

[4] Web Service:
http://hu.wikipedia.org/wiki/Web_service

[5] PostgreSQL:
http://www.postgresql.org/

[6] GlassFish application server
http://glassfish.java.net/

[7] Java LookAndFeel hibajegy:
http://bugs.sun.com/bugdatabase/view_bug.do;jsessionid=26b5f22f12a67ffffffffe7498f127f536e4?bug_id=6875229

6. Függelék

6.1. Rövidítések

Rövidítés	Jelentés
API	Application Programming Inerface – Alkalmazásprogramozási felület
EJB	Enterprise Java Bean – Java szerveroldali komponens
HTTP	HyperText Transfer Protocol – Információátviteli protokoll a világhálón
JDBC	Java DataBase Connectivity – Java adatbázis kapcsolat
JPA	Java Persistence API – Java perzisztencia réteg
JVM	Java Virtual Machine – Java virtuális gép
SQL	Structured Query Language – Strukturált lekérdező nyelv
XML	Extensible Markup Language – Kiterjeszthető leíró nyelv
WSDL	Web Service Descriptor Language – Web szolgáltatás leíró nyelv