DIPLOMAMUNKA

Kriszt Sámuel

Debrecen
2007

Debreceni Egyetem
Informatikai Kar

Webes portál fejlesztése Java környezetben

Témavezetők:

Gábor András
PhD hallgató

Dr. Fazekas Gábor
egyetemi docens

Készítette:

Kriszt Sámuel
programtervező matematikus

Debrecen
2007

Tartalomjegyzék

Bevezetés

A téma

Diplomamunkám témájául egy konkrét portál fejlesztése szolgált. Az egyetemi éveim során voltak olyan tantárgyak, ahol egészen életszerűnek ható feladatokat kellett megoldanunk (ilyen volt például egy régiségek aukcióit lebonyolító webes alkalmazás megvalósítása), végül mégis csak magunknak készítettük a programokat, nem voltak felhasználók, akik tesztelhették volna az alkalmazásaink működőképességét (a tanárokat leszámítva, akik csupán bizonyos esetekre tesztelték azokat). Ezért gondoltam rá, hogy – mivel a későbbiekben a munkám során csak ilyen feladatokkal fogok találkozni – olyan rendszert fejlesszek, amely valós problémával foglalkozik, amely a gyakorlatban is használható lesz, és amit ténylegesen használni is fognak. Ezért választottam a diplomamunkám témájául, hogy az ELIT Nyelvstúdió kft-nek fejlesszek webes portált.

ELIT Nyelvstúdió kft.

Az ELIT Nyelvstúdió kft. gyermekek angol oktatásával foglalkozik, jelenleg több településen vannak nyelviskoláik (többek között Debrecenben is) több száz gyerekkel, sok tanárral és néhány adminisztrátorral. Néhány hónapot magam is dolgoztam diákként a kft-nél, az én feladatom volt a tanfolyamokkal kapcsolatban felmerülő problémák és a befizetések kezelése. Ennek során valamelyest betekintést nyerhettem a tanfolyamok menetébe, a nyelviskolák működésébe.

Mivel a kft. nem saját, hanem külföldi módszert használ a gyermekek angol tanítására, ezért ez bizonyos kötelezettségekkel is jár. Többek között interneten keresztül vezetni kell a módszert biztosító külföldi cég felé a tanfolyamok alakulását, így nyilván kell tartani a tanárokat, azok kurzusait és a kurzusokban résztvevő diákokat. Ez a nyilvántartás azonban nem az itteni, sokkal inkább az ottani igények alapján történik, így ez inkább csak kötelezettséget jelent a kft-nek, segítséget már kevésbé. Ezt látva gondoltam arra, hogy kifejlesszek egy olyan nyilvántartórendszert, amely az itteni igényeket próbálja meg kielégíteni, és amely a cég számára tényleges segítséget nyújt. A fejlesztés megkezdése előtt valamelyest tanulmányoztam a már működő külföldi rendszert, amely segítséget nyújtott a nyilvántartással kapcsolatos megkötések kialakításában (például kurzusok esetében a

módosításokkal kapcsolatos megkötések).

Célok

A portál fejlesztésének alapvető célja az volt, hogy megkönnyítse a kft-ben dolgozó tanárok, adminisztrátorok és a cégvezető munkáját, főleg az oktatással kapcsolatos nyilvántartás területén (eddig ez a nyilvántartás főleg papír alapon folyt, annak minden hátrányával együtt), továbbá, hogy hasznos információkkal szolgáljon a gyermekeiket tanfolyamra járató gondviselők, illetve a még csak érdeklődő szülők számára. Ezeken kívül a célok közt szerepelt még a kapcsolattartás, illetve a céghez való szorosabb kötődés kialakításának elősegítése is a szolgáltatást igénybevevőkkel. A tervek rögzítéséhez szükség volt a rendszerrel szemben támasztott konkrét igények feltárására, amire a cégvezetővel és egy adminisztrátorral való egyeztetések folyamán került sor.

Aktuális állapot

Jelenleg a portál dinamikus része, a webes alkalmazás van megvalósítva (az alkalmazás neve: *elitChildren*). Az alkalmazás használatával így lehetőség nyílik a nyelvoktatással kapcsolatos nyilvántartás vezetésére. A portál többi része főként statikus információkat fog szolgáltatni magáról a kft-ről, ezen kívül azonban néhány dinamikus információ is elérhető lesz publikusan – mint például az aktuálisan meghirdetett tanfolyamok.

A dolgozat felépítése:

A dolgozat 5 fejezetből épül föl. Az első fejezetben röviden ismertetem a fejlesztéshez alkalmazott technológiákat. A második fejezet foglalkozik magával a rendszerrel, nagy hangsúlyt fektetve az alkalmazás által használható funkciók ismertetésére. A harmadik fejezet az alkalmazással kapcsolatos konfigurációs beállításokat tartalmazza. A negyedik fejezet írja le a rendszer felépítését az adatbázisrétegtől kezdve az adatelérési- és alkalmazási rétegen át a felhasználói rétegig. Az ötödik fejezetben mutatom be a webes alkalmazással és magával a portállal kapcsolatos további terveket.

Szeretném megragadni a lehetőséget, hogy köszönetet mondjak a témavezetőmnek, Gábor Andrásnak, aki észrevételeivel, ötleteivel, tanácsaival segített a rendszer tervezésében, fejlesztésében és a téma kifejtésében.

1. A fejlesztéshez alkalmazott technológiák

A rendszer szerveroldali részében ötvöződik a szervlet és a JSP technológia. A webes alkalmazás fejlesztése NetBeans 4.1-es fejlesztői környezetben folyt.

Az alkalmazás a PostgreSQL 8.1 adatbáziskezelő-rendszert használja. Van lehetőség más adatbázis-kezelő használatára is, az átálláshoz azonban módosítani kell bizonyos lekérdezéseket. Az alkalmazás fejlesztése során ugyanis kihasználtam az adatbáziskezelő-rendszer nyújtotta lehetőségeket, így egyes lekérdezések PostgreSQL specifikusak. Az alkalmazás használ triggereket is, amelyek a PostgreSQL kezelőnyelvén, a PL/pgSQL-ben íródtak. Más adatbázisrendszerek esetén ezeket a triggereket nyilván más nyelven kell megírni (Oracle esetén például PL/SQL-ben).

A relációs adatbázisréteg és az objektum-orientált alkalmazási réteg között a Hibernate 3.0 biztosítja a kapcsolatot.

A megjelenítés a CSS, a Javascript és az XHTML technológia segítségével történik.

A portál fejlesztése és tesztelése során Apache 2 webszervert és Tomcat 5.5 alkalmazásszervert használtam.

A tervezéshez a VisualParadigm programot használtam, a diplomamunkában szereplő diagramok (használati eset, csomag, valamint adatbázisséma diagram) is ennek az eszköznek a segítségével készültek.

Az alkalmazásban szereplő grafika a GIMP és a 3D Flash Animator szoftverekkel készült.

1.1 . Java

A Java egy objektum-orientált programozási nyelv. A Sun Microsystems fejlesztette a 90-es évektől kezdve és fejleszti napjainkban is. A kifejlesztésekor fontos szempont volt, hogy operációs rendszertől független legyen. A Java-kódban megírt programok így egy köztes, úgynevezett bájtkódra fordíthatók, amelyek nem kötődnek az adott operációs-rendszerhez. A bájtkód futtatása a Java virtuális géppel (JVM) történik, amely az abban található műveleteket gépi kódú utasításokra fordítja, majd futtatja.

A Java elsőként tette lehetővé kisalkalmazások távoli gépeken való biztonságos futtatását. Szintaxisát elsősorban a C és a C++ nyelvektől örökölte.

1.2 . Szervletek

A szervletek olyan Java nyelven készített szerveroldali komponensek, amelyek HTTP kliensek számára nyújtanak különféle szolgáltatásokat. Ilyen szolgáltatás például a dinamikus tartalom előállítása vagy HTML-űrlapoknak a feldolgozása.

Mivel a szervletek nem önálló programok, csupán komponensek, ezért nem is a szabványos bemenetről kapják a kliensektől az adatokat, és nem is a szabványos kimenetre írják a válaszukat. Ehelyett a szervletek a szolgáltatásaikat megvalósító metódus paramétereként átadott bemeneti csatornán keresztül férnek hozzá a kliens által küldött adatokhoz, és az ugyancsak paraméterként megadott kimeneti csatornára írják vissza válaszukat.

A szervletek a *javax.servlet.Servlet* interfészt kell, hogy implementálják. Három fő metódusa ennek az interfésznek az *init()* (egy paramétere van: *ServletConfig config)*, a *service()* (két paramétere van: *ServletRequest kérés*, *ServletResponse válasz*) és a *destroy()*. Az *init()* metódus a szervletet implementáló osztály példányosításakor kerül meghívásra. Miután ez lefutott, csak azután kerülhet sor a szervlet szolgáltatásait implementáló *service()* metódus meghívására. A *destroy()* metódus meghívása jelenti a szervlet életciklusának a végét, ezt a metódust a szervletet futtató környezet hívja meg, amikor már úgy véli, hogy a szervletre nincs tovább szükség. A *Servlet* interfészt implementáló osztályok implementálják ezeket az absztrakt metódusokat.

1.3 . JSP

A Java Server Pages dinamikus tartalom szerveroldali előállítására szolgáló technológia. A szervletek is alkalmasak dinamikus tartalom előállítására, ott a Java-kódba beágyazva jelennek meg a szöveget a kimenetre író utasítások, a JSP technológiánál ezzel szemben pont ellenkezőleg, a szöveges dokumentumba (pl.:HTML, XML) beágyazva szerepelnek a tartalmat módosító utasítások.

A JSP technológiával megírt dinamikus weboldalak megjelenése így egy erre fölkészített webszerkesztővel is könnyen alakítható, szemben a szervletekkel, ahol ez nem kivitelezhető, mivel a formázási utasítások a Java-kódba elrejtve jelennek meg. Másik fontos előnye a JSP technológiának, hogy használatával lehetőség nyílik rá, hogy a dinamikus

tartalom előállításáért felelős programkód ne keveredjen össze a statikus tartalommal és a megjelenítéssel.

Miután kérés érkezik a szerverre egy klienstől egy JSP oldal elérésére, a JSP fordító előállítja az oldal alapján a neki megfelelő Java-szervletet, majd az így létrehozott Java kódot a Java fordítóval lefordíttatja egy *.class* kiterjesztésű fájlba. Ezután kerül sor a szervlet futtatására, melynek során a szervlet a kérés alapján előállítja a dinamikus tartalmat, ezt kapja meg a kliens válaszként. A JSP oldal szervletté alakítására az első kérés kiszolgálásakor kerül sor, ekkor természetesen lassabban töltődik be az oldal, de miután elindult a szervlet, a további kérések már egyenesen a szervlethez kerülnek. (Van lehetőség azonban az oldal előfordítására is, ekkor már az első kérés sem lesz lassabb.)

Webalkalmazások készítésekor a munkamegosztás elősegítésére célszerű ötvözni a JSP és a szervletes technológiát. A két technológia között a bean-ek képezhetik az összekötő kapcsot, így a szervletek a kéréseket feldolgozva előállíthatják a válaszokat, a JSP oldalak pedig a bean-ek segítségével kaphatják meg ezeket a válaszokat, és építhetik be azokat a statikus tartalomba.

1.4 . Hibernate

A Hibernate egy objektum-relációs leképezést megvalósító keretrendszer, mely a programozó számára Java objektumok relációs adatbáziskezelő rendszerben történő tárolását-manipulálását teszi lehetővé. Segítségével jól elkülöníthetővé válik az adatelérési- és az alkalmazásréteg.

A Hibernate megvalósítja a Java osztályok adatbázis-táblákba történő leképezését (és a Java adattípusok SQL adattípusokra való leképezését), adat lekérdezési és visszakeresési lehetőségeket biztosít. Csökkenti a fejlesztésre fordított időt, hiszen a használatával fölöslegessé válik, hogy a programozó saját maga írja meg a perzisztencia-réteget, kezelje az adatbázis-kapcsolatot JDBC-n keresztül vagy SQL lekérdezéseket hajtson végre.

Nagy előnye a Hibernate-nek, hogy egyedi (SQL-szerű) lekérdezőnyelvének segítségével kiküszöbölhetőek az adatbáziskezelő-rendszer specifikus SQL-lekérdezések, így adatbáziskezelő-rendszertől független alkalmazásokat fejleszthetünk ki.

A Hibernate nem elsősorban azoknak az alkalmazásoknak a fejlesztéséhez nyújt segítséget, amelyekben az üzleti logika jó része az adatbázisban, tárolt eljárások formájában

található, inkább azoknak, ahol az üzleti logika a középső, Java alapú rétegben található.

1.5 . Apache Ant

Az Apache Ant egy fejlesztői segédeszköz, amely Java alkalmazások fejlesztésekor jelentősen megkönnyítheti a programozó munkáját többek között az alkalmazások fordításában, futtatásában, tesztelésében. Ötvözi a Java platform-függetlenségét az XML könnyű olvashatóságával és bővíthetőségével. Használatához egy *build.xml* nevű fájlt kell megfelelően konfigurálni. Az Ant futtatása nagyon egyszerűen, az *ant* paranccsal történik (persze, ahhoz, hogy ez működjön, a *PATH* környezeti változóban el kell helyezni az Ant *bin* könyvtárát). Ekkor az Ant az éppen aktuális könyvtárban lévő *build.xml*-t értelmezi, és futtatja az abban alapértelmezettként megadott tevékenységet.

1.6 . JavaScript

A JavaScript egy olyan objektum-alapú leíró nyelv, melyet weboldalak, webes alkalmazások kliens oldali programozására fejlesztettek ki. Egy kliens alkalmazásban az erre felkészített böngésző képes arra, hogy felismerje a webes tartalomba beágyazott JavaScript utasításokat, és végrehajtsa azokat. Mivel a JavaScript kód értelmezése a kliens oldalon történik, ezért a böngésző azonnal képes reagálni felhasználói eseményekre, anélkül, hogy azokat a szerverre kellene továbbítania.

Annak ellenére, hogy a nyelv nevében szerepel a Java, a két nyelv között nincs igazi kapcsolat, hasonlóság a szintaxisuk alapjaiban lelhető fel (mindkettő szintaxisa a C nyelvből származtatható). A szemantikájuk eléggé eltérő, az objektum modelljük között pedig semmilyen kapcsolat nincs. A Java 6-ban már be van építve olyan csomag (*javax.script*), amely lehetővé teszi a Java alkalmazások számára, hogy futási időben értelmezzék és futtassák a JavaScript-ben megírt szkripteket.

A JavaScript nem egy szigorúan típusos nyelv, a Java-val ellentétben nincs benne típusellenőrzés. Csupán kis számú adattípust különböztet meg: numerikus, logikai és sztring típus. Vannak benne beépített függvények és kiterjeszthető objektumok, de nincsenek benne osztályok, és nem támogatja az öröklődést.

1.7. CSS

A CSS egy stíluslapnyelv, melyet eredetileg HTML-oldalak formázására fejlesztettek ki. Ma már XML-dokumentumok formázására is használják. HTML dokumentumok esetén CSS stílusokat kapcsolt, beágyazott stíluslapok illetve soros stílusok segítségével lehet alkalmazni. Kapcsolt stíluslapok esetén a stílusra vonatkozó formázási szabályok külön fájlban – ún. stíluslapban – helyezkednek el, így a HTML dokumentumban elég egy hivatkozást megadni erre a stíluslapra. Beágyazott stíluslapok esetén a stílusra vonatkozó formázási szabályok nem külön dokumentumban, hanem az adott HTML dokumentumban, egy *style* elemen belül jelennek meg. A soros stílus pedig egy meghatározott elempéldányra közvetlenül alkalmazott szabály. XML dokumentumok esetén CSS formázást stíluslapokra történő hivatkozással lehet megadni.

1.8. XHTML

Az XHTML a HTML technológián alapul. A HTML egy jelölőnyelv, mely lehetővé teszi az interneten letölthető tartalmak formázását. Ezen kívül a HTML egyik legfontosabb tulajdonsága, hogy HTML-dokumentumokba beépíthetők más HTML-dokumentumokra vonatkozó hivatkozások, továbbá lehetőség van hivatkozni más formátumban tárolt erőforrásokra is.

Az XHTML HTML-alapokon nyugszik, de XML-szabványokat követ, így egy XHTML-dokumentum XML-dokumentum is egyben. Ez természetesen megszorításokkal jár együtt. Mivel az XML-ben nem mindegy, hogy az elemek és a jellemzőnevek kis- vagy nagybetűket tartalmaznak, ezért az XHTML-ben egységesen kisbetűvel kell írni minden elemet és attribútumot. Ezen túlmenően az üres elemeknek is „/>” jellel kell záródniuk. Nincs lehetőség az attribútumok nevének elhagyására illetve az attribútumok értékét mindig idézőjelek között kell megadni.

Az XHTML technológiát úgy fejlesztették ki, hogy teljesüljön a visszafelé kompatibilitás, így az XHTML-dokumentumokat a HTML-böngészők is képesek feldolgozni.

2. A rendszer áttekintése

Az *elitChildren* alkalmazás alapvető célja, hogy segítséget nyújtson a gyermekek angol-oktatásával foglalkozó ELIT Nyelvstúdió kft-nek az oktatásban részvevő gyermekek és gondviselőjükhöz kapcsolódó adatok nyilvántartásának megkönnyítésében, valamint a kurzusokkal kapcsolatos befizetések kezelésében. (Jelenleg az alkalmazáson keresztül nem lehet a befizetést elintézni, abban csupán rögzíteni lehet a befizetés tényét – nyilvántartás céljából.)

Mivel a cégnek több különböző településen is van nyelviskolája, ezért nehezen jut el az információ az egyik nyelviskolától a másikig – ezt az információcserét próbálja megkönnyíteni az alkalmazás.

A rendszerbe felhasználói név és jelszó megadásával lehet bejelentkezni, az alkalmazás funkciói csak a bejelentkezés után érhetőek el. (Tervbe van véve a rendszer publikus, mindenki számára elérhető része.)

2.1 . A rendszer funkciói

Számítógépes alkalmazások fejlesztése során a végcél mindig egy megfelelő funkciókkal bíró alkalmazás létrehozása. Az, hogy milyen funkciókkal bír egy alkalmazás, illetve, hogy rendelkezik-e a tőle elvárt funkciókkal, döntő kérdés lehet az alkalmazás használhatóságát illetően, hiszen tulajdonképpen a funkciói írják le magát az alkalmazást. Így az *elitChildren* alkalmazás esetében is fontos kérdés, hogy milyen funkciókkal rendelkezik, illetve, hogy rendelkezik-e a tőle elvárt funkciókkal.

Mivel egy nyilvántartórendszerről van szó, így alapvetően a nyilvántartáshoz szükséges funkciók jelennek meg az alkalmazásban. A középpontban a nyelviskolákban folyó kurzusok nyilvántartása, és ehhez kapcsolódva a kurzusban résztvevő gyermekek, és a kurzussal kapcsolatos befizetések nyilvántartása áll. Ehhez azonban szorosan kötődnek azok a személyek is, akik végzik ezt a nyilvántartást, és azok is, akik betekintést nyerhetnek ebbe a nyilvántartásba. Így például a cégvezető, az adminisztrátor vagy a tanár, akik felelnek a nyilvántartás vezetéséért, és a gondviselők, akik bele tekinthetnek a nyilvántartás számukra lényeges részébe. Ez további funkciókat követel meg az alkalmazástól, hiszen a megfelelő személyekhez kapcsolódó rekordokat is nyilván kell tartani, kezelni kell őket, és mivel

többféle szerepkör is van, ezért meg kell oldani a különféle jogosultságok kezelését is. Minthogy a különböző jogosultságú felhasználók más és más nyilvántartási jogokkal rendelkeznek, ezért az általuk elérhető funkciók sem egyeznek meg, a különféle jogosultságokhoz más és más funkciók tartoznak. A fejezet további részében, a funkciók részletezése során feltüntetett használati eset diagramok nem fednek le minden egyes, a rendszer által használható funkciót, de segítségükkel könnyebben megérthető, hogyan is oszlanak meg a funkciók az egyes felhasználói jogosultságok alapján.

2.1.1. Jogosultságkezelés

A rendszer jelenleg négyféle jogosultságot különböztet meg: vezetői, adminisztrátori, tanári és gondviselői jogosultság. A legkisebb jogosultsággal a gondviselő bír. Ő az alkalmazásba belépve értesülhet a befizetéseiről, a számára lényeges tanfolyamok menetéről, valamint láthatja a saját illetve a hozzá tartozó gyermekek adatait, azokat módosíthatja. Őt követi a tanári jogosultság.

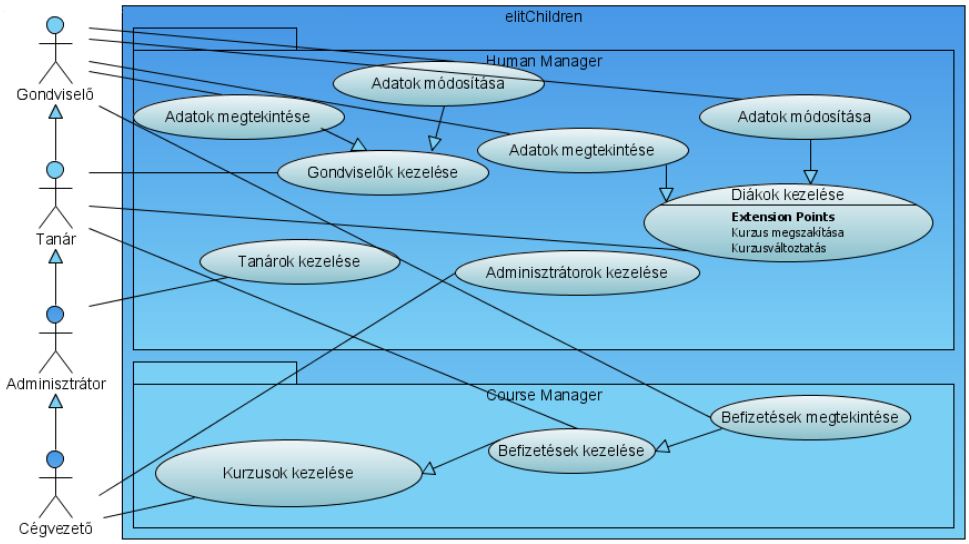
A tanár feladatkörébe tartozik a tanfolyamaira járó gyermekek nyilvántartása, a befizetések, valamint az órák dokumentálása. Új diákokat vehet fel a kurzusaiba, már meglévőket törölhet. Természetesen a tanárok csak a saját kurzusaikat és a kurzusaikba járó gyerekeket látják, csak azoknak az adatait módosíthatják.

A hierarchiában a következő helyen az adminisztrátor áll. Ő a tanár által is elérhető funkciókon túl létrehozhat új kurzusokat, új tanárokat vihet fel az adatbázisba. Ő csak azokat a tanárokat látja, akik a hatáskörébe tartozó nyelviskolákban tanítanak, és csak azokat a gyerekeket, akiket ezek a tanárok tanítanak.

A hierarchia legfelső szintjén van a cégvezető, aki az előbb említett funkciókon túl új adminisztrátorokat, új nyelviskolákat vihet fel az adatbázisba, illetve új kedvezménytípusokkal, új kurzustípusokkal bővítheti az adatbázist, a meglévőket módosíthatja. A cégvezető az adatbázisban fent lévő összes gyermeket, gondviselőt, kurzust és tanárt elérheti, elvégezheti a velük kapcsolatos módosításokat.

A jogosultságok tehát egy olyan hierarchia mentén rendeződnek, amelynél felülről lefelé haladva szűkül a hierarchiában szereplő felhasználók által elérhető funkciók halmaza. A hierarchia tetején áll a cégvezető, akinek a jogosultsága mindenre kiterjed, az alján pedig a gondviselő, aki a legkisebb jogosultsággal bír.

2.1.2. Gondviselői jogosultsággal elérhető funkciók:



2.1. ábra: Gondviselői funkciókat részletező használati eset diagram

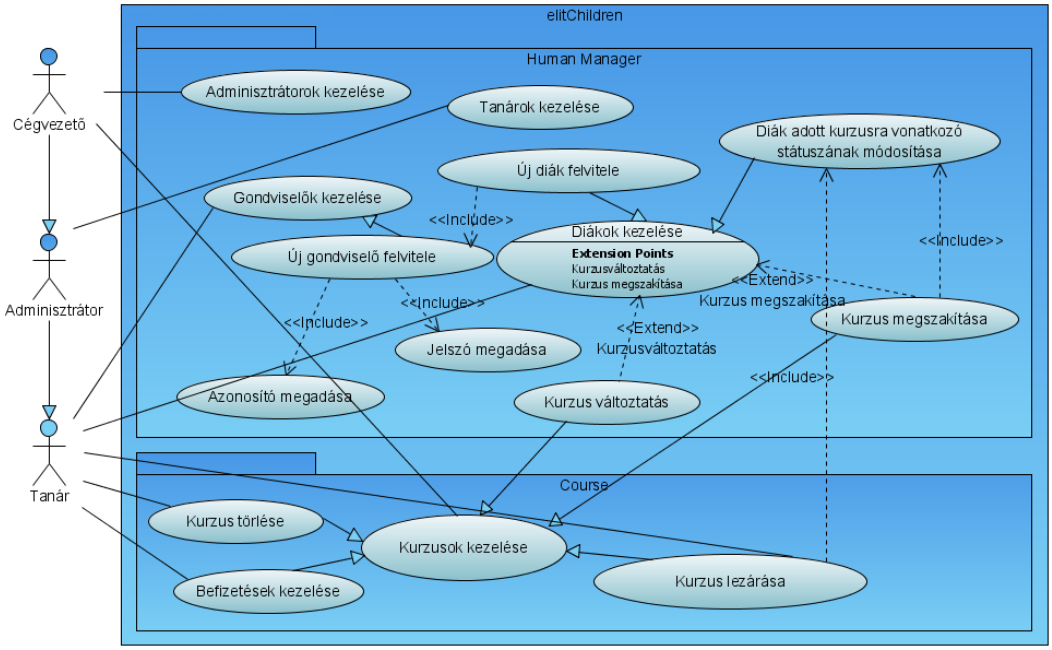
Gondviselői jogosultsággal a felhasználó jogosult egyrészt a saját személyes adatainak megtekintésére – módosítására, valamint a hozzá tartozó gyermekekkel kapcsolatos nyilvántartás megtekintésére, illetve a gyermekek személyes adatainak módosítására. A gyermekekkel kapcsolatos nyilvántartásba beletartoznak a gyermekek által látogatott kurzusok, az egyes kurzusokon belül az adott kurzushoz kapcsolódó befizetések, illetve megtartott órák nyilvántartása. A gondviselő ezeknek az adatoknak csupán a megtekintésére jogosult.

2.2. ábra: Gondviselői oldal

A gondviselő az alkalmazásba való belépése után látja a hozzátartozó gyermekek listáját, az egyes nevekre kattintva egy új oldalon láthatja az adott gyermek már elvégzett, illetve folyamatban levő kurzusait. A kurzusokra kattintva pedig nyomon követheti a kurzusok menetét, a megtartott órákat, illetve a befizetéseket. A befizetéseket egyszerűbben a *Befizetések* menüpontra kattintva érheti el a felhasználó, így minden addigi befizetése listázásra kerül.

A személyes adatok megtekintésére és módosítására a *Személyes adatok* menüpontra kattintva nyílik lehetőség.

2.1.3. Tanári jogosultsággal elérhető funkciók:



2.3. ábra: Tanári funkciókat részletező használati eset diagram

Nyelviskolákkal kapcsolatos funkciók

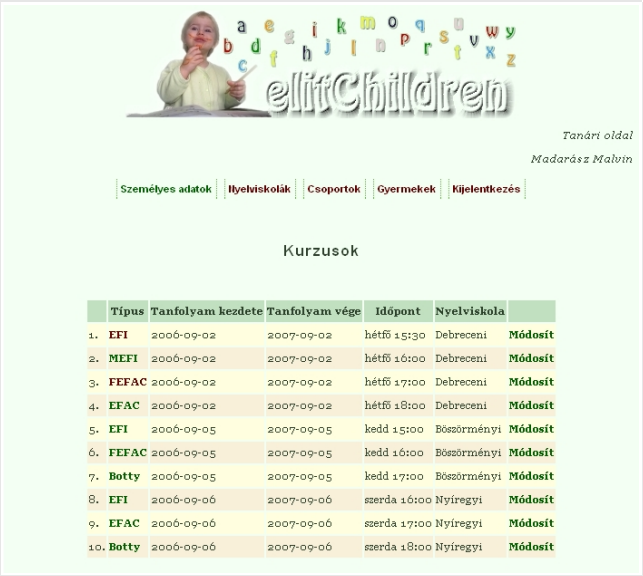
A tanár a nyelviskolák menüpontra kattintva láthatja azon nyelviskolák listáját, melyekben oktat. A nyelviskolák listájában a nyelviskolák fontosabb adatai jelennek meg (ilyen a név, a cím, a telefonszám). Ezeknek az adatoknak a módosítására a tanárnak nincs jogosultsága.

Kurzusokkal kapcsolatos funkciók

Tanári jogosultsággal elérhetőek azok a funkciók, amelyek gondviselői jogosultsággal

is elérhetőek, ezen túlmenően további funkciók is elérhetőek. Míg a gondviselő csupán megtekintheti a számára lényeges kurzusokkal kapcsolatos nyilvántartást, addig a tanár mindenféle módosításra jogosult a kurzusokkal kapcsolatban.

A tanár az alkalmazásba való belépés után látja az általa vezetett kurzusok listáját. A listában láthatóak az egyes kurzusok fontosabb adatai. Itt van lehetőség a kurzusokkal kapcsolatos módosítások elvégzésére is. A *Módosít* hivatkozásra kattintva egy új ablak nyílik meg, amely egy űrlapon részletezve jeleníti meg a kurzussal kapcsolatos aktuális adatokat, melyeknek egy része módosítható, egy része pedig nem.



	Típus	Tanfolyam kezdete	Tanfolyam vége	Időpont	Nyelviskola	
1.	EFI	2006-09-02	2007-09-02	hétfő 15:30	Debreceni	Módosít
2.	MEFI	2006-09-02	2007-09-02	hétfő 16:00	Debreceni	Módosít
3.	FEFAC	2006-09-02	2007-09-02	hétfő 17:00	Debreceni	Módosít
4.	EFAC	2006-09-02	2007-09-02	hétfő 18:00	Debreceni	Módosít
5.	EFI	2006-09-05	2007-09-05	kedd 15:00	Böszörményi	Módosít
6.	FEFAC	2006-09-05	2007-09-05	kedd 16:00	Böszörményi	Módosít
7.	Botty	2006-09-05	2007-09-05	kedd 17:00	Böszörményi	Módosít
8.	EFI	2006-09-06	2007-09-06	szerda 16:00	Nyíregyi	Módosít
9.	EFAC	2006-09-06	2007-09-06	szerda 17:00	Nyíregyi	Módosít
10.	Botty	2006-09-06	2007-09-06	szerda 18:00	Nyíregyi	Módosít

2.4. ábra: Tanári oldal

Módosítani lehet a kurzus időpontját, a státuszát és a befejezési dátumát. Nincs lehetőség viszont a kurzus típusának, kezdési dátumának módosítására. Az, hogy melyik nyelviskolában és melyik tanár vezeti a kurzust, a kurzus indításakor kerül rögzítésre, és később az sem módosítható.

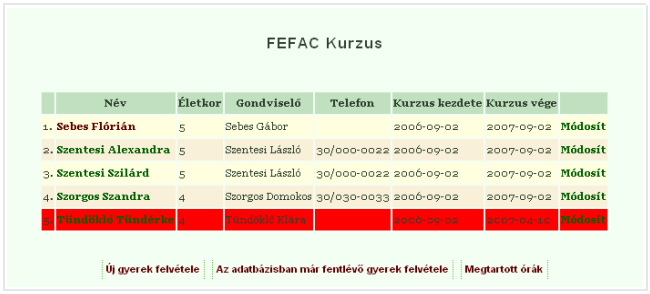
2.5. ábra: Kurzus módosítása

Minden kurzusban résztvevő gyermek nyilvántartásához hozzátartozik az is, hogy mikor kezdte el az adott kurzust, és mikor fejezte – vagy várhatóan mikor fejezi – azt be. A kurzus befejezési dátumának, illetve a kurzus státuszának módosításakor trigger biztosítja, hogy a kurzusban résztvevő gyermekek esetében is módosuljon a kurzus tervezett befejezési dátuma, illetve a gyermekek, mint a kurzusban résztvevő diákok státusza.

A kurzus lezárása a befejezési dátum megadásával és a státusz *lezárult*-ra történő módosításával hajtható végre. A kurzus törlése annyiban különbözik a kurzus lezárásától, hogy itt a státuszt *törölt*-re módosítjuk. A különböző státuszú kurzusok más-más stílusban jelennek meg a kurzusok listájában. Így már a megjelenés alapján könnyen el lehet dönteni, hogy egy adott kurzus milyen státusszal bír.

A kurzusok listájában a kurzus típusára kattintva a felhasználó egy új oldalon láthatja az adott kurzusban résztvevő gyermekek listáját.

Gyermekekkel kapcsolatos funkciók

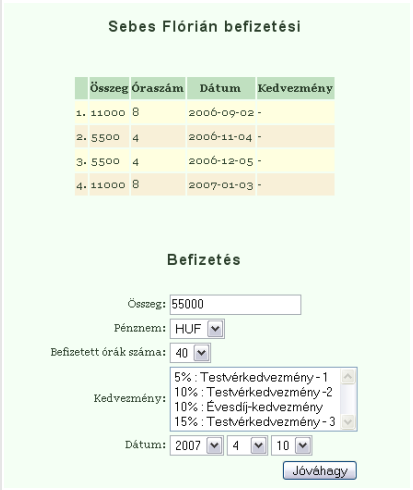


2.6. ábra: FEFAC kurzusban résztvevő gyermekek listája

A gyermekek listájában szerepelnek a gyermekekhez kapcsolódó fontosabb adatok (így a név, a gondviselő neve, a kurzusba való csatlakozás időpontja, a kurzus végének időpontja stb.). A listában a gyermekek adatainak megjelenítési stílusa a gyermekhez kapcsolódó rekord státuszától függ, tehát a kurzusokhoz hasonlóan itt is már a megjelenés alapján könnyen eldönthető, hogy egy résztvevő milyen státusszal bír. A 2.5. ábrán az 5. sorszámú gyermek adatai piros háttérszínnel jelennek meg. Ebből lehet következtetni, hogy a státusza *törölt*, mivel ez a stílus a *törölt* státuszhoz köthető.

A tanári jogosultsággal rendelkező felhasználó módosíthatja az egyes gyermekek adatait, törölhet gyermeket a listáról (amennyiben a gyermeknek már van rögzítve befizetése az adott tanfolyamra, úgy a törlés csupán logikai lehet, a státuszt lehet *törölt*-re állítani. A

törölt státuszú diákok piros háttérszínnel jelennek meg, személyes adataik nem módosíthatóak). A tanár a listát bővítheti az adatbázisban már szereplő (más kurzusokat végzett) gyermekekkel vagy azoknak testvérével illetve új gyermekekkel. Amennyiben a tanár valamelyik csoportjába egy, az adatbázisban már fent lévő gyermeket vagy annak testvérét akarja felvinni, keresési opciók megadásával választhatja ki az adatbázisban fent lévő gyermekek közül a megfelelőt. Új résztvevő felvitelekor a gyermek általános adatain túl a gondviselő adatait is fel kell tüntetni, illetve megadhatóak a gyermekhez kapcsolódó esetleges kedvezmények, illetve a kurzushoz való csatlakozás időpontja. A gondviselő jogosult a rendszer használatára, így egy új gondviselő felvitelekor a személyes adatainak megadásán túl egy azonosítót is létre kell hozni, melynek segítségével be tud majd jelentkezni a rendszerbe. Egy gyermekhez egyszerre több kedvezménytípus is tartozhat, így egy kurzus új gyermekkel történő bővítésekor lehetőség van több kedvezmény megadására is. Az itt rögzített kedvezmények a befizetések dokumentálásakor az ár kiszámításában automatikusan megjelennek.

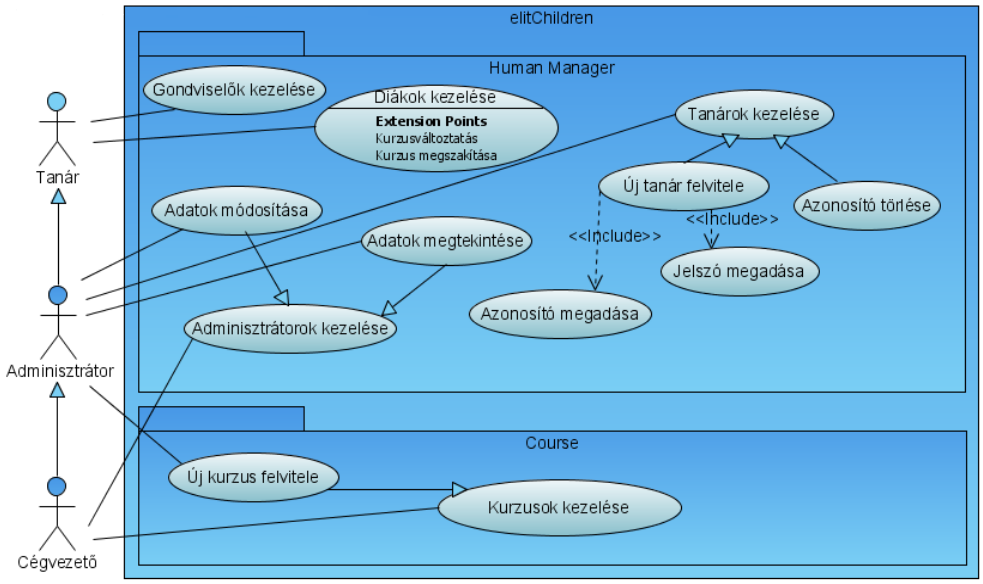


2.7. ábra: Befizetések nyilvántartása

A tanár feladata még a megtartott órák dokumentálása és a befizetések vezetése. A megtartott órák dokumentálása során a tanár rögzíti az órával kapcsolatos fontosabb információkat (a megtartott óra időpontja, az óra során a tananyagból átvett fejezetek sorszáma). A befizetések vezetése a kurzusban résztvevő gyermekekhez kapcsolódik. Egy kurzusban résztvevő gyermekek listájában a gyermekek nevére kattintva láthatóak a már elvégzett befizetések, és lehetőség van új befizetés rögzítésére is. Befizetésekhez

kapcsolódhatnak kedvezmények is, amennyiben az illető gyermek jogosult valamilyen kedvezmény igénybevételére. Amikor új gyermekkel bővíti a tanár egy kurzusát, lehetőség van az adott gyermekhez kapcsolódó kedvezmények rögzítésére (ez később is módosítható). A kedvezmények százalékokban jelennek meg, egy gyermek akár több kedvezményre is lehet jogosult (pl.: éves-díj kedvezmény, testvérkedvezmény). Új befizetés rögzítésekor a *Kedvezmény* mezőben megjelennek az adott gyermekhez kapcsolódó kedvezmények, ezek a befizetéskor is változtathatóak. Az adott kurzustípushoz tartozó kurzus-díj, az órák száma és a kedvezmények mértéke alapján az alkalmazás automatikusan kiszámolja a fizetendő összeget, és azt az *Összeg* mezőben jelzi. Persze lehetőség van ettől eltérő összeg bevitelére is.

2.1.4. Adminisztrátori jogosultsággal elérhető funkciók:



2.8. ábra: Adminisztrátori funkciókat részletező használati eset diagram

Tanárokkal kapcsolatos funkciók

Az adminisztrátori jogosultsággal rendelkező felhasználó az alkalmazásba belépve látja a nyelviskoláiban tanító tanárok listáját. A listában az egyes nevekre kattintva eléri az adott tanár oldalát, és mivel rendelkezik a tanári jogkörrel, így azokat a módosításokat, amelyeket a tanár megtehet, ő is elvégezheti. Ezen túl joga van új tanárt is felvinnie az adatbázisba. Az *Új tanár* hivatkozásra kattintva egy új ablak nyílik meg, ahol a tanár adatainak valamint a tanárhoz kapcsolódó azonosítónak és jelszónak megadásával rögzítheti az adminisztrátor az új tanárhoz kapcsolódó rekordot az adatbázisban. Ha sikeres volt az

adatok megadása, az adatbázisba bekerül az új tanárt reprezentáló rekord, a tanárokat tartalmazó lista automatikusan frissül, és a lista bővül az új tanár adataival.

2.9. ábra: Adminisztrátori oldal

Kurzusokkal kapcsolatos funkciók

A *Kurzusok* menüpontra kattintva az adminisztrátor keresési opciók megadásával kereshet a hozzá tartozó tanárok kurzusai között. Itt van lehetőség új kurzus felvitelére is. Új kurzus indításához meg kell adni a kurzus típusát, a kurzus időpontját, a kezdési és a tervezett befejezési dátumát, valamint, hogy melyik nyelviskolában indul be a kurzus és melyik tanár vezeti azt. Az új kurzus státusza automatikusan aktív lesz, ennek módosítására a tanárhoz kötődő funkcióknál leírt módon van lehetőség.

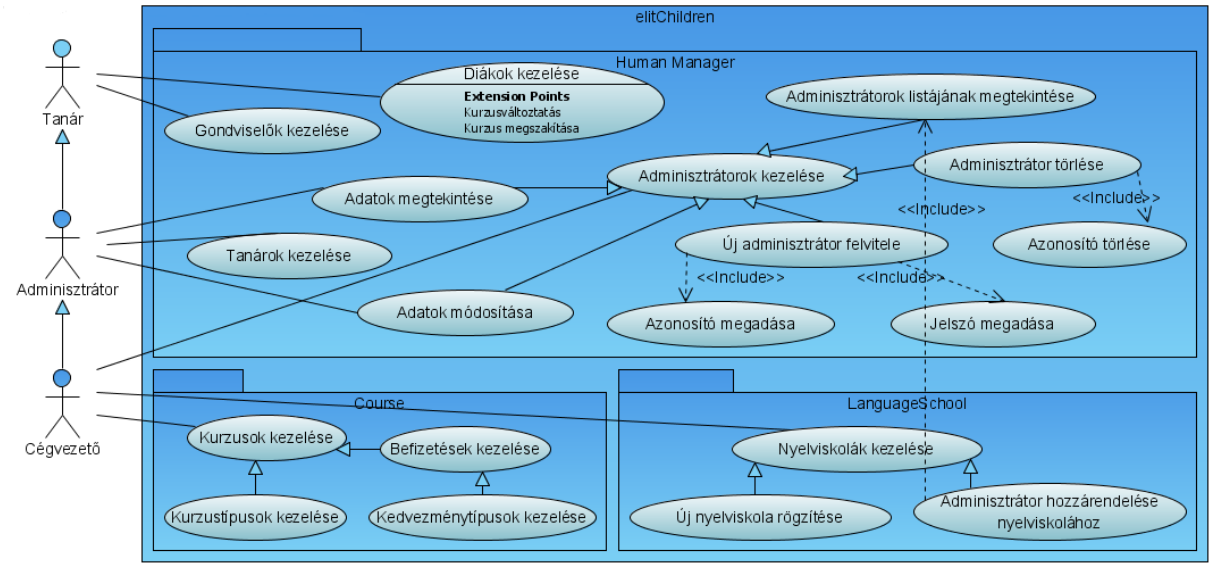
2.10. ábra: Keresés gyermekek között

Gyermekekkel kapcsolatos funkciók

A *Gyermekek* menüpontra kattintva az adminisztrátor különféle keresési opciók

megadásával kereshet az adatbázisban fent levő diákok között. A keresés eredménye egy lista, amely tartalmazza a keresési opcióknak eleget tevő gyermekek adatait. A listában a névre kattintva megjelennek az adott gyermek által látogatott kurzusok, amelyek közül választva a tanári oldalra jut el az adminisztrátor, ahol a keresett gyermek adott kurzusával kapcsolatos funkciók elérhetőek a számára.

2.1.5. Vezetői jogosultsággal elérhető funkciók:



2.11. ábra: Vezetői funkciókat részletező használati eset diagram

Adminisztrátorokkal kapcsolatos funkciók

A vezető az alkalmazásba való belépése után látja a nyelviskolákat vezető adminisztrátorok listáját, ahol az egyes nevekre kattintva eléri az illető adminisztrátor oldalát, és ott az adminisztrátori jogkör szerinti lehetséges módosításokat ő is elvégezheti. Ezen kívül új adminisztrátort vihet fel az adatbázisba az *Új adminisztrátor* hivatkozásra kattintva. Ekkor egy új oldal jelenik meg, ahol az új adminisztrátor adatait és az adminisztrátorhoz tartozó azonosítót és jelszót megadva lehet rögzíteni az adatbázisban az új adminisztrátorhoz kapcsolódó rekordot.

Nyelviskolákkal kapcsolatos funkciók

A *Nyelviskolák* menüpontra kattintva a cégvezető láthatja a nyelviskolák listáját. A listában szerepelnek az egyes nyelviskolák nevei és egyéb adatai, valamint, hogy az adott

nyelviskola melyik adminisztrátor hatásköre alá tartozik. Ha még nincs adminisztrátor hozzárendelve egy nyelviskolához, akkor az *Adminisztrátor* oszlopban a nyelviskolához tartozó sorban a *Felvesz* hivatkozás szerepel, melyre kattintva egy új ablakban jelenik meg egy az összes adminisztrátort tartalmazó lista, ahol választani lehet az adminisztrátorok közül, hogy ki legyen az adott nyelviskola felelőse. Lehetőség van módosítani is a nyelviskolák adatait a nyelviskolák listájának utolsó oszlopában szereplő *Módosít* hivatkozásra kattintva.

A cégvezető jogosult új nyelviskolák felvitelére is. A nyelviskolák listája alatt az *Új Nyelviskola* űrlapot kitöltve bővítheti az adatbázist újabb nyelviskolákkal. Az űrlapot elküldve megtörténik a bevitt adatok ellenőrzése, helyes adatok esetén bekerül az adatbázisba az új rekord, és frissül a nyelviskolák listája. A nyelviskola felvitelekor még nincs rögzítve, hogy melyik adminisztrátor a felelőse, ezt a felvitel után a nyelviskolák listájánál a korábban említett módon adhatja meg a cégvezető.

2.12. ábra: Vezetői oldal *(Nyelviskolák)*

Tanárokkal kapcsolatos funkciók

A cégvezető rendelkezik az adminisztrátori jogosultsággal, így az adminisztrátor tanárokkal kapcsolatos funkciói az ő számára is elérhetőek. A tanári oldalt a cégvezető az adminisztrátorokra kattintva közvetett módon elérheti, lehetőség van azonban közvetlenül is, a *Tanárok* menüpontra kattintva keresési opciók megadásával elérni a keresett tanár oldalát. A keresés eredménye az opcióknak eleget tevő tanárok listája, melyben a tanárok adatai kerülnek kilistázásra. A listában a keresett névre kattintva a cégvezető közvetlenül az illető tanár oldalára jut el.

Gyermekekkel és kurzusokkal kapcsolatos funkciók

Az adminisztrátoroknál említett módon a cégvezető is kereshet a kurzusok és a gyermekek között. Az eltérés csupán annyi, hogy amíg az adminisztrátor csak a hatásköre alá tartozó kurzusok és gyermekek között, addig a cégvezető az adatbázisban szereplő összes kurzus, és a kurzusokban résztvevő összes gyermek között kereshet.

Kedvezmény- és kurzustípusokkal kapcsolatos funkciók

A *Konfigurálás* menüpontra kattintva a cégvezető láthatja az adatbázisban szereplő kurzus- és kedvezménytípusok listáját. Az *Új kurzustípus* és az *Új kedvezménytípus* hivatkozásra kattintva a szükséges adatok megadásával új kurzus- illetve kedvezménytípusokkal bővítheti az adatbázist.

2.2 . Kivételkezelés és tranzakciókezelés

2.2.1. Kivételkezelés

Amikor egy JSP oldal fordítása közben kivétel következik be, alapértelmezés szerint a JSP konténer kapja el a kivételt, és a böngészőben jeleníti meg a kivétel üzenetét, és a verembejegyzéseket. A JSP technológia lehetőséget ad saját hibalapok definiálására, amellyel az alkalmazás használata során bekövetkező kivételek felhasználó-barát módon jeleníthetők meg. A saját hibalapokat az alkalmazás telepítésleírójában (a *web.xml* fájlban) lehet feltüntetni az *error-page* elemnél, ahol a hibalap elérési útját, és a hibalappal kezelendő hibák típusát lehet megadni. Hibalapot lehet definiálni a Java-kivételek típusai alapján, illetve a HTTP hibakódok alapján. Lehetőség van arra is, hogy bizonyos lapok saját hibalapot definiálhassanak maguknak: meg lehet adni az adott oldal *page* direktívájának *errorPage* attribútumában az oldal fordítása közben fellépő kivételek kezelésére megírt hibalapot, amely így eltérhet a *web.xml*-ben megadottól. (Az itt megadott hibalap felülírja a telepítésleíróban megadottat.)

Saját hibalapok fejlesztésekor a hibalap *page* direktívájában meg lehet adni az *isErrorPage="true"* attribútumot, amely jelzi a JSP fordító számára, hogy az adott oldal egy hibalap, és így a JSP oldalon belül használható az *exception* objektum, melynek segítségével elérhetjük a kiváltódott kivételt – lehetőség van így a kivétellel kapcsolatos információk

felhasználóbarát megjelenítésére.

Amennyiben a web-konténerhez jut egy kivétel, az megnézi, hogy van-e konkrétan az adott kivételhez hibalap definiálva. Amennyiben ilyet nem talál, keres tovább, megnézi, hogy van-e az adott kivétel szuperosztályát kezelő hibalap definiálva. Ezt egészen addig folytatja, amíg az öröklődési fában meg nem találja a megfelelő hibalapot vagy el nem éri a *Throwable* ősosztályt. Hibalapok segítségével tehát többszintű hibakezelést lehet megvalósítani.

Az *elitChildren* alkalmazás használ egy általános célú saját hibalapot, amely mindenféle kivételt elkap. Ezen kívül vannak specifikus hibalapok, például az adatbázissal kapcsolatos hibák kezelésére.

2.2.1. Tranzakciókezelés

A tranzakciókezelés kérdése főleg azokban az esetekben merül föl, amikor egyszerre több adatbázistáblában is kell változtatást végrehajtani. Ha a végrehajtás során probléma merülne föl, akkor a konzisztencia miatt az egész műveletet vissza kellene görgetni. Az *elitchildren.db.manager* csomagban található *PersistenceManager* osztály tartalmazza az adatbázis-műveletek végrehajtásához megírt metódusokat. A tranzakciókezelést a Hibernate végzi. Az *org.hibernate.Session* és az *org.hibernate.Transaction* interfész segítségével lehet megnyitni egy tranzakciót és lezárni azt. A *PersistenceManager* osztály *save()*, *update()* és *delete()* metódusai ezen interfészeket használva mind egy-egy tranzakcióként futnak le. Hiba bekövetkezésekor a Hibernate visszagörgeti a tranzakciót, és kivételt dob. Összetett műveletek esetén, például, amikor új diákot viszünk fel az adatbázisba, többször kellene meghívni a *PersistenceManager* osztály adatbázis-műveletet végző metódusát, ami több tranzakciót jelentene. Az alkalmazásban egy diák felvitelekor ugyanis egy tranzakcióban kell megoldani a gyermek és az ő gondviselője felvitelét és a gyermek, mint részvevő felvitelét az adott kurzusba. Mivel a tranzakciókezelés az adatelérési réteg része, nem akartam összevegyíteni az alkalmazási réteggel, így a szervletek szintjére már nem vittem fel a tranzakciókezelést. A Hibernate viszont lehetőséget ad olyan mechanizmus használatára, amivel ez a probléma kiküszöbölhető. A leképezési xml-fájlban a perzisztens osztályok közötti kapcsolatoknál kell jelezni ezt a Hibernate felé a *cascade* elem megadásával – így egyfajta szülő-gyerek viszony jön létre a két osztály között. A *cascade=save-update* alapján a szülő osztálybeli perzisztens objektum mentése illetve frissítése automatikusan eredményezi a

gyermek objektum mentését illetve frissítését, így elegendő egyszer meghívni a *PersistenceManager* osztály *save()* vagy *update()* metódusát, és ennek a tovagyűrűző mechanizmusnak köszönhetően egy tranzakcióként fognak lefutni az összetett műveletek is.

2.3 . A biztonság kérdése

2.3.1. Jogosulatlan hozzáférés

Az *elitChildren* a https protokollt használja az adatok biztonságos továbbítása érdekében. A https protokoll az SSL (Secure Socket Layers) eljárást használja, amely nyilvános kulcsú titkosításon alapul. A nyilvános kulcsokat tanusítványok formájában osztják ki erre feljogosított szervezetek. Ha a kiszolgáló tanusítványában szereplő adatok nem egyeznek meg a kiszolgáló gazdanevével, a felhasználóval a kapcsolat nem jön létre. A https használatával az adatok nem közönséges, hanem titkosított formában kerülnek továbbításra, jelentősen megnehezítve így lehallgatás esetén a lehallgatott adatok értelmezését.

2.3.2. Bejelentkezési autentikáció:

Az alkalmazásba csak felhasználói név és jelszó megadásával lehet bejelentkezni. Az alkalmazás bármely lapjához való hozzáférés esetén lezajlik egy hitelesítés, amit egy szervlet irányít. Bejelentkezéskor a szervlet meghívja az *elitchildren.authentication* csomag *AuthenticationManager* osztályának *getUser()* metódusát, amely az adatbázis segítségével ellenőrzi, hogy az adott felhasználói név szerepel-e az adatbázisban. Ha igen, akkor ellenőrzi, hogy a megadott jelszó egyezik-e a felhasználói névhez tartozó jelszóval. Ha valamelyik feltétel nem teljesül, akkor jelzi a hibát a felhasználónak, és a felhasználót ismét a bejelentkezési laphoz irányítja. Ha teljesül a feltétel, akkor megvizsgálja az adott felhasználói névhez kapcsolódó jogkört, és a jogkörhöz tartozó fő oldalra továbbítja a felhasználót, a meneti hatókörben pedig tárolja bean formájában a felhasználóval kapcsolatos információkat a későbbi hitelesítések miatt. Amennyiben a felhasználó az alkalmazás bármely oldalát megpróbálja elérni, újra lefut egy hitelesítés, amely ellenőrzi, hogy be van-e jelentkezve az adott felhasználó, és hogy van-e jogosultsága az adott oldal letöltéséhez. Ha be van jelentkezve és van megfelelő jogosultsága is, akkor a hitelesítést végző szervlet továbbítja a kérést az adott oldalnak, ellenkező esetben a bejelentkezési oldalra kalauzolja a felhasználót.

3. A rendszer konfigurálása

A rendszer fejlesztésekor szükséges volt megfelelő konfigurációs fájlok beállítása többek között az alkalmazásszerverhez, a Hibernate-hez és az Ant-hez. Ez a fejezet ezeknek a konfigurációs beállításoknak a rövid ismertetését tartalmazza.

3.1 . Az alkalmazás telepítésleírója

Minden Java alapú webes alkalmazás esetén az alkalmazásszerver számára egy *web.xml* nevű úgynevezett telepítésleíró fájlban kell elhelyezni az alkalmazással kapcsolatos konfigurációs beállításokat. A NetBeans egy új webes alkalmazás létrehozásakor automatikusan legenerálja ezt a fájlt, amit a *WEB-INF* könyvtárban helyez el. Az alkalmazás könyvtárszerkezetében a legfelső szinten helyezkednek el az alkalmazáshoz kapcsolódó weblapok (HTML és JSP oldalak), illetve egyéb hozzájuk kapcsolódó fájlok (képek, videók, stb.). A *WEB-INF* könyvtárban találhatóak az egyéb típusú erőforrások (pl.: Java osztályok) fájljai és alkönyvtárai. Két fontos alkönyvtára van a *WEB-INF* könyvtárnak: a *lib* és a *classes*. A *lib* könyvtárban az alkalmazáshoz szükséges jar fájlok vannak elhelyezve, a *classes* könyvtárban pedig az alkalmazáshoz tartozó Java-osztályok class fájljai.

A telepítésleíróba be kell jegyezni az alkalmazásban használt szervleteket. Meg kell adni a szervlet nevét, annak a Java osztálynak a nevét, amelyik az adott szervletet írja le, illetve azt az URI-t, amely a szervletet azonosítja. NetBeans használata esetén egy új szervlet létrehozásakor automatikusan történik a szervlet bejegyzése a *web.xml* fájlba. Az alkalmazás kurzus végét figyelő szervletje például a következőképpen kerül bejegyzésre:

```
<servlet>
    <servlet-name>FinishingDateListenerServlet</servlet-name>
    <servlet-class>elitchildren.listener.FinishingDateListenerServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>FinishingDateListenerServlet</servlet-name>
    <url-pattern>/FinishingDateListenerServlet</url-pattern>
</servlet-mapping>
```

Mivel a fenti szervletnek az alkalmazás indításakor már be kell töltődnie, hogy a *FinishingDateListener* osztály *TimerTask*-ját meghívva biztosítsa a kurzusok befejezési

dátumának figyelését, ezért szerepel a *<load-on-startup>* elem, amely jelzi az alkalmazásszerver felé, hogy az adott szervert a megadott prioritás alapján (jelen esetben ez 1) töltse be az alkalmazás indításakor.

Lehetőség van JSP oldalakat is bejegyezni a telepítésleíróba, így saját URI-kat definiálhatunk az egyes JSP oldalak alkalmazáson belüli elérésére. Ebben az esetben a bejegyzés teljesen hasonló a szervleteknél alkalmazott technikával, annyi csupán a különbség, hogy a szervlet Java osztályának megadása helyett a JSP oldal elérési útját adjuk meg:

```
<servlet>
   <servlet-name>director</servlet-name>
   <jsp-file>/director_login/director.jsp</jsp-file>
</servlet>
<servlet-mapping>
   <servlet-name>director</servlet-name>
   <url-pattern>/director</url-pattern>
</servlet-mapping>
<servlet-mapping>
   <servlet-name>director</servlet-name>
   <url-pattern>/director/*</url-pattern>
</servlet-mapping>
```

Webes alkalmazások Java környezetben történő fejlesztése esetén van lehetőség úgynevezett szűrő használatára. Ennek lényege, hogy minden egyes dinamikus vagy statikus erőforrás szolgáltatása előtt, illetve után végrehajtódhat. Az *elitChildren* alkalmazás használ szűrőt abból a célból, hogy a karakter-kódolás a JSP oldalaknál megfelelő legyen.

A telepítésleíróban a filter elemmel lehet jelezni az alkalmazásszerver felé a használni kívánt szűrőt. Itt meg kell adni a szűrő nevét, az őt implementáló osztályt, azt az url-mintát, amelyre illeszkedő oldalakra vonatkozzon a szűrő és esetlegesen az inicializáló paramétereket:

```
<filter>
   <filter-name>SimpleFilter</filter-name>
   <filter-class>elitchildren.util.SimpleFilter</filter-class>
   <init-param>
     <param-name>encoding</param-name>
     <param-value>ISO-8859-2</param-value>
   </init-param>
</filter>
<filter-mapping>
```

```
    <filter-name>SimpleFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

Ezen túlmenően a telepítésleíróban kell jelezni az alkalmazásszerver felé az alkalmazásban használt egyedi tag függvénytárakat az URI és az elérési út megadásával. A JSP oldalakon később az itt megadott taglib-uri segítségével lehet hivatkozni az adott tag függvénytárra.

```
<jsp-config>
    <taglib>
      <taglib-uri>/usertaglibs</taglib-uri>
      <taglib-location>/WEB-INF/user-taglibs.tld</taglib-location>
    </taglib>
</jsp-config>
```

A telepítésleíróban lehet jelezni saját hibalapok használatát a hiba jellegének és a hibalap URI-jának megadásával. A hiba lehet Java-kivétel, ekkor az *exception-type* nevű elemmel lehet megadni a kivétel nevét, illetve lehet HTTP hiba, ekkor az *error-code* nevű elemmel lehet megadni a hibakódot.

```
<error-page>
    <exception-type>java.lang.Exception</exception-type>
    <location>/error.jsp</location>
</error-page>
<error-page>
    <error-code>404</error-code>
    <location>/http_error.jsp</location>
</error-page>
```

Biztonsági mechanizmusok használata esetén pedig jelezni kell a mechanizmus típusát, a védendő erőforrásokat és egyéb biztonsággal kapcsolatos információkat.

Az *elitChildren* alkalmazás SSL-t használ az adatok biztonságos adattovábbítására, a bejelentkezés pedig form alapú, ennek megfelelően a biztonsági megszorítások a telepítésleíróban a következőképpen néznek ki:

```
<security-constraint>
    <web-resource-collection>
      <web-resource-name>elitChildren</web-resource-name>
      <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <user-data-constraint>
```

```
      <transport-guarantee>CONFIDENTAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>
<login-config>
    <auth-method>FORM</auth-method>
    <form-login-config>
      <form-login-page>/login</form-login-page>
      <form-error-page>/login</form-error-page>
    </form-login-config>
</login-config>
```

3.2 . A Hibernate és az Ant konfigurálása

Ahhoz, hogy a Hibernate tudja, hogyan töltse be és tárolja a perzisztens osztályok objektumait, szükség van egy leképezésre. Ezt a leképezést a Hibernate-ben egy XML fájllal lehet megadni. Ez a fájl adja tudtára a Hibernate-nek, hogy az adatbázisban mely táblákat mely Java-osztályoknak kell megfeleltetni. (Azt, hogy ebben az XML fájlban milyen XML-elemeket lehet használni, a Hibernate keretrendszert tartalmazó jar állományban található DTD fájl írja le.)

Minden egyes perzisztens osztályhoz szükséges megadni egy ilyen leképezési fájlt. Az objektum-relációs megfeleltetésen túl ez a fájl egyéb információkat is szolgáltat a Hibernate-nek a perzisztens osztályokkal kapcsolatban. Itt lehet például az előző fejezetben említett tovagyűrűző mechanizmussal kapcsolatos beállításokat is megadni.

A gyermekekhez tartozó XML fájl például így néz ki:

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
      "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="elitchildren.db.model.Student" table="student" lazy="false">
        <id name="id" type="long" column="id" unsaved-value="null">
                <generator class="sequence">
                        <param name="sequence">student_id_seq</param>
                </generator>
         </id>
        <property name="name" type="string" column="name" length="200" not-null="true"/>
        <property name="birthDate" type="date" column="birth_date" not-null="true"/>
```

```
        <property name="healthProblem" type="string" column="health_problem" not-null="false"/>
        <property name="motherTongue" type="string" column="mother_tongue" not-null="true"/>
        <many-to-one name="caretaker" column="caretaker_id" class="elitchildren.db.model.Caretaker"
cascade="save-update" not-null="true"/>
   </class>
</hibernate-mapping>
```

A fenti XML dokumentum alapján a Hibernate megfelelteti az *elitchildren.db.model.Student* Java-osztálynak a *student* adatbázis-táblát. A dokumentum tartalmazza az osztály, illetve tábla egyéb tulajdonságainak, oszlopainak jellemzőit. A fenti példa alapján a *Student* osztálynak hat tulajdonsága van: az *id*, a *name*, a *birthDate*, a *healthProblem*, és a *motherTongue*, melyeknek típusa a *type* attribútum által meghatározott típus, valamint a *caretaker*, amely egy *Caretaker* objektumra hivatkozó referencia. Az xml dokumentumban található *many-to-one* elem jelzi a Hibernate-nek, hogy az attribútumként megadott perzisztens osztálynak – jelen esetben a *Caretaker* osztálynak – megfeleltetett adatbázistábla és a *student* tábla között 1:N kapcsolat áll fönn.

A *many-to-one* elemnél a *cascade=save-update* attribútum jelzi, hogy a *Student* objektum mentése, illetve frissítése során automatikusan végrehajtódik a kapcsolt *Caretaker* objektum mentése, illetve frissítése is. A *Caretaker* perzisztens osztályhoz tartozó leképezési xml fájlban is van egy *many-to-one* kapcsolat, amely a *registered_user* táblával kapcsolja össze a *caretaker* táblát.

Egy részlet a *Caretaker* perzisztens osztályhoz tartozó leképezési xml-ből:

```
<hibernate-mapping>
    <class name="elitchildren.db.model.Caretaker" table="caretaker" lazy="false">
        <!-- … -->
          <many-to-one name="registeredUser" class="elitchildren.db.model.RegisteredUser"
column="registered_user_id" cascade="save-update" unique="true"/>
    </class>
</hibernate-mapping>
```

Itt is szerepel a *cascade=save-update* attribútum, így egy *Caretaker* objektum mentésekor, illetve frissítésekor automatikusan mentésre, illetve frissítésre kerül a kapcsolt *RegisteredUser* objektum is. Tehát egy *Student* objektum mentésének a hatására egyrészt a hozzákapcsolt *Caretaker* objektum, másrészt a *Caretaker* objektumhoz kapcsolt *RegisteredUser* objektum is mentésre kerül egy tranzakción belül.

A leképezési xml-eken túl szükség van egy konfigurációs XML fájlra, amely

tartalmazza az adatbázis-kapcsolat kialakításához szükséges információkat (pl.: adatbázis neve, felhasználó neve, jelszava, stb.), valamint a leképezéshez használható XML fájlok elérési útját. Ezt a konfigurációs fájlt az alkalmazás elérési útjában kell elhelyezni.

Miután elkészültek a konfigurációs és leképezési beállítások, a leképezési xml-ek alapján az Ant segítségével automatikusan legenerálhatók a leképezés egyik oldalán álló Java osztályok és a másik oldalon álló adatbázisbeli táblák és adatbázis-objektumok. Ezután kerülhet sor a rendszer lefordítására, futtatására. Ahhoz, hogy az Ant segítségével mindezt el lehessen végezni, szükség van egy *build.xml* fájl előállítására, amely az ezen feladatok elvégzéséhez szükséges információkat tartalmazza. Ennek a fájlnak tartalmaznia kell egy projektet, illetve hozzá kapcsolódóan legalább egy targetet. A targetek közül egyet alapértelmezettként kell megjelölni. Egy target több tevékenységet (taszkot) összekapcsoló egység. A taszkok a legkisebb egységek, amelyek tulajdonképpen végzik a műveleteket. A NetBeans fejlesztői környezetbe be van építve az Ant, egy új projekt létrehozásakor az IDE automatikusan előállítja ezt a *build.xml* fájlt. Így csak ki kell egészíteni a megfelelő parancsokkal, hogy alkalmas legyen az adatbázisséma és a perzisztens Java-osztályok generálására is.

4. A rendszer felépítése

A rendszer felépítése követi az MVC tervezési mintát (Modell-View-Controller/Modell-Megjelenítő-Vezérlő). Az MVC egy klasszikus tervezési minta, amelyet gyakran használnak olyan alkalmazásokhoz, ahol fontos, hogy ugyanazt az adatot többféleképpen is meg lehessen jeleníteni. A tervezési minta az objektumokat három szerepkörbe csoportosítja: modell – az adat kezelésére, megjelenítő – az adat megjelenítésére, és vezérlő – olyan események kezelésére, melyek hatással vannak a modellezésre vagy a megjelenítésre. Ez az elkülönítés elősegíti a fejlesztés hatékonyságát, hiszen, ha módosítani kell később a rendszeren, akkor elég a módosítást csak az adott szerepkörben elvégezni, nem kell külön hozzányúlni mindegyikhez.

Az MVC tervezési mintát követve többféle gyakorlati megvalósítás közül lehet választani. Lehetőség van arra, hogy tiszta JSP lapokból álló alkalmazást készítsünk, ahol a megjelenítő és a vezérlő is JSP lapokból tevődik össze. Másik lehetőség, hogy csupán a megjelenítői szerepkörben használjuk a JSP lapokat, a vezérlői szerepkörben pedig a szervleteket. További lehetőségként EJB-komponenseket is beépíthetünk az alkalmazásba, ezzel bővítve a modell és a vezérlő szerepkört.

Először úgy fejlesztettem az alkalmazást, hogy mind a vezérlő, mind a megjelenítő csupán JSP lapokból állt össze. Ez előnyös volt abból a szempontból, hogy egyszerűen és gyorsan ki lehetett próbálni az alkalmazást, tehát a prototípus készítésére minden tekintetben megfelelő volt. Később azonban problémássá vált a hibakeresés – debuggolás – ennél a módszernél. Másrészt nem különült így el fizikailag olyan jól a két szerepkör (vezérlő, megjelenítő), ami a későbbi fejlesztést is megnehezíti. Ezért miután elkészült a prototípus, ötvöztem a JSP technológiát a szervletekkel. Mivel a tisztán JSP lapokból álló prototípust már eleve úgy fejlesztettem, hogy a megjelenítői és a vezérlői szerepkörbe tartozó JSP-lapok ne keveredjenek, hanem jól elkülönüljenek, így nem volt nehéz ötvözni a két technológiát, csupán a vezérlői szerepkörbe tartozó JSP lapokat kellett szervletekre cserélnem. Az alkalmazásban így a JSP lapok mind a megjelenítői, a szervletek pedig a vezérlői szerepkörbe tartoznak. Az elkülönülés ennél a módszernél magától adódik, hiszen a szervletek fizikailag a JSP lapoktól teljesen elkülönülve helyezkednek el. A szervleteknél – mivel azok maguk is Java osztályok – a hibakeresés a fejlesztői környezetben megszokott módon történik. A

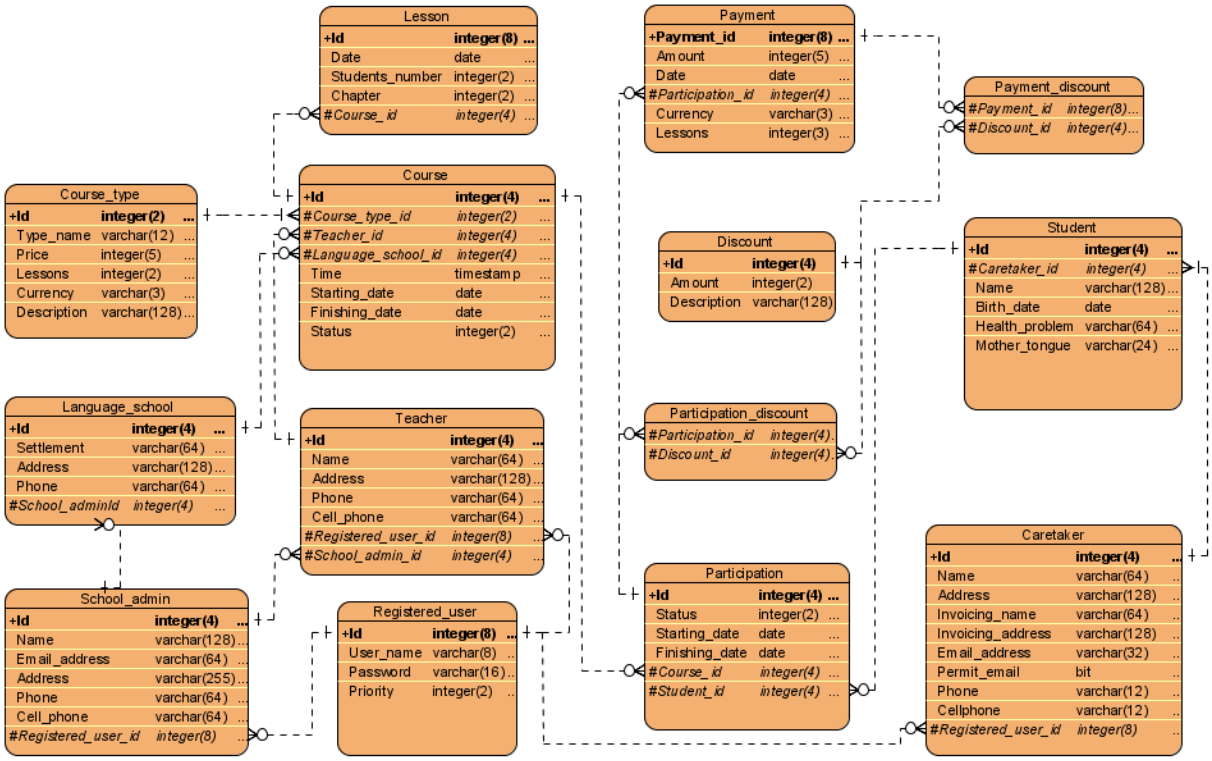
szervletek és a JSP lapok között a kommunikáció a bean-ek segítségével valósul meg.

A modell szerepkörbe tartoznak a Hibernate követelményeknek megfelelő perzisztens Java-osztályok és a különböző JavaBeans komponensek. A vezérlői szerepkörbe tartoznak a Java-szervletek, melyek az egyes JSP-lapok közötti vezérlést valósítják meg. A megjelenítői szerepkör fő elemei a JSP lapok, amelyek a szervletektől kapott bean-ek segítségével jelenítik meg a dinamikus tartalmat.

4.1 . Adatbázis-réteg:

Az adatbázis-réteg felelős az adatok perzisztens tárolásáért, ezzel biztosítva, hogy az alkalmazás használata során létrejövő új adatok és az adatokon elvégzett módosítások az alkalmazás bezárása után is megmaradjanak. Ebbe a rétegbe tartozik az adatbázis és az adatbáziskezelő-rendszer egésze.

Az alkalmazás a PostgreSQL adatbáziskezelő-rendszert használja, az alkalmazáshoz tartozó adatbázis pedig az *elitchildren* adatbázis. Az ebben az adatbázisban található adatbázis-objektumok a következők: adatbázis-táblák, szekvenciák és triggerek. Az adatbázisséma felépítését szemlélteti az alábbi ábra:



4.1. ábra: Az elitChildren alkalmazás adatbázissémája

4.1.1. Adatbázis-objektumok:

Adatbázistáblák: a táblák generálása a Hibernate leképezési xml-ek alapján történik.

Szekvenciák: a Hibernate biztosít olyan lehetőséget, hogy egy új perzisztens objektum létrehozásakor automatikusan generálódjon az új objektum egyedi azonosítója. Ezt a Hibernate szekvencia segítségével valósítja meg, ezeket a szekvenciákat a Hibernate automatikusan generálja. Az elitChildren alkalmazás ezt a mechanizmust veszi igénybe az adatbázisban szereplő táblák soraiban az elsődleges kulcs meghatározására.

Triggerek: triggerek segítségével lehetőség nyílik tárolt metódusok adott események bekövetkezésekor történő meghívására. Az *elitChildren* alkalmazásban triggerek segítségével oldottam meg azt, hogy tanfolyamok befejezési dátumának változtatásakor konzisztens módon változzon a tanfolyamon résztvevők esetében a kurzus elvégzésének várható dátuma. Hasonlóan, egy kurzus státuszának módosításakor trigger biztosítja, hogy konzisztens módon változzon a tanfolyamon résztvevő gyermekek státusza is.

A triggerek megvalósítása PostgreSQL használata esetén két lépcsőben történik. Először is meg kell írni magát a trigger-függvényt, ami majd az adott esemény bekövetkezésekor fog lefutni. A trigger-függvénynek nem lehetnek argumentumai és a visszatérési értéke trigger kell, hogy legyen. Miután elkészült a trigger-függvény, a *CREATE TRIGGER* utasítás segítségével létrehozhatjuk magát a triggert, amely az előbb megírt függvényhez kapcsolódik. Ugyanahhoz a trigger-függvényhez több trigger is kapcsolódhat. Lehetőség van utasítás- vagy sorszintű triggerek létrehozására (*FOR EACH ROW*, *FOR EACH STATEMENT*). Az utasításszintű trigger csak egyszer fut le, az őt kiváltó *INSERT*, *DELETE* vagy *UPDATE* utasításhoz kapcsolódóan, a sorszintű trigger viszont annyiszor, ahány sort érint az őt kiváltó utasítás. A trigger lefuthat a triggert kiváltó utasítás előtt vagy után is (*BEFORE*, *AFTER TRIGGER*). Az utasításszintű triggerek által meghívott trigger-függvények *NULL* értékkel térnek vissza. Egy esemény előtt lefutó sorszintű trigger esetében, ha a trigger-függvény *NULL* értékkel tér vissza, akkor nem fog végrehajtódni az adott sorra az utasítás, amely a triggert elindította, *INSERT* illetve *UPDATE* esetében pedig, ha a trigger az adatbázistábla egy sorával tér vissza, akkor ez a sor kerül beszúrásra illetve módosításra.

4.1.2. Adatbázis-objektumok generálása

Az adatbázis-réteg tehát két részből tevődik össze: az adatbázisból és az

adatbáziskezelő-rendszerből. Az adatbázis felépítése a Hibernate leképezési xml-ekben van rögzítve, ezek alapján lehet az Ant segítségével generálni a táblákat és a szekvenciákat. A Hibernate-nek köszönhetően így az adatbázis sémája nem az adatbáziskezelő-rendszerhez, hanem a Hibernate-hez kötődik. Adatbáziskezelő-rendszer cseréjekor tehát ugyanezekkel a leképezési xml-ekkel újragenerálható ugyanez az adatbázisséma. Adatbáziskezelő-rendszer cseréje esetén a Hibernate konfigurációs xml fájlját kell megfelelően módosítani, és az új adatbáziskezelő-rendszer JDBC-driverét elhelyezni a lib könyvtárban, a többi jar fájl mellé.

4.2 . Adatelérési- és alkalmazás-réteg:

4.2.1. Adatelérési-réteg:

Az adatbázis elérése a Hibernate segítségével történik. Az adatelérési-rétegbe tartozó Java osztályok az *elitchildren.db* csomagban találhatóak. Ez a csomag két további alcsomagra bomlik: *elitchildren.db.model* és *elitchildren.db.manager*. Az *elitchildren.db.model* csomagban vannak a Hibernate leképezési xml-ek, és a leképezéseknek megfelelő Java-osztályok. Az *elitchildren.db.manager* csomagban vannak a perzisztens objektumokkal dolgozó menedzser-osztályok. A *PersistenceManager* osztály tartalmazza az adatbázis eléréséhez szükséges metódusokat (többek között *query()*, *delete()*, *update()*, *save()*). Vannak még a csomagban olyan statikus metódusokat tartalmazó menedzser-osztályok, amelyek a *PersistenceManager* osztály metódusait használva manipulálják az egy-egy konkrét osztályhoz kötődő táblákban található adatokat. Ilyen osztály például a *StudentManager* osztály, amely tartalmaz minden, a tanfolyamokon résztvevő diákokkal kapcsolatos lekérdezést (pl.: diákok közti keresést adott kritériumok alapján, adott csoport diákjainak listázása, stb.), valamint a diákok adatbázisba történő beszúrását, frissítését illetve törlését megvalósító metódusokat. (Ez utóbbi három semmi egyebet nem csinál, csupán meghívja a paraméterben kapott diákra a *PersistenceManager* osztály megfelelő – *save()*, *update()* vagy *delete()* – metódusát.)

4.2.2. Alkalmazási réteg:

Az alkalmazási rétegbe tartoznak a szervletek, a JavaBeans komponensek, valamint az egyedi akcióelemek megvalósításához szükséges Java osztályok.

A háttérben a szervletek végzik az irányítást. Hozzájuk érkeznek be az adatok lekérésére, módosítására, törlésére irányuló kérések, vezérlik a hitelesítést, összeállítják a választ, amelyet továbbítanak a bean-ek segítségével. Az alkalmazásban mindegyik jogkörhöz tartozik egy-egy központi szervlet, amely elvégzi a szükséges hitelesítést, és az adott jogkörhöz kapcsolódó kérések feldolgozását. Ez a szervlet dönti el, hogy az adott kérést melyik jsp laphoz esetleg egy másik szervlethez továbbítsa. Így az alkalmazás bármely oldalához csak egy ilyen központi szervleten keresztül lehet hozzáférni (az illetéktelen hozzáféréseket elkerülendő). A rendszerbe történő bejelentkezéskor tehát attól függően, hogy az illető felhasználó milyen jogkörrel bír, más és más szervlethez kerül a vezérlés.

A felhasználói adatbevitel ellenőrzése JavaBeans komponensek segítségével történik. Az *elitchildren.presentation.validate* csomagban találhatóak az ellenőrzést végző bean-ek. A JSP oldalakon a *jsp:useBean* akció segítségével lehet használni ezeket a validációs bean-eket. Ahhoz, hogy az ellenőrzés automatikusan működhessen, fontos, hogy a bean-ekben szereplő tulajdonságok nevei megegyezzenek a JSP oldalon szereplő megfelelő űrlapelemek neveivel. Az űrlap kitöltése után, mikor a felhasználó elküldi az általa bevitt adatokat, lefutnak az űrlapelemek neveivel megegyező tulajdonságok beállító metódusai. (Abban az esetben, ha egy űrlapelemet nem töltött ki a felhasználó, nem fut le a megfelelő bean neki megfelelő tulajdonságának beállító metódusa.) A validációs bean-ek segítségével tehát felügyelni lehet a felhasználói bevitel helyességét, figyelmeztetni lehet a felhasználót a felmerülő hibákra. Az esetlegesen felmerülő hibákról a validációs bean-ekben külön lekérdező függvények szolgáltatnak információt. JSP oldalról ezeket a *jsp:getProperty* akció segítségével lehet lekérdezni.

Időzített taszk figyeli az aktív kurzusok befejezési dátumát, és azok lejártakor elvégzi a szükséges módosításokat. Ezt a taszkot az alkalmazás indításakor egy szervlet, a *FinishingDateListenerServlet* indítja el. (A telepítésleíróban a *load-on-startup* elem biztosítja, hogy az alkalmazás indításakor ez a szervlet is lefusson). Ez a szervlet hívja meg a *FinishingDateListener* osztály *start()* metódusát. (A *FinishingDateListener* a *TimerTask* absztrakt osztályt terjeszti ki.) A *start()* metódus meghívja a *Timer* típusú *timer* statikus változó *scheduleAtFixedRate()* metódusát. Ennek három paramétere van: az első egy *TimerTask*, ami az időzítésre kerülő tevékenységet írja le, a második paramétere *Date* típusú, amely a tevékenység első lefutásának idejét adja meg, a harmadik paraméter pedig *long*

típusú, amellyel milliszekundumokban lehet megadni, hogy milyen időközönként fusson le az adott tevékenység. Első paraméterként egy új *FinishingDateListener* objektumra hivatkozó referencia, másodikként az aktuális dátum, harmadik paraméterként pedig egy olyan *long* típusú érték kerül átadásra, amely egy nap hosszát adja meg milliszekundumokban mérve. Így tehát naponta kerül ütemezésre a *FinishingDateListener* osztály *run()* metódusának futtatása. A *run()* metódus futásakor történik meg a kurzusok befejezési dátumának ellenőrzése, és lejárat esetén a lejárt kurzusok státuszának módosítása.

Az alkalmazási rétegbe tartoznak továbbá a felhasználói rétegben használt egyedi akcióelemekhez (tag library) kapcsolódó Java-osztályok. Minden egyes egyedi akcióelemhez tartozik egy tag-kezelő Java-osztály, melynek implementálnia kell a *javax.servlet.jsp.tagext.Tag* interfészt. Ez a tag-kezelő osztály írja le, hogy mire is szolgál a hozzá tartozó egyedi akció. Egy tag függvénytárt leíró fájlban (TLD – Tag Library Descriptor) kell megadni minden egyes egyedi akció esetében a neki megfelelő tag-kezelő osztályt. Ebben a fájlban kell feltüntetni még az egyes egyedi akciók által támogatott összes attribútumot. Ahhoz, hogy használni tudjuk a JSP oldalakon az egyedi akciókat, szükséges még a *web.xml* fájlban is feltüntetni az egyedi akciók neveit és az őket azonosító uri-kat.

Az *elitChildren* alkalmazásban az egyedi akciók tag-kezelő osztályai az *elitchildren.usertaglibs* csomagban találhatóak.

4.3 . Felhasználói réteg

A felhasználói rétegbe tartozik mindaz, ami a felhasználó szeme előtt van, amivel közvetlenül érintkezik a rendszer használata során.

Az *elitChildren* alkalmazásnál az adatok megjelenítése JSP oldalak segítségével történik.

A JSP technológiánál nincs külön megkötés, hogy a megjelenítés konkrétan milyen eszköz segítségével történjen, lehet akár XML, akár XHTML vagy WML, stb. Az *elitChildren* alkalmazás XHTML segítségével jeleníti meg a tartalmat. Ezt minden egyes JSP oldalon jelezni kell a *page* direktíva *contentType* attribútumának megadásával. Ebben az esetben az attribútum értéke *"text/html"*.

A felület kialakításánál elsődleges szerepet töltött be az átláthatóság, valamint a főbb funkciók könnyű és gyors elérése.

Az alkalmazásban a megjelenített tartalom formázása CSS segítségével történik. Minden egyes oldalon, amely CSS-t használ, meg kell adni a CSS fájl elérésére vonatkozó információkat. A JSP technológia lehetőséget nyújt az *include* direktíva segítségével arra, hogy különálló JSP oldalakat fűzzünk össze, ezáltal megspórolva egyazon kódrészletek többszöri kézi előállítását. Így nem kell minden egyes JSP oldalon feltüntetni a CSS fájl elérési útját, hanem elég a fő oldalon ezt megtenni, azaz azon az oldalon, amely tartalmazza az esetlegesen betöltésre kerülő többi JSP oldalt (Természetesen ugyanez igaz például a JSP oldalak tartalomra vonatkozó információira – tartalom típusa, nyelvkészlete, stb.).

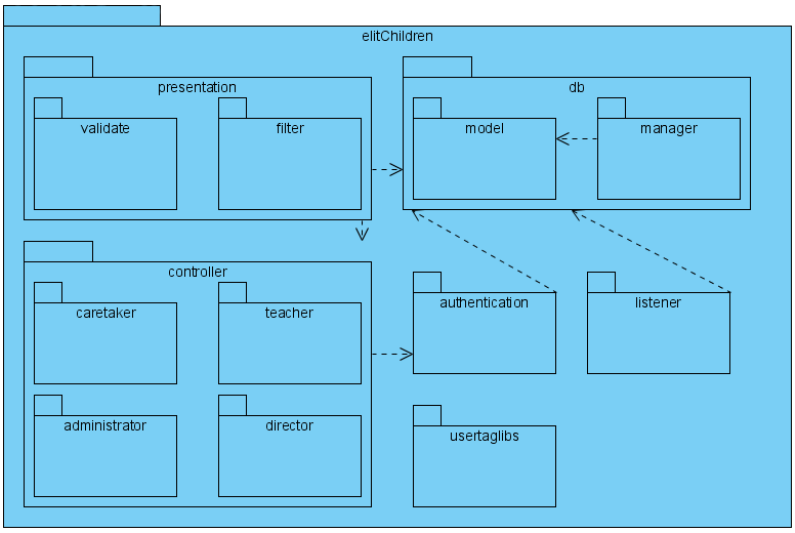
Ahhoz, hogy elkerülhető legyen a Java-kód használata a JSP oldalaknál, a JSP technológia lehetőséget nyújt JSP akcióelemek használatára. Az akcióelemek mindig valamilyen akciót hajtanak végre az aktuális információk birtokában. A standard akcióelemeken túl a technológia engedélyezi saját akcióelemek kifejlesztését, melynek segítségével elrejthetővé válik az alkalmazáson belül az üzleti logika.

Az *elitChildren* alkalmazás is használ saját akcióelemeket. Ezek az akcióelemek *elit* előtaggal rendelkeznek. Ahhoz, hogy egy JSP oldalon fel lehessen használni saját akcióelemeket, fel kell tüntetni az oldal elején egy összerendelést, amely megadja, hogy melyik előtaghoz melyik saját akcióelem-könyvtár tartozik. Az *elit* előtag például a következőképpen van összerendelve a hozzá tartozó akcióelem-könyvtár uri-jával: *<%@ taglib uri=”/usertaglibs” prefix=”elit” %>* Ezután a JSP oldalon már lehet használni az adott könyvtárban található akcióelemeket az elit előtag megadásával.

Az *<elit:iteration .../>* akcióelem például az attribútumban megadott kollekció (mely a *Collection* interfészt kell, hogy implementálja) elemeit járja be. Ezen kívül vannak kevésbé általános, a konkrét adatbázis-táblákat reprezentáló Java osztályokhoz kapcsolódó akcióelemek, melyek segítségével azonosító illetve egyéb tulajdonságok megadásával betölthetők az adatbázisbeli objektumok.

4.4 . A forráskód felépítése

4.4.1. Java-osztályok



4.1. ábra: Csomag diagram

A Java forráskód felépítését szemlélteti a 4.1-es ábra. Az ábrán láthatóak az alkalmazás csomagjai.

A *presentation* csomag a megjelenítéssel kapcsolatos Java-osztályokat tartalmazza, ennek egy alcsomagja a *validate* csomag, amely az űrlapok validációit végző bean-eket tartalmazza, egy másik alcsomagja pedig a *filter* csomag, amely az alkalmazás által használt *Filter* osztályokat tartalmazza.

A *db* csomagban az adatbázissal kapcsolatban lévő Java-osztályok találhatóak, a *model* alcsomag tartalmazza a Hibernate-hez készített perzisztens osztályokat, a *manager* alcsomag pedig ezeknek az osztályoknak a menedzser-osztályait.

A *controller* csomagban találhatóak a vezérlést végző Java-szervletek. Ezek a jogosultságok alapján külön alcsomagokra bomlanak: *caretaker*, *teacher*, *administrator* és *director*.

Az *authentication* csomag az autentikációval kapcsolatos Java-osztályokat tartalmazza.

A *listener* csomagban találhatóak az események figyelését végző Java-osztályok (mint a kurzusok befejezési dátumát figyelő Java-osztályok).

A *usertaglibs* csomagban találhatóak az egyedi akcióelemek Java-osztályai.

4.4.2. JSP lapok

A JSP lapok az alkalmazás főkönyvtárában helyezkednek el. Az alkalmazás induló oldala az *index.jsp* lap, amely betöltéskor automatikusan továbbítja a kérést a *login* könyvtárban található *login.jsp* lapnak.

Külön könyvtárakban találhatóak a különböző jogosultságokhoz tartozó JSP oldalak (*caretaker_login*, *teacher_login*, *admin_login*, és *director_login*). Ezeknek a könyvtáraknak a felépítése egyforma: a *forms* alkönyvtárban találhatóak az adott jogosultsághoz kapcsolódó űrlapok, a *menu* alkönyvtárban pedig az adott jogosultság alapján elérhető funkciókkal kapcsolatos JSP oldalak.

Az alkalmazás által használt képek az *images* könyvtárban, a JSP oldalakon használt CSS-lapok pedig a *css* könyvtárban helyezkednek el.

5. A továbbfejlesztés irányvonala

5.1 . A publikus rész kifejlesztése

Az elkészült webes alkalmazás a céggel már kapcsolatban lévő személyek számára nyújt segítséget – a cégben dolgozók és a cég szolgáltatásait igénybe vevők számára. A cégnek viszont természetesen rendkívül fontos, hogy minél többen megismerhessék a szolgáltatásait, és gyarapodhasson a szolgáltatást igénybe vevők száma. Erre az alkalmazás nem alkalmas, hiszen azt csak regisztrált felhasználók használhatják. A problémára a portál mindenki számára elérhető, publikus része nyújthat megoldást. A továbbfejlesztés tehát elsősorban ennek a mentén kell, hogy haladjon. A publikus rész többnyire statikus oldalakból fog állni, amelyek a céggel kapcsolatos fontosabb aktuális információkat közlik majd a portált látogató felhasználókkal. A tervek között szerepel azonban az is, hogy az alkalmazás adatait felhasználva dinamikus tartalmak is megjelenjenek a publikus részben. Így például az újonnan induló kurzusokról dinamikus oldalakon értesülhetnek majd az érdeklődők, valamint az aktuális kurzus-, illetve kedvezménytípusok és kurzusdíjak is az adatbázisból kinyert adatok alapján kerülnek majd fel a portálra. Az is a tervek között szerepel, hogy a kurzusokban résztvevő gyermekek gondviselői a saját felhasználói oldalukon írhassanak az oktatással kapcsolatos benyomásaikról, és ezek a hozzászólások megjelennek majd a mindenki számára elérhető publikus részben is.

5.2 . Az alkalmazás továbbfejlesztése

Az alkalmazás továbbfejlesztésének irányvonalát elsősorban nyilván az alkalmazást használók benyomásai, igényei, elvárásai határozzák meg. Azonban a fejlesztés közben is felmerültek olyan fontos funkciók, amelyek jelentősen megkönnyítik az alkalmazás használatát. Ezek közül némelyik már megvalósult, sok funkció megvalósítása azonban idő híján későbbre halasztódott. A továbbfejlesztéssel kapcsolatos igények tehát részben saját elgondolás, részben pedig az adminisztrátorral történt egyeztetések alapján születtek meg.

Tervbe van véve olyan órarend elkészítése a tanári oldalon, amelyben napokra és órákra bontva jelennek meg az illető tanár által vezetett kurzusok időpontjai, megkönnyítve

ezzel a tanár számára a saját kurzusainak az átlátását.

Az alkalmazás használata során többször is előfordul, hogy a felhasználó a tanárok, kurzusok, illetve gyermekek listájában böngészve keresi a számára lényeges információkat. A keresés hatékonyságának növelését célozza meg az a terv, hogy ezen listák esetében lehetőség nyíljon többféle rendezési elv alkalmazására is. Jelenleg egyféle rendezési elv – a betűrend – alapján kerülnek egymás után a listába a rekordok. A továbbfejlesztett változatban lehetőség lesz arra, hogy a táblázat oszlopaira kattintva lehessen választani az adott oszlop szerinti növekvő, illetve csökkenő rendezést. Ezen túlmenően előfordulhat, hogy a lista mérete olyan nagy, hogy a teljes listát egy oldalra bezsúfolva jelentősen csökken az átláthatóság. A tervek szerint, ha a lista mérete meghalad egy bizonyos korlátot, akkor a lista több lapra darabolódik, és az oldalak közt lapozva lehet majd bejárni a teljes listát.

Jelenleg egy kurzusba járó gyermekek befizetéseit egyenként, a gyermekek listájában az egyes gyermekekre kattintva lehet megtekinteni. A tervek között szerepel az is, hogy egy oldalon összegezve is megjelenjenek egy adott kurzusba járó gyermekek összes addigi befizetései. Ezen túlmenően az is a továbbfejlesztés része, hogy a nyomtatás elősegítése céljából a listák (kurzusok, gyermekek, tanárok listája) esetében lehessen nyomtatási képet kérni, amely dekorációk nélkül, csak az adott listát és a listához szorosan kapcsolódó információkat tartalmazza. A listák xsl formátumban történő mentésére is lehetőség lesz.

Felmerült olyan igény is, hogy a nyelviskoláknál olyan plusz információ is megjelenhessen, mint például, hogy egy iskolánál mennyi eladó oktatócsomag van raktáron.

A kurzusokkal kapcsolatosan pedig igény van folytatólagos kurzus bevezetésére, azaz arra, hogy egy kurzus befejezése után legyen lehetőség a kurzus folytatására úgy, hogy a folytatásként megjelölt kurzusba ne kelljen újra egyenként felvenni a gyermekeket, hanem azt megadva, hogy az új kurzus melyik korábbi kurzusnak a folytatása, legyen automatikus a korábbi kurzusban résztvevő gyerekek új kurzusba történő felvitele.

A fenti igények kielégítésén túl még a továbbfejlesztés fontos része a befizetésekkel kapcsolatos figyelmeztetési mechanizmus beépítése a rendszerbe. Ez két oldalú lesz, egyrészt az adminisztrátorokat fogja figyelmezteti az esedékes illetve elmaradt befizetésekről az alkalmazáson keresztül, másrészt a gondviselőket is figyelmezteti majd emailben az esedékes vagy elmaradt befizetéseikről.

Összefoglalás

A fejlesztés

Az *elitChildren* alkalmazás fejlesztése sok olyan problémát vetett föl, melyeknek a megoldása révén hasznos tapasztalatokat szerezhettem a dinamikus weboldalak fejlesztésének területén. A gyakorlatban próbálhattam ki és ismerhettem meg a fejlesztéshez alkalmazott technológiákat, a tervezés és a megvalósítás során pedig életszerű problémákkal kerülhettem szembe.

Voltak olyan területek a fejlesztés során, amelyek a tervezettnél több időt vettek igénybe. Ilyen volt például a Hibernate konfigurációs beállításaival kapcsolatos terület. A Hibernate hibaüzenetei ugyanis nem mindig voltak kielégítőek, nem minden esetben lehetett azok alapján olyan egyszerűen megtalálni a hiba okát. Nagy segítséget nyújtott ebben a Hibernate fóruma, ahol a Hibernate-tel kapcsolatos problémák és az azokra adott megoldások között lehetett böngészni.

Megvalósult és megvalósításra váró célok

A portállal kapcsolatos célok közt az első volt a cégben dolgozók munkájának a megkönnyítése a nyilvántartás területén. Az *elitChildren* alkalmazás kifejlesztésével ezt a célt sikerült elérni; az alkalmazás több nyilvántartással kapcsolatos funkciót is tartalmaz, amelyeknek használata jelentős segítséget nyújthat a cég alkalmazottai számára.

A célok közül megvalósításra vár a céggel, illetve annak szolgáltatásaival kapcsolatos információk szolgáltatása az érdeklődők számára. Ezt a célt a portál publikus része fogja betölteni. A célok közt szerepelt a szolgáltatást igénybe vevőkkel történő kapcsolattartás is, amely részben valósult meg: a kurzusokban résztvevő gyermekek gondviselőik az alkalmazáson keresztül értesülhetnek a kurzusokkal kapcsolatos friss információkról. A kapcsolattartás részeként van tervbe véve az 5. fejezetben említett figyelmeztetési mechanizmus is, valamint az is, hogy a gondviselőknek lehetőségük legyen a cég általuk igénybe vett szolgáltatásaival kapcsolatos benyomásaikat, tapasztalataikat megosztani az alkalmazáson keresztül.

Technológiák

A fejlesztéshez alkalmazott technológiák kiválasztásánál a fő szempont az ingyenesség és a platform-függetlenség volt. Az *elitChildren* alkalmazás így például egyaránt telepíthető Linux-os és Windows-os környezetre is, anélkül, hogy a forráskódon bármiféle módosítást kellene végrehajtani.

Funkcionalitás

A rendszer funkcióinak kialakításakor az elsődleges szempont a rendszerrel kapcsolatos követelményeknek, illetve elvárásoknak való megfelelés, a másodlagos szempont pedig a funkcióikat igénybe vevő felhasználók munkájának megkönnyítése volt. Így a fejlesztés során először az alapvető funkciókat építettem bele a rendszerbe, azokkal készítettem el a prototípust, majd ezeket további hasznos funkciókkal bővítettem. Ilyen funkció például az, hogy különféle keresési opciók beállításával lehet keresni gyermekek, kurzusok, tanárok között, és így azok közvetlenül elérhetővé válnak. A továbbfejlesztés során a rendszer funkcionalitása az a terület, ami a leghangsúlyosabb, így további újabb funkciókkal bővül majd a rendszer.

Irodalomjegyzék

[1] Hans Bergsten: JavaServer Pages
Kossuth kiadó, 2001

[2] Nyékyné Gaizler Judit: J2EE útikalauz Java programozóknak
ELTE TTK Hallgatói Alapítvány, Budapest, 2002

[3] Neil Bradley: Az XML kézikönyv
Szak Kiadó, 2005

[4] Christian Bauer, Gavin King: Hibernate in Action
Manning Publications Co., 2005

[5] Hibernate Reference Documentation, 2007
http://www.hibernate.org/hib_docs/v3/reference/en/html_single/

[6] Viczián István: Apache Ant, Java-based build tool (cikk), 2004
http://www.codexonline.hu/CodeX11/16.pdf,

[7] Kozsik Tamás: Szervletek, 2007
http://aszt.inf.elte.hu/~kto/teaching/java/material

[8] JavaScript dokumentáció, 2007
http://www.intermedia.c3.hu/javascript/main.html

[9] Java programozási nyelv, Wikipédia, a szabad lexikon, 2007
http://hu.wikipedia.org/wiki/Java_programozási_nyelv

Függelék

A Hibernate konfigurációs fájlja az elitChildren alkalmazáshoz:

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

    <session-factory>

        <!-- Database connection settings -->
        <property name="connection.driver_class">org.postgresql.Driver</property>
        <property name="connection.url">jdbc:postgresql://localhost/elitChildren</property>
        <property name="connection.username">echildren</property>
        <property name="connection.password">password</property>

        <!-- SQL dialect -->
        <property name="dialect">org.hibernate.dialect.PostgreSQLDialect</property>

        <!-- Echo all executed SQL to stdout -->
        <property name="show_sql">true</property>

        <mapping resource="elitchildren/db/model/Student.hbm.xml"/>
        <mapping resource="elitchildren/db/model/Teacher.hbm.xml"/>
        <mapping resource="elitchildren/db/model/Course.hbm.xml"/>
        <mapping resource="elitchildren/db/model/Caretaker.hbm.xml"/>
        <mapping resource="elitchildren/db/model/SchoolAdmin.hbm.xml"/>
        <mapping resource="elitchildren/db/model/Sybling.hbm.xml"/>
        <mapping resource="elitchildren/db/model/LanguageSchool.hbm.xml"/>
        <mapping resource="elitchildren/db/model/CourseType.hbm.xml"/>
        <mapping resource="elitchildren/db/model/Payment.hbm.xml"/>
        <mapping resource="elitchildren/db/model/Discount.hbm.xml"/>
        <mapping resource="elitchildren/db/model/Lesson.hbm.xml"/>
        <mapping resource="elitchildren/db/model/Participation.hbm.xml"/>
        <mapping resource="elitchildren/db/model/RegisteredUser.hbm.xml"/>

    </session-factory>

</hibernate-configuration>
```

Részlet az alkalmazáshoz készített build.xml fájlból

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="elitChildren" default="default" basedir=".">
    <description>Builds, tests, and runs the project elitChildren.</description>
    <import file="nbproject/build-impl.xml"/>
    <property name="build" value="build"/>
    <property name="dist" value="dist"/>
    <property name="classes" value="${build}/web/WEB-INF/classes"/>
    <property name="src" value="src"/>
    <property name="lib" value="lib"/>
    <path id="classpath">
        <fileset dir="${basedir}">
              <include name="**/*.jar"/>
        </fileset>
        <pathelement location="${basedir}/src"/>
            <pathelement location="${classes}"/>
    </path>
    <!-- For generating database schema -->
    <target name="schemaexport">
        <taskdef name="schemaexport"
            classname="org.hibernate.tool.hbm2ddl.SchemaExportTask"
            classpathref="classpath"/>
        <schemaexport
            properties="${src}/hibernate.properties"
            quiet="no"
            text="no"
            drop="no"
            delimiter=";"
            output="${src}/schema-export.sql">
            <fileset dir="${src}">
                <include name="**/*.hbm.xml"/>
            </fileset>
        </schemaexport>
    </target>

<!-- … -->

</project>
```