Debreceni Egyetem

Informatikai Kar

Web alapú környezet fejlesztése

Iskolai adminisztrációs rendszer

Témavezető
Dr. Almási Béla
Egyetemi Docens

Készítette
Bodnár József
Programtervező Informatikus MSc

Debrecen

2009

Tartalomjegyzék

1. fejezet

Bevezetés

Diplomamunkám témája: Web alapú környezet fejlesztése - Iskolai információs rendszer. Miért is választottam ezt a témát? Több okból is. Mielőtt jelenlegi munkahelyemen kezdtem el dolgozni, mint pedagógus szoftverfejlesztéssel foglalkoztam. Abban az időben a kétrétegű kliens-szerver architektúrájú alkalmazásokat fejlesztettek. Azóta a technológia sokat fejlődött, így ez az elgondolás, vagyis a 2 rétegű kliens szerver architektúra már "elavultnak tekinthető" - hogy az egyik fejvadász cégnél történt interjún elhangzott egyik kijelentést idézzem: "old school módszer". Bár nagy tapasztalatom van a szoftverfejlesztés területén, az a fent említett technológiára vonatkozik, melyet már csak ritkán használnak, így ez elég nagy hátránynak tekinthető a munkaerőpiacon. Így szükségességét éreztem ismereteim felfrissítésének. A téma választása lehetőséget biztosít számomra, hogy megismerjem ezen új, modern szoftverfejlesztési technológiákat. Miért pont iskolai információs rendszer? Jelenleg a közoktatásban dolgozó pedagógusokra, és egyéb dolgozókra rendkívül nagy adminisztrációs feladat hárul, melyet jelenleg minden dolgozó saját módszerei szerint - kézzel vagy csekély mértékben informatikai megoldásokkal megtámogatva - végez. Ez rendkívül időigényes feladat. Sokszor nem fér bele a heti 40 órás kötelező munkaidőbe, néha a tanórákra való felkészülés rovására megy, így romolhat az oktatás minősége és akkor még nem is említettük a dolgozók szabadidejéből elvett időt. Jelenleg az iskolai dolgozók ilyen jellegű munkája nincs megfelelően támogatva

informatikai eszközökkel. Bár léteznek ezen feladatok ellátására kifejlesztett szoftverek, ezek túl bonyolultak összetettek és nagyon drágák. Bár interneten valamiért nem publikálják ezen szoftverek árait - talán az "érdekes" közbeszerzési eljárások miatt-, a sok-sok keresés meghozta eredményét. Példaként álljon, itt az állam által preferál Taninform rendszer. Kaposvár város közgyűlése 2008-ban 13 922 400 Ft-ot fizetett a rendszerért. Ezek az árak nagyon magasak egy közoktatási intézmény számár, és nem feltétlenül szükséges ilyen összetett szoftver alkalmazása. Az említett Taninform rendszer valamilyen szinten fel van készítve bérszámfejtésre is. Miért fizessen ilyen funkcióért az intézmény mikor a bérszámfejtést amúgy is a TÁKISZ [1] végzi? Azon túlmenően, hogy sok, a mi intézményünk számára felesleges funkciók vannak ezekben a szoftverekben, bizonyos feladatokra, feladatkörökre nem megfelelően, vagy egyáltalán nincsenek felkészítve.

1.1. Intézményi háttér

A szoftver egy Szabolcs-Szatmár-Bereg megyei középiskola tantestületének készül. Az intézmény kb. 900 fős tanulói létszámmal és 80 fős pedagógus dolgozóval rendelkezik. A képzés szakközép- és szakiskolai szinten folyik. Szak- illetve szakközépiskola után lehetősége van a szakiskolai tanulóknak érettségit szerezni, illetve a szakközép iskolai tanulóknak a technikus képzésben részt venni. Emellett egyre nagyobb szerephez jut az intézményben a felnőttképzés. Az adminisztrációs feladatok talán a felnőtt képzést kivéve mindenütt jelentős erőforrásokat igényelnek. Az intézményben komoly informatikai képzés folyik 1993 óta. Informatikai eszközökkel rendkívül jól felszerelt és jól felkészült szakemberekkel ellátott. Kb. 300 számítógép, 30 notebook, 10-12 jól kvalifikált informatikai tanár és szakember található az intézményben. Mindezek figyelembevételével adódik az ötlet, hogy ilyen informatikai háttérrel és szakember gárdával talán készíthetnénk egy saját adminisztrációs rendszert, amely kielégíti igényeinket, nem tartalmaz felesleges funkciókat, melyek megnehezítik a vele való munkát és figyelembe veszi a felhasználói

[1] Területi Államháztartási és Közigazgatási Információs Szolgálat

hátteret és annak megfelelően alakítja ki a szoftver szerkezetét. Sajnos ez az elgondolás végül kivitelezhetetlennek bizonyult.

1.2. Célkitűzés

A diplomamunkával a célom hogy a kollégáim számára egy könnyen használható, megbízható szoftvert fejlesszek ki, mely megkönnyít a midennapi munkájukat. Másrészt cél volt egy olyan rendszer kialakítása mely későbbiekben könnyen módosítható, továbbfejleszthető, mely nem gördít akadályokat a szoftverevolúció elé. A fentiek biztosításához a Java EE környezetet választottam, mely környezet a kezdetektől e kritériumokat szemelőt tartva lett fejlesztve. Ezzel lehetőségem nyílott a Java EE technológia megismerésére, mellyel olyan technológiai tudásra tettem szert, mely naprakész piacképes tudást biztosít számomra, melynek segítségével bárhol eladhatom tudásom a munkaerőpiacon. Nem csak Magyarországon, hanem külföldön is. Jó alapot szolgáltat ez a projekt arra, hogy a magam elé célként kitűzött nemzetközi szinten elismert Sun SCJD[2] vizsgához felkészülést biztosítson.

[2]http://www.sun.com/training/certification/java/scjd.xml

2. fejezet

A fejlesztés menete

Mivel egy teljesen új rendszerről van szó, illetve nincs konkrét megrendelő, így teljesen az elképzeléseink szerint alakíthattuk a fejlesztés menetét, a felhasznált technológiákat és módszereket. A fejlesztés menetének megtervezésében nagy segítségemre volt Ian Sommerwille: Szoftverrendszerek fejlesztése című könyve. A szoftverfolyamata során a vízesés modell alkalmazása mellet döntöttem. A modell lapvető szakaszai alapvető szoftverfejlesztési lépésekre képezhetők le (2.1. ábra)[1]:

1. Követelmények elemzése és meghatározása: a rendszer szolgáltatási céljai, megszorításai a felhasználókkal konzultálva alakul ki (ez nem történhetett meg az én esetemben, bár én is a rendszer egyik felhasználója vagyok). Ezt később részletesen kifejtik, és később ezek szolgálják a rendszerspecifikációt.

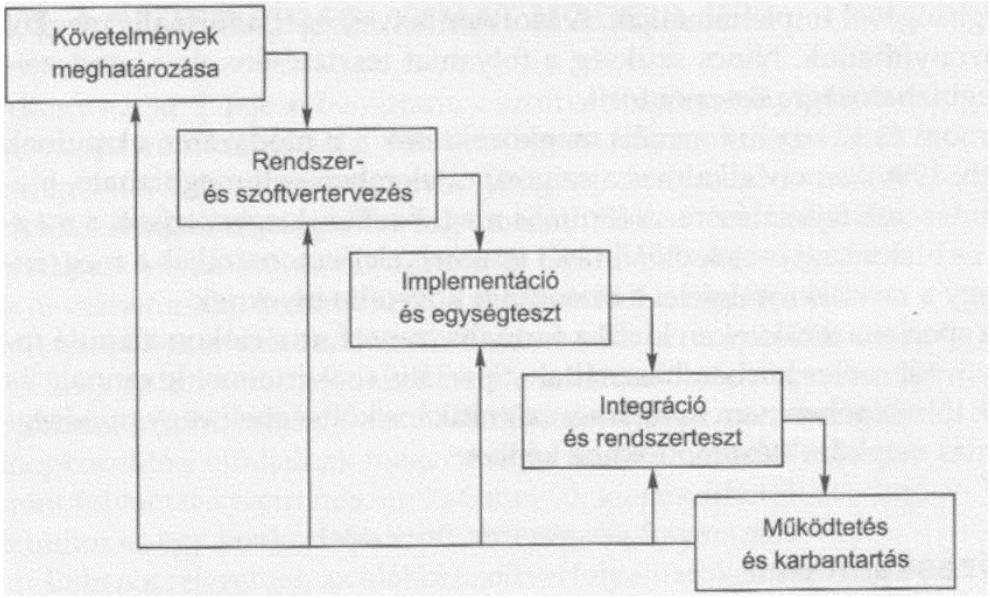
2. Rendszer- és szoftvertervezés: itt választódnak szét a hardver és szoftverkövetelmények, itt kell kialakítani a rendszer átfogó architektúráját. Az alapvető szoftver absztrakciók, ill. a köztük lévő kapcsolatok azonosítását és leírását is magában foglalja.

3. Implementáció: itt történik a kódolás. Ebben a szakaszban készülnek el a programok és a programegységek.

[1]Forrás: Ian Sommerville: Szoftverrendszerek fejlesztése

4. Rendszerteszt: Megtörténik a programok és a programegységek teljes rendszerként történő tesztelése, hogy a rendszer megfelel e a követelményeknek.

5. Működtetés és karbantartás: Általában ez a szoftver életciklusának leghosszabb fázisa. Ebben a szakaszban történik az esetleges hibák kijavítása, ill. a rendszer továbbfejlesztése a felmerülő új követelményeknek megfelelően. Az első elképzelés szerint az iskola informatikai munkaközösség tagjainak aktív részvételével és tanári kar minden tagjának bevonásával történne a fejlesztés a teljes fejlesztési életciklusban, a tervezéstől a tesztelésen át, egészen a jövőbeni továbbfejlesztésig. Mint később kiderült ez utópisztikus elképzelésnek bizonyult.



2.1. ábra. Alapvető szoftverfejlesztési lépések

3. fejezet

Követelmények elemzése, meghatározása

A legfontosabb követelmény az iskola dolgozóinak adminisztrációs tevékenységeinek támogatása informatikai eszközökkel. Az adminisztrációs feladatok rendkívül sok munkát rónak a pedagógusokra, különös tekintettel az osztályfőnöki feladatokat ellátó kollégákra. Nekik elsősorban a hiányzások adminisztrálása az, ami rengeteg munkával jár. Minden héten a naplóban összesíteniük kell a hiányzásokat, majd a törvényi előírásoknak megfelelően értesíteni kell a szülőt, ill. a területileg illetékes jegyzőt. A kollégák értesítése, érdeklődésének felkeltése és a projektben való részvételi meghívásnak, legjobb módját egy, az iskola belső levelezési listájára küldött elektronikus levél jelentette, melyet minden pedagógusnak postáztunk. 80 pedagógussal történő személyes elbeszélgetés túlságosan időigényes lett volna. A levelet úgy kellet megfogalmazni, hogy minden kolléga számára - még azoknak is akik nem rendelkeznek komoly informatikai ismeretekkel - érthető legyen, mit akarunk megvalósítani és a megértésen kívül keltse is fel az érdeklődését, legyen tudatában annak, hogy ez a projekt, ha megvalósul a munkáját nagymértékben meg fogja könnyíteni. A levélben példaként a hiányzások adminisztrálás lett felhozva, mert ez mindenki számára érthető és égető probléma(1. számú melléklet). A következő napon meglepődve tapasztaltam mennyi kolléga jelezte, hogy olvasta a levelet és szeretne ezzel

kapcsolatban konzultálni velem. Kezdeti lelkesedés meglepett. Még aznap sikerült ezekkel a kollégákkal néhány szót váltani, és ekkor jött a következő meglepetés. Mindenki nagyon lelkesedett, hogy egy ilyen szoftvert lesz lehetősége használni, majd az volt szinte az első kérdésük, hogy mikor lesz kész. Eszük ágában sem volt bármilyen módon részt venni a fejlesztésben. Még csak ötleteket sem tudtak - vagy akartak - adni arra vonatkozóan, hogy milyen nehézségeik vannak az adminisztráció során, mivel tudnánk megkönnyíteni a munkájukat. Az informatikai munkaközösség tagjai bármennyire is szerettek volna komolyabban belefolyni a munkálatokba, sajnos rengeteg elfoglaltságuk miatt erre nem igazán volt alkalmuk, ennek ellenére sikerült a fejlesztés ideje alatt konzultálni velük. A követelmények meghatározása teljes mértékben az én feladatom volt, segítségre már itt sem számíthattam. A követelmények meghatározásánál hamar szembetűnt, ahhoz, hogy ezeket az adminisztrációs munkafeladatokat megkönnyítsük, nagyon sok adatok kell nyilvántartanunk mind a tanulókról, mind az órákról, hogy ilyen adatok birtokában egy komplett elektronikus naplót is fel tudunk építeni. Olyan ötletek jöttek elő a követelmények meghatározása során, melyekkel rendszeres napi problémákra tudok megoldást találni. Olyan problémákra melyekbe folyamatosan belefutottam munkám során és eszembe nem jutott volna, hogy ezeket egyszerűsíteni lehet ilyen eszközökkel. Példaként lehet megemlíteni a tanteremkeresés problémáját. Ha egy tanterem valami miatt foglalt - ahol egyébként a tanár az órát szokta tartani - a pedagógusnak keresni kell egy másik termet. Ez mindig körülményes, mert lehet, hogy egy tanterem kulcsa a helyén található, de a teremfelelős tanárnak saját kulcs van és ő tart órát akkor. A hirdetőtáblán található A2-es méretű órarenden nehéz eligazodni, és abból nem derül ki a terem befogadó képessége. Ha minden adat rendelkezésünkre áll, akkor akár valós időben meg tudjuk jeleníteni mely termek szabadok ebben a pillanatban vagy akár a következő órán, vagy az osztály megadása után csak azokat a tantermeket jelenítené meg a rendszer, amelyben biztos elfér az a létszámú osztály. A követelmények meghatározásának folyamatában hamar kiderült, hogy az elérendő funkcionalitás olyan mértékű, amely már túlmutat egy szakdolgozat keretein, és a teljes rendszer lefejlesztése akár másfél évet is igénybe vehet. A követelmények

meghatározásánál - és szoftverfolyamat későbbi szakaszaiban is - az, hogy szinte semmilyen segítséget nem kaptam, valamilyen szinte előny is volt. Egyrészt minden ötletemet meg lehetett valósítani. A követelmények tervezésénél gyakran használt eszköz volt a papír és a ceruza. Mindig kéznél volt, gyorsan lehetett vele dolgozni. Hátránya a sok papíron, papír fecnin, több helyen megtalálható információhalmaz, melyet időnként egy szöveges dokumentumban hatékonyan rendszerezni kellet, a könnyebb felhasználhatóság érdekében a tervezési és implementálási folyamatban. A 2. számú mellékletben található néhány a követelmények meghatározása során összeállított igény.

4. fejezet

Rendszer és szoftvertervezés

A szoftverfolyamat e részében választódnak szét a hardver és szoftverkövetelmények, itt kell kialakítani a rendszer átfogó architektúráját, az alapvető szoftver absztrakciókat. A rendszer és szoftvertervezéshez megpróbáltam felhasználni Ian Sommerwille: Szoftverrendszerek fejlesztése című könyvben található módszereket. Itt több szempontot is figyelembe kellett venni. Milyen a hardver és szoftverkörnyezet, ebben a környezetben milyen technológiák használhatók. Ha a jelenlegi hardver és szoftverkörnyezet nem alkalmas a rendszer kiszolgálására milyen lehetőségek vannak a továbbfejlesztésre. Első körben csak a saját intézményre korlátoztam a különböző szempontok vizsgálatát, majd a későbbiekben általánosságban kezdtem a követelményeken gondolkodni. A tervezés kezdetekor mind a hardver mind a szoftverkörnyezet nagyon elavult volt szerver oldalon. Olyannyira, hogy még az olyan viszonylag kis erőforrást igénylő környezet sem tudta kiszolgálni, mint a LAMP (Linux, Apache, MySQL, PHP/Perl/Python)[1]. Majd vezetőséggel történt egyeztetés során kiderült, hogy olyan infrastruktúra kerül kialakításra rövidesen, mely nem gördít a fejlesztéshez használt technológiák elé semmilyen akadályt.

[1] http://en.wikipedia.org/wiki/LAMP_(solution_stack)

4.1. Hardver követelmények meghatározása

A hardverkövetelmények meghatározása viszonylag könnyű volt, miután meghatározásra került a szoftverkörnyezet. Mivel a kiválasztott szoftverkörnyezet futtatásához elegendő egy alsó kategóriás szerver, mely elvileg minden intézményben rendelkezésre áll, illetve hiánya esetén olcsón beszerezhető, így nem kellet sokat bajlódni a követelmények meghatározásával. A minimális hardverkövetelményt kellet csak előírni. Megint csak a szoftverkörnyezetből adódóan, ha nagyobb erőforrásigényre van szükség a szoftver futtatásához a rendszer skálázhatóságából adódóan könnyen bővíthető a hardver infrastruktúra.
Mindenesetre pontosan definiálni kell egy minimális hardver konfigurációt, mely lehetővé teszi a szoftver megfelelő teljesítményű futtatását. Ez a konfiguráció a következő:

- min. 2 GHz dupla magos processzor
- 2 GB RAM
- min 200 GB háttértár

Természetesen ebben a felsorolásban csak a legszükségesebb összetevők vannak felsorolva, melyek nélkülözhetetlenek a rendszer futtatásához. Itt nem említek, és nem részletezel olyan dolgokat, melyek megléte szükséges egy rendszer biztonságos üzemeltetéséhez, mint pl.: redundáns háttértér, biztonsági mentések készítéséhez szükséges eszközök, stb. Ez az üzemeltetés feladata, de a szoftver hardver követelményeinek meghatározáskor ennél a szoftvernél nem kritikus. Bár itt x86-os processzort jelölünk meg, a Java felépítésének köszönhetően és ebből adódóan a Java Enterprise Edition környezetet is lehetőségünk van más architektúrán is futtatni. Tehát aki úgy gondolja, hogy egy nagy teljesítményű mainframe számítógépen szeretné üzemeltetni a rendszert a lehetősé adott, amennyiben létezik Java futtatókörnyezet az adott platformra. Mivel a Java Enterprise Edition és az alkalmazásszerverek is teljes egészében Java nyelven íródtak a hordozhatóság elviekben teljes mértékben biztosított.

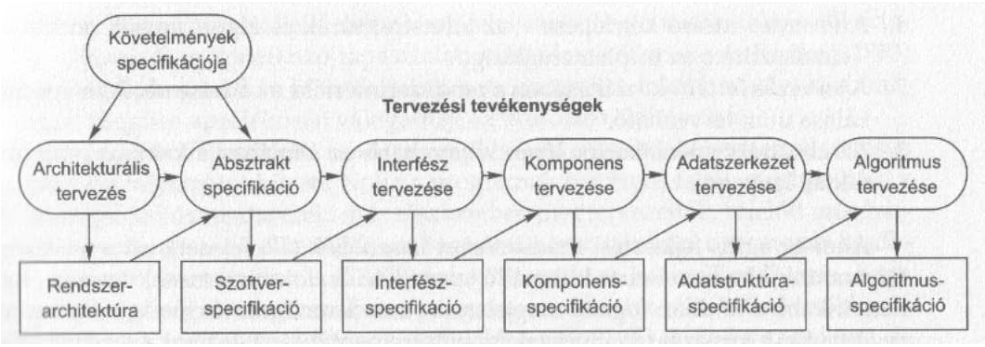
4.2. Szoftverkövetelmények meghatározása

A szoftverkövetelmények meghatározásánál is a legfőbb befolyásoló tényező a fejlesztéshez használt Java nyelv. Mint már említettem a Java egy platformfüggetlen nyelv, így elvileg bármely operációs rendszer melyre létezik Java futtató környezet képes a szoftvert futtatni. De mielőtt megadnánk egy szoftverkonfigurációt, mely szükséges a rendszer futtatásához nézzük meg hogyan is épül fel a szoftverkörnyezet, melyben futni fog az alkalmazásunk. Az operációs rendszer felett található a Java futtató környezet. A Java virtuális gép fogja futtatni az alkalmazásszervert, mely az üzleti logikát biztosítja. A megjelenítést szintén az alkalmazás szerver fogja biztosítani. Az alkalmazás üzleti logikája által kezelt adatok a OR leképezés után valamilyen relációs adatbázis kezelőben kerülnek tárolásra. Az RDBMS kiválasztásánál egyetlen megkötés van: létezzen az RDBMS-hez JDBC driver. Az alkalmazásszerver perzisztencia providere lesz az, ami majd az adott adatbáziskezelőben a megfelelő API hívásokkal - a JDBC driveren keresztül - és SQL utasításokkal az adatokat ténylegesen rögzíti. Az adott RDBMS SQL dialektusával sem kell foglalkozni, ugyanis a Java EE környezet biztosít számunkra egy saját lekérdezőnyelvet az EJB-QL -t (Enterprise JavaBeansQuery Language).
Íme, egy pontos szoftverkörnyezet, mely képes az alkalmazást futtatni:

- Sun GlassFissh alkalmazásszerver
- Apache Derby vagy MySQL adatbázisszerver
- Linux vagy Windows operációs rendszer

Ezek a szoftverek léteznek Linux és Windows operációs rendszerre így a felhasználó igényeinek és anyagi lehetőségeinek megfelelő operációs rendszert lehet választani.
Mivel a követelmények meghatározásánál hamar kiderült, hogy rendszer túlnőtt az eredeti elképzeléseken, úgy kellet megtervezni, hogy részekből, modulokból tevődjön össze, melyek valamilyen módon egymástól függetlenül, vagy egymásra épülve tudnak működni. Erre azért volt szükség, mert nem lehet megvárni, míg a követelményspecifikációban

meghatározott szolgáltatások mind elkészülnek. Több ok miatt sem. Egyrészt magának a rendszer fejlesztésének elindítója az a tény, hogy ez egy szakdolgozat, melyet határidőre kell leadni. Másrészt miért ne használnánk ki azt a lehetőséget, ha a rendszert modulokra lehet bontani és az egyes modulok egymástól függetlenül vagy egymásra épülve tudnak működni, a már elkészült modul vagy modulok használhatóak, miközben a többi modul még fejlesztés alatt áll. A szoftver tervezése a szoftver implementálandó struktúrájának és az adatoknak mint a rendszer részeinek a meghatározása valamint a rendszerkomponensek közötti interfészek és néha a használt algoritmusok leírása.



4.1. ábra. A tervezési folyamat általános modellje

A 4.1. ábrán látható a tervezési folyamat általános modellje, melyet a tervezés folyamán igyekeztem követni.

1. Architekturális tervezés. A rendszert építő alrendszereket és a köztük található kapcsolatokat azonosítani és dokumentálni kell.

2. Absztrakt specifikáció. Minden egyes alrendszer esetében meg kell adni a szolgáltatásaik absztrakt specifikációját és azokat a megszorításokat, melyek mellet azok működnek.

3. Interfész tervezése. Minden egyes alrendszer számára meg kell tervezni és dokumentálni kell annak egyéb alrendszerek felé mutató interfészeit. Ennek az interfész-specifikációnak egyértelműnek kell lennie, azaz lehetővé kell tennie, hogy anélkül használhassunk egy alrendszert, hogy tudnánk, hogyan is működik.

4. Komponens tervezése. A szolgáltatásokat el kell helyezni a különböző komponensekben és meg kell tervezni a komponensek interfészeit.

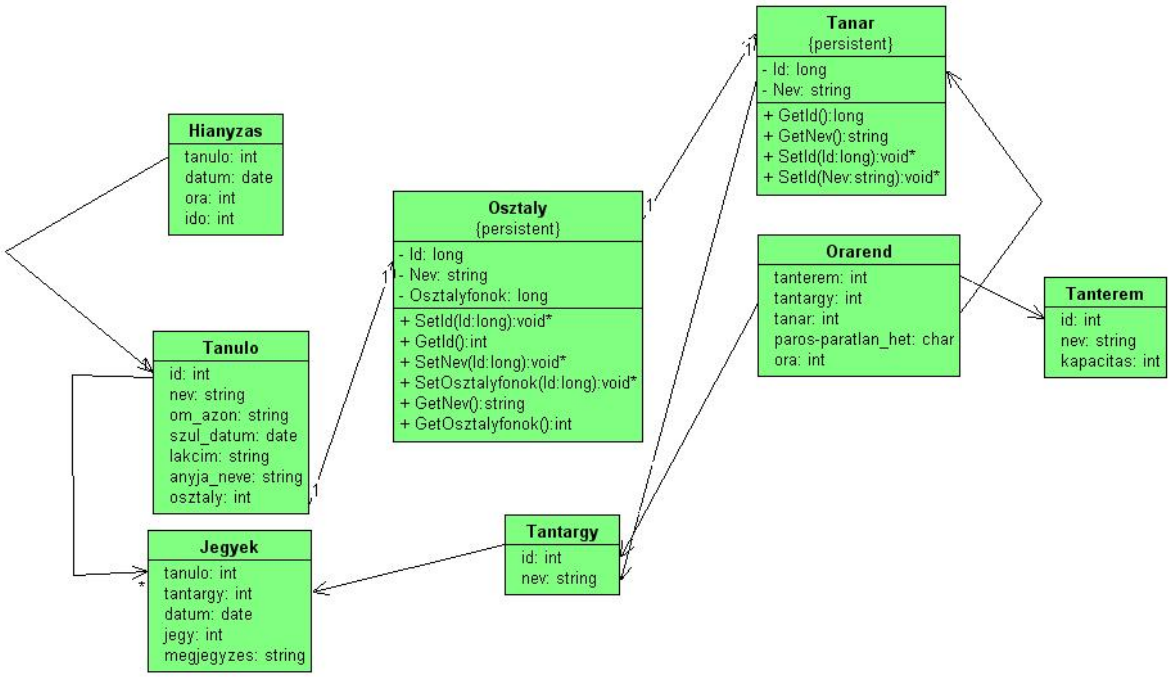
5. Adatszerkezetek tervezése. Meg kell határozni és részletesen meg kell tervezni a rendszer implementációjában használt adatszerkezetet.

6. Algoritmus tervezése. Meg kell tervezni és pontosan meg kell határozni a szolgáltatások biztosításához szükséges algoritmusokat.

Ezt a tervezési folyamatot többé-kevésbé sikerült betartani, bár az egyes lépések között voltak nagyobb ugrások. Az architekturális tervezésnél került meghatározásra, hogy a rendszer milyen főbb alrendszerekből álljon. Az alrendszereket én moduloknak neveztem el. A rendszerben hét modul került kialakításra melyek a következőek:

- törzsadat modul,
- hiányzások modul,
- elektronikus napló modul,
- tanmenet modul,
- órarend modul,
- értesítések modul,
- SMB (Small Message Board) modul

A törzsadat modulban kerül rögzítésre minden a rendszerben használt adat úgy, mint a tanulók és pedagógusok adatai, tantermek adatai, csengetési rend, stb. Ez a modul sosem tekinthető teljesen elkészültek, mert új funkciók, modulok rendszerbe kerülése esetén szükség lehet ennek a modulnak a bővítésére is. A 4.2. ábrán láthatóak a legfontosabb osztályok, melyek a rendszer működéséhez feltétlenül szükségesek.



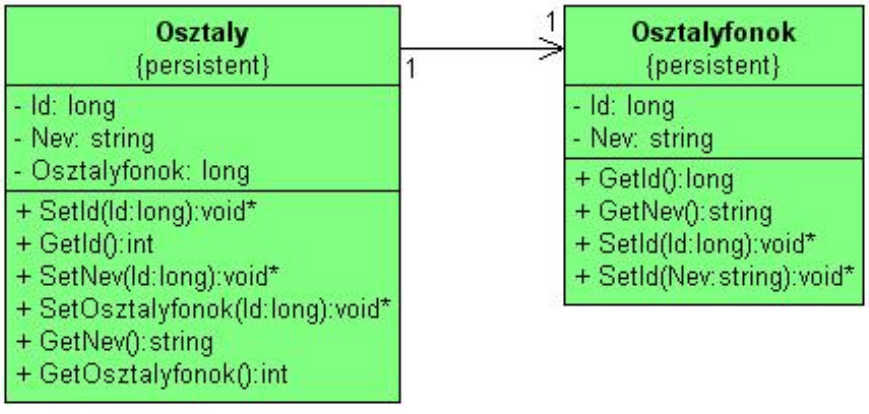
4.2. ábra. A legfontosabb osztályok

A hiányzások modulban vannak kezelve a tanulók hiányzásai. Ez a modul felelős a hiányzások összesítéséért, az értesítő levelek legenerálásáért. A törzsadat modul és a napló modul segítségével nyilvántartott adatokból pontosan ki lehet számolni a tanulók hiányzásait, majd a kalkulált információból kiindulva a program automatikusan elvégzi a szükséges műveleteket, pl. az értesítő levelek legenerálását. A 4.2. ábrán láthatjuk, hogy a `Hiányzas` osztály - ami egyben egy perzisztens osztály is - reprezentálja a tanulók hiányzásait. Ebben tartjuk nyílván, hogy melyik tanuló, mely napokon, mely órákról hány percet volt távol. Ebből könnyen ki lehet számolni egy tanuló eddigi igazol, illetve igazolatlan hiányzásait. Majd az összes hiányzás illetve igazolatlan hiányzás függvényében a szoftver jelez az osztályfőnöknek miután bejelentkezett a rendszerbe, hogy az összes hiányzása meghaladta az óraszám bizonyos százalékát és osztályozó vizsgát kell tennie, vagy az igazolatlan óráinak száma elérte az `x` órát és értesíteni kell a szülőt, illetve a területileg illetékes jegyzőnél feljelentést kell tenni. A kigyűjtött adatok birtokában a rendszer egy sablon alapján elkészíti a leveleket, melyek nyomtatás után postázásra kerülnek.

Az elektronikus napló lényegében egy hagyományos iskolai napló feladatát látja el. Ide lehet rögzíteni az osztályzatokat, a haladási naplót. Az osztályzatoknál az érdemjegy, az érdemjegy megjegyzésének dátuma, és egy megjegyzés rögzítésére van lehetősége a pedagógusnak. Felmerült az igény, hogy a jegyeket súlyozni kellene. Ennek a lehetőségét később a szoftverevolúció során kell majd megvalósítani. A haladási naplóban kell lekönyvelni az aktuális óra óraszámát és a tanított tananyagot. Amennyiben a tanmenetek modul is tökéletesen elkészül, melyben a tantárgyak éves tantervét óraszámokra lebontva rögzítjük, a rendszer automatikusan felajánlja a napló modulban a haladási napló kitöltésekor az adott órai tananyagot. Így nem kell papír alapú tanmenetekben keresgélni és "begépelni" az aznapi óra anyagát a napló modul haladási napló rovatába. Itt is lehetőség van a hiányzások rögzítésére mely adatokat a hiányzások modul fogja feldolgozni. A tanmente modulban lehet elkészíteni a tanmeneteket, melyek lekérdezhetőek így ha valaki helyettesíteni megy egy órára, meg tudja nézni mit kell azon az órán tanítania, illetve az elektronikus napló modul az itt rögzített tanmenet alapján automatikusan ki tudja tölteni a haladási naplót. Órarend modul hasonló a tanmenetek modulhoz. Az órarendet lehet vele nyilvántartani melyet más modulok szükség esetén hasznosítani tudnak. Pl. az elektronikus napló a rendszeridő alapján képes megállapítani, hogy bejelentkezett pedagógus adott időpillanatban milyen órát tart és az alapján kitölteni az adott óra haladási naplóját illetve a hiányzókat is az adott órára fogja rögzíteni. Értesítési modul a különböző személyeknek (pl. szülő) illetve szervezeteknek küldendő értesítések előállításában és tárolásában kap szerepet. SMB funkció a pedagógusoknak különböző formában - e-mail, sms, web, megjelenítő - történő információközlésre használható. Az interfészek, komponensek és adatszerkezetek lényegében objektumok. Ezek tervezésénél az UML eszközrendszerének segítségét vettem igénybe. Bár sok esetben a hagyományos papír-ceruza páros volt segítségemre a tervezésnél a későbbiekben ez lefordításra került az UML nyelvezetére. A 4.3. ábrán lévő példán látható pl. az osztály és osztályfőnök, mint adatszerkezet és a köztük lévő kapcsolat.

4.2.1. A szoftverkörnyezettel szemben támasztott elvárások

- Perzisztencia. A rendszer számára biztosítani kell, hogy az adatok a program leállítása után is megmaradjanak. Erre a feladatra leginkább relációs adatbázisokat (RDBMS) alkalmaznak. Az alkalmazásban meg kell oldani ezen relációs adatbázisok elérését. Mindezt olyan formában, hogy elfedjük az egyes adatbázis kezelő rendszerek alacsony szintű gyártóspecifikus részeit, mint az alkalmazott hálózati protokoll, az egyes RDBMS rendszerek SQL "nyelve - dialektusai" közötti különbségek, stb. Illetve, hogy összeegyeztessük a relációs adatbázisok alapelveit az objektum orientált szemléletmóddal. A perzisztencia alkalmazásával az egyik legégetőbb problémát sikerül kiküszöbölni egy rendszer fejlesztése során. Melyik adatbázis kezelővel biztosítsam az adatok tárolását? Ha nagyon drága, robosztus RDBMS-t választok, mint pl.: Oracle, vagy IBM DB2, akkor azt a kisebb ügyfelek nem tudják megfizetni illetve a hozzá kapcsolódó hardver erőforrások költsége is magas lesz, komoly szakember szükséges az üzemeltetéséhez, egyszóval rendkívül drága. Csak nagyobb vállalatok engedhetik meg maguknak. Ellenben ha valami költséghatékonyabb megoldását választunk az a nagyvállalatok igényeit nem biztos, hogy ki tudja



4.3. ábra. Osztály és osztályfőnök osztálydiagramja

elégíteni. Ha az alkalmazást saját magam akarom felkészíteni mindkét esetre, akkor az erőforrás igények növekedése nálam jelentkezik. A perzisztencia ezeket a problémákat hivatott orvosolni.

- Többszálúság. Több kérés párhuzamos kiszolgálásánál jelenthet megoldást a többszálúság.

- Tranzakciókezelés. Ha több felhasználó konkurensen akar hozzáférni ugyanazokhoz az adatokhoz, ügyelni kell arra, hogy az adatok mindig konzisztens állapotban maradjanak. Biztosítani kell, hogy az összetartozó módosítások vagy mindegyike sikerüljön vagy egyik sem. Ezekre a problémákra a tranzakciókezelés nyújt megoldást úgy, hogy a tranzakciókezelés alacsony szintű műveleteit elrejti a felhasználó elől.

- Távoli elérés. Ha rendszerünk webes felülettel rendelkezik, vagy valamilyen vékony vagy vastag klienssel, ez egyfajta elosztottságot feltételez a rendszerünkben. Ilyen esetben szükség van valamilyen protokollra, mely lehetővé teszik a távoli metódushívást. Szükség lehet egy olyan futási környezetre, amely elfedi a számos kommunikációs protokoll hálózati szintű részleteit a fejlesztő elől.

- Névszolgáltatás. Elosztott rendszerekben bizonyos objektumokat, erőforrásokat elengedhetetlen, hogy név szerint regisztrálni és regisztráció után a nevére hivatkozva, a konkrét hálózati cím ismerete nélkül elérni tudjunk.

- Skálázhatóság. A terhelés idővel növekedhet, ezért rendszerünket fel kell készíteni arra, hogy a megnövekedett igények miatti terhelésnövekedést kezelni tudja anélkül, hogy a felhasználók jelentős teljesítménycsökkenést tapasztalnának. Mindezt anélkül, hogy a programkódot módosítani kellene. Ennek két lehetséges megoldása van. Az egyik a vízszintes a másik a függőleges skálázási mód. Függőleges skálázási módnál egyre erősebb hardverre (processzor, memória, háttértár) telepítjük ugyan azt a szoftvert. Ennek a skálázási módszernek a felső határa az adott költségek

mellet ésszerűen beszerezhető legerősebb hardver. Vízszintes skálázásnál több gépből álló együttműködő csoportot úgynevezett klasztert hozunk létre. Ez a skálázási módszer ésszerűbb és nem utolsó sorban olcsóbb megoldást kínál a teljesítmény növelésére a függőleges skálázással szemben. Arról nem is beszélve, hogy a függőleges skálázásnak van egy felső határa, mely a beszerezhető legerősebb hardver, míg a vízszintes skálázás elviekben "korlátlan" bővíthetőséget biztosít.

- Magas rendelkezésre állás. Bármely szoftverrendszernél kellemetlen lehet, kritikus rendszerek esetében akár emberéletekbe is kerülhet a rendszer meghibásodása vagy teljes leállása szoftver vagy hardver hiba miatt. A skálázhatósághoz hasonlóan az egyik legjobb megoldást ebben az esetben is a klaszterezés jelenti, így nagy előnyt jelent, ha a futtatási környezet fel van rá készítve és nem a fejlesztőnek kell az ilyen megoldásokat implementálni.

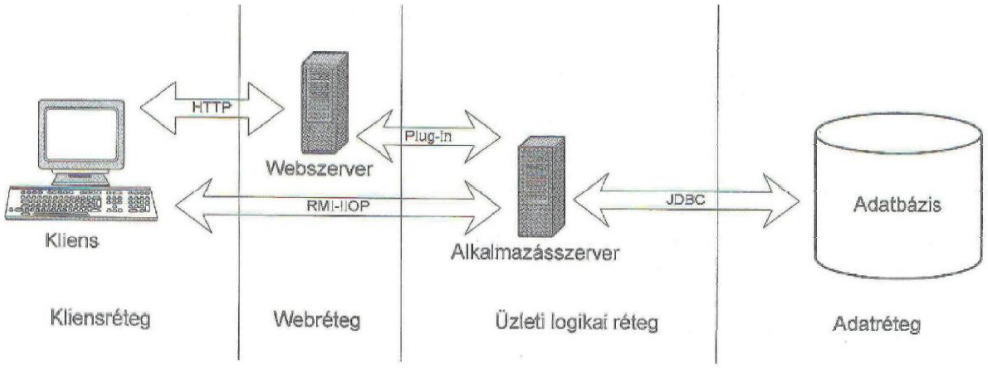
- Aszinkron üzenetkezelés. Egymással nem túl szoros kapcsolatban álló rendszerek közötti üzenetküldést megvalósító rendszer. Az aszinkron üzenetkezelő rendszer az elküldött üzeneteket akkor is megőrzi, ha a partnerrendszer ép nem működik. Jelen alkalmazás esetében a Java EE rendszer ezen szolgáltatását nem használjuk ki. A későbbi továbbfejlesztésnél szóba került SMS értesítőrendszernél lehetséges a kihasználása abban az esetben, ha a mobilszolgáltató SMS szervere valamiért nem elérhető.

- Biztonság. Többfelhasználós rendszereknél felmerül a biztonság kérdése. Mit tehet meg az adott felhasználó? Egyáltalán hogyan azonosítjuk a felhasználót? Mivel többrétegű elosztott alkalmazásokról beszélünk és ezek az alkalmazások a hálózaton kommunikálnak, legtöbbször nem csak a céges hálózaton belül, hanem az Interneten is, felmerül a kérdés, hogyan védekezzünk a hálózati forgalom rosszindulatú módosítása vagy lehallgatása ellen.

- Monitorozás és beavatkozás. Bizonyos esetekben szükség lehet a működés nyomon

követésére hibakeresés vagy statisztikakészítés céljából, illetve, hogy bizonyos esetekben beavatkozzunk a működésbe.

A fenti szempontok figyelembe vételével a választás végül a Java Enterprise Edition technológiára és a hozzá kapcsolódó eszközkészletre esett, melyet később részletesebben ismertetek.



4.4. ábra. Többrétegű architektúra

Felvetődött a lehetőség, hogy más intézmények is használnák a későbbiekben ezt a rendszert, így ezt a tényt is figyelembe kellett venni az alkalmazandó technika, technikák kiválasztásakor. A jövőben elképzelhető, hogy több intézményt vonnak össze, így megnövekszik az egy rendszeren belül kezelendő tanulók száma, ezzel együtt megnövekszik az erőforrásigény. Emiatt már a kezdetektől fogva skálázhatóvá kell tenni a rendszert. Amiatt, hogy erre ne kelljen sok fejlesztői erőforrást "pazarolni", így ki lehet használni a Java EE által nyújtott lehetőségeket. Biztosítani kellett azt is, hogy a felhasználói szempontból a lehető legegyszerűbb legyen a kliensoldali réteg. Legalábbis ami a kliens oldal üzemeltetését jelenti. A felhasználók többsége kezdetekben az intézmény dolgozói lesznek. Ők nem informatikai szakemberek, nem feltétlenül tud egy új szoftververzió megjelenése esetén önállóan egy kliens oldali szoftvert feltelepíteni vagy frissíteni. A szűk humán erőforrások miatt lehetséges, hogy az intézményben nincs erre a célra kijelölt informatikai szakember, aki a kliens oldali karbantartást elvégzi. Így a legjobb megoldásnak a Webes

kliens tűnt minden szempont szerint. Így bárki elérheti egy egyszerű böngészőből az alkalmazást, bárhol is tartózkodjon. A későbbi továbbfejlesztési igényeket szem előtt tartva is a webes technológia alkalmazása jelentette a legjobb megoldást. Nem csak intézményen beül érhető el az alkalmazás, hanem bárhonnan egy web böngésző segítségével. Így a későbbi továbbfejlesztések során lehetőséget tudunk majd biztosítani a szülőknek illetve a diákoknak, hogy ők is, akár otthonról hozzáférhessenek a rendszerhez. A távoli hozzáférés segítségével a dolgozók, akár otthonról is elvégezhetik feladataikat. Nem kell bent maradnia az intézményben munkaidő után, hogy elvégezze adminisztratív feladatait, mert hivatalos dokumentumokat, mint pl. osztálynapló nem vihet ki az intézményből. Látható, hogy a Java EE környezet rendkívüli összetettsége révén rengeteg szolgáltatást nyújt a fejlesztőnek, sok munkát "levesz a válláról" így a fejlesztőnek csak a lényegi dolgokra, az üzleti logikára kell koncentrálnia és nem kell azokat az alkalmazás működése szempontjából fontos dolgokat implementálnia, amelyeket biztosít a keretrendszer. Így a fejlesztés hatékonyabb lesz, rövidül a fejlesztési idő, kevesebb a hibalehetőség, ugyanis a keretrendszert rengetegen használják minden terülten, a kis vállalatoktól kezdve egészen a bankszektorig, így a hibák is hamarabb előjönnek.

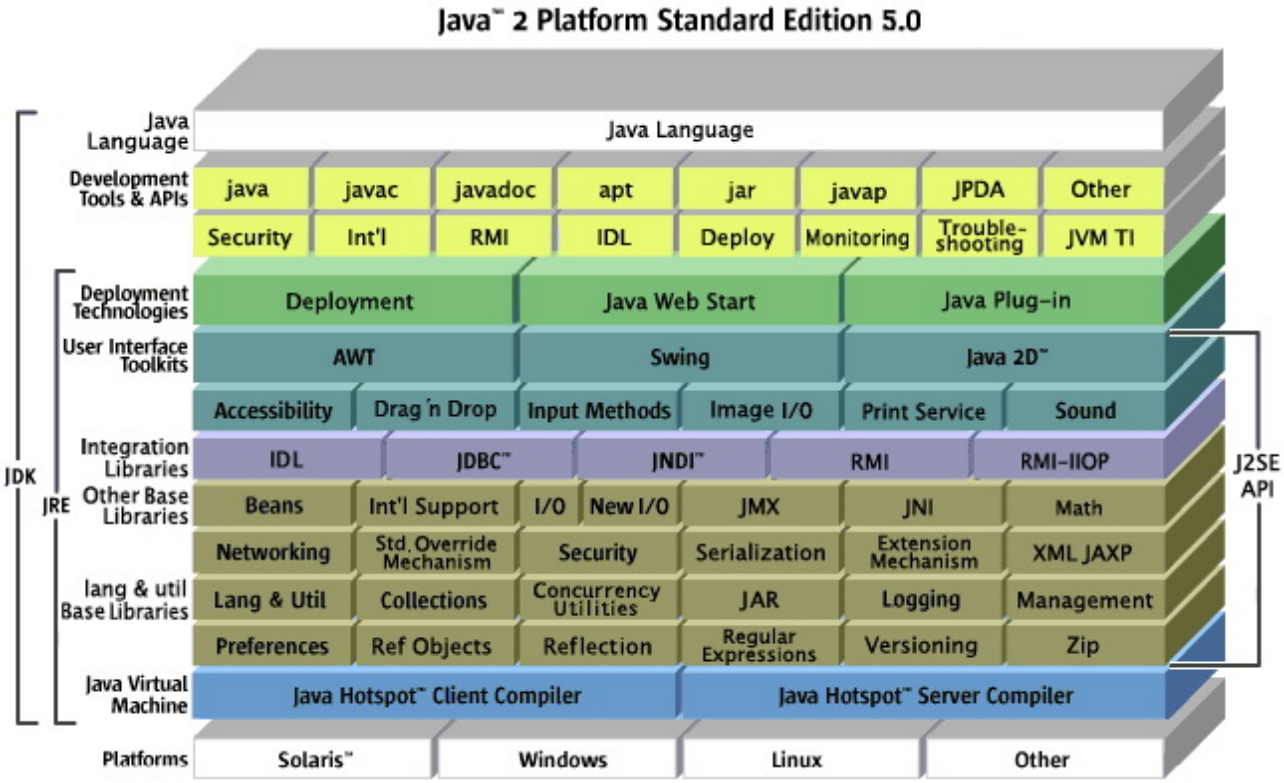
4.2.2. A felhasznált szoftverkörnyezet

4.2.2.1. Java nyelv

Java[2] nyelv egy objektumorientált programozási nyelv melyet a Sun Microsystems[3] fejlesztett ki a 90-es évek elején. A Java egyik legfontosabb tulajdonsága a platformfüggetlenség. Ez azt jelenti, hogy a Javaban írt programok hasonlóan fognak futni a különböző hardvereken. Például egy Java alkalmazás elvileg változtatás nélkül fut egy komoly asztali munkaállomáson ugyan úgy, mint egy mobiltelefonon. A nyelvnek ez a tulajdonság komolyan befolyásolta a mellette való döntést. Mivel iskolai adminisztrációs rendszerről van szó és a szoftver oktatási intézményeknek készül, az oktatási

[2]http://java.sun.com

[3]http://www.sun.com



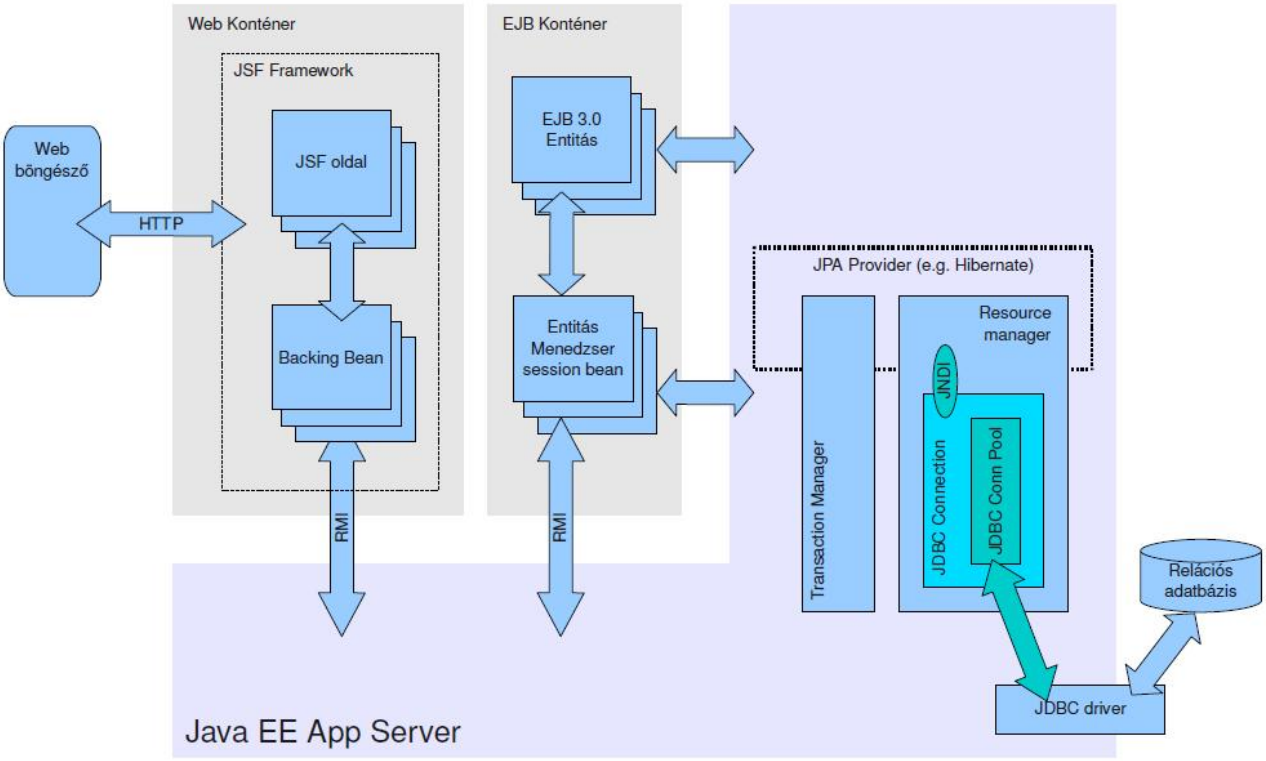
4.5. ábra. Java 2 SDK felépítése

intézmények változatos hardver- és szoftverellátottsággal rendelkeznek, mely döntően befolyásolja, milyen szoftvereket tudnak futtatni meglévő rendszerükön. Bár jelenleg az állam jóvoltából az oktatási intézmények ingyen jutnak Microsoft licenchez, látva a jelenlegi gazdasági helyzetet, hosszú távon nem biztos, hogy ez a lehetőség fenntartható. Így lehetőséget kell biztosítani arra, hogy a szoftver többféle hardver és szoftverkörnyezetben is használható legyen. Ennek egyik módja, ha valamilyen platform független megoldást választunk, melyre talán a legjobb választás a Java illetve ennek kiterjesztése a Java Enterprise Edition. A platformfüggetlenséget úgy valósítja meg a Java, hogy a fordítóprogram csak byte kódra fordítja le a forráskódot, ami ezután futtatásra kerül a Java Virtuális Gépen (JVM) amely lefordítja az illető hardver gépi kódjára. Továbbá léteznek szabványos könyvtárcsomagok, melyek lehetővé teszik az adott hardver illetve szoftver sajátosságainak, - mint pl.: grafika, hálózat, szálkezelés - egységes módon való kezelését.

4.2.2.2. Java Enterprise Edition

> „A Java EE egymondatos meghatározása az alábbi lehetne: architektúra vállalati méretű alkalmazások fejlesztésére a Java nyelv és internetes technológiák felhasználásával."[4]

Miért nem elegendő nekünk a Java nyelv? Nagyméretű rendszerek fejlesztésénél gyakran fogalmazódnak meg hasonló követelmények, mint a skálázhatóság, biztonság és más rendszerekkel való integrálhatóság. A Java EE ezekre a gyakori problémákra nyújt egységes megoldást, ezekre a problémákra nyújt egy egységes keretrendszert, mely kielégíti a fent említett igényeket.



4.6. ábra. Java EE felépítése

[4]Imre Gábor: Szoftverfejlesztés Java EE platformon

5. fejezet

Implementáció

Az implementáció elkezdésekor az első felvetődő kérdések, hogy milyen alkalmazás és adatbázis szervert válasszunk a fejlesztéshez és a teszteléshez. Milyen IDE legyen alkalmazva a fejlesztés során, illetve, hogy használjunk-e valamilyen verziókezelő rendszert a fejlesztés folyamán. Szerencsére mindhárom eszközből bő választék található. Nem gondolkodhattam úgy, hogy mi lenne, ha egy nagy szoftverfejlesztő cégnek kellene kiválasztanom ezeket az eszközöket. Így maradnom kellett a realitásnál, mely szerint egy iskolai adminisztrációs rendszert fejlesztek egyedül és ennek megfelelően kell az eszközöket megválasztanom. Az alkalmazásszervernél a legkézenfekvőbb megoldásnak a Sun által készített GlassFish alkalmazásszerver tűnt, mely a Java EE technológia referencia megoldása. Az RDBMS kiválasztása is könnyen ment. Nem kellet foglalkozni a teljesítménnyel, robosztussággal, hogy milyen szoftver és hardver környezetben kell majd futnia az RDBMS-nek, milyen mennyiségű adatot kell kezelnie, hány kérést/tranzakciót kell kiszolgálnia egy időben. A legfontosabb szempont a kiválasztásnál a könnyű kezelhetőség volt. Ne kelljen sokat vesződni a telepítéssel, konfigurálással, adminisztrálással. Fontos szempont volt az alkalmazásszerverrel és az fejlesztőkörnyezettel való egyszerű integrálhatóság. A fejlesztéshez használt "szövegszerkesztő" megtalálása már izgalmasabb feladat volt. A notepad-et első körben kiejtettem és nem csak az én lustaságom miatt, hanem mert egy egyszerű szövegszerkesztő tényleg nem elegendő egy ilyen méretű

fejlesztéshez.

Kézenfekvő volt egy, a környezethez fejlesztett IDE használata. A bőség zavarában kellet megtalálni a nekem leginkább megfelelőt. A következő eszközök jöttek szóba:

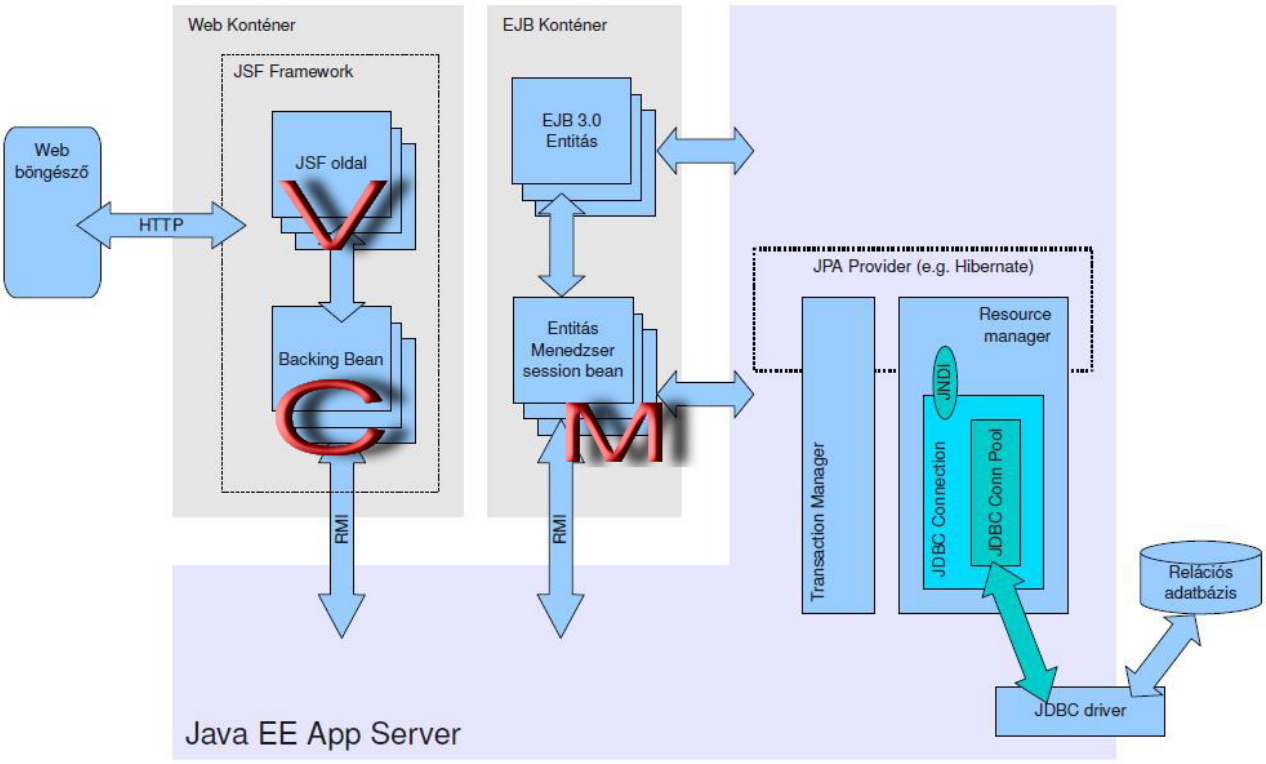
- IntelliJ IDEA [1]
- IBM - WebSphare Studio [2]
- Oracle - Jdeveloper [3]
- NetBeans [4]
- Eclipse [5]

A kereskedelmi termékek itt is első körben kiestek, tehát maradt a NetBeans és az Eclipse. Az Eclipse nagyon szimpatikus IDE számomra. A szövegszerkesztő része a legjobb a létező eszközök között. Viszont az Eclipse integrálhatósága a GlassFisshel nem a legjobb, nagyon körülményes, sok probléma van vele és jelenleg nem is a legstabilabb. Így a fejlesztés kezdeti szakaszában tapasztalt kellemetlenségek arra az elhatározásra juttattak, hogy ki kellene próbálni a NetBeanst, mely a Sun gyermeke és tökéletesen integrálva van a GlassFish alkalmazás szerverrel. Bár nem olyan produktív a NetBeans mint az Eclipse, de az utóbbi „hiányosságai" miatt a fejlesztést teljes mértékben a NetBeans segítségével végeztük. A fejlesztés folyamán a Sun kiadott a GlassFish alkalmazásszerverből egy Eclipsbe integrálható változatot, mely modulként épül be a fejlesztőkörnyezetbe, de akkor már nem lett volna célszerű lecserélni az IDE-t. Verziókezelő rendszer alkalmazásának addig, míg a fejlesztést egyedül végzem, nem látom értelmét. A cp parancs és a gz tömörítő segítségével nagyon jól tudom menedzselni a forráskódot.

[1] http://www.jetbrains.com/idea
[2] http://www-01.ibm.com/software/websphere
[3] http://www.oracle.com/technology/products/jdev/index.html
[4] http://netbeans.org
[5] http://eclipse.org

5.1. Az alkalmazás szerkezete

A fejlesztés során az MVC minta alkalmazására törekedtem. Az MVC (Modell-View-Controller) tervezési minta a modell, a nézet és a vezérlés logikai szétválasztásán alapul. Modell alatt egy vagy több osztály értünk mely a valóság egy darabját leíró entitásokat reprezentálja. A megjelenítést a View végzi, mely képes az információkat rendezett formában megjeleníteni. A vezérlő pedig kontrollálja a kiszolgáló és a felhasználó közötti párbeszédet. A controller szerepét a backing beanek töltik be. A modellt az entitás menedzser beanjei adják. A View-t a JSF keretrendszer generálja a backing beanek segítségével.



5.1. ábra. MVC tervezési minta

5.2. Adat réteg

Az rendszer fejlesztése során az adatelérési réteghez az Apache Derbyt [6] használtam, melynek okairól későbbiekben írok. Az adatelérési réteg fölött a JPA provider található. Magára a JPA providerre van bízva az adatréteg kezelése. Ezzel több dolgot is sikerült elérni. Egyrészt nincs "beledrótozva" az alkalmazásba egy konkrét szerver, ezzel lehetőséget biztosítunk a megrendelőnek, hogy tetszőleges adatbázisszervert alkalmazzon. Másrészt fejlesztői oldalról megközelítve, a fejlesztőnek sem kell az egyes DBMS rendszerek helyi dialektusával foglalkoznia. Magát az adatbázis sémát is a JPA generálja. A JPA ellenőrzi az adatbázissémát, és amennyiben nem találja benne a szükséges adatbázis elemeket, automatikusan legenerálja ezen elemek létrehozásához szükséges DDL utasításokat, majd végre is hajtja azokat.

5.3. Entitás réteg

Ennek a rétegnek a feladata, hogy egy adott objektum adatait perzisztes módon tárolja és biztosítsa az alkalmazás számára a hozzáférhetőséget.

5.4. JSF réteg

A JSF a Java EE környezet által biztosított webes keretrendszer. A JSF webes keretrendszer célja, hogy összekapcsolja a grafikus és felülettervező által megtervezett felületet és a felhasználói felület mögött álló üzleti logikát. A JSF réteg tette lehetővé azt, hogy a fejlesztés során ne nagyon keljen foglalkoznom a felhasználói felület kinézetével. Egy viszonylag egyszerű, puritán felülete van az alkalmazásnak, így elsősorban az üzleti logikára koncentrálhattam, majd a későbbiek folyamán a JSF réteget megfelelően módosítva lehetőség lesz egy ízlésesebb felhasználói felületet kialakítani anélkül, hogy az üzleti logikát módosítani kellene.

[6] http://db.apache.org/derby

5.5. Web kliens réteg

A réteg úgy lett kialakítva, hogy a legtöbb böngészővel kompatibilis legyen, az oldalak mindenhol hasonlóan jelenjenek meg.

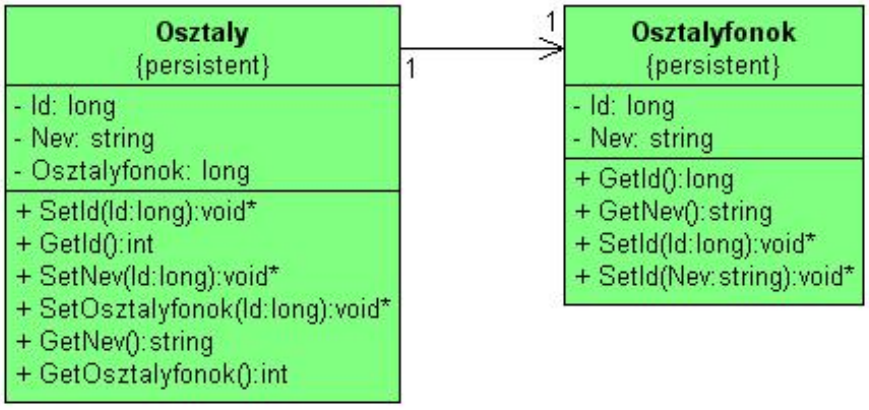
5.6. Felhasznált technológiák

5.6.1. JPA

A JPA (Java Persistence API) automatikus objektumrelációs leképezés megvalósítását támogató technológia mely része az EJB3 specifikációnak, de a Java SE környezetben is használható [7].

5.6.2. Entitások

Nézzünk egy egyszerű példát, melyben egy iskolai osztály reprezentálásához kialakított entitást hozunk létre.



5.2. ábra. Osztály és osztályfőnök entitás

[7]Forrás: Imre Gábor: Szoftverfejlesztés Java EE platformon

```
package core;

import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.Column;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

/**
 *
 * @author Bodnár József
 */
@Entity
public class Osztaly implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    @Column(name = "nev", length = 50, unique=true, nullable = false)
    private String nev;

    public String getNev() {
        return nev;
    }

    public void setNev(String nev) {
        this.nev = nev;
```

```
}

public Long getOsztalyfonok() {
    return osztalyfonok;
}

public void setOsztalyfonok(Long osztalyfonok) {
    this.osztalyfonok = osztalyfonok;
}
private Long osztalyfonok;

public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

@Override
public int hashCode() {
    int hash = 0;
    hash += (id != null ? id.hashCode() : 0);
    return hash;
}

@Override
public boolean equals(Object object) {
```

```
        if (!(object instanceof Osztaly)) {
            return false;
        }
        Osztaly other = (Osztaly) object;
        if ((this.id == null && other.id != null) ||
           (this.id != null && !this.id.equals(other.id))) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return "core.Osztaly[id=" + id + "]";
    }

}
```

De mi is az entitás? Láthatjuk, hogy az osztály adatainak reprezentálásához, három adattal kell rendelkeznünk. Az osztály nevével, az osztályfőnök személyével és az osztály azonosítójával. Ezeket az adatokat az alkalmazáshoz használt relációs adatbázis kezelőkben tartják nyilván a leggyakrabban. A Java nyelv felépítéséből adódóan viszont objektumokkal dolgozunk, tehát a relációs adatbázisban tárolt adatokat valahogy meg kell feletetnünk Java környezetben használt objektumoknak. Ehhez az objektum-relációs leképezést használják, melynek lényege, hogy az osztályokat adatbázis tábláknak, az osztályok attribútumait pedig az adatbázistábla oszlopainak felelteti meg. Használat során az adott entitásosztály egy példánya pedig egy adatbázistábla egy rekordjának feleltethető meg. Entitásokat hagyományos POJO objektumok és annotációk segítségével nagyon egyszerűen létrehozhatunk. A POJO (Plain Old Java Object) osztály egy

hagyományos osztály melynek létrehozásakor a következő szempontokat kell figyelembe venni:

- attribútumai - melyek az adatbázis oszlopainak felelnek meg - privátak
- mezőket csak az osztályon kívülről, publikus metódusokon keresztül érhetjük el mely metódusok setter/getter páros (a fenti példában setNev/getNev). A getterek illetve setterek létrehozása elvileg nem jelent többletterhet a fejlesztőre ugyanis egy jó fejlesztőeszköz támogatja ezeket és létrehozza a programozó helyett.
- van alapértelmezett konstruktora. Az alapértelmezett konstruktor segítségével az osztály objektumait a JPA reflectiont használva tudja példányosítani

5.6.2.1. Annotáció

A Java EE a POJO objektumokat az annotációk segítségével tudja leképezni a relációs adatbázisba. Az annotáció, mint nyelvi elem a Java Standard Edition 5.0 verziójával kerültek bevezetésre. A Java EE 5.0 óta az OR mapping ezzel a technológiával van megoldva, mely nagymértékben egyszerűsítette az adatbázisban tárolt rekordok leképezését objektumokká. A Java EE 5. verziója előtt az objektumrelációs leképezés sokkal fájdalmasabb volt a fejlesztők számára. Speciális kommentek és XML állományok segítségével volt megvalósítható. Ez nagymértékben nehezítette a fejlesztést, mert több helyen, a kommentekben és az XML állományokban volt szétszórva az objektumrelációs leképezéshez használt kód. Az adatbázisok leképezéséhez lényegében nem kell mást tennünk, mint egy osztályt definiálnunk. Az elkészült osztályt kiegészítve néhány annotációval el is készült a leképezést végző osztályunk. Majd ezt az osztály példányosítva a létrejövő objektum fogja reprezentálni az adattábla egy rekordját a memóriában. Az annotáció típusát egy interfészen keresztül definiálhatjuk. Ezt az interfészt kell implementálni az osztályban. A mi esetünkben az entitás osztályban használandó annotációhoz a javax.persistence csomagot kell használnunk. A forrásban az @Entity annotációval tudjuk jelezni, hogy egy entitásról van szó. Az osztályt el kell látni attribútumokkal,

melyek az adattábla egyes rekordjait fogják jelenteni. Az egyes attribútumoknak privát adattagoknak kell lenniük, tehát ezeket az osztályon kívülről közvetlenül nem lehet elérni. Majd az egyes adattagokhoz létre kell hozni a setter/getter metódusokat, melyekkel az adattagok értékét tudjuk megváltoztatni illetve lekérdezni. A fenti példában ahol egy iskolai osztályt reprezentálunk a private String nev attribútum tárolja az osztály nevét. Ez az attribútum private láthatóságú, tehát nem érhető el közvetlenül az osztályon kívülről. Ehhez az attribútumhoz létrehozásra került a public String getNev() és a public void setNev(String nev) metódusok. Mint láthatjuk ezeknek a metódusoknak a láthatósága public, tehát már osztályon kívülről is elérhetőek. Ezek szerkezetét a Java EE specifikációja pontosan definiálja annak érdekében, hogy az alkalmazásszerver megtalálni, és kezelni tudja ezeket a metódusokat. Az elsődleges kulcs definiálására az @Id annotációt vezette be a Java EE. Az elsődleges kulcs entitás osztályok attribútumaira bizonyos megszorítások érvényesek. Nem vehetnek fel bármilyen típust. Csak bizonyos primitív típusúak, illetve String és Date típusúak lehetnek. Ezzel a pár lépéssel lényegében el is készült a fejlesztő az entitással. Csak ennyi dolga volt. Innentől kezdve az alkalmazás szerver feladata, hogy a rendszerhez kapcsolt konkrét adatbázisszerverben létrehozza a táblákat és egyéb adatbázis elemeket. Az adattáblák nevei az osztályok nevei lesznek, a mező neveket pedig az osztályban található attribútumok alapján fogja képezni a rendszer. Természetesen ettől el lehet térni, a fejlesztőnek lehetősége van felüldefiniálni a Java EE által létrehozott adattábla és mezőneveket. Erre szintén az annotációk használhatók. A tábla nevének felüldefiniálására a @Table annotáció használható, míg a mezőnevek felüldefiniálására és tulajdonságaik módosítására a @Column annotációt vezették be. A @Column annotációt megfelelően paraméterezve nem csak az adatbázis tábla mezőinek nevét adhatjuk meg, hanem a mezőre vonatkozóan különböző megszorításokat is tehetünk. A length paraméterrel a mező hosszát tudjuk szabályozni. Természetesen csak olyan típusú mezők esetén ahol ennek van értelme, mert egy logikai típusnál nincs értelme mező hosszról beszélni. A keretrendszer lehetőséget biztosít számunkra az adatbázismező egyediségének biztosítására. Erre szolgál a @Column annotáció unique paramétere. En-

nek segítségével a JPA az objektumrelációs leképezés során egy unique constraintet fog az adattábla megfelelő mezőjére állítani. A paraméter boolean típusú, tehát ha az értéke igaz (unique=true) abban az esetbe a futtatókörnyezet létrehozza az adatbázismezőhöz tartozó unique constrainter, amennyiben hamis értéket adun paraméterül (unique=false) az egyediséget biztosító constraint nem kerül létrehozásra. Fontos még az adattábla mezőinél szabályoznunk, hogy az adott mező kitöltése kötelező-e vagy nem, vagyis hogy az adott mező tartalmazhat-e null értéket. Ez a nullable paraméterrel jelölhetjük, mely hasonlóan logikai igaz vagy hamis értéket vár, mint az unique paraméter. A Java EE 5.0-val a Sun a Java EE egyik nagy hiányosságát orvosolta, mégpedig az elsődleges kulcsok automatikus generálását, vagyis a szekvenciákat. Ehhez az entitás osztályon belül kell kiegészítenünk az elsődleges kulcsot a @GeneratedValue annotációval. Az így létrehozott elsődleges kulcsokra még erősebb megszorítások érvényesek, mint fentebb említettem. Csak és kizárólag egész típussal rendelkező elsődleges kulcsot lehet automatikusan lehet generálni. A generálás típusát a strategy paraméterben lehet definiálni mely a következő értékek egyikét veheti fel:

- SEQUENCE: a relációs adatbázis kezelő által kezelt számláló segítségével állítja be az elsődleges kulcs értékét automatikusan. A számlálót a @SequenceGenerator annotációban kell definiálni.
- IDENTITY: egy adatbázisbeli tábla egy autóinkrementálandó oszlopára fog leképeződni az elsődleges kulcs attribútuma.
- TABLE: egy adatbázis adattáblájának egy adott mezőjében lévő érték lesz a következő generálandó érték.
- AUTO: érték esetén a fenti három stratégia közül automatikusan választ a keretrendszer, természetesen attól függően, hogy a keretrendszer alatt lévő relációs adatbázis kezelő melyik megoldást támogatja.

A keretrendszer összetett kulcsok definiálására is lehetőséget biztosít az @IdClass annotáció segítségével. Mivel az entitások az adattáblák leképezései és az adattáblák között bizonyos kapcsolat létezik, ezért a környezet biztosítja számukra, hogy az entitások között is - hasonlóan az adattáblákhoz, ezt a kapcsolatot meg tudjuk valósítani. Ez szintén a JPA segítségével oldható meg a megfelelő annotációk alkalmazásával. A @OneToOne, @OneToMany, @ManyToOne @ManyToMany annotációkat kell alkalmaznunk, melyet a kapcsolat másik végét reprezentáló attribútumra vagy metódus alapú elérés esetén a megfelelő setter és getter metódusra kell alkalmazni. A kapcsolat lehet egyirányú vagy kétirányú. Egyirányú esetében csak az egyik entitásból lehet elérni a másik oldalt. Egyirányú esetében a tulajdonos oldal egyértelmű. Kétirányú esetén az alábbi szabályok érvényesek:

- Egy-egy kapcsolat esetén az az oldal a tulajdonos, amelyik az idegen kulcsot tartalmazza.
- Egy-több kapcsolat esetén a "több" oldal a tulajdonos.
- Több-több kapcsolatban bármelyik oldal lehet tulajdonos.

5.6.3. Entitások életciklusa

Az entitás mindig a 5.3 ábrán látható állapot valamelyikében található. [8]

5.6.4. Perzisztencia provider

A perzisztenci provider biztosítja az entitások életciklusának kezelését, az adatbázis és memóriabeli adatok szinkronizálását. Alapvető interfészeket definiál, mely mögött tetszőleges megvalósítás állhat. Elterjedt perzisztencia providerek pl. TopLink - mely a Sun [9] által készített referencia implementáció -, Hibernate, OpenJPA. Az

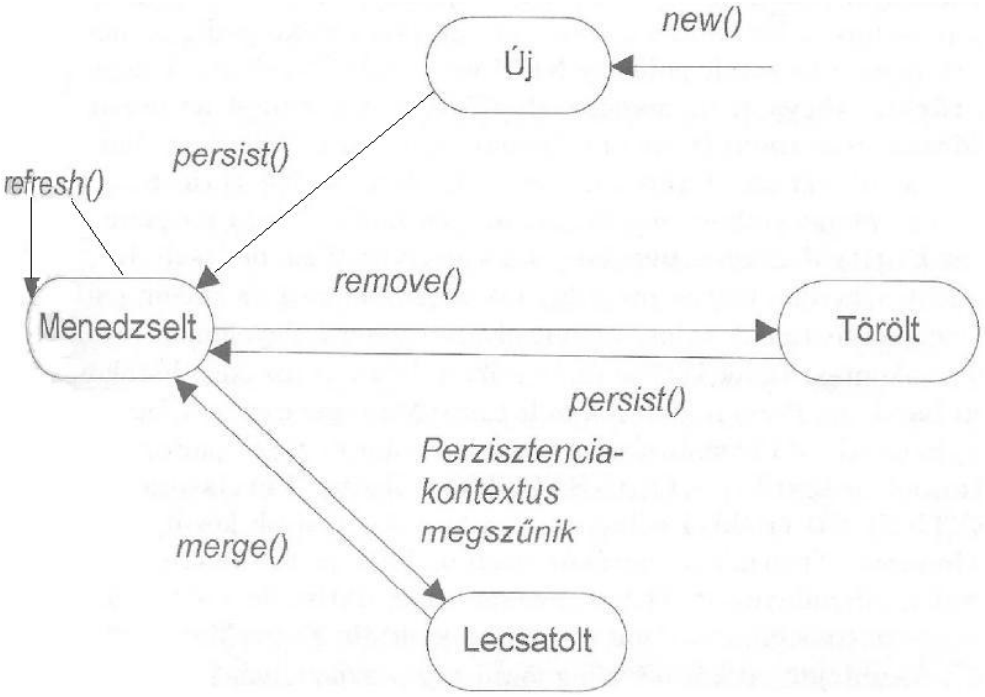
[8] Forrás: Imre Gábor: Szoftverfejlesztés Java EE platformon

[9] http://www.sun.com

entitásokat az EntityManager interfészén keresztül kezeljük. Minden EntityManager példány entitások olyan halmazával dolgozik, mely minden elsődleges kulccsal rendelkező perzisztens példányhoz egyetlen egyedi memóriabeli objektumpéldány tartozik. Az ilyen entitáshalmazokat perzisztenciakontextusoknak hívjuk.

5.6.5. JSF

A JavaServer Faces technológia az egyik legelterjedtebb webes keretrendszer. A Java EE technológia része, ezért az alkalmazásszerve alapértelmezetten tartalmazza. A legtöbb fejlesztői környezet támogatja e technológiát olya módin is, hogy drag and drop módon összerakhatunk vele felhasználói felületeket is. Mindezt akár még azelőtt, hogy a mögöttes üzleti logika elkészült volna. A technológia támogatja a saját JSF komponensek fejlesztését is, így rengeteg megoldás létezik belőle, a legváltozatosabb komponenseket találhatjuk meg a különböző gyártók és szervezetek kínálatában. Több komponenst



5.3. ábra. Entitások életciklusa

is kipróbáltam, de a generálandó kód mennyisége miatt maradtam a standard komponenseknél. A későbbiek folyamán, ha mégis szükség lenne valamelyik másik komponens extra szolgáltatásaira, nagyon könnyen le lehet majd cserélni az alap komponenseket az üzleti logika módosítása nélkül. Bár a JSF technológiát nagyon könnyű használni, a fejlesztés során rá kellett döbbennem, hogy a mögöttes technológiák ismerete nélkül, melyekre épül a JavaServer Faces - bár meg lehet lenni -, de a hatékony fejlesztéshez és a JavaServer Faces használata során felmerülő hibák gyors és hatékony felderítéséhez és kijavításához ezen technológiák ismerete nélkülözhetetlen. Így el kellet mélyednem a JSP (Java Server Pages) és a Servlet technológiákban. Annak ellenére, hogy a Java Servlet technológia nem volt ismeretlen számomra, mivel az előző szakdolgozatomat ebben a témában írtam, a Java EE 5.0 verziójában történt változásokat, amelyek a J2EE[10] verzió után kerültek bele, a kellemetlenségek megelőzése érdekében át kellet tanulmányoznom. Mivel a JavaServer Faces nem közvetlenül a Java Servletekre épül, hanem van egy köztes technológia a JSP (Java Server Pages) mely a Servletekre épül, át kellet tanulmányoznom a JSP technológiát is.

5.6.5.1. Servlet

A Servletek olyan speciális Java osztályok, melyek segítségével hatékonyan és könnyen fejleszthetünk dinamikus tartalmakat generáló szerveroldali megoldásokat. Általában webes alkalmazások tartalmának generálására használják, de lehetőség van a Servletek segítségével valamilyen rich kliens megoldás kiszolgálására is.

5.6.5.2. JSP

A JSP (Java Server Pages) a Servleteket kiegészítő szerveroldali technológia. A hangsúly a kiegészítésen van, vagyis a JSP a Servletekre épül, Servletek segítségével van megvalósítva. Milyen előnyökkel jár a JSP a Servletekkel szemben: A Servletekben a Java kódba vannak beleágyazva a HTML jelölőelemek melyek így jelentősen rontják az átláthatóságot,

[10] J2EE: A Java EE 5 előző verziójának neve.

kezelhetőséget. A Java Server Pages bevezetésével ezt a keveredést szüntették meg úgy, hogy egyszerű szöveg fájlként statikus elemek segítségével állít elő dinamikus tartalmat.

6. fejezet

Rendszerteszt

A rendszertesztelés a szoftverfejlesztés jelenlegi szakaszában ad-hoc jelleggel zajlik. Néhány lelkes kolléga vállalta, hogy a napi rutinjait a "kézi" feldolgozás mellet elvégzi az elkészült szoftvermodulokkal. Így ellenőrizni tudjuk a szoftver helyes működését, mert a régi jól bevált folyamatokkal egy időben zajlik a teszt így rendelkezésre állnak pontos tesztadatok is, meg tudjuk állapítani, hogy a szoftver megfelel-e a specifikációnak. Mivel egy teljes év van a tesztelésre - ugyanis a szoftver éles környezetben való használata csak a 2010/2011-es tanévben kerül sor, nem jelent túl nagy problémát, hogy nem képzett informatikusok, profi tesztelők végzik a funkcionalitás ellenőrzését és a hibakeresést. De ez kizárólag azért van így, mert egy teljes tanéven keresztül tesztelhetjük és fejleszthetjük még a rendszert. Ha a határidő rövidebb lenne egy tervezett tesztelési eljárás nélkül, melyet szakemberek végeznek, nem lehetne hatékonyan megoldani. Kifejezetten örültem, hogy a végfelhasználók - akik nem informatikai szakemberek - is tesztelik a rendszer, mert 10 éves fejlesztői pályafutásom során tapasztaltam, hogy a végfelhasználók hihetetlen érzékkel találják meg a hibát. Olyan dolgokat sikerül "előhozniuk" a rendszerből, amire a fejlesztők és tesztlelők soha nem is gondoltak volna. Természetesen a képzett tesztelők hiánya érezhető volt. Főleg a kommunikációban, az észlelt hibák jelentésében. Nagyon nehezen tudták megfogalmazni, ha valami ötletük volt egy adott funkció javításával, egyszerűsítésével kapcsolatban. A hibákat nem tudták megfelelően dokumentálni, így

nehézkes volt a hibakeresés. A programegységek, mint a függvények vagy az objektumok tesztelését az implementáció után elvégeztem. Már az implementálás kezdetétől szerencsés lett volna valamilyen tesztautomatizálási módszert bevezetni. Sajnos ilyen irányú ismereteim elég hiányosak voltak és az idő szűkössége miatt úgy gondoltam nem éri meg az implementációt késleltetni amiatt, hogy elmélyülök a tesztautomatizálásban. Így utólag visszagondolva az implementáció közben a programegységek tesztelésére fordított rengeteg idő egy részében jobb lett volna a tesztautomatizálásra is kellő figyelmet és energiát fordítani, és lehet még a módszer megismerésére és elsajátítására fordított idővel együtt is kisebb energia és idő ráfordítással lehetett volna végezni a programegységek tesztelésével. A jövőben tervezem bevezetni a JUnit[1] használatát. A JUnit Java osztályok olyan halmaza, amelyet a felhasználó kibővíthet annak érdekében, hogy létrehozzon egy automatikus tesztkörnyezetet. A JUnit támogatott a legnépszerűbb fejlesztőeszköz által, mit az Eclipse és a Netbeans. A két alapvető tesztelési tevékenység, a komponenstesztelés, vagyis a rendszer egyes részeinek a tesztelése melyet a fejlesztő végez, viszonylag jól működik, a hiányosságokat a tesztautomatizálással részben orvosolni lehetne. A másik a rendszertesztelés melyet jelenleg a rendszer leendő felhasználói végeznek viszont nem jó megoldás, rendkívüli mértékben hátráltatja a szoftverfejlesztési folyamatot. Ezen változatni kell a jövőben és egy független tesztelő csoportra kell bízni a tesztelést. A legnagyobb hiányossága a tesztelési folyamatnak, hogy nincs tervszerűsítve. A szoftverfejlesztési folyamatban a rendszertesztre nagyobb figyelmet kell a jövőben fordítani és nem ad-hoc jelleggel kell a tesztelést végezni, hanem tervszerűen, valamilyen bevált tesztelési módszert alkalmazva.

[1] http://www.junit.org

7. fejezet

Működtetés és karbantartás

A működtetésnél is szinte ugyan azok a kérdések fogalmazódtak meg első körben, melyek az implementációnál. Milyen alkalmazás - és adatbázis szervert alkalmazunk? A legelterjedtebb megoldások az alkalmazásszerver tekintetében:

- Sun - GlassFish
- JBoss Application Server[1]
- Apache - Geronimo[2]
- BEA Systems - WebLogic Application Server[3]
- IBM - WebSphare Application Server[4]
- Oracle Application Server[5]

Az utolsó hármat oktatási intézmények esetében, mivel kereskedelmi termékekről van szó magas árazása miatt első körben elvetettük. Az Apache Geronimo szintén kiesett a

[1] http://www.jboss.org/jbossas
[2] http://geronimo.apache.org
[3] http://www.oracle.com/bea/index.html
[4] http://www-01.ibm.com/software/webservers/appserv/was
[5] http://www.oracle.com/us/products/middleware/application-server/index.htm

listából mivel nem volt vele üzemeltetési tapasztalatunk. Így maradt a Sun GlassFish és JBoss Application Server. ? Adatbázis kezelők tekintetében is több lehetséges alternatíva is szóba került:

- Oracle[6]
- IBM DB2[7]
- MS SQL Server[8]
- MySQL[9]
- PostgreSQL[10]
- Apache - Derby[11]

Az első kettő szintén kereskedelmi volta és magas árazása miatt esett ki a rostán. MS SQL Server licencel rendelkezik az intézmény a Tisztaszoftver program jóvoltából, de ha a program megszűnik, akkor le kell cserélni, így célszerű lenne elkerülni a használatát. A MySQL ingyenes robosztus RDBMS jó választásnak tűnik minden szempontból. Felmerült még lehetőségként a Derby melyet a fejlesztés során is használtunk és integrálva van a GlassFish alkalmazás szerverbe és elegendő teljesítménypotenciál van benne, hogy az intézmény igényeit kiszolgálja. A legkézenfekvőbb és talán a legoptimálisabb megoldásnak a Sun GlassFish alkalmazás szerver az integrált Derby RDBMS-el tűnik, így ez kerül bevezetésre az intézményben, mely igény szerint bármikor lecserélhető más megoldásra, ha szükséges.

[6] http://www.oracle.com/us/products/database/index.htm
[7] http://www-01.ibm.com/software/data/db2
[8] http://www.microsoft.com/sqlserver/2008/en/us/default.aspx
[9] http://www.mysql.com
[10] http://www.postgresql.org
[11] http://db.apache.org/derby

8. fejezet

Összefoglalás

Az eddig elkészült rendszer bár még közel sem érte el teljes funkcionalitását, elérte célját. Az iskola dolgozóinak nagymértékben meg fogja könnyíteni a munkáját, ha ténylegesen bevezetésre kerül a 2010/2011-es tanévben. Tényleges bevezetésen a hivatalos formában történő bevezetést értem, mely az iskolavezetés és a fenntartó jóváhagyásával és támogatásával történik. Természetesen, ha hivatalosan mégsem kerülne bevezetésre, vagy esetleg csúszna egy tanévet, az intézmény dolgozói akkor is hasznát tudják venni. Ha teljesen elkészül, minden feladatukat el tudják végezni, amit eddig nehézkes, kézi munkával kellet megtenniük. Meglepő módon néhány intézmény is érdeklődik a rendszer iránt, minden konkrét tájékoztatás nélkül. Rövid beszélgetés után, mikor megtudták milyen terheket lehet levenni a vállukról némi informatikai támogatással, rögtön elkezdtek érdeklődni a lehetőségek iránt. Látok jövőt a rendszer számára, ha csak pár intézmény tudja hasznosítani, már megérte elkészíteni, azon túl, hogy a szakdolgozat kedvező elbírálás esetén közelebb juttathat a diplomához. Keresnem kell egy fejlesztőtársat, aki a jövőben a fejlesztések nagy részét viszi tovább, mert márciustól egy nagy projektben veszek részt és emiatt nagyon kevés időt tudok majd fordítani a felmerülő új igények és az eddig tervbe feladatok megvalósítására. A fejlesztés során sikerült felfrissítenem Java-s ismereteimet, új ismereteket szereznem. Sokat segített a szakdolgozat elkészítése abban, hogy megfelelően fel tudjak készülni a Sun SCJD vizsgára, és nem csak a vizsgára való

felkészülésben töltött be jelentős szerepet, hanem terveim szerint a rendszer lesz az alapja annak az alkalmazásnak melyet az SCJD egyik vizsgarészeként kell elkészítenem.

Irodalomjegyzék

[1] at all, *Java 2 Útikalauz programozóknak 5.0*, ELTE TERMÉSZETTUDOMÁNYI KAR , 2009., ISBN: 9789630640923

[2] Imre Gábor, *Szoftverfejlesztés Java EE platformon*, Szak Kiadó , 2007., ISBN: 9789639131972

[3] Ian Sommerville, *Szoftverrendszerek fejlesztése*, Második, bővített, átdolgozott kiadás, Panem Kiadó, 2007., ISBN: 9789635454785

[4] Harold Störrle, *UML 2*, Panem Kiadó, 2007., ISBN: 9789635454655

[5] GlassFish OpenSource Application Server, *https://glassfish.dev.java.net*, weboldal

[6] Java Persistence API FAQ, *http://java.sun.com/javaee/overview/faq/persistence.jsp*, weboldal

[7] JSFMatrix, *http://www.jsfmatrix.net*, weboldal

[8] Java Forum, *http://javaforum.hu*, weboldal

9. fejezet

Függelék

9.1. 1. számú melléklet

Tisztelt Kollégák!

Idén vagyok utolsó éves hallgató a Debreceni Egyetem Programtervező Informatikus mester (MSC) képzésén. A szakdolgozatom elkészítéséhez kérem a segítségeteket. A szakdolgozatom címe: Web alapú környezet fejlesztése - Iskolai adminisztrációs rendszer. Célom egy olyan informatikai rendszer fejlesztése, mely nagymértékben automatizálja egy iskola adminisztrációs és egyéb feladatait. Hogy miben számítok a segítségedre? Ha van olyan feladatod, amely sok adminisztrációs munkát igényel a részedről, rengeteg időd megy el vele, sok papírmunkával jár, keress meg és megpróbáljuk informatikai eszközök segítségével megkönnyíteni - optimális esetben teljesen automatizálni. Nézzünk egy példát mire gondolok: Hiányzások! Nagyon sok macera van vele. Össze kell számolni a naplóban. Ha sok a hiányzás, akkor értesíteni kell a szülőt, jegyzőt, stb. Számolnod kell, levelet kell írnod ... . De ha lenne egy elektronikus naplónk, amelyben vezetnénk a hiányzásokat, akkor a rendszer mindent kiszámolna helyetted. Szólna, hogy L V-nak, K O-nak, O V-nak és Gy F-nek x óra hiányzása van, levelet kell küldeni a szülőknek, jegyzőknek, stb. A rendszer meg is írná a leveleket egy sablon alapján, neked csak ki kellene nyomtatni, aláírni és elküldeni - még a borítékcímzést is el tudná végezni a rendszer helyet-

ted. Vagy másik példaként említhetnénk mindenki kedvenc kéthavi teljesítmény kimutatását. Neked nem kellene kézzel kitölteni, az illetékesnek nem kellene 80 pedagógusét rögzíteni, átnézni. A rendszer mindezt automatikusan végezné. Kedves kolléga, ha vannak ilyen és ehhez hasonló ötleteid - legyen az bármilyen extrém is - és azt megosztanád velem, hogy minél színvonalasabb munkát tudjak felmutatni a szakdolgozatommal, szívesen venném javaslataidat és nagyon hálás lennék neked. Előre is köszönök minden segítséget. Tisztelettel Bodnár József(leendő Programtervező Informatikus Mester:))

9.2. 2. számú melléklet

Órák adminisztrálása

==========

Napló. Automatice felajánla, ha van tanmenet az aznapi órára beírandó dolgot. Ha elmaradt egy óra akkor a naplóba ez bevezethető. HA tanítás nélküli nap van akkor a napló nem enged oda beírni órát. Kell egy hibalista ha esetleg valaki mégis írt be oda órát akkor az kiderülhessen. A havi/kéthavi teljesítménykimutatás készítésekor, illetve a rendszerbe való belépéskor figyelmeztesse a tanárt. A órákat számozza autómatikusan. Engedjen a tanmenetből egy órára tobb órai anyagot is beírni. Ezt tudja megjelölni a tanár, hogy szándékosan csinálta. Hibalista, ami jelzi, hogy adott óraszámhoz nem a hozzá tartozó tanmenet szerinti óra lett megtartva. Tudja a tanár ezt esetleg érvényesíteni ha ez szándékosan történt. Tudjon a tanmenetben szereplő helyett ill. mellé új/más/kiegészítő tananyagot írni. Tanmenetben rövid név ha esetleg év végén napló kerülne kinyomtatáásra akkor ne foglaljon sok heleyet. Csak az a tanár írhat az adott órához aki tartja azt az órát. A helyettesítést hivatalosan kell megtenni a rendszerben, és a rendszer autómatikusan bevezeti legyen az baráti vagy hivatalos, ill. elmaradt óra. A helyettesítést jóvá kell hagyni valakinek (ehhez jogosultságot kell adni v.melyi, usernek, ig., ig.h., munkaköz.vez). A helettesítést kezdeményezheti a tanár maga, a munkaköz vez. vagy ig., ig.h. A kijelöt helyettesítőnek ezt jóvá kell hagynia (bejelentkezés után megjeleni és vagy elfogadja vagy visszautasítja a felkérést.) és belejentkezéskor figyelmezteti x idővel, hogy helyettesíteni fog.

SMB (Small Message Board Funkció)

==========

Lehetőleg RSS-el megoldva, hogy ne csak az LCD-n hanem máhol is meg lehessen tekinteni Ha RSS-el van megoldva akkor a desktopra ki lehet tenni RSS olvasókat és nem kell külön Java Aplettel vagy valami full screense browserrel megoldani Több féle információ megjelenítése több féle elrenedezésben. Adott időszakban esetleg más-más információk,

más más gyakorisággal frissítve(pl. szünetben más infók jelennek meg mint tanórákon). - Hírek - Napi események - Ügyeletes tanárok - Szabad termek - Helyettes H-betűs, ill. adott órán/következő x órán/napon kik kiket helyettesítenek.

Egyéb

=====

Szőlő értesítése - emailben, esetleg SMS-ben(Jegyek, figyelmeztetése, egyéb értesítések / közlemények, azonnal vagy akár, heti/havi/féléves rendszerességgel).

Ábrák jegyzéke