

DIPLOMAMUNKA

Diószeghy Szilárd

**Debrecen
2010**

Debreceni Egyetem
Informatika Kar
Információ Technológia Tanszék

Webes alkalmazásfejlesztés

Témavezető:

Kósa Márk Szabolcs
egyetemi tanársegéd

Készítette:

Diószeghy Szilárd
programtervező matematikus

Debrecen
2010

Tartalomjegyzék

1. Bevezetés.....	4
1.1 Bevezető.....	4
1.2 Témaválasztás.....	5
2. Alkalmazásfejlesztés	7
2.1 A szoftver.....	7
2.1.1 A szoftver minősége.....	7
2.2 Modellezés	8
2.3 UML.....	9
2.4 A fejlesztés fázisai	10
2.5 Egységesített eljárás.....	11
3. Követelmények.....	14
3.1 Követelmények feltárása.....	14
3.1.1 Felhasználói követelmények	14
3.1.2 Rendszerkövetelmények.....	16
3.2 Fogalomszótár.....	16
3.2 Aktorok	17
3.3 Használati esetek.....	18
3.3.1 Adminisztrátor	19
3.3.2 Hallgató.....	22
3.3.3 Oktató	26
3.4 Követelmény feltárás eredménye.....	28
4. Elemzés	29
4.1 Követelmények elemzése	29
4.2 Objektum modell	30
4.2 Dinamikus modell.....	32
4.3 Funkcionális modell.....	34
4.4 Elemzés eredménye	34

5. Tervezés	35
5.1 A tervezés feladata.....	35
5.2 Architektúrális tervezés	36
5.2.1 A többretegű technológia.....	36
5.2.2 A rendszer architektúrájának megtervezése	38
5.3 Felülettervezés	40
5.4 Objektumtervezés	42
5.5 A tervezés eredménye	44
6. Megvalósítás	45
6.1 Java Enterprise Edition	45
6.2 Enterprise JavaBeans	46
6.2.1 Entitások	47
6.2.2 Entitások menedzselése	48
6.2.3 Entitások élettartama	49
6.2.4 Entitások elérése	50
6.2.5 Entitások közötti kapcsolatok	52
6.2.6 Az üzleti logika.....	53
6.2.7 Session bean.....	53
6.2.8 Callbacks metódusok	56
6.2.9 Dependency injection	56
6.3 JavaServer Faces.....	58
6.3.1 A JSF keretrendszer felépítése.....	59
6.3.1.1 JavaServer Pages	59
6.3.1.2 Menedzselt bean	62
6.3.2 A JSF élekciklusa	63
6.3.3 Navigáció	65
7. Tesztelés	66
8. Összefoglalás	67
9. Irodalomjegyzék	68
10. Függelék.....	69

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani témavezetőmnek, Kósa Márk tanár úrnak, aki hasznos tanácsaival, észrevételeivel segítette a diplomamunkám megírását és szakmai kérdésekben is segítségemre volt. Egyben köszönöm mindazon tanáraimnak a fáradságos munkáját, akik segítettek az egyetemi éveim alatt, átadták azt a tudást és tapasztalatot, amivel remélem, olyan útravalóval láttak el, ami elegendő lesz a pályakezdéshez.

1. Bevezetés

1.1 Bevezető

Az idők során a kutatók mindig arra törekedtek, hogy újabb és újabb felfedezéseikkel segítsék, megkönnyítsék az ember mindennapi munkáját.

A XX. században az ipar és a technika robbanásszerű fejlődésen ment keresztül, megindult a nemzetek közötti verseny, a nagyhatalmak nemcsak politikailag és gazdaságilag, hanem technikailag is igyekeztek a világhatalom megszerzésére. Az informatika, a távközlés és a számítástechnika fejlődése, akár hardveres, akár szoftveres területekre gondolunk, lényegesen megváltoztatta az emberek életmódját, életfelfogását.

Napjainkban a szoftvergyártás az egyik legdinamikusabban fejlődő üzletággá vált, hiszen életünk minden területét uralják azok a berendezések, kisebb-nagyobb számítógépek, mikrovezérlők, amelyek különböző programok segítségével működnek.

A számítógép megjelenése óta óriási fejlődésen ment keresztül, azonban egy dolog nem változott, mégpedig az, hogy a számítógépet manapság is szoftverek segítségével vezéreljük, ezért elengedhetetlen feladat, hogy megbízható és egyre hatékonyabb szoftvereket készítsünk. Kezdetben a szoftverfejlesztés nehézségekbe ütközött, melynek oka, hogy az elkészült programok nagy része használhatatlan volt. A hiba eredendő okát a fejlesztés módszerességének és szervezésének hiányában találták meg. Ezután már elkezdődhetett a szoftverfejlesztés technológiájának kialakítása. Számos programfejlesztési módszer, eszköz, technika látott napvilágot, melyek mind a hatékony fejlesztést hivatottak elősegíteni. Napjaink programfejlesztési szemléletét az objektumorientáltság jellemzi, ennek köszönhetően a fejlesztők egyre jobb szoftvereket hoznak létre. A számos alkalmazási területen belül, a webes alkalmazásfejlesztés a szoftverkészítés egyik kiemelkedő ága. Az interneten elérhető alkalmazások egyre népszerűbbek és változatosabbak, melyet az egyre nagyobb sebességű világháló tesz lehetővé.

Az internet egyik legnépszerűbb szolgáltatása a World Wide Web. Kezdetben a webes alkalmazások statikus HTML lapok voltak, azonban a technika fejlődésével és az elérési sebesség egyre gyorsabbá válásával a felhasználók igényei is megváltoztak. Létrejötték az első web áruházak, online bankok, amelyek a szolgáltatások forradalmasításával jelentősen

megkönnyítették mindennapjainkat. Manapság az életünk már szinte elképzelhetetlen internet nélkül: átalakította a tér és idő fogalmát.

1.2 Témaválasztás

Egy alkalmazás fejlesztése hónapokat, sőt éveket vehet igénybe, gondoljunk csak bele, hogy mennyit kell várni egy új operációs rendszer vagy egy újabb játékszoftver megjelenéséig. A fejlesztői munka egyre nagyobb része gépesítve van, de a számítógép, legalábbis egyelőre, nem tud helyettünk gondolkodni, ezért a munka lényegi részét mégiscsak az embernek kell elvégeznie. Ebből adódóan előfordulhatnak hibák is a programban, de az ember éppen ezért más mint a gép, hogy ezeket a problémákat kezelni tudja.

Dolgozatom témája a felsőoktatásban használatos tanulmányi rendszerrel kapcsolatos. Célom az, hogy bemutassam egy fejlesztési folyamat egyes fázisait, az igény felmerülésétől a kivitelezésig. Az igény az volt, hogy megkönnyítsék a felsőoktatásban tanulók különböző ügyeinek intézését, ezért tanulmányi rendszert vezettek be, melyek az idők folyamán fokozatosan cserélődtek. Minden egyes rendszer próbált újításokkal szolgálni az előzőekhez képest, akár a gyorsaságra, akár a funkcionalitásra gondolunk, de a különböző hiányosságok miatt mégsem érték el tökéletesen a kívánt célt.

Dolgozatomban bemutatok egy általam fejlesztett tanulmányi rendszert, melyben két hasznos újítást kívántam megvalósítani.

- Az egyik a fiktív gyakorlatok problémájával kapcsolatos
- A másik a diplomamunka és szakdolgozat témák tanulmányi rendszerben történő meghirdetése

Fiktív gyakorlatok megszüntetése

A jelenlegi tanulmányi rendszer a tárgyfelvétel esetén nem tesz különbséget azon hallgatók között, akik aláírással rendelkeznek, de az adott tárgyat még nem teljesítették, illetve azok között, akik nem rendelkeznek aláírással az adott tárgyból. Ezért a mostani rendszerben azok részére, akik már a korábbi években megszerezték az aláírást egy „fiktív gyakorlatot” hirdetnek meg. Az aláírással rendelkező hallgatóknak ezt a gyakorlatot feleslegesen kell felvenniük, mivel számukra a rendszer nem teszi lehetővé annak elhagyását.

Megoldás: az aláírással rendelkező hallgatók számára lehetővé tenni a gyakorlat elhagyását.

Diplomamunka és szakdolgozat témák tanulmányi rendszerben történő meghirdetése:

Jelenleg a diplomamunka és szakdolgozat témák nem a tanulmányi rendszeren keresztül, hanem egy honlapon kerülnek meghirdetésre. A hallgatók egy kérdőív segítségével jelentkezhetnek a kiválasztott témára, amelyet a tanulmányi osztályon kell leadniuk, feltüntetve a témavezetőt és a választott téma címét.

Megoldás: A hallgatók a tanulmányi rendszerben egy új menüpont alatt tekinthetik meg a kiírt diplomamunka és szakdolgozat témákat, és ugyanitt jelentkezhetnek is rá.

Célom egy olyan rendszer megtervezése és implementálása volt, mely alkalmas ezen módosítások szemléltetésére. Az implementációhoz a Java nyelvet választottam, mivel tanulmányaim során számos kisebb alkalmazást fejlesztettünk ezen a nyelven, így egyrészt jó alapokkal rendelkeztem ebből a nyelvből, másrészt - ismereteim bővítése céljából - szerettem volna megismerni olyan java alapú technológiákat, melyek segítségével többretegű webes alkalmazás fejleszthető.

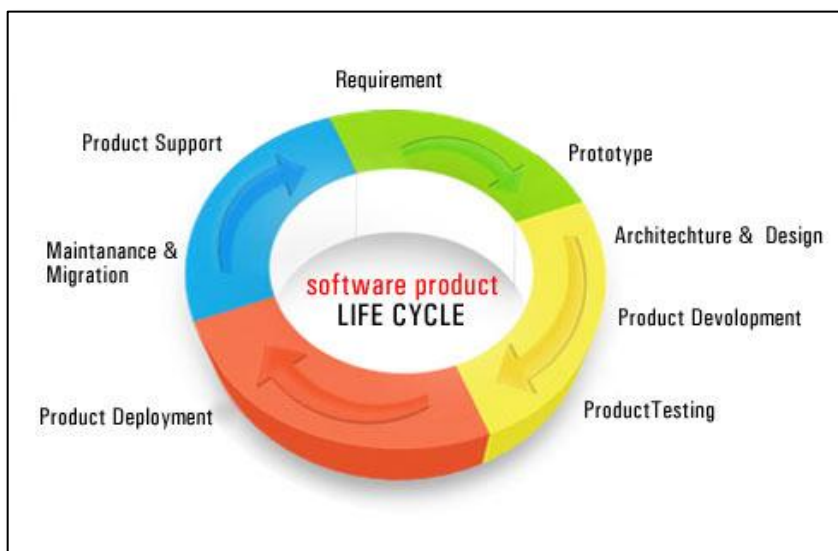
2. Alkalmazásfejlesztés

2.1 A szoftver

A szoftver egy szellemi termék, amely alatt a programok, adatok és dokumentációk összességét értjük. Az elnevezés az angol *hard* - *soft* ellentétpárból adódik:

- a hardver gyártás után már nem változtatható
- a szoftver könnyedén módosítható, továbbfejleszthető valamilyen cél érdekében

Egy szoftver élettörténete, életciklusa az igény felmerülésétől a termék használatból való kivonásáig tart. Általában életciklusról beszélhetünk, mert az alkalmazásokat többször továbbfejlesztjük. Az **1. ábra** szemlélteti a szoftver életciklusát.



1. ábra. Szoftver életciklusa

Az egyes szakaszok súlya jelentősen eltér egymástól. A fejlesztés a szoftver élettörténetének egyik legmeghatározóbb szakasza, szemben például az előállítással, ami már nem okoz különösebb problémát.

2.1.1 A szoftver minősége:

A szoftver minősége külső és belső tényezőktől függ. A külső tényezők határozzák meg, hogy a szoftvernek milyen feltételeknek kell megfelelnie, a belső tényezők a kódolási

módszert jellemzik. A végeredmény tekintetében csak a külső tényezők számítanak, viszont ezek a belső tényezők függvényei.

Egy jól felépített alkalmazás külső és belső tulajdonságai:

Külső tulajdonságok:

- *Helyes:* a szoftver helyes, ha a feladat specifikációja szerint működik
- *Robosztus:* ha a szélsőséges eseteket is kezelni tudja, vagyis elég stabil
- *Karbantartható, bővíthető:* ha könnyen javítható, és az újabb felhasználói igények könnyen bevezethetők
- *Felhasználóbarát:* ha megjelenése kellemes, használata kényelmes, egyértelmű
- *Kompatibilis:* más szoftverekkel képes együtt működni
- *Hordozható:* ha könnyedén átvihető más hardver, szoftver környezetbe
- *Hatékony:* ha a hardver az erőforrásokat maximálisan kihasználja

Belső tulajdonságok:

- *Áttekinthető:* érthetően, tiszta logikával kódolt
- *Moduláris:* az egyes részek tagoltsága, külön feladatok ellátásának képessége

Egy alkalmazás hatékonyságát a külső jegyek határozzák meg, a felhasználó számára az a fontos, hogy a program helyesen, gyorsan és megbízhatóan működjön.

A fejlesztés első fázisa még elméleti síkon folyik, melynek fő célja a modellek kialakítása. Az alkalmazásfejlesztés nem egy ember munkája, hanem egy csapat összehangolt, precíz tevékenysége, ezért a fejlesztők között elengedhetetlen a megfelelő kommunikáció, információcsere, az elért eredmények dokumentálása, és kulcsfontosságú, hogy a feladatról mindenkiben ugyanaz a kép alakuljon ki.

2.2 Modellezés

Modellezés során a valós világról egy gondolati képet alakítunk ki, amit modellnek nevezünk. Modellalkotáskor azokra a tulajdonságokra koncentrálunk, melyek fontosak a cél elérése szempontjából. Azonban egy valós dologról több modell is készülhet, melyek mindegyike más-más nézőpontból világítja meg a problémát.

Legtöbbször komplex rendszerekkel foglalkozunk, így nem tudjuk az egész problémát egyszerre átlátni, ezért olyan modelleket kell kialakítanunk, melyekkel a bonyolult rendszereket még kezelni tudjuk. Ebben segít az absztrakció és a dekompozíció.

■ *Absztrakció*

Ennek során a fontos részleteket kiemeljük, a kevésbé fontos részekről pedig eltekintünk. Az absztrakciónak különböző szintjei vannak. Minél inkább közeledünk a magasabb szintről az alacsonyabb felé, annál jobban koncentrálnunk az apró részletekre.

■ *Dekompozíció*

A rendszert egyszerűbb részekre bontjuk, melyek egymással együttműködve adják az eredeti rendszer funkcionalitását.

- *A strukturált szemlélet* alapján történő dekompozíciónál a darabolás a lényeges funkciók alapján történik, melyeket tovább osztva épül fel a hierarchia.
- *Az oo szemléletnél* viszont a valós világhoz közelebb álló objektumokra bontjuk a rendszert, melyek egymással együttműködve valósítják meg a komplex feladatot.

Modellalkotásnál célunk, hogy egyértelmű modellt hozzunk létre, ehhez viszont a modelleket kellőképp formálisan kell leírni. Egy modell szöveges formában való leírása nem elég egyértelmű, ezért grafikus jelölésrendszert használunk, melynek segítségével sokkal pontosabban, precízebben tudjuk kifejezni magunkat.

Egy olyan grafikus modellező nyelvre van szükség, amely mindenki számára egyértelmű információt hordoz. Ilyen például az Unified Modeling Language (UML).

2.3 Unified Modeling Language (UML)

Az UML az objektumorientált gondolkodásmódra épülő vizuális modellező nyelv, melyben grafikus elemekkel absztrakt modelleket tudunk leírni. Az UML korábbi jelölésrendszerek egyesítésével keletkezett, és vált szabvánnyá 1997-ben. Az UML nyelvet szintaktikai és szemantikai szabályok alkotják. A szintaktikai szabályok a teljes szimbólumrendszer formáját és kapcsolódási módjait határozzák meg. A szemantika

szabályok definiálják a szimbólumok, és azok kapcsolatainak értelmezését. A modellek viselkedésének és szerkezetének leírása diagramok segítségével történik. Az, hogy milyen modellek a legalkalmasabbak egy bizonyos helyzetben a kívánt tartalom visszaadására, hogy milyen jelölések vagy jelölési elemek jöhetnek szóba, és hogyan építendők fel a diagramok, képletek és szövegek, számos tényezőtől függ, beleértve az életciklus fázisait, a célt, a rendelkezésre álló médiát és a közönséget. Ezek megítélése sok tapasztalatot igényel.

Elmondhatjuk tehát, hogy UML kellően flexibilis ahhoz, hogy bármely szoftverfejlesztéssel foglalkozó cég használni tudja a saját fejlesztési folyamatához, illetve elég formális ahhoz, hogy a fejlesztők számára nagy pontossággal bírjon. Ezért az UML a szoftverfejlesztők közötti kommunikáció alapja.

2.4 A fejlesztés fázisai

Ugyan minden fejlesztés során más és más terméket állítunk elő, azonban egy szoftver elkészítésének lépései nagyjából azonosak, találunk mérföldköveket, amelyek minden fejlesztésben megtalálhatók.

Lássuk azokat a főbb fázisokat, melyek a fejlesztés során előjönnek:

- *Követelmények feltárása*

Azon feladatok meghatározása, amit a rendszernek meg kell oldani.

- *Elemzés*

Meghatározzuk a rendszer struktúráját, majd a követelményeket úgy rendezzük át, hogy az a további fejlesztés kiindulópontja legyen.

- *Tervezés*

A követelményekben meghatározott feladatot ellátó rendszer megtervezése, az elemzés során kialakított váz kiegészítése.

- *Implementáció*

A tervnek megfelelően a rendszert valamilyen nyelven lekódoljuk, ezzel elkészül az alkalmazás.

- *Tesztelés*

A szoftver minőségének ellenőrzése, illetve hiba, hiányosság esetén módosítás.

2.5 Egységesített eljárás - RUP (Rational Unified Process)

Az első életciklus modellek a 70-es években alakultak ki, alapjukat a fentebb említett fázisok alkották. Manapság már rengeteg modell közül választhatjuk ki a számunkra legmegfelelőbbet. Azonban a modellválasztás sok tényezőtől függ, hiszen minden életciklus modellnek megvannak a maga előnyei és hátrányai, illetve a választást erősen befolyásolja a feladat nagysága is. A sok különböző folyamatmodell között azonban megjelent egy olyan egységesített eljárás, ami elég általános ahhoz, hogy a szoftverfejlesztés valamennyi területén alkalmazni lehessen. Ez a módszer 1997-ben jelent meg az akkori három legelterjedtebb szoftverfejlesztési módszer (*OMT*, *Booch*, *OOSE*) egyesítésével.

A RUP egy nagyon összetett, bonyolult fejlesztési folyamat, mely magában foglalja az előző módszertanok előnyeit. Ennek köszönhetően ez a típusú fejlesztés világszerte nagyon elterjedt. A RUP is mint sok elődje az UML modellező nyelvre épül, és az objektum orientált szoftverfejlesztés irányelveit vallja.

Fontosabb tulajdonságai:

- *Használati eset vezérelt*
Az egyes fejlesztések elején meghatározott használati esetek (use-case) végigkísérik a teljes fejlesztési folyamatot.
- *Robosztus architektúrára összpontosító tervezés*
A fejlesztés során meghatározó szerepe van a rendszer architektúrájának, megkönnyíti az újrafelhasználhatóságot, és a későbbi karbantartást.
- *Iteratív és inkrementális eljárás*
Minden iterációban egy működő verziót készítünk el, ahol először a lényegi részeket valósítjuk meg, a fejlesztés kisméretű egységekben, inkrementekben történik, majd ezt építjük bele a rendszerbe.
- *A fejlesztés ellenőrzött*
A fejlesztés tervezhető, a fejlesztés kockázata és erőforrásigénye megbecsülhető.

Fázisai:

Az egységesített eljárást négy fázisra oszthatjuk, ahol minden fázis egy jól meghatározott mérföldkövel rendelkezik. Ezek a fázisok a következők:

- *Előkészítés, Felmérés (Inception)*

Ebben a fázisban a követelmények feltárása és rögzítése a feladat, meghatározzuk a feladat nagyságát, a legfontosabb használati eseteket, vagyis a fő funkciókat.

- *Kidolgozás (Elaboration)*

Ebben a fázisban az architektúra kialakítása, modulok megtervezése, a köztük lévő kommunikáció kialakítása, az egyes használati esetek részletezése a legfőbb feladatunk.

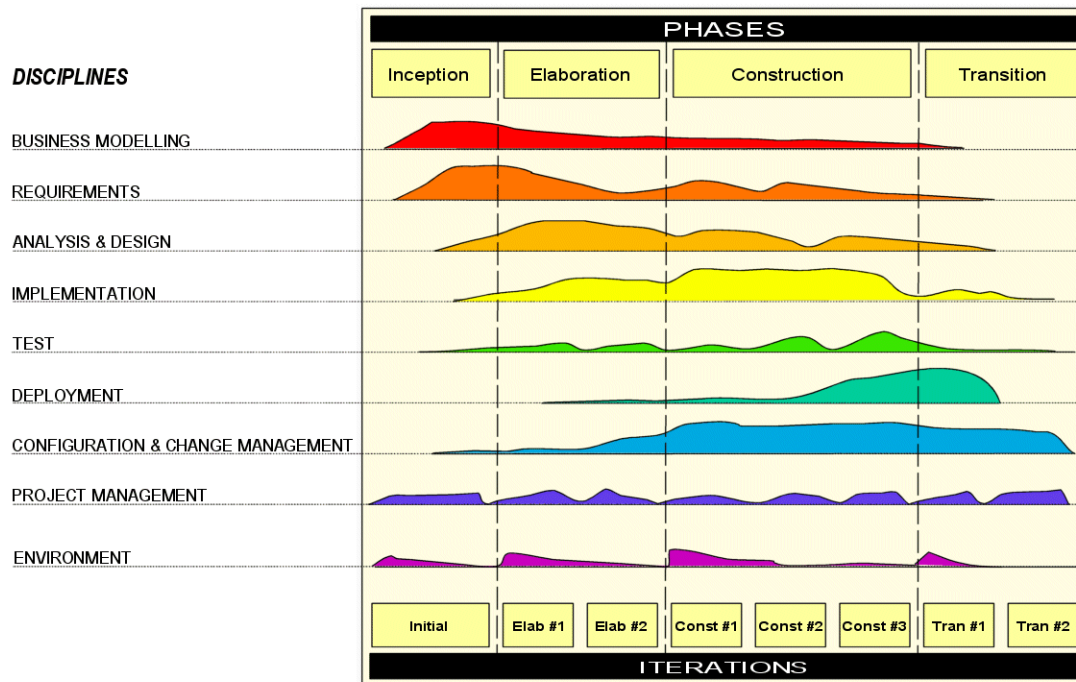
- *Megvalósítás (Construction)*

Itt történik az implementáció, amikor az előző fázisban meghatározott tervek alapján megkezdődik a rendszer megvalósítása. A fázis végén a rendszer egy futtatható verziója is elkészül.

- *Átadás (Transition)*

A fejlesztők átadják a rendszert a felhasználóknak, akik a használat közben felmerülő hibákat jelzik, melyeket a fejlesztők kijavítanak.

A fázisokat további részekre, iterációkra bontjuk. Ezek számát az egyes fázisban az egységesített eljárás nem szabja meg, ez a fejlesztők feladata, mint az is, hogy meghatározzák az egyes iterációk feladatát. Ezek az iterációk egy mini fejlesztésként képzelhetők el, hiszen mindegyik iteráció végén egy működő alkalmazásnak kell előállnia. Persze az újabb iterációkban újabb funkciók készülnek el. Az egyes iterációk során végigmegyünk mind az öt munkafolyamaton (követelmények meghatározása, elemzés, tervezés, implementálás, tesztelés). Minden egyes fázisban más munkafolyamatok fontosak, ahogy ezt a **2. ábra** is mutatja.



2. ábra. A RUP fázisai

Például a megvalósítás fázisban az implementálás kapja a főszerepet. A fázisok végén ott vannak az adott fázist lezáró mérföldkövek. Ezek alapján kell dönteni a folytatásról és értékelni az eredményeket.

3. Követelmények

3.1 Követelmények feltárása

Amikor egy szoftvert tervezünk kezdetben még nem tudjuk, hogy ez mennyi munkát jelent, mennyi ideig tart majd az alkalmazás elkészítése, ezért a munkát azzal kezdjük, hogy rögzítjük, hogy a rendszer milyen szolgáltatásokkal kell rendelkezzen.

A követelményeket kétféleképpen osztályozhatjuk:

- *Felhasználói követelmények*

Itt a funkcionális követelmények szerepelnek, melyek azt írják le, hogy a rendszernek milyen szolgáltatásai legyenek majd, milyen körülmények között működjenek. Például a tanulmányi rendszerben tantárgyat tudjunk felvenni.

- *Rendszerkövetelmények*

Ezek azok a követelmények, melyek a rendszer egészére vonatkoznak, függetlenek az egyes funkcionalitásoktól. Például egy rendszertől elvárjuk, hogy mindig elérhető legyen.

Ahhoz, hogy a követelményeket feltárjuk, beszélnünk kell a szoftver megrendelőjével, illetve azokkal, akik a rendszert használni fogják, ők az úgynevezett *kulcsfigurák*.

3.1.1 Felhasználói követelmények

A felhasználói követelmények meghatározása mindig a kulcsfigurák közreműködésével történik, sokszor előfordul azonban, hogy a megrendelő nem tudja pontosan megfogalmazni az igényeit, vagy épp a fejlesztő nem ismeri az adott szakterületet. Ilyenkor hiányos, esetleg egymásnak ellentmondó specifikációk születnek. Jelen esetben egy tanulmányi rendszert fejlesztünk, amivel az egyetemisták, főiskolások többsége nap mint nap találkozik mint felhasználó, és a mindennapi használat során megismerkedik a rendszer előnyeivel és hátrányaival.

Az általam fejlesztett rendszer nem egy teljes funkcionalitással bíró tanulmányi rendszer, mivel az túlmutatna ezen dolgozat keretein. Azok a funkciók kaptak benne helyet, amelyeket fontosnak tartottam ahhoz, hogy az újítások bemutatathatók legyenek.

- A fiktív gyakorlatok megszüntetése, amely leginkább a hallgatók szempontjából fontos. Az aláírással rendelkező hallgatóknak ezután elegendő az előadást felvenni.
- A diplomamunka téma tanulmányi rendszerben történő meghirdetése, amely több szempontból is hasznos: a hallgatók ezután egy kattintással jelentkezhetnek az általuk szimpatikusnak tartott témára, elkerülve a hosszú sorokban való várakozást; az oktatók saját nevük alatt új témát hirdethetnek meg, és megnézhetik, hogy mekkora az igény az egyes témákra; az adminisztrátorok ezután elektronikusan dolgozhatják fel a jelentkezett hallgatók adatait, nem kell a jelentkezési lapokkal foglalkozniuk.

Mivel a rendszer legfőbb célja, hogy ezt a két újítást bemutassa, ezért a követelmények feltárása során azt vettem figyelembe, hogy a rendszer milyen funkcionalitásokkal kell bírjon ahhoz, hogy ezek szemléltethetőek legyenek.

A rendszer által megvalósítandó feladatok az aktorok alapján három csoportra oszthatók:

- *Hallgatói funkciók:*
kurzusok listázása, kurzusok felvétele, kurzusok leadása, leckeekönyv megjelenítése, kiírt diplomamunka témák megtekintése/ felvétele illetve módosítása
- *Oktatói funkciók:*
oktatott tárgyak listázása, aláírás/jegybeírás, az adott oktató alatt kiírt témák illetve erre jelentkezett hallgatók megtekintése, új téma meghirdetése a rendszerben
- *Adminisztrátori funkciók:*
aktuális félév kiválasztása, kurzusok indítása, kurzusokhoz előadás és gyakorlat rendelése, diplomamunka témák meghirdetése

Az alkalmazás a felhasználó névhez tartozó jogkör alapján dönt, hogy ki milyen szolgáltatásokkal rendelkezik a rendszerben.

3.1.2 Rendszerkövetelmények

Miután a felhasználói követelményeket megvizsgáltuk rögzíteni kell a rendszerrel szemben támasztott követelményeket is. Itt határozzuk meg, hogy a rendszert milyen környezetben fogjuk használni. Mint ahogy azt már említettük a rendszer webes felületen lesz elérhető a felhasználók számára, ezért olyan környezetet kell biztosítani, amely támogatja a webes felületen történő elérést.

A rendszer elkészítéséhez a Java nyelvet választottam, amely futási környezetként a Glassfish v3 alkalmazásszerveret használja. A Glassfish v3 az egyetlen olyan opensource alkalmazás, amely már támogatja a Java EE 6-os szabványokat. Ez azért fontos, mert számos olyan dolgot támogat, amely csak a Java EE 6-tól érhető el (*EJB 3.1, JSF 2.0, JPA 2.0*).

Az adatok perzisztens tárolására a MySQL adatbázist választottam, amely egy könnyen és gyorsan használható ingyenes adatbázisrendszer.

A rendszer nem került fel az internetre, mert az ingyenes webtárhelyek által biztosított alkalmazásszerverek nem támogatják a Java EE 6-os szabványokat, így nem alkalmasak arra, hogy a fejlesztett rendszert futassuk rajtuk.

3.2 Fogalomszótár

Ennek során egy olyan dokumentumot készítünk, melyben a rendszerrel kapcsolatos fogalmakat definiáljuk. A fogalomszótár elkészítése nagyon fontos, mert a követelmények feltárásánál a legnehezebb probléma a kommunikáció. Minden szakterület saját szakzsargonnal rendelkezik, ezért szükség van egy olyan dokumentumra, amely rögzíti és összefoglalja a legfontosabb problémákat, aktorokat, hogy azok mindenki számára ugyanazt jelentsék (*1. táblázat*).

Hallgató	Hallgatói jogviszonnyal rendelkező felhasználó.
Oktató	A hallgatók oktatásáért és teljesítményük értékeléséért felelős személy.
Adminisztrátor	Speciális jogkörrel rendelkező felhasználó. Feladata a rendszer karbantartása.
Félév	Minden félévhez tartoznak a hallgatók számára felvehető kurzusok és diplomamunka témák. Az aktuális félévet az adminisztrátor választja ki.

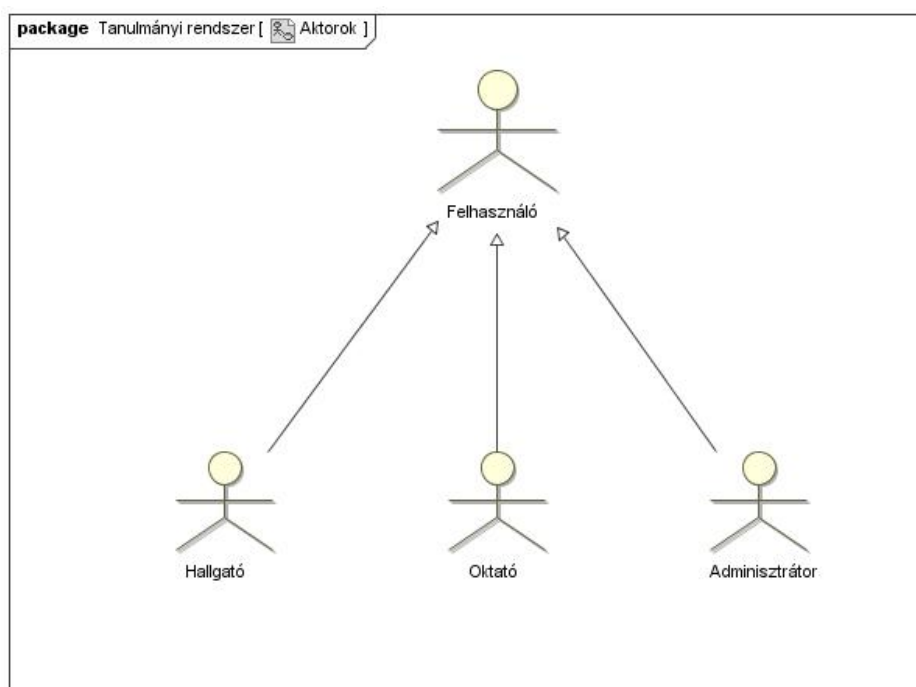
<i>Kurzus</i>	<i>A hallgatók számára felvehető tárgyak. Minden kurzushoz tartozik előadás és esetleg gyakorlat is. Kurzust csak az adminisztrátorok indíthatnak.</i>
<i>Leckekönyv</i>	<i>Minden hallgató rendelkezik egy leckekönyvvel, melyben megnézheti, hogy melyik félévben milyen tárgyakat vett fel és melyeket teljesített.</i>
<i>Előadás</i>	<i>Mindegyik kurzushoz tartozik egy kötelezően felveendő előadás.</i>
<i>Gyakorlat</i>	<i>Egyes kurzusokhoz gyakorlat is tartozik, azok a hallgatók, akik korábbról még nem szerezték meg az aláírást, ezt is fel kell vegyék a kurzusjelentkezés során.</i>
<i>Diplomamunka téma</i>	<i>A hallgatók számára választható diplomamunka címek.</i>
<i>Aláírás</i>	<i>Ha egy hallgató teljesíti a gyakorlatot, akkor megszerzi az aláírást, amely a vizsgán való részvétel előfeltétele. Ha a vizsgát nem teljesíti, akkor a következő alkalommal elegendő már csak az előadást felvennie.</i>
<i>Jegybeírás</i>	<i>Az oktató értékelése a hallgató teljesítményéről.</i>

1. táblázat. Fogalomszótár

3.3 Aktorok

Az aktor mindig egy szerepkört képvisel az UML terminológiában. Egy aktor szerepét nemcsak egy személycsoport, hanem egy hardverelem vagy egy másik alkalmazás is betöltheti, melyekkel a rendszer képes együttműködni. Az aktorokat úgy azonosíthatjuk, hogy megvizsgáljuk, kik azok akik a rendszert használják. A tanulmányi rendszert következő jogkörrel rendelkező felhasználók használhatják:

- *Hallgatók*
- *Oktatók*
- *Adminisztrátorok*



3. ábra. Az aktorokhoz tartozó UML használati eset diagram

Mindegyik aktort egy Felhasználó nevű aktorból származtatjuk, mely három adattaggal rendelkezik: felhasználónév, jelszó, típus. Az egyes aktoroknak egy jól meghatározott célja van a rendszerrel, mely célt a rendszer szolgáltatások szintjén megvalósítja. Ezeket a szolgáltatásokat fogjuk a következőkben use case-ek segítségével bemutatni.

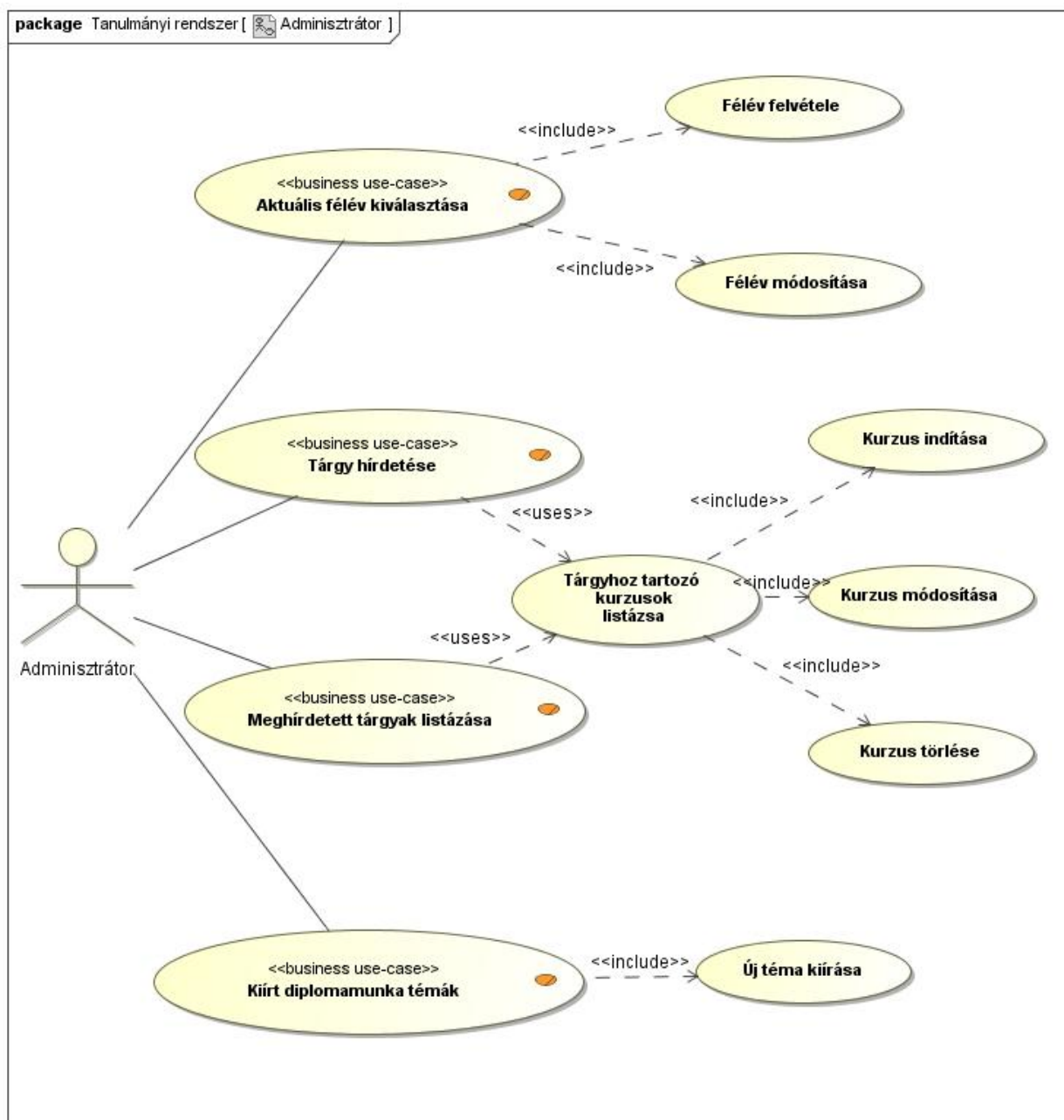
3.3 Használati esetek

A használati esetek olyan diagramok, melyek segítségével a rendszer funkcionalitását mutathatjuk be. Ezek olyan akciók együttese, melyek az aktor számára megfigyelhető eredményt produkálnak. Fontos hangsúlyozni, hogy a használati eset csak a felhasználó és a rendszer közötti együttműködésre koncentrál, figyelmen kívül hagyja a megvalósítás részleteit.

Dolgozatomban a használati eseteket a felhasználói jogkör alapján választottam szét, mert így átláthatóbb feladat-specifikációhoz jutunk. A diagramokat a MagicDraw UML program segítségével készítettem, amely már támogatja az UML 2.0-as verziót is.

3.3.1 Adminisztrátor

Az adminisztrátor speciális feladatkörrel és jogokkal felruházott felhasználó. Feladata a rendszer karbantartása, aktuális félév kiválasztása, kurzusok indítása, kurzusokhoz előadás és/vagy gyakorlat rendelése, diplomamunka témák meghirdetése.



4. ábra. Az adminisztrátorhoz tartozó UML használati eset diagram

Aktuális félév kiválasztása:

A rendszerben egy félév három állapotban szerepelhet: *új, lezárt, aktív*. Az adminisztrátornak lehetősége van az aktuális félév lezárására és új félév aktiválására.

Forgatókönyv:

- *Az adminisztrátor kiválasztja az AKTUÁLIS FÉLÉV menüpontot.*
- *Kijelöli azt az új félévet, melyet aktiválni akar, majd rákattint az OK gombra*
- *A rendszer ellenőrzi a módosításokat, hiba esetén visszajelzést küld (például ha egyszerre több aktív félévet jelöltünk ki).*

Tárgy hirdetése és kurzus hozzáadása:

Az adminisztrátornak lehetősége van az aktív időszakban tárgyak indítására. Ezt a tárgyak indítása menüpont alatt teheti meg. Itt megjelenik az összes tárgy és ezekből az adminisztrátor kiválaszthatja, melyek azok amiket meghirdet. A tárgy kiválasztása után lehetőség van az előadások és a gyakorlatok hozzáadására. Az egyes kurzusoknál be kell állítani az oktatót és a maximális létszámot.

Forgatókönyv:

- *Az adminisztrátor kiválasztja a TÁRGY MEGHIRDETÉSE menüpontot.*
- *Kiválasztja a meghirdetendő tárgyat.*
- *Ezután lehetőség van előadások/gyakorlatok hozzáadására.*
- *Egy új kurzusnál kiválasztja az oktatót és megadja a limitet.*
- *Majd véglegesíti a kurzusindítást.*

Meghirdetett tárgyak listázása:

A meghirdetett tárgyak menüpont alatt látható az aktuális félévben meghirdetett tárgyak listája. Egy tárgy kiválasztása esetén lehetőség van annak módosítására, melynek során új

kurzusok rendelhetők az adott tárgyhoz, valamint változtatható a létszám és az oktató személye. Erre azonban csak az aktív félévben meghirdetett tárgyak esetén van lehetőségünk. A korábbi félévekben meghirdetett tárgyak listázhatók, megtekinthetők, de nem módosíthatók.

Forgatókönyv:

- *Az adminisztrátor kiválasztja a MEGHIRDETETT TÁRGYAK menüpontot.*
- *Ha az aktuális félévben meghirdetett tárgyak közül valamelyiket módosítani akarja, ezt a MÓDOSÍTÁS gombra kattintva teheti meg.*
- *Ezután új kurzusokat adhat hozzá, a meglévőket módosíthatja, illetve törölheti.*
- *Korábbi félév is listázható, de nem módosítható.*

Diplomamunka témák kiírása:

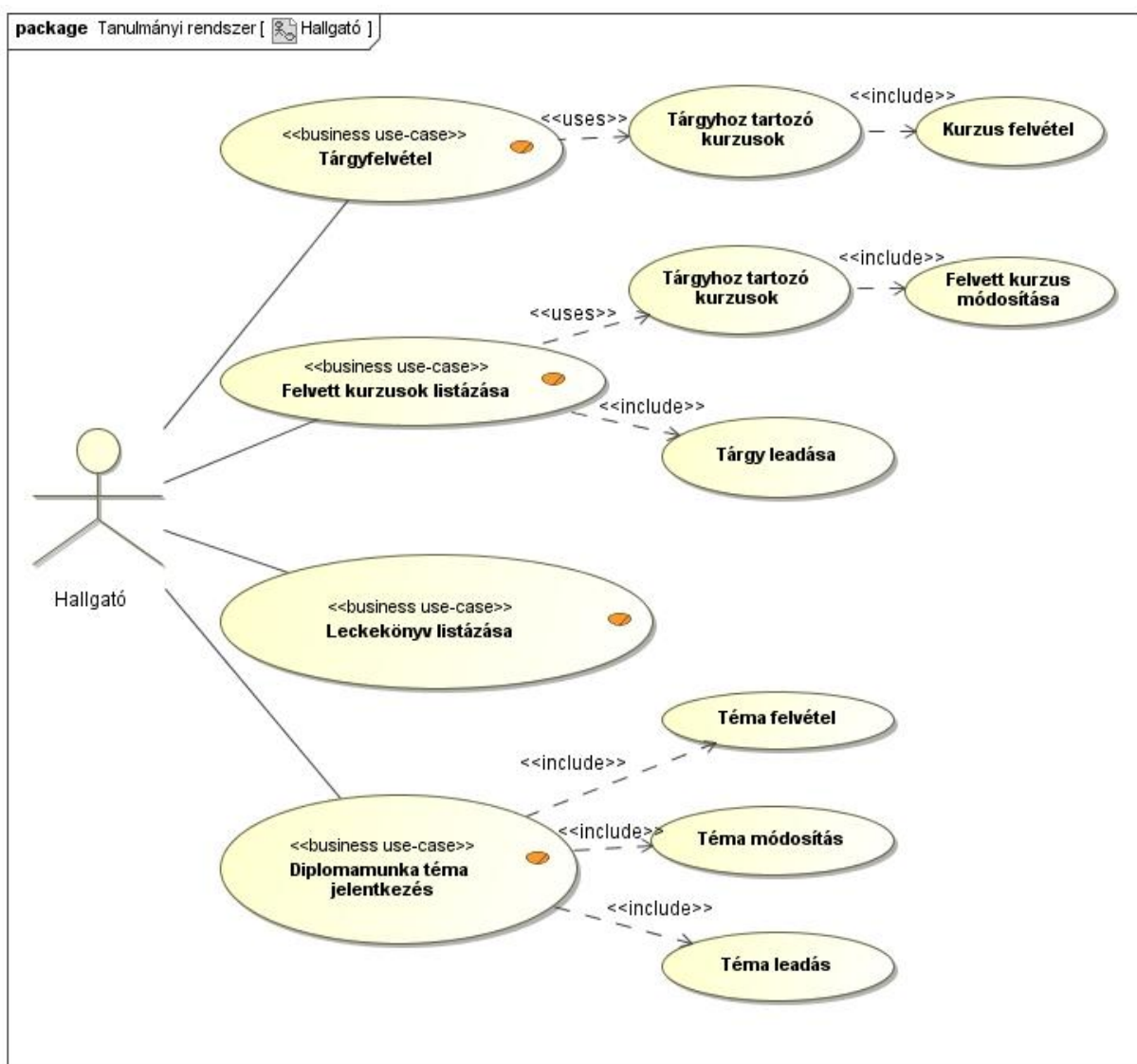
Az adminisztrátor a rendszerben diplomamunka témákat hirdethet meg, megadva a konzulens személyét, a téma címét, rövid leírását, és a maximális létszámot. Megnézheti a korábbi félévekben meghirdetett diplomamunka címeket, de új témát csak az aktuális félévben adhat hozzá.

Forgatókönyv:

- *Az adminisztrátor kiválasztja a DIPLOMAMUNKA menüpontot.*
- *Ezután megjelennek az eddig kiírt témák az aktuális félévben.*
- *Ha kiválasztja az ÚJ TÉMA HOZZÁADÁSA menüpontot, megadhatja az új diplomamunka címet, a téma rövid leírását, a konzulens személyét, és maximális létszámot.*
- *Ezután véglegesíti a témaindítást.*
- *Ebben a menüpontban is lehetőség van a korábbi félévekben meghirdetett diplomamunka címek megtekintésére, de új téma hozzáadása ebben az esetben már nem lehetséges.*

3.3.2 Hallgató

A hallgató ebben az esetben egy olyan felhasználó, aki az intézménnyel kapcsolatos adminisztrációs ügyeit intézheti a rendszeren keresztül. Ide tartozik: kurzusok felvétele, leadása, leckekönyv megtekintése, kiírt diplomamunka témák listázása, kiválasztása és felvétele.



5. ábra. A hallgatóhoz tartozó UML használati eset diagram

Tárgyfelvétel

A hallgató a meghirdetett tárgyak menüpont alatt listázhatja azokat a tárgyakat, melyek meghirdetésre kerültek az adott félévben. Láthatja az egyes tantárgyakhoz tartozó kreditpontok számát, hogy szerzett-e már korábban aláírást valamelyik tárgyból és hogy milyen tárgyakat teljesített már. A teljesített tárgyak esetében a FELVESZ gomb inaktívvá válik. Ha kiválaszt egy olyan tantárgyat, amelyikből még nincs meg az aláírása, akkor az adott tárgyból előadás és gyakorlat felvétele is kötelező. Ha egy olyan tárgyat választ ki, amelyikből már megvan az aláírása akkor csak az előadást veheti fel. Ha a tárgyfelvételek során téves vagy hiányos bejelölés történik, egy hibaüzenet jelenik meg a hallgató számára.

Forgatókönyv:

- *A hallgató kiválasztja a MEGHIRDETETT TÁRGYAK menüpontot.*
- *Ezután egy listából kiválaszthatja azt a tárgyat, amelyet szeretne felvenni.*
- *Azokat a tárgyakat, melyeket már korábban teljesített nem veheti fel még egyszer.*
- *Ezután a tárgyhoz tartozó kurzusokból választhat, attól függően, hogy megszerezte-e már az aláírást. Ha nincs aláírás, akkor egy előadás és egy gyakorlat megjelölése után lesz csak sikeres a jelentkezés. Ha olyan tárgyat vesz fel, amelyből aláírással rendelkezik, csak az előadást kell felvennie.*
- *A rendszer ellenőrzi az adatokat, véglegesíti a jelentkezést, hiba esetén értesíti a felhasználót (például ha betelt a létszám).*

Felvett kurzusok listázása

A hallgató a felvett kurzusok menüpont alatt megtekintheti a felvett kurzusokat. Ugyanitt lehetősége van a felvett tárgy leadására, illetve az egyes tárgyaknál módosíthatja a kurzusjelentkezést.

Forgatókönyv:

- *A hallgató kiválasztja a FELVETT TÁRGYAK menüpontot.*
- *Megtekinti a felvett tárgyakat, ezután kiválaszthatja azt a tárgyat, amelyet leadni/ módosítani szeretne. Ennek előfeltétele, hogy az adott tárgyból még ne rendelkezzen érdemjeggyel vagy aláírással.*
- *Módosítás esetén megjelennek a tárgyhoz tartozó kurzusok. Itt lehetőség van a kurzusváltásra.*
- *A rendszer ellenőrzi az adatokat, véglegesíti a módosítást, hiba esetén értesíti a felhasználót.*

Leckekönyv

A hallgatók itt nyomon követhetik tanulmányaik előrehaladását: az elért kreditpontok számát, az aláírások meglétét és a szerzett érdemjegyeket. Tanulmányaik megkezdése óta eltelt összes félév itt megtekinthető.

Forgatókönyv:

- *A hallgató kiválasztja a LECKEKÖNYV menüpontot.*
- *Először az aktuális félév eredményei jelennek meg, de lehetőség van korábbi félévek listázására is.*

Diplomamunka témára jelentkezés

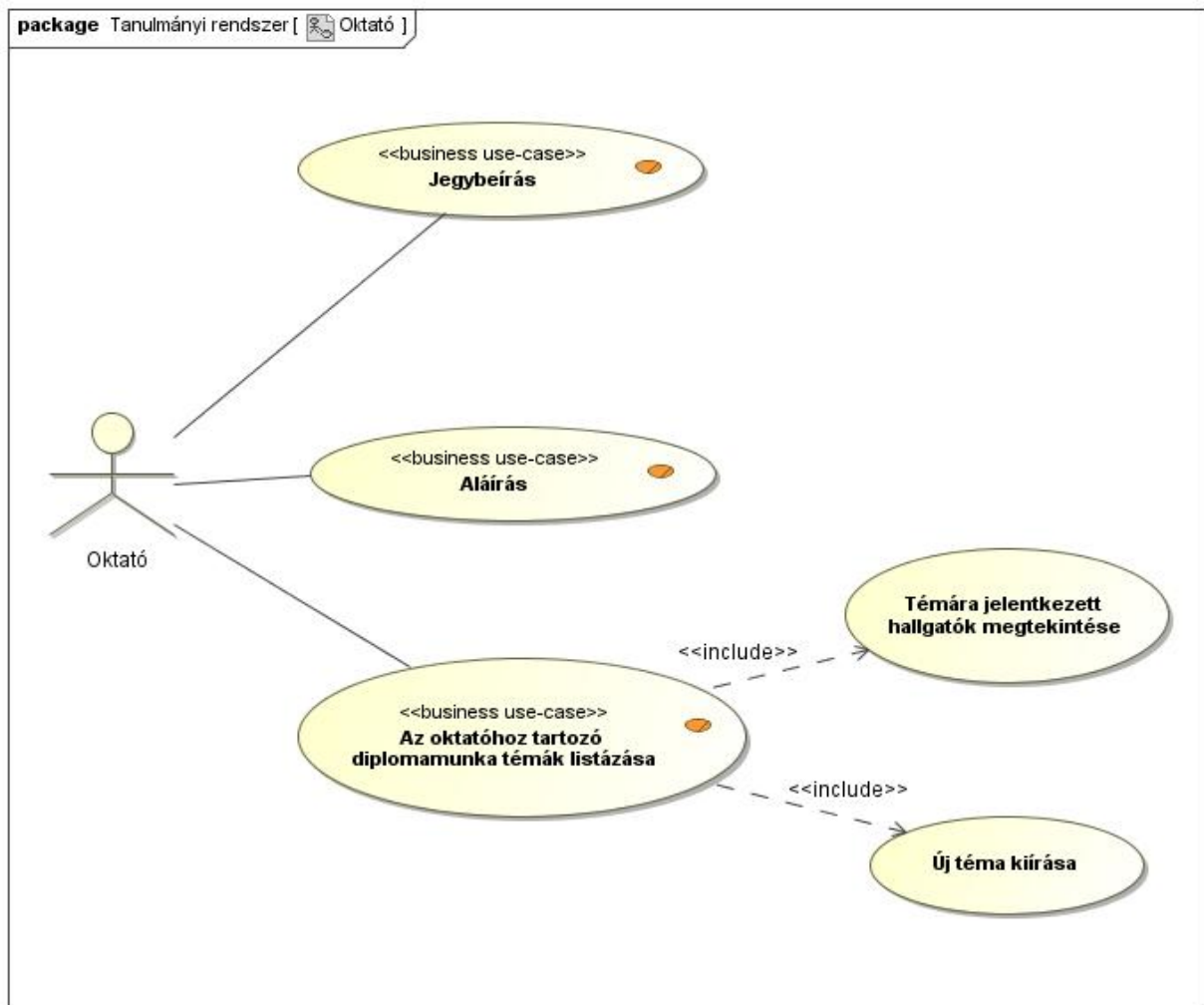
A hallgató a Diplomamunka menüpont alatt jelentkezhet a kiírt diplomamunka témákra. Ha még nem választott témát itt megjelölhet egyet, előtte tájékoztatást kaphat a téma leírásáról, a maximális létszámról és a konzulens személyéről. Ugyanezen menüpont alatt módosíthatja, illetve törölheti korábbi jelentkezését. A hallgató csak egy témát jelölhet meg.

Forgatókönyv:

- *A hallgató kiválasztja a DIPLOMAMUNKA menüpontot.*
- *Ha korábban még nem jelentkezett egyik témára sem, megjelölheti az általa megírni kívánt diplomamunka címét.*
- *Az ÚJ TÉMA FELVÉTELE gomb megnyomásával megjelennek az adott félévben meghirdetett témák, ahol az egyes témákról a BŐVEBBEN gomb megnyomásával további hasznos információkat olvashat.*
- *A megfelelő téma megjelölése után az OK gombbal véglegesítheti a jelentkezést.*
- *Ha már jelentkezett és módosítani szeretné jelentkezését, a MÓDOSÍT gomb segítségével megteheti.*
- *A hallgatónak lehetősége van a téma leadására is, a felvett téma mellett megjelenő LEADÁS gombra kattintva.*

3.3.3 Oktató

Az oktató egy olyan felhasználó, akinek lehetősége van az általa oktatott tárgyak listázására, a jelentkezett hallgatók megtekintésére, aláírás- vagy jegybeírásra, új diplomamunka témák meghirdetésére.



6. ábra. Az oktatóhoz tartozó UML használati eset diagram

Aláírás és jegy beírása

Az oktatók az általuk oktatott kurzusokat listázhatják, megtekinthetik az egyes kurzusaikra jelentkezett hallgatókat. Attól függően, hogy a kurzus előadás vagy gyakorlat, lehetőség van az aláírás és az érdemjegy beírására. Visszamenőleg a korábbi félévek is megtekinthetők, de már nem módosíthatók.

Forgatókönyv:

- *Az Oktató kiválasztja a KURZUSOK/JEGYBEÍRÁS menüpontot.*
- *Listázza, majd kiválasztja a megfelelő kurzust, és attól függően, hogy ez előadás vagy gyakorlat, lehetősége van aláírás illetve jegy beírására.*
- *A megfelelő kurzus kiválasztása után megjelenik egy lista azokról a hallgatókról, akik az adott kurzust felvették. Lehetőség van külön-külön az egyes hallgatók aláírásának vagy jegyeinek beírására is.*
- *Az OK gombbal véglegesíti a módosításokat.*

Diplomamunka témák listázása

Ebben az opcióban az oktatók listázhatják az aktuális félév összes diplomamunka témáját, megtekinthetik az általuk meghirdetett témákat, és új témákat is megadhatnak. Lehetőség van az előző félévek diplomamunka témáinak a megtekintésére is.

Forgatókönyv:

- *Az oktató kiválasztja a DIPLOMAMUNKA menüpontot.*
- *Ekkor megjelenik az aktuális félév összes diplomamunka témája.*
- *Egy menüpont segítségével listázhatja az általa meghirdetett diplomamunka témákat.*
- *Az ÚJ TÉMA HOZZÁADÁSA gomb segítségével új diplomamunka témákat írhat ki.*
- *A BŐVEBB gomb segítségével megtekintheti a már kiírt témákra jelentkezett hallgatók névsorát.*

3.4 Követelményspecifikáció eredménye

A követelményanalízis során tehát meghatároztuk a funkcionális és nem funkcionális követelményeket. Az előbbieket használati eset diagramokat használtunk, azonosítottuk az egyes aktorokat, meghatároztuk a szerepkörüket. Az utóbbiak esetében pedig meghatároztuk a rendszer futtatási környezetét. A követelmény feltárás végére részletesen megismertük az elkészítendő rendszerrel kapcsolatos funkciókat és elvárásokat.

4. Elemzés

4.1 Követelmények elemzése

Az elemzést és a tervezést nehéz élesen elválasztani egymástól. Az elemzés a „mit” kérdésre próbál választ adni, a tervezés pedig arra, hogy „hogyan” oldjuk meg a feladatot. Az elemzés során a feladat megértése a hangsúlyos, vagyis mi az, amit el szeretnénk készíteni, a tervezés során pedig arra koncentrálunk, hogy hogyan valósítsuk meg a feladatot.

Az elemzést azzal indítjuk, hogy létrehozuk az elkészítendő rendszer fogalmi modelljét, melynek során megismerjük a problémátér számunkra fontos fogalmait és a köztük lévő kapcsolatokat. Elemzés során meg kell vizsgálni a projekt kivitelezhetőségét, a követelmények valódiság tartalmát, illetve meg kell becsülni a szükséges erőforrásokat. Az elemzési fázisban a fontosabb folyamatok részletes áttekintése, leírása a feladat, ezzel a valóságnak egy áttekinthető korrekt modelljét hozzuk létre. Ez a modell azt ábrázolja, hogy mit csinál a rendszer függetlenül a konkrét megvalósítástól.

Az elemzés során három különböző modell segítségével vizsgálódunk:

- *Objektum modell*

A rendszer statikus tulajdonságait, struktúráját írja le, az adat áll a középpontban.

- *Dinamikus modell*

A vezérlésre helyezi a hangsúlyt, az időbeliség rögzítése a cél.

- *Funkcionális modell*

A rendszer által végrehajtandó funkciókra fókuszál.

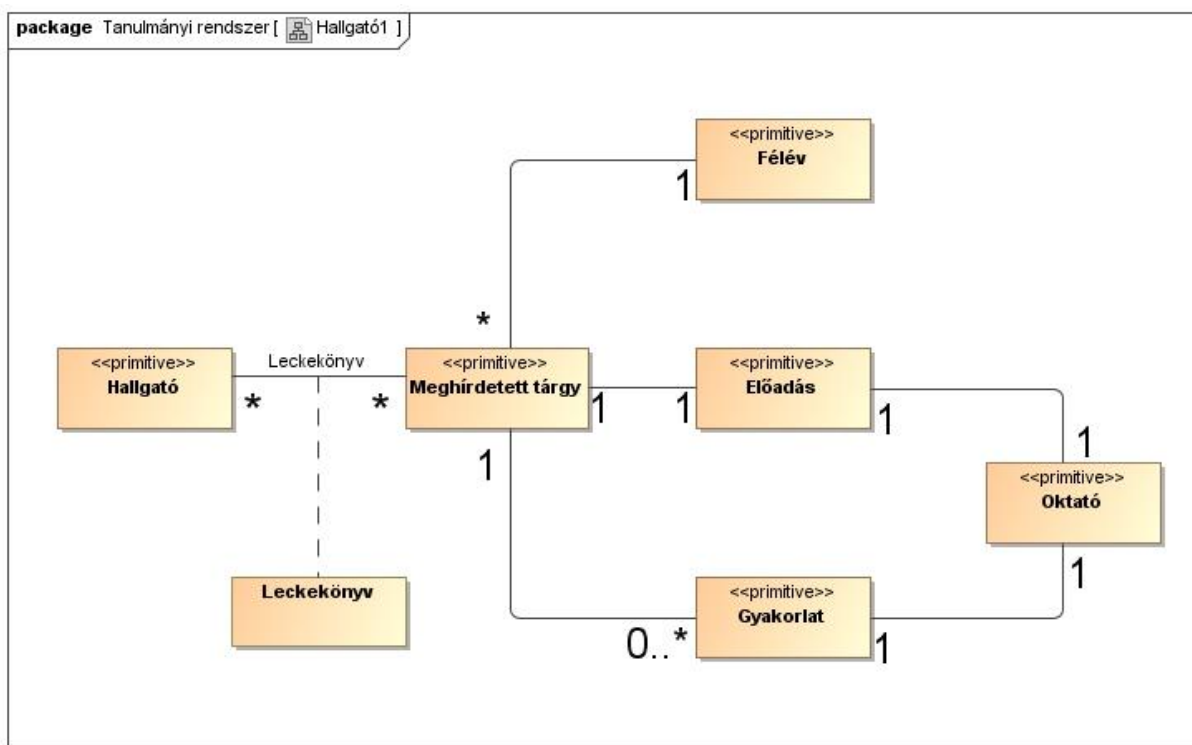
Ez a három modell szoros kapcsolatban áll egymással, hiszen ugyanazt a rendszer írják le három különböző aspektusból. Ezek együttese szolgál a későbbi fázisok alapjául.

Az elemzési folyamat utolsó lépései a validáció és a verifikáció. A validáció során azt bizonyítjuk, hogy a feltárt követelményrendszer a megrendelő igényeinek megfelelő rendszert specifikálja. A validálási folyamatnak a követelménydokumentációra, követelményspecifikációra kell vonatkoznia. A verifikáció azt jelenti, hogy ellenőriznünk kell a követelménydokumentációban, hogy a majdan kifejlesztendő rendszer esetében milyen módon lehet majd bizonyítani azt, hogy megfelel a támasztott követelményeknek.

4.2 Objektummodell

Az objektummodell segítségével megkeressük a rendszer objektumait, és az ezek között lévő kapcsolatokat. Az objektum modell egy statikus modell, melynek viszonylagos egyszerűsége abból adódik, hogy a statikus részek könnyebben értelmezhetők, kevésbé függnék az alkalmazás részleteitől.

A mi esetünkben nem tudjuk egy modellben szemléltetni a szakterületről szerzett ismereteinket, ezért először megvizsgáljuk azokat a szakterületi fogalmakat és a köztük lévő kapcsolatokat, melyek a hallgatókkal, és a hallgatói funkciókkal kapcsolatosak. A szakterület lényeges fogalmai lesznek az osztályok, melyek összefüggéseit osztálydiagramokkal ábrázoljuk. Az alábbi diagramon (7. ábra) a hallgatók, a meghirdetett tárgyak, a leckekönyv és az aktuális félév közötti kapcsolatot szemléltetjük.

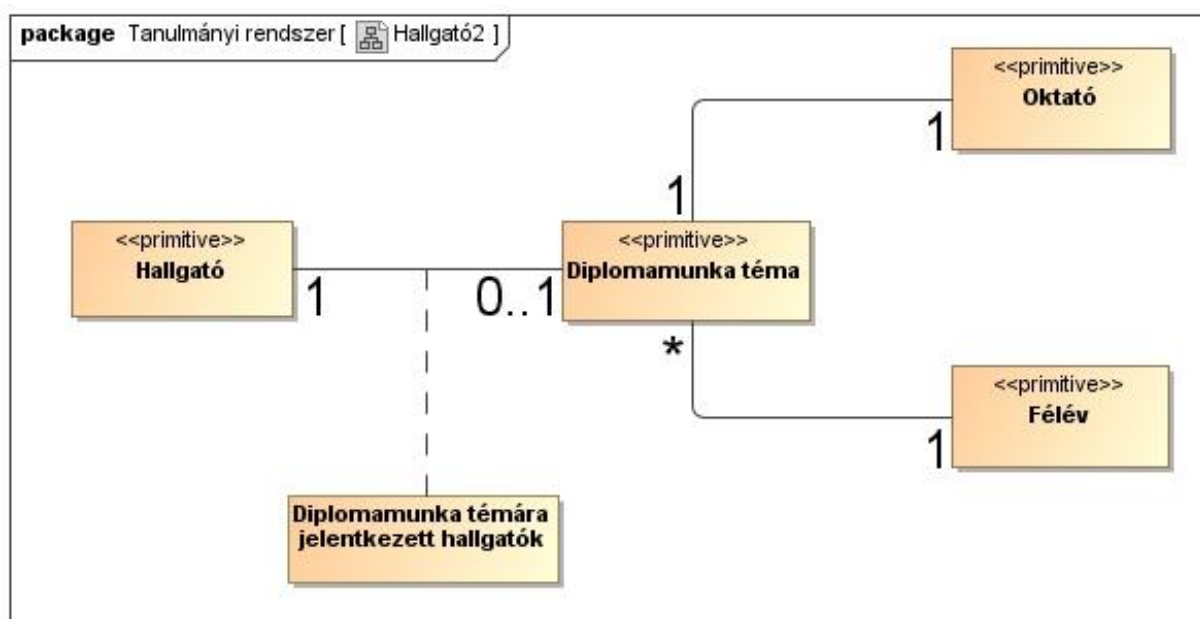


7. ábra. A hallgatóhoz tartozó UML osztálydiagram

Az osztályok közötti viszonyt az asszociációk írják le, amit egy vonallal jelölünk. Az egyes vonalak végeinél számokat, illetve csillagokat találunk, ez utal a kapcsolat számosságára. A félév és a meghirdetett tárgy közötti [1, *] kapcsolat azt jelöli, hogy egy félévben több

tárgyat is meghirdethetünk. A meghirdetett tárgyak és az előadás között $[1, 1]$ kapcsolat van, mivel minden tárgyhoz pontosan egy előadás tartozik. Azonban a gyakorlat esetében $[1, 0..*]$ találunk, amely azt mutatja, hogy egy tárgyhoz több gyakorlat is tartozhat, és vannak olyan tárgyak, amelyekhez nem tartozik gyakorlat. A leckeönyv egy asszociációs osztály, mivel ez nem csupán egy osztály, hanem egy asszociáció is a hallgató és a meghirdetett tárgyak között. A leckeönyv osztály, a hallgató és a meghirdetett tárgy közötti vonalhoz szaggatott vonallal csatlakozik, ami azt jelenti, hogy az egyes kapcsolatok saját identitással rendelkeznek.

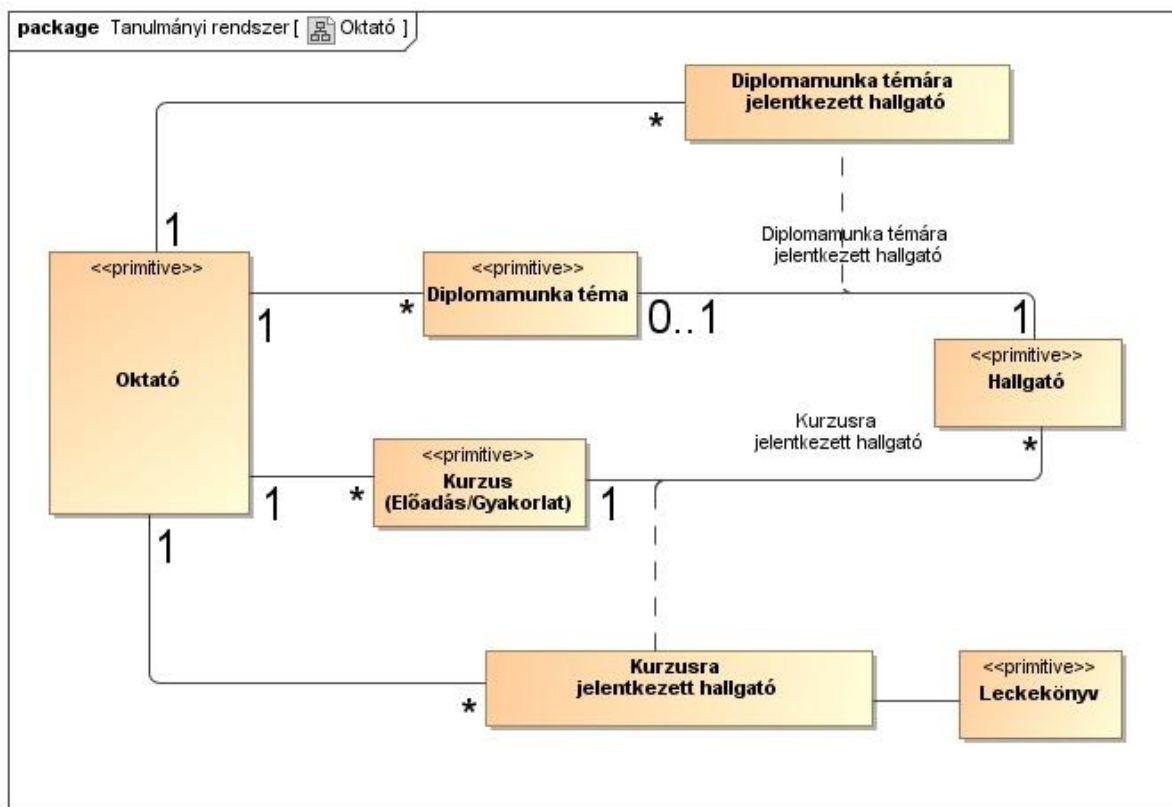
A következő osztálydiagram (8. ábra) még mindig a hallgatókkal kapcsolatos: itt a hallgatók, a diplomamunka témára jelentkezett hallgatók, a diplomamunka téma, és a félév közötti kapcsolatokról kaphatunk átfogó képet.



8. ábra. A hallgatóhoz tartozó UML osztálydiagram

A 8. ábrán látható osztálydiagramon is találkozhatunk egy asszociációs osztállyal, melyben a hallgató és a diplomamunka téma közötti kapcsolatokat tároljuk. Az $[1, 0..1]$ kapcsolat azt jelöli, hogy egy hallgató legfeljebb egy témára jelentkezhet, de vannak olyan hallgatók is, akik még nem jelöltek meg egy témát sem.

Az eddigiekben a hallgatókkal kapcsolatban néztük meg az egyes fogalmakat, és a köztük lévő kapcsolatokat. Vizsgáljuk meg ugyanezt az oktatók szemszögéből (9. ábra).



9. ábra. Az oktatóhoz tartozó UML osztálydiagram

Az oktató a saját kurzusaihoz tartozó hallgatókat listázva tudja beírni az aláírásokat és az érdemjegyeket a leckekönyvbe. Az oktatók a diplomamunka témára jelentkezett hallgatók asszociatív osztály segítségével azokat a hallgatókat láthatják, akik már jelentkeztek valamelyik meghirdetett diplomamunka témára.

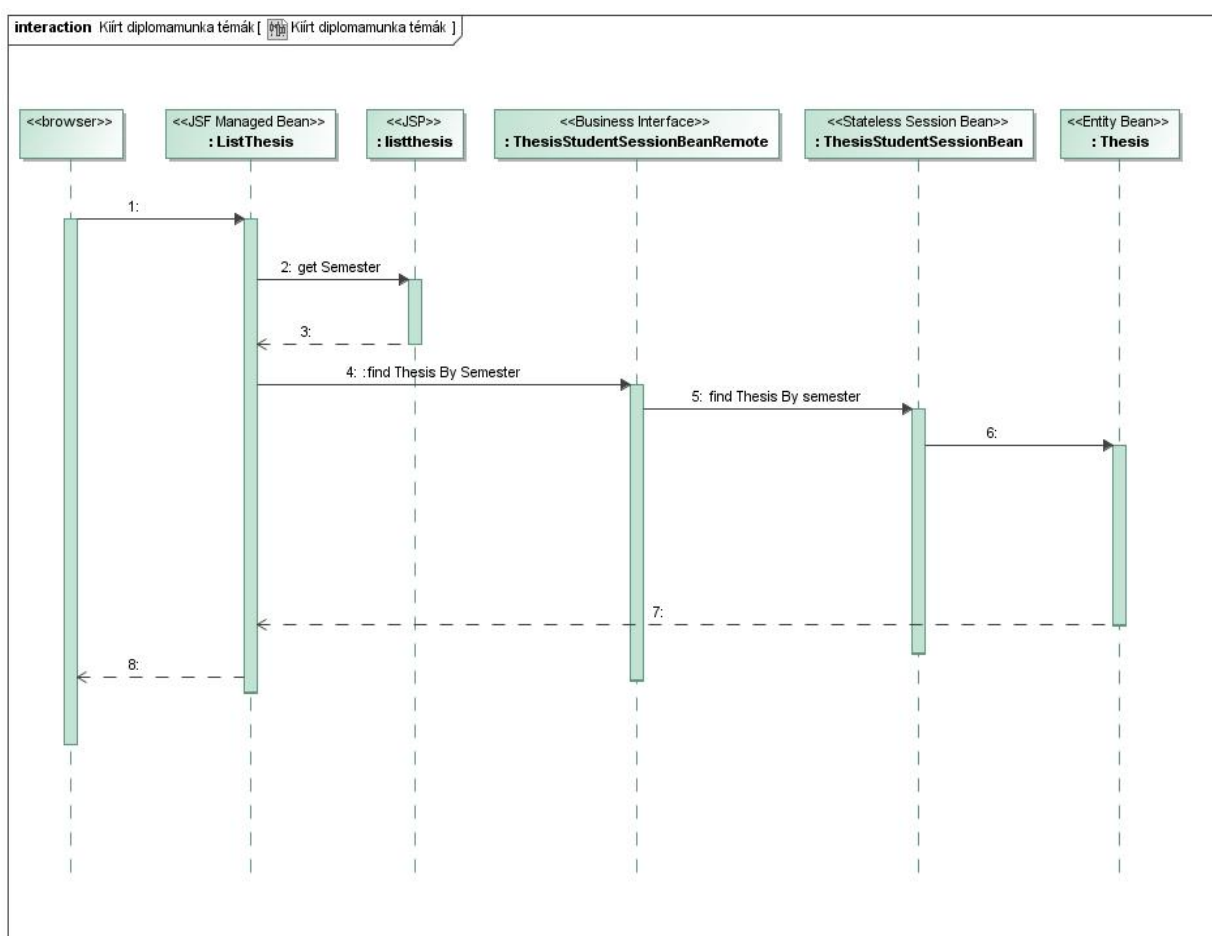
Az alábbiakban átfogó képet kaptunk a rendszer elkészítéséhez szükséges osztályokról és a közöttük lévő kapcsolatokról.

4.2 Dinamikus modell

A dinamikus modell segítségével a rendszer objektumainak időbeli viselkedését szemléltetjük. Először meg kell keresnünk az egyes eseményeket, melyek a felhasználó és a

rendszer közötti interakciókból származnak, ezeket egy eseménylistában tároljuk, majd ezeket belső eseményre kell visszavezetnünk. Ezután minden eseményhez meg kell keresnünk azokat az objektumokat, melyek közvetve vagy közvetlenül, részt vesznek az esemény kezelésében és a válasz létrehozásában. Az egyes objektumok, osztályok közötti cselekmények időbeli leírására szekvencia diagramokat használunk, ezek felépítése során csak az események sorrendjére koncentrálnunk.

Az eseménylistában található minden egyes eseményhez új szekvencia diagramra lenne szükség, az alábbiakban egy példán keresztül láthatjuk a szekvencia diagram felépítését.



10. ábra. A Diplomamunka témák listázásához tartozó szekvencia diagram

Itt a hallgató által kiváltott esemény a diplomamunka témák listázása. A **10. ábrán** jól látszik, hogy egy külső eseményhez több osztály összehangolt működése szükséges, illetve hogy az egyes belső események milyen sorrendben követik egymást.

4.3 Funkcionális modell

A funkcionális modell segítségével azt vizsgáljuk, hogy a rendszer az eredményeit milyen módon állítja elő. Ez a modell egy olyan rendszernek tekinthető, amely a külvilágból érkező, a rendszer használói által kiváltott eseményekre válaszol. A felhasználók azt várják el a rendszertől, hogy kiszámítható legyen, és a specifikációnak megfelelő eredményeket szolgáltatson. A modell megadja, hogy a bemeneteket milyen folyamatoknak kell átvenniük, és ezek milyen részeredményeket szolgáltatnak. A dinamikus modellel ellentétben itt a felhasználók és a rendszer közötti adattartalom áll a középpontban, és nem pedig az, hogy a folyamatok milyen vezérlés szerint követik egymást.

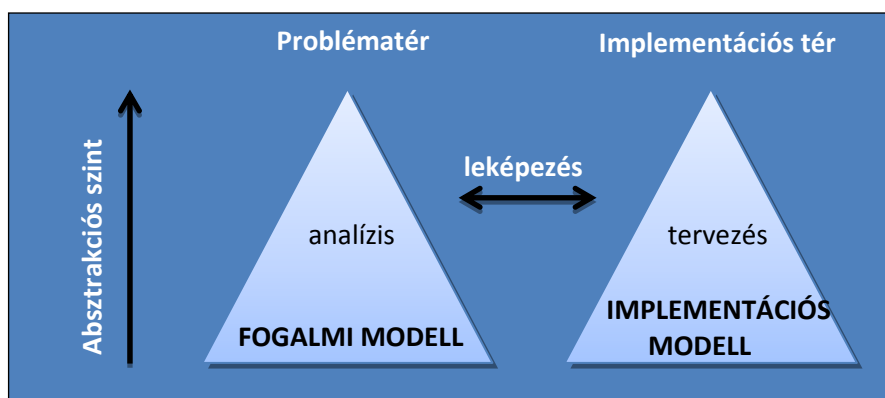
4.4 Az elemzés eredménye

Az elemzés során elkészítettük a rendszer fogalmi modelljét. Célunk a problématerület számunkra fontos elemeinek, és a köztük lévő összefüggéseknek a meghatározása és modellezése volt. Definiáltuk azokat az objektumokat, melyek a használati esetekben meghatározott viselkedést végrehajtnak. A rendszer fogalmi modellje, amely három eltérő nézőpont alapján felépült modellek együttese (objektum modell, dinamikus modell, funkcionális modell) elegendő információval szolgál ahhoz, hogy tovább lépjünk a következő fázisra.

5. Tervezés

5.1 A tervezés feladata

A tervezés során a feladatunk, hogy azokat a követelményeket, melyeket az analízis során kialakítottunk, továbbfejlesszük. Itt már nem az a fontos, hogy a rendszer milyen funkciókat lát el, hanem az, hogy ezt hogyan teszi. A tervezés során az implementációs térben dolgozunk. Célunk, hogy elkészítsük a rendszer implementációs modelljét, amelyhez figyelembe kell vennünk a technikai szempontokat is.



11. ábra. Problématér és implementációs tér közötti kapcsolat

Megkeressük azokat az eszközöket, technológiákat, amelyek leginkább alkalmasak a megoldás megvalósításához. A tervezés során egy implementációs modellt készítünk, amelyet az elemzés során elkészített fogalmi modellből képezünk le.

A leképezésnek két alaptípusát különböztetjük meg:

- *Top-down*

Ebben az esetben először a fogalmi modellt hozzuk létre teljes egészében, mégpedig úgy, hogy az absztrakciós szintet egyre csökkentjük, döntéseket hozunk annak érdekében, hogy addig nem részletezett tulajdonságokat, fogalmakat hogyan valósítsunk meg egyszerűbb komponensekből. Döntések sorozatán keresztül jutunk el a konkrétumok szintjére. Az implementációs modell kialakításánál is ugyanezt a stratégiát követjük.

- *Bottom-up*

Ez hasonlít az előző módszerhez, hiszen itt is először elkészítjük az analízis modellt az absztrakciós szint csökkentésével. Az eltérés a tervezés során jelentkezik, ahol az implementációs modell konkrétumaiból összerakjuk a fogalmi modell konkrétumait, majd az implementáció kezelhetősége érdekében egyre magasabb szintű implementációs fogalmakat alkotunk.

Egyik stratégia sem garantálja a pontos megfeleltetést a két modell között, és így a fogalmi modellben történő legkisebb módosítás az implementációs modell teljes átalakítását teheti szükségessé. Ezért a gyakorlatban az a módszer terjedt el, amikor egy kicsit finomítunk a fogalmi modellben és ezt a módosítást végrehajtjuk az implementációs modellben is, így a megfeleltetés bármely absztrakciós szinten pontos lesz. Ennek nagy előnye, hogy a problémátér fogalmai könnyen felismerhetőek lesznek az implementációs modellben is.

A tervezés folyamán egy olyan formális leírást állítunk elő, mely az általunk használt programozási nyelven a rendszert szimuláló programot definiálja. A tervezés az implementáció előkészítése, melyet három részre osztunk:

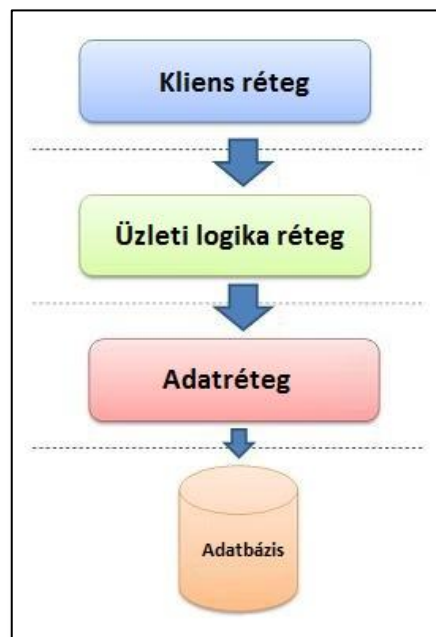
- *Architektúrális tervezés*
- *Felülettervezés*
- *Objektumtervezés*

5.2 Architektúrális tervezés:

Az architektúrális tervezés elviekben független az implementációtól, azonban ahhoz, hogy jó tervet készítsünk, figyelembe kell vennünk bizonyos implementációs dolgokat. A fejlesztő feladata, hogy megtalálja a megfelelő átmenetet, hiszen egy rossz architektúrális terv az egész projekt kivitelezését veszélyeztetheti. A pontos tervezés több felhasználós esetekben különösen fontos, hiszen itt a rendszer hibája egyszerre több ember munkáját is akadályozza.

5.2.1 A többretegű technológia

A tervezett rendszer egy webes alkalmazás, amely lényeges szempont, mert ennek megfelelően kell felépíteni az architektúrát. A webes felületen elérhető alkalmazások általában több egymástól jól elkülöníthető rétegből épülnek fel (**12. ábra**).



12. ábra Rétegelt architektúra

Az ábrán látható, hogy 3 réteget különböztetünk meg: megjelenítés, üzleti logika és adatbázis kezelés. Vizsgáljuk meg az egyes rétegek feladatát:

- *Kliensréteg*

A kliensréteg feladata a megjelenítés megvalósítása, a felhasználók ezen a felületen keresztül tudnak szolgáltatásokat elérni, műveleteket végrehajtani. Kétféle típust különböztetünk meg:

- a *vastagklienst*, ezek a desktop alkalmazások
- a *vékonyklienst*, amely böngészőkön keresztül éri el az alkalmazást

A gazdag webes klienseknek köszönhetően a vékony kliensek is a desktop alkalmazásokhoz hasonló hatást nyújtanak a felhasználók számára. Ha az alkalmazást vékony kliensen keresztül szeretnénk elérni, akkor egy úgynevezett *webrétegre* is szükségünk lesz, mely a böngészőből érkező HTTP kéréseket értelmezi, majd megfelelő választ generál.

- *Üzleti logikai réteg*

Ennek a rétegnek az a feladata, hogy megvalósítsa azokat a funkcionalitásokat és műveleteket, amit a rendszertől elvárunk. Ezt a Java EE esetében alkalmazásszerverre telepítjük annak érdekében, hogy elérjük a middleware szolgáltatásokat. Ezeket majd az implementáció során részletesen tárgyaljuk.

- *Adatréteg*

Feladata az adatok perzisztens tárolása, az adatokon végezhető elemi műveletek támogatása, létrehozása, lekérdezése, törlése és módosítása. Leginkább relációs adatbázisokat használunk, de megjelennek az objektumorientált adatbázisok is.

Az elosztott architektúra számos előnyös tulajdonsággal rendelkezik, a következőkben a legfontosabbakat láthatjuk:

- *Skálázhatóság*

Ez az egyik legfontosabb tulajdonság, a terhelés megnövekedése esetén meg tudjuk oldani a kliensek több szerverhez történő csatlakozását. Minden szerver képes ellátni ugyanazt a feladatot, ezért a kliens szempontjából mindegy, hogy melyik szerverhez csatlakozik. Egy szerver meghibásodása még nem okozza a teljes rendszer összeomlását.

- *Alternatív megjelenés*

A 3 rétegű alkalmazásoknál az üzleti logika egy szerveren található, ezért a rendszerhez vékony illetve vastag klienssel is csatlakozhatunk a felhasználó igénye szerint anélkül, hogy az üzleti logikába belenyúlnánk.

- *Biztonság*

A bizalmas információk a szerver oldalon maradnak, ezért ha a biztonsági stratégia jó, a kliensek csak egy jól kontrolált környezetben keresztül férhetnek hozzájuk. Így csökkenthető a visszaélések, támadások száma.

- *Adatbázis cseréje*

A 3 rétegű alkalmazásoknál az adatbázis bármikor kicserélhető anélkül, hogy az üzleti logikához nyúlnánk.

Láthatjuk, hogy számos előnnyel jár a többretegű alkalmazásfejlesztés, ennek segítségével egy korszerű és dinamikus rendszert tudunk készíteni.

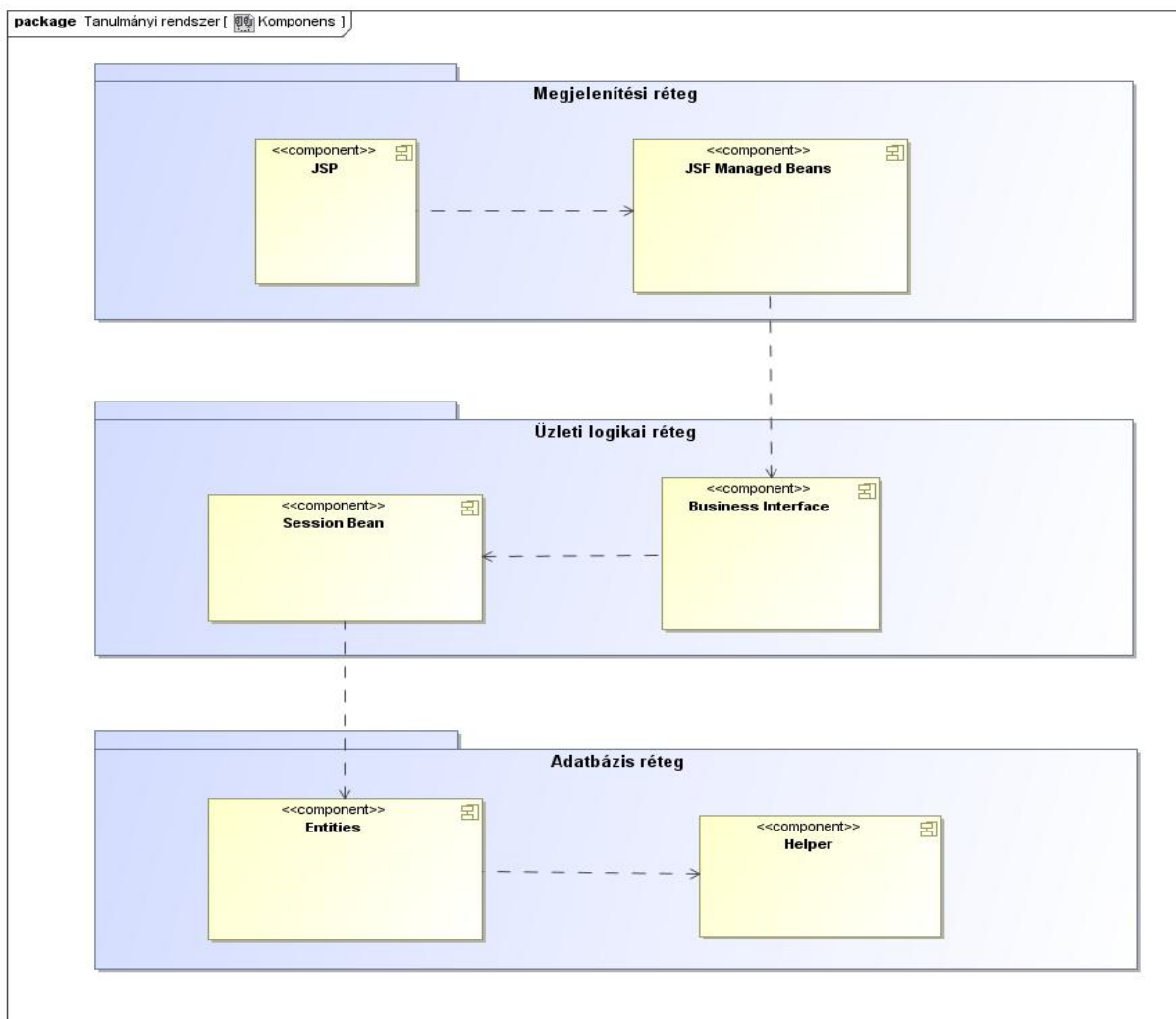
5.2.2 A rendszer architektúrájának megtervezése

A rendszer architektúra megtervezése során első dolgunk, hogy meghatározzuk azokat az alrendszereket és modulokat, melyekből a rendszert felépítjük. Ezután szétosztjuk az egyes rétegek között a különböző alrendszerek által nyújtott szolgáltatásokat. Fontos szempont az

interfészek pontos specifikációja, hiszen az egyes rétegek ezen keresztül érik el egymás szolgáltatásait. A megvalósításnál arra kell figyelni, hogy minden réteg csak az alatta lévő réteg szolgáltatását veheti igénybe, ami viszont lassíthatja a rendszert, ha sokszor hívunk meg alsóbb szintű szolgáltatásokat.

A tervezés során fontos, hogy a megfelelő struktúrát alakítsuk ki, ezzel tökéletesítve az alkalmazás átláthatóságát, használhatóságát. Amikor a rendszert kisebb részekre - úgynevezett alrendszerekre bontjuk - azokat az osztályokat kapcsoljuk össze, melyek együttesen egy nagyobb funkció elvégzéséért felelősek. Az egyes részek egymással párhuzamosan is fejleszthetők, mivel az egyes alrendszereket csak egymással logikailag összetartozó részek alkotják. Ezek a részek az újrafelhasználás határait is kijelölik.

A rendszer struktúráját a **13. ábrán** látható komponensdiagram szemlélteti.



13. ábra. A rendszer komponens diagramja

A komponensek tetszőleges egységeket jelentenek, amelyek önállóak, más komponensektől függetlenül működtethetők, cserélhetők. Ahogy az ábrán látható, a rendszer architektúráját az előzőekben ismertetett többretegű technológiát figyelembe véve terveztük meg. A megjelenítési rétegben a JSP komponensek és az őket vezérlő JSF osztályok kaptak helyet. A JSF osztályok a rendszer szolgáltatásait az interfészekon keresztül hívhatják meg, melyek implementációi a session-ben találhatók. Az adatok perzisztens tárolásában az entitás osztályok segítenek. A megvalósítással kapcsolatos részleteket az implementáció során tárgyaljuk.

5.3 Felülettervezés

A tervezés egyik fontos lépése a felhasználói felületek (*user interface*) kialakítása. Ezzel általában külön szakemberek, úgynevezett webdesignerek foglalkoznak, hiszen ez a program szempontjából döntő fontosságú. A felhasználók ezen a felületen keresztül lépnek kapcsolatba a rendszerrel, ezen keresztül mérik fel a rendszer használhatóságát. A legjobb alkalmazás is használhatatlanná válik, ha a hozzá tartozó felület nehezen értelmezhető, bonyolult, esetleg olyan színeket tartalmaz, melyek fárasztják a szemet. Ha az információk megjelenítése zavaros, az könnyen megtévesztheti a felhasználót, ezért lényeges, hogy rendszerüzeneteket küldjünk a felhasználó felé ezzel jelezve, ha valami probléma merült fel, vagy ha valamit sikeresen elvégzett. A webes kliensek már lehetővé teszik, hogy szinte bármilyen grafikus felületet létrehozzunk, mert a felhasználók manapság már ezt várják el egy alkalmazástól.

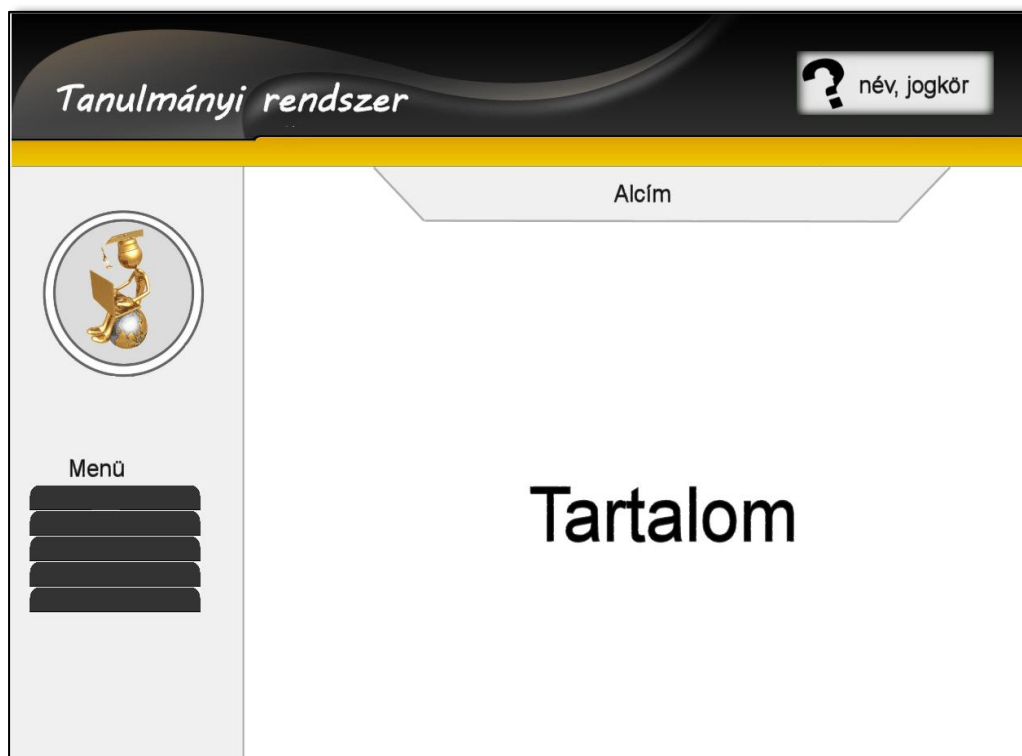
Amikor felületet tervezünk fontos néhány elvet szemünk előtt tartani:

- Mivel a felhasználók általában korlátozott rövidtávú memóriával rendelkeznek, ezért könnyen összezavarhatja őket az egyszerre megjelenő nagy mennyiségű információ, és ezért könnyebben hibáznak.
- A felhasználók már korábban is használtak rendszereket, ezért a felületeknek olyan kifejezéseket kell használnia, amely megfelel a felhasználók tapasztalatainak. A felhasználók által megszokott terminológiát ajánlott használni.
- A felhasználók megszoktak egyfajta szabványműködést, tehát a felületet úgy érdemes megtervezni, hogy az a felhasználónak ne okozzon meglepetést.

- A felületeknek hasonló műveleteket hasonló módon kell megvalósítani, ez az úgynevezett alacsony szintű konzisztencia. Például ha a menüben az utolsó menüpont a kilépés, akkor ez mindenhol ugyanott kell, hogy legyen. A konzisztens felületek lerövidíthetik a felhasználók betanulási idejét.
- A felhasználók gyakran hibáznak, ezért a felületnek támogatnia kell az utolsó lépés, lépések visszavonását. Érdemes megerősítést kérni a felhasználótól, ezzel is csökkentve a hibázás lehetőségét.
- A felhasználói felületeknek megfelelő interakciós lehetőségekkel kell rendelkezniük a különféle felhasználók számára. Például vannak olyanok, akik a parancsok segítségével szeretik használni a rendszert, míg mások a „drag and drop” módszert kedvelik.

Ezeket az alapelveket szem előtt tartva készítettem el a felhasználói felületet, melyet ebben az esetben egy böngészőből érhetünk el. Ahhoz, hogy bejuthassunk a tanulmányi rendszerbe, azonosítanunk kell magunkat a felhasználónév és jelszó megadásával.

Amennyiben sikeresen bejelentkeztünk a **14. ábrán** látható ablak jelenik meg a képernyőn.



14. ábra. A rendszer felülete

A tanulmányi rendszer felülete egy statikus és egy dinamikus részből áll:

- *Statikus rész*

Az oldal felső részén egy fejrész található, melynek a jobb oldalán lévő kis négyzetben jelenik meg a felhasználó neve és jogköre. Az oldal bal szélén egy oldalsáv kapott helyett, itt látható a rendszer logója, alatta pedig a menüsor, melynek segítségével navigálhatunk az egyes menüpontok között. Ezek az elemek bejelentkezés után változatlanok maradnak.

- *Dinamikus rész*

Itt jutunk hozzá a lényeges információkhoz oly módon, hogy mindig a felhasználó által választott menüpontnak megfelelő kiírás, illetve tartalom jelenik meg. Ha szeretnénk további információt kapni az egyes tartalmakról, azt a függelékben érhetjük el.

5.4 Objektumtervezés

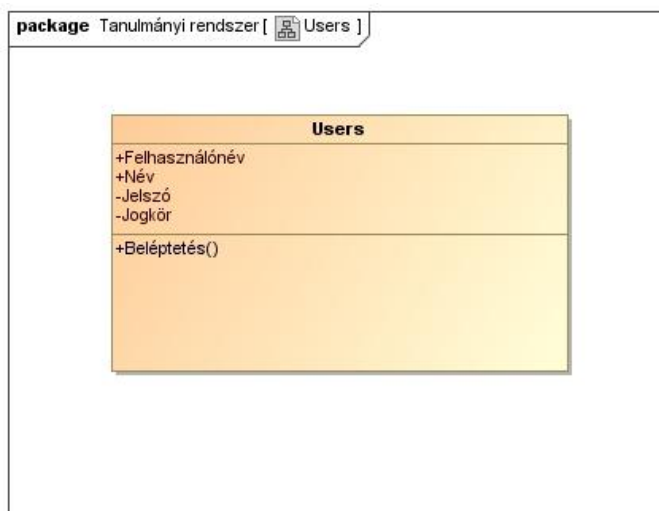
Az objektumtervezés folyamán az osztályok, objektumok belső szerkezetét és külső viselkedését kell meghatároznunk. Az implementációs objektumokkal kiegészített fogalmi modell az objektumtervezés alapja, melynek során össze kell kapcsolnunk a különböző szempontok szerint készített objektum modelleket, dinamikus modelleket és funkcionális modelleket. Az objektum modellből indulunk ki és a többi modell által képviselt információval ruházzuk fel az osztályokat, tehát a strukturáló elem az objektum modell.

Először létrehozuk az osztályok deklarációját, az attribútumok és a metódusok deklarációjával együtt. A legnagyobb problémát a funkcionális modell beépítése jelenti, mivel az objektum és a funkcionális modell között nincs szoros kapcsolat.

Az attribútumok az objektumok egyedi tulajdonságait fejezik ki, ezek meghatározása során definiáljuk az általa felvehető értékek halmazát is. Elsőnek az alkalmazás szempontjából jelentőséggel bíró attribútumokat vegyük fel, majd finomítsuk a modellt. Az attribútumok láthatóságát és elérhetőségét a Javához hasonlóan különböző értékek megadásával korlátozhatjuk.

Az objektum viselkedését a metódusok specifikációja határozza meg, amely egy beérkező üzenetre valamilyen cselekvéssel reagál.

Ennek megfelelően, az elemzés során megjelenő Users osztályt - amit az objektumtervezés során attribútumokkal és metódusokkal egészítettünk ki - a következőképpen ábrázolhatjuk osztálydiagram segítségével (**15. ábra**).



15. ábra. A Users osztály osztálydiagramja

Az egyes osztályokat nem csak diagram, hanem leíró táblázat segítségével is specifikálhatjuk, amely a következőképpen néz ki (**2. táblázat**).

Név:	Users
Informális leírás:	A rendszer használóinak közös reprezentációja, ez alapján tartjuk nyilván azokat, akik a rendszert használják.
Attribútumok: Felhasználónév Név Jelszó Jogkör	String típusú változó, amely belépéshez szükséges nevet tartalmazza. String típusú változó, amely a felhasználó teljes nevét tartalmazza. String típusú változó, amely belépéshez szükséges jelszót tartalmazza. Enum típusú változó, belépés után a rendszer ez alapján dönti el, hogy a felhasználó milyen funkciókkal fog rendelkezni.
Metódusok: Beléptetés	A rendszer ennek segítségével dönti el, hogy a felhasználó jogosult-e a belépésre, vagy sem. A visszatérési típusa boolean.

2. táblázat. A Users osztály leíró táblázata

Az objektumtervezés során elkészül a rendszert alkotó objektumok specifikációja.

5.5 Tervezés eredménye:

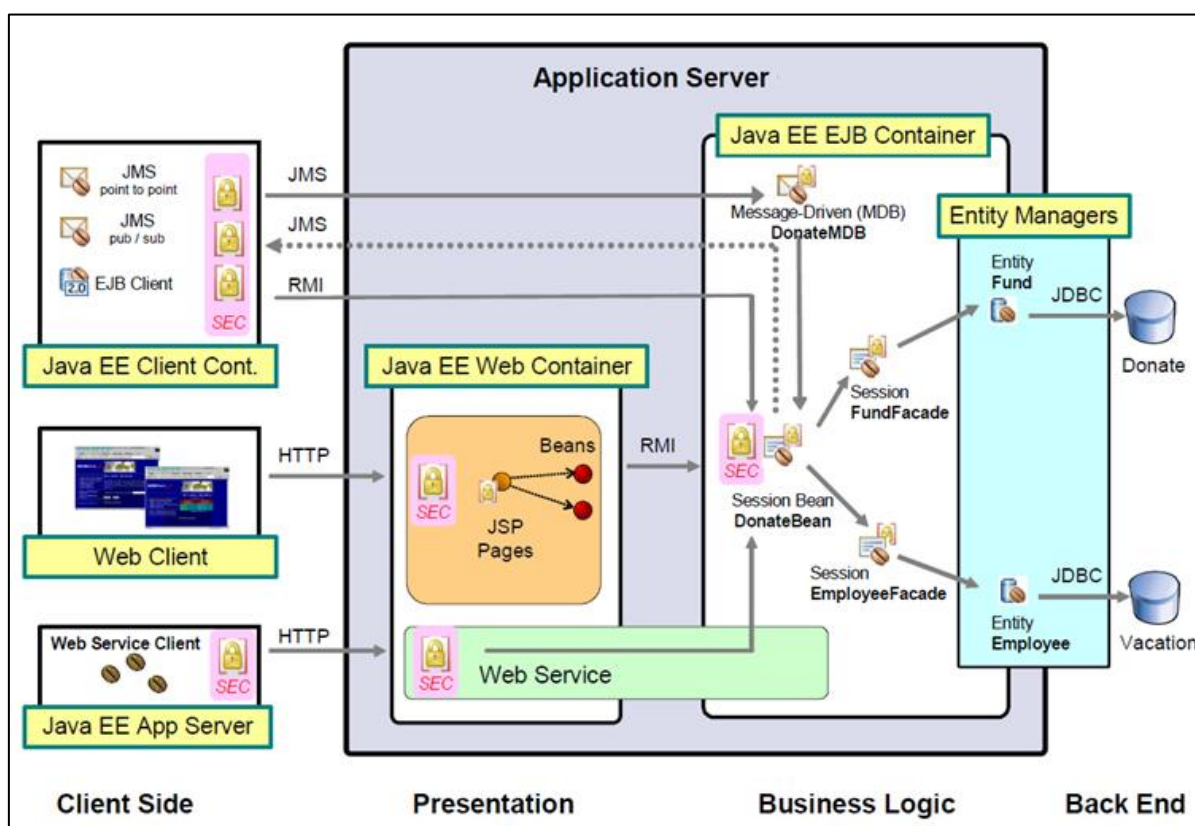
A tervezés során célunk egy olyan programterv megvalósítása, amelyben a lényeges problémákat már megoldottuk, és amely alapján a forráskód már könnyedén elkészíthető. Fontos, hogy a terv minél tökéletesebb legyen, mert a hibák javítása a későbbiek során a rendszer bonyolultsága miatt egyre nehezebb. A tervezés során meghatároztuk a szoftver architektúrát, a felhasználói felületet és az osztálymodelleket.

6. Megvalósítás

Miután a programterv elkészült, következhet a program megvalósítása. Ekkora már a lényeges döntéseket meghoztuk, így a programterv implementálásánál felmerülő technikai részletekre tudunk koncentrálni. Az implementáció során tehát a programtervet kódoljuk le egy adott programnyelven létrehozva a forrásprogramot. Egy forrásprogram akkor jó, ha pontos a feladat-specifikáció, a programterv alapján készült, áttekinthető, nem tartalmaz sem szintaktikai, sem szemantikai hibákat.

6.1 Java Enterprise Edition

A megvalósítás során a Java EE-t használtam, amely elosztott, sok felhasználóval rendelkező, vállalati méretű szoftverrendszerek fejlesztéséhez nyújt támogatást. A Java EE technológia napjaink egyik legnépszerűbb, legelterjedtebb szerveroldali megoldásává vált. A tervezés során már bemutattuk a többretegű rendszerek felépítését, előnyeit, most lássuk, hogy ezt a Java EE hogyan valósítja meg (16. ábra).



16. ábra. Java EE architektúra

A Java EE támogatja a megjelenítés és az üzleti folyamatok szétválasztását, ezért a megjelenítés függetlenül módosítható, alakítható. Ennek megvalósítása során JSF technológiát használtam. Ennek futtatásához saját futási környezetre, úgynevezett webkonténerre van szükség. Az EJB-t az üzleti logikai réteg és a perzisztencia réteg implementálása során használtam, ezek futtatásához EJB konténerek szükségesek. A fejlesztés során a Glassfish v3 alkalmazásszervert használtam, amely már támogatja a Java EE 6-os technológiákat is, és amely együtt tartalmazza a web és EJB konténereket.

A fejlesztés során számos új technológiát, és keretrendszert ismertem meg, melyek segítettek az alkalmazás létrehozásában. A következőkben ezeket a technikákat szeretném ismertetni.

6.2 Enterprise JavaBeans

Az Enterprise JavaBean-ek, röviden EJB-k olyan serveroldali komponensek, melyek a Java EE alkalmazásokban központi szerepet töltenek be, hiszen az üzleti logikát ezek segítségével valósíthatjuk meg. Ezenkívül EJB-t használjuk az adatréteg megvalósítására is. A legtöbb üzleti alkalmazás közös server oldali szolgáltatásokkal rendelkezik, ilyen például a perzisztencia kezelés, tranzakció kezelés, biztonság megvalósítása, távoli elérés, többszálúság, névszolgáltatás. Ezeket a típusú szolgáltatásokat *middleware szolgáltatásoknak* nevezzük. Az EJB-k és a konténer - ami az EJB-k futási környezete - együtt egy olyan keretrendszert alkotnak, amely biztosítja ezeket a middleware szolgáltatásokat. Az EJB technológia előnye, hogy implicit middleware szolgáltatásokat vehetünk igénybe, ami azt jelenti, hogy egy külön leíró fájlban deklaráljuk, hogy milyen szolgáltatásokat szeretnénk, így a programkódban már ténylegesen az üzleti logikával foglalkozhatunk. Bármely EJB komponens igénybe veheti ezeket a szolgáltatásokat, amely a 3.0-ás verziótól kezdve speciális metaadatok, úgynevezett annotációk segítségével történik. A korábbi verzióknál egy külön XML konfigurációs fájlra is szükség volt. Amikor annotációkat használunk, olyan mintha deklaratív módon programoznánk. A fejlesztő meghatározza mit kell tenni, a rendszer pedig hozzáadja a megfelelő kódot. Az EJB 3.0 specifikáció háromfajta EJB-t definiál:

- *Session bean*
- *Entity bean*
- *Message-driven bean*

Ezek közül a Session és a Message-driven bean-eket az üzleti folyamatok megvalósításánál használjuk, míg az Entity bean-ek a perzisztencia réteg felépítésénél játszanak szerepet.

6.2.1 Entitások

Az EJB 3-as új perzisztencia megoldást biztosít a fejlesztők számára, amely automatikus objektumrelációs leképezést biztosít, ez a Java Persistence API, röviden JPA. A fejlesztés során a Netbeans 6.8-as integrált fejlesztői környezetet használtam, melyben a TopLink Essentials az alapértelmezett JPA implementáció. A TopLink egy objektumrelációs leképező eszköz, amely mindkét irányban használható. Java osztályokból képes adatbázis struktúrát alkotni, és adatbázis táblák alapján entitás osztályokat tud létrehozni. Az implementáció során először létrehoztam a szükséges Java osztályokat, majd a TopLink ezek alapján generálta az adatbázis táblákat.

Az alábbi példán a `Users` osztályt definiálok, mely a felhasználók adatait tartalmazza.

```
@Entity
public class Users implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    public Long getId() {
        return id;
    }

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

    @Column(name="NAME", length=50)
    protected String name;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

1. példa. A Users entitás osztály

Ahogy a **1. példán** láthatjuk egy EJB entitás nem sokban különbözik egy szokványos Java osztálytól. Az annotációkat leszámítva egy egyszerű Java osztályt látunk, melyben minden argumentumhoz tartozik egy lekérdező és beállító metódus, valamint tartalmaznia kell egy olyan konstruktort, amelynek nincsenek argumentumai. Vizsgáljuk meg, hogy melyek azok az annotációk - ezeket a `javax.persistence` csomag tartalmazza - melyeket az entitás osztály elkészítése során használunk. Az `@Entity` segítségével határoztuk meg az osztály típusát, vagyis ez egy entity bean lesz. Az entitás osztály az adatbázisban lesz tárolva, ezért szükség van egyedi azonosítókra. Az `@Id` annotációt a megfelelő argumentum előtt használjuk. A `@Column` segítségével a tábla egyes oszlopainak a nevét adhatjuk meg, illetve lehetőségünk van az adott oszlopban tárolandó elemek maximális hosszának megadására is. A `@Column` megadása nem kötelező, ekkor az oszlop neve az adott argumentum nevével egyezik meg.

A Toplink az `Users` osztályhoz a következő adatbázis táblát generálja (**2. példa**).

Users	
	Id: INTEGER
	Name: VARCHAR(50)
	User_name: VARCHAR(20)
	Password_2: VARCHAR(20)
	Userrole: ENUM

2. példa. A `Users` tábla felépítése

6.2.2 Entitások menedzselése (EntityManager)

Az `EntityManager` a JPA egyik legfontosabb része, hiszen olyan interfészeket és metódusokat biztosít, melyekkel az entitásokat kezelhetjük. Amikor egy új entitást létrehozunk az `EntityManager` feladata az adatbázisban történő megjelenítés. Azt mondhatjuk tehát, hogy az `EntityManager` egy hidat képez az oo és a relációs világ között, ahogy az a **17. ábrán** látható.



17. ábra. `EntityManager`

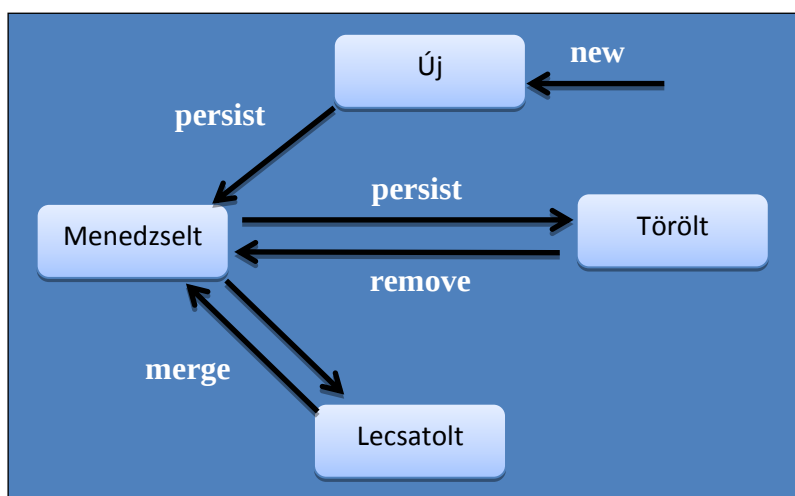
Minden EntityManager példány entitások egy halmazával dolgozik, ezeket az entitáshalmazokat perzisztencia kontextusnak nevezzük. A @PersistenceContext annotációt használva referenciát kérünk egy EntityManager-re. A konténer feladata az lesz, hogy az EntityManager példányhoz hozzárendelje az aktuális JTA tranzakcióhoz tartozó perzisztencia kontextust. Amikor egy entitás menedzselt entitássá válik, az EM feladata, hogy szinkronizálja az objektumot és az adatbázist. Amíg egy entitás menedzselt, az EM egy referenciát tart nyilván, és ha érzékeli, hogy valami megváltozott, akkor automatikusan szinkronizálja az adatokat. Ha egy entitást lecsatolunk, akkor azt az EntityManager már nem vezérli.

Szükségünk van egy konfigurációs xml fájlra, vagyis a persistence unit-ra, mely az entitásokkal kapcsolatos kiegészítő információkat tartalmazza. Ha módosítani szeretnénk a beállításokon, akkor minden entitáshoz új persistence unit-ot készíthetünk. Például akkor, ha az egyik entitást egy másik adatbázisban szeretnénk tárolni, vagy másik providert szeretnénk használni. Amennyiben a beállításokat nem akarjuk változtatni, akkor egy unit több entitáshoz is tartozhat.

6.2.3 Entitások élettartalma:

Az entitások mindig négy állapot valamelyikében szerepelnek (**18. ábra**):

új, menedzselt, lecsatolt, törölt



18. ábra. Az entitások állapotai

Egy entitás a `persist`, `merge` vagy `refresh` metódusok hatására válik vezérelté. Ha egy új entitást akarunk létrehozni azt a `new` metódussal tehetjük meg. A `persist` metódussal egy entitást menedzselt állapotba hozhatunk, amely hatására bekerül az `EntityManager` példányához tartozó perzisztencia kontextusba (**2. példa**).

```
@PersistenceContext
private EntityManager em;

public void newUser(Users user) {
    em.persist(user);
}
```

2. példa. Perzisztencia kontextus

Az `EntityManager`-hez tartozó entitásoknál a tranzakciók végén történik a szinkronizáció. A perzisztencia kontextus megszűnésekor az addig menedzselt entitások lecsatolt állapotba kerülnek, ilyenkor hagyományos java objektumként viselkednek. Ha egy entitást lecsatolunk, utána azt más rétegekben is használni tudjuk, például átvisszük a webrétegbe, ott módosítjuk az értékét és újra meghívjuk az EJB réteget. Egy lecsatolt entitást a `merge` metódussal hozhatjuk újra menedzselt állapotba. A `remove` meghívásával az entitás a tranzakció végén törlődik az adatbázisból, de a `persist` segítségével a tranzakció vége előtt megint menedzselt állapotba hozható.

6.2.4 Entitások elérése

A JPA segítségével háromféleképpen tudjuk a relációs adatbázisban tárolt adatokat elérni:

- Az `EntityManager` `find` metódusával
- Java Persistence Query Language (JPQL) használatával
- SQL lekérdezések használatával

A `find` metódussal csak elsődleges kulcs alapján tudunk keresni. Gyakran előfordul, hogy az elemek egy csoportját keressük - melyek rendelkeznek valamilyen közös tulajdonsággal – ebben az esetben más eszközt kell választanunk. Ekkor használhatjuk a `javax.persistence.query` csomagban lévő query API-t, melynek segítségével saját lekérdezéseket írhatunk. A JPA-ban lévő query API lehetővé teszi számunkra, hogy hagyományos SQL, illetve JPQL lekérdezéseket írjunk. Kétféleképpen definiálhatunk lekérdezéseket, az alábbiakban a statikus névvel ellátott eset bemutatását láthatjuk (**3. példa**).

```
@NamedQueries({  
    @NamedQuery(name="Users.findById",  
        query="SELECT u FROM Users u WHERE u.id = :id"),  
    @NamedQuery(name="Users.byUserName",  
        query="SELECT u FROM Users u  
        WHERE u.userName = :userName")  
})
```

3. példa. Statikus névvel ellátott lekérdezések a users osztályban

Statikus esetben a lekérdezéseket előre definiálnunk kell az entitás osztályokban, és a nevük alapján hivatkozunk rájuk a session bean-ekből (**4. példa**).

```
public List<Users> allUser() {  
    Query query = em.createNamedQuery("Users.findAll");  
    return query.getResultList();  
}
```

4. példa. A statikus lekérdezés meghívása

A névvel ellátott, statikusan előre definiált lekérdezéseknek az egyik legnagyobb előnye az újrafelhasználhatóság. Miután létrehoztuk, a nevével bármikor hivatkozhatunk rá, ezzel is fokozva a rendszer teljesítményét. Ezeket a lekérdezéseket az entitás osztályokban helyezzük el, így ha bármit módosítani akarunk, nem kell az üzleti logikai réteghez hozzányúlnunk, mindezt megtehetjük a perzisztens rétegben.

Tisztáznunk kell az entitások elérésével kapcsolatban a JPQL és az SQL közötti különbséget. Amíg a JPQL a java osztályokon, objektumokon operál, addig az SQL a relációs adatbázisban. A JPQL lekérdezéseket egy processzor (JPQL Query Processor) fordítja le SQL lekérdezésekké. Az alkalmazás fejlesztése során a JPQL segítségével hoztam létre a lekérdezéseket, mert az EJB 3.0-tól kezdve számos újítást tartalmaz, mellyel bármilyen lekérdezést létrehozhatunk.

6.2.5 Entitások közötti kapcsolatok

A JPA segítségével az entitások közötti kapcsolatok objektumrelációs leképezése is könnyen megoldható a megfelelő annotációk használatával. Ezek a következők: `@OneToOne`, `@OneToMany`, `@ManyToOne`, `@ManyToMany`. A köztük lévő kapcsolat lehet egyirányú illetve kétirányú. Egyirányú esetben csak az egyik entitásból lehet a másik oldalt elérni, fordítva nem lehetséges.

```
@JoinColumn(name = "Semester_Id", referencedColumnName = "ID")
@ManyToOne
private Semester semesterId;
public Semester getSemesterId() {
    return semesterId;
}
```

5. példa. Az entitások közötti kapcsolatok kialakítása annotáció segítségével

Az **5. példában** a külső kulcsot tartalmazó oszlopot egy külön annotációval a `@JoinColumn` segítségével jelöljük meg. Lehetőség van azonban a `@JoinTable` kulcsszót is használni. Az alkalmazásfejlesztés korábbi fázisaiban, már elkészítettük azokat a terveket, melyek során elemeztük, hogy az egyes osztályok milyen kapcsolatban vannak egymással. Ennek segítségével és a megfelelő annotációkkal a szükséges adatbázis struktúrát tudjuk kialakítani.

6.2.6 Az üzleti logika

Egy alkalmazásnak számtalan feladatot kell ellátnia, ebben a konkrét esetben: a felhasználó beléptetése, a tárgyfelvétel, a tárgyak hirdetése, a diplomamunka témák listázása, stb. Az alkalmazás legmeghatározóbb része az üzleti logika, a fejlesztés során ennek elkészítése jelenti a legnagyobb kihívást. A fejlesztők számára az lenne az ideális, ha a fejlesztés során elég lenne csak az üzleti logikára koncentrálni. Az üzleti logikát megvalósító session bean-ek számos olyan szolgáltatást vehetnek igénybe, melyek megkönnyítik a fejlesztést.

A session bean-ek által igénybe vett szolgáltatások:

- *Többszálúság*

Az egyik legfontosabb tulajdonsága a szerver oldali alkalmazásoknak, hogy egyszerre sok felhasználót képesek kiszolgálni. Ezekre a párhuzamosan érkező kérésekre a legkézenfekvőbb megoldás a többszálúság, melyet a futási környezet biztosít. Mivel a részleteket elrejtjük előlünk, ezért úgy fejleszthetjük a rendszerünket, mint egy hagyományos alkalmazást.

- *Távoli elérés*

A futási környezet a kommunikációs protokoll hálózati szintű részleteit elrejt, ezzel jelentősen könnyíti a fejlesztők munkáját.

- *Tranzakció kezelés*

A tranzakciók lebonyolítását rugalmasan változtathatjuk anélkül, hogy az alacsony szintű részletekkel törődnénk.

- *Biztonság*

A futási környezet hatékony és megbízható biztonsági rendszert nyújt a fejlesztők számára.

6.2.7 Session Bean

Egy session bean két részre osztható: implementációs osztály és interfész. Az utóbbira azért van szükség, mert a kliensek közvetlenül nem férhetnek hozzá az implementációs osztályhoz, így a session bean által nyújtott bármely szolgáltatást ezeken az interfészekeken keresztül érhetik el. Az interfész tehát tartalmazza mindazon metódusokat, melyeket meghívhatunk a kliens oldalról.

Az interfész lehet lokális - ez az alapértelmezett - és lehet távoli, ezt a `@Remote`, illetve a `@Local` annotációk segítségével jelöljük. A **6. példán** egy távoli interfészt láthatunk.

```
@Remote
public interface UserSessionBeanRemote {

    public void newUser(entities.Users user);

    public void update(entities.Users user);

    public java.util.List<entities.Users> allUser();

    public java.util.List<entities.Users> findTeacher();

    public entities.Users findById(long id);

}
```

6. példa. A *UserSessionBean* osztály *Remote* interfésze

Ha egy interfész lokális, akkor csak azonos Java Virtual Machine-ben (JVM) futó programból lehet elérni. Ha távoli interfészt használunk, azt bármely JVM-ben futó programból el tudjuk érni. Az interfészben előforduló metódusok specifikációja az implementációs osztályban található. Az interfész és az osztály közötti összerendelés az `implements` kulcsszó segítségével történik. Előfordulhat, hogy egy bean osztályhoz egyszerre helyi és távoli interfészek is tartoznak.

Az EJB kétféle session bean-t definiál:

- *Állapotmentes (Stateless Session Bean)*
- *Állapottal rendelkező (Statefull Session Bean)*

Az állapotmentes session bean-ek esetén a `@Stateless` annotációt kell feltüntetnünk az osztály elején, ahogy ezt a **7. példa**n lévő kódrészletben láthatjuk.

```
@Stateless
public class UserSessionBean implements UserSessionBeanRemote {

    @PersistenceContext
    private EntityManager em;

    public void newUser(Users user) {
        em.persist(user);
    }

    public Users login(String name, String password)
        throws DatabaseException{
        List<Users> user = this.findUserByUserName(name);
        Users u;
        if(user.isEmpty())
            throw new DatabaseException("Nincs ilyen felhasználó")
        else
            if(!user.get(0).getPassword().equals(password))
                throw new DatabaseException("Hibás jelszó");
        u = user.get(0);
        return u;
    }
}
```

7. példa. A *UserSessionBean* osztály eleje

Az ilyen típusú session bean-eket akkor használjuk, ha nem akarjuk, hogy a hívott bean állapota megmaradjon az egyes metódushívások között. Ebből adódóan bármely állapotmentes session bean példány akármelyik klienst kiszolgálhatja. Ezek a példányok egy úgynevezett *instance pool*-ba rendezhetők, így ezek újrafelhasználhatók, amely sokkal hatékonyabb működést eredményez, mintha minden egyes esetben létrehoznánk és törölnénk őket.

Az állapottal rendelkező session bean-ek esetén a `@Stateful` annotációt kell használnunk. Ezeket a típusú bean-eket akkor használjuk, ha fontos, hogy a hívott session bean állapota megmaradjon a metódushívások között. Ebben az esetben is használhatunk *instance pool*-t, azonban itt az aktiválás és a passziválás segít abban, hogy ezt megvalósíthassuk. A passziválás hatására a memóriabeli példány perzisztens módon tárolódik, aktiválás után pedig visszakerül a memóriába az eredeti állapotban.

6.2.8 Callbacks metódusok

Vannak olyan esetek, amikor tudni szeretnénk bizonyos eseményekről, például, hogy egy példány mikor jött létre, vagy mikor semmisült meg. Ekkor alkalmazzuk a *callbacks* metódusokat. Ezeket korábban kötelező volt implementálni, függetlenül attól, hogy használni akartuk-e vagy sem. Az EJB 3.0-tól kezdődően a callback metódusok implementálása nem kötelező, ezért csak akkor használjuk, ha valóban az eseményekről szeretnénk értesülni.

A `@PreDestroy` a példány megsemmisítése előtt fut le, míg a `@PostConstruct` közvetlenül a példány létrejötte után. Ez a két callback metódus mindkét session bean típus esetén létezik. Az állapottal rendelkező bean még kiegészül két callbackkel, ezek a `@PrePassivate`, `@PostActivate`.

6.2.9 Dependency Injection

A session bean-ek elérése az EJB 3.0-tól a dependency injection segítségével egyszerűen megoldható. Ez az jelenti, hogy az `@EJB` használva a konténer feladata lesz a megfelelő interfész példányt biztosítani. Korábban ezt csak JNDI kereséssel lehetett megoldani.

A **16. példában** láthatjuk, hogy a Login osztályból hogyan érhetjük el a Users osztály távoli interfészét a dependency injection segítségével.

```
public class Login {

    @EJB
    private UserSessionBeanRemote userSessionBean;

    public String enterAction() {
        Users u;
        try {
            u = this.userSessionBean.login(userName, password);
        } catch (DatabaseException ex) {
            addMessage(ex.getMessage());
            return null;
        }
    }
}
```

8. példa. *Dependency injection szemléltetése a Login osztályon keresztül*

Ahogy a Login osztályban láthatjuk, a dependency injection használatával a fejlesztés jóval egyszerűbbé válik, nem kell már JNDI kereséseket használnunk, ezt egyszerűen rábízhatjuk a

konténerre. A dependency injection-t csak konténeren belül tudjuk így megoldani, konténeren kívül továbbra is explicit módon kell JNDI keresést végrehajtani. A dependency injection segítségével nemcsak EJB-k hanem más erőforrások is elérhetők, ehhez a `@Resource` annotációt kell használnunk.

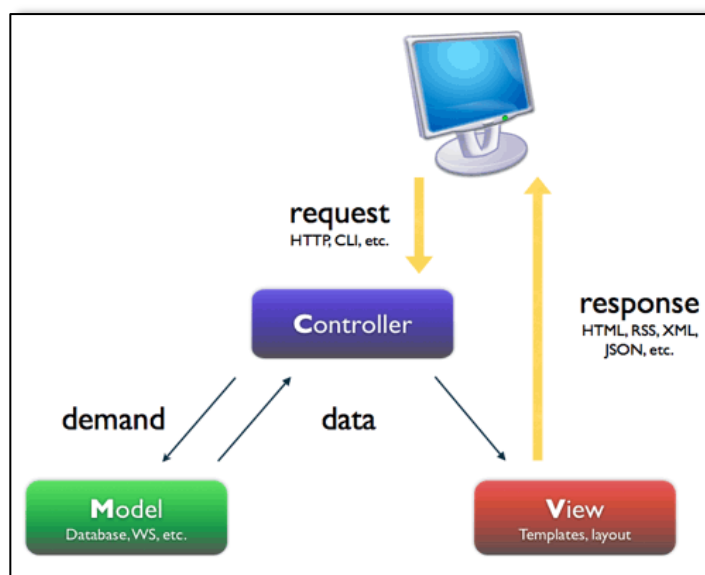
Ezzel áttekintettük azokat az EJB-vel kapcsolatos legfontosabb kérdéseket, melyek az alkalmazás fejlesztése során előfordultak. Ezután rátérhetünk a webréteg fejlesztése során használt technológiára.

6.3 Java Server Faces

A webes alkalmazások rohamos elterjedésével felmerült az igény, hogy a webréteg fejlesztésénél felmerülő közös problémákat – ilyen például a konverzió, validáció, vagy éppen a rugalmas navigáció – a rendszer kezelni tudja. Ezeknek a feladatoknak az ellátására számos keretrendszer alakult ki. 2004-ben a Sun, a korábbi keretrendszerek egyesítésével, megalkotta a Java Server Faces keretrendszert, amely része a Java EE 5 platformjának.

A webes keretrendszerek célja, hogy összekapcsolja a grafikus felületet az üzleti logikával. A JSF egy komponensorientált webes keretrendszer, amely megkönnyíti a webes alkalmazások fejlesztését, hiszen számos felhasználói interfészt tartalmaz. A legtöbb keretrendszer, köztük a JSF is az *Modell View Control* tervezési minta mentén épülnek fel. A *Modell* részben található elemek az üzleti logikáért és az adatstruktúráért felelősek, míg a *View* rész feladata a megjelenítés. A *Controller* megteremti a *Modell* és a *View* közötti kapcsolatot, a felhasználói eseményeket átalakítja a Modell által végrehajtható akcióra, majd az eredményeket a View segítségével megjeleníti a felhasználó számára.

Az MVC tervezési mintát a **19. ábrán** látjuk.



19. ábra. MVC tervezési minta

6.3.1 A JSF keretrendszer felépítése

6.3.1.1 Java Server Pages (JSP)

A JSF esetén a nézet megírása Java Server Pages lapok segítségével történik. A JSP olyan szerver oldali technológia, amely segítségével dinamikus tartalmat állíthatunk elő. Ennek a technológiának az egyik legfontosabb tulajdonsága, hogy külön tudjuk választani a programozó és a webfejlesztő munkáját. A JSP és a szervlet technológia egymástól nem független, hiszen a JSP oldalak is szervletté alakulnak át. A JSP végrehajtásakor a hozzá tartozó szervlet `do` módszere fut le.

Ezeket a JSP oldalakat egészítjük ki JSF specifikus tagkönyvtárakkal:

- az „f” prefixű tagkönyvtárban,

JSF központi elemei vannak, melyeket a következő direktíva után használhatunk:

```
<%@taglib prefix="f" uri="http://java.sun.com/jsf/core"%>
```

Ezek a tagek a keretrendszer működését a következőképpen befolyásolják:

- `f:view`: ezen belül használhatunk más JSF komponenseket
- `f:convertNumber`: egy `numberconverter` példányt regisztrál
- `f:validateLength`: regisztrál egy validátort és ezzel ellenőrizhetjük, hogy a karakterlánc megfelelő hosszúságú-e
- ...

- a „h” prefixű tagkönyvtárban,

a HTML elemek találhatók melyeket a következő direktíva után használhatunk:

```
<%@taglib prefix="h" uri="http://java.sun.com/jsf/html"%>
```

Nézzük meg a legfontosabb HTML tageket:

- `h:inputText`: szöveget kérhetünk be illetve írhatunk ki vele
- `h:outputText`: szöveget írhatunk ki vele
- `h:commandButton`: akciógomb
- `h:dataTable`: adattábla megvalósítására használható
- ...

A **9. példán** egy JSF tagekkel kiegészített JSP fájlt látunk.

```
<%@taglib prefix="f" uri="http://java.sun.com/jsf/core"%>
<%@taglib prefix="h" uri="http://java.sun.com/jsf/html"%>

<f:view>
<html>
<head></head>
<body>
  <h:form>
    <div class="box">
      <div class="cim">
        <h:panelGrid >
          <f:facet name="header">
            <h:outputText value="Bejelentkezés" />
          </f:facet>
          <f:facet name="footer">
            <h:panelGroup>
              <h:commandButton action="#{Login.createAction}"
                              value="Regisztráció"/>
              <h:commandButton action="#{Login.enterAction}"
                              value="Belép" />
            </h:panelGroup>
          </f:facet>
          <h:outputText value="Loginnév"/>
          <h:inputText id="name" value="#{Login.userName}"/>
          <h:outputText value="Jelszó"/>
          <h:inputSecret id="password" value="#{Login.password}"/>
        </h:panelGrid>
      </div>
    </div>
    <h:messages/>
  </h:form>
</body>
</html>
</f:view>
```

9. példa. Egy JSF tageket tartalmazó JSP osztály

A JSP oldalak a JSF Expression Language segítségével tartják a kapcsolatot a menedzselt osztályokkal. A `h:inputText` vagy `h:outputText`, `value` attribútuma segítségével lehet a komponensben tárolt értéket a menedzselt osztály példányához rendelni. Az `inputText` esetén, ha megváltoztatjuk a komponens értékét, akkor azt a setter-ek segítségével a menedzselt osztálypéldányokban is végrehajtjuk. Ha a modellben változik meg valamely tulajdonság, akkor azt a változást a komponensben is végrehajtjuk. Láthatjuk tehát, hogy folyamatos szinkronizáció van a JSP oldalak és a hozzájuk tartozó menedzselt java bean-ek között. A parancsokat a `commandbutton` segítségével hajtjuk végre, ezzel a hozzá tartozó példány megfelelő metódusát hívjuk meg.

A JSF tageket használó JSP fájl, ugyanúgy szervletté fordítódik, mint a más JSP fájl, azonban van egy különlegességük. Amikor lefut a fájlhoz tartozó szervlet `service` metódusa, felépít egy memóriabeli fastruktúrát, ez a View. A legfontosabb elem a fa gyökerben található, ez a ViewRoot, innen a teljes nézet bejárható.

Komponensek

Minden komponens az *UIComponent* osztályból származik, amely a *StateHolder* interfészt implementálja, ez azért fontos, mert a JSF képes a kérés feldolgozás során az egyes komponensek állapotát helyreállítani, majd újra elmenteni.

Lássuk a legfontosabb UI komponenseket:

- Az *UIForm* osztály a `h:form` taghez tartozik, egy oldalon több form is előfordulhat, ezért mikor kérés érkezik a JSF-hez, a megfelelő formot be kell állítanunk.
- Az *UICommand* azokat az elemeket tartalmazza melyek segítségével a felhasználó funkciókat aktiválhat, parancsokat adhat.
- Az *UIMessage* osztályok segítségével a JSF kérésfeldolgozás közben keletkezett üzeneteket tudjuk megjeleníteni.
- *UIOutput* osztály segítségével lehetőségünk van adatok megjelenítésére, ezek az értékek minden esetben String típusúak.
- Az *UIInput*, amely az *UIOutput* osztály egyik leszármazottja, segítségével a felhasználó adatokat képes megadni, illetve azokat módosítani és elmenteni a modellben.

Render Kit-ek

A JSF úgy lett kialakítva, hogy ne csak HTML alapú környezetben tudjuk használni, hanem bárhol, ahol megjelenítésre van szükség. Ehhez úgynevezett *Render Kit*-ekre van szükség, amely az osztály és a konkrét implementáció közötti megfeleltetést biztosítja. Ezt úgy képzelhetjük el, mint egy tolmácsot a szerver és a kliens oldal között. A szerver oldalról nézve úgy végzi el a kódolást, hogy azt a kliens is megérti. Amikor a JSF kap egy kérést a felhasználótól, dekódolja azt, beállítja a paramétereket, és meghatározza a komponens értékét. A Render Kit-eknek több típusát különböztetjük meg:

- HTML
- WML, amit a WAP böngésző használ

6.3.1.2 Menedzselt Bean

Ezek olyan egyszerű Java osztályok, amelyek segítségével a nézet és a modell réteg összekapcsolható. Egyrészt vezérlő szerepet töltenek be, másrészt a nézet réteg számára modellként viselkednek. Minden JSP oldalhoz létrehozunk egy neki megfelelő menedzselt osztályt, melyben a nézet adatokat tud eltárolni, és ugyaninnen tud adatokat lekérni. A különböző események megvalósításához, a menedzselt osztályhoz tartozó metódusok hívódnak meg. A **10. példában** a Login oldalához tartozó menedzselt bean-t látjuk.

```
@ManagedBean(name="Login")
@SessionScoped
public class Login {
    @EJB
    private UserSessionBeanRemote userSessionBean;

    Users user;
    String userName, password;

    public Login() {
        user = new Users();
    }

    public String enterAction(){...}

    public String createAction(){...}

    protected void addMessage(String message){...}

    ...
}
```

10. példa. A Login.jsp laphoz tartozó menedzselt bean

A JSF 1.2-ben még egy külön fájlban (faces-config.xml) kellett beállítani a menedzselt bean-ek definícióját, a navigációs szabályokat és itt kellett beállítani a menedzselt bean-ek környezetét is. Azonban a JSF 2.0-tól kezdődően itt is megjelennek az annotációk – úgy, mint az EJB-ben – ezután már elég ezeket feltüntetni, a többit a program kezeli.

Egy egyszerű Java osztályból két annotáció segítségével készíthetünk menedzselt bean-t:

- Az egyik `@ManagedBean` annotáció, melynek segítségével jelöljük, hogy Menedzselt osztályt akarunk készíteni, a hozzá tartozó `name` paraméterben adjuk meg, hogy az osztály példányát milyen néven szeretnénk elérni.

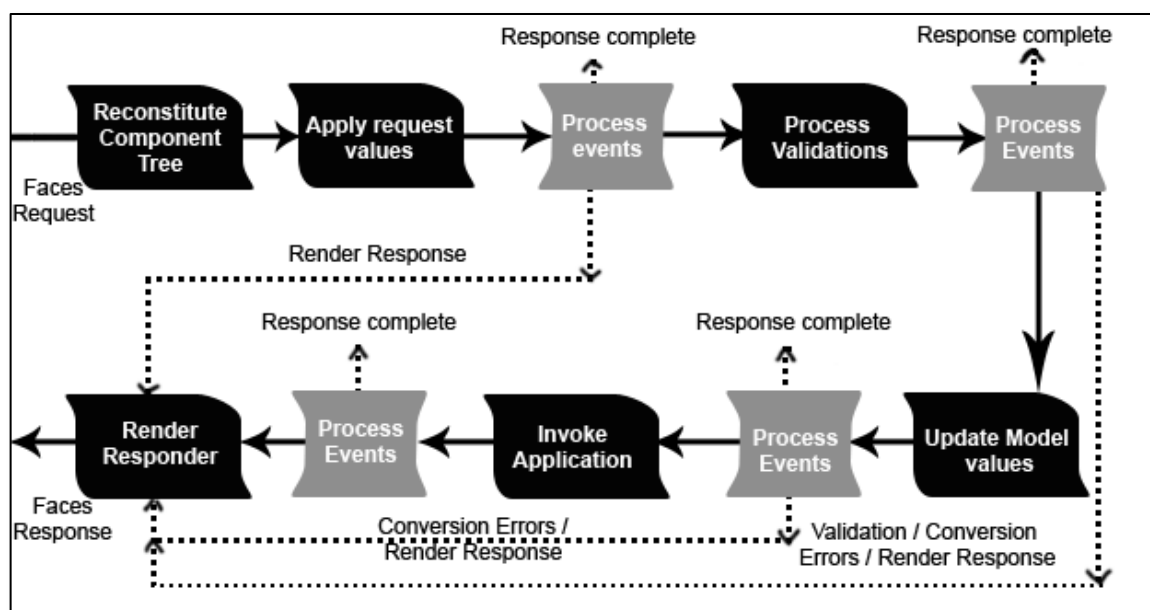
- A másik annotáció a menedzselt bean élettartamát határozza meg: `@RequestScoped`, `@SessionScoped`, `@ApplicationScoped`

A menedzselt bean-ek egy példányát a JSF deklaratív módon hozza létre a kérelmfeldolgozás alatt, különbség köztük az, hogy ezek a példányok mikor kerülnek szemétygyűjtésre. A `@RequestScoped` esetén közvetlenül a kérelmfeldolgozás után, `@SessionScoped` esetén addig nem kerülnek szemétygyűjtésre, amíg a felhasználó session-je él. Amikor `@ApplicationScoped` annotáció segítségével hozzuk létre a példányt, akkor az alkalmazás teljes futása alatt megmarad.

Ha egy egyszerű java osztály tartalmazza ezeket az annotációkat menedzselt osztállyá válik, azonban ezen osztályoknak tartalmazniuk kell - a megfelelő változókhoz tartozó - lekérdező és beállító metódusokat, ugyanis az UI komponensek ezek segítségével tartják a kapcsolatot egymással.

6.3.2 A JSF élekciklusa

A JSF minden kérés feldolgozásánál egy állapotgépet használ, ez vezérli a folyamatokat. Hat különböző állapotot tartalmaz, és az egyes állapotok között történik az esemény feldolgozás. Az egyes állapotokat a **20. ábra** szemlélteti.



20. ábra. A JSF állapotgépe

- *Nézet visszaállítása*

Azokat a kéréseket, melyek olyan JSF lapokra vonatkoznak, amik állapota egyszer már elmentésre került, postback-nek nevezzük. A nézet először eldönti, hogy ez a kérés postback-e. Postback esetén a JSF megkeresi az adott nézet komponensfáját, amiben az egyes komponensek állapota is megtalálható, majd visszaállítja az egyes komponensek állapotát.

Nem postback esetén létrehozza a komponensfa gyökerét, majd Nézet generálása állapotra ugrik.

- *Kérés paraméterek rögzítése*

Minden megjelenítendő input komponens ellenőrzi, hogy a kérésben nincs-e hozzárendelt paraméter, ha talál ilyet, akkor azt az UIInput osztály submittedValue változójában tárolja el. Ez egy karaktersorozat, amely majd a későbbiek során válik objektummá.

- *Validáció futtatása*

A JSF konverterek a karaktersorozatot a megfelelő java objektumra alakítják, ezután már a validátorok is értelmezni tudják a komponensekben szereplő értékeket. Vannak beépített konverterek, ilyenek például a szám vagy dátum konverterek, de explicit módon is létrehozhatjuk a saját konverterünket. Ezután kezdődik a validáció. A ViewRoottól kezdve, az egyes komponensek validálása mélységi bejárással történik, de csak azoknál a komponenseknél, melyekre teljesülnek a következő feltételek:

- a felhasználó által küldött úrlapon szerepelnek
- a kérésben megtalálható a hozzájuk tartozó paraméter
- a rendered attribútuma true
- az immediate attribútum false

A validáció során, ha egy komponensben a required attribútum igaz, viszont a hozzá tartozó paraméter üres, a JSF hibát jelez. Ha a paraméter nem üres, akkor a komponenshez tartozó minden validátort le kell futtatni. Ha az egyik hibát jelez, akkor a JSF hibaüzenetet küld. Ha minden validátor sikeresen lefutott, akkor a JSF a konvertált, validált értékeket beírja az UIOutput osztály value tagváltozójába. Ha hiba lépne fel, a Nézet generálása pontra ugrik.

- *Modell aktualizálása*

Ebben a fázisban az ellenőrzött adatokat beírjuk a value attribútumban megadott Managed bean változójába, ehhez a hozzá tartozó setter metódust használjuk fel.

- *Az alkalmazás meghívása*

A fázishoz tartozó események itt továbbítódnak a komponensekhez, vagyis az UICommand-ból származó összes komponens ActionListener-jében, illetve ActionListener-jében található metódus meghívása itt történik.

- *Nézet generálása*

A komponensfát bejárva generálja le a megfelelő kimenetet a Render Kit segítségével.

6.3.3 Navigáció

A JSF 2.0-tól kezdve a navigáció beállításához már nem szükséges külön xml konfigurációs fájl készítenünk, és ott kézzel beállítani a navigációt. Ezután elég, ha a `CommandLink`, `CommandButton` által meghívott metódus egy String-et tér vissza, melyben a célt tartalmazó JSP oldal neve szerepel a .jsp kiterjesztés nélkül.

Az alábbiakban a fejlesztés során használt technológiákat kívántam bemutatni, ezzel betekintést nyújtani az egyes eszközrendszerek használatába.

7. Tesztelés

A forráskód elkészítésével még nem ért véget az alkalmazás fejlesztése, mert az így elkészített programról még nem tudjuk, hogy az elvárásoknak megfelelően működik-e. Ezért a tesztelés során a programot tudatosan ellenőrizzük hibakeresés céljából. Ebben a fázisban is megjelenik a verifikáció és a validáció, csak itt már az elkészült alkalmazást ellenőrizzük.

A verifikáció során azt vizsgáljuk, hogy az előállított szoftver megfelel-e a specifikációnak, kielégíti-e a követelményeket. Ez a folyamat a hiányosságtesztelés, ami akkor hatékony, ha minél több hibát talál.

A validáció során azt ellenőrizzük, hogy az elkészült alkalmazás tényleg a felhasználó igényeinek megfelelően készült-e el. Előfordulhat olyan eset, hogy az alkalmazás a követelményspecifikációnak eleget tesz, de nem a felhasználó elvárásainak megfelelően működik. Ahhoz, hogy ezt elkerüljük, minden fázis végén validálni kell az előállított dokumentumokat. Az elkészült program esetében a validáció tesztelésére a statisztikai tesztelést szokták alkalmazni, amely leginkább a nem funkcionális követelményeket vizsgálja: válaszügy, megbízhatóság, használhatóság.

A tesztelés során kiderül, ha a szoftverben hiányosságok vannak, ha nem a specifikációnak megfelelően működik, vagy teljesítmény szempontjából elmarad a várakozásoktól. Miután a fejlesztő csapat, „jónak” ítélte az alkalmazást előáll az alfa verzió, amit a fejlesztőktől független személyeknek adnak ki béta tesztelésre. Ha ez a tesztelés is sikeresen zárul, az alkalmazás elkészült.

8. Összefoglalás

Amíg egy eltervezett alkalmazásból a felhasználók által eredményesen használható szoftver születik, sok munkára van szükség, a felmerülő problémákat megoldva egyre közelebb kerülünk az elérendő célhoz. Ebben az esetben, a jelenlegi tanulmányi rendszert vizsgáltam meg, ez szolgált alapként a követelmények feltárása során, és ez alapján döntöttem el, melyek azok a funkciók, szolgáltatások, melyek az általam fejlesztett rendszerben is helyet kapnak majd.

A cél egy olyan tanulmányi rendszer megtervezése és megvalósítása volt, amely a mostani rendszerből hiányzó, hasznos újításokat mutat be. Azonban a fejlesztett rendszer esetében nem beszélhetünk teljes funkcionalitással rendelkező alkalmazásról, és így a rendszer gyakorlati felhasználásáról sem. A feladat az volt, hogy egy összefüggő és használható alkalmazást készítsek, amely alkalmas az újítások szemléltetésére. Úgy érzem ezt sikerült maradéktalanul megvalósítsam.

A rendszer terve kellően átfogó, hiszen érinti mind a hallgatók, mind az oktatók, mind pedig az adminisztrátorok feladatát. Ezen feladatok eléggé függetlenek ahhoz, hogy további finomításokat, módosításokat hajtsunk végre rajtuk. Az egységes adatbázisnak köszönhetően mégis egy komplex rendszer képét nyújtják.

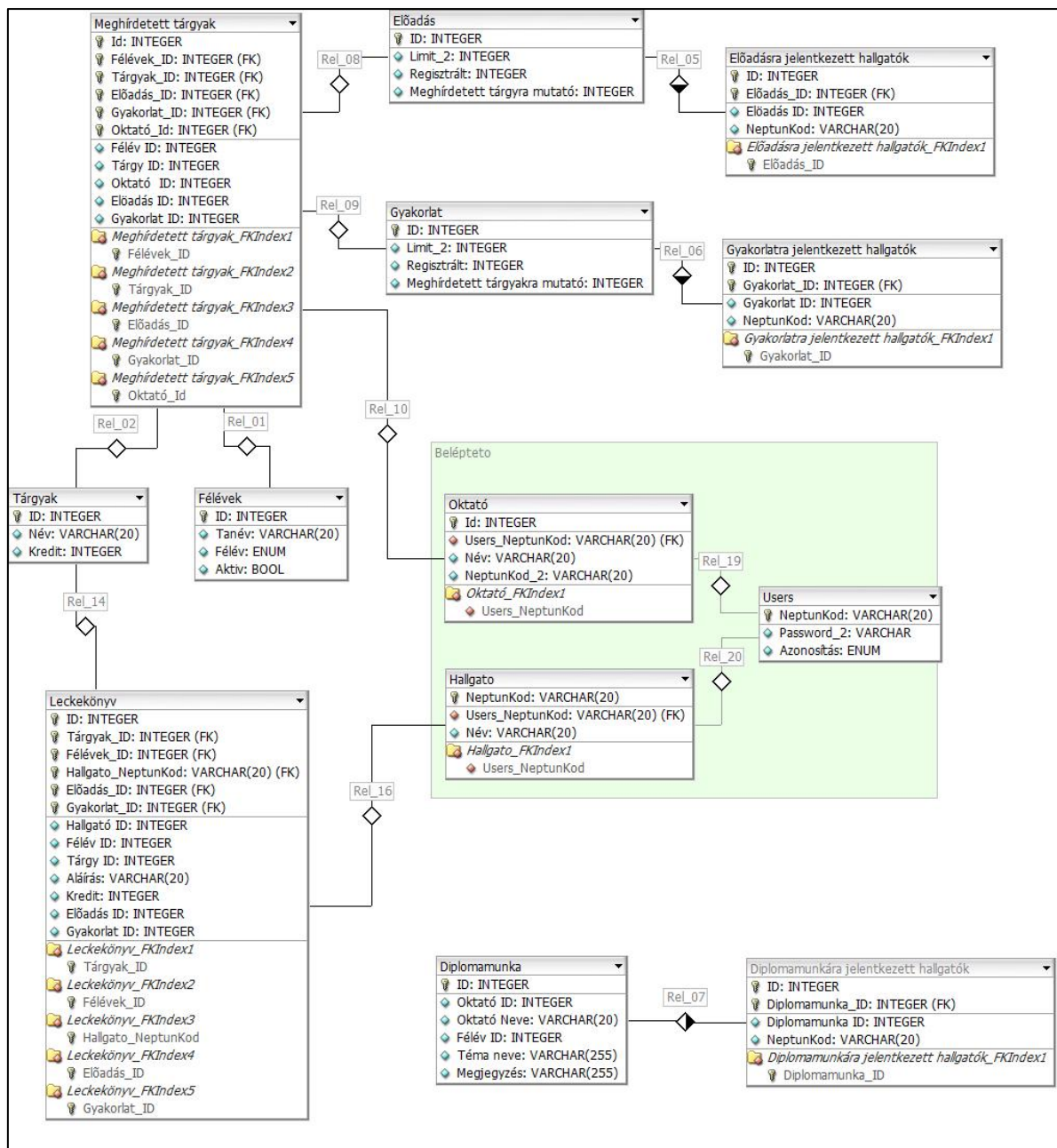
Ezzel a fejlesztéssel az volt a célom, hogy rámutassak a jelenlegi rendszer néhány hiányosságára, melyek egy későbbi fejlesztés során már gyakorlati alkalmazást is kaphatnak a tanulmányi rendszerben.

9. Irodalomjegyzék

1. *Harald Störrle – UML2*
2. *Imre Gábor – Szoftverfejlesztés Java EE platformon*
3. *Kondorosi Károly, László Zoltán, Szirmay-Kalos László –
Objektum-orientált szoftverfejlesztés*
4. *Angster Erzsébet – Objektumorientált tervezés és programozás*
5. *Sólyom-Nagy Péter – Többrétegű alkalmazások*
6. *Rendszerfejlesztés technológiája jegyzet*
7. *Debu Panda, Reza Rahman, Derek Lane –
EJB 3 in Action*
8. *Kito D. Mann – JavaServer Faces in Action*

10. Függelék

A 21. ábrán az alkalmazás mögött található adatbázis struktúrát láthatjuk.

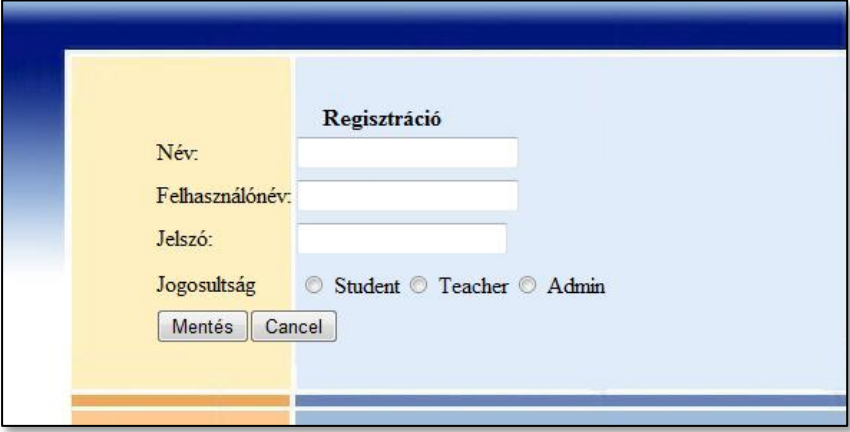


21. ábra. A rendszer adatbázis struktúrája

Az alábbiakban a rendszert képek segítségével mutatom be.



22. ábra. A rendszer bejelentkező képernyője



*23. ábra. A rendszer regisztrációs képernyője.
(A tesztelési fázisban bármely jogkörre szabad a regisztráció)*

Tanulmányi rendszer

Admin József
ADMINISZTRÁTOR

Aktuális félév kiválasztása



Aktuális félév
Tárgyak hirdetése
Meghirdetett tárgyak
Diplomamunka
Kijelentkezés

Félév	Állapot
2005/2006/1	Lezárt
2005/2006/2	Lezárt
2006/2007/1	Lezárt
2006/2007/2	Lezárt
2007/2008/1	Lezárt
2007/2008/2	Lezárt
2008/2009/1	Lezárt
2008/2009/2	Lezárt
2009/2010/1	Lezárt
2009/2010/2	Aktív
2010/2011/1	Új
2010/2011/2	Lezárt
2011/2012/1	Új
2011/2012/2	Új

Vissza Mentés

Oldal: 1 / 1

24. ábra. Adminisztrátor - aktuális félév kiválasztása

Tanulmányi rendszer

Admin József
ADMINISZTRÁTOR

Tárgy indítása



Aktuális félév
Tárgyak hirdetése
Meghirdetett tárgyak
Diplomamunka
Kijelentkezés

Kurzus neve:	Kredit
Adatstruktúrák, algoritmusok, objektumok	2 Indít
Adatbázisok	4 Indít
Adatbázisok használata I.	3 Indít
Bevezetés az informatikába	2 Indít
Hálózati szolgáltatások tervezése és m7kódvetése	3 Indít
Informatika elméleti alapjai	4 Indít
Informatikai rendszerek biztonságtechnikája	2 Indít
Java alkalmazások fejlesztése	2 Indít
Java programozási nyelv	2 Indít
Kis er7forrás-igény? adatbázisok	2 Indít
Logikai programozás	4 Indít
Mobil informatika	2 Indít
Modellezés és tervezés	3 Indít
Multimédia alapjai	2 Indít
Nagysebesség? hálózati technológiák	2 Indít

Vissza

Oldal: 1 / 3>

25. ábra. Adminisztrátor - tárgy indítása



Aktuális Félév
Tárgyak hirdetése
Meghirdetett tárgyak
Diplomamunka
Kijelentkezés

Tanulmányi rendszer

Admin József
ADMINISZTRÁTOR

Meghirdetett tárgyak

2005/2006/1 Listáz

Kurzus neve:	Kredit	
Az adat és hangkébelezés alapjai	3	Módosít
Az adat és hangkébelezés alapjai	3	Módosít
Bevezetés a bioinformatikába	4	Módosít
Bevezetés a bioinformatikába	4	Módosít
Alkalmazásszolgáltatás	2	Módosít
Cisco hálózatok	4	Módosít
Digitális képfeldolgozás	3	Módosít
Digitális technika I.	3	Módosít
Az informatikai biztonság alapjai I.	2	Módosít
Delphi II.	3	Módosít
Az alkalmazásmenedzsment alapjai	2	Módosít
Adatbázisok használata II.	3	Módosít
Digitális rendszerek I.	2	Módosít
Digitális Fotófeldolgozás	3	Módosít
Digitális technika II.	2	Módosít

Vissza

Oldal: 1 / 3>

26. ábra. Adminisztrátor - meghirdetett tárgyak listázása



Aktuális Félév
Tárgyak hirdetése
Meghirdetett tárgyak
Diplomamunka
Kijelentkezés

Tanulmányi rendszer

Admin József
ADMINISZTRÁTOR

Előadás / gyakorlat indítása

Cisco hálózatok

Előadások:

Oktató	Max létszám	
Oláh István	100	Módosít

Hozzáad

Gyakorlatok:

Oktató	Max létszám	
Varga Dániel	50	Módosít
Albert István	50	Módosít

Hozzáad

Vissza

Mentés

27. ábra. Adminisztrátor – tárgyakhoz kurzus rendelése

Tanulmányi rendszer

Admin József
ADMINISZTRÁTOR

Kiírt témák

2005/2006/1 Listáz

Téma neve	Oktató
Silverlight alapú játék fejlesztése	Varga Dániel
JavaFX alapú szoftverfejlesztés	Albert István
Adattárolási technológiák	Szabó Róbert
Fejlesztői kódok disztribúciója	Rónyai Lajos
Térinformatikai szoftver továbbfejlesztése	Rónyai Lajos

Vissza Új téma

Oldal: 1 / 1

Félév
Tárgyak hirdetése
Meghirdetett tárgyak
Diplomamunka
Kijelentkezés

28. ábra. Adminisztrátor – Kiírt témák megtekintése

Tanulmányi rendszer

Admin József
ADMINISZTRÁTOR

Új téma kiírása

Konzulens: Rónyai Lajos

Téma címe

Téma leírása

Maximális létszám 0

Megjegyzés

Vissza Mentés

Aktuális félév
Tárgyak hirdetése
Meghirdetett tárgyak
Diplomamunka
Kijelentkezés

29. ábra. Adminisztrátor – Új téma hirdetése

Tanulmányi rendszer

Nagy Gábor
HALLGATÓ

Meghirdetett tárgyak

Felvett tárgyak

Leckekönyv

Diplomamunka kiírások

Kijelentkezés

Meghirdetett tárgyak

Kurzus neve:	Kredit	Aláírás	Teljesített	
Digitális rendszerek II.	4	☐	☐	Felvez
Adatbányászat	3	☒	☐	Felvez
Diszkrét matematika I.	2	☐	☐	Felvez
Formális nyelvek és gépek	2	☐	☐	Felvez
Haladó programozás	4	☐	☐	Felvez
Formális módszerek az informatikában	3	☐	☐	Felvez
Diszkrét idejű irányítási rendszerek	3	☐	☐	Felvez
Döntéstámogató rendszerek	3	☐	☐	Felvez
Információ- és kódelmélet	4	☐	☐	Felvez
Hálózatmentesítés	4	☐	☐	Felvez
Diszkrét matematika II.	2	☐	☐	Felvez
Haladó web programozás	4	☐	☐	Felvez
Képfeldolgozás	3	☐	☐	Felvez
Képfeldolgozás gyakorlat	3	☐	☐	Felvez
Az informatikai biztonság alapjai II.	3	☐	☐	Felvez

Vissza
<Oldal: 2 / 3>

30. ábra. Hallgató – Tárgyfelvétel

Tanulmányi rendszer

Nagy Gábor
HALLGATÓ

Meghirdetett tárgyak

Felvett tárgyak

Leckekönyv

Diplomamunka kiírások

Kijelentkezés

A tárgyhoz tartozó kurzusok

Cisco hálózatok

Előadások:

	Előadó	Létszám
☐	Oláh István	100 / 0

Gyakorlatok:

	Előadó	Létszám
☐	Varga Dániel	50 / 0
☐	Albert István	50 / 0

Vissza
Mentés

31. ábra. Hallgató – Tárgyfelvétel, kurzus választás

Tanulmányi rendszer

Nagy Gábor
HALLGATÓ

Meghirdetett tárgyak

Felvett tárgyak

Leckekönyv

Diplomamunka kiírások

Kijelentkezés

Felvett tárgyak

Kurzus neve:	Kredit	
Az adat és hangkábelezés alapjai	3	Módosit / Lead
Cisco hálózatok	4	Módosit / Lead
Vissza		

Oldal: 1 / 1

32. ábra Hallgató – Felvett tárgyak listázása

Tanulmányi rendszer

Nagy Gábor
HALLGATÓ

Meghirdetett tárgyak

Felvett tárgyak

Leckekönyv

Diplomamunka kiírások

Kijelentkezés

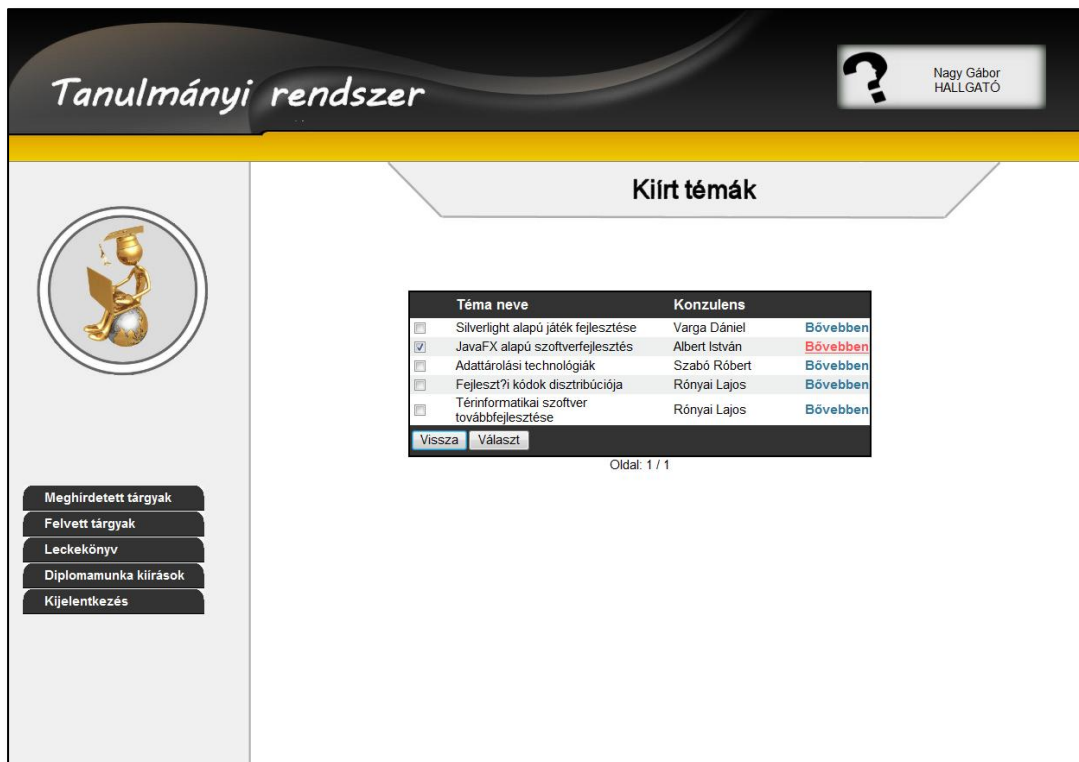
Leckekönyv

2009/2010/2 [Listáz](#)

Kurzus neve	Kredit	Aláírás	Jegy
Az adat és hangkábelezés alapjai	3	Aláírva Rónyai Lajos	közepes (3) Rónyai Lajos
Vissza			

Oldal: 1 / 1

33. ábra. Hallgató – Leckekönyv megtekintése



34. ábra. Hallgató – Kiírt diplomamunka témák



35. ábra. Hallgató – A téma részletei

Tanulmányi rendszer



Nagy Gábor
HALLGATÓ



Meghirdetett tárgyak

Felvett tárgyak

Leckekönyv

Diplomamunka kiírások

Kijelentkezés

Diplomamunka téma

Téma neve	Konzulens	
JavaFX alapú szoftverfejlesztés	Albert István	Módosítás/Törlés

36. ábra. Hallgató – Felvett diplomamunka téma

Tanulmányi rendszer



Rónyai Lajos
OKTATÓ



Kurzusok/Jegybeírás

Diplomamunka kiírások

Kijelentkezés

Oktatott kurzusok

2005/2006/1

Listáz

Tárgy	Kurzus típusa	Létszám	
Digitális rendszerek II.	Előadás	0/0	Jegybeírás
Digitális rendszerek II.	Előadás	100/0	Jegybeírás
Az adat és hangkébelezés alapjai	Előadás	100/2	Jegybeírás
Az adat és hangkébelezés alapjai	Gyakorlat	10/1	Aláírás
Az adat és hangkébelezés alapjai	Gyakorlat	20/1	Aláírás
Digitális technika II.	Előadás	200/0	Jegybeírás
Matematikai logika	Előadás	100/0	Jegybeírás
Matematikai logika	Gyakorlat	10/0	Aláírás

Vissza

Oldal: 1 / 1

37. ábra. Oktató – Oktatott kurzusok

-77-

Tanulmányi rendszer Rónyai Lajos OKTATÓ

Aláírás / Jegy beírása



Félév
Tárgyak hirdetése
Meghirdetett tárgyak
Diplomamunka

Az adat és hangkébelezés alapjai

Hallgató:	Aláírás	Jegy
Nagy Gábor	☒	közepes (3)
Kis Mariann	☒	-
Vissza Mentés		elégtelen (1) elégséges (2) közepes (3) jó (4) jeles (5)

Oldal: 1 / 1

38. ábra. Oktató – Jegy beírása

Tanulmányi rendszer Rónyai Lajos OKTATÓ

Kiírt témák

2005/2006/1 [Listáz](#)

Téma címe	Jelentkezettek	
Fejlesztői kódok disztribúciója	1	Bővebben
Térinformatikai szoftver továbbfejlesztése	0	Bővebben

[Vissza](#) [Új téma](#)

Oldal: 1 / 1

Kurzusok/Jegybeírás
Diplomamunka kiírások
Kijelentkezés

39. ábra Oktató – Kiírt diplomamunka témák