DIPLOMAMUNKA

Kiss Sándor Ádám

Debrecen
2017

Debreceni Egyetem
Informatikai Kar

Java EE Alkalmazásfejlesztés

Témavezető: Dr. Adamkó Attila
Beosztása: egyetemi adjunktus

Készítette: Kiss Sándor Ádám
Programtervező Informatikus M.Sc.

Debrecen
2017

Tartalomjegyzék

1. fejezet

Bevezetés

2015 februárjában, amikor még az utolsó félévemet kezdtem el a Debrecen Egytem - Informatikai Karán, Programtervező Informatikus alapképzésen, jelenlegi témavezetőmnek, az egyetemnek és a Liferay Hungary Kft-nek köszönhetően teljesen térítésmentesen részt vehettem egy majdnem 3000 € értékű, [6] úgynevezett *Developing for the Liferay Platform 1* képzésen, ahol megtanultam a Liferay 6.2 tartalomkezelő rendszer [3] használatát, illetve azt is elsajátítottam, hogy hogyan lehet saját alkalmazásokat, portleteket fejleszteni a platformra és hogyan lehet testre szabni a portált mind működésileg, mind kinézetileg az egyéni megrendelői igények szerint. A képzésen való részvétel egyetlen feltétele az volt, hogy a megszerzett tudás segítségével minden, a képzésen résztvevő hallgató készítsen egy olyan használható és működőképes webalkalmazást az egyetem számára, amelyet az új kari honlapon lehetne használatba venni.

A három teljes napot átölelő intenzív kurzus után a témavezetőm egy Redmine [7] nevű feladatkezelő rendszerben összegyűjtötte a fejlesztésre váró feladatokat és a képzésen résztvevő csoport minden egyes tagja kiválasztott egyet ezen feladatok közül és elkezdett fejleszteni egy-egy programot. A témavezetőmmel történő többszöri egyeztetés után arra a következtetésre jutottunk, hogy a legmegfelelőbb alkalmazás, amelyet az egyetem is hasznosítani tudna, egy sillabusz menedzselő vagy más néven sillabusz karbantartó alkalmazás lenne, amelyet az egyetem oktatói, hallgatói és esetleg a Tanulmányi Osztály munkatársai vennének használatba az új kari honlap elkészülte után. Amikor ennek az alkalmazásnak a szükségessége felmerült, még senki sem tudta, hogy az új kari honlap már nem a korábban is használt Liferay nevű tartalomkezelő rendszerre fog épülni, ezért belevágtam a fejlesztésébe. Az ötlet felmerülése után két évvel végre elkészült az új honlap, amelyhez már nem Java, hanem főleg PHP programozási nyelvet és egy általam nem ismert tartalomkezelő rendszert használtak a fejlesztői, ezért a továbbiakban

az általam készített alkalmazás a jelenlegi formájában már sosem lesz éles körülmények között használva.

Diplomamunkámban egy olyan Java EE technológiákra és a Liferay platforma épülő alkalmazás fejlesztését mutatom be, amely a fent említett sillabusz karbantartó megvalósítási lépéseit tartalmazza. A sillabuszra [5] nem létezik pontos definíció, de ha rákeresünk az interneten, akkor olyan szavakat, fordításokat és szinonimákat kaphatunk találatként, mint például "kivonat", "vázlat", "összefoglaló" vagy netalántán "útmutató". Általában az egyetemen sillabusznak neveznek minden olyan dokumentumot, amely egy adott tantárgyról tartalmaz kivonatolt, összegző információkat. Kivonatolt információnak tekinthető például egy tantárgy kódja, előfeltételei, a tantárgy teljesítéséhez szükséges követelmények, illetve az ajánlott irodalom is ilyen típusú információnak számít. Természetesen az előbbi felsorolás egyáltalán nem teljes, hogy egészen pontosan mi kerülhet bele és minek kell belekerülnie egy sillabuszba, az mindig az adott egyetemi kartól, az adott tanszéktől illetve az adott tantárgy oktatójától vagy oktatóitól függ. A diplomamunkámhoz készített program szükségességét éppen ez a differenciáltság indokolta, ugyanis az egyetem vezetősége szerette volna elérni, hogy mindenki, aki valamilyen formában az egyetemen oktat, azonos felépítésű sillabuszt készítsen, azaz arra volt szükség, hogy minden sillabusz hasonlóképpen nézzen ki, ugyan azon részegységeket és fejezeteket tartalmazza a könnyebb áttekinthetőség és a szabályzatoknak való megfelelés kedvéért.

A diplomamunkám további fejezeti arról szólnak, hogy hogyan fejlesztettem ki egy, a Java EE technológiákra és a Liferay platformra épülő alkalmazást, milyen problémákba ütköztem, hogyan oldottam meg ezen problémákat és milyen tapasztalatokat szereztem a program készítése közben.

2. fejezet

Követelmények meghatározása

2.1. A probléma megfogalmazása

Minden szoftverfejlesztési folyamat a követelmények meghatározásával, a probléma vagy másképpen nevezve a feladat megfogalmazásával kezdődik. Mivel minden program azért jön létre, hogy egy előre meghatározott problémát vagy feladatot oldjon meg, ezért én is egy megoldandó probléma megfogalmazásával kezdem.

Jelen esetben, az egyetem problémája, ahogyan azt már a bevezetésben is leírtam, nem más, mint az, hogy egységes megjelenítést biztosítson az Informatikai Kar által meghirdetett tantárgyakhoz tartozó sillabuszok számára. Ha nincs egységes felület, amelyet kitöltés után egy számítógép érvényesítene, akkor mindenki tetszés szerint, a saját maga számára megfelelő módon törölhetne ki és adhatna hozzá új fejezeteket és részegységeket egy-egy sillabuszhoz.

Természetesen tisztában vagyok vele, hogy a jelenlegi problémára számtalan megoldás létezik, mégis mindenki úgy gondolta, hogy a leghatékonyabb módon egy számítógépes program létrehozásával lehetne megoldani az egyetem sillabuszokkal való problémáit. A fejezetben található további alfejezetek a programmal támasztott követelményeket tartalmazzák rövidebb formában. A diplomamunkán belül azért nem jelenik meg a teljes specifikáció funkcionális és nem funkcionális követelményekre lebontva, mert maga a teljes specifikáció több oldalt venne igénybe, mint amennyi elvárható az egész diplomamunkától.

2.2. Webalkalmazás

Tekintettel arra, hogy a megoldandó probléma még akkor vetődött fel, amikor a kari honlap még Liferay 6.1-re épült és tervbe volt véve a 6.2-es verzióra történő migrálás, illetve a mai napig bevett szokás az egyetemen, hogy az oktatók a hallgatók számára minden sillabuszt elérhetővé tesznek a kari honalapon keresztül, ezért senki számára nem volt kérdés, hogy a programot szintén a kari honlapon kellene elhelyezni, tehát egy új webalkalmazásra lenne szükség. Az egyetlen kérdés az volt, hogy mit kell tudnia és hogyan kell működnie annak a programnak, amely a sillabuszhoz tartozó adatok megadását, megjelenítését és karbantartását teszi lehetővé.

Szinte fel sem merült annak a lehetősége, hogy asztali, azaz nem webalkalmazás készüljön, ugyanis ezzel számos probléma adódott volna.

Egy asztali alkalmazás során állandó problémát jelent, hogy mely platformok által támogatott (Windows, Linux, macOS, esetleg mobil platformok), illetve az is kérdéses, hogy a felhasználók frissítik-e, naprakészen tartják-e az alkalmazást a saját eszközükön vagy sem. Egy webalkalmazás esetén sosem lehet ilyen problémákkal találkozni, ugyanis egy weboldal megnyitásakor a felhasználónak nem kell frissítésekkel törődnie, mert ezt általában az adott weboldal üzemeltetői és fejlesztői maguktól szokták elvégzik egy olyan időpontban, amikor a lehető legkevesebben használják a rendszert, illetve egy ilyen típusú alkalmazás minden olyan platformon elérhető, ahol van a kornak és az oldalnak támogatottságának megfelelő webböngésző.

Természetesen azzal, hogy asztali alkalmazás helyett webalkalmazás készül nem oldódik meg minden szoftverfejlesztési probléma. Egy webalkalmazásnak számos előnye van egy asztali alkalmazással szemben, de ugyan ennyi ellenérvet is lehetne ellene felhozni, éppen ezért egy másik fejezetben összeszedtem mindazon problémákat, amelyekkel a sillabusz karbantartó alkalmazás fejlesztése során én is találkoztam.

2.3. Portlet - JSR 286

A korábbiakban már számtalanszor említést tettem a Liferay-re, itt az idő, hogy tisztázzam mit is értek alatta. A Liferay nem más [1], mint egy nyílt forráskódú, Java EE technológiákra épülő portál és tartalomkezelő rendszer. Portálnak azért nevezhető, mert számos alkalmazást, jelen esetben portletet integráltak össze benne, tartalomkezelő rendszernek pedig azért lehet nevezni, mert támogatja a felhasználók által létrehozott tartalmak dinamikus módon történő kezelését.

Számomra a Liferay az egyik legjobban testre szabható és funkcionálisan a legkönnyebben

bővíthető tartalomkezelő rendszer, amellyel valaha is találkoztam és amelyre valódi alkalmazást is fejlesztettem. A portál funkcionalitásának bővítésére az egyik legjobb lehetőség a portletek fejlesztése. A portlet [4] megfelel a JSR 286-nak, azaz a Java Portlet Specification 2.0-nak.

A sillabuszkezelő alkalmazás fejlesztése során törekednem kellett a Liferay platformmal történő szoros integrációra annak érdekében, hogy az egyetem teljes mértékben ki tudja használni a portál adottságait. Ezekből már egyértelműen lehet következtetni arra, hogy a feladatom egy Liferay kompatibilis portlet létrehozása volt.

2.4. Űrlap

Egy sillabusz információkat tárol egy tantárgyról. Ezen információk egy része félévente változik, másik része csak a mintatantervek megújításával kerül megváltoztatásra. Az egyszerű és könnyű használhatóság érdekében szükség van egy olyan űrlapra, amelyen a sillabuszhoz tartozó összes félévente változó adatot meg lehet adni. Mivel a program egy webalkalmazáson belül lesz használva, célszerű úgynevezett WYSIWYG[1] szerkesztőket használni azokon a helyeken, ahol hosszabb, több soros szöveg megadására is szükség van. Ez lehetőséget biztosít a végfelhasználók számára arra, hogy HTML kód szerkesztése nélkül rakjanak össze formázott HTML szövegeket, így azok a felhasználók is használatba tudják venni, akik nem rendelkeznek mélyebb informatikai ismeretekkel.

Az űrlap kitöltése után a programnak tudnia kell ellenőrizni a szükséges megszorításokat a létrehozott sillabusszal kapcsolatban és ha a kitöltött adatok nem felelnek meg a rendszernek, akkor jeleznie kell a létrehozó számára a hibát. Jelenleg elég ha csak a kötelező mezők és a helyes beviteli formátumokat ellenőrzi a rendszer, de később elképzelhető, hogy egyéb megszorításoknak is eleget kell majd tenni.

2.5. Adminisztrációs felület

A sillabuszok kezeléséhez szükség van egy olyan felületre, ahol a megfelelő jogosultságokkal rendelkező felhasználók meg tudják adni a sillabuszok alapadatait. Még pontosabban meghatározva egy olyan felülre van szükség, ahol a Tanulmányi Osztály dolgozói vagy megfelelő hatáskörrel rendelkező más személyek fel tudják tölteni vagy meg tudják adni a tantárgyak alapadatait. Ezen alapadatokat, magát a sillabuszokat létrehozó oktatók nem tudják módosí-

[1] WYSIWYG - What you see is what you get. - Amit látsz, azt kapod.

tani, ők csak tantárgyat tudnak választani a sillabusz hozzáadása során, amelynek hatására a megfelelő alapadatok automatikusan bekerülnek a sillabuszba.

2.6. Nyelvesítés

Egy olyan intézmény, mint az egyetem, nem engedheti meg magának, hogy csak magyar nyelven tegye elérhetővé adatait a honlapján keresztül, ezért fontos volt, hogy minden alkalmazás amely az új kari honlaphoz készül, egyaránt támogassa az angol és magyar nyelvű lokalizációt. Az általam készített alkalmazásnak csak akkor van értelme, ha az összes tantárgy esetén ezt használják a sillabuszok kezelésére. Ha a külföldi, magyar nyelvet nem értő hallgatóknak más módszer szerint készülne el egy sillabusz, akkor az alkalmazás teljesen hasztalan lenne, mert akkor ismét azon problémákba lehetne belefutni, mint amelyek ennek az alkalmazásnak a létrehozását is indukálták.

2.7. Exportálás, Importálás

Jelenleg az Informatikai Karon félévente több száz tantárgy kerül meghirdetésre, amelyekhez minden félévben, az adott tantárgy oktatóinak sillabuszt kell készíteniük. Mivel már adott a Neptun-nak nevezett tanulmányi rendszer, amely már tartalmazza az összes tantárgy alapadatait, ezért senki sem szeretne azzal foglalkozni, hogy kézzel átvezesse és konzisztensen tartsa a sillabusz kezelőben található tantárgyak alapadatait, ezért szükség van egyfajta automatizmusra ami a Neptun-ból kiexportált adatokat automatikusan, emberi beavatkozás nélkül be tudja importálni a sillabuszkezelő rendszerbe.

A sillabuszkezelőből történő adatok kiexportálására szintén szükség van. Ennek több oka is van. Egyrészt lehetőséget kell teremteni a hallgatók számára a sillabuszok PDF formátumban történő hozzáféréséhez, másrészt pedig, szükség van egy XML vagy CSV exportálási lehetőségre is, mert egy esetleges verziófrissítés miatt lehet, hogy rengeteg adatot kellene módosítani, ami igen sok időt vehet igénybe, ha nem számítógép végzi a módosításokat egy megfelelően formázott szöveges állományon. Az XML, illetve CSV formátumokban történő exportálásnak és importálásnak meg van az az előnye, hogy ha új attribútumokkal bővülnek az adatbázis táblák, akkor a bennük lévő entitások új attribútumait automatizált módszerrel lehet feltölteni egy három lépéses módszerrel:

1. Ki kell exportálni a módosítandó adatokat.

2. Módosítani kell a kiexportált adatokat.

3. Vissza kell tölteni a módosított adatokat.

2.8. Sillabusz duplikáció

Általában, ha egy tantárgyat ugyanazon oktató oktat több egymást követő évben vagy félévben, akkor szinte semmit, vagy csak alig szokott változni egy-egy sillabusz tartalma. Mivel a sillabuszok kezelése egy tartalomkezelő rendszerre lesz bízva, ezért célszerű lenne beépíteni egy olyan megoldást a rendszerbe, amely az aktuális félévben egy meghirdetett tantárgy esetén le tudja másolni az ugyanezen a tantárgyhoz tartozó, de egy korábbi félévben meghirdetett tantárgy sillabuszainak adatait. Ha mindez egy kattintással elérhető, akkor ezentúl olyan tantárgyakhoz is lehet sillabusz, amelyekhez évek óta nem készült egy sem, illetve nem kell felesleges időt eltölteni ugyanazon adatok ismételt begépelésével. Magától értetődik, hogy ennek a funkciónak a használata azt feltételezné, hogy valamikor az elmúlt években egy adott tantárgyhoz már létrehoztak egy sillabuszt és a benne lévő adatok továbbra is helytállóak.

2.9. Jogosultságkezelés

Egy olyan webalkalmazás esetén, amely felhasználók által bevitt adatokat tartalmaz, a jogosultságkezelés szinte elkerülhetetlen része a fejlesztésnek. Ha jobban belegondolunk, akkor csakis a sillabuszkezelő rendszer esetén három különböző szerepkörű felhasználót lehet megkülönböztetni:

1. *Adminisztrátor:* kizárólag tantárgyi alapadatokat kezelhet

2. *Oktató:* kizárólag sillabuszokat hozhat létre

3. *Hallgató:* kizárólag sillabuszokat tekinthet meg

Szinte biztosan kijelenthető, hogy egy élesített weboldal esetén ez a három szerepkör nagyon kevés lenne. Vannak hallgatók, akik demonstrátorként dolgoznak, ezért nekik is lehetőséget kellene adni sillabuszok létrehozására, továbbá az oktatóktól meg kell tiltani azt a lehetőséget, hogy mások által létrehozott sillabuszokat módosítsanak, de a tantárgyfelelősnek mindig minden joggal rendelkeznie kell egy tantárgy esetén.

Ezt a fajta jogosultságbeli szemcsézettséget lehetne még tovább fokozni, de fejlesztői szemmel nézve nincs értelme elgondolkozni azon, hogy milyen szerepkörökre lesz szükség az alkalmazás élesítése közben, sőt, személy szerint nem is tudhatom, hogy melyik felhasználóhoz milyen jogosultságokat kellene rendelni, ezért elég csak azzal foglalkoznom, hogy valamilyen lehetőséget biztosítsak az oldal üzemeltetőinek és adminisztrátorainak arra, hogy a felhasználói szerepköröket a megfelelő jogosultságokkal saját maguk tetszés szerint alakítsák ki a megfelelő szemcsézettséggel együtt. A lényeg tehát az, hogy lehetőséget kell teremteni a megjeleníthető adatok hozzáférhetőségének korlátozására, illetve a hozzáadási és módosítási jogok megfelelő szintű, futási időben tetszőleges módon történő beállítására.

Jogosultságkezelésre az egyik legjobb példa a tanszékvezető-tantárgyfelelős-oktató hármas között fennálló viszony. Egy oktató létrehozhat egy sillabuszt, ha ő az adott tantárgy oktatója. Ugyanezt a sillabuszt a tantárgyfelelős és tanszékvezető szintén tudja módosítani, de a tantárgy oktatója már nem tudja módosítani a tantárgyfelelős és tanszékvezető által oktatott egyéb tantárgyakhoz tartozó sillabuszokat. Ez szintén igaz a többi oktatóra is: ha egy oktató nem az adott tantárgy oktatója és se nem tantárgyfelelős, se nem tanszékvezető, akkor semmilyen joggal nem rendelkezik ahhoz, hogy az adott sillabuszt módosíthassa.

2.10. Munkafolyamat

Ahhoz, hogy egy sillabusz elérhetővé váljon a hallgatók számára, végig kell mennie egy jóváhagyási folyamaton. Egy sillabuszt tipikusan minden félév első két hetében szokás elkészíteni és ez idő alatt is kell jóváhagyatni a megfelelő illetékes személyekkel. Az egyetemi holnapra csakis jóváhagyás után kerülhetnek ki a sillabuszok, ezért a sillabusz kezelő alkalmazásnak valamilyen módon támogatnia kellene ezt a jóváhagyási folyamatot.

Tekintettel arra, hogy egy Liferay portlet-ről van szó, így adja magát az ötlet, hogy a Liferay saját munkafolyamat kezelési keretrendszerével kellene összeintegrálni az alkalmazást. Ennek a beépített folyamatnak a legnagyobb előnye az, hogy az üzemeltetők által szabadon konfigurálható, így megadja számukra azt a szabadságot, hogy úgy állítsák be, ahogyan azt az egyetem vezetősége megköveteli. Ha valamilyen oknál fogva nem lenne szükség a jóváhagyási folyamatra, akkor egész egyszerűen ki lehet kapcsolni a portlet-hez tartozó munkafolyamatot, ha pedig szigorítani kellene rajta, akkor is elég csak a beállításokon változtatni.

A Liferay munkafolyamat kezelést támogató keretrendszere nemcsak azért jó, mert szabadon konfigurálható, hanem azért is, mert egyszerű használni, teljesen mértékben nyomon

követhető, hogy ki, mit és mikor hagyott jóvá, esetleg utasított el. Minden egyes jóváhagyási folyamat során a felelős személyek véleményt is írhatnak, így a kommunikációt is megkönnyíti az egyes felek között.

2.11. Asset Framework integráció

A portletek után a Liferay legjobb és egyben leghasznosabb funkcióit az "Asset Framework", nyers fordításban az "Eszköz Keretrendszer" nyújtja. Ez a keretrendszer teszi lehetővé a Service Builder által létrehozott egyedek tetszőleges módon történő kezelését. A Service Builder-ről és annak működéséről egy későbbi fejezetben ejtek néhány szót, egyelőre elég annyit tudni róla, hogy egyedek definiálására lehet használni.

Az Asset Publisher-nek nevezett portlet az Asset Framework szerves részét képezi, amely az egyedek tetszés szerinti megjelenítéséért felelős. A megjelenítő teljes mértékben testre szabható, egy oldalon akár többet is el lehet helyezni belőle. A rengeteg beállítási lehetőségei közül csak néhányat sorolnék fel:

1. Egyedek lista szerű megjelenítése
2. Szűrési feltételek használata
3. Korlátozható hozzáférési jogosultságok
4. Import/Export lehetőségek különböző formátumokban
5. Közösségi szolgáltatások integrációja
6. Egyedek kategorizálhatósága, címkézhetősége

A sillabusz karbantartó alkalmazást célszerű úgy kialakítani, hogy a sillabuszok létrehozása, módosítása, megjelenítése az Asset Publisher segítségével is megtörténhessen, így nem lesz szükség saját megjelenítő portlet készítésére, elég csak egy az Asset Publish-ert konfigurálni a megfelelő oldalakon.

2.12. Webszolgáltatás

Az eredeti elképzelés szerint szükség van egy olyan webszolgáltatásra, amely lehetővé teszi az alkalmazás funkcióinak más rendszerekkel történő integrációját. A Liferay szinte minden

portlet-hez nyújt saját REST interfészt, így célszerű a sillabusz kezelőhöz is egy ilyet kialakítani.

3. fejezet

A webalkalmazások kérdéskörei

3.1. Általánosan a kérdéskörökről

Ebben a fejezetben szeretném bemutatni, hogy milyen problémákkal, kérdésekkel lehet találkozni egy-egy webalkalmazás fejlesztése közben. Az alábbi alfejezetek között több problémával is szembesültem a sillabusz kezelő alkalmazás fejlesztése közben.

3.2. Komplex tudást igényel

Akármennyire is tűnik egyszerűnek egy webalkalmazás, a kifejlesztése hihetetlenül komplex tudást igényel. Ha valaki akár egyedül, akár csapatban egy rendes webalkalmazást szeretne kifejleszteni, akkor legalább az alábbi felsorolás mindegyik eleméhez kell, hogy a fejlesztők valamilyen szinten értsenek:

1. Adatbázisrendszerek
2. Alkalmazásszerverek
3. Üzleti logika
4. Webszolgáltatások
5. Megjelenítés

3.2.1. Adatbázisrendszerek

Egy webalkalmazás adatait jó esetben valamilyen adatbázisrendszer segítségével szokták tárolni. Ahhoz, hogy ez megtörténhessen, minden fejlesztőnek fel kell tudnia telepíteni az adatbázisrendszert és közösen ki kell tudniuk alakítani a megfelelő adatbázis sémát az alkalmazás számára. Az egyes létező adatbázisrendszerek között óriási különbség lehet még annak ellenére is, hogy szinte mindegyik relációs adatbázisrendszer támogatja az SQL lekérdező nyelvet. Egy esetleges adatbázisrendszer csere nagyon meg tudja bonyolítani a fejlesztők életét, ezért nem árt még a program tényleges fejlesztésének megkezdése előtt utánanézni a lehetőségeinknek, illetve az egyes rendszerek költségeinek.

3.2.2. Alkalmazásszerverek

Egy webalkalmazás futtatásához szinte minden esetben szükség van egy önálló alkalmazásszerverre is. Ha csak a Java EE alkalmazásszervereket vesszük figyelembe, még akkor is feltűnő, hogy rengeteg van belőlük, és mindegyik más-más tudással és konfigurálási lehetőségekkel rendelkezik. Egy ideális világban egy elkészült webalkalmazás tetszőleges szerveren képes futni, de a gyakorlatban ez közel sem így működik, éppen ezért célszerű egy program fejlesztésének elkezdése előtt tisztázni azt is, hogy mely alkalmazásszerveren fog futni az adott program. A választás során mérlegelni kell, hogy kinek milyen tapasztalata van és melyik szerver lenne a legideálisabb a készülő alkalmazás számára.

A Liferay a 7.0 verzió kiadása óta hivatalosan csak az Apache Tomcat-et és a WildFly szervereket támogatja, de ez nem jelenti azt, hogy a többin ne lehetne működésre bírni. A sillabusz kezelő alkalmazás fejlesztés közben az Apache Tomcat-et használtam, mert észrevételeim szerint ez jóval gyorsabban reagált ugyan azon kérésekre, mint a WildFly.

3.2.3. Üzleti logika

Egy alkalmazás üzleti logikájának kialakítása több dolgot is igényel. Egyrészt a fejlesztőnek meg kell értenie az üzleti folyamatokat, másrészt ezt meg is kell tudnia valósítania. Ha valaki a két dolog közül csak az egyikhez ért, akkor az alkalmazás vagy rossz minőségű lesz vagy nem azt fogja csinálni, mint amit a megrendelő vár a terméktől.

3.2.4. Webszolgáltatások

Manapság már nagyon kevés olyan rendszer létezik, amely ne lenne összeintegrálva más rendszerekkel. Ahhoz, hogy két alkalmazás együtt tudjon működni, szükség van valamilyen kapcsolódási pontra, interfészre, amelyen keresztül meg tudják szólítani egymást, vagyis kommunikálni tudnak egymással. A webszolgáltatások egy ilyen interfészt nyújtanak a külvilág számára HTTP protokollon keresztül.

3.2.5. Megjelenítés

Egy alkalmazás felhasználó felületének kialakítása először egyszerűnek tűnik, de amint elkezd foglalkozni vele valaki, akkor nagyon hamar rengeteg megválaszolatlan kérdéssel és technikai nehézséggel találhatja szembe magát. Nem véletlen, hogy Ian Sommerville - Szoftverrendszerek fejlesztése című [2] könyvében egy teljes fejezetet szánt a felhasználó felületek tervezési kérdéseire. Ha valaki jó felületet szeretne kialakítani, akkor nem árt ha tisztában van a könyvben szereplő elvekkel és kérdésekkel.

3.3. Tartalomkezelő rendszerek

Minden alkalmazás esetén el kell gondolkozni azon, hogy a teljes alkalmazást saját magunk alakítjuk ki, vagy a funkcionalitás egy részét már meglévő szoftverekre bízzuk. Ugyan ezt kell mérlegelni a webalkalmazások esetén is. Ha több olyan funkciót is szeretnénk az oldalon biztosítani, amelyeket mások már jóval előttünk megírtak, akkor célszerű egy, a feladatnak megfelelő tartalomkezelő rendszer köré felépíteni a saját alkalmazásunkat.

Talán nem túlzás azt kijelenteni, hogy tartalomkezelő rendszerből még több található meg a piacon, mint alkalmazásszerverből összesen. Itt is igaz, hogy ahány tartalomkezelő rendszer létezik, annyiféle megoldás is van az egyes problémákra, éppen ezért nehéz megtalálni a legideálisabb rendszert, mert mindegyik számos előnnyel és legalább ugyan ennyi hátránnyal is rendelkezik. Nekem viszonylag egyszerű volt a dolgom, ugyanis az egyetem már jóval azelőtt elkötelezte magát a Liferay mellett, hogy elkezdtem volna a sillabusz kezelő alkalmazás fejlesztését.

Számomra a legnehezebb dolog a Liferay által biztosított fejlesztői eszközök megismerése, a teljes rendszer kiismerése, majd pediglen az eszközök a gyakorlatban történő alkalmazásuk okozta a legnagyobb kihívást, mert korábban még sosem fejlesztettem egyetlen egy tartalomke-

zelő rendszerre sem semmilyen alkalmazást, éppen ezért jóval több időt vett igénybe a Liferay-re való fejlesztési módszerek elsajátítása, mint amennyi egy tapasztaltabb fejlesztő számára lett volna szükséges.

A tartalomkezelő rendszerek közötti különbségek szemléltetésére igen csak jó példa az OpenCms és Liferay rövid összehasonlítása. Néhány hónapja részt vehettem egy projektben, ahol egy teljes alkalmazást az OpenCms nevű alkalmazásszerver köré kellett építenünk. A kettő közötti különbség feltűnően nagy volt. Tapasztalataim szerint az előbbi funkcionalitásban, utóbbi pedig teljesítményben volt jobb a másiknál. A teljesítménykülönbség a kérések kiszolgálásánál másodpercekben, az elindításaikhoz szükséges idő közötti különbség pedig percekben mérhető. Ez a különbség a Liferay testre szabhatósága és gazdag funkcionalitása miatt volt ennyire szembetűnő. Tapasztalataim és mások véleménye szerint is, a Liferay képes volt összerakni egy funkciókban gazdag, a kornak megfelelő tartalomkezelő rendszert, de mindez a teljesítmény rovására ment.

3.4. Állandó változás

A mai világban szinte semmi sem változik olyan gyorsan, mint az informatikai rendszerek. Az internet robbanásszerű elterjedése alapjaiban változtatta meg az emberek életét. Az internet hajnalán a webet olvasott webnek nevezték, de a technológia fejlődésnek köszönhetően igen hamar eljutottunk az írott-olvasott webhez, amelyre web 2.0-ként szokás hivatkozni. A web 2.0 megjelenésével egy időben számos cég fogott tartalomkezelő rendszer fejlesztésébe, köztük a Liferay első verziója is ekkora tehető.

Az elmúlt 17 évben a Liferay is óriási változásokon ment keresztül. A számomra is érzékelhető legnagyobb változások a 6.2 és a 7.0 verziók közötti különbségekben mutatkoznak meg. Eredetileg a sillabusz kezelő alkalmazás Liferay 6.2-re lett tervezve és a fejlesztést is ezen kezdtem el, mert akkor még távol volt az új 7.0 verzió megjelenése. Közel egy évnyi fejlesztés után megjelent az új főverzió a portálból. A portlet migrálása és az új portállal történő kompatibilitás kialakítása majdnem egy teljes hónapot vett igénybe még egy ilyen kis méretű alkalmazás esetén is. Számos új fejlesztői eszköz használatát kellett megtanulnom, köztük a Gradle és a Blade CLI programok használatát is, mert korábban csak az Apache Ant és az Apache Maven volt támogatott a Liferay által, mint a fordítási folyamatot támogató eszközök. Az eszközök használata mellett a Liferay SDK is alapvetően megváltozott, nem is beszélve a rengeteg egyik napról a másikra *Deprecated*, azaz elavult API-ra.

3.5. Böngésző inkompatibilitás

Nem létezik olyan webfejlesztő aki ne találkozott volna kompatibilitási problémákkal. Néhány éve, amikor az Internet Explorer még a fénykorát élte, rosszabb volt a helyzet, mint amilyen manapság szokott lenni. Szinte biztos, hogy ami az egyik böngészőben jól jelenik meg, az egy másikban valahol, valamilyen eltérést fog mutatni. Megszámolni sem tudom, hogy hány különböző típusú böngésző létezik. Szerencsére általában elég csak a legelterjedtebb böngészőkre optimalizálni egy oldalt, a látogatók nagy része úgy is csak ezeket használja, a többi böngésző többségét pedig ugyan azon motorok hajtják, melyek a legelterjedtebbeket is.

Az alkalmazás fejlesztése közben számtalanszor találkoztam kompatibilitási problémákkal. Az egyik ilyen hiba a JavaScript-hez és az Internet Explorer-hez volt köthető. Nem is olyan régen egy oldalon képfeltöltés funkciót próbáltam bevezetni, majd mikor azt láttam, hogy Google Chrome és Mozilla Firefox alatt a funkció hibátlanul működik, akkor kipróbáltam Internet Explorer 11 alatt is. Egy viszonylag kis méretű teszt kép feltöltése során szinte azonnal összeomlott a teljes böngésző és az egészet újra kellett indítani. Nem sokkal később kiderült, hogy az Internet Explorer nem rendelkezik egy általam használt JavaScript függvény implementációjával és erre a teljes böngésző összeomlásával reagált. A hibát úgy sikerült kijavítanom, hogy külön feltételben vizsgáltam meg a böngésző típusát és ha az elágazásnál Internet Explorer-t érzékel a program, akkor más módszert használ a képfeltöltés funkcióra, mint amilyet a többi böngésző használ.

A sillabusz kezelő alkalmazás fejlesztése közben is találkoztam hasonló hibával, szerencsére itt nem kellett az Internet Explorer kompatibilitással törődnöm, elég volt csak a Google Chrome-ra és a Mozilla Firefox-ra optimalizálnom. Az egyik ilyen hiba az egymástól függő legördülő menüknél volt megfigyelhető. Valamilyen oknál fogva Chrome esetén ha kiválasztottam egy elemet az első legördülő listában és nem tartozott hozzá elem a második listában, akkor a második lista tartalma módosult a DOM-on belül, de a böngésző rosszul jelenítette meg, továbbra is a régi elemeket listázta ki. A hiba pontos okát sosem sikerült kideríteném, egyszer csak egy Chrome frissítés után elkezdett jól működni, így nem fordítottam különösebb figyelmet rá. Ha a hiba nem szűnt volna meg magától, akkor igen nehéz dolgom lett volna a hibajavítás során, ugyanis két JavaScript keretrendszert is használtam, az egyik a jQuery a másik pedig a Liferay által is erőszeretettel használt Alloy UI volt.

3.6. Biztonsági kérdések

Egy weboldal esetén mindig kulcsfontosságú kérdés a biztonságosság. Szinte minden évben lehet olyan hírekről hallani, amelyek arról szólnak, hogy több millió személyes adatot loptak el vagy csaltak ki egy oldal felhasználóitól. Jó példa erre a 2011-ben történt Playstation hálózat feltörése vagy a legutoljára nagy port kavaró veszprémi pizzéria weboldala, ahol nem is a weboldalt törték fel, hanem az alkalmazást futtató szervert.

Számtalan oka lehet annak, hogy hogyan lehet illetéktelen adatokhoz jutni, most csak néhányat sorolnék fel ezen lehetőségek közül:

1. Nem biztonságos a webalkalmazás
2. Nem biztonságos a kiszolgáló
3. Adathalászattal kicsalt személyes adatok
4. Social engineering
5. Stb...

Ennek a diplomamunkának egyáltalán nem célja az összes létező ok és támadási módszer bemutatása, mégis fontosnak tartom, hogy említést tegyek a biztonsági kérdésekről, mert akárki akármit is gondol, ez mindig elengedhetetlen része az összes alkalmazásnak. Az egyetemen sem örülne senki annak, ha a hallgatók tetszés szerint egy biztonsági hibát kihasználva illetéktelenül módosítanák a sillabuszok adatait, ezért nekem, mint a sillabusz kezelő alkalmazás fejlesztőjének is gondosan ügyelnem kellett, hogy a lehető legbiztonságosabb alkalmazást hozzam létre.

A jogosultságkezelés szorosan összefügg a biztonságosság kérdéskörével. Jogosultságkezeléssel meg lehet szabni, hogy ki, mikor és milyen adatokhoz férhet hozzá. A Liferay esetén erre egy külön alrendszer ad lehetőséget, a fejlesztőknek mindössze annyi a dolguk, hogy saját szabályokat definiáljanak és az oldal megfelelő részein az alrendszer megszólításával ellenőrizzék, hogy a felhasználó rendelkezik-e a megfelelő jogokkal.

3.7. Megjelenítés

Mindenki számára egyértelmű, hogy ha a felhasználói felület rosszul van megtervezve, akkor a felhasználók nem fogják használatba venni az oldalt. Manapság elengedhetetlen, hogy

nem csak funkcionalitásban, hanem megjelenítésben is megfelelő legyen egy-egy webalkalmazás. Rossz felhasználói felületre rengeteg példa létezik, mégis szerintem az egyik legjobb példa a nemrég bevezetett Ügyfélkapu. Rengeteg funkciója van, de az ember többször is elgondolkozik azon, hogy tényleg használja-e vagy inkább elmenjen a legközelebbi Önkormányzati Hivatalba az egyszerűbb ügyintézés érdekében.

Az okos eszközök széles körben történő elterjedése óta állandó követelmény, hogy minden weboldal legyen megtekinthető kis méretű eszközökön is. Az okostelefonok közel tíz éve robbantak be a köztudatba, azóta folyamatosan fejlődtek, velük együtt a rajtuk futó operációs rendszerek is hatalmas változásokon mentek keresztül, éppen ezért nem meglepő, hogy a StatCounter [8] weboldal szerint 2017 márciusában az Android operációs rendszert futtató eszközök száma meghaladta a Windowst futtató számítógépek számát.

Egyetlen egy webalkalmazás fejlesztése közben sem lehet szó nélkül elmenni az adaptív vagy reszponzív megjelenítés kérdése mellett. Adaptívnak akkor nevezhető az oldal, ha a különböző típusú és méretű eszközök más és más oldalhoz jutnak hozzá. A gyakorlatban ez úgy lett megvalósítva, hogy a HTTP kérések User-Agent fejléce alapján a kiszolgáló el tudja dönteni, hogy mely típusú tartalom továbbítására van szükség, azaz például, ha egy mobilról érkezik a HTTP kérés, akkor a kiszolgáló a mobilos tartalmat, ha pediglen asztali operációs rendszeren futó böngészőtől érkezik be a kérés, akkor meg a neki megfelelő változatot küldi vissza a kiszolgáló. Weboldalak tervezésekor egy másik lehetőség a reszponzív oldalak kialakítása. A reszponzív megjelenítés teljesen szembe megy az adaptív megjelenítés fogalmával. Reszponzív oldalak esetén minden eszköz, a küldött felhasználói ágenstől függetlenül ugyan azt a tartalmat kapja meg válaszként, cserébe az oldalon megjelenített elemek elrendezése csakis kizárólag az oldalt megjelenítő böngésző szélességétől, magasságától, illetve a stíluslapoktól függ.

A reszponzív megjelenítés magával vonja a *mobile first* szemléletmódot, amely mindössze annyit jelent, hogy az oldalt kinézetét először mobilra készítjük el és csak ezután kezdünk el nagyobb méretű kijelzőkre optimalizálni. Sajnos a fejlesztők egy része nem követi ezt a szemléletmódot, ezért igen gyakori, hogy a reszponzívnak nevezett oldalak mobilon teljesen használhatatlanok. Ezt személy szerint én is átéltem, mert nem is olyan régen én is részt vettem egyszer egy olyan projektben, ahol a tervező nem volt tisztában ezzel a szemléletmóddal, ezért hiába mondta azt, hogy az oldal reszponzív, mert mobilon teljesen széthullott az oldal kinézete és nekem kellett több hetet eltölteni ilyen a hibák javításával.

3.8. Keresőoptimalizálás

A keresőoptimalizálás egy igen fontos és egyben nagyon összetett területe a webfejlesztésnek, akár egy teljes diplomamunkát is lehetne róla készíteni, éppen ezért csak néhány megjegyzést tennék róla.

Keresőoptimalizálásra azért van szükség, mert ha valaki nem fizet a keresőszolgáltatóknak a találati listában történő megjelenítésért, akkor ezek a szolgáltatók valamilyen módszer segítségével pontokat rendelnek az egyes oldalakhoz és ezen pontok alapján rangsort állítanak fel az egyes oldalak között. Minél jobbal van rangsorolva egy oldal, annál többször lesz megtalálható egy-egy kulcsszavas keresés esetén a találati listában. Ahhoz, hogy egy weboldal magasan rangsorolt legyen, szükség van néhány szabály és irányelv betartására. Ezen szabályoknak és irányelveknek a keresőszolgáltatóknál lehet utánanézni. Kiindulásként a *Google Keresőmotor-optimalizálási útmutató kezdőknek* című dokumentumot lehet használni, amely bármikor ingyenesen letölthető a Google keresőoptimalizálással foglalkozó oldalairól.

4. fejezet

A megvalósítás lépései

4.1. Szójegyzék

Mielőtt belemennék bármilyen konkrét dologba, fontosnak tartom, hogy a későbbi fejezetekben megjelenő leggyakoribb szavakat előre bevezessem magyar és angol nyelven egyaránt:

Mintatanterv	Curriculum
Tantárgy	Subject
Kurzus Típus	Course Type
Kurzus	Course
Félév	Semester
Órarendi Kurzus	Timetable Course
Oktató	Lecturer
Sillabusz	Syllabus

4.2. Projektstruktúra kialakítása

Liferay 7.0-tól kezdve minden projekt Gradle alapokra épül. A Gradle egy olyan ingyenes és nyílt forráskódú eszköz, amelyet a fordítási folyamatok egyszerűsítése érdekében hoztak létre, nagyban hasonlít az Apache Ant és az Apache Maven eszközökre. A sillabuszkezelő alkalmazás három projektből épül fel:

1. syllabus-manager-api
2. syllabus-manager-service

3. syllabus-manager-web

A három projektet a Gradle fogja össze és ez teszi lehetővé, hogy egyetlen parancs kiadásával mind a hármat egyszerre lehessen lefordítani. Ez a fajta projektfelépítés nagyban hasonlít az Apache Maven által használt többmodulos projektekhez.

A *syllabus-manager-api* és a *syllabus-manager-service* projektek egymással szorosan kapcsolatban állnak. A Liferay Service Builder által generált szolgáltatás interfészek, entitás modellek és egyéb segéd osztályok a syllabus-manager-api, ezek implementációi pedig a syllabus-manager-service projekten belül lettek elhelyezve. Ez a struktúra azért is jó, mert bárki bármikor megváltoztathatja a szolgáltatások működését anélkül, hogy az api-hoz hozzá kellene nyúlnia.

A *syllabus-manager-web* projekten belül található meg az összes megjelenítéssel összefüggő dolog. Ez az a projekt, amely tartalmazza az általam létrehozott portlet-et és függőségként használja a másik két projektet. Tehát az api és service projektek szolgáltatásokat nyújtanak, a web projekt pedig igénybe veszi ezen szolgáltatásokat.

4.3. Liferay Service Builder

Liferay esetén, ha új adatbázis táblákat szeretnénk létrehozni és a portál alatt futó adatbázist szeretnénk az adatok tárolására használni, akkor nincs más dolgunk, mint a Liferay Service Builder segítségével egyedeket, tulajdonságokat és kapcsolatokat definiálni. Mindez a megfelelő projektstruktúra kialakítása után a service.xml állomány létrehozásával, illetve annak módosításával tehető meg. A service.xml állományt kétféleképpen lehet módosítani, vagy a Liferay SDK által biztosított grafikus felület segítségével adjuk meg a megfelelő elemeket, vagy egy átlagos szöveges állományként szerkesztjük úgy, mint egy valódi XML dokumentumot. Bármelyik módszert is használjuk a végeredmény ugyan az lesz.

A service.xml elkészítése után a *gradlew buildService* parancs hatására lefut a Service Builder és minden szükséges interfészt és osztályt legenerál számunkra. A továbbiakban nincs más dolgunk, mint a szolgáltatások tényleges implementálása.

Maga a service.xml a syllabus-manager-service projekt főkönyvtárában található meg. A dev.liferay.com fejlesztőknek szóló oldala szerint egy ilyen service.xml állományt hét lépésben lehet lehet létrehozni, de én ezt nyolcra egészíteném ki:

1. A service.xml állomány létrehozása.

2. Globális információk megadása.

3. Entitások definiálása.

4. Tulajdonságok, attribútumok definiálása.

5. Kapcsolatok definiálása.

6. Rendezések definiálása.

7. Finder metódusok definiálása.

8. Kivételek definiálása.

A további alfejezetekben ezt a nyolc lépést magyarázom el példákkal együtt.

4.3.1. A service.xml állomány létrehozása

A service.xml állományt minden esetben a service projekt főkönyvtárában kell elhelyezni a következő tartalommal:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE service-builder PUBLIC
"-//Liferay//DTD Service Builder 7.0.0//EN"
"http://www.liferay.com/dtd/liferay-service-builder_7_0_0.dtd">

<service-builder package-path="hu.unideb.inf">
...
</service-builder>
```

A service-builder elem package-path attribútuma azt határozza meg, hogy mely csomagba kerüljenek a legenerált interfészek és osztályok.

4.3.2. Globális információk megadása

Globális információk alatt a szerző és a névtér nevét kell érteni. A szerző általában a fejlesztő cég neve szokott lenni, de mivel én egyedüli fejlesztő voltam, ezért a saját nevemet adtam meg. A névtér név azért fontos, mert az adatbázis táblák nevei előtt ez a karaktersorozat fog megjelenni, így elkerülhető az esetleges táblanév ütközés. Ezt a két információt közvetlenül a service-builder elem első leszármazottaiként kell megadni. A sillabuszkezelő alkalmazás esetén ez így néz ki:

```
<service-builder package-path="hu.unideb.inf">
        <author>Adam Kiss</author>
        <namespace>syllabus_manager</namespace>
        ...
</service-builder>
```

4.3.3. Entitások definiálása

Az entitások definiálása közvetlenül a namespace elem után lehetséges. A következő példában a Curriculum entitást definiáltam:

```
<entity name="Curriculum" local-service="true">
...
</entity>
```

4.3.4. Tulajdonságok, attribútumok definiálása

Egy entitásból adatbázisbeli tábla lesz, tulajdonságaiból pedig oszlopnevek. A Curriculum entitás attribútumainak definiálása az alábbi példában tekinthető meg:

```
<entity name="Curriculum" local-service="true">
        <column name="curriculumId" type="long"
                primary="true"></column>
        <column name="groupId" type="long"></column>
        <column name="companyId" type="long"></column>
        <column name="userId" type="long"></column>
        <column name="userName" type="String"></column>
        <column name="createDate" type="Date"></column>
        <column name="modifiedDate" type="Date"></column>
        <column name="curriculumCode" type="String"></column>
        <column name="curriculumName" type="String"></column>
        ...
</entity>
```

Fontos, hogy minden tulajdonság rendelkezzen valamilyen típussal, illetve nélkülözhetetlen az elsődleges kulcsok meghatározása, amelyet a primary="true"-val lehet megtenni.

4.3.5. Kapcsolatok definiálása

Az adatbázisban lévő egyedek más egyedekkel is kapcsolatban állhatnak. A kapcsolatok modellezését adatbázis szinten külső kulcsok segítségével lehet megoldani. Sajnos a Liferay már évek óta nem biztosít lehetőséget külső kulcsok tényleges definiálására, de több módszer is létezik arra, hogy két egyedet össze lehessen benne kapcsolni.

Az egyik módszer, amikor egy új attribútumot vezetünk be a meglévőek mellé, viszont az adatbázisbeli táblákra sajnos nem fognak rákerülni a külső kulcsmegszorítások, ezt kézzel kell majd a fejlesztőknek megtenniük. Jó példa erre a tantárgy-mintatanterv kapcsolat. Ebben az esetben egy tantárgy csak egy mintatantervhez, viszont egy mintatantervhez több tantárgy is tartozhat:

```
<entity name="Subject" local-service="true">
        ...
        <column name="curriculumId" type="long"></column>
        ...
</entity>
```

A másik módszer a kapcsolótábla létrehozása, amelyet M:N számosságú kapcsolatok ábrázolására lehet használni. Erre az egyetlen példa, amelyet a sillabuszkezelő alkalmazás során alkalmaztam az oktatók és órarendi kurzusok között fennálló viszony. Egy ilyen kurzusnak akármennyi oktatója lehet és egy oktató több kurzust is tarthat. A service.xml-en belül ez úgy jelenik meg, hogy a TimetableCourse entitásnál az alábbi:

```
<entity name="TimetableCourse" local-service="true">
        ...
        <column name="lecturers"
                type="Collection"
                mapping-table="Lecturers_TimetableCourses"
                entity="Lecturer">
        </column>
        ...
</entity>
```

a Lecturer egyednél pedig a következő kódrészlet jelenik meg:

```
<entity name="Lecturer" local-service="true">
        ...
        <column name="timetableCourses"
                type="Collection"
                mapping-table="Lecturers_TimetableCourses"
                entity="TimetableCourse">
        </column>
        ...
</entity>
```

Ennek hatására a Liferay tudni fogja, hogy egy újabb tábla létrehozására lesz szükség, melynek két oszlopa lesz. Az egyik oszlop timetableCourseId-ket, a másik pedig lecturerId-ket fog tartalmazni.

4.3.6. Rendezések definiálása

Rendezések definiálása alatt azt kell érteni, hogy a service.xml-en belül külön jelezni kell azon attribútumokat, amelyek alapján rendezve szeretnénk eredményeket visszakapni egy-egy adatbázisból történő lekérdezés esetén.

Az előző leírást egyszerűbb megérteni egy példa alapján:

```
<order by="asc">
        <order-column name="curriculumCode"
                order-by="asc"></order-column>
        <order-column name="curriculumName"
                order-by="asc"></order-column>
</order>
```

Az előbbi példakód feltételezi, hogy az entitás, amelynél rendezést szeretnénk használni, már rendelkezik a *curriculumCode* és a *curriculumName* nevű attribútumokkal. A példa azt mondja meg, hogy először a curriculumCode, majd azon belül a curriculumName attribútum szerint kell rendezni a lekérdezések eredményét. Olyan ez, mintha egy SQL SELECT esetén az

```
ORDER BY curriculumCode, curriculumName
```

rész is jelen lenne a lekérdezés során.

4.3.7. Finder metódusok definiálása

Java alkalmazások esetén azokat a metódusokat, amelyek közvetlenül az adatbázisból nyernek ki információkat, tipikusan a "find" metódusnév előtaggal szoktuk ellátni. Például:

```
public abstract Curriculum findByPrimaryKey(long id);

public abstract Curriculum findByCurriculumCode(
                String curriculumCode);

public abstract List<Curriculum> findAll();
```

A Service Builder használata esetén ezen finder metódusok automatikusan legenerálódnak, fejlesztőként mindössze csak annyit kell tudni megmondani, hogy mely entitás esetén, mely attribútumok alapján, milyen visszatérési értékekkel jöjjenek létre a szükséges metódusok. Magától értetődik, hogy ezt is a service.xml-en belül lehet megtenni.

Példaként tekintsük meg az alábbi kódot, amelyet a Subject, azaz tantárgy entitás esetén használtam:

```
<finder name="C_S" return-type="Subject" unique="true">
        <finder-column name="curriculumId"></finder-column>
        <finder-column name="subjectCode"></finder-column>
</finder>
<finder name="Curriculum" return-type="Collection">
        <finder-column name="curriculumId"></finder-column>
</finder>
```

A példában a tantárgyak entitáshoz két finder metódust definiáltam.

Az egyik a *curriculumId* és *subjectCode* attribútumok alapján tud lekérdezni egy tantárgyat. A finder elem *unique="true"* attribútumával azt határoztam meg, hogy tegyen unique megszorítást az adatbázisbeli Subject tábla két megfelelő oszlopára, így garantálva az egyedek tényleges egyediségét. Magyarán szólva, minden egyednek a Subject táblán belül egyedinek kell lennie a curriculumId és subjectCode attribútumokra nézve.

A második finder Subject entitásokat tartalmazó Java kollekciót tud visszaadni. A visszaadott kollekción belül minden egyed esetén a curriculumId meg fog egyezni a paraméterként átadott azonosítóval, azaz minden olyan tantárgy lekérdezésre kerül, amely az adott mintatantervhez tartozik.

4.3.8. Kivételek definiálása

Liferay esetén a kivételeket nem közvetlenül a java.lang csomagban található Exception és RuntimeException osztályokból szokás származtatni, hanem be kell építeni ezeket a Liferay saját kivételosztály hierarchiájába. Erre több módszer is létezik, a legegyszerűbb az, ha a service.xml végén felsoroljuk az általunk szükséges kivétel osztály neveket, hasonlóan, mint ahogyan az a következő példában is látható:

```
<exceptions>
        <exception>CurriculumCode</exception>
        <exception>CurriculumName</exception>
</exceptions>
```

Ennek hatására az alábbi kódrészlethez hasonló programkód fog legenerálódni:

```
import aQute.bnd.annotation.ProviderType;
import com.liferay.portal.kernel.exception.PortalException;

@ProviderType
public class CurriculumCodeException
                extends PortalException {
        ...
}
```

A kivételek ilyen módon történő definiálásának legnagyobb előnye a hibaüzenetekké történő átalakításban rejlik. A Liferay lehetőséget biztosít arra, hogy a kivételekhez szöveges üzenetet társítsunk, amelyeket felugró üzenetként tud megjeleníteni a felhasználók számára.

A konverzióhoz nem kell mást tenni, mint a liferay-ui:error elemet elhelyezni a megfelelő JSP-ben, illetve a megfelelő kulcs-érték párokat berakni a lokalizációs állományokba. Az alábbi példában az általam definiált CurriculumCodeException osztályt alakítom át szöveges üzenetekké:

```
<liferay-ui:error
        exception="<%=CurriculumCodeException.class%>"
        message="curriculum-code-exception" />
```

A példához tartozó angol nyelvű lokalizációs állomány egyik sora:

```
curriculum-code-exception=Invalid Curriculum Code
```

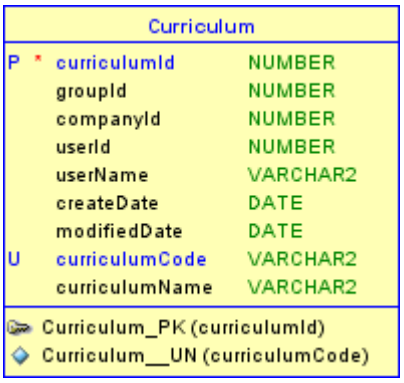
Tehát, ha a program futása közben CurriculumCodeException váltódik ki, akkor a megfelelő

helyen az "Invalid Curriculum Code" hibaüzenet jelenik meg.

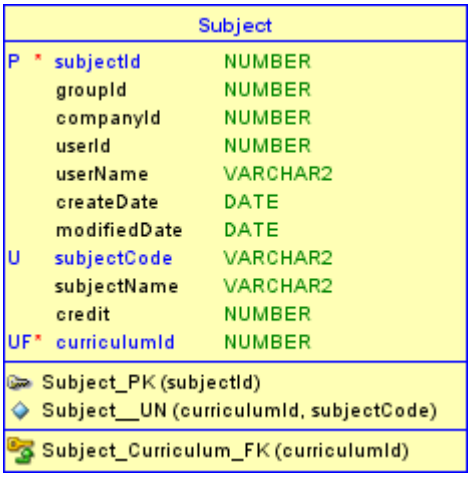
4.4. Modellezés

Ebben a fejezetben azt szeretném bemutatni, hogy milyen egyedeket hoztam létre a Liferay Service Builder segítségével a sillabusz kezelő alkalmazás számára. A teljes service.xml állomány túl nagy terjedelme miatt az Oracle SQL Developer Data Modeler alkalmazás segítségével készítettem belőle egy relációs modellt, ami egy az egyben megfelel a service.xml-ben leírtaknak. Sajnos a kiexportált diagram minden esetben vagy túl széles vagy pedig túl magas volt, hogy teljes egészében kiférjen egy oldalra, ezért úgy döntöttem, hogy minden egyedet külön ábrán helyezek el.

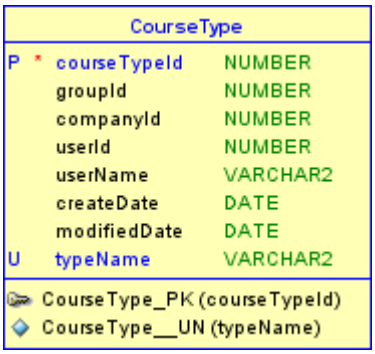
4.4.1. Mintatanterv



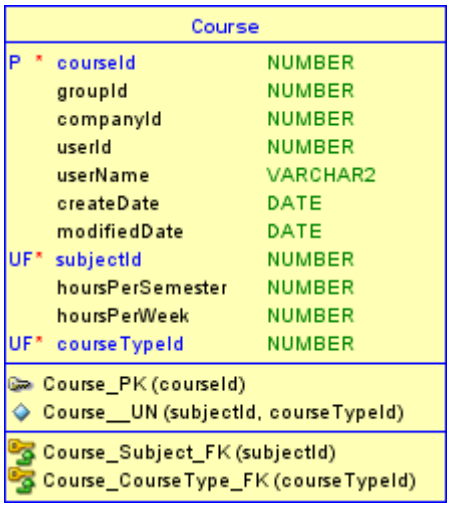
4.4.2. Tantárgy



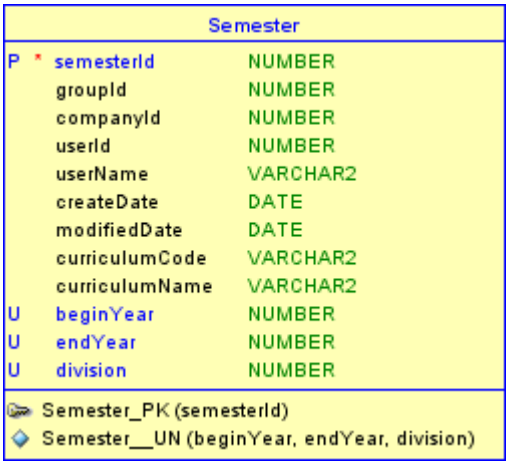
4.4.3. Kurzus Típus



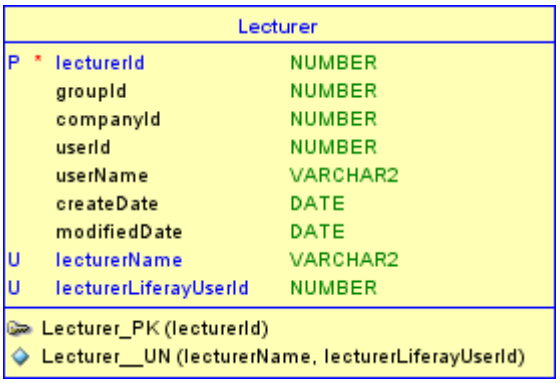
4.4.4. Kurzus



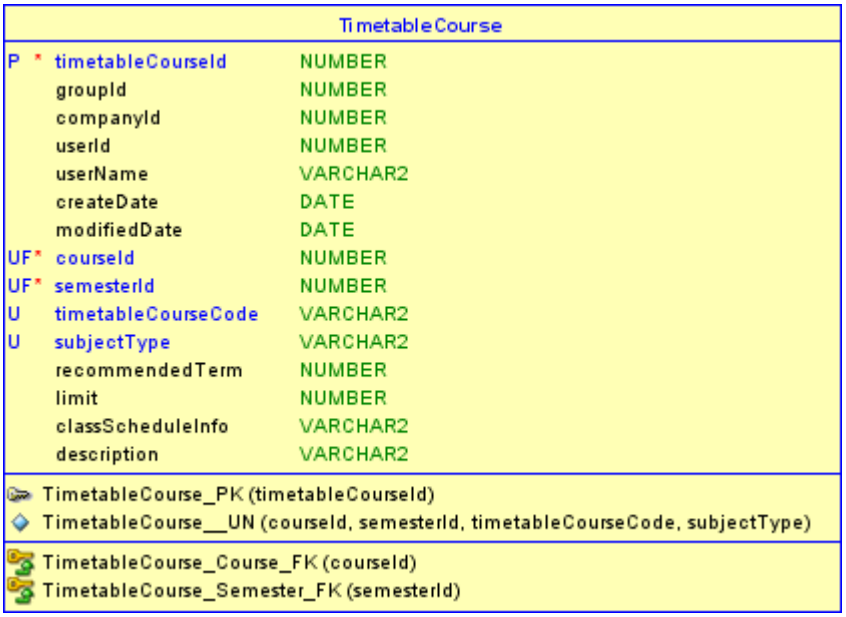
4.4.5. Félév



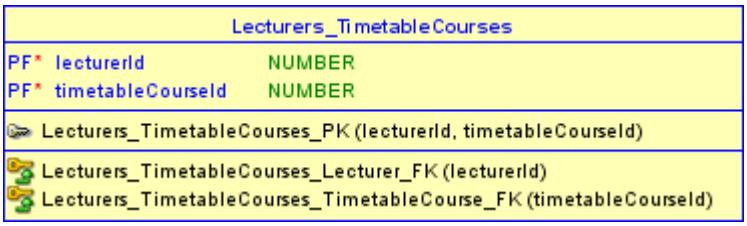
4.4.6. Oktató



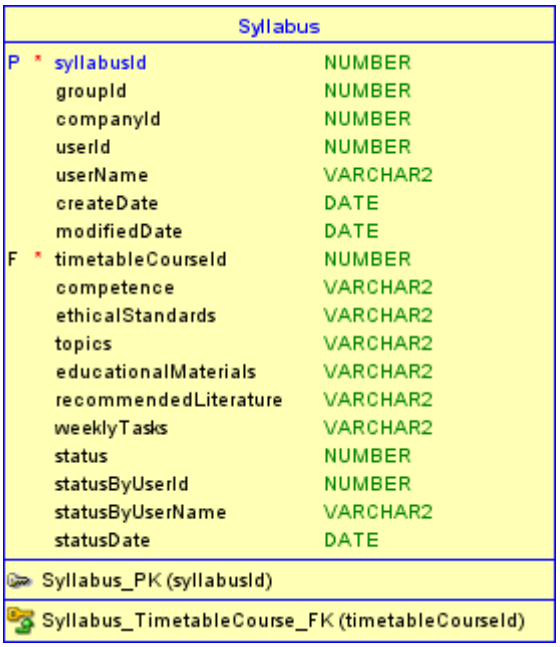
4.4.7. Órarendi Kurzus



4.4.8. Oktató - Órarendi Kurzus kapcsolat



4.4.9. Sillabusz



4.5. Implementációs kérdések

4. lépés implementáció. Konkrét kódrészletek jönnek ide! Egy sima entity hozzáadás (pl.: Syllabus Entity) hogyan lett elkészítve.

4.5.1. Fejlesztéshez használt eszközök

Eclipse

Gradle

Git

4.5.2. Liferay Service Builder

Mi ez? Mire jó? Mit nem csinál? Mit csinál? Hogyan épülnek egymásra a rétegek? Megemlíteni a diagramon is!

Service builderben: companyId groupId egyébId-k

4.5.3. Portlet class-ról leírás

4.5.4. JSP-kről példa leírás

Alloy UI http://proliferay.com/liferay-alloy-ui-basics/

Ide jöhet még az AJAX-os dependent dropbdown lista a syllabus hozzáadása példával, mert az jó hosszú.

4.5.5. Keresőintegráció

Mire jó ez? Hogyan lett megvalósítva?

4.5.6. Asset integráció

Mi az az Asset Framework Liferay-en belül? Mire jó? Hogyan kell megvalósítani? (megemlíteni a két *Asset* osztályt. Libreoffice-szal konfigurálva mire jó? Export/import.

4.5.7. Workflow

Létrehozok egy syllabuszt, ezután vagy elfogadja valaki vagy elutasítja. Csak akkor látható a nagyvilág számára, ha el lett fogadva. Mindez a Liferay beépített workflow kezelésével lett elérve.

4.5.8. Templating

Apache Velocity, Apache Freemarker-t itt meg lehetne említeni, hogy mit lehetett volna elérni vele.

4.5.9. Export/Import

Lehessen importálni az adatokat csv fájlból és persze exportálni is.

OpenCSV megfelel az RFC csv-nek. Leírni mi az az rfc, hivatkozni rá, stb...

4.5.10. nyelvesítés

Hogyan lett megvalósítva? Konkrét liferay-es példákkal lesz majd leírva.

4.5.11. WYSIWYG szerkesztők

Mi ez? Hol van ez használva? Mire jó?

5. fejezet

Az alkalmazás bemutatása

Az alábbiak során felhasználói szemszögből fogom bemutatni az elkészült alkalmazást.

5.1. PanelApp portles cucc ide

ControlPanel volt régen, de most már PanelApp van 7.0 óta.

5.2. Sima instanceable portlet

Ez az a portlet, amely a weboldalakra lesz kihelyezve, instanceable=true és konfigurálható, hogy mit mutasson. (Akár félév halapján, akár egyesével konfigurálható).

6. fejezet

Összefoglalás

Röviden, tömören és világosan ismertetni kell a fontosabb megállapításokat és következtetéseket. Bátran és egyértelműen közölni kell, ha valamely célkitűzést nem sikerült megvalósítani. (Megfelelő indoklás mellett ez egyáltalán nem csökkenti a diplomadolgozat értékét!)

6.1. Elért eredmények

6.2. Továbbfejlesztési lehetőségek

Köszönetnyilvánítás

Köszönet: család, barátok, egyetemnek, témavezetőmnek és a Liferay Hungary Kft. képzésének???

Irodalomjegyzék

Könyvek

[1] Richard Sezov Jr.: Liferay in Action, 2011, ISBN 9781935182825

[2] Ian Sommerville: Szoftverrendszerek fejlesztése, 2007, ISBN 9789635454785

Online források

[3] Liferay
`https://www.liferay.com/`

[4] JSR 286 - Portlet Specification 2.0
`https://www.jcp.org/en/jsr/detail?id=286`

[5] Sillabusz
`https://hu.wiktionary.org/wiki/sillabusz`

[6] Developing for the Liferay Platform 1
`https://www.liferay.com/services/training/6.2/topics/developer-1`

[7] Redmine
`http://www.redmine.org/`

[8] StatCounter `http://gs.statcounter.com/os-market-share`

A. függelék

Első függelék

A.1. leírás

Ide kerülnek azok a nagyobb méretű táblázatok, ábrák. Ide helyezhető el továbbá a kérdőíves felmérés alapjául szolgáló dokumentumok, továbbá a statisztikai és matematikai számítások alaptáblái is. Egyes esetekben rövidebb szöveges dokumentumok (pl. szerződése, jogszabály részletek, stb.) is helyet kaphatnak itt.

B. függelék

Második függelék

függelékes cucc