

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. Bevezetés	5
2. Követelmények meghatározása	7
2.1. A probléma megfogalmazása	7
2.2. Webalkalmazás	8
2.3. Portlet - JSR 286	8
2.4. Űrlap	9
2.5. Adminisztrációs felület	9
2.6. Nyelvesítés	10
2.7. Exportálás, Importálás	10
2.8. Sillabusz duplikáció	11
2.9. Jogosultságkezelés	11
2.10. Munkafolyamat	12
2.11. Asset Framework integráció	13
2.12. Webszolgáltatás	14
3. A webalkalmazások kérdéskörei	15
3.1. Általánosan a kérdéskörökről	15
3.2. Komplex tudást igényel	15
3.2.1. Adatbázisrendszerek	16
3.2.2. Alkalmazásszerverek	16
3.2.3. Üzleti logika	16
3.2.4. Webszolgáltatások	17
3.2.5. Megjelenítés	17
3.3. Tartalomkezelő rendszerek	17
3.4. Állandó változás	18

3.5.	Böngésző inkompatibilitás	19
3.6.	Biztonsági kérdések	20
3.7.	Megjelenítés	20
3.8.	Keresőoptimalizálás	22
4.	A megvalósítás lépései	23
4.1.	Szójegyzék	23
4.2.	Projektstruktúra kialakítása	23
4.3.	Liferay Service Builder	24
4.3.1.	A service.xml állomány létrehozása	25
4.3.2.	Globális információk megadása	25
4.3.3.	Entitások definiálása	26
4.3.4.	Tulajdonságok, attribútumok definiálása	26
4.3.5.	Kapcsolatok definiálása	27
4.3.6.	Rendezések definiálása	28
4.3.7.	Finder metódusok definiálása	29
4.3.8.	Kivételek definiálása	30
4.4.	Relációs adatmodell	31
4.4.1.	Mintatanterv	31
4.4.2.	Tantárgy	32
4.4.3.	Kurzus Típus	33
4.4.4.	Kurzus	33
4.4.5.	Félév	34
4.4.6.	Oktató	34
4.4.7.	Órarendi Kurzus	34
4.4.8.	Oktató - Órarendi Kurzus kapcsolat	35
4.4.9.	Sillabusz	35
4.5.	Szolgáltatások kialakítása	36
4.6.	Implementációs kérdések	38
4.6.1.	Portlet létrehozása	38
4.6.2.	Jogosultságkezelés	39
4.6.3.	Munkafolyamat támogatás	41

5. Az alkalmazás bemutatása	43
5.1. Főoldal	43
5.2. Listázó oldalak	44
5.3. Sillabusz létrehozása	48
5.4. Importálás / Exportálás	50
6. Összefoglalás	52
Köszönetnyilvánítás	54
Irodalomjegyzék	55
A. Első függelék	57
A.1. leírás	57
B. Második függelék	58

1. fejezet

Bevezetés

2015 februárjában, amikor még az utolsó félévemet kezdtem el a Debrecen Egyetem - Informatikai Karán, Programtervező Informatikus alapképzésen, a 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 egyedüli 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 syllabusz menedzselő vagy más néven syllabusz 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 később végre elkészült az új honlap, amelyhez már nem Java, hanem főleg PHP programozási nyelvet és egy általam nem ismert tartalomkezelő rendszert használtak a készítői, ezért várha-

tóan a továbbiakban az általam készített alkalmazás a jelenlegi formájában már sosem lesz éles körülmények között használva. Amikor kiderült, hogy a Liferay alapú honlap már sosem fog elkészülni, a témavezetőmmel való újabb egyeztetések után úgy döntöttem, hogy nem dobom ki az addig elkészített alkalmazást, hanem teljes egészében befejezem úgy, ahogyan az eredetileg is tervbe volt véve és diplomamunkát készítek belőle.

Diplomamunkámban egy olyan Java EE technológiákra és a Liferay platformra 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 szeretne 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 egyszerűbb megfelelés kedvéért.

A diplomamunkám további fejezetiben összegyűjtöttem a sillabusz kezelő alkalmazással szemben támasztott követelményeket, megemlítettem néhány általános webalkalmazásokkal kapcsolatos problémát, leírtam számos olyan lépést, amelyek elvégzésével alkalmazást lehet fejleszteni a Liferay platformra, illetve képekkel illusztráltam az elkészült alkalmazás egyes részeit, továbbá számos fejezetben említést tettem az alkalmazás fejlesztése során szerzett tapasztalataimról is.

2. fejezet

Követelmények meghatározása

2.1. A probléma megfogalmazása

Minden szoftverfejlesztési folyamat a követelmények meghatározásával, a megoldandó probléma vagy másképpen nevezve a megoldandó feladat megfogalmazásával kezdődik. Mivel minden program azért jön létre, hogy egy előre meghatározott problémát vagy feladatot oldjon meg, ezért én is egy ilyen 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 rövidített 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 lenne 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 honlapon 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 sillabuszokhoz 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 elvégezni 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 oldal támogatottságának megfelelő webböngésző, mint például a Google Chrome vagy a Mozilla Firefox.

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 szembesültem.

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 syllabuskezelő 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 syllabusz 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 syllabuszhoz tartozó összes félévente változó adatot meg lehet adni. Mivel a program egy webalkalmazáson belül lesz használva, célszerű úgynevezett WYSIWYG¹ 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 létrehozandó syllabusszal kapcsolatban és ha a kitöltött adatok nem felelnek meg a rendszerbe épített megszorításoknak, akkor jeleznie kell a syllabuszt létrehozó felhasználó 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 tennie.

2.5. Adminisztrációs felület

A syllabuszok kezeléséhez szükség van egy olyan felületre, ahol a megfelelő jogosultságokkal rendelkező felhasználók meg tudják adni a syllabuszok alapadatait. Még pontosabban meghatározva egy olyan felületre van szükség, ahol a Tanulmányi Osztály dolgozói vagy megfelelő hatáskörrel rendelkező más személyek fel tudják tölteni vagy meg tudják adni a tantárgyak

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

alapadatait. Ezen alapadatokat, magukat a sillabuszokat létrehozó oktatók nem tudják módosítani, ők csak tantárgyat tudnak választani a sillabusz hozzáadása során, amelynek hatására a megfelelő alapadatok automatikusan bekerülnek a létrehozandó 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 menedzselésére. 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.

Nem csak az adatok importálására, hanem a sillabuszkezelőből történő adatok kiexportálására is szükség van. Ennek számos oka 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 a következő három lépésből álló algoritmus végrehajtá-

sával:

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 tantárgyhoz tartozó sillabusz 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 fejlesztési folyamatnak. 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 szükséges 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 jóváhagyni 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 sillabusz kezelőhöz tartozó munkafo-

lyamatot, 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, mikor és mit hagyott jóvá, esetleg utasított el. Minden egyes jóváhagyási folyamat során a felelős személyek véleményt is írhatnak, így ez a keretrendszer 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 megjelenítésükért 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 lehetséges legyen, így nem lesz szükség saját megjelenítő portlet készítésére, elég lesz egy-egy Asset Publisher konfigurálása 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 beépített portlet-hez nyújt egy-egy 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 szillabusz 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 valósítani. 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 találkozhatja 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 bízunk. 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 alkalmazáserverből 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 pedig 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 alkalmazáserver köré kellett építenünk. A két közötti különbség feltűnően nagy volt. Tapasztalataim szerint az előbbi funkcionalitásban, utóbbi pedig teljesítményben volt jobb a másikinál. 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 funkcióban 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 tizenhét 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 [9] és a Blade CLI [10] programok használatát is, mert korábban csak az Apache Ant [11] és az Apache Maven [12] 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 szillabusz 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 kiderítenem, 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 [15] a másik pedig a Liferay által is erőszertetel használt Alloy UI [16] 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 kavaráó veszprémi pizzéria weboldala, ahol nem is a weboldalt törték fel, hanem az alkalmazást futtató szervert és ezáltal minden adathoz hozzá tudtak férni, amely a pizzériával volt kapcsolatos.

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 szillabuszok adatait, ezért nekem, mint a szillabusz 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 ellenőrizték, 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 funkcionalításban, hanem megjelenítésben is megfelelő legyen egy-egy webalkalmazás. Rossz felhasználói felületre rengeteg példa létezik, mégis szerintem az egyik legjobb példa a nemrég bevezetett Ügyfélkapu rendszere. 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. Adaptívnek 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 pedig 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 jobban 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 megvalósítással kapcsolatos dologba, fontosnak tartom, hogy a későbbi fejezetekben megjelenő leggyakoribb egyedek neveit előre bevezessem magyar és angol nyelven egyaránt:

Magyarul	Angolul
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 függőseggként 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.

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 syllabuskezelő 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. Relációs adatmodell

Ebben a fejezetben azt szeretném bemutatni, hogy milyen egyedeket hoztam létre a Liferay Service Builder segítségével a syllabusz kezelő alkalmazás számára. A teljes service.xml állomány túl nagy terjedelme miatt az Oracle SQL Developer Data Modeler alkalmazás segítségével készítettem belőle egy olyan relációs modellt, amely egy az egyben megfelel a service.xml-ben leírtaknak. Sajnos a kiexportált diagram minden esetben vagy túl széles vagy pedig túl magas volt, így teljes egészében nem fért volna ki, ezért minden egyedet külön ábrán helyeztem el.

Az alfejezetekben lévő diagramokon észre lehet venni, hogy minden tábla rendelkezik az alábbi attribútumok mindegyikével:

Attribútum neve	Leírás
groupId	Egy szervezet vagy webhely azonosítója
companyId	Egy portál példány azonosítója
userId	A tulajdonos egyedi felhasználói azonosítója
username	A tulajdonos felhasználóneve
createDate	Létrehozás dátuma
modifiedDate	Utolsó módosítás dátuma

A *groupId* és *companyId* attribútumokon kívül mindegyik jelentése egyértelmű. A *groupId* egy szervezet vagy egy webhely azonosítója lehet. A Liferay lehetőséged ad az adminisztrátorok számára, hogy különböző szervezeteket és webhelyeket definiáljanak és ezen mindegyikéhez automatikusan létrejön egy-egy azonosító, amelyet *groupId*-nek neveztek el. Ha a Liferay portálból egyszerre több virtuális példány is fut, akkor a *companyId* alapján lehet eldönteni, hogy melyik egyed, melyik példányhoz tartozik, ezáltal minden virtuális példány csak a saját egyedeihez férhet hozzá.

4.4.1. Mintatanterv

Egy mintatanterv két fontos információval rendelkezik, mindig van egy egyedi kódja és egy neve. Például:

Mintatanterv kódja	Mintatanterv neve
IBN_PTI_07	Programtervező Informatikus BSc nappali 2007
IMN_PTI_07	Programtervező Informatikus MSc nappali 2007

Curriculum		
P *	curriculumId	NUMBER
	groupId	NUMBER
	companyId	NUMBER
	userId	NUMBER
	userName	VARCHAR2
	createDate	DATE
	modifiedDate	DATE
U	curriculumCode	VARCHAR2
	curriculumName	VARCHAR2
Curriculum_PK (curriculumId)		
Curriculum__UN (curriculumCode)		

4.1. ábra. A mintatanterv tábla

4.4.2. Tantárgy

Egy tantárgy mindig egy adott mintatantervhez kötődik. Három fontos attribútummal rendelkezik: tantárgykód, név és kredit érték.

Subject		
P *	subjectId	NUMBER
	groupId	NUMBER
	companyId	NUMBER
	userId	NUMBER
	userName	VARCHAR2
	createDate	DATE
	modifiedDate	DATE
U	subjectCode	VARCHAR2
	subjectName	VARCHAR2
	credit	NUMBER
UF *	curriculumId	NUMBER
Subject_PK (subjectId)		
Subject__UN (curriculumId, subjectCode)		
Subject_Curriculum_FK (curriculumId)		

4.2. ábra. A tantárgy tábla

4.4.3. Kurzus Típus

A kurzus típus egyed bevezetésére azért volt szükség, mert egy kurzusból többféle is létezhet és nem akartuk, hogy a forráskódot kelljen módosítani újabb típusok bevezetése esetén. Tipikusan elmélet, gyakorlat, labor vagy vizsgakurzus típusokat reprezentál.

CourseType		
P *	courseTypeId	NUMBER
	groupId	NUMBER
	companyId	NUMBER
	userId	NUMBER
	userName	VARCHAR2
	createDate	DATE
	modifiedDate	DATE
U	typeName	VARCHAR2
CourseType_PK (courseTypeId)		
CourseType__UN (typeName)		

4.3. ábra. A kurzus típus tábla

4.4.4. Kurzus

Egy kurzus mindig egy tantárgyhoz és egy kurzus típushoz kötődik. A tantárgyhoz a kurzus adja meg a heti és féléves óraszámokat.

Course		
P *	courseId	NUMBER
	groupId	NUMBER
	companyId	NUMBER
	userId	NUMBER
	userName	VARCHAR2
	createDate	DATE
	modifiedDate	DATE
UF *	subjectId	NUMBER
	hoursPerSemester	NUMBER
	hoursPerWeek	NUMBER
UF *	courseTypeId	NUMBER
Course_PK (courseId)		
Course__UN (subjectId, courseId)		
Course_Subject_FK (subjectId)		
Course_CourseType_FK (courseTypeId)		

4.4. ábra. A kurzus tábla

4.4.5. Félév

A félév egyed, ahogyan azt a neve is sugallja egy egyetemi félévet reprezentál.

Semester		
P *	semesterId	NUMBER
	groupId	NUMBER
	companyId	NUMBER
	userId	NUMBER
	userName	VARCHAR2
	createDate	DATE
	modifiedDate	DATE
	curriculumCode	VARCHAR2
	curriculumName	VARCHAR2
U	beginYear	NUMBER
U	endYear	NUMBER
U	division	NUMBER
Semester_PK (semesterId)		
Semester__UN (beginYear, endYear, division)		

4.5. ábra. A félév tábla

4.4.6. Oktató





Az oktató egyed bevezetésére azért volt szükség, mert nem biztos, hogy egy oktató azt a nevet szeretné megjeleníteni egy syllabusban, mint amellyel a Liferay-es felhasználói fiókja is rendelkezik.

Lecturer		
P *	lecturerId	NUMBER
	groupId	NUMBER
	companyId	NUMBER
	userId	NUMBER
	userName	VARCHAR2
	createDate	DATE
	modifiedDate	DATE
U	lecturerName	VARCHAR2
U	lecturerLiferayUserId	NUMBER
Lecturer_PK (lecturerId)		
Lecturer__UN (lecturerName, lecturerLiferayUserId)		

4.6. ábra. Az oktató tábla

4.4.7. Órarendi Kurszus




Egy órarendi kurzus akkor jön létre, amikor egy adott félévben egy adott kurzust meghirdetnek a hallgatók számára.

TimetableCourse		
P *	timetableCourseId	NUMBER
	groupId	NUMBER
	companyId	NUMBER
	userId	NUMBER
	userName	VARCHAR2
	createDate	DATE
	modifiedDate	DATE
UF *	courseId	NUMBER
UF *	semesterId	NUMBER
U	timetableCourseCode	VARCHAR2
U	subjectType	VARCHAR2
	recommendedTerm	NUMBER
	limit	NUMBER
	classScheduleInfo	VARCHAR2
	description	VARCHAR2
 TimetableCourse_PK (timetableCourseId)		
 TimetableCourse__UN (courseId, semesterId, timetableCourseCode, subjectType)		
 TimetableCourse_Course_FK (courseId)		
 TimetableCourse_Semester_FK (semesterId)		

4.7. ábra. Az órarendi kurzus tábla

4.4.8. Oktató - Órarendi Kurzus kapcsolat

Ez az a tábla, amely képes eltárolni, hogy mely oktató, mely órarendi kurzushoz tartozik. Erre azért volt szükség, mert egy ilyen kurzusnak akárannyi oktatója lehet és egy oktató is akárannyi kurzust tarthat egyszerre egy félévben.

Lecturers_TimetableCourses		
PF *	lecturerId	NUMBER
PF *	timetableCourseId	NUMBER
 Lecturers_TimetableCourses_PK (lecturerId, timetableCourseId)		
 Lecturers_TimetableCourses_Lecturer_FK (lecturerId)		
 Lecturers_TimetableCourses_TimetableCourse_FK (timetableCourseId)		

4.8. ábra. Az oktatókat és órarendi kurzusokat összekapcsoló tábla

4.4.9. Sillabusz

Egy sillabusz mindig egy órarendi kurzushoz tartozik. Ez a tábla tartalmazza azon adatokat, amelyek félévente változhatnak egy újabb sillabusz kiadásakor.

Syllabus		
P *	syllabusId	NUMBER
	groupId	NUMBER
	companyId	NUMBER
	userId	NUMBER
	userName	VARCHAR2
	createDate	DATE
	modifiedDate	DATE
F *	timetableCourseId	NUMBER
	competence	VARCHAR2
	ethicalStandards	VARCHAR2
	topics	VARCHAR2
	educationalMaterials	VARCHAR2
	recommendedLiterature	VARCHAR2
	weeklyTasks	VARCHAR2
	status	NUMBER
	statusByUserId	NUMBER
	statusByUserName	VARCHAR2
	statusDate	DATE
Syllabus_PK (syllabusId)		
Syllabus_TimetableCourse_FK (timetableCourseId)		

4.9. ábra. A sillabusz tábla

4.5. Szolgáltatások kialakítása

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 szükséges szolgáltatások implementálása.

Liferay esetén kétféle szolgáltatást különböztetünk meg: helyi szolgáltatásokat és webszolgáltatásokat. Általában a webszolgáltatások csak egy interfészt nyújtanak a külvilág számára a helyi szolgáltatások elérésére, ezért én csak a helyi szolgáltatások létrehozását mutatom be.

Az egyes egyedekhez a *hu.unideb.inf.service.impl* csomagban találhatóak meg azok az osztályok, melyeket újabb metódussal lehet bővíteni. Ebben a csomagban minden egyedhez két osztály tartozik: egy *ServiceImpl* és egy *LocalServiceImpl* nevre végződő osztályok. Az előbbire a webszolgáltatások, az utóbbira pedig a helyi szolgáltatások miatt van szükség.

Az egyszerűség kedvéért az összes szolgáltatás közül csak a *CurriculumLocalServiceImpl* osztály lényeges részeit szeretném bemutatni.

```

@ProviderType
public class CurriculumLocalServiceImpl
    extends CurriculumLocalServiceBaseImpl {

    public Curriculum getCurriculumByCode(
        String curriculumCode)
        throws SystemException ,
            NoSuchCurriculumException {...}

    public Curriculum fetchCurriculumByCode(
        String curriculumCode)
        throws SystemException {...}

    public Curriculum addCurriculum(
        String curriculumCode , String curriculumName ,
        ServiceContext serviceContext)
        throws PortalException , SystemException {...}

    public Curriculum updateCurriculum(
        long userId ,
        long curriculumId ,
        String curriculumCode ,
        String curriculumName ,
        ServiceContext serviceContext)
        throws PortalException , SystemException {...}

    public Curriculum deleteCurriculum(
        long curriculumId ,
        ServiceContext serviceContext)
        throws PortalException , SystemException {...}
}

```

A `getCurriculumByCode(...)` metódus egy mintatanterv kódja alapján vissza tudja adni a megfelelő mintatantervet, viszont ha nincs a kódnak megfelelő adat az adatbázisban, akkor `NoSuchCurriculumException` fog kiváltódni.

A `fetchCurriculumByCode(...)` metódus ugyan azt csinálja, mint a `getCurriculumByCo-`

de(...), mindössze annyi a különbség a kettő között, hogy ez a metódus null értékkel tér vissza abban az esetben, ha az adatbázisban nem található meg a megfelelő mintatanterv.

Az `addCurriculum(...)` segítségével új mintatantervet lehet hozzáadni az adatbázishoz, az `updateCurriculum(...)` metódussal frissíteni lehet egy létező mintatanterv adatait, a `deleteCurriculum(...)` meghívásával pedig törölni lehet egy mintatantervet az adatbázisból. Ha törlés során kiderülne, hogy a törlendő mintatantervhez legalább egy tantárgy is kapcsolódik, akkor `DeleteSubjectsFirstException` kivétel váltódik ki és a törlés nem hajtodik végre.

Az `add` és `update` metódusok validációt is végrehajtanak. Abban az esetben, ha a létrehozandó vagy módosítani egyed egyik attribútuma nem megfelelő értékkel rendelkezik, akkor szintén kivétel dobódik és a művelet végrehajtása megszakad, amely hatására hibaüzenetet lehet megjeleníteni a felhasználói felületen.

A `CurriculumServiceImpl` osztályon belül szinte minden metódus először azt ellenőrzi, hogy a felhasználó rendelkezik-e a megfelelő jogosultsággal a művelet végrehajtására. Ha igen, akkor a `CurriculumLocalServiceImpl` osztályon belül található megfelelő metódust hívódik meg, ha viszont valamilyen oknál fogva a felhasználó nem rendelkezne a megfelelő jogosultsággal, akkor `PrincipalException` váltódik ki és biztonsági okokból a művelet végrehajtása el sem kezdődik.

A szolgáltatások implementálása után újra kell generáltatni a Service Builder által létrehozott többi forráskódot.

4.6. Implementációs kérdések

4.6.1. Portlet létrehozása

A szolgáltatások kialakítása utána már csak egy felhasználói interfész létrehozására volt szükség. Követelményként az volt meghatározva, hogy legyen egy adminisztrációs felület, ahol a tantárgyak alapadatait meg lehet adni. A szillabusz kezelő alkalmazás adminisztrációs felületét ott helyeztem el, ahol a többi alkalmazás adminisztrációs felülete is megtalálható, azaz az Liferay 7.0-ban a régi vezérlőpultot felváltó oldalsáv "Tartalom" menüpontja alatt.

A felület létrehozására mindössze egy portlet létrehozására, a megfelelő műveletek definiálására és a megjelenítő oldalak kialakítására volt szükség. A technikai részletekbe nem szeretnék belemenni, mert a teljes diplomamunka is kevés lenne mindezek elmagyarázására.

4.6.2. Jogosultságkezelés

A kialakított felületen minden elem esetén ellenőrizve van, hogy a felhasználó rendelkezik-e a megfelelő jogosultságokkal, annak használatára. Ha nem rendelkezik, akkor az elem vagy akár az egész oldal egyszerűen nem jelenik meg, még hibaüzenet sem jelenik meg arról, hogy nincs meg a felhasználónak a megfelelő joga, mert ezzel csak a biztonságosság szintjét csökkentettem volna.

A Liferay támogatja a szerepkör alapú jogosultságkezelést. A portál három különböző típusú szerepkört különít el:

- normál,
- webhely,
- szervezet

típusú szerepköröket. A jogosultságkezelés beállításához mindössze egy megfelelő típusú szerepkört kell létrehozni és magára a szerepkörre kell meghatározni, hogy mely jogosultságokkal rendelkezik. Ezután a szerepkört hozzá lehet rendelni egy felhasználóhoz és máris rendelkezik a szükséges jogosultságokkal. Sok helyen az ilyen fajta jogosultságkezelésre "Role Object Pattern" névként szokás hivatkozni, amely egy széles körben elfogadott tervezési mintának felel meg.

Forráskód szinten a jogosultságkezelés kialakításához a következő dolgokra volt szükségem:

1. Jogosultságok definiálása
2. Jogosultság ellenőrző osztályok kialakítása
3. Jogosultság ellenőrzés a szükséges helyeken

Jogosultságok definiálása

A syllabusz kezelő alkalmazás esetén minden jogosultságot a syllabus-manager-service projekten belül az *src/main/resources/META-INF/resource-actions/default.xml* állományon belül definiáltam. A Liferay két szintű jogosultságokat különböztet meg: felsőbb szintű (top level) és erőforrás szintű jogosultságokat. Felsőbb szintű akkor lesz egy jogosultság, ha nem tartozik semmilyen erőforráshoz. Előfordulhat, hogy egyes helyeken a felsőbb szintű jogosultságokra a magyar nyelvben általános jogosultságokként hivatkoznak. Példa jogosultságokra:

4.1. táblázat. Példák felsőbb szintű jogosultságokra

Jogosultság neve	Leírás
ADD_CURRICULUM	Mintatanterv hozzáadási jog
DELETE_CURRICULUMS	Csoportos mintatanterv törlési jog

4.2. táblázat. Példák mintatanterv esetén erőforrás szintű jogosultságokra

Jogosultság neve	Leírás
DELETE	Egyéni törlési jog egy adott mintatantervre
VIEW	Megtekintési jog egy adott mintatantervre

Jogosultság ellenőrző osztályok kialakítása

Ezeket az osztályokat a kódismétlés csökkentése érdekében kellett kialakítanom. Mindegyik tartalmaz egy `check` és `contains` nevű metódust. A `check` metódus `PrincipalException` kivételt dob minden olyan esetben, amikor a felhasználó nem rendelkezik adott jogosultsággal, a `contains` pedig igaz/hamis értékkel tér vissza annak függvényében, hogy a felhasználó rendelkezik-e a megfelelő jogosultságokkal.

Ezek a jogosultságokat ellenőrző osztályok a *hu.unideb.inf.service.permission* csomagon belül lettek elhelyezve. Felsőbb szintű jogosultságok ellenőrzésére a `ModelPermission` osztályt, az erőforrás szintűekre pedig a minden egyednek külön létrehozott osztályokat lehet használni.

Jogosultság ellenőrzés a szükséges helyeken

A jogosultság ellenőrzése úgy történik, hogy a megfelelő helyen a megfelelő jogosultság ellenőrző osztály megfelelő metódusa a megfelelő paraméterekkel meghívódik. Jó példa erre a `syllabusok` listázásánál végzett erőforrás szintű ellenőrzés:

```
<c:if test='<%=SyllabusPermission.contains(
    permissionChecker, syllabus.getSyllabusId(), "VIEW")%'>
    ...
</c:if>
```

4.6.3. Munkafolyamat támogatás

Az egyik legfontosabb követelmény a munkafolyamat támogatása volt. Ezt a portálba épített keretrendszer segítségével tettem lehetővé. A munkafolyamat kialakítása egyszerű folyamatnak tűnik:

1. Asset integráció
2. Új attribútumok felvétele
3. Szolgáltatások módosítása
4. WorkflowHandler osztály létrehozása

Asset integráció

Asset integráció alatt azt kell érteni, hogy egy entitás kezelhetővé válik az Asset Publisher nevű portleten keresztül. Erre a részre amúgy is szükség lett volna, mert követelményként volt meghatározva, hogy egy sillabuszt az Asset Publisher-en keresztül kell tudni létrehozni, módosítani és legfőképpen megjeleníteni.

Új attribútumok felvétele

A sillabusz egyed esetén négy új attribútum felvételére volt szükség:

Attribútum neve	Leírás
status	Az egyed munkafolyamatbeli állapotát jelzi
statusByUserId	Az egyedet létrehozó felhasználó azonosítója
statusByUserName	Az egyedet létrehozó felhasználóneve
statusDate	Az egyed munkafolyamatbeli állapotának utolsó módosítási dátuma

Szolgáltatások módosítása

A SyllabusLocalServiceImpl osztályon belül található metódusokat úgy kellett módosítani, hogy hozzáadáskor és frissítéskor új munkafolyamat induljon el, törléskor pedig szűnjenek meg.

Ugyan ebben az osztályban szükség volt még egy újabb, updateStatus(...) nevű metódus bevezetésére is. A metódus megfelelő paraméterekkel történő meghívásával lehet állítani az egyes sillabuszok munkafolyamatbeli állapotait.

WorkflowHandler komponens létrehozása

A Liferay a 7.0 verzió megjelenése óta komponensekkel és modulokkal dolgozik. Mindezt az OSGI keretrendszer bevezetése tette lehetővé. Munkafolyamat kezelésnél nekem is egy újabb komponenst kellett létrehoznom, annak érdekében, hogy az OSGI futtatórendszer tudomást szerezzen arról, hogy a sillabusz egyedek is támogatják a portálba épített munkafolyamat keretrendszert.

5. fejezet

Az alkalmazás bemutatása

Számtalan képet lehetne itt elhelyezni arról, hogy hogyan is néz ki a valóságban a sillabusz kezelő alkalmazás, de az oldalakkal való takarékoság érdekében csak a leglényegesebb funkciókat szeretném bemutatni felhasználói szemszögből.

5.1. Főoldal

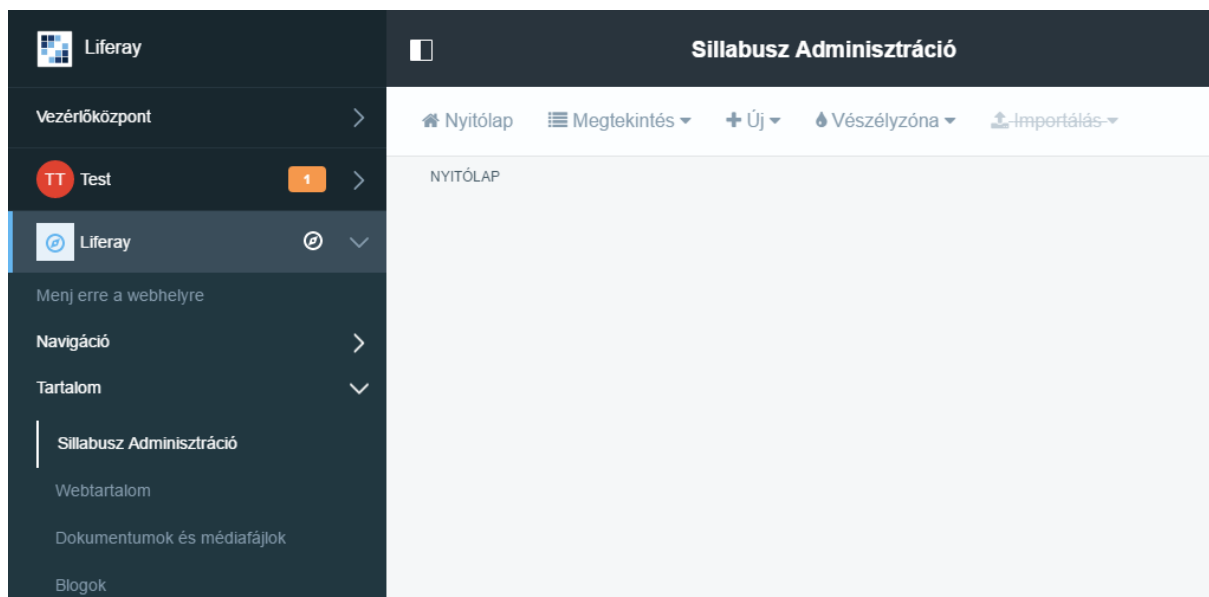
A főoldal egész egyszerű felépítést kapott. Ezen az oldalon mindössze egy menüsor és egy breadcrumb található meg. A képen jól látszik a bal oldali oldalsáv és a benne elhelyezett *Sillabusz Adminisztráció* menüpont, amely a régi vezérlőpultot váltotta fel. A menüsor és breadcrumb szinte az összes oldalon ugyan így néz ki, értelemszerűen az utóbbi egyes helyeken több elemet is tartalmaz, nem csak egyet, mint ahogyan az a főoldalon is látható.

A menüsor *Megtekintés* menüpontja alatt négy elem található: *Mintatantervek*, *Kurzus típusok*, *Félévek* és *Oktatók*.

Az *Új* menüpont alatt minden egyed felsorolásra került. A megfelelő egyed kiválasztása után a rendszer lehetőséget biztosít egy új elem hozzáadására.

A *Veszélyzóna* menüpont olyan műveleteket foglal magában, amelyek az egész rendszer működését befolyásolhatják. Jelenleg két almenüpont tartozik hozzá: *Adatbázis törlése* és *Exportálás / Importálás*.

A menüsor utolsó eleme egy halványszürke, áthúzott betűkkel megjelenített *Importálás* menüpont. Ez azért így jelenik meg, mert használata erősen ellenjavallott, erre csak az eredetileg az egyetemről kapott adatok betöltése során volt szükségem.

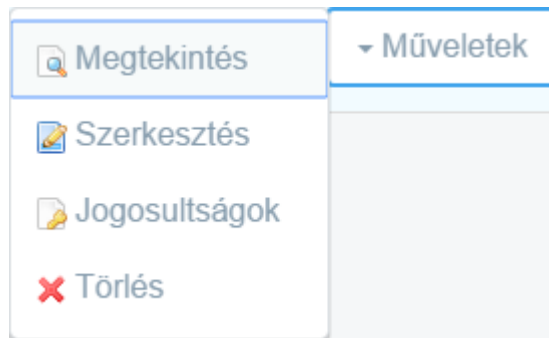


5.1. ábra. A sillabusz kezelő portlet főoldala

5.2. Listázó oldalak

Minden általam definiált egyedhez külön-külön listázó oldalt hoztam létre. A kialakítás során törekedtem egyfajta hierarchikus megjelenítésre, azaz a mintatantervből, illetve a félévekből kiindulva el lehet jutni a sillabuszok megjelenítéséig.

Minden listázó oldalon minden egyes bejegyzés mellett található egy *Műveletek* menüpont. Ez egyetlen egy esetet kivéve minden egyed esetén három elemet tartalmaz: *Szerkesztés*, *Jogsultságok* és *Törlés*. Az egyetlen kivétel ez alól az órarendi kurzusok, ahol megtalálható egy negyedik elem is, a *Megtekintés* menüpont. Ezt a menüpontot kiválasztva lehet eljutni az adott órarendi kurzushoz tartozó sillabusz listázó oldalra.



5.2. ábra. Egy listázóban található elem lehetséges műveletei

Syllabus Adminisztráció			
Nyitólap Megtekintés Új Vészélyzóna Importálás			
NYITÓLAP > MINTATANTERVEK			
Oldal 1 a következőből: 2		Tételek oldalanként	
1 - 20 / 26 tételek megjelenítése.		← Első Előző Következő utolsó →	
<input type="checkbox"/>	Mintatanterv kódja	Mintatanterv neve	
<input type="checkbox"/>	IBL_PTI_07	Programtervező informatikus BSc levelező 2007	Műveletek
<input type="checkbox"/>	IBN_GAZDINF_10_angol	Business Informatics BSc 2010	Műveletek
<input type="checkbox"/>	IBN_GAZDINF_14_angol	Business Informatics BSc 2014	Műveletek
<input type="checkbox"/>	IBN_GI_EGAZD_angol	e-Business Specialization	Műveletek

5.3. ábra. A mintatanterv listázó

Sillabusz Adminisztráció				
Nyitólap Megtekintés Új Vészélyzóna Importálás				
NYITÓLAP > MINTATANTERVEK > IBL_PT1_07 - PROGRAMTERVEZŐ INFORMATIK...				
<input type="checkbox"/>	Tantárgykód	Tantárgynév	Kredit	
<input type="checkbox"/>	ILDK231-K3	Az internet eszközei és szolgáltatásai	3	Műveletek
<input type="checkbox"/>	ILDS001-K10	Szakdolgozat 1	10	Műveletek
<input type="checkbox"/>	ILDS002-K10	Szakdolgozat 2	10	Műveletek

5.4. ábra. A tantárgy listázó

Sillabusz Adminisztráció				
Nyitólap Megtekintés Új Vészélyzóna Importálás				
NYITÓLAP > MINTATANTERVEK > IBL_PT1_07 - PROGRAMTERVEZŐ INFORMATIK... > ILDK231-K3 - AZ INTERNET ESZKÖZEI ÉS SZOL...				
<input type="checkbox"/>	Kurzus típus	Féléves óraszám	Heti óraszám	
<input type="checkbox"/>	Elmélet	12	0	Műveletek

5.5. ábra. A kurzus listázó

Syllabus Adminisztráció											
Nyitólap Megtekintés + Új Vészélyzóna Importálás											
NYITÓLAP > MINTATANTERVEK > IBL_PT1_07 - PROGRAMTERVEZŐ INFORMATIK... > ILDK231-K3 - AZ INTERNET ESZKÖZEI ÉS SZOL... > ELMÉLET: 12 FÉLÉVES ÓRASZÁM, 0 HETI ÓRAS..											
<input type="checkbox"/>	Mintatanterv kódja	Mintatanterv neve	Tantárgykód	Tantárgynév	Kredit	Félév	Kurzus típus	Féléves óraszám	Heti óraszám	Órarendi kurzus kódja	Tantárgy típusa
<input type="checkbox"/>	IBL_PT1_07	Programtervező informatikus BSc levelező 2007	ILDK231-K3	Az internet eszközei és szolgáltatásai	3	2017/2018/2	Elmélet	12	0	órarendi kurzus kód	tantárgytípus

5.6. ábra. Az órarendi kurzus listázó

Syllabus Adminisztráció									
Nyitólap Megtekintés + Új Vészélyzóna Importálás									
NYITÓLAP > MINTATANTERVEK > IBL_PT1_07 - PROGRAMTERVEZŐ INFORMATIK... > ILDK231-K3 - AZ INTERNET ESZKÖZEI ÉS SZOL... > ELMÉLET: 12 FÉLÉVES ÓRASZÁM, 0 HETI ÓRAS.. > ÓRARENDI KURZUS KÓD									
<input type="checkbox"/>	Kompetencia	Etikai normák	Témák	Oktatási segédanyagok	Ajánlott irodalom	Heti bontás	Státusz	Syllabus utolsó módosítási dátuma	
<input type="checkbox"/>	Kompetencia leírása	Etikai normákra való hivatkozás ide jöhet	A témák ide jönnek	Oktatási segédanyag lista lenne	Nincs	1. első 2. második 3. harmadik	Jóváhagyva	Ennyi ideje: 1 Perc	Műveletek
<input type="checkbox"/>	Sok minden	Teszt szöveg	Néhány elem jöhet ide	Könyvek	A kiadott diasorozat.		Függő	Ennyi ideje: 3 Perc	Műveletek

5.7. ábra. A syllabus listázó

5.3. Sillabusz létrehozása

Egy sillabusz hozzáadása során az első lépés mindig az alapinformációk megadása. Az adatok megadása alatt a megfelelő egyedek kiválasztását kell érteni. Az oldalon elhelyezett legördülő menük tartalma az eggyel felette lévő menü értékétől függ, kivéve a legelső esetben, tehát például csak azon tantárgyak kerülnek kilistázásra, amelyek a kiválasztott mintatantervhez tartoznak.

Az alapinformációk meghatározása után szöveges beviteli mezők találhatók. Ezek mindegyike kötelező mező, azaz, ha egyiket valaki nem töltene ki, akkor hiba üzenetet küld a rendszer arról, hogy a mező tartalma üres. A legutolsó szöveges beviteli mező egy WYSIWYG szerkesztő.

A szöveges beviteli mezők után még megtalálható egy utolsó rész, amely a Liferay Asset Publisher miatt jelenik meg. Itt lehet címkéket és kapcsolódó tartalmakat hozzárendelni a létrehozandó sillabuszhoz. Az Asset Publisher számos beállítási lehetőséggel rendelkezik, a címkék alapján történő szűrés az egyik leggyakrabban használt funkciója.

☐

Sillabusz Adminisztráció

Sillabusz hozzáadása

Tantárgyválasztó

Mintatanterv *

IBL_PTI_07 - Programtervező informatikus BSc levelező 2007

Tantárgy *

ILDK231-K3 - Az internet eszközei és szolgáltatásai

Kurzus *

Elmélet: 12 Féléves óraszám, 0 Heti óraszám

Órarendi kurzus *

Órarendi kurzus kódja: órarendi kurzus kód, Tantárgy típusa: tantárgytípus, Ajánlott félév: 1, Létszámkorlát: 20, Órarendi információk:

5.8. ábra. Egy sillabusz létrehozása 1/4

Sillabusz Adminisztráció

Sillabusz adatainak megadása

Kompetencia *

Etikai normák *

Témák *

Oktatási segédanyagok *

Ajánlott irodalom *

5.9. ábra. Egy sillabusz létrehozása 2/4

Sillabusz Adminisztráció

Ajánlott irodalom *

Heti bontás

B

I

U

S

x₂

x²

I_x

A

A

Stilus

Mé...

Súgó

Alt+0

5.10. ábra. Egy sillabusz létrehozása 3/4

49

Sillabusz Adminisztráció

Kategorizálás

Címkék

Új

Választ

Kapcsolódó tartalmak

Egyik sem

Választ

Mentés

Mégsem

5.11. ábra. Egy sillabusz létrehozása 4/4

5.4. Importálás / Exportálás

Az alkalmazás képes az összes általa tárolt adat exportálására és ezen exportált adatok újbóli importálására is. Jelenleg a rendszer támogatja mind az XML, mind a CSV formátumú állományok kezelését. A sillabuszok exportálása gombok esetén, nem csak a sillabuszok, hanem az összes mintatanterv, tantárgy, kurzus és órarendi kurzus is exportálódik, ezért nincs külön gomb ezekre az egyedekre elhelyezve az oldalon.

Sillabusz Adminisztráció

Nyitólap

Megtekintés

+ Új

Vészélyzóna

Importálás

Exportálás

Sillabuszok exportálása CSV

Oktatók exportálása CSV

Szemeszterek exportálása CSV

Kuruzstípusok exportálása CSV

Sillabuszok exportálása XML

Oktatók exportálása XML

Szemeszterek exportálása XML

Kuruzstípusok exportálása XML

Importálás

Fájl

Fájl kiválasztása

Nincs fájl kiválasztva

Tartalom típusa

☒ CSV ☐ XML

Entitás típusa

☒ Sillabuszok ☐ Oktatók ☐ Félévek ☐ Kurzus típusok

Mentés

5.12. ábra. Az importálás/exportálás oldala

6. fejezet

Összefoglalás

Büszkén jelenthetem ki, hogy az elkészült sillabusz kezelő alkalmazás teljes egészében megfelel az összes vele szemben támasztott követelménynek. Az alkalmazás fejlesztése közben sikerült elérnem egy olyan szoros integrációt magával a Liferay tartalomkezelő rendszerrel, hogy az alkalmazás telepítése után szinte senki sem tudná megmondani, hogy az alkalmazás nem beépített portlet volna.

Az alkalmazás fejlesztése közben számtalan új tudásra tettem szert. Egyrészt megtanultam egy mindenki által elismert, már több mint tizenhét éve a piacon lévő és széles körben használt tartalomkezelő rendszer használatát és a rendszerre történő fejlesztés különböző módszereit, másrészt pedig olyan problémákkal találtam szemben magamat az elmúlt két évben, amelyekkel éles körülmények között is találkozhatnak a szoftverfejlesztők. A legnagyobb probléma, amelyet le kellett küzdenem a verziómigráció volt, ugyanis mire megtanultam a Liferay 6.2-re való fejlesztés folyamatát és otthon éreztem magam benne, addigra megjelent a 7.0 verzió rengeteg újítással, ezért egy csomó mindent újra kellett írnom az alapvető változtatások miatt. Ennek köszönhetően megtanultam használni a Gradle, a Blade CLI eszközöket, elsajátítottam az Apache Velocity és Apache FreeMarker templating motorok [13] használatát, jobban megértettem, hogy miért van szükség egy szolgáltatásokat nyújtó interfész kialakítására egy ilyen vagy ehhez hasonló alkalmazások esetén, otthonosabban érzem magam JavaScript kódok írása közben, jobban rálátok a biztonsági kérdésekre, illetve elsajátítottam az Alloy UI keretrendszer [16] használatának alapjait.

Természetesen, mint minden alkalmazás ez sem teljes. Mindig mindent tovább lehet fejleszteni, ez igaz a sillabusz kezelő alkalmazásra is. Talán az egyik lehetséges továbbfejlesztési lehetőség az Application Display Template-ek (ADT) támogatása lenne, amely lehetővé tenné,

hogyan az Asset Publisher-en keresztül megjelenített sillabuszok kinézetét futás időben, az oldal adminisztrátorai határoznák meg egy egyszerű script nyelv segítségével. Az ADT támogatja mind az Apache Velocity, mind az Apache FreeMarker templating motorokat, így mindenki tetszés szerint azt használhatja megjelenítő sablonok készítésére, amelyik szimpatikusabb számára.

Egy másik továbbfejlesztési lehetőség a portálba épített keresőmotor integráció lehet. Ahhoz, hogy a tartalmak között keresni lehessen, az adatbázisban lévő adatok megfelelő módon történő indexelésére van szükség, amelyre a Liferay szintén egy saját fejlesztésű keretrendszert biztosít a fejlesztők számára.

Harmadik továbbfejlesztési lehetőségként az általam kialakított webszolgáltatások esetén a jogosultságkezelés átalakítását tudom javasolni. Jelenleg van olyan része a szolgáltatásoknak, amelyek akkor is meghívhatóak, amikor a szolgáltatást megszólító felhasználó nincs feljogosítva a szolgáltatás használatára. A szolgáltatások esetén az egyedek hozzáadása, frissítése és törlése minden esetben le van védve az illetéktelen hozzáférés ellen, viszont az adatok lekérdezésére szolgáló interfészek során csak egy részük esetén lett jogosultságkezelés kialakítva. Amikor még ezeket a szolgáltatásokat alakítottam ki, még úgy gondoltam, hogy ilyen tantárgyi adatok lekérdezéséből nem lehet baj, de mivel a felhasználói felület esetén minden esetben megtörténik az összes jogosultság ellenőrzése, ezért ma már jobbnak látom, ha a szolgáltatások esetén is mindenhol ellenőrzésre kerülnek ezen hozzáférési jogok.

Köszönetnyilvánítás

Ez a diplomamunka nem jöhetett volna létre a családom, a barátaim, az egyetem, a témavezetőm, illetve a Liferay Hungary Kft. támogatása nélkül, ezért szeretném megragadni az alkalmat, hogy mindenkinek köszönetet mondjak, aki az elmúlt években mellettem állt és lehetőséget teremtett arra, hogy ez a diplomamunka elkészülhessen.

Köszönöm a családomnak, amiért az elmúlt két évben mindent meg tettek azért, hogy idáig eljuthassak az életben, illetve mindazért, amiért mindvégig kitartottak mellettem.

Köszönöm a barátaimnak is, közülük legfőképpen László Zsoltnak, aki néhány évvel ezelőtt bevezetett a \LaTeX szövegformázó rendszer használatába és rendelkezésemre bocsátotta egy általa készített korábbi dokumentum vázlatát.

Végül szeretném megköszönni témavezetőmnek, Dr. Adamkó Attilának, amiért lehetőséget teremtett arra, hogy díjmentesen részt vehessek a Liferay Hungary Kft. által tartott *Developing for the Liferay Platform 1* című képzésen, illetve köszönet illeti mindazért a maradék támogatásáért is, amit az elmúlt néhány évben nyújtott számomra.

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>
- [9] Gradle
<https://gradle.org/>

- [10] **Blade CLI**
https://dev.liferay.com/develop/tutorials/-/knowledge_base/7-0/blade-cli
- [11] **Apache Ant**
<http://ant.apache.org/>
- [12] **Apache Maven**
<https://maven.apache.org/>
- [13] **Apache Velocity**
<http://velocity.apache.org/>
- [14] **Apache FreeMarker**
<http://freemarker.org/>
- [15] **jQuery**
<https://jquery.com/>
- [16] **Alloy UI**
<http://alloyui.com/>

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