MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Rozšíření a údržba rezervačního systému laboratoře LEMMA

BAKALÁŘSKÁ PRÁCE

Pavel Jelínek

Brno, jaro 2008

Prohlášení

Prohlašuji, že tato bakalářská práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Vedoucí práce: RNDr. Petr Sojka, Ph.D.

Shrnutí

Cílem mé bakalářské práce bylo rozšíření a údržba rezervačního systému laboratoře LEMMA[1]. Do systému byly přidány nové funkce, které usnadní správu vybavení laboratoře jejím pracovníkům a taktéž zajistí vyšší komfort uživatelům systému. Mezi nejvýznamnější mnou provedená vylepšení patří:

- vypůjčované položky lze slučovat do větších celků;
- nová role v systému – výdejář vybavení, s více možnostmi pro správu rezervací;
- zaznamenávání historie výpůjček pro pozdější potřeby laboratoře;
- možnost sledování aktuálních stavů rezervací;
- v neposlední řadě uživatele jistě potěší vylepšený vzhled systému přinášející též lepší ovladatelnost.

V prvních kapitolách se práce věnuje studii původního systému, popisu jeho částí a použitých technologií. Následují analýza a implementace provedených rozšíření a v příloze lze nalézt uživatelský návod na práci se systémem.

1. `http://www.fi.muni.cz/lemma/`

Klíčová slova

laboratoř LEMMA, rezervační systém, dokumentace, webová aplikace, Java

Obsah

Kapitola 1

Úvod

1.1 Důvody vzniku rezervačního systému

LEMMA (Laboratoř elektronických multimediálních aplikací Fakulty informatiky) se specializuje na technologie zpracování a publikování rozsáhlých textových a multimediálních kolekcí dat včetně produkce a postprodukce filmů. V současné době je LEMMA spojena s předměty PV110 Softwarové elektronické publikace I a navazujícím předmětem PV113 Softwarové elektronické publikace II, jejichž náplní je filmová tvorba s podporou počítače. Studenti těchto předmětů potřebují pro své projekty využívat vybavení laboratoře (dále jen zdroje). Počet studentů těchto předmětů značně převyšuje počet dostupných zdrojů. Proto byl vytvořen rezervační systém, ve kterém mají studenti možnost zjistit dostupnost zdrojů a v dostatečném předstihu rezervovat zdroje, které pro svoji tvorbu potřebují.

Druhým důvodem pro vytvoření systému byla povinnost studenta vyplnit výpůjční formulář, kterým stvrzuje převzetí hmotné zodpovědnosti za vypůjčené zdroje mimo území školy. Systém nepovolí vypůjčení zdroje bez předchozího vyplnění formuláře.

1.2 Důvody rozšiřování systému

Zřejmě u každého většího systému tohoto typu vznikají v průběhu jeho používání nové požadavky na funkcionalitu systému. Nejinak tomu je i u rezervačního systému LEMMA. Původní rezervační systém sice umožňoval evidovat žádosti o rezervování zdrojů, již ale neposkytoval možnost sledovat skutečný průběh rezervace. Taktéž nebylo možné dostatečně kontrolovat a ovlivňovat chování uživatelů. Především nadměrné využívání zdrojů některými uživateli negativně ovlivňovalo práci ostatních uživatelů. Dalším důvodem pro rozšíření systému byla nutnost zaznamenávat využívání zdrojů pro pozdější vykazování činnosti laboratoře.

Během vývoje systému vznikaly neustále další náměty na jeho rozšíření či vylepšení (což je typické pro každý větší systém) [2], podstatná část z nich byla

implementována, i tak je ovšem v systému stále velký prostor pro jeho zdokonalování.

1.3 Prostředky užité při práci

Pro úpravu systému jsem využil vývojové prostředí NetBeans[1] ve verzi 6, které mi výrazně usnadnilo práci na systému a také jeho testování. Pro vytvoření ERD diagramu jsem použil program Case Studio 2[2], pro vysázení této práce systém LaTeX[3] s využitím stylu fithesis[4].

1. http://www.netbeans.org/
2. http://www.casestudio.com/
3. http://www.latex-project.org/
4. http://www.fi.muni.cz/~xpavlov/fithesis/

Kapitola 2

Použité technologie

Všechny technologie použité v souvislosti s rezervačním systémem LEMMA jsou dle mého názoru obecně dobře známé, tudíž si u každé uvedeme jen stručný popis a případné postřehy z vlastních zkušeností.

2.1 Java

Rezervační systém LEMMA je webová aplikace napsaná v programovacím jazyce Java[1]. Diskutování výhod a nevýhod tohoto jazyka přenecháme jiným pracím. Jako největší přínosy Javy pro vývoj rezervačního systému LEMMA bych zmínil velmi rozsáhlou a snadno přístupnou dokumentaci. Dále pak objektový přístup k programování, který mi výrazně usnadnil rozšiřování a úpravy systému při zachování jeho původní funkčnosti. Pro mě ale největší výhoda Javy spočívala v přenositelnosti aplikací napsaných v Javě mezi různými platformami, díky níž jsem mohl systém vyvíjet a testovat na odlišné platformě, než na jaké systém v reálném provozu běží. Mnohdy je za největší nevýhodu aplikací napsaných v Javě považována jejich větší hardwarová náročnost a s tím spojená nižší rychlost, já jsem se ale s tímto problémem v průběhu práce nesetkal.

2.2 Apache Tomcat

Systém pracuje na webovém serveru Apache Tomcat[2] verze 5.5. Jeho hlavní výhodou je, že vzniká pod licencí Open Source a je certifikován jako plně J2EE (Java 2 Enterprise Edition) kompatibilní. Správu aplikací spuštěných na serveru výrazně usnadňuje webové rozhraní služby Tomcat Web Application Manager.

---

1. `http://www.sun.com/java/`
2. `http://tomcat.apache.org/`

2.3 PostgreSQL

Data systému jsou ukládána do PostgreSQL databáze. Tato databáze ve spojení s funkcí Reverse engineering programu Case Studio 2 [3] mi velmi pomohla při studii původního systému. Jedná se o velmi vyspělou databázi s kvalitní podporou pro její uživatele. Další výhodou je licence, pod kterou je PostgreSQL distribuována (jde o Open Source software s licenčními podmínkami BSD), díky níž nebyl problém s instalací databáze na můj vlastní počítač. Jako velmi užitečný se při práci s databází ukázal nástroj pgAdmin III[3].

3. http://www.pgadmin.org/

Kapitola 3

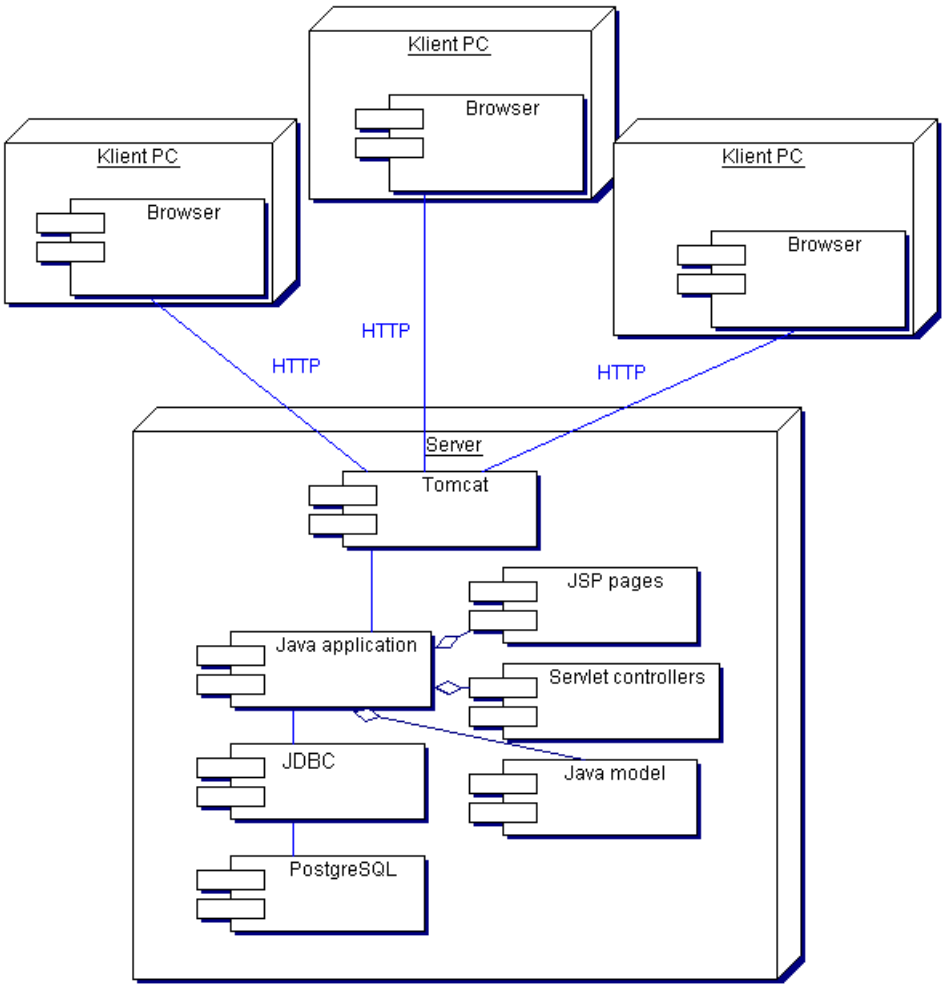
Architektura stávajícího systému

Architektura rezervačního systému LEMMA implementuje návrhový vzor Model-View-Controller (MVC). MVC (někdy také označovaná jako Model-2) je softwarová architektura, která rozděluje datový model aplikace, uživatelské rozhraní a řídicí logiku do tří nezávislých komponent tak, že modifikace některé z nich má minimální vliv na ostatní [1]. V našem případě tento vzor implikuje rozdělení na následující části:

- **Model:** jádro aplikace reprezentující informace (data), s nimiž aplikace pracuje, a metody určené pro práci s těmito daty. Data modelu v podstatě mapují objekty uložené v databázi aplikace. Model bude dále popsán v sekci 3.1 na následující straně;
- **View:** pohled využívající JSP[1] stránky, které dynamicky převádí data reprezentovaná modelem do podoby vhodné k interaktivní prezentaci ve webovém prohlížeči uživatele. K tomuto účelu používá metody modelu. Více v sekci 3.2 na straně 21;
- **Controller:** servlety reagující na události vyvolané uživateli. Servlety, s využitím metod modelu, dle těchto událostí zajistí změny v datech modelu nebo v pohledu. Při ukončení servletu se vždy volá příslušná JSP obstarávající formátování odpovědi. Controller, vzhledem k jeho úzkému propojení s pohledem, bude také popsán v sekci 3.2 na straně 21.

1. JavaServer Pages `http://java.sun.com/products/jsp/`

Mapování architektury softwaru na architekturu hardwaru demonstruje diagram nasazení:



Obrázek 3.1: Mapování architektury softwaru na architekturu hardwaru

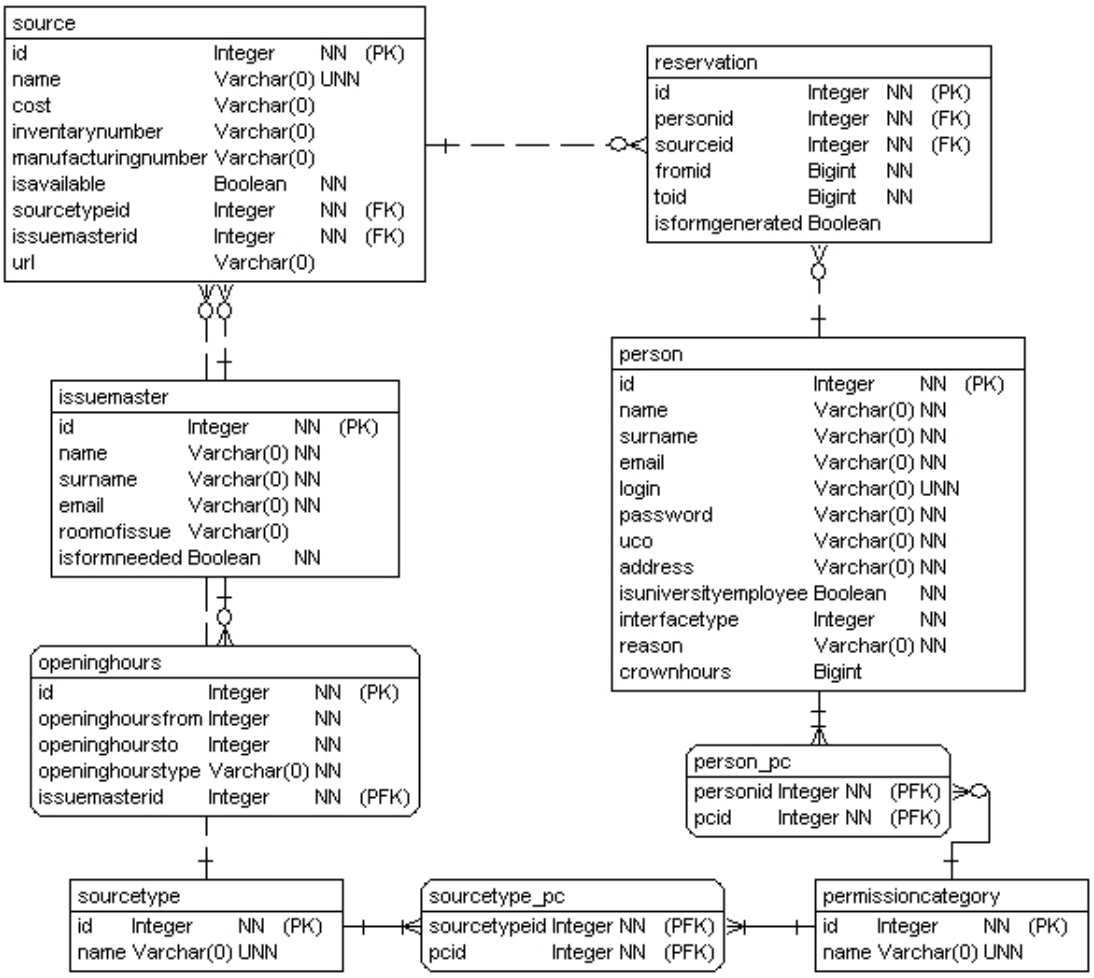
3.1 Jádro systému

Jádro systému obstarává mapování persistentních objektů aplikace do relační databáze. Jádro taktéž poskytuje metody pro manipulaci se všemi objekty aplikace pro servlety a JSP. Jádro si pro přehlednost rozdělíme na tři části.

V sekci 3.1.1 bude popsán způsob uložení persistentních objektů aplikace. Metody přistupující do databáze lze nalézt v sekci 3.1.2 na straně 11. Všechny objekty aplikace a jejich metody, které nepřistupují do databáze, budou popsány v sekci 3.1.3 na straně 15.

3.1.1 Datový model

Následující obrázek ukazuje entitně-relační diagram (dále jen ERD) objektů uložených v databázi aplikace:



Na výše uvedeném ERD nejsou pro přehlednost zobrazeny názvy relací. Názvy relací a jejich význam v systému si vysvětlíme na následujících řádcích. Taktéž si zde uvedeme základní popis objektů, které jednotlivé entity z ERD v systému reprezentují.

Entita **person** uchovává informace o všech osobách, které mohou s rezervačním systémem pracovat. Jejím primárním klíčem a jednoznačným iden-

tifikátorem je atribut *id*. Při vkládání nového záznamu (nové osoby) do této entity je atribut *id* vygenerován uloženou sekvencí *s_id_person*. Uložené sekvence slouží v databázi PostgreSQL jako automatické číselné generátory. Sekvence vygeneruje nové číslo vždy po zavolání funkce `nextval('název_sekvence')`. V našem případě se bude jednat o poslední vygenerované číslo inkrementované o 1. Tímto způsobem se v databázi rezervačního systému generují všechny primární klíče (vždy označované jako *id*), a proto se o nich u dalších entit již nebudeme zmiňovat. Dalším omezením atributů entity **person** je požadavek unikátnosti atributu *login*. Unikátnost tohoto atributu je vyžadována z důvodu rozlišení jednotlivých osob při přihlašování do systému.

Entita **person** je v relaci s entitou **permissioncategory**, která reprezentuje kategorie oprávnění v systému. Jedná se o relaci s kardinalitou N:M. Pro uchování informací o přidělených kategoriích oprávnění osobám slouží vazební entita **person_pc**. Entita **person** je dále v relaci s entitou **reservation**. Tato relace o kardinalitě 1:N udává, které rezervace vlastní jaká osoba. Sama entita **reservation** pak nese informace o všech rezervacích v systému. Do každé rezervace vstupuje mimo jedné osoby také jeden zdroj. Tento vztah je v datovém modelu vyjádřen relací mezi entitami **reservation** a **source**.

Entita **source** uchovává informace o všech zdrojích laboratoře v systému. Zdroje se dále člení podle svého účelu do skupin nazvaných druhy zdrojů. Informace o druzích zdrojů jsou uloženy v entitě **sourcetype**. Každý zdroj musí být zařazen právě do jednoho druhu zdroje. Tento vztah vyjadřuje relace mezi entitami **source** a **sourcetype**. Druhy zdrojů mají, stejně jako osoby, přiděleny kategorie oprávnění. Informace o tom, které kategorie oprávnění byly přiděleny kterému druhu zdrojů, nese vazební entita **sourcetype_pc**.

Dále je v databázi aplikace uložena entita **issuemaster**, která nese informace o výdejářích zdrojů. Výdejář zdrojů reprezentuje osobu, která rezervované zdroje půjčuje a přebírá zpět. O každý zdroj se stará právě jeden výdejář, což značí relace mezi entitami **source** a **issuemaster**. Entita **openinghours** uchovává informace o výpůjční době výdejářů, což je doba, ve kterou se mohou rezervované zdroje u výdejářů vypůjčit či vrátit. Jak udává relace mezi entitami **issuemaster** a **openinghours**, každý výdejář může mít více výpůjčních dob.

Poslední entitou v databázi rezervačního systému LEMMA, která ovšem není uvedena v ERD diagramu, je entita **reservationsystem**. Tato entita uchovává nastavení systému a není v relaci s žádnou jinou entitou systému. Atributy této entity může měnit pouze administrátor při konfiguraci systému.

Do datového modelu lze taktéž zařadit třídu **Constant**, ve které jsou uloženy cesty k externím souborům využívaných systémem, dále pak údaje pro přístup do databáze a některá nastavení systému. Tato třída odráží informace uložené v souboru *lemma.properties*. Při vstupu na úvodní stránku systému se načtou

aktuální informace z tohoto souboru, uloží jako atributy třídy **Constant** a pak s nimi systém může pracovat. Jedná se o tyto údaje:

- **conPath:** cesta k databázi zadávaná JDBC ovladači při vytváření spojení do databáze;
- **conName:** uživatelské jméno, pod kterým se systém přihlašuje do PostgreSQL databáze;
- **conPassword:** heslo pro přihlášení do databáze aplikace;
- **useSSL:** hodnota typu `boolean` udávající, zda se při komunikaci s databázovým strojem má použít zabezpečný protokol SSL (pro komunikaci s databází v ostrém provozu se protokol SSL používá vždy, vypnout jeho používání se může hodit pro ladění aplikace, testování či vývoj na lokálním počítači);
- **xslPath:** cesta k adresáři, v němž jsou uloženy konfigurační a transformační soubory pro procesor FOP, který v systému obstarává generování výpůjčního formuláře;
- **mailSmtpHost:** dostupný SMTP server, který zajistí odesílání emailů generovaných rezervačním systémem;
- **mailFromSystem:** řetězec, který je používán jako odesílatel v emailech generovaných rezervačním systémem;
- **urlPath:** URL adresa systému uváděná v emailech odesílaných systémem pro snažší dostupnost systému pro uživatele.

3.1.2 Metody pracující s databází

Každá entita databáze je v systému reprezentována stejně pojmenovanou třídou. Tyto třídy odrážejí objekty, se kterými systém pracuje. O těchto objektech se dozvíme více v následující sekci, zde si popíšeme ty metody objektů, které pro svoji práci vyžadují připojení do databáze.

Veškeré metody využívající ve svém těle připojení do databáze se pro snadnější správu systému slučují do tříd nazývaných fasády (anglicky facade). Nikde jinde v systému, kromě těchto fasád, by se nemělo vyskytovat připojení do databáze ani žádné SQL[2] dotazy.

Všechny třídy fasád jsou z důvodu snahy o odstranění redundance v kódu potomci abstraktní třídy `AbstractFacade`. Od této třídy dědí fasády následující metody pro navázání spojení s databází a provedení SQL dotazu:

2. Structured Query Language `http://www.sql.org/`

- *getCon()* vrací spojení do databáze, které naváže ve svém těle;
- *castSQL (String sqlString)* je metoda určená pro provedení SQL dotazu. Dotaz se metodě předává jako parametr **sqlString**, který musí být správně formulovaným dotazem v jazyce SQL.

`PersonFacade` je fasáda slučující metody, jejichž cílem je správa dat entity **person**. Fasáda obsahuje základní metodu pro vkládání objektů typu `Person` (dále jen osob) s názvem *savePerson(Person person)*, která kromě uložení parametru **person** zajistí i uložení oprávnění osoby do entity **person_pc**. Fasáda dále obsahuje metodu *deletePerson(Person person)*, která smaže osobu zadanou parametrem **person**, a metodu *updatePerson(Person person)*, která provede úpravu atributů osoby dle zadaného parametru **person**. Následuje seznam zbylých metod fasády `PersonFacade`, kde většina těchto metod slouží ke specializovanému vyhledávání osob v systému dle různých kritérií:

- *findByLogin(String personLogin)* vyhledá a vrátí osobu, která má atribut **login** shodný se zadaným parametrem metody;
- *findById(int id)* přetěžuje následující metodu;
- *findById(String id)* zajišťuje vyhledání a vrácení osoby s atributem **id** rovným parametru metody;
- *allPersons()* vrátí seznam (kolekci `List`) všech osob v databázi aplikace;
- *allPersonsInSystem()* seznam (`List`) osob zaregistrovaných v systému, což jsou všechny osoby s atributem **interfacetype** různým od 0;
- *findByInterfaceType(int interfaceType)* vyhledá a vrátí seznam (`List`) osob s atributem **interfacetype** rovným parametru metody;
- *degradeUsers()* změní atribut **interfacetype** na hodnotu 1 všem osobám v databázi, jež měly tento atribut nastavený na hodnotu 2.

`PermissionCategoryFacade` je fasáda zahrnující metody, které pracují s daty entity **permissioncategory**. Fasáda obsahuje metodu *savePC(PermissionCategory pCategory)* pro ukládání objektů typu `PermissionCategory` (dále jen kategorie oprávnění). Tato metoda zajistí kromě uložení nové kategorie oprávnění i uložení vztahů se všemi druhy zdrojů, které jsou do této kategorie oprávnění zařazeny. Další metodou této fasády je *updatePC(PermissionCategory pCategory)*, která upravuje kategorii oprávnění dle parametru **pCategory**,

což znamená, že může i měnit vztahy této kategorie oprávnění s druhy zdrojů. Dále fasáda obsahuje metodu pro mazání kategorií oprávnění *deletePC(PermissionCategory pCategory)*. Zbylé metody této fasády jsou metoda *allPermissionCategories()* vracející seznam (`List`) všech kategorií oprávnění v systému a přetížená metoda *findById()*, jejíž parametr **id** může být typu `String` či `int`, vracející nalezenou kategorii oprávnění.

Další fasádou v systému je `SourceFacade`, která pracuje převážně s daty z entity **source** (dále jen zdroji). Kromě klasických metod pro vkládání zdrojů (*saveSource(Source source)*), mazání (*deleteSource(Source source)*) a upravování zdrojů (*updateSource(Source source)*) obsahuje tyto metody:

- *findById(int id)* přetěžuje následující metodu;
- *findById(String id)* vyhledá zdroj podle zadaného parametru **id**;
- *getAvailableSourcesForPerson(Person person)* vrátí seznam (`List`) zdrojů, které jsou ve stejné kategorii oprávnění jako osoba z parametru metody a zároveň jsou dostupné;
- *allSourceForAdminOrder(String sql)* je metoda, která vrací všechny zdroje (jako `List`) seřazené podle SQL výrazu v řetězci, jenž je parametrem metody;
- *allSources()* vrací seznam (`List`) veškerých zdrojů v systému;
- *sourcesOutOfStore()* vrací aktuálně vypůjčené zdroje, a to ve formě dvojic (`Map`) klíč a hodnota, kde klíčem je vždy **id** vypůjčeného zdroje a hodnotou osoba, která si zdroj vypůjčila.

`SourceTypeFacade` je fasáda obsahující metody, které pracují s daty entity **sourcetype** (dále jen druhy zdrojů). Fasáda obsahuje metodu pro ukládání druhů zdrojů *saveSourceType(SourceType sourceType)*, metodu pro mazání druhů zdrojů *deleteSourceType(SourceType sourceType)* a metodu pro úpravu druhů zdrojů *updateSourceType(SourceType sourceType)*. Navíc fasáda obsahuje pouze dvě metody pro práci s druhy zdrojů, a to metodu *findById(String id)*, která vyhledá druh zdrojů podle parametru **id**, a metodu *allSourceTypes()*, která vrací seznam (`List`) všech druhů zdrojů uložených v databázi.

`IssueMasterFacade` je fasáda obsahující metody, které pracují nejen s daty entity **issuemaster**, ale vzhledem k velké provázanosti výdejářů a jejich otevíracích hodin i s daty entity **openinghours**, která tudíž nemá svoji vlastní

fasádu. `IssueMasterFacade` obsahuje metodu *saveIssueMaster(IssueMaster issueMaster)*, která zajišťuje uložení objektu typu `IssueMaster` (dále jen výdejář), který je metodě předán jako její parametr. Kromě uložení výdejáře poskytuje metoda možnost uložení otevíracích hodin výdejáře, pokud byly pro tohoto výdejáře nějaké zadány. Dále tato fasáda obsahuje metodu *updateIssueMaster(IssueMaster issueMaster)* pro úpravu atributů výdejáře a metodu *deleteIssueMaster(IssueMaster issueMaster)*, která kromě smazání výdejáře smaže i všechny výdejní hodiny patřící výdejáři z parametru metody. Ostatní metody fasády jsou:

- *findById(String id)* vyhledá výdejáře dle zadaného parametru **id**;
- *allIssueMastersIntoMap()* vrátí dvojice klíč a hodnota (`Map`) pro každého výdejáře v systému, kde klíč vždy odpovídá atributu **id** a hodnota výdejáři s daným **id**, metoda se používá kvůli efektivitě při rezervaci zdrojů;
- *allIssueMasters()* vrací seznam (`List`) všech výdejářů v systému;
- *insertOpeningHourToIssueMaster(IssueMaster issueMaster, OpeningHours oh)* uloží do entity **openinghours** otevírací hodiny z parametru **oh** a jako jejich vlastníka nastaví výdejáře z parametru **issueMaster**, kterému poté tyto otevírací hodiny přidá do atributu **openingHoursId**, a nakonec metoda vrátí tohoto výdejáře;
- *deleteOpeningHourFromIssueMaster(IssueMaster issueMaster, OpeningHours oh)* smaže otevírací hodiny z parametru **oh** a vrátí výdejáře z parametru **issueMaster** již bez těchto otevíracích hodin;
- *setUpOpeningHours (IssueMaster issueMaster)* nastaví výdejáři z parametru metody otevírací hodiny do atributu OpeningHoursId, tato metoda existuje z důvodu ušetření režie při volání konstruktoru výdejáře, který je možno volat bez nutnosti nastavení otevíracích hodin výdejáře.

`ReservationFacade` je fasáda obsahující metody, které pracují jak s daty entity **reservation**, tak s daty entit **person** a **source**, jelikož na každé rezervaci se podílí jedna osoba a jeden zdroj. Fasáda obsahuje metodu *saveReservation(Reservation reservation)* pro ukládání rezervací a metodu *deleteReservation(Reservation reservation, float crownHoursPercFreeCoef)*, která kromě smazání rezervace z parametru **reservation** zajistí i snížení atributu **crownhours** u osoby vlastnící tuto rezervaci o hodnotu této rezervace vynásobenou koeficientem z parametru **crownHoursPercFreeCoef**. Dále fasáda obsahuje tyto metody:

- *findById(int id)* vyhledá a vrátí rezervaci, jejíž **id** je shodné s parametrem této metody;
- *findByFromRange(long from, long to)* najde a vrátí seznam (`List`) rezervací, které začínají v rozmezí daném parametry **from** a **to**;
- *findActualByPerson(Person personId)* vyhledá a vrátí seznam (`List`) rezervací, které patří osobě z parametru této metody a jejichž konec (vyjádřený atributem **toid**) je větší než aktuální čas systému;
- *allReservationsMap()* vrací dvojice klíč a hodnota (`Map`), kde klíčem je **id** rezervace a hodnotou je pak samotná rezervace s tímto **id**;
- *actualReservations()* vrátí seznam (`List`) aktuálních (ještě neskončených) rezervací;
- *allReservations()* vrátí seznam (`List`) všech rezervací.

Poslední fasádou systému je `ReservationSystemFacade`, která pracuje s nastavením systému uloženým v entitě **reservationsystem**. Tato fasáda obsahuje pouze dvě metody, a to metodu *getSystem()* vracející objekt typu `ReservationSystem` a metodu *updateReservationSystem(ReservationSystem lemma)*, která ukládá nastavení systému, jenž nese parametr této metody ve svých atributech.

3.1.3 Objektový model

V předchozích sekcích jsme si vysvětlili, jak systém ukládá data do databáze a metody určené pro práci s těmito daty. V této sekci budeme pohlížet na tato data s větší mírou abstrakce a popíšeme si, jakými objekty jsou tato data v systému reprezentována, jaké mají tyto objekty vlastnosti (atributy) a jaké úkoly mohou tyto objekty v systému vykonávat (metody objektů). Co které objekty systému reflektují z reálného světa, bude popsáno v kapitole 4 na straně 25.

Každý jednotlivý objekt vystupující v systému je instancí nějaké třídy. Na následujících řádcích si tedy vyjmenujeme všechny třídy v systému a popíšeme objekty, které vzniknou instancováním těchto tříd.

Třída `Person`. Objekty třídy `Person` odrážejí v systému data uložená v entitě **person**. Tyto objekty představují veškeré osoby, které mohou s rezervačním systémem pracovat.

Každá osoba, která žádá o přístup do systému, se musí nejdříve zaregistrovat. Při registraci je každé osobě přidělen primární klíč, který danou osobu

v systému jednoznačně identifikuje. Primární klíč nese atribut *id* a jeho hodnotu určuje sama databáze tak, aby byl v systému jedinečný (způsob generování hodnot atributů *id* byl vysvětlen v sekci 3.1.1 na straně 9).

Dalším atributem, který je vyplňován systémem, je *interfaceType*. Tento atribut slouží k určování rolí v systému a může nabývat následujících hodnot:

0 – **Host** (více v sekci 4.1.1 na straně 26);

1 – **Uživatel s omezením** (více v sekci 4.1.2 na straně 26);

2 – **Uživatel** (více v sekci 4.1.3 na straně 27);

3 – **Administrátor** (více v sekci 4.1.4 na straně 27).

Poslední atribut, který vyplňuje samotný systém, tvoří atribut *crownhours*. Tento slouží k určování využívanosti systému danou osobou. Při každé rezervaci osoby se zvýší hodnota jejího atributu *crownhours* o cenu rezervovaného zdroje vynásobenou počtem hodin, na který si zdroj rezerovovala. Vysoká hodnota tohoto atributu svědčí o velké využívanosti zdrojů laboratoře danou osobou. Administrátoři mají poté možnost zrušit další rezervace této osoby a umožnit tak i ostatním osobám v systému využívat zdroje laboratoře dle jejich potřeb.

Všechny další údaje o osobě, které odpovídají atributům entity **person**, je každý žadatel o přístup do systému povinen vyplnit při registraci, tudíž žádný atribut osoby nesmí nést hodnotu `null`. Význam všech atributů, které zde nebyly vysvětleny, plně koresponduje s názvy těchto atributů v entitě **person**. Jen pro upřesnění, atribut *login* musí být v systému unikátní, využívá se při přihlašování a systém podle něj zjistí, která osoba přistupuje do systému. Atribut *reason* vyplňuje žadatel o přístup do systému tak, aby jasně vypovídal o důvodu žádosti o přístup do systému. Administrátor pak dle tohoto údaje povolí nebo zamítne danou žádost o přístup do systému.

Ostatní atributy osob, které nejsou uloženy v entitě **person**, jsou *pCategories*, ve kterém je uložen seznam (`List`) kategorií oprávnění přiřazených této osobě, a atribut *reservationRequest*, v němž je uložena žádost (objekt typu `ReservationRequest`) o rezervaci zdojů danou osobou. Tato žádost je dále zpracována metodami osoby, jež jsou:

- *tryToReserve()* je metoda osoby, která kontroluje bezkonfliktnost žádosti o rezervaci zdrojů této osoby. To znamená, že metoda rozliší zdroje z žádosti této osoby na ty, které je možné rezervovat, a na ty, které jsou v požadovaný čas již rezervované, nebo požadovaný čas neodpovídá otevíracím hodinám výdejářů těchto zdrojů. Metoda toto rozlišení provede následovně: projde v cyklu všechny zdroje z žádosti o rezervaci a po-

mocí metod této žádosti (viz následující objekt – `ReservationRequest`) rozdělí zdroje do atributů žádosti:

- zdroje, které je možné zarezervovat, jsou zařazeny do atributu *noConflicts* a jejich rezervace je provedena v metodě *reserve()*;
- zdroje, které jsou v konfliktu s jinou rezervací, se přidají do atributu *reservationConflicts* a osobě požadující rezervaci těchto zdrojů je oznámeno, proč je tato žádost neuskutečnitelná;
- zdroje, které není možné rezervovat z důvodu konfliktu s otevíracími hodinami výdejářů, jsou zařazeny do atributu *issueMasterConflicts* a osobě požadující rezervaci těchto zdrojů je nabídnuta možnost nechat si žádost přizpůsobit dle otevíracích hodin výdejářů (zajistí metoda *adjustForIssueMastersOpeningHours()* této žádosti);

- *reserve()* slouží k zarezervování zdrojů z žádosti o rezervaci, které byly zařazeny metodou *tryToReserve()* do atributu *noConflicts* této žádosti. Dále tato metoda zajistí odeslání e-mailu (pomocí metody *sendMail()* objektu **Mailer**) informujícího výdejáře zdrojů o nové rezervaci.

Třída `ReservationRequest`. Objekty této třídy vystupují v systému jako **žádosti o rezervaci zdrojů** a jedná se o jedny z nejdůležitějších objektů systému. Vždy předtím, než je jakákoliv rezervace v systému uložena do databáze, je vytvořen objekt typu `ReservationRequest` a přiřazen osobě, která tuto žádost o rezervaci zdrojů podala, jako její atribut *reservationRequest*. Pomocí metody osoby *tryToReserve()* jsou poté volány metody této žádosti o rezervaci zdrojů, které zjistí, zda je tato žádost bezkonfliktní. Teprve poté, co metoda osoby *tryToReserve* zjistí, že je tato žádost bezkonfliktní, se mohou uložit požadované rezervace zdrojů do databáze.

Objekty typu `ReservationRequest` se vyskytují pouze jako atributy osob, jež tyto žádosti podaly a jsou tudíž jejich majiteli. Atributy objektů typu `ReservationRequest` jsou *from* a *to* (oba typu `WildDate`) s významem začátku a konce žádosti o rezervaci, dále pak atribut *sources*, ve kterém je uložena množina (`Set`) zdrojů, které si osoba vlastnící tuto žádost chce rezervovat.

Následující atributy se nastavují až po zavolání metod tohoto objektu, které zjišťují, zda nejsou zdroje z této žádosti v konfliktu s jinými rezervacemi, či otevíracími hodinami výdejářů. Jde o atributy *noConflicts*, což je množina (`Set`) obsahující zdroje, jenž nejsou v žádném konfliktu, dále atribut *reservationConflicts*, který mapuje (typ `Map`, klíčem je zdroj, hodnotou rezervace) rezervace na zdroje, které jsou spolu v konfliktu. Posledním atributem je mapa (typ `Map`,

klíčem je zdroj, hodnotou výdejář) *issueMasterConflicts*, která mapuje konflikty mezi požadovanými zdroji a otevíracími hodinami výdejářů.

O kontrolu, zda jsou požadované zdroje v některém z výše uvedených konfliktů, se starají následující metody objektů `ReservationRequest`:

- *collideWithReservationTime(Reservation reservation)* je metoda zjišťující, zda je žádost, na které byla tato metoda zavolána, v konfliktu s rezervací, jež byla této metodě předána jako parametr **reservation**. Žádost o rezervaci zdroje se dostane do konfliktu s jinou rezervací téhož zdroje, pokud tato rezervace zasahuje do časového úseku vymezeného atributy žádosti *from* a *to*. Pokud je zjištěn konflikt, metoda vrací `true`, jinak vrací `false`;

- *collideWithIssueMasterOpeningHours(IssueMaster iMaster)* kontroluje, zda oba z atributů *from* a *to* žádosti o rezervaci zdrojů, na které byla tato metoda zavolána, spadají do otevíracích časů výdejáře z parametru **iMaster** této metody. Pokud nespadají oba tyto atributy do otevíracích časů výdejáře, pak metoda vrátí `true`, jelikož nastal konflikt, jinak vrací metoda `false`.

O přizpůsobení časů žádostí o rezervace, které jsou v konfliktu s otevíracími časy výdejářů, se stará následující metoda:

- *adjustForIssueMastersOpeningHours()* přizpůsobí (lépe řečeno zkrátí) čas žádosti o rezervaci zdrojů tak, aby začátek i konec této žádosti spadaly do otevíracích hodin výdejářů zdrojů z této žádosti.

Třída `IssueMaster`. Objekty třídy `IssueMaster` reprezentují data uložená v entitě **issuemaster**. Tyto objekty představují v systému výdejáře zdrojů. Výdejář zdrojů se stará o vydávání zdrojů a přebírání vypůjčených zdrojů nazpět. V původním systému není výdejář žádnou z rolí v systému, ani nemá žádné jiné možnosti jak pracovat se systémem. Tudíž výdejáři, kteří jsou skutečnými osobami, musí být v systému vedeni též jako uživatelé systému, aby mohli mít přístup do systému a sledovat rezervace zdrojů, které jsou v jejich správě. Výdejáře do systému přidává administrátor.

Atributy výdejářů odpovídají atributům entity **issuemaster** z databáze systému. Atribut *roomOfIssue* nese informaci o místnosti, ve které daný výdejář vydává nebo přijímá zdroje. Atribut *isFormNeeded* má hodnotu `true`, pokud je pro zapůjčení zdrojů od daného výdejáře nutné nechat si vygenerovat výpůjční formulář. Následným podepsáním tohoto formuláře uživatel stvrzuje převzetí hmotné zodpovědnosti za vypůjčené zdroje mimo území Fakulty informatiky. Pokud má atribut *isFormNeeded* hodnotu `false`, pak za zdroje vydávané

tímto výdejářem nenese uživatel hmotnou zodpovědnost (jedná se především o zdroje, které při použití neopouští území Fakulty informatiky).

Jediným atributem výdejářů, který není uveden v entitě **issuemaster**, ale vyplývá ze vztahu s entitou **openinghours**, je atribut *openingHoursId*. V tomto atributu je uložen seznam (`List`) otevíracích hodin výdejáře. Otevírací hodiny výdejáře udávají, kdy si může uživatel u daného výdejáře vyzvednout či vrátit rezervované zdroje. Systém nedovolí nastavit počátek nebo konec rezervace tak, aby byl mimo otevírací hodiny výdejáře požadovaných zdrojů.

Třída `OpeningHours`. Objekty této třídy mají v systému význam otevíracích hodin výdejářů. Každý objekt otevíracích hodin je přiřazen jednomu výdejáři, jehož *id* je uvedeno v atributu *issuemasterid*. Ostatní atributy otevíracích hodin jsou *openingHoursFrom* udávající hodinu začátku a *openingHoursTo* udávající hodinu konce otevíracích hodin. Posledním atributem těchto objektů je *openingHoursType*, jenž říká, na jaký den v týdnu se dané otevírací hodiny vztahují. Pokud má atribut *openingHoursType* hodnotu mezi 1-7, pak toto číslo udává den v týdnu, na který se tyto otevírací hodiny vztahují (pondělí-pátek). Hodnota 8 znamená, že tyto otevírací hodiny jsou platné každý všední den.

Třída `Source`. Objekty této třídy reprezentují v systému data entity **source** a představují všechny zdroje v systému. Zdroje vyjadřují vybavení laboratře LEMMA, které je určeno k podpoře výuky a uživatelé systému si ho tudíž mohou rezervovat a následně vypůjčit pro svoji tvorbu.

Atributy zdrojů odpovídají atributům entity **source**. Jedná se o atributy *name*, udávající název zdroje, který musí být v systému jedinečný, aby byly zdroje rozlišitelné i pro uživatele systému bez znalosti jejich identifikátorů *id*. Dále atribut *cost*, který informuje o ceně zdroje v korunách českých, následuje atribut *inventaryNumber* s inventárním číslem zdroje a atribut *manufacturingNumber* zaznamenávající výrobní číslo zdroje. Atribut *url* uchovává URL[3] stránky s popisem a případně i manuálem k použití daného zdroje. Následuje *isAvailable*, což je atribut typu `boolean` a hodnota `false` tohoto atributu značí, že daný zdroj se momentálně nedá rezervovat (například z důvodu opravy) a systém tedy ani uživatelům nenabídne možnost si tento zdroj rezervovat. Naopak hodnota `true` tohoto atributu znamená, že zdroj je dostupný a uživatelé si ho mohou zarezervovat a vypůjčit.

Následují dva atributy objektů typu `Source`, které nejsou uloženy v entitě **source**, ale vyplývají ze vztahů této entity s jinými entitami. Atribut *sourceTypeId* koresponduje s relací s entitou **sourcetype**. V tomto atributu je uložen druh zdroje (typu `SourceType`), do kterého byl daný zdroj zařazen. A atri-

3. `http://en.wikipedia.org/wiki/Uniform_Resource_Locator`

but *issueMasterId* zase reflektuje relaci s entitou **issuemaster**, proto je typu `IssueMaster` a obsahuje pro každý zdroj jednoho výdejáře, který se stará o vydávání tohoto zdroje.

Třída `SourceType`. Objekty této třídy vystupují v systému jako **druhy zdrojů** a odpovídají datům uloženým v entitě **sourcetype**. Druhy zdrojů sdružují zdroje v systému do větších logických celků. Zdroje z jednoho druhu zdrojů by měly mít nějakou společnou charakteristiku, například podobný účel využití, stejnou cenovou kategorii, či shodný předmět, ve kterém je uživatelé budou potřebovat. Druhy zdrojů mají jediný atribut, a to *name* nesoucí jejich název. Kromě toho, že druhy zdrojů slučují zdroje do větších logických celků, mají ještě význam při uplatňování **kategorií oprávnění** na zdroje. Každý zdroj totiž musí být zařazen právě do jednoho druhu zdroje a druhům zdrojů jsou poté přidělovány kategorie oprávnění.

Třída `PermissionCategory`. Objekty třídy `PermissionCategory` mají v systému význam **kategorií oprávnění**. Kategorie oprávnění mají atribut *name*, který nese jméno kategorie oprávnění. Dále mají objekty tohoto typu atribut *sourceTypes*, ve kterém je uložen seznam (`List`) druhů zdrojů, které byly zařazeny do dané kategorie oprávnění.

Kategorie oprávnění dávají administrátorům systému možnost přidělovat a odebírat uživatelům systému práva na rezervování jednotlivých zdrojů. Konkrétně, každý uživatel systému si může rezervovat pouze ty zdroje, které patří do druhu zdrojů, jenž je zařazen do stejné kategorie oprávnění jako uživatel žádající o rezervaci těchto zdrojů. Systém řeší toto omezení tak, že uživateli jsou při výběru zdrojů pro rezervaci zobrazeny pouze zdroje, které je oprávněn si rezervovat.

Třída `Mailer`. Objekt této třídy je vytvořen pouze pokud systém potřebuje zajistit odeslání informativního e-mailu. Objekty tohoto typu nemají žádný atribut, pouze metodu *sendMail()*. Tato metoda zajišťuje odesílání e-mailů a jako své parametry vyžaduje čtyři řetězce (`String`), které mají následující význam: **emailFrom** udává adresu odesílatele e-mailu; **emailTo** nese adresu příjemce; **subj** značí řetězec, jenž bude použit jako subjekt této zprávy; **co** obsahuje text odesílaného e-mailu. Metoda připojí k textu zprávy informaci, že e-mail byl automaticky vygenerován rezervačním systémem LEMMA a e-mail následně odešle na SMTP server, který získá z atributu **mailSmtpHost** třídy `Constant` (popsáno v sekci 3.1.1 na straně 11).

Třída `WildDate`. Objekty této třídy značí v systému časový okamžik. Slouží k zaznamenání začátku a konce rezervací a žádostí o rezervace. Tato třída je

potomkem třídy `Date` a přetěžuje dvě její metody. Metodu *equals(Object o)* přetěžuje z důvodu snížení její přesnosti z milisekund na hodiny a metodu *toString()* upravuje pro lepší čitelnost zobrazovaných časů v systému a taktéž upravuje její přesnost pouze na hodiny.

Třída `Reservation`. Objekty této třídy reprezentují v systému veškeré rezervace, což je v podstatě časově omezený vztah mezi osobami a zdroji. Rezervace jsou v databázi ukládány do entity **reservation** a jejich objekty mají atribut *personId*, který je typu `Person` a nese informaci o tom, která osoba vytvořila tuto rezervaci. Dále pak atribut *sourceId* typu `Source`, jenž udává zdroj, který byl v této rezervaci obsažen. Dalšími atributy rezervací jsou *fromId* a *toId*, oba typu `WildDate`, které určují časové okamžiky začátku a konce rezervace. Posledním atributem je *isFormGenerated*, což je hodnota typu `boolean` udávající, zda byl pro tuto rezervaci vygenerován výpůjční formulář, či nikoliv.

3.2 Prezentační vrstva

Jak bylo řečeno na začátku této kapitoly na straně 7, prezentační vrstva zajišťuje převod dat jádra systému do podoby vhodné k prezentaci uživatelům. Uživatelé přistupují do systému pomocí webových prohlížečů, proto je potřeba data jádra systému transformovat do formátu čitelného pro tyto prohlížeče. Dále je potřeba zajistit, že kdykoliv nastane změna stavu systému (změní se data jádra systému, typicky nastane po akci provedené některým z uživatelů systému), bude tato změna dynamicky reflektována v datech prezentovaných uživatelům.

O tento dynamický převod dat jádra systému do formátu přijatelného webovými prohlížeči se v systému stará technologie JavaServer Pages[4] (dále jen JSP). Tato technologie je v systému použita ke generování výstupních HTML stránek a zaznamenávání uživateli provedených akcí. K procesu vytvoření těchto stránek získávají JSP potřebná data z jádra systému použitím **metod pracujících s databází** (popsány v sekci 3.1.2 na straně 11). JSP dají získaným datům výslednou podobu, přidají elementy pro interakci uživatele se systémem a vygenerují HTML stránku, která je následně zobrazena uživateli. Uživatel tedy vždy pracuje s aktuálními informacemi.

Jelikož tato dynamická forma prezentace dat systému úzce souvisí s prováděnými akcemi uživatelů, vysvětlíme si zde, jak jsou systémem tyto akce zpracovávané. Následuje tedy přehled průběhu zpracování uživatelova požadavku:

4. JavaServer Pages `http://java.sun.com/products/jsp/`

- uživatel provede nějakou akci v systému použitím interaktivního elementu (typicky stisk tlačítka);
- JSP zaznamená tuto akci a předá ji příslušnému servletu spolu s parametry akce (obvykle popis stisknutého tlačítka a případně další parametry);
- servlet zpracovávající akci zavolá dle typu a parametrů akce příslušné metody jádra systému. V důsledku provedení metod jádra se může změnit stav systému. Po dokončení akce servlet volá příslušné JSP;
- JSP zobrazí uživateli výslednou stránku, která reflektuje případné změny stavu jádra systému po provedení akce;
- systém čeká na další uživatelovu interakci, po níž začne cyklus znovu.

Veškeré uživatelské akce jsou v systému vyjádřeny vznikem objektu typu `HttpServletRequest`. Tyto objekty jsou posílány z JSP metodou *post* servletům. Servlety zpracují tento požadavek, což znamená, že dle parametrů objektu `HttpServletRequest` zjistí, o jaký typ požadavku se jedná, a podle toho zavolají metody jádra systému. Nakonec servlety nastaví pomocí metody *getRequestDispatcher()* objektu `HttpServletRequest` která JSP stránka se po jejich skončení zavolá. Každý servlet v systému typicky zpracovává požadavky z jedné JSP stránky.

Hlavní akce, které mohou uživatelé systému provádět, spolu s JSP stránkami, na kterých se tyto akce dají provést, a servlety, jenž zpracovávají požadavky způsobené těmito akcemi uživatelů, budou uvedeny v sekci **Užití systému** 4.2 na straně 27.

S prezentační vrstvou také souvisí uživatelské prostředí, jež je uživatelům systému poskytováno. Uživatelské prostředí původního systému bylo řešeno čistě s ohledem na jeho funkcionalitu, tudíž se v něm neobjevovaly žádné stylizující prvky a bylo použito minimum kaskádových stylů. Po přihlášení byli uživatelé přesměrováni na hlavní stránku, na které mohli v přehledném grafu sledovat rezervovanost zdrojů a podávat vlastní žádosti o rezervace zdrojů. Z hlavní stránky vedly odkazy na stránky všech ostatních částí systému. Na stránkách těchto částí již byl pouze odkaz, který vedl zpět na hlavní stránku, jež tedy sloužila jako rozcestník v systému.

Interaktivita systému byla vyřešena za použití některých postupů z návrhových vzorů pro uživatelská prostředí, což se v systému projevilo tak, že uživatel nemusí při použití systému zadávat žádné údaje pomocí klávesnice (kromě přihlášení a úvodní registrace) [4]. Tím se uživateli zjednodušší práce a také se zamezí zadávání nesmyslných vstupů. Veškeré údaje zadávané uživatelem jsou tedy řešeny pomocí výběru z předvyplněných možností (selectbox), nebo

označením vhodného políčka (checkbox). Po vybrání nebo označení možností uživatel potvrdí zadané údaje stiskem tlačítka (submit), čímž zahájí provedení akce v systému.

Dále se návrhový vzor uplatnil při řešení situace, kdy si uživatel rezervuje více zdrojů najednou. Uživatel není nucen opakovat stejnou operaci pro rezervování více zdrojů, ale má možnost vybrat více zdrojů najednou (multiselekce) a odeslat žádost o jejich rezervaci stisknutím pouze jednoho tlačítka, i když v systému jsou rezervace poté ukládány jako vztah právě jednoho zdroje s právě jedním uživatelem.

Kapitola 4

Logický model stávajícího systému

Rezervační systém slouží laboratoři LEMMA k evidování a dohlížení na využívanost jejího vybavení. **Uživateli** rezervačního systému jsou tedy především studenti předmětů, k jejichž absolvování je potřeba využívat vybavení laboratoře. **Administrátory** systému jsou pak pracovníci či vedoucí této laboratoře. Technické vybavení, kterým laboratoř disponuje a jež je poskytováno studentům k využití při studiu, je v systému vedeno jako **zdroje**.

Zdroje laboratoře jsou velmi rozmanité, proto byly vytvořeny v systému **druhy zdrojů**. Zdroje se řadí do těchto druhů podle svého účelu, způsobu použití či ceny. Hlavním důvodem rozdělování zdrojů na druhy je ale následné přiřazení **kategorie oprávnění** těmto druhům. Jelikož každý student potřebuje pro svoji práci jen část zdrojů laboratoře, jsou také všem uživatelům přidělovány kategorie oprávnění. Tímto se studentům povolí využívat jen ty zdroje laboratoře, na které dostali od vedení laboratoře oprávnění. Kategorie oprávnění byly též vytvořeny z důvodu vysoké ceny některých zdrojů. U těchto zdrojů je tedy nutné omezit možnost jejich využití jen na oprávněné osoby.

Studentů je o hodně více než dostupných zdrojů, a proto vznikají často konflikty, kdy si chce více studentů vypůjčit jeden zdroj ve stejném časovém rozmezí. Proto byly v systému vytvořeny **rezervace** zdrojů, které se snaží tyto problémy řešit. Studenti mají možnost prohlížet si rezervace ostatních studentů a naplánovat si svoje rezervace s dostatečným předstihem. Případné konflikty s rezervacemi jiných studentů mohou řešit kontaktováním a domluvou s těmito studenty, nebo nahlášením administrátorům, kteří mohou tyto spory rozhodnout.

Studenti si nemohou zdroje z laboratoře samovolně odnášet, ale o vypůjčování zdrojů a jejich přebírání zpět se starají k tomu odpovědné osoby. Tyto osoby jsou v systému nazývány **výdejáři**. Výdejáři jsou obvykle zaměstnanci fakulty a tudíž mají v systému uvedené místnosti, kde si můžou studenti zdroje vyzvednou. Výdejáři nemohou být studentům k dispozici pořád, proto mají v systému zavedeny **otevírací hodiny**. Tyto hodiny je možné si u výdejářů rezervované zdroje vyzvednout nebo vrátit. Systém automaticky kontroluje, aby rezervace začínaly i končily v těchto hodinách.

Některé zdroje se ovšem studentům nepůjčují do vlastních rukou, ale pouze prezenčně v budově fakulty. Jde o zdroje, které se nedají lehce přemísťovat, například střižna. U těchto zdrojů nevystupují jako výdejáři reálné osoby, ale pouze smyšlení zástupci, což může být třeba vrátný místnosti, ve které se tento zdroj nachází. Otevírací hodiny těchto zástupných výdejářů potom udávají časy, kdy je daný zdroj přístupný. Studenti si u takovýchto zdrojů rezervují čas, který mohou využít prací se zdrojem.

Jelikož některé zdroje zapůjčované studentům jsou velmi hodnotné, je potřeba zaručit, že na dobu jejich vypůjčení student přebírá za tyto zdroje hmotnou zodpovědnost a v případě jejich poškození či ztráty je povinen nahradit fakultě vzniklou ztrátu. Toto zaručuje v systému **výpůjční formulář**, který si student může nechat automaticky vygenerovat pro všechny jím rezervované zdroje. Následně formulář podepíše a při přebírání zdrojů předá výdejáři. U zdrojů, které se neodnáší z fakulty, to není potřeba, jelikož tyto zdroje jsou pojištěny fakultou.

4.1 Role v systému

Přístup do systému je možný ve čtyřech různých rolích. Každá z těchto rolí má jinak koncipované uživatelské prostředí, jiné možnosti přístupu do různých částí systému a také jiné možnosti užití systému. Jedná se o následující role s obecnou charakteristikou jejich možných činností v systému.

4.1.1 Host

Hostem v systému je každá osoba, která není zaregistrovaná, nebo čeká na potvrzení registrace administrátory. Host může podat žádost o uživatelský účet v systému. Žádost podá vyplněním a odesláním přihlašovacího formuláře, který je následně doručen administrátorům. Administrátoři poté potvrdí nebo zamítnou žádost hosta o uživatelský účet s ohledem na údaje vyplněné v přihlašovacím formuláři. Pokud je žádost hosta o uživatelský účet v systému potvrzena, je hostovi přidělena role uživatel.

4.1.2 Uživatel s omezením

Uživatelem s omezením je osoba, která je již v systému zaregistrovaná, ale byl jí omezen přístup do systému. Uživatelem s omezením se stane uživatel systému, kterému je tato role přidělena administrátory. Jedním z důvodů přidělení této role uživateli může být ukončení akademického ročníku. Pokud administrátoři systému ukončí ročník, je každému uživateli omezen přístup do systému.

Dalším důvodem pro přidělení této role může být nevhodné chování uživatele v systému. Pokud administrátoři shledají chování uživatele systému za nevhodné, mohou přidělením této role uživateli omezit přístup do systému. Uživatel s omezením se může přihlásit do systému a podat žádost o odblokování účtu. Ta je poté odeslána administrátorům k potvrzení. V případě potvrzení žádosti je uživateli s omezením přidělena role uživatele systému.

4.1.3 Uživatel

Uživatel systému je základní role osoby v systému, která je zaregistrovaná a systém běžně používá. Uživatel může prohlížet rezervovanost zdrojů pomocí přehledného grafu. Dále může podávat žádosti o rezervaci zdrojů, na které má příslušná oprávnění.

V případě konfliktu uživatelovy žádosti o rezervaci zdrojů s rezervací jiného uživatele jsou uživateli zobrazeny kontaktní údaje uživatele, jenž vlastní danou rezervaci, čímž je umožněno uživateli vyřešit podle vlastního uvážení vzniklý konflikt. Pokud je uživatelova žádost v konfliktu s otevíracími časy výdejáře daného zdroje, pak je uživatelovi nabídnuta možnost nechat si systémem upravit jeho žádost podle těchto otevíracích časů.

Dále má uživatel možnost prohlížet si jím vytvořené rezervace, popřípadě je rušit, nebo si nechat vygenerovat předvyplněný formulář pro rezervované zdroje. Tento formulář je potřebný pro vydání zdrojů uživateli.

4.1.4 Administrátor

Administrátoři spravují a konfigurují systém. Mají možnost upravovat data v databázi systému. Mohou spravovat údaje o osobách, zdrojích, rezervacích a výdejářích v systému. Dále mohou měnit nastavení systému. Administrátorům jsou zasílány žádosti o uživatelský účet nebo odblokování účtu, které mohou potvrdit či zamítnout. Dále mohou přidělovat uživatelům kategorie oprávnění pro rezervaci různých druhů zdrojů. Administrátoři také mohou přidělovat uživatelům role v systému.

4.2 Užití systému

V této sekci bude vysvětleno, která užití systému se vážou k jednotlivým rolím v systému, jaké události vznikají v systému při těchto užití a které servlety tyto události zpracovávají.

4.2.1 Host

- Žádost o účet v systému; požadavek z **accountRequest.jsp** s parametrem *newAccount* zpracovává **AccountRequestServlet.java**.

4.2.2 Uživatel s omezením

- Žádost o odblokování účtu v systému; požadavek z **bannedUser.jsp** s parametrem *upgradeAccount* zpracovává **AccountRequestServlet.java**.

4.2.3 Uživatel

- Rezervace zdrojů; požadavek z **user.jsp** s parametrem *tryToReserve* zpracovává **ReservationRequestServlet.java**.

- Rušení rezervace zdrojů; požadavek z **reservations.jsp** s parametrem *deleteRes* zpracovává **ReservationsServlet.java**.

- Generování výpůjčního formuláře; požadavek z **reservations.jsp** s parametrem *generateForm* zpracovává **ReservationsServlet.java**.

4.2.4 Administrátor

- Potvrzení nebo zamítnutí žádosti o uživatelský účet; požadavek z **newUser.jsp** s parametrem *acceptUser* při potvrzení nebo *rejectUser* pro zamítnutí, zpracovává **NewUserServlet.java**.

- Upravení nebo smazání uživatele; požadavek z **editUser.jsp** s parametrem *updateUser* při upravení uživatele nebo *deleteUser* pro smazání uživatele, zpracovává **EditUserServlet.java**.

- Ukončení ročníku v systému; požadavek z **editUser.jsp** s parametrem *degradeUsers* zpracovává **EditUserServlet.java**.

- Zrušení uživatelské rezervace; požadavek z **editReservation.jsp** s parametrem *deleteReservation* zpracovává **EditReservationServlet.java**.

- Přidání zdroje; požadavek z **newSource.jsp** s parametrem *insertSource* zpracovává **NewSourceServlet.java**.

- Přidání druhu zdrojů; požadavek z **newSource.jsp** s parametrem *insertSourceType* zpracovává **NewSourceServlet.java**.

- Upravení nebo smazání zdroje; požadavek z **editSource.jsp** s parametrem *updateSource* při úpravě zdroje nebo *deleteSource* pro smazání zdroje, zpracovává **EditSourceServlet.java**.
- Upravení nebo smazání druhu zdrojů; požadavek z **editSource.jsp** s parametrem *updateSourceType* při úpravě druhu zdroje nebo *deleteSourceType* pro smazání druhu zdroje, zpracovává **EditSourceServlet.java**.
- Přidání výdejáře; požadavek z **newIssueMaster.jsp** s parametrem *insertIssueMaster* zpracovává **NewIssueMasterServlet.java**.
- Upravení nebo smazání výdejáře; požadavek z **editIssueMaster.jsp** s parametrem *updateIssueMaster* při úpravě výdejáře nebo *deleteIssueMaster* pro smazání výdejáře, zpracovává **EditIssueMasterServlet.java**.
- Přidání kategorie oprávnění; požadavek z **newPC.jsp** s parametrem *insertPC* zpracovává **NewPCServlet.java**.
- Upravení nebo smazání kategorie oprávnění; požadavek z **editPC.jsp** s parametrem *updatePC* při úpravě kategorie oprávnění nebo *deletePC* pro smazání kategorie oprávnění, zpracovává **EditPCServlet.java**.
- Upravení nastavení systému; požadavek z **editSystem.jsp** s parametrem *editSystem* zpracovává **EditSystemServlet.java**.

Kapitola 5

Analýza požadovaných rozšíření

Během používání původního rezervačního systému LEMMA vzniklo na základě námětů a zkušeností uživatelů systému mnoho návrhů na jeho úpravu či rozšíření. Zde si přiblížíme důvody, které vedly ke vzniku nejdůležitějších rozšíření, jež byly následně implementovány.

5.1 Výdejář zdrojů

Výdejáři zdrojů musejí se systémem pracovat téměř každý den, zejména před koncem akademického roku, kdy práce na studentských projektech vrcholí a rezervací rapidně přibývá.

V původním systému neměli výdejáři žádnou vlastní roli, nebo speciální funkce, které by jim usnadnily práci se systémem. Výdejáři byli v systému vedeni jako běžní uživatelé a měli k dispozici pouze uživatelské rozhraní s jeho funkcemi. Uživatelské rozhraní sice umožňuje sledovat všechny rezervace v systému, ale nedovoluje již s nimi nadále pracovat, jako například třídit dle počátku, vypsat ve formě vhodné pro tisk nebo rušit rezervace zdrojů, které si uživatelé nevyzvedli.

Jelikož výdejáři nemají všechny zdroje, které spravují, ve své místnosti, stává se z vydávání těchto zdrojů časově náročný proces. Výdejářům by tedy výrazně usnadnila práci služba v systému, která by jim dovolovala na začátku každého dne zjistit, jaké zdroje budou v daný den vydávat. Výdejář by si poté mohl seznam zdrojů vytisknout a při prvním vydávání zdrojů by si mohl zajistit veškeré zdroje, které bude v daný den vydávat. Dále systém nedovoloval výdejářům vkládat své dovolené, tudíž uživatelské rezervace mohly začínat v době, kdy zdroje nebyly vydávány.

Výše uvedené požadavky vedly k úpravě, která do systému zavádí funkci **Výdejář zdrojů**. Tato funkce není novou rolí v systému, jelikož výdejářem nemusí být vždy reálná osoba, ale třeba jen smyšlený zástupce, o jehož údaje se stará administrátor. Dále to není nová role z důvodu usnadnění přenositelnosti funkce výdejáře mezi různými osobami. Pokud osoba přestane vykonávat

v systému funkci výdejáře zdojů, bude stačit předat tuto funkci jiné osobě v systému a nebude potřeba zavádět do systému nového výdejáře a přiřazovat mu všechny zdroje, které měl na starost bývalý výdejář. Implementace této úpravy je popsána v kapitole **Implementace úprav a rozšíření** v sekci 6.1 na straně 35.

5.2 Sledování průběhu rezervací

V původním systému nemohli uživatelé zjistit, zda si uživatel opravdu vyzvedl rezervovaný zdroj. Tudíž nastávaly situace, kdy byl zdroj ve skutečnosti volný k zapůjčení, ale v systému byl vedený jako zarezervovaný a nikdo jiný si ho tedy nemohl zarezervovat a vypůjčit. Nebo vznikaly situace, kdy student včas nevrátil vypůjčený zdroj. Pokud si tento zdroj chtěl zarezervovat další student, neměl žádnou možnost zjistit, že zdroj ještě nebyl vrácen.

Pro řešení těchto situací byl v systému vyvinut **stav** rezervací. Podle tohoto stavu mohou uživatelé sledovat průběh cizích rezervací a případně podle něj přizpůsobovat vlastní rezervace. Stavy rezervací jsou v uživatelském prostředí rozlišeny různými barvami v grafu rezervovanosti zdrojů. Pokud nebyl rezervovaný zdroj opravdu vypůjčen, uživatelé toto v systému vidí a mohou si zdroj rezervovat, čímž zruší původní nevyzvednutou rezervaci. Dále systém rozlišuje rezervace, u kterých ještě nebyl zdroj vrácen, takže ostatní uživatelé vidí, že si tento zdroj nemohou vypůjčit.

Ke sledování průběhu rezervací je ovšem nutná aktivní účast výdejářů zdrojů. Ti musí do systému zaznamenat každé vydání nebo přijetí zdrojů, což se jim systém snaží usnadnit tím, že je po přihlášení přesměruje na stránku, kde jsou zobrazeny rezervace čekající na potvrzení a výdejáři mohou stiskem jednoho tlačítka potvrdit vydání či vrácení zdrojů. Uživatelé jsou za nevyzvednuté nebo nevrácené zdroje penalizováni. Důsledky penalizace jsou vysvětleny v sekci 5.5 na následující straně. Implementaci sledování průběhu rezervací naleznete v kapitole **Implementace úprav a rozšíření**, přesněji její sekci 6.2 na straně 36.

5.3 Kontrola rezervací

V důsledku zavedení průběhu rezervací do systému je potřeba kontrolovat, jestli v systému nezůstávají nevyzvednuté nebo nevrácené rezervace. Pokud systém zjistí, že u rezervace nebylo potvrzeno vydání zdrojů více jak hodinu po jejím začátku, pak upozorní e-mailem uživatele, aby si zdroje vyzvedl. Systém také změní stav rezervace tak, že ji mohou ostatní uživatelé systému zrušit svými rezervacemi. V případě nevrácených zdrojů systém taktéž upozorní

uživatele e-mailem a změní stav rezervace, aby ostatní uživatelé viděli, že zdroj ještě nebyl vrácen. Implementace kontroly rezervací je vysvětlena v kapitole **Implementace úprav a rozšíření** v sekci 6.3 na straně 37.

5.4 Kolekce zdrojů

Některé zdroje laboratoře se dají ke svému účelu využít pouze spolu s jiným zdrojem, například pro světlo je potřeba stojan. Popřípadě může být tato vazba způsobena bezpečnostním opatřením, například vhodný obal pro přenos kamery.

Nutnost řadit zdroje do větších nedělitelných celků byla příčinou vzniku **Kolekcí zdrojů**. Kolekce zdrojů mohou vytvářet a upravovat administrátoři systému. Zdroje z jedné kolekce je možné zarezervovat pouze všechny najednou, nebo žádný z nich. Taktéž není možné zrušit rezervaci části zdrojů z kolekce, ale pouze rezervaci celé kolekce. Implementace kolekcí je vysvětlena v kapitole **Implementace úprav a rozšíření** v sekci 6.4 na straně 37.

5.5 Nadměrné rezervace

Dalším požadavkem na rozšíření systému byla možnost kontroly neúměrného využívání zdrojů laboratoře. Někteří studenti si totiž rezervovali zdroje na neúnosně dlouhou dobu, nebo příliš mnoho zdrojů najednou, čímž zamezovali ostatním studentům v kvalitní práci. Proto byly v systému vytvořeny **nadměrné rezervace**.

K určení, zda je rezervace nadměrná, slouží tři kritéria, která mohou nastavovat administrátoři systému. Těmito kritérii jsou počet rezervovaných zdrojů, délka rezervace ve dnech a celková cena rezervovaných zdrojů vynásobená počtem hodin trvání rezervace (v systému nazýváno korunohodiny a vyjádřeno atributem osob *crownhours*). Pokud rezervace zdrojů uživatele přesáhne alespoň jedno z kritérií, je tato rezervace označena za nadměrnou a musí ji schválit administrátor. Než administrátor rozhodne o nadměrné rezervaci, je uživateli odebrána možnost nechat si vygenerovat výpůjční formulář, bez kterého si nemůže zdroje vypůjčit.

Uživatelům se za každou jejich rezervaci zvýší hodnota atributu *crownhours* (dále jen korunohodiny). Korunohodiny tedy udávají využívanost systému uživateli. Při nevyzvednutí nebo pozdním vrácení zdrojů se uživatelům přičte trojnásobek korunohodin. Administrátorům při rozhodování o nadměrných rezervací pomáhá percentil, který udává využívanost systému uživatelem, jenž vlastní nadměrnou rezervaci, v porovnání s ostatními uživateli. Imple-

mentaci tohoto rozšíření naleznete v kapitole **Implementace úprav a rozšíření** v sekci 6.5 na straně 38.

5.6 Historie rezervací

Laboratoř potřebuje uchovávat záznamy o provedených výpůjčkách pro pozdější potřeby, případně jako podklady při vykazování činnosti laboratoře. Proto bylo v systému zrušeno mazání starých rezervací a všechny rezervace zůstávají v systému v rámci historie rezervací. Prohlížet historii rezervací mohou administrátoři a výdejáři. Historii je možné prohledávat buď podle rezervací jednotlivých uživatelů, nebo podle rezervací zdrojů. Implementace je vysvětlena v kapitole **Implementace úprav a rozšíření** v sekci 6.6 na straně 39.

Kapitola 6

Implementace úprav a rozšíření

6.1 Výdejář zdrojů

Do entity **issuemaster** byl přidán atribut *personid* typu `int`. Atribut má z objektového hlediska význam osoby, které byla přiřazena funkce výdejáře. Třídě `Person`, jejíž instance jsou osoby v systému, přibyla metoda *isIssueMaster()*, která zjišťuje, zda osoba vykonává v systému funkci výdejáře zdrojů.

Dále byla v databázi vytvořena nová entita **vacationdates**, která uchovává informace o dovolených výdejářů. Objektům třídy **IssueMaster** tedy přibyl atribut *vacationDatesId* typu `List`, ve kterém jsou uloženy dovolené výdejáře. V důsledku vzniku dovolených výdejářů musí být v žádosti o rezervaci kontrolováno, zda požadovaná rezervace nezačíná nebo nekončí v průběhu dovolené výdejáře. Třídě `ReservationRequest` a jejím instancím tedy přibyla metoda *collideWithIssueMasterVacationDates(IssueMaster iMaster)*, která zjistí, zda žádost nezačíná, nebo nekončí v průběhu dovolené výdejáře ze svého parametru.

Ve fasádě `IssueMasterFacade`, která slučuje metody pracující s entitou **issuemaster** v databázi, přibyly následující metody:

- *allIssueMastersPersons()* vrací množinu (`Set`) obsahující **id** všech osob v systému, které mají funkci výdejáře;
- *findByPersonId(int id)* vyhledá a vrátí výdejáře, který je přiřazen osobě s **id** shodným s parametrem této metody;
- *insertVacationDatesToIssueMaster(IssueMaster issueMaster, VacationDates vd)* uloží dovolenou z parametru **vd** do entity **vacationdates** a jako jejího vlastníka nastaví výdejáře z parametru **issueMaster**, tohoto výdejáře poté metoda vrací;
- *deleteVacationDatesFromIssueMaster(IssueMaster issueMaster, VacationDates vd)* smaže dovolenou **vd** výdejáře **issueMaster** a následně vrátí upraveného výdejáře;
- *dropObsolateVacations()* smaže všechny dovolené v systému, které již proběhly;

- *allVacations()* vytvoří a vrátí seznam (`List`) všech dovolených, které jsou vedené v systému;
- *setUpVacationDates(IssueMaster issueMaster)* vyhledá dovolené patřící výdejáři z parametru metody a vloží mu tyto dovolené ve formě seznamu (`List`) do atributu vacationDatesId, tato metoda snižuje čas potřebný k vytvoření výdejáře, jelikož konstruktor výdejáře nenastavuje atribut vacationDatesId, tento atribut je nastaven, až když je opravdu potřeba pracovat s dovolenou výdejáře.

V systému bylo též změněno přihlašování, které teď kontroluje, zda nemá přihlašovaná osoba funkci výdejáře. Výdejáři jsou po přihlášení do systému přesměrováni do výdejářského prostředí, kde mají následující možnosti užití systému:

- potvrzení vydání nebo přijetí zdrojů;
- rušení rezervací;
- vypsání rezervací začínajících dnes nebo zítra ve formě vhodné pro tisk;
- prohlížení historie rezervací;
- vypsání všech zdrojů v systému, vhodné pro inventuru;
- editování vlastních údajů i se zadáváním dovolených.

6.2 Sledování průběhu rezervací

Pro možnost sledování průběhu rezervace v systému byl do entity **reservation** přidán atribut *state* typu `int`. Tento atribut vyjadřuje u rezervací v systému (objekty typu `Reservation`) stav v jakém se dané rezervace nachází. Stavy rezervací jsou rozlišeny v uživatelském prostředí v grafu rezervací a mohou být následovné:

- **nadměrná rezervace** (*state* roven 0) popsáno v sekci 6.5 na straně 38;
- **rezervováno** (*state* roven 1) zdroje byly rezervovány a rezervace čeká na začátek;
- **nevyzvednuto** (*state* roven 2) rezervace již začala, ale výdejář nepotvrdil vyzvednutí zdrojů, více v sekci 6.3 na následující straně;
- **vypůjčeno** (*state* roven 3) rezervace probíhá a zdroje byly řádně vypůjčeny;

- **nevráceno** (*state* roven 4) rezervace skončila, ale nebylo potvrzeno vrácení zdrojů, více v sekci 6.3;
- **ukončeno** (*state* roven 5) rezervace zdárně proběhla a je v systému vedena již jen v historii rezervací.

6.3 Kontrola rezervací

Systém každou hodinu zkontroluje, zda se v něm nenachází rezervace, které mají atribut *state* roven 1, ale nebylo u nich potvrzeno vydání zdrojů déle jak hodinu po jejich začátku. Takovým rezervacím změní systém stav na **nevyzvednuto** (*state* roven 2) a uživatele těchto rezervací upozorní e-mailem, aby si zdroje vyzvedli. Za každou hodinu, po kterou si uživatelé zdroje nevyzvedli, se těmto uživatelům zvýší atribut *crownHours* o trojnásobek ceny zdroje.

Dále systém kontroluje rezervace ve stavu **vypůjčeno**. Pokud zjistí, že rezervace je v tomto stavu déle jak hodinu po skončení, změní její stav na **nevráceno** a upozorní uživatele. Uživatelům se taktéž zvyšuje atribut *crownHours* o trojnásobek ceny zdroje za každou hodinu, po kterou zdroje nevrátí.

Kontrolu rezervací zajišťuje třída `CheckReservationsState` a její metoda *checkStates()* spolu s daemonem *cron*. Cron spustí každou hodinu metodu *checkStates()* svým přístupem na checkReservations.jsp. Do fasády `ReservationFacade` byly přidány následující metody:

- *findReservationsForCheck()* vrátí seznam (`List`) rezervací, které měly začít nebo skončit před více než hodinou, ale nebylo u nich potvrzeno vydání či vrácení zdrojů;
- *findReservationsForConfirm()* vrátí seznam (`List`) rezervací, které mají v následujících 12 hodinách začínat nebo končit, spolu se staršími rezervacemi, u nichž nebylo potvrzeno vydání či vrácení zdrojů;
- *updateState(Reservation reservation)* zapíše do databáze změnu hodnoty atributu **state** rezervace, která je parametrem této metody;
- *updateCrownHours(Person person)* zaznamená do databáze postih uživatele za nevyzvednutí či nevrácení zdrojů změnou hodnoty atributu **crownhours** u osoby z parametru této metody.

6.4 Kolekce zdrojů

Kolekce zdojů byly do systému implementovány přidáním atributu *collection* entitě **source**. Atribut je typu `int` a zdroje se shodnou hodnotou atributu patří

do jedné kolekce. Hodnota 0 znamená, že zdroj nepatří do žádné kolekce. Administrátorům byly přidány možnosti pro vytváření, upravování a rušení kolekcí. Uživatelům je při označování zdrojů pro rezervaci umožněno označení pouze celé kolekce najednou. Do fasády `SourceFacade` přibyly následující metody pro práci s daty entity **source**:

- *getAvailableSourcesForCollection()* vrátí seznam (`List`) zdrojů, které je možné zařadit do kolekce zdrojů, což jsou takové zdroje, které nejsou v žádné jiné kolekci a jsou dostupné (jejich atribut **isavailable** je roven `true`);
- *getMaxNumberOfCollection ()* vrátí nejvyšší nalezenou hodnotu (typu `int`) atributu **collection** v databázi, která taktéž udává aktuální počet kolekcí v systému;
- *newCollection(String sources)* vytvoří novou kolekci zdrojů a to tak, že nastaví atribut **collection** zdrojům, jejichž **id** je obsaženo v řětezci předávaném jako parametr metody, na hodnotu o jedna vyšší než je aktuální počet kolekcí v systému;
- *deleteCollection(int idCollection)* smaže kolekci zdrojů, což znamená, že změní hodnotu atributu **collection** na 0 u zdrojům s hodnotou tohoto atributu rovnou parametru metody;
- *addToCollection(int collection, int source)* přidá zdroj s **id** rovným parametru **source** do kolekce zdrojů s atributem **collection** nastaveným na hodnotu parametru metody **collection**;
- *getCollection(int id)* vrátí seznam (`List`) zdrojů, které mají hodnotu atributu **collection** rovnou hodnotě parametru metody;
- *getCollectionCount(int id)* spočítá a vrátí počet zdrojů v koleci;
- *removeSourcesFromCollection(String sources)* odstraní z kolekce zdroje, jejichž **id** je obsaženo v řetězci parametru metody.

6.5 Nadměrné rezervace

Pro potřeby ohlídání vzniku nadměrných rezervací byly v systému vytvořeny tři hodnoty, které udávají limit pro normální rezervace. Tato kritéria jsou uložena v entitě **reservationsystem** jako atributy: *maxSourcesInReservation*, *maxDaysOfReservation*, *maxCrownHours*. Při zpracování žádosti o rezervaci

je zkontrolováno, zda rezervace splňuje kritéria, pokud ne, je označena za nadměrnou (její atribut *state* je nastaven na 0).

O vzniku nadměrných rezervací je informován administrátor pověřený k rozhodování o nadměrných rezervací (e-mail administrátora je uložen v entitě **reservationsystem** jako atribut *mailOnBigReservationTo*). Administrátorům byly přidány užití systému, kde mohou potvrzovat (stav rezervace se změní na 1) nebo zamítat (zruší rezervaci) nadměrné rezervace. Do fasády `ReservationFacade` přibyla metoda pro práci s nadměrnými rezervacemi:

- *findBigReservations()* najde a vrátí seznam (`List`) příliš velkých rezervací (rezervace s hodnotou atributu **state** rovnou 0);

Administrátorům pomáhá při rozhodování o nadměrné rezervaci percentil, který udává využívanost systému daným uživatelem. Pro výpočet percentilu byly přidány následující metody do `PersonFacade`:

- *personsCount()* vrací počet osob vedených v databázi, návratová hodnota je typu `int`;
- *personsUnderCrownHoursCount(long crown)* vrací počet (`int`) osob, jejichž atribut **crownhours** je ostře menší než hodnota parametru metody.

6.6 Historie rezervací

Historie rezervací byla v systému implementována tak, že se již nikde v systému nepoužívá metoda *dropObsolateReservations()*, která mazala ukončené rezervace. Všechny rezervace tedy zůstávají v databázi systému a výdejářům s administrátory přibyly možnosti pro prohlížení historie rezervací. Historii lze prohlížet buď podle osob, které rezervace vytvořily, nebo dle zdrojů, které byly rezervovány. K prohlížení historie rezervací byly přidány tyto metody do `ReservationFacade`:

- *findBySource(Source sourceId)* vrátí seznam (`List`) rezervací, jejichž atribut **sourceid** obsahuje **id** zdroje z parametru této metody;
- *findByPerson(Person personId)* vrátí seznam (`List`) rezervací, jejichž atribut **personid** obsahuje **id** osoby z parametru této metody;

6.7 Úprava uživatelského prostředí

Do systému bylo implementováno používání kaskádových stylů, které jsou využity též na stránkách laboratoře LEMMA[1]. Styly kromě změny vzhledu

1. `http://www.fi.muni.cz/lemma/`

uživatelského prostředí systému umožňují také lepší navigaci v systému. Uživatelé mají teď možnost procházet systém pomocí přehledného menu, které je na všech stránkách umístěno v levé části obrazovky.

6.8 Odhlášení neaktivních uživatelů

V původním systému vznikaly situace, kdy uživatel po delší době (30 minut a více) nečinnosti provedl nějakou akci, ale systém ji nedokázal zpracovat, protože server již zneplatnil uživatelovu `Session`[2]. Uživateli byla poté vypsána chybová hláška. Implementováním kontroly, zda je uživatelova `Session` stále aktivní při provedení jakékoliv akce, se tyto problémy vyřešily. V případě zjištění, že uživatelova `Session` je neplatná, přesměruje systém uživatele na úvodní stránku a zobrazí vhodnou hlášku.

6.9 Nasazení systému

Systém pracuje s databází **pgdb** na serveru **db.fi.muni.cz**. Původní verze systému využívala schéma databáze, ke kterému se dalo přistupovat pouze pod účtem jednoho z tvůrců původního systému. Požádal jsem tedy správce databáze o vytvoření nového schématu a účtu na databázovém serveru. Bylo mi vyhověno a nová verze rezervačního systému již pracuje se schématem **lemma**, ke kterému se dá přistupovat pod účtem **lemma**. Účet je plně ve správě laboratoře, není závislý na konkrétním studentovi fakulty a nebude tudíž zrušen po ukončení studia žádného studenta.

Samotné nasazení systému na stroj **kore.fi.muni.cz** se stalo použitím již zmiňovaného nástroje **Tomcat Web Application Manager** triviálním. Po spuštění nové verze se ovšem v systému vyskytl problém a to nesprávné generování českých znaků ve výpůjčním formuláři, přestože část systému, která se stará o generování formuláře, nebyla nijak upravována. Problém způsobovaly knihovny, které zajišťují generování formuláře. Na server byla kromě správné knihovny nahrána i knihovna, která byla nastavena pro generování formuláře na vývojovém počítači. Aplikace se zřejmě z důvodu abecedního pořadí knihoven, ve kterém nesprávná knihovna předcházela správnou, snažila načítat fonty českých znaků z umístění, které na serveru neexistovalo. Problém byl vyřešen odebráním nevhodně nastavené knihovny z aplikace.

2. `http://tomcat.apache.org/tomcat-5.5-doc/config/context.html`

Kapitola 7

Závěr

Mezi hlavní změny v systému tedy patří:

- výdejář zdrojů;
- sledování průběhu rezervací;
- kontrola rezervací;
- kolekce zdrojů;
- nadměrné rezervace;
- úprava uživatelského prostředí;
- historie rezervací.

Dle mého názoru provedené úpravy a rozšíření splňují požadavky kladené vedením laboratoře. Systém je úspěšně používán v předmětech PV110 a PV113. I přes nová rozšíření zůstává systém přehledným a snadno použitelným nástrojem, který, jak pevně věřím, přispěje ke zkvalitnění studia na fakultě.

Systém je dostatečně obecný, aby se dal použít i na jiných pracovištích s podobnými požadavky. Jako směry budoucího vývoje bych si dokázal představit integrování autorizace uživatelů přes Informační systém Masarykovy univerzity, či zlepšení služeb pro inventarizaci vybavení.

Při práci na systému jsem se důkladněji seznámil s technologií Java EE a základy vývoje informačních systémů. Dané problematice bych se rád věnoval hlouběji při dalším studiu či zaměstnání.

Literatura

[1] Wikipedie: Otevřená encyklopedie. *Model-view-controller*. [online]. c2008 [citováno 28. 04. 2008]. Dostupné z WWW: `http://cs.wikipedia.org/wiki/Model-view-controller`.

[2] Jiří Filipovič. *Systém alokace zdrojů laboratoře*. [bakalářská práce]. Fakulta informatiky Masarykovy univerzity v Brně, 2006. Dostupné v archivu závěrečných prací Informačního systému Masarykovy univerzity z WWW: `https://is.muni.cz/auth/th/72898/fi_b/`.

[3] Zdeněk Nováček. *Reverzní inženýrství v prostředí CASE Studio 2*. [bakalářská práce]. Fakulta informatiky Masarykovy univerzity v Brně, 2006. Dostupné v archivu závěrečných prací Informačního systému Masarykovy univerzity z WWW: `https://is.muni.cz/auth/th/72725/fi_b/`.

[4] Vojtěch Vild. *Dokumentace rezervačního systému laboratoře*. [seminární práce PV168]. Fakulta informatiky Masarykovy univerzity v Brně, 2006. Dostupné na přiloženém CD.

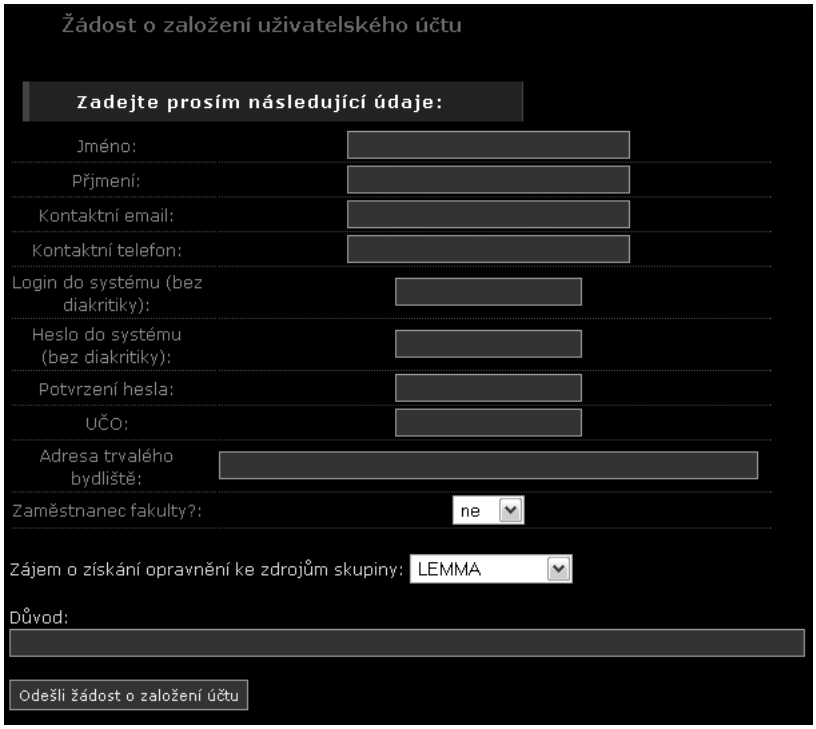
Dodatek A

Rezervační systém LEMMA v kostce

Stručný úvod do základů práce s rezervačním systémem laboratoře LEMMA

A.1 Registrace

Pro získání účtu v systému se musíte nejdříve zaregistrovat. Všechny údaje



Obrázek A.1: Registrační formulář

v registračním formuláři (obrázek A.1) jsou povinné. V části **Zájem o získání oprávnění ke zdrojům skupiny** máte na výběr možnost **LEMMA**, která zahrnuje většinu vybavení laboratoře a je standardním oprávněním uživatelů.

Druhou možností je oprávnění **LEMMAnad100**, které vám umožní rezervovat vybavebí laboratoře, jehož cena přesahuje částku sto tisíc korun. Jedná se především o kamery.

Jako **Důvod** uveďte název projektu, který budete pomocí vybavení laboratoře realizovat. Po odeslání bude registrační formulář doručen administrátorům systému, kteří o vaší žádosti rozhodnou. Jejich rozhodnutí vám bude zasláno na zadaný e-mail.

A.2 Rezervace zdrojů

Vybavení laboratoře, které potřebujete zapůjčit pro svoji práci, si musíte nejdříve v systému zarezervovat. Rezervacemi se předchází konfliktům při žádostech více studentů o zapůjčení stejných zdrojů. Systém neumožní rezervovat zdroje, které jsou již rezervované na požadovanou dobu jiným studentem.

Na obrázku A.2 je znázorněno jak vyplnit žádost o rezervaci zdrojů. Rezervovatelné zdroje jsou zobrazeny v tabulce, která je dále rozdělena pro

Obrázek A.2: Rezervace zdrojů

lepší přehlednost podle druhů zdrojů. Při rezervaci více zdrojů stačí označit všechny požadované zdroje a odeslat pouze jednu žádost o rezervaci.

Pro výpis rezervací ostatních uživatelů použijte tlačítko **Vykresli přehled rezervací**. Následně vám bude zobrazen graf rezervovanosti zdrojů, kde jsou barevně rozlišeny vaše rezervace od rezervací jiných uživatelů. Pro zjištění

vlastníků cizích rezervací použijte znovu výše uvedené tlačítko, které bude mít teď popisek **Vykresli podrobný přehled rezervací**.

A.3 Neúspěšné rezervace

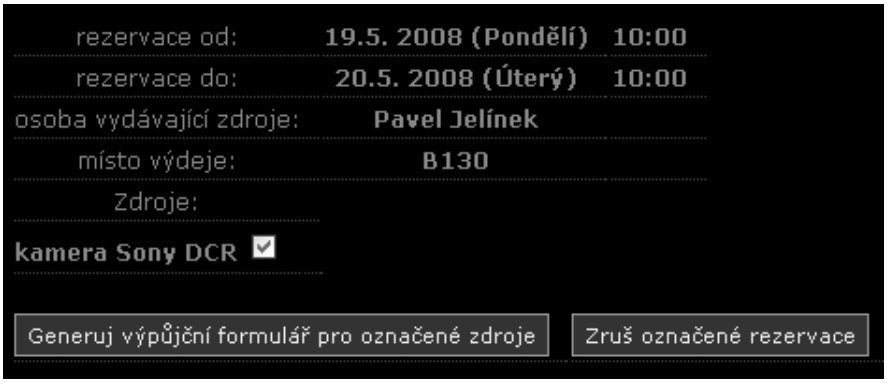
Neúspěch při žádosti o rezervaci zdrojů nastane, pokud je některý z požadovaných zdrojů již rezervován jiným uživatelem. V tom případě vám systém zobrazí rezervaci, se kterou je vaše žádost v konfliktu, a nabídne možnost pokračovat v částečné rezervaci. Částečná rezervace vám zarezervuje pouze volné zdroje. Pro rezervaci všech požadovaných zdrojů se nejdříve podívejte do přehledu rezervací, kde zjistíte dostupnost zdrojů.

Dalším důvodem neúspěchu vaší rezervace může být konflikt s otevíracími hodinami výdejáře zdrojů. **Výdejář zdrojů** je osoba (typicky zaměstnanec fakulty) pověřená k vypůjčování a přebírání zdrojů. Výdejáři mají v systému uvedeny otevírací hodiny, které udávají, kdy si u nich můžete rezervované zdroje vypůjčit nebo vrátit.

Pokud začátek nebo konec vaší rezervace nespadá do otevíracích hodin výdejáře, pak vám systém nabídne možnost automaticky upravit časy rezervace podle otevíracích hodin výdejáře požadovaných zdrojů.

A.4 Úspěšné rezervace

V případě úspěšných rezervací budou tyto zobrazeny v části **Správa rezervací uživatele**. Obrázek A.3 ukazuje příklad úspěšné rezervace. Na řádku **Osoba**



Obrázek A.3: Přehled úspěšných rezervací

vydávající zdroje vidíte jméno výdejáře, od kterého si zdroje z této rezervace

vyzvednete. Místnost, ve které vám výdejář zdroje předá, je zobrazena na řádku **Místo výdeje**.

Před vyzvednutím zdrojů od výdejáře si na této stránce vygenerujte výpůjční formulář. Formulář vytiskněte, podepište a při přebírání zdrojů předejte výdejáři. Podepsáním formuláře stvrzujete převzetí hmotné zodpovědnosti za vypůjčené zdroje, bez podepsaného formuláře vám nebudou zdroje zapůjčeny.

U některých rezervací vám systém nedovolí vytisknout výpůjční formulář. Jedná se o rezervace zdrojů, které při používání neopustí území fakulty, kde jsou pojištěné fakultou. Dále vám systém nedovolí vytisknout formulář u příliš velkých rezervací. Příliš velké rezervace vám musí nejdříve schválit administrátor systému a až poté si můžete formulář vytisknout.

Pokud si rezervované zdroje vyzvednete nebo vrátíte se zpožděním či dokonce vůbec nevypůjčíte, pak budete systémem penalizováni. Administrátoři systému vidí vaše penalizace a mohou na jejich základě rozhodnout o zrušení vašich rezervací, zamítnout příliš velké rezervace nebo dokonce omezit váš přístup do systému.

Dodatek B

Obsah přiloženého CD

Následuje výčet adresářů na přiloženém CD a přiblížení jejich obsahu:

- **conf**: konfigurační a transformační soubory pro generování výpůjčního formuláře a soubor *lemma.properties* s nastavením systému;
- **database**: soubor *lemmaDump.sql* obsahující SQL příkazy pro vytvoření systémové databáze;
- **doc**: dokumentace původního systému;
- **libs**: knihovny potřebné pro běh systému;
- **src**: zdrojové kódy všech tříd a servletů;
- **web**: JSP soubory, kaskádové styly a další soubory potřebné pro generování HTML stránek systému.