

UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

DIPLOMOVÁ PRÁCE

2016

Bc. Richard Matula

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Knihovní informační systém pomocí frameworku Spring

Bc. Richard Matula

Diplomová práce

2016

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Richard Matula**
Osobní číslo: **I14272**
Studijní program: **N2646 Informační technologie**
Studijní obor: **Informační technologie**
Název tématu: **Knihovní informační systém pomocí frameworku Spring**
Zadávající katedra: **Katedra softwarových technologií**

Z á s a d y p r o v y p r a c o v á n í :

Cílem práce je navrhnout a realizovat jednoduchý knihovní IS na platformě Java. V teoretické části potom provést rešerši známých knihovních informačních systémů a seznámit čtenáře s moderními frameworky, které jsou dnes využívány u firem, které vyvíjejí produkty na platformě Java. V práci bude popsán framework Spring, ORM nástroj Hibernate, Maven a vysvětleny principy webových služeb (WSDL, REST). V praktické části budou tyto technologie implementovány v jednoduchém knihovním IS, který bude obsahovat několik poboček knihovny. V rámci každé pobočky bude možné spravovat uživatele a knihovní fond. Budou vytvořeny minimálně tři role - administrátor, knihovník a klient.

Rozsah grafických prací:

Rozsah pracovní zprávy: cca 60 stran

Forma zpracování diplomové práce: tištěná

Seznam odborné literatury:

BAUER, Christian a Gavin KING. Java persistence with Hibernate. rev. ed.

Greenwich: Manning Publications, c2007, xxiii, 408 s. ISBN 19-323-9488-5.

WALLS, Craig. Spring in action. Fourth Edition. Shelter Island, NY: Manning, 2015, xxiv, 600 s. ISBN 161729120x.

AMUTHAN, G. Spring MVC Beginner's Guide. Packt Publishing, 2014. ISBN 1-78328-488-9.

SANDOVAL, José. RESTful Java web services: master core REST concepts and create RESTful web services in Java. Birmingham, U.K.: Packt Pub., 2009, v, 241 s. ISBN 978-1847196460.

Vedoucí diplomové práce:

Ing. Zdeněk Šilar, Ph.D.

Katedra informačních technologií

Datum zadání diplomové práce:

31. října 2015

Termín odevzdání diplomové práce:

13. května 2016



prof. Ing. Simeon Karamazov, Dr.
děkan



prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 15. listopadu 2015

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 11. 5. 2016



Richard Matula

PODĚKOVÁNÍ

Na tomto místě bych chtěl poděkovat panu Ing. Zdeňku Šilarovi, Ph. D. za ochotu a cenné rady při tvorbě této práce. Můj vděk také patří mé rodině za podporu při mých studiích.

ANOTACE

Cílem práce je navrhnout a realizovat jednoduchý knihovní IS na platformě Java. V teoretické části potom provést rešerši známých knihovních informačních systémů a seznámit čtenáře s moderními frameworky, které jsou dnes využívány u firem, které vyvíjejí produkty na platformě Java. V práci bude popsán framework Spring, ORM nástroj Hibernate, Maven a vysvětleny principy webových služeb (WSDL, REST). V praktické části budou tyto technologie implementovány v jednoduchém knihovním IS, který bude obsahovat několik poboček knihovny. V rámci každé pobočky bude možné spravovat uživatele a knihovní fond. Budou vytvořeny minimálně tři role – administrátor, knihovník a klient.

KLÍČOVÁ SLOVA

Knihovní informační systém, Spring, Hibernate, ORM, Maven, webové služby

TITLE

Library information system using Spring framework

ANNOTATION

The aim of the thesis is to design and implement a simple library IS based on Java platform. The theoretical part then perform a search-known library information systems and familiarize the reader with modern frameworks, which are now used by companies developing products on the Java platform. In the thesis will be described framework Spring, ORM tool Hibernate, Maven and explained the principles of Web services (WSDL, REST). In the practical part will be these technologies implemented in an simple library IS, which will include several branches of the library. Within each branch will be able to manage users and library fund. In the application will be at least three roles – administrator, librarian and client.

KEYWORDS

Library information system, Spring, Hibernate, ORM, Maven, web services

Obsah

Obsah	8
Seznam ilustrací a tabulek	10
Seznam zkratek a značek	12
Úvod.....	14
1 Porovnání vybraných knihovných IS.....	15
1.1 Automatizace knihovných procesů	15
1.2 Kritéria pro porovnávání knihovných IS	16
1.3 Knihovní systém Univerzity Pardubice	17
1.4 Knihovní systém města Olomouce.....	18
1.5 Implementovaný knihovní IS	18
1.6 Závěr	19
2 JEE architektura.....	20
2.1 MVC architektura.....	20
2.2 JEE technologie použité na webové vrstvě	21
2.2.1 Servlety	21
2.2.2 JSP a Expression Language	21
2.2.3 EJB.....	22
2.2.4 JDBC.....	23
2.2.5 JPA.....	23
3 Spring Framework	24
3.1 Spring moduly	25
3.1.1 Spring Core Container	25
3.1.2 AOP modul	25
3.1.3 Data access and integration.....	25
3.1.4 Web.....	26
3.1.5 Test.....	26
3.2 Spring MVC Framework.....	27
4 ORM nástroj Hibernate.....	33
4.1 Objektově relační mapování	33
4.2 Persistence v Javě.....	35
4.3 Mapování.....	37
4.4 Konfigurace Hibernate projektu.....	43
4.5 HQL, Query a Criteria API	44
5 Webové služby.....	46

5.1	SOAP.....	46
5.2	REST	47
5.3	REST API pomocí Spring MVC.....	48
6	Implementace knihovního informačního systému.....	52
6.1	Popis praktické části.....	52
6.2	Použité nástroje pro návrh a implementaci	53
6.2.1	Enterprise Architect	53
6.2.2	Java, Spring a Hibernate Framework.....	53
6.2.3	MySQL databáze	54
6.2.4	Apache Tomcat	54
6.2.5	Maven	54
6.2.6	Netbeans IDE.....	55
6.2.7	Technologie použité na webové vrstvě.....	56
6.3	Analýza a návrh.....	56
6.4	Implementace	59
6.5	Představení hotové aplikace	63
	Závěr	68
	Literatura a použité zdroje	69
	Přílohy.....	73

Seznam ilustrací a tabulek

Obrázek 1 – Schéma komunikace jednotlivých komponent MVC architektury	20
Obrázek 2 – MVC architektura na platformě Java	21
Obrázek 3 – Příklad části JSP stránky s JSTL tagy a EL	22
Obrázek 4 – Spring moduly	25
Obrázek 5 – Rozložení jednotlivých funkcí aplikace v rámci modulů	26
Obrázek 6 – Zpracování požadavku DispatcherServletem	27
Obrázek 7 – Příklad konfigurace web.xml souboru	28
Obrázek 8 – Příklad konfigurace souboru dispatcher-servlet.xml.....	29
Obrázek 9 – Příklad anotace @Controller a @RequestMapping	29
Obrázek 10 – Příklad anotace @Service a @Autowired	30
Obrázek 11 – Příklad metody pro přesměrování na stránku s formulářem a jeho zpracování	31
Obrázek 12 – Formulář pomocí Spring form tagů.....	32
Obrázek 13 – Porovnání terminologie objektového a relačního světa	34
Obrázek 14 – Přehled důležitých tříd Hibernate a princip fungování	36
Obrázek 15 – Představení anotací pro mapování třídy	42
Obrázek 16 – Příklad HQL dotazu	44
Obrázek 17 – Dotaz pomocí Criteria API.....	45
Obrázek 18 – Příklad SOAP komunikace.....	46
Obrázek 19 – Komunikace klienta s REST API.....	48
Obrázek 20 – Příklad anotací @RequestBody a @ResponseBody	49
Obrázek 21 – Příklad volání konzumenta REST služeb.....	51
Obrázek 22 – Definice závislostí v souboru pom.xml.....	55
Obrázek 23 – Definice modulů v souboru pom.xml.....	55
Obrázek 24 – Use case diagram knihovního IS	56
Obrázek 25 – Rozdělení tříd do balíčků	57
Obrázek 26 – Třída HibernateUtil	60
Obrázek 27 – Konfigurační soubor Hibernate (hibernate.cfg.xml)	61
Obrázek 28 – REST metoda pro získání všech zaměstnanců	62
Obrázek 29 – Výstup REST dotazu na získání všech zaměstnanců	62
Obrázek 30 – Úvodní obrazovka aplikace	64
Obrázek 31 – Seznam knížek s vyhledáváním	65
Obrázek 32 – Formulář na editaci knihy	65

Obrázek 33 – Obrazovka se zaměstnaneckým profilem.....	66
Obrázek 34 – Obrazovka s vytvářením výpůjčky.....	66
Obrázek 35 – Obrazovka se seznamem zaměstnanců.....	67

Tabulka 1 – Nejdůležitější metody třídy Session	36
Tabulka 2 – Popis hodnot atributu cascade	41
Tabulka 3 – Seznam vlastností souboru hibernate.cfg.xml	43
Tabulka 4 – CRUD operace a jejich HTTP ekvivalenty	47
Tabulka 5 – Popis vybraných metod třídy RestTemplate.....	50
Tabulka 6 – Adresářová struktura Mavenu.....	54
Tabulka 7 – Uživatelsky definované výjimky	59
Tabulka 8 – Book REST API	74
Tabulka 9 – Branch REST API.....	75
Tabulka 10 – Client REST API	76
Tabulka 11 – Employee REST API.....	78

Seznam zkratek a značek

API – Application programming interface

AOP – Aspect-oriented programming

BPMN – Business Process Model and Notation

CRUD – Create, Read, Update, Delete

DAO – Data Access Object

DBMS – Database Management System

DI – Dependency Injection

EJB – Enterprise JavaBeans

EL – Expression Language

HQL – Hibernate Query Language

HTML – Hypertext Markup Language

HTTP – Hypertext Transfer Protocol

IDE – Integrated Development Environment

IS – Informační systém

ISBN – International Standard Book Number

JAR – Java Archive

JDBC – Java Database Connectivity API

JPA – Java Persistence API

JSON – JavaScript Object Notation

JSP – JavaServer Pages

JSTL – JavaServer Pages Standards Tag Library

MARC – Machine-Readable Cataloging

MDT – Mezinárodní desetinné třídění

MVC – Model-View-Controller

MVS – Meziknihovní výpůjční služba

OPAC – Online public access catalog

ORM – Objektově-relační mapování

POJO – Plain Old Java Object

REST – Representational State Transfer

SOAP – Simple Object Access Protocol

SQL – Structured Query Language

UDDI – Universal Description, Discovery, and Integration

UML – Unified Modelling Language

URL – Uniform Resource Locator

UTF-8 – UCS Transformation Format

UUID – Universally unique identifier

WAR – Web Application Archive

WSDL – Web Services Description Language

XML – Extensible Markup Language

Úvod

Vývoj informačních systémů je dnes rozsáhlá disciplína. V současnosti existuje nespočet metodik a nástrojů, které nám k jejich vývoji pomáhají. Jedním z těchto nástrojů je robustní programovací jazyk Java, který je v dnešní době velmi rozšířený.

Práce je zaměřená na framework Spring a Hibernate, přičemž si klade za cíl tyto nástroje popsat a vysvětlit moderní metodiky vývoje software. Vzhledem k tomu, že je práce zaměřena na implementaci vlastního knihovního IS, je součástí práce také provedení řešerše vybraných knihovních IS. Porovnání je pak zpracováno v první kapitole této práce. V práci je dále vysvětlen princip objektově-relačního mapování, MVC architektury, sestavovacího nástroje Maven a v neposlední řadě webových služeb. Cílem práce není popsat všechny tyto nástroje do detailu, ale nastínit základní metodiky vývoje software na platformě Java a těchto konkrétních frameworkcích.

V praktické části je pomocí těchto nástrojů napsán knihovní informační systém. Praktická část práce, včetně zadání, analýzy a návrhu je pak zdokumentována v poslední kapitole.

1 Porovnání vybraných knihovních IS

Všichni máme potřebu někam ukládat a zpracovávat nějaká data. Mohou to být osobní fotky, pracovní dokumenty nebo důležité podnikové informace. S vývojem výpočetní techniky se naskytla možnost tato data, a to nejen podniková, zpracovávat prostřednictvím počítače. Tato data jsou pak v případě webových aplikací přístupná v podstatě odkudkoliv a jejich zpracovávání je několikanásobně rychlejší.

Z toho důvodu vznikly informační systémy, které tato data spravují a dokáží je uživatelům okamžitě zpřístupnit. To samé platí samozřejmě i u knihoven, kde by evidence všech knih, zaměstnanců a čtenářských kont byla jak časově, tak datově náročná.

1.1 Automatizace knihovních procesů

Pro nikoho nebude novinkou, že knihovní informační systémy se vyvíjejí s rozmachem výpočetní techniky a automatizace knihovních procesů je v důsledku množství dat, se kterými se pracuje, žádoucí [34].

Pro automatizaci procesů knihovny vznikly automatizované knihovní systémy. Pro pochopení dalšího textu, si řekneme, co vůbec automatizovaný knihovní systém znamená. Podle [25] je to: „*Aplikační software provozního (transakčního) typu, určený k automatizaci procesů realizovaných v knihovně. Obvykle má modulární strukturu; typické moduly jsou akvizice, katalogizace, OPAC, výpůjčky a MVS, správa seriálů. Zpravidla obsahuje i nástroje pro zapojení do sítě knihoven a pro komunikaci s externími zdroji.*“

Z předchozího odstavce lze pochopit, že vývoj automatizovaných knihovních IS je rozsáhlá disciplína a z důvodu velkého počtu knihovních procesů (nákup knih, správa dokumentů, meziknihovní výpůjční služby, katalogizace, atd.) je použití automatizovaného IS dnes již nezbytné. Historie automatizovaných systémů se datuje někdy s érou mikropočítačů, kdy se tato vědní disciplína začala vyvíjet. V dnešní době existuje mnoho automatizovaných knihovních IS a je jen na knihovně jaký použije nebo jestli se rozhodne pro vlastní řešení [34], [37].

Dnešní automatizované knihovní systémy stojí na modulární architektuře. To znamená, že celek tvoří jednotlivé moduly, a knihovna si může zvolit, které moduly použije. Moduly automatizovaných knihovních systémů shrnuje následující seznam [33]:

- Akvizice – modul má za úkol automatizovat proces výběru a získávání nových dokumentů pro zařazení do knihovního fondu. Je také určen pro vedení objednávek a faktur.

- Katalogizace – tento modul poskytuje informace o dokumentech v knihovním fondu a vkládání informací o nových dokumentech. Záznamy jsou ukládány ve formátu MARC21.
- OPAC – veřejný katalog knihovny, poskytuje informace o dokumentech.
- Výpůjční modul a MVS – výpůjční modul má za zodpovědnost spravovat výpůjčky uživatelů. Vedeny jsou také informace o rezervacích a stavech výtisků (vypůjčený, rezervovaný, vrácený). MVS pak zajišťuje výpůjčky mezi ostatními knihovnami.
- Správa seriálů – používá se v případě, kdy nejsou kupovány pouze jednotlivé tituly, ale jejich celé série. Je schopen tedy objednat nové výtisky seriálů.

V následujícím textu se budeme již věnovat existujícím automatizovaným IS. Dnes existují tyto nejznámější české systémy [34], [37]:

- CLAVIUS – systém firmy ArrowSys ve spolupráci se sdružením KAVka, předností je přizpůsobivost požadavků uživatele na systém.
- LANIUS – vývojem se zabývala také firma ArrowSys v kontaktu s Okresní knihovnou v Táboře.
- Smartlib – produkt od firmy Lumare, systém pro veřejné, odborné, vědecké, školní nebo univerzitní knihovny. Systém je schopen zajistit automatizaci všech knihovních procesů.
- Kpwin a DAWINCI.

V dalším seznamu jsou pak popsány některé ze zahraničních automatizovaných IS [31], [34]:

- BIBIS – produkt firmy SQUARE B. V. Je určen pro všechny typy knihoven a lze si volit jednotlivé moduly.
- ALEPH – systém vyvíjený na půdě Hebrejské univerzity v Jeruzalémě, u nás systém zastupuje firma Ex-Libris. Systém je také rozdělen na moduly, a je jen na knihovně, které moduly použije.
- PC-LIB – systém vyvíjený na Slovensku pro střední a větší knihovny.
- Evergreen – open source knihovní systém vyvíjený na půdě Georgijské veřejné knihovny.
- Koha – další z open source knihovních systémů, vyvíjí jej novozélandská firma Katipo Communications Ltd.

1.2 Kritéria pro porovnávání knihovních IS

V následujících kapitolách bude provedeno srovnání několika vybraných knihovních informačních systémů. Systémy budou porovnávány jako celek, bude tedy porovnávána jak část klientská, tak zaměstnanecká. Prvním vybraným systémem je pardubický univerzitní

knihovní systém. Dalším knihovním systémem je olomoucký městský knihovní IS. Nakonec budou tyto systémy srovnány s tím, který je představen v praktické části této práce. Knihovní IS jsou srovnávány jako celek, tedy jak aplikace pro zaměstnance, tak klientský katalog.

Kritéria pro srovnávání byla tedy následující:

- porovnání klientských částí aplikací,
- zajímavé funkce katalogu a na jakém online katalogu systém stojí,
- s jakými automatizovanými knihovními systémy spolupracují.

1.3 Knihovní systém Univerzity Pardubice

Pardubický univerzitní systém byl volen z toho důvodu, že studenti Univerzity Pardubice mají ke katalogu tohoto systému přístup. Knihovna na svých internetových stránkách [20] uvádí, že knihovní procesy měl do roku 2010 na starosti systém KPwinSQL, knihovna ale v roce 2012 přešla na nástupce tohoto systému, a to systém Verbis [16].

Knihovna poskytuje veřejný online katalog Portaro vyrobený firmou KP-SYS. Pro studenty univerzity je přihlášení přesměrováno na stránky IS STAG, kde se po úspěšném přihlášení dostanou zpátky do svého katalogu. Pro ostatní uživatele jsou přihlašovací údaje čárový kód a rok narození. Tento způsob přihlašování se po průzkumu dalších knihovních IS zdá jako standard v této oblasti. Pro programátora, ale i běžného uživatele, by mohl být zarážející a nemyslím si, že je úplně bezpečný, protože například při ztrátě peněženky se někdo může jednoduše dostat na náš účet. Pro přihlašování je nutné zadat PIN, tedy ne klasické heslo. Volba tohoto PINu lze naštěstí změnit, ale zase jen na číselnou hodnotu, tzn., že by nejspíš nesplňoval zásady pro tvorbu bezpečného hesla. Na druhou stranu ve webové aplikaci po přihlášení nemáme tolik možností, jak takto napadenou osobu nějak poškodit. Nicméně po přístupu do aplikace máme v mnohých IS záložku se svými údaji, jako například bydliště a mnohdy i jiné, které bychom nechtěli určitě nikomu neznámému prozrazovat.

Abychom se ale vrátili k účelu tohoto textu, a tedy porovnání informačních systémů. Při prohlížení webu katalogu v neautorizované formě si můžeme všimnout lokalizace do třech jazyků, na úvodní stránce vyhledávání a seznam novinek. Užitečnou funkcí by mohl být rejstřík, podle kterého pak můžeme také vyhledávat podle různých kritérií. Následuje záložka zajímavé tituly, kde jsou knihy seřazené podle různých kritérií (novinky, nejsledovanější, nejlépe hodnocené a nejpůjčovanější). Webový katalog také poskytuje statistiky. Po přihlášení

se nám pak v záložce můj účet zobrazí osobní údaje spolu s informacemi o výpůjčkách, rezervacích, atd.

1.4 Knihovní systém města Olomouce

Dalším knihovním IS je olomoucký městský knihovní IS. Tento knihovní IS stojí na systému Clavius. Zaměstnanci tedy pracují s klasickou desktopovou aplikací, kde provádí knihovní procesy jako vytváření výpůjček, rezervací, prodlužování výpůjček, správu knihovního fondu, atd.

Jako online katalog olomoucká knihovna používá katalog Carmen od firmy Lanius. Katalog pro nepřihlášené uživatele poskytuje informace o novinkách, vyhledávání knih, zobrazení nejlépe hodnocených knih a seznamu knih podle nejlepšího hodnocení. Webové rozhraní je lokalizované do pěti jazyků. Po přihlášení do IS si můžeme prohlížet své čtenářské konto. Zajímavé jsou funkce pro hodnocení knížky a funkce „Vybrat“, která nám danou knížku přidá do seznamu našich oblíbených a v záložce „Vybrané položky“ si je pak můžeme prohlížet. Rozhraní webové aplikace si také můžeme pomocí uživatelského nastavení přizpůsobit podle sebe. Dále si pak samozřejmě můžeme prohlížet své rezervace, výpůjčky a poplatky, které jsme knihovně platili. Užitečnou funkcí z uživatelského hlediska je zobrazení posledního přihlášení, poslední návštěvy knihovny a návrh k nákupu knížky. Na záložce „Navrhnout k nákupu“ pak vyplníme údaje jako jméno knihy, jméno autora, ISBN, nakladatelství, rok a odešleme návrh. Knihovna se pak rozhodne, zda danou knížku koupí do svého fondu. Poslední zajímavou funkcí jsou statistiky, kde si na webu můžeme nechat zobrazit souhrn různých typů statistik seskupených podle data (statistika přístupů, přihlašování, hodnocení a hledání).

1.5 Implementovaný knihovní IS

Nakonec bude tedy uvedeno srovnání těchto IS s tím, který je implementován v praktické části této práce. Implementovaný knihovní IS není tak komplexní řešení, jako již dostupné automatizované knihovní IS, tedy neposkytuje všechny moduly.

V knihovním IS implementovaným v této práci je přístup do autorizované zóny povolen na základě emailu a hesla, a to jak pro klienta, tak i pro zaměstnance. Tím by mělo být dosaženo vyššího zabezpečení. Pro nepřihlášené uživatele je zde možnost si tedy prohlížet webové stránky, a to informace o knihovně a katalog knih. Po úspěšném přihlášení se dostáváme do zabezpečené zóny webu a pro klienty se odkrývají záložky pro prohlížení aktuálních a předchozích vypůjčených knížek a rezervací. Po přihlášení v roli zaměstnance se odkrývají

větší možnosti, ty jsou pak popsány v praktické části práce. Aplikace pro svůj účel splňuje v této verzi dostatečnou funkcionalitu pro správu celé knihovny a po nasazení by v této verzi mohl fungovat jako knihovní IS se základními knihovními procesy.

Implementovaný knihovní IS se od předchozích dvou liší v tom, že zobrazuje informace o knihovně a zároveň poskytuje funkcionalitu zpracování dat (správa čtenářského katalogu, vytváření výpůjček). Naproti tomu předchozí dva systémy mají odděleně webovou stránku, výpůjční systém a katalog, kde si pak uživatel může prohlížet své konto.

1.6 Závěr

Na závěr této rešeršní části je třeba dodat, že vývoj automatizovaných knihovních systémů je rozsáhlá disciplína. Na vývoj takového komplexního systému je většinou potřeba celý vývojový tým. Ten si ale většina dnešních knihoven nedokáže sama financovat, a proto si většinou volí nějaké již existující řešení. Těchto řešení (výše zmíněných) je na trhu v dnešní době mnoho a je tedy jen na knihovně, jakým způsobem bude své procesy automatizovat.

2 JEE architektura

Než si řekneme něco o frameworkích, je nutné znát, jak vůbec pracuje Java na serveru. Tomu se bude věnovat následující kapitola.

V dnešní době je nejčastější požadavek mít vše po ruce. Tento požadavek je samozřejmě i v informatice a spousta aplikací se tedy píše z důvodu dostupnosti jako tenký klient, tedy pro web. Pro business logiku, práci s databází a vůbec správu webového kontextu lze použít nespočet programovacích jazyků. Java pro tento případ nabízí svoji Enterprise edici [21].

2.1 MVC architektura

Z důvodu modulárnosti programů, tedy rozdělení programu do logických celků, vznikl návrhový vzor model-view-controller, který JEE architektura ve svém jádru implementuje. Tento návrhový vzor rozděluje aplikaci do třech samostatných vrstev, kde každá vrstva má svou zodpovědnost a komunikuje, tedy předává informace ostatním vrstvám. Existují samozřejmě i jiné návrhové vzory pro vývoj webových aplikací, nicméně MVC je nejrozšířenější a stal se de facto standardem [22].

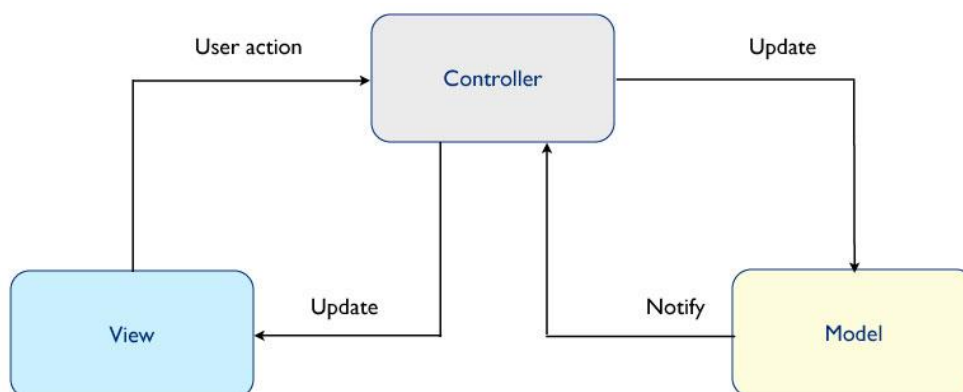
To, co mají jednotlivé vrstvy za úkol je popsáno níže [22].

Model – tato vrstva reprezentuje objekty nebo data, se kterými se v aplikaci pracuje.

View – vrstva view (pohledu) se stará o zobrazení dat, tedy modelu.

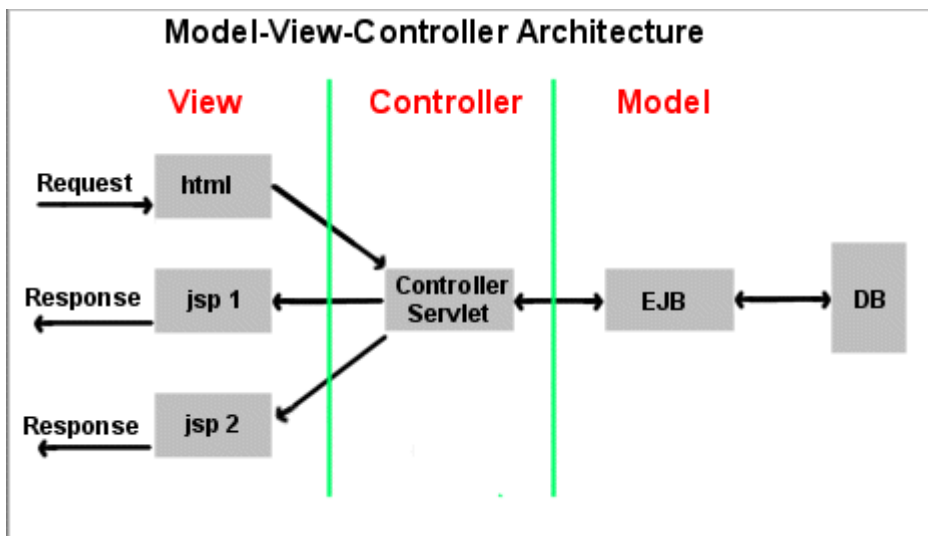
Controller – neboli řadič, je objekt zodpovědný za aktualizaci modelu a generování obsahu (view). Controller tedy zpracovává požadavky a generuje odpověď.

Obrázek č. 1 popisuje schéma komunikace jednotlivých komponent MVC architektury.



Obrázek 1 – Schéma komunikace jednotlivých komponent MVC architektury 70 [19]

Schéma, jak může vypadat MVC architektura konkrétně na Java platformě, je na následujícím obrázku (Obrázek č. 2).



Obrázek 2 – MVC architektura na platformě Java [28]

2.2 JEE technologie použité na webové vrstvě

Tato podkapitola představí základní technologie, které Java používá na webové vrstvě. Těchto technologií je mnoho, proto zde budou zmíněny pouze ty nejdůležitější.

2.2.1 Servlety

Servlety se píší jako klasické Java třídy, s tím, že reagují na určité události (například GET nebo POST požadavky). Servlet tedy v aplikaci pracuje jako controller [8], [9], [21].

2.2.2 JSP a Expression Language

JSP, neboli JavaServer Pages technologie se používá pro generování dynamického obsahu. Při kompilaci se z těchto stránek stane servlet a rozhoduje, jaký obsah má být přidán do HTML stránky. Textový dokument založený na JSP technologii se pak sestává ze statického obsahu (XML nebo HTML) a JSP elementů. U JSP stránek je třeba zmínit ještě knihovnu JSTL, která rozšiřuje JSP technologii o funkce jako procházení kolekcí, formátování, SQL tagy, práci s XML a další funkce [8], [9], [21].

JSTL se tedy skládá z následujících skupin [8], [9], [21]:

- core tagy,
- formatting tagy,
- SQL tagy,

- XML tagy,
- JSTL funkce.

Dalším mechanismem pro komunikaci mezi pohledovou vrstvou a modelem je Expression Language. EL umožňuje přistupovat k datům přímo v JSP pomocí speciálních výrazů. K těmto datům se přistupuje pomocí znaku „\$“ a následují složené závorky obsahující daný výraz. Výraz také může obsahovat speciální operátory pro práci s daty. Pro seznam operátorů a další informace o EL lze čtenáře odkázat například na [21].

Příklad JSP kódu s core JSTL tagy a EL je na následujícím obrázku (Obrázek č. 3). Kód zobrazuje HTML tabulku, kde se pomocí cyklu for-each prochází kolekce objektů (zde zaměstnanců). K této kolekci se přistupuje pomocí EL `${employees}` a atributy každého zaměstnance se pak vypisují do buněk tabulky. O výpis se stará `<c:out>` tag a hodnotou je pak příslušný výraz EL jazyka.

```
<table class="table table_text">
  <thead>
    <th>Jméno</th>
    <th>Příjmení</th>
    <th>Email</th>
  </thead>
  <c:forEach items="${employees}" var="employee" >
    <tr>
      <td>
        <c:out value="${employee.name}" />
      </td>
      <td>
        <c:out value="${employee.surname}" />
      </td>
      <td>
        <c:out value="${employee.email}" />
      </td>
    </tr>
  </c:forEach>
</table>
```

Obrázek 3 – Příklad části JSP stránky s JSTL tagy a EL

2.2.3 EJB

Enterprise JavaBeans (EJB) jsou serverové komponenty, sloužící pro implementaci business logiky. Existují dva druhy těchto bean – session beany a message-driven beany [8], [9], [21].

2.2.4 JDBC

Je to API poskytující funkce pro připojení do databáze a komunikaci s ní. Toto API má 2 části [8], [9], [21]:

- application-level interface – pro připojení do databáze,
- service provider interface – sloužící k připojení JDBC driveru k JEE platformě.

2.2.5 JPA

Toto API poskytuje funkce pro persistenci dat v Javě, tedy i ORM. Blíže bude JPA zmíněno v kapitole o frameworku Hibernate [8], [9], [21].

3 Spring Framework

Po lehkém úvodu do programování webových aplikací v Javě přijde konečně řeč na Framework Spring, ve kterém je pak napsána praktická část této práce. Tento open-source framework vznikl hlavně z důvodu usnadnění vývoje tzv. enterprise aplikací. Spring je tedy odlehčený kontejner a je postaven na DI (Dependency Injection) a AOP (Aspect-oriented Programming).

DI je koncept, který řeší to, aby byly třídy na sobě co nejméně závislé. Řeší volné vazby (loose coupling) mezi objekty, a to jejich injekcí pomocí předání parametrů. Objekty tedy nejsou inicializovány klasickou cestou, ale o tuto inicializaci se stará jiný subjekt. Tato injekce je vyvolána právě Springem, který všechny tyto objekty spravuje [32], [36].

Dependency injection představuje 3 základní způsoby předání závislostí [40]:

- setter injection,
- constructor injection,
- interface injection.

Pro představení toho, proč vůbec framework Spring vznikl, si můžeme vzít příklad z článku pana Matulíka [26], který uvádí, že: „*Tvorba jakékoliv netriviální aplikace obnáší nutnost zabývat se některými běžně se vyskytujícími aspekty návrhu, jakými jsou například autentizace a autorizace, správa transakcí, trvalé uložení dat v databázi apod. Dalším problémem běžných aplikací je slabá strukturovanost kódu. Tu lze řešit použitím některých návrhových vzorů, ovšem i takováto vlastní dekompozice kódu představuje zbytečné stále se opakující kódování.*

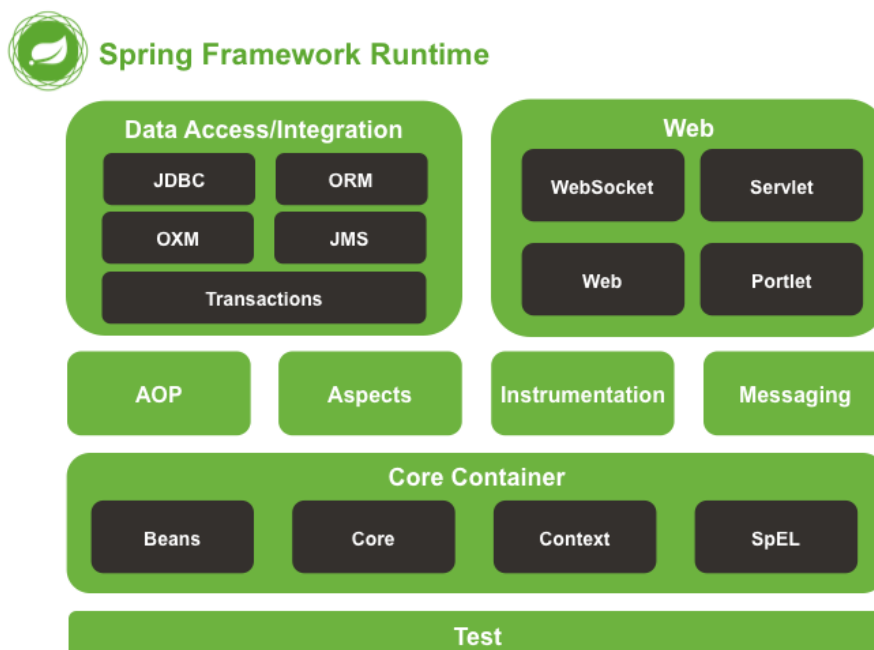
Framework (někdy označován jako kontejner) tyto problémy řeší tak, že běžně se opakující funkce aplikací implementuje ve svých vlastních třídách a poskytuje tak hotovou infrastrukturu, kterou aplikace pouze využívají. Aplikační programátor se tak může soustředit jen na vlastní problém a zbytek nechat na frameworku, jenž je jeho aplikací využíván.“

Poslání tohoto článku je jasné, neplatí to tedy jen pro Spring, ale obecně pro jakýkoliv framework.

Spring pracuje na modulární architektuře, tedy jeho funkčnost je rozdělena do jednotlivých modulů [40]. Popisu těchto modulů se věnuje následující podkapitola.

3.1 Spring moduly

V úvodu do Frameworku Spring bylo zmíněno, že stojí na modulech. Moduly jsou zobrazeny na obrázku č. 4 a ty nejdůležitější budou v následující podkapitole popsány detailněji.



Obrázek 4 – Spring moduly [24]

3.1.1 Spring Core Container

Tento kontejner je srdce frameworku a obsahuje Beans, Core, Context a Expression language moduly. Také samozřejmě poskytuje DI kontejner pro vytváření, konfiguraci a správu bean [24].

3.1.2 AOP modul

Pichlík [32] popisuje AOP modul jako: „AOP je modul implementující podporu pro aspektově orientované programování. Umožňuje separovat části kódu prolínající se celou aplikací (autorizace, logování, transakce) do takzvaných aspektů a jejich následnou aplikaci na jakýkoli POJO objekt. Využití AOP modulu se prolíná celým frameworkem a jedná se o jednu z nejsilnějších vlastností Springu.“

3.1.3 Data access and integration

Tato vrstva poskytuje základní moduly pro práci s daty a integraci. Poskytuje tedy například i funkce pro objektově-relační mapování ve svém ORM modulu. Pomocí ORM modulu

můžeme pak aplikaci integrovat s některým z ORM nástrojů. Tato vrstva také obsahuje DAO modul, který se používá pro zjednodušení práce s databází [40].

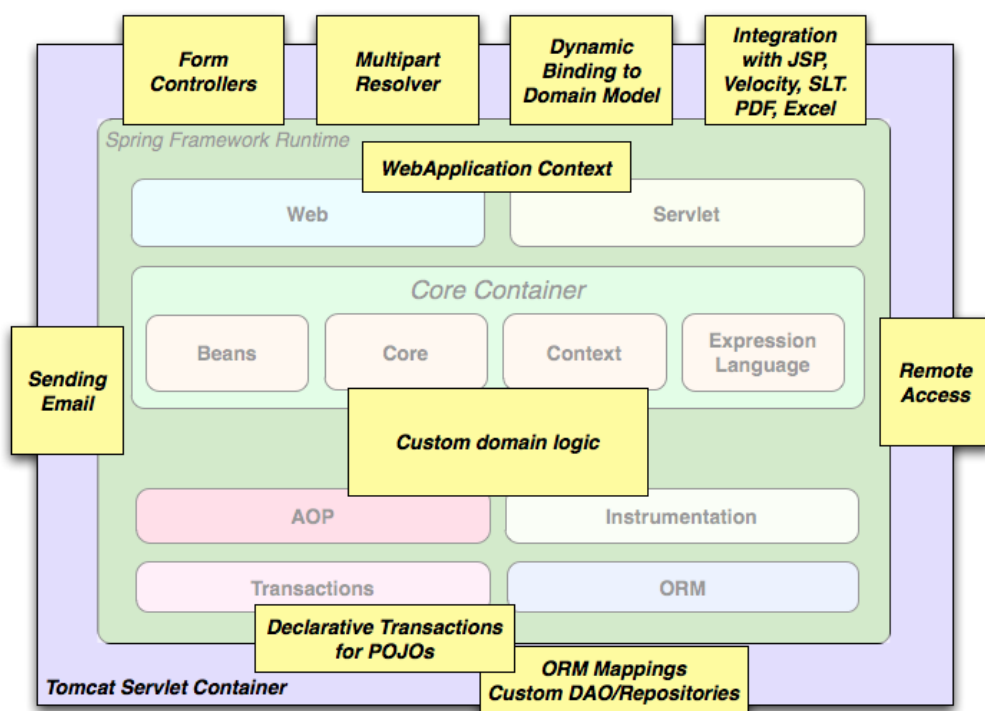
3.1.4 Web

Nejdůležitějším modulem z vrstvy Web je modul spring-mvc. Ten obsahuje implementaci návrhového vzoru MVC a REST služeb [24], [40].

3.1.5 Test

Vrstva Test nám umožňuje provádět testy pomocí JUnit nebo TestNG. Poskytuje tedy funkcionální pro jednotkové a integrační testy [24].

Po představení následujících modelů si už dokážeme udělat obrázek o tom, jaké části aplikace stojí na daných modulech. Názornou ukázkou architektury webové aplikace postavené na Springu lze představit na následujícím obrázku (Obrázek č. 5).



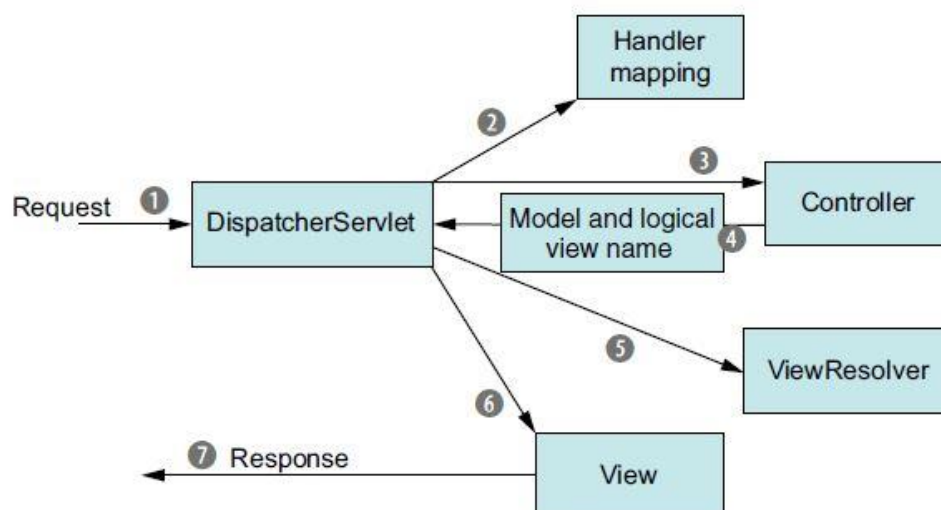
Obrázek 5 – Rozložení jednotlivých funkcí aplikace v rámci modulů [24]

3.2 Spring MVC Framework

V předchozích kapitolách byl představen Framework Spring v jeho základní podobě. Tato kapitola se zaměří jen na jeho MVC část, a to včetně její konfigurace, která je pak použita v praktické části.

Webové aplikace pracují na principu požadavek – odpověď. Tyto HTTP požadavky přicházejí zejména z webového prohlížeče a jedná se nejčastěji o požadavek na URL adresu nebo zpracování formuláře. Jako odpověď je pak generován view, tedy například výsledná HTML stránka. O tuto činnost se, jak již bylo zmíněno výše, stará controller. Než se ale tento požadavek dostane k controlleru, je nejdříve zpracován vstupním bodem do každé Spring webové aplikace, a to třídou `DispatcherServlet`. Třída `DispatcherServlet` je součástí návrhového vzoru Front Controller, který pracuje na principu jednoho obecného controlleru a ten pak deleguje požadavky na jednotlivé konkrétní řadiče. Jak již vyplývá z názvu – dispatcher, jedná se tedy vlastně o takového dispečera, který řídí, co se s jakým požadavkem stane a k jakému handleru (metodě) bude přiřazen [1], [40].

Funkčnost `DispatcherServletu` si lze vysvětlit na následujícím obrázku (Obrázek č. 6) .



Obrázek 6 – Zpracování požadavku `DispatcherServletem` [40]

Při zpracování požadavku `DispatcherServletem` je nejdříve rozhodnuto, který controller přebírá za tento požadavek odpovědnost. O pomoc je požádána třída `HandlerMapping`, která nám vrátí název controlleru, jež bude daný požadavek zpracovávat. Controller následně zpracuje požadavek a vygeneruje model (data) a vrátí logický název view, který má být s daty

následně zobrazen. Všechny tyto informace jsou vráceny nazpět třídě DispatcherServlet, která si spojí logické jméno view s daty a požádá view resolver o fyzickou instanci view, tedy například JSP stránku. Po výběru view je tento spolu s daty vrácen jako odpověď [40].

Aby mohlo tohle všechno fungovat, musí se Spring aplikace nakonfigurovat. Konfigurace DispatcherServletu probíhá v souboru web.xml, kde je třeba tuto třídu patřičně zaregistrovat. Dále je potřeba nakonfigurovat soubor dispatcher-servlet.xml, kde jsou informace o uložení .jsp souborů, registraci bean, <context:component-scan...> tag pro správné naskenování balíčku s controllery a <mvc:annotation-driven /> tag pro aktivaci Spring MVC a jeho anotací [1], [23].

Konfiguraci web.xml souboru si lze představit na následujícím obrázku (Obrázek č. 7).

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/applicationContext.xml</param-value>
  </context-param>
  <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
  </listener>
  <servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>2</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>
      30
    </session-timeout>
  </session-config>
  <welcome-file-list>
    <welcome-file>redirect.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

Obrázek 7 – Příklad konfigurace web.xml souboru

Část zdrojového kódu konfigurace souboru dispatcher-servlet.xml ilustruje obrázek č. 8.

```

<bean class="org.springframework.web.servlet.mvc.support.ControllerClassNameHandlerMapping"/>
<context:component-scan base-package="com.librarysystem.web.Controllers" />
<mvc:annotation-driven />

<bean id="viewResolver"
      class="org.springframework.web.servlet.view.InternalResourceViewResolver"
      p:prefix="/WEB-INF/jsp/"
      p:suffix=".jsp" />

<bean name="indexController"
      class="org.springframework.web.servlet.mvc.ParameterizableViewController"
      p:viewName="index" />

<mvc:resources mapping="/resources/**" location="/resources/" />

```

Obrázek 8 – Příklad konfigurace souboru dispatcher-servlet.xml

Controller je klasická Java třída rozšířená o anotaci `@Controller`. Pomocí této anotace potom překladač zjistí, že se jedná o controller a tento pak zaregistruje jako beanu. Další důležitou anotací je `@RequestMapping`, která pak už mapuje samotnou URL adresu na metodu v daném controlleru [1], [23], [40].

Důležitou třídou je třída Model. Ta nám zajistí navázání dat do pohledu. Tato třída obsahuje metodu pro přidání atributů a jejich hodnot, které se pak v pohledu vykreslují nebo případně jinak zpracovávají [1], [23], [40].

Na následující části kódu v obrázku č. 9 je předvedeno použití anotace `@Controller` a `@RequestMapping`.

```

@Controller
@RequestMapping("/clients")
public class ClientsController {

    private final RestTemplate template = new RestTemplate();
    private final String LIBRARY_SYSTEM_BE_ENDPOINT = "http://localhost:8084/librarySystem-be/rest/";

    private final static Logger LOGGER = Logger.getLogger(ClientsController.class.getName());

    @RequestMapping("/")
    public String index(Model model, HttpServletRequest request) {
        Book[] newBooks = template.postForObject(LIBRARY_SYSTEM_BE_ENDPOINT +
            "book/getNewBooks", null, Book[].class);
        model.addAttribute("newBooks", newBooks);
        return "index";
    }

    @RequestMapping(value = "/loginPage")
    public String loginPage(Model model) {
        Client client = new Client();
        model.addAttribute("login", client);
        return "login";
    }
}

```

Obrázek 9 – Příklad anotace `@Controller` a `@RequestMapping`

Při psaní aplikace ale nepíšeme jen controllery. Business logika aplikace a kód pro přístup k datům je samozřejmě taky v nějakých třídách. Pro tyto třídy má také Spring speciální anotace. DAO třídy, neboli ty, které pracují s datovým úložištěm, se anotují `@Repository`. Další anotací je `@Service`, kterou se pak anotují třídy obsahující business logiku (tzv. servisní třídy). Tak jako Spring vytváří a spravuje beanu pro každý zaregistrovaný controller, i třídy anotované jako `@Repository` a `@Service` spravuje tímto způsobem [1].

Návrhový vzor dependency injection byl zmíněn výše. Proto, abychom mohli předat závislost do nějaké třídy, používá Spring anotaci `@Autowired`. Tato anotace pak Springu říká, aby pro danou beanu, která má být předána našel existující referenci příslušného objektu [1], [40].

Příklad anotace `@Service` a `@Autowired` je na následujícím obrázku (Obrázek č. 10).

```
@Service
public class BooksService implements IBooksService {

    @Autowired
    private IBooksDao booksDao;
    @Autowired
    private IBranchDao branchDao;
```

Obrázek 10 – Příklad anotace `@Service` a `@Autowired`

Další užitečnou funkcí Spring MVC jsou Spring form tagy. Je to rozšíření JSTL jazyka o speciální tagy, pomocí kterých lze navázat modelové třídy na prvky formuláře. Jde tedy o navázání dat na tyto prvky a pomocí toho jsme ušetřeni zbytečného parsování dat z formulářů [40].

Na následujících dvou obrázcích (Obrázek č. 11 a Obrázek č. 12) si představíme, jak toto navázání tříd na prvky formuláře může vypadat.


```

@RequestMapping(value = "/registerClientPage")
public String registerClientPage(Model model) {
    Client client = new Client();
    model.addAttribute("newClient", client);
    return "clients/registerClient";
}

@RequestMapping(value = "/registerClient")
public String registerClient(@ModelAttribute("newClient") Client client,
    RedirectAttributes redirectAttribute) {
    boolean valid = validateClient(client);

    //hash password
    client.setPassword(WebUtils.hashPassword(client.getPassword()));

    if (valid) {
        try {
            int id = template.postForObject(LIBRARY_SYSTEM_BE_ENDPOINT +
                "client/addClient", client, Integer.class);
            redirectAttribute.addFlashAttribute("client_registered", true);
        } catch (Exception e) {
            redirectAttribute.addFlashAttribute("client_registered", false);
        }
    } else {
        redirectAttribute.addFlashAttribute("client_registered", false);
    }

    return "redirect:/clients/registerClientPage";
}

```

Obrázek 11 – Příklad metody pro přesměrování na stránku s formulářem a jeho zpracování

Na obrázku č. 11 jsou tedy dvě metody. První metoda registruje požadavek na registraci klienta a vrací stránku registerClient spolu s objektem třídy Client, na který budou vázány prvky formuláře. Druhá metoda už je pak pro samotné zpracování formuláře. Pomocí anotace @ModelAttribute metoda dokáže přijmout data z formuláře jako objekt. Zde tedy přijatý objekt nového klienta zpracujeme a zjistíme, jestli je validní, zkonvertujeme heslo do hashe a nakonec odesíláme pomocí REST volání na backend. Pomocí třídy RedirectAttributes můžeme ještě přidat atributy, které pak můžeme vykreslit do pohledu po přesměrování na jinou stránku [1].

Příklad JSP stránky s formulářem je zobrazen na následujícím obrázku (Obrázek č. 12).

```

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>

<form:form modelAttribute="newClient" action="registerClient" method="POST">
  <table class="table-small">
    <tr>
      <td><label>Jméno:</label></td>
      <td><form:input class="form-control" path="name"></form:input></td>
    </tr>
    <tr>
      <td><label>Příjmení:</label></td>
      <td><form:input class="form-control" path="surname"></form:input></td>
    </tr>
    <tr>
      <td><label>Email:</label></td>
      <td><form:input class="form-control" path="email"></form:input></td>
    </tr>
    <tr>
      <td><label>Heslo:</label></td>
      <td>
        <form:input class="form-control" path="password" type="password"></form:input>
      </td>
    </tr>
    <tr>
      <td colspan="2" align="center"><input class="btn btn-success"
        type="submit" value="Registrovat" /></td>
    </tr>
  </table>
</form:form>

```

Obrázek 12 – Formulář pomocí Spring form tagů

Nejdříve se pomocí taglib přidá knihovna core a form a následně je vytvořen samotný formulář. Je zde vidět provázání objektu s formulářem atributem modelAttribute a atribut action, určující, jaká akce se má po potvrzení formuláře provést. Každý si určitě všiml, že pro vytvoření formuláře není použit klasický form HTML tag, ale form tag ze Spring knihovny. Je zde použit i na vstupy (HTML input tagy) a pomocí těchto prefixů je tedy možné data navázat na dané prvky formuláře [40].

4 ORM nástroj Hibernate

ORM nástroj Hibernate je nástroj pro objektově-relační mapování. K tomu, abychom si vysvětlili, jak samotný nástroj pracuje, je potřeba znát principy ORM.

Při vývoji software se setkáváme s rozdílnými způsoby popisu dat. První, používaný v databázovém světě, je relační princip, tedy ukládání dat do tabulek o řádcích (samotných informací) a sloupcích (jejich atributů). Druhý princip je objektově orientovaný. Popisuje tedy vlastnost něčeho pomocí objektu, který má určité vlastnosti a odkazy na jiné objekty. Při spojení těchto dvou světů nám ale vznikne problém, jak namapovat atributy z tabulky na atributy objektu. Vznikne nám velké množství kódu, a to v důsledku dotazů do databáze, ale i převedení těchto dat do objektové formy. Proto vzniklo objektově-relační mapování [41].

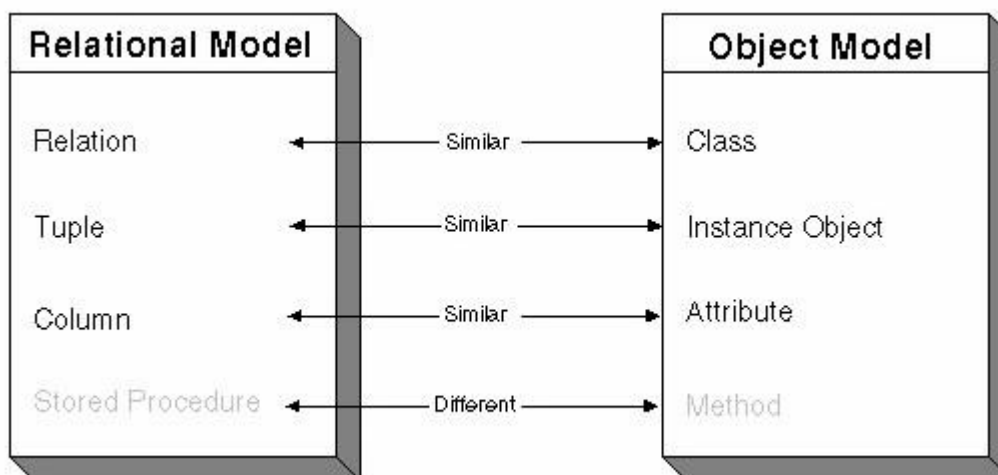
4.1 Objektově relační mapování

Pomocí klasického psaní dotazů do Java kódu pomocí JDBC lze docílit stejné funkčnosti jako pomocí nástrojů objektově relačního mapování (ORM), ale za cenu více kódu a možnosti chyb v kódu při špatně napsaném dotazu [5].

Problém objektově-relačního principu, tedy převod mezi objekty a relačním uložením dat se nazývá „Object-Relational Impedance Mismatch“ nebo také někdy jen „Paradigm mismatch“. Při ukládání a načítání objektů z relační databáze potom nastávají následující problémy [14], [41]:

- Dědičnost – relační databázový systém standardně nepodporuje dědičnost tak, jak ji známe z objektového světa.
- Granularita – v některých situacích máme v našem objektovém modelu více tříd než tabulek v databázi.
- Identita – v relačním databázovém systému má každý řádek svůj jednoznačný klíč – identifikátor, pomocí kterého se dají objekty porovnávat. Java definuje pro tento účel více způsobů, a to pomocí totožnosti objektů (==) a pomocí metody equals (rovnost objektů).
- Asociace – v objektově orientovaném světě jsou asociacemi myšleny vazby mezi objekty, tedy jejich reference na jiné objekty. V relačním světě jsou to pak cizí klíče, tedy odkaz do řádku jiné tabulky.
- Data navigation – přístup k datům v objektově orientovaném světě je odlišný od přístupu k relační databázi. V Javě se k referencím přistupuje pomocí get metod, například `objA.getObjB().getString()`. V relačním světě bychom pak k získání těchto dat museli provést připojení jedné tabulky k druhé pomocí operace join a následně vybrat potřebné informace.

Na následujícím obrázku (Obrázek č. 13) je vysvětleno porovnání terminologie objektového a relačního světa.



Obrázek 13 – Porovnání terminologie objektového a relačního světa [2]

ORM je tedy převod dat mezi relačními databázemi a objektově orientovanými jazyky. ORM také řeší problémy jmenované výše [14].

Podle [5] lze stupně ORM dělit následovně:

- Pure relational (čistě relační) – celý proces ukládání, načítání a práce s daty je založen na klasických SQL dotazech do relační databáze. Toto řešení lze doporučit pro malé aplikace. U velkých aplikací se toto řešení nedoporučuje, protože pak roste kód řešící CRUD operace.
- Light object mapping – entity jsou reprezentovány jako třídy a jsou mapovány manuálně do tabulek. Kód je pak odstíněn od business logiky a toto řešení je možné použít pro aplikace s menším počtem entit.
- Medium object mapping – objektový model aplikace se ukládá pomocí generovaných SQL dotazů při sestavování aplikace nebo pomocí některého z frameworků. Zde již existuje podpora asociací, dotazů pomocí objektově orientovaného jazyka, či načítání objektů z cache. Tato úroveň by byla vhodná pro středně velké aplikace.
- Full object mapping – podporuje vše z objektového modelování, tedy kompozici, dědičnost a polymorfismus. Tato úroveň podporuje také zaváděcí strategie, jako lazy a eager fetching a „caching“ strategie. Tato úroveň se používá u komplexních enterprise aplikací.

Hibernate framework lze označit tedy jako full ORM nástroj [5].

4.2 Persistence v Javě

Každá aplikace potřebuje ke svému běhu nějaká data. Aby mohla tato data být použita i při dalším spuštění aplikace, je potřeba je trvale uložit na disk. Data v operační paměti jsou totiž krátkodobá a po ukončení aplikace bychom o ně přišli. Persistence se tedy zabývá trvalým uložením dat.

V předchozí kapitole o ORM padlo pár úvodních slov o tom, jak ORM pracuje. Aby byla tato funkčnost správná a opravdu se nám podařilo uložit objekty do databáze, musíme si nadefinovat entitní třídy. Tyto entitní třídy pak představují tabulky v databázi a objekty těchto tříd jsou pak řádky v těchto tabulkách [38].

Entitní třídy musí splňovat následující podmínky [38]:

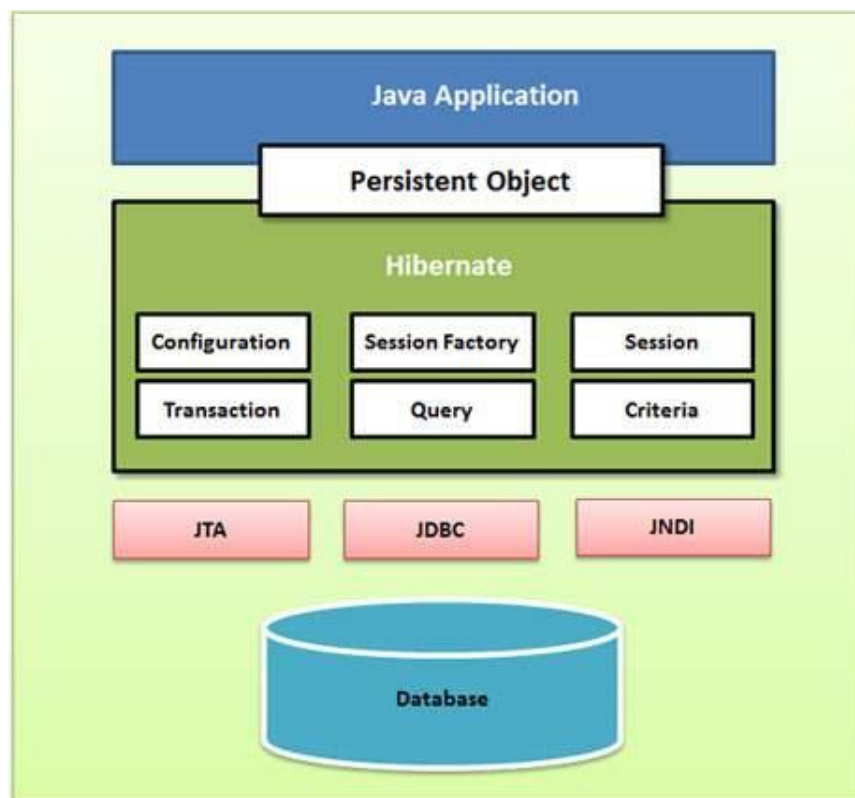
- Třída musí mít anotaci `@Entity`.
- Třída musí mít bezparametrický `public` nebo `protected` konstruktor.
- Třída nesmí být deklarována jako `final`. `Final` modifikátor nesmí být použit i na žádnou metodu nebo persistentní proměnnou.
- Pokud bude instance pracovat například se `session bean` typu `remote`, je potřeba, aby třída implementovala rozhraní `Serializable`.
- Atributy musí být deklarovány jako `protected`, `private` nebo `package-private` a přístup k nim musí být pomocí metod – `getterů` a `setterů`.

Java k persistenci dat specifikuje Java Persistence API zmíněné v kapitole 2.2.5, nicméně není to samotná implementace, ale pouze specifikace. Jednou z implementací JPA je právě Hibernate [5]. Přehled důležitých tříd pro persistenci dat a princip fungování Hibernate představuje obrázek č. 14.

Lze říct, že třídou, na které celá funkčnost Hibernate stojí, je `SessionFactory`. Tato třída se stará o vytváření tzv. `Session` objektů, tedy otvírání, zavírání a správu jednotlivých sezení. Tyto `Session` objekty potom už mají přístup k samotným datům a poskytují CRUD operace. V aplikaci se mohou instance vyskytnout ve 3 stavech [5], [15], [17]:

- `transient` – instance není svázána s žádným konkrétním objektem třídy `Session` a není tedy persistentní (není svázána s žádným konkrétním řádkem tabulky), o životní cyklus těchto objektů se stará `garbage collector`,
- `persistent` – objekt asociovaný s objektem třídy `Session`, má reprezentaci v databázi (s primárním klíčem),

- detached – objekt, který byl persistentní, ale už není – například při odpojení z databáze.



Obrázek 14 – Přehled důležitých tříd Hibernate a princip fungování [11]

Jelikož se třída Session používá pro samotnou práci s databází, v následující tabulce (Tabulka č. 1) si představíme některé důležité metody, které třída nabízí [15], [17].

Tabulka 1 – Nejdůležitější metody třídy Session

Název metody	Popis
beginTransaction()	značí začátek kódu, který se týká transakce a vrací Transaction objekt
close()	uzavírá sezení s databází
createCriteria()	vrací instanci třídy Criteria a tuto instanci vytváří pro třídu, která je vstupem do metody
createQuery()	vytváří instanci třídy Query pro HQL dotaz, který je vstupem do metody
delete()	odstraní persistentní objekt z databáze, z objektu se pak stává transient objekt
get()	vrací persistentní objekt z datového úložiště na základě třídy a id, v případě že objekt neexistuje, vrací null

Název metody	Popis
save()	uloží transient objekt do databáze, ten se pak stává persistentní
saveOrUpdate()	uloží nebo aktualizuje objekt v databázi
update()	aktualizuje daný objekt

Další třídou je Transaction, která se stará o vytváření transakcí. Část kódu, kterou chceme provést atomicky, tedy ta, které se transakce týká, je pak ohrazena voláním metody beginTransaction() třídy Session a metodou commit() potvrzující transakci. Když pak aplikace například spadne v polovině provádění tohoto kódu, je proveden tzv. roll-back a všechna data se vrátí do původního stavu [5].

Pomocí třídy Query můžeme psát dotazy do databáze, a to pomocí klasického SQL nebo pomocí speciálního jazyka Hibernate, a to HQL. HQL je plně objektově-orientovaný dotazovací jazyk a použití bude popsáno níže. Další třídou, pomocí které se můžeme dotazovat do databáze, je třída Criteria. Jak již název napovídá, můžeme si určit kritéria, pomocí kterých se budou data vybírat. Výhoda použití této třídy je, že nemusíme používat klasické řetězce pro dotazování [5].

Poslední třídou je Configuration. Ta se používá k získání SessionFactory objektu. Třída tedy čte konfiguraci frameworku Hibernate ze souboru hibernate.cfg.xml a pomocí metody configure() a buildSessionFactory() pak získáme potřebný objekt pro práci s databází. Konfigurace souboru hibernate.cfg.xml bude popsána níže [5], [40].

4.3 Mapování

Než se dostaneme k samotnému mapování, řekneme si něco o scénářích vývoje aplikace. V praxi je známé, že vývoj aplikace by měl začít návrhem modelu (zde databázového schématu). Mohou nám tedy nastat takové dva hlavní případy [40].

V prvním případě máme již databázové schéma navrhnuté nebo vyvíjíme aplikaci na již hotovém schématu, tedy nad databází, která někde už reálně běží. Tomuto scénáři se také říká bottom-up. U tohoto případu můžeme užít některého z reverse engineering nástrojů a vygenerovat si příslušné Java třídy nebo konfigurační XML soubor z metadat databázového schématu. Nicméně při tomto scénáři není možné vše generovat automaticky a k vytváření modelových tříd je potřeba i ruční zásah [40].

Druhý případ nám nastane tehdy, když vyvíjíme úplně novou aplikaci nebo část aplikace s vlastním databázovým schématem a databázové schéma zatím nemáme. Můžeme si samozřejmě databázové schéma navrhnout v některém z dostupných programů, ale Hibernate pro generování schématu nabízí svůj vlastní mechanismus. Tento scénář se nazývá top-down a funguje tak, že nejdříve naimplementujeme klasické Java třídy a ty pak rozšíříme o tzv. anotace nebo si patřičně nakonfigurujeme XML soubor s metadaty. Výhoda frameworku Hibernate je, že má nástroj hbm2ddl, který databázové schéma podle konfigurace vygeneruje za nás. Tento nástroj dále umí i databázové schéma automaticky aktualizovat po restartu aplikace [40].

Dále se budeme věnovat už samotnému mapování tříd při vývoji aplikací stylem top-down.

Ke správnému ukládání a načítání objektů z databáze potřebujeme třídy namapovat. K mapování lze použít dva způsoby, a to pomocí XML souboru nebo pomocí anotací. Nejprve je tedy nutné naimplementovat samotné modelové třídy. Třída by měla obsahovat id, tedy nějaký identifikátor. Mapování je proces, kterým říkáme Hibernate frameworku, jak se která entita bude ukládat a načítat [40].

Prvním způsobem, jak se dá třída namapovat, je pomocí XML souboru. V následujících dvou zdrojových kódech je ukázková Java třída a příslušný konfigurační soubor. Ukázková třída obsahuje atribut id, jméno a příjmení zaměstnance a jeho plat. (zdrojové kódy převzaty z [13])

```
public class Employee {
    private int id;
    private String firstName;
    private String lastName;
    private int salary;

    public Employee() {}
    public Employee(String fname, String lname, int salary) {
        this.firstName = fname;
        this.lastName = lname;
        this.salary = salary;
    }
    public int getId() {
        return id;
    }
    public void setId( int id ) {
        this.id = id;
    }
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName( String first_name ) {
        this.firstName = first_name;
    }
    public String getLastName() {
        return lastName;
    }
}
```

```

    }
    public void setLastName( String last_name ) {
        this.lastName = last_name;
    }
    public int getSalary() {
        return salary;
    }
    public void setSalary( int salary ) {
        this.salary = salary;
    }
}

```

XML soubor pro konfiguraci by pak vypadal následovně:

```

<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD//EN"
"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name="Employee" table="EMPLOYEE">
        <meta attribute="class-description">
            This class contains the employee detail.
        </meta>
        <id name="id" type="int" column="id">
            <generator class="native"/>
        </id>
        <property name="firstName" column="first_name" type="string"/>
        <property name="lastName" column="last_name" type="string"/>
        <property name="salary" column="salary" type="int"/>
    </class>
</hibernate-mapping>

```

V této konfiguraci říkáme frameworku Hibernate, že tabulka v databázi se bude jmenovat EMPLOYEE. Dále definujeme id spolu se strategií, jak se bude generovat a nakonec jednotlivé atributy, jejich typ a název korespondujícího sloupce v databázovém schématu. Tento soubor by měl být uložen pod názvem <název_třídy>.hbm.xml (zde Employee.hbm.xml) [13].

Druhým způsobem, jak namapovat třídu, je pomocí Java anotací. Tento způsob má tu výhodu, že nepotřebujeme žádný konfigurační XML soubor, ale upravují se o anotace příslušné persistentní Java třídy. V úvahu budeme brát stejnou třídu, jen tentokrát rozšířenou o anotace [10].

```

import javax.persistence.*;

@Entity
@Table(name = "EMPLOYEE")
public class Employee {
    @Id @GeneratedValue
    @Column(name = "id")
    private int id;
}

```

```

@Column(name = "first_name")
private String firstName;

@Column(name = "last_name")
private String lastName;

@Column(name = "salary")
private int salary;

public Employee() {}
public int getId() {
    return id;
}
public void setId( int id ) {
    this.id = id;
}
public String getFirstName() {
    return firstName;
}
public void setFirstName( String first_name ) {
    this.firstName = first_name;
}
public String getLastName() {
    return lastName;
}
public void setLastName( String last_name ) {
    this.lastName = last_name;
}
public int getSalary() {
    return salary;
}
public void setSalary( int salary ) {
    this.salary = salary;
}
}

```

Nejdůležitější anotací je zde anotace `@Entity`. Ta frameworku říká, že se jedná o persistentní entitu. Třída tedy musí splňovat podmínky pro persistentní entitu. Další anotací je `@Table`, která umí blíže specifikovat některé atributy tabulky, jako třeba její název. Následuje důležitá anotace `@Id`, která je povinná a určuje primární klíč v tabulce. Je často spojená s anotací `@GeneratedValue`, která určuje, jakou strategií se bude id generovat. Poslední anotací v této ukázce je `@Column`. Tato anotace frameworku říká, jak bude atribut vázán na konkrétní sloupec tabulky. Specifikuje tedy jméno sloupce, délku, jestli může obsahovat null hodnotu a vlastnost unique, určující, jestli je tato hodnota v rámci sloupce unikátní [10].

K pokročilejší konfiguraci se zaměříme na konfiguraci pomocí anotací.

V reálné aplikaci samozřejmě nemáme takto jednoduché třídy, a třídy často obsahují reference na jiné třídy. Může to být například reference na jednu třídu (kompozice) nebo na kolekci objektů daného typu. V prvním případě jde o asociaci 1:1 (one-to-one), ve druhém pak 1:M (one-to-many). Dále existují i anotace M:N (many-to-many) a M:1 (many-to-one). V Javě pro

ukládání kolekcí objektů slouží kromě klasických polí rozhraní typu Set, List, Collection a Map. Popisem toho, co jaké rozhraní umí, se zabývat nebudeme, ale ukážeme si, jak tyto kolekce namapovat pomocí anotací. JPA k mapování těchto asociací poskytuje tedy analogicky anotace @OneToOne, @OneToMany, @ManyToMany a @ManyToOne. Asociace dále mohou být buď jednosměrné nebo obousměrné a anotace se použijí přímo u těchto referencí nebo kolekcí objektů [5], [6].

Další důležitou vlastností je kaskádování. Ta nám říká, jak se bude chovat vytváření, ukládání, editace nebo mazání u tříd, které obsahují reference na jiné třídy. Kdybychom měli například třídu A, která by obsahovala referenci na třídu B, a obě takto provázané třídy bychom chtěli uložit, museli bychom pomocí setteru nastavit referenci B na A a následně obě třídy uložit metodou save. U tříd s více referencemi, popřípadě kolekcí objektů by zde bylo volání těchto save metod redundantní a zbytečně by rostla složitost kódu. Při nastavení kaskádování se tedy tyto operace zautomatizují a například při uložení třídy A se automaticky uloží i její reference [5].

Při konfiguraci pomocí anotací se kaskádové operace definují v atributu cascade jednotlivých anotací pro mapování asociací. Těchto vlastností může být víc, proto jsou definovány jako pole hodnot. Jednotlivé hodnoty atributu cascade, tak jak je definuje JPA, jsou v následující tabulce (Tabulka č. 2) [6]:

Tabulka 2 – Popis hodnot atributu cascade

Hodnota atributu cascade	Vysvětlivka
CascadeType.PERSIST	kaskáduje ukládání objektu
CascadeType.MERGE	kaskáduje editaci objektu
CascadeType.REMOVE	kaskáduje odstraňování objektu
CascadeType.REFRESH	kaskáduje aktualizaci objektu
CascadeType.DETACH	kaskáduje odpojení objektu
CascadeType.ALL	kaskáduje všechny předešlé operace

Dalším typem anotace použité v aplikaci je @Lob. Touto anotací se anotují velké objekty, jako například obrázky (byte[]). Posledním důležitým popisovaným typem je anotace @Inheritance, kterou se anotuje dědičnost. Hibernate poskytuje 3 strategie pro generování zděděných tříd do tabulek. Ty jsou InheritanceType.SINGLE_TABLE (všechny objekty uložené do jediné tabulky), InheritanceType.JOINED (tabulky vytvářené podle tříd a ty jsou

pak spojeny pomocí operace join) a InheritanceType.TABLE_PER_CLASS (samotná tabulka pro každou třídu) [5].

Nakonec se dostáváme ke strategii zavádění. Ta se nastavuje atributem fetch u anotace pro mapování asociace a může mít volby FetchType.EAGER a FetchType.LAZY. První volba načte kolekci objektů hned při dotazu na objekt rodiče. Druhá volba nenačítá všechny vztahy hned, ale načte je, až když chceme s touto asociací pracovat (na žádost – až když zavoláme příslušný getter na získání této například kolekce) [6].

```
@Entity
@Table
public class Book implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private int id;

    @Column(nullable = false)
    private String ISBN;

    @Column(nullable = false)
    private String author;

    @Column(nullable = false)
    private String name;

    @Column(nullable = false)
    private String publisher;

    @Temporal(javax.persistence.TemporalType.DATE)
    @Column(nullable = false)
    private Date publishedDate;

    @OneToMany(mappedBy = "book", fetch = FetchType.EAGER, cascade = CascadeType.ALL)
    private List<Print> prints;
```

Obrázek 15 – Představení anotací pro mapování třídy

Výše zmíněné teoretické poznatky si lze představit na předchozím obrázku (Obrázek č. 15). Zde je třída Book, která mimo jiné atributy obsahuje i seznam svých výtisků. U tohoto seznamu si můžeme všimnout anotace @OneToMany, která značí, že pro jednu knížku existuje více jejích výtisků. Atributy této anotace jsou mappedBy, určující, jakým atributem třídy Print je třída provázána se třídou Book. Dále pak strategie zavádění EAGER a kaskádování pro všechny operace (CascadeType.ALL) [5].

4.4 Konfigurace Hibernate projektu

Konfigurace celého projektu stojí na příslušném XML souboru, ze kterého je pak pomocí Javy tato konfigurace nahrána. Příklad takového XML souboru je v následujícím zdrojovém kódu. (zdrojový kód převzat z [12])

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration SYSTEM
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <property name="hibernate.dialect">
      org.hibernate.dialect.MySQLDialect
    </property>
    <property name="hibernate.connection.driver_class">
      com.mysql.jdbc.Driver
    </property>
    <!-- Assume test is the database name -->
    <property name="hibernate.connection.url">
      jdbc:mysql://localhost/test
    </property>
    <property name="hibernate.connection.username">
      root
    </property>
    <property name="hibernate.connection.password">
      root123
    </property>
    <!-- List of XML mapping files -->
    <mapping resource="Employee.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

Konfigurační soubor obsahuje mimo jiné důležité vlastnosti (zmíněné v tabulce č. 3) atribut mapping, který popisuje soubor, ve kterém leží informace o mapování dané třídy (zde Employee) nebo persistentní třídě s anotacemi [12].

Tabulka 3 – Seznam vlastností souboru hibernate.cfg.xml [12]

vlastnost	vysvětlivka
hibernate.dialect	určuje příslušný dialekt SQL jazyka (jako například MySQLDialect, OracleDialect, PostgreSQLDialect, a další)
hibernate.connection.driver_class	třída JDBC řadiče
hibernate.connection.url	URL adresa, na které databáze běží
hibernate.connection.username	přihlašovací jméno do databáze
hibernate.connection.password	přihlašovací heslo do databáze

Když se nám podaří tento soubor nakonfigurovat, můžeme začít pracovat s databází. Po inicializaci a získání objektu třídy SessionFactory z tohoto konfiguračního souboru pak můžeme pomocí metody openSession() získat objekt třídy Session a připojit se tak do databáze [5].

4.5 HQL, Query a Criteria API

Občas potřebujeme získat i jiné informace než jen objekt určitého typu daný podle zadaného id. Máme potřebu filtrovat informace podle zadaných kritérií nebo napsat dotaz ručně sami. Pro ruční psaní dotazů obsahuje Hibernate svůj vlastní jazyk HQL. Abychom mohli tento dotaz psát, je nutné vytvořit si objekt třídy Session. Ten pak obsahuje metodu createQuery, které pošleme HQL řetězec s naším dotazem (třída poskytuje samozřejmě i metodu createSQLQuery pro klasický SQL dotaz, ale touto volbou se dále zabývat nebudeme). Metoda createQuery vrací objekt třídy Query, který pak obsahuje metody pro vrácení seznamu objektů podle zadaného řetězce, vrácení unikátního výsledku, metodu setParameter pro vázání atributů do dotazu (pro předcházení SQL injection) nebo například nastavení hranic stránkování výsledků [5].

Příklad HQL dotazu pro přihlášení uživatele je na následujícím obrázku (Obrázek č. 16). Dotaz vybírá klienta podle emailu a hesla a v případě shody vrací unikátní výsledek.

```
Query query = session.createQuery("from Client where email = :email"
    + " and password = :password");

query.setParameter("email", email);
query.setParameter("password", password);

Client client = (Client) query.uniqueResult();
```

Obrázek 16 – Příklad HQL dotazu

Pro ty z nás, kteří nemáme rádi ruční psaní dotazů nebo chceme jinou alternativu, poskytuje Hibernate Criteria API. To poskytuje dotazování na základě kritérií (query by criteria) a na základě daného příkladu (query by example) [5].

Vše stojí na třídě Criteria a abychom mohli s touto třídou pracovat, je potřeba získat její instanci. Tu získáme ze třídy Session pomocí metody createCriteria, která přijímá jako parametr třídu, na kterou budou kritéria aplikovaná. Po vytvoření této instance už pak můžeme pomocí metody add jednotlivá kritéria přidávat. Jako vstup této metody jsou například výstupy statických metod třídy Restrictions, která poskytuje operace na porovnání

objektů, vlastností, testování, jestli je objekt null, větší než, menší než, logické operace a mnohé další. Další metodou třídy Criteria je addOrder, ve které určujeme, podle čeho se bude výstup řadit. Nakonec si výsledek necháme vrátit metodou list jako seznam objektů odpovídajících daných kritériím [5].

Příklad dotazu vytvořeného pomocí Criteria API je na následujícím obrázku (Obrázek č. 17). Zobrazena je zde metoda findBook, která hledá knížky na základě jména, autora nebo kategorie. Pomocí metody like třídy Restrictions se vybírají dané atributy podle zadané hodnoty, a to kdekoliv v řetězci (MatchMode.ANYWHERE). Výstupy těchto metod přijímá jako parametr metoda add třídy Disjunction, a to pro sjednocení všech výsledků. Následně přidáme tuto instanci pomocí add metody do objektu typu Criteria a nakonec si pomocí metody list necháme vrátit seznam výsledků, které těmto kritériím odpovídají.

```
@Override
public List<Book> findBook(String name, String author, String bookCategory) {
    Session session = sessionFactory.openSession();
    Criteria criteria = session.createCriteria(Book.class);
    criteria.createAlias("bookCategory", "b");
    Disjunction or = Restrictions.disjunction();

    if (name != null && !name.isEmpty()) {
        or.add(Restrictions.like("name", name, MatchMode.ANYWHERE));
    }
    if (author != null && !author.isEmpty()) {
        or.add(Restrictions.like("author", author, MatchMode.ANYWHERE));
    }
    if (bookCategory != null && !bookCategory.isEmpty()) {
        or.add(Restrictions.like("b.categoryName", bookCategory, MatchMode.ANYWHERE));
    }

    criteria.add(or);

    return criteria.list();
}
```

Obrázek 17 – Dotaz pomocí Criteria API

Technika dotazování je obsáhlé téma, proto zde byly zmíněny jen základy využití v praktické části práce. Pro psaní složitějších dotazů lze odkázat čtenáře například na [5].

5 Webové služby

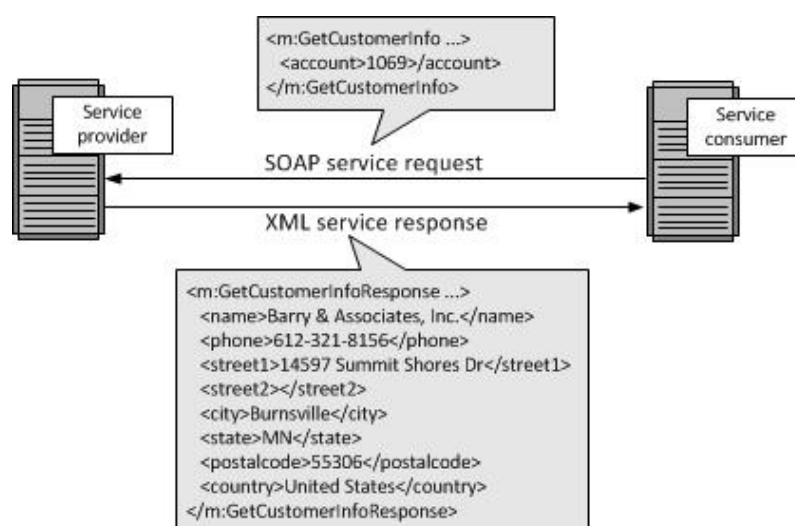
Dalším standardem, který se v dnešní době často používá, jsou webové služby (web services). Pomocí webových služeb lze odstínit jejich konzumenta od konkrétní platformy a zajistit integraci aplikací. Slovo webové je zde na svém místě, protože k výměně informací se používá protokol HTTP. Odstínění od konkrétního jazyka, ve kterém jsou webové služby napsány, se zajišťuje pomocí některého z jazyků pro výměnu dat mezi aplikacemi. Pro komunikaci s webovými službami se nejčastěji serializuje a deserializuje do XML nebo jazyka JSON. Komunikující strany jsou tedy dvě, a to producent webové služby a její konzument (klient) [4], [38].

Dnes existují dva způsoby popisu webových služeb, a to pomocí SOAP a REST, kterým se budeme věnovat v následujících podkapitolách [4].

5.1 SOAP

Když se rozhodneme používat SOAP, musíme si něco říct o architektuře, jak webové služby na tomto protokolu vytvářet. SOAP je protokol, pomocí kterého s rozhraním webové služby komunikujeme. Abychom mohli toto rozhraní naprogramovat, potřebuje k tomu samozřejmě nějaký jazyk. Pro tento účel je zvolen jazyk WSDL. Poslední částí této architektury je UDDI, který nám umožňuje registrovat a vyhledávat dané služby. Dále je potřeba zmínit, že odpověď od producenta webové služby dostáváme ve formátu XML [4].

Příklad komunikace pomocí protokolu je na následujícím obrázku (Obrázek č. 18).



Obrázek 18 – Příklad SOAP komunikace [4]

Jelikož je cílem práce napsat knihovni informační systém pomocí REST rozhraní, nebudeme se dále SOAP protokolem zabývat.

5.2 REST

Dalším způsobem popisu webové služby a alternativa k SOAP je REST. Principy protokolu REST byly prvně zmíněny v části disertační práce Roy Fieldinga v roce 2000. Zkratka znamená Representational State Transfer, a co jednotlivé části zkratky značí, si vysvětlíme v dalším textu. Representational znamená, že zdroje mohou být reprezentovány v jakékoliv formě (XML, JSON nebo například HTML). State – při práci s REST nás více zajímá stav zdroje, než akce, které proti zdroji můžeme vykonat. Posledním slovem zkratky je Transfer a znamená, že REST zahrnuje přenos zpráv v některé reprezentační formě z jedné aplikace do druhé. Jednou větou lze princip REST shrnout jako přenos stavu zdrojů v jisté reprezentační formě mezi aplikacemi [35], [40].

Jak již bylo řečeno, pomocí REST rozhraní přistupujeme ke zdrojům. Tyto zdroje jsou definovány URL adresou. K této adrese se pak přistupuje pomocí protokolu HTTP a jeho metod. Se zdroji chceme většinou nějak pracovat a to, co se se zdroji dá vykonat, lze obecně shrnout jako CRUD operace. Mechanismus přístupu k těmto zdrojům a manipulaci s těmito zdroji by se dal tedy shrnout do operací vytvoření zdroje, získání zdroje, editace zdroje a jeho smazání. Výčet CRUD operací a jejich mapování na HTTP metody je zobrazen v následující tabulce (Tabulka č. 4) [35], [40].

Tabulka 4 – CRUD operace a jejich HTTP ekvivalenty

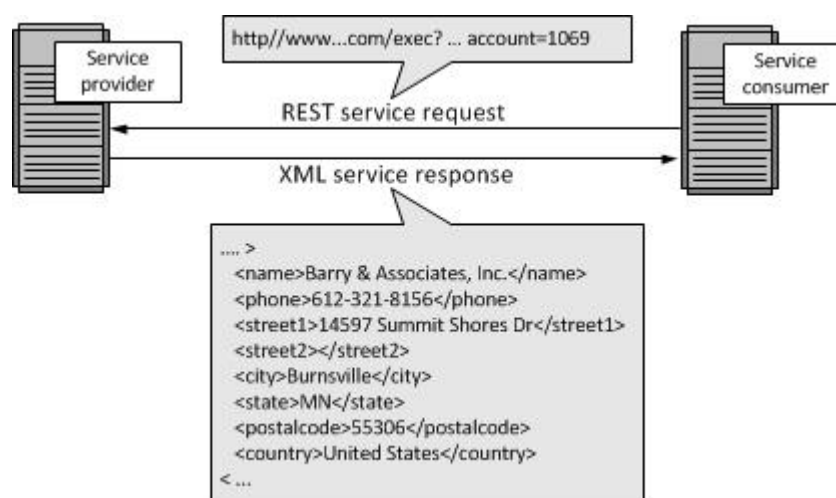
Akce se zdrojem	HTTP metoda
Create	POST
Read	GET
Update	PUT nebo PATCH
Delete	DELETE

Tyto operace nemusí být pevně svázány s těmito HTTP ekvivalenty. Může například nastat případ, že při vytvoření zdroje můžeme použít PUT metodu. Není tedy striktně vyžadováno dodržovat údaje uvedené v tabulce [40].

Abychom mohli považovat systém za RESTful systém, musí splňovat následující podmínky [4], [35]:

- musí to být klient-server systém,
- musí být tzv. stateless, neboli nesmí obsahovat žádná sezení s klienty, každý požadavek musí být nezávislý na ostatních,
- musí podporovat caching mechanismus, síťová architektura by měla podporovat cache na více úrovních,
- zdroje musí být jednotně přístupné – dané svou jedinečnou adresou,
- musí být rozdělen do vrstev, a to z důvodu škálovatelnosti,
- měla by umožňovat poskytnout kód na vyžádání – například Java Applety.

Příklad komunikace klienta s poskytovatelem služby si lze představit na obrázku č. 19. Konzument REST služby vyvolá požadavek na danou URL adresu a producent mu pak vrací výsledek ve formě XML.



Obrázek 19 – Komunikace klienta s REST API [3]

5.3 REST API pomocí Spring MVC

Výhody Spring frameworku asi již není nutné zmiňovat. Klasická Java EE umožňuje vytvářet jak webové služby, tak psát kód pro klienta pro konzumaci těchto webových služeb. Samozřejmě i Spring není s touto funkcionalitou pozadu a modul MVC nám umožňuje napsat si vlastní webové služby. Tato kapitola se bude tedy věnovat tomu, jak si lze ve Springu naprogramovat vlastní REST API [40].

Jestliže pracujeme s frameworkem Spring, a chceme si vytvořit REST controller, stačí vytvořit klasický controller s anotací `@Controller` s několika změnami oproti klasickému controlleru ve webové aplikaci. Hlavní změna nastává u anotací jednotlivých metod. Zůstává

zde anotace `@RequestMapping`, určující relativní URL adresu zdroje. Rozdílem oproti klasickému Spring MVC controlleru jsou anotace `@ResponseBody` a `@RequestBody`. Anotací `@ResponseBody` se anotuje tělo metody a říká Springu, že objekt, který metoda vrací, má být vrácen v dané reprezentační formě, tedy ne jako klasický Java objekt nebo název view. Pomocí anotace `@ResponseBody` se tedy navracený objekt z metody zkonvertuje například do JSON nebo XML a odešle jako HTTP odpověď klientovi. Anotace `@RequestBody` tedy analogicky pracuje opačně a přijímaný JSON nebo XML řetězec automaticky zkonvertuje do daného Java objektu, který je vstupem do metody. Příklad praktické ukázky těchto dvou anotací je na následujícím obrázku (Obrázek č. 20) [1], [40].

```
@Controller
@RequestMapping("book")
public class BookRestService {

    @Autowired
    private IBooksService bookService;
    @Autowired
    private IEmployeesService employeeService;

    @RequestMapping(value = "editBook")
    @ResponseBody
    public Book editBook(
        @RequestParam(value = "signToken", required = true) String signToken,
        @RequestBody(required = true) Book book) throws NotValidSignToken {
        if (!employeeService.validateSignToken(signToken)) {
            throw new NotValidSignToken(signToken);
        }

        return bookService.editBook(book);
    }
}
```

Obrázek 20 – Příklad anotací `@RequestBody` a `@ResponseBody`

Na daném příkladu je tedy vstupem do metody povinný parametr přihlašovací token a dále je vstupem daný objekt `Book`, který představuje editovanou knihu. Výstup této metody je pak objekt `Book` rozšířený o `@ResponseBody`, kterým se vrací nově editovaná kniha.

Pro vysvětlení toho, jak se implementuje REST producent, není potřeba žádného dlouhého vysvětlování. Při testování těchto metod můžeme například při HTTP metodě GET nebo POST užít některého z webových prohlížečů. Nicméně většinou potřebujeme s REST API komunikovat pomocí aplikace. Java ve své Enterprise edici samozřejmě poskytuje nástroje pro vyvolání příslušného požadavku, zde se ale zaměříme na to, jak tuto funkcionalitu implementuje Spring [1].

V následujícím textu si řekneme, jak postupovat při psaní REST klienta. Spring pro tento účel nabízí svoji třídu `RestTemplate`. Ta, oproti kódu napsaném v klasické Javě EE, poskytuje metody, pomocí kterých jediným voláním vyvoláme nějaký požadavek, a popřípadě se nám ihned vrátí data. Při použití Springu jsme tak ušetřeni zbytečného kódu navíc. Třída `RestTemplate` definuje hned několik metod pro práci s REST zdroji. Některé důležité z nich shrnuje následující tabulka (Tabulka č. 5) [40].

Tabulka 5 – Popis vybraných metod třídy `RestTemplate` [7], [40]

Název metody	HTTP ekvivalent	Popis
<code>delete()</code>	DELETE	provede HTTP DELETE požadavek
<code>getForEntity()</code>	GET	pošle HTTP GET požadavek a vrátí objekt <code>ResponseEntity</code> obsahující objekt zkonvertovaný z těla odpovědi
<code>getForObject()</code>	GET	pošle HTTP GET požadavek a vrátí objekt zkonvertovaný z těla odpovědi
<code>headForHeaders()</code>	HEAD	pošle HTTP HEAD požadavek a vrátí HTTP hlavičky pro specifickou URL adresu
<code>optionsForAllow()</code>	OPTIONS	pošle HTTP OPTIONS požadavek a vrátí <code>Allow</code> hlavičku pro specifickou URL adresu
<code>postForEntity()</code>	POST	pošle HTTP POST požadavek a vrátí objekt <code>ResponseEntity</code> obsahující objekt zkonvertovaný z těla odpovědi
<code>postForLocation()</code>	POST	pošle HTTP POST požadavek a vrátí URL nově vytvořeného zdroje
<code>postForObject()</code>	POST	pošle HTTP POST požadavek a vrátí objekt zkonvertovaný z těla odpovědi
<code>put()</code>	PUT	vloží zdroj (data) na specifickou URL adresu
<code>exchange()</code>	jakýkoliv	provede jakýkoliv požadavek definovaný v metodě a vrátí objekt typu <code>ResponseEntity</code>
<code>execute()</code>	jakýkoliv	provede jakýkoliv požadavek definovaný v metodě a vrátí objekt zkonvertovaný z těla odpovědi

Na zdrojovém kódu následujícího obrázku (Obrázek č. 21) si vysvětlíme příklad volání `postForObject`. Je to část kódu spolupracující s metodou `editBook`, která je ilustrovaná výše. Metoda `postForObject` má několik přetížených variant, nicméně zvolena zde byla metoda, která přijímá URL adresu, dále objekt požadavku a třídu návratového typu (pro deserializaci). V následujícím příkladu si tedy zjistíme přihlašovací token zaměstnance a dále na již vytvořeném objektu typu `RestTemplate` je volána metoda `postForObject`. Ta přijímá jako parametr tedy URL adresu, na které REST API běží spolu s názvem metody (`editBook`) a parametrem `signToken` (přihlašovací token). Dalším parametrem je odesílaný objekt a posledním parametrem je typ objektu, který API vrací.

```
String signToken = employee.getSignToken();
Book updated = template.postForObject(LIBRARY_SYSTEM_BE_ENDPOINT
    + "book/editBook?signToken=" + signToken, book, Book.class);
```

Obrázek 21 – Příklad volání konzumenta REST služeb

To ale není vše, co pro správné fungování jak konzumenta, tak producenta, musíme udělat. Dále musíme do projektu připojit message convertor. Ten se stará o konverzi dat z a do příslušné reprezentační formy. `DispatcherServlet` pak s příchozím požadavkem zjistí jeho `Accept` header, zvolí daný message convertor a vrací pak odpověď v reprezentaci, jakou si klient zvolil. Spring podporuje více druhů těchto konvertorů jako např. `Jaxb2RootElementHttpMessageConverter` (XML reprezentace), `MappingJackson2HttpMessageConverter` (JSON reprezentace) a jiné [40].

V aplikaci se pak dále vyskytuje i anotace `@JsonIgnore`. Ta by mohla být matoucí, netýká se totiž Springu ani Hibernate, ale právě konverze dat do reprezentační formy. Tato anotace říká, že daná proměnná nemá být serializována. U XML reprezentace se pak vlastnosti, které nechceme serializovat, anotují `@XmlTransient` [1].

Další anotací může být anotace `@ResponseStatus`, aplikovaná na metodu nebo třídu výjimky. Ta vrací spolu s odpovědí také informaci o HTTP statusu, který byl ze serveru vrácen. Anotace přijímá jako parametr konstantu třídy `HttpStatus`. Konstanty představují dané stavové kódy, které může server vracet (např. `HttpStatus.OK`, `HttpStatus.UNAUTHORIZED` a další) [40].

6 Implementace knihovního informačního systému

V teoretické části práce byly předvedeny frameworky, ve kterých bude napsán knihovní informační systém. Následující podkapitoly se tedy budou věnovat samotnému popisu funkčnosti aplikace, dále jejímu návrhu a analýze, implementaci a zvoleným nástrojům, díky kterým mohla být tato aplikace vytvořena.

6.1 Popis praktické části

Jako téma pro představení nástrojů popsanych v teoretické části byl zvolen knihovní informační systém. Nebude to tak komplexní řešení, jako již existující knihovní IS, ale bude se snažit zachytit základní knihovní procesy. V tomto systému budou vystupovat tři role, a to klient, zaměstnanec (pracovník knihovny) a administrátor. Požadavek na tento systém byl takový, aby mohl spravovat všechny její pobočky. Bude se tedy týkat knihovny například v nějakém větším městě.

Jelikož jsou knihovní IS často rozděleny na aplikaci pro zaměstnance a katalog s uživatelským rozhraním, byl požadavek aplikaci sjednotit do jedné webové aplikace. Knihovní procesy pak budou řešeny přes jednotné webové rozhraní.

Klienti si budou moci prohlížet jednotlivé tituly a popřípadě si zarezervovat výtisk dané knihy. Zaměstnanci mají logicky práva větší, mohou tedy spravovat, přidávat a mazat knihy z knihovního fondu a jejich výtisky. Registrace klientů je povolena pouze roli zaměstnanec, protože samotné fyzické registraci předchází většinou nějaký poplatek (klienti se registrují na pobočkách). Dále mohou zaměstnanci spravovat klientské účty, tedy klienty registrovat, mazat, editovat a vytvářet těmto klientům rezervace a výpůjčky. Největší práva má administrátor, ten může takto spravovat pobočky i zaměstnance.

V systému musí být logicky evidovány výpůjčky a rezervace. Výpůjčku bude vytvářet zaměstnanec ve webovém rozhraní. Počet výpůjček na jednoho uživatele bude omezen na pět. Vypůjčený výtisk si pak bude moci rezervovat kterýkoliv jiný klient. Samozřejmě bude možné rezervovat i nevypůjčený výtisk. Maximální počet rezervací je také pět, poté bude uživatel v případě potencionálního překročení informován a rezervace nebude povolena. Rezervační doba je dva týdny, poté rezervace expiruje. Rezervován nebo vypůjčen může být jen jeden výtisk dané knížky. Výpůjční doba je jeden měsíc, ale klient si bude moci ve webovém rozhraní tuto dobu prodloužit, a to maximálně o další jeden měsíc. Podmínkou pro prodloužení výpůjčky je, že výtisk již nesmí být zarezervován jiným klientem. Proces prodloužení lze využít maximálně jednou, pak už musí knihu vrátit. Aplikace musí uživatele

upozornit v případě expirace výpůjčky. Klient si bude moci prohlížet i historii svých výpůjček a rezervací.

Role pobočky v systému je taková, že uchovává informace o svých zaměstnancích a výtiscích knih, které se na ní nachází. To znamená, že každá kniha může mít více výtisků a každý může být na jiné pobočce. Úkolem systému bude také vypsát u každé knihy seznam jejích výtisků, spolu s tím, kde jsou umístěny.

Na úvodní stránce knihovního systému budou novinky (deset nově přidaných knížek) a základní informace o knihovně. Webová aplikace bude také obsahovat jednoduchý katalog knih včetně filtru (filtrování podle názvu, autora a kategorie).

V části s rozšířenými právy (zaměstnanec, administrátor) bude možné i přerazovat výtisky na jinou pobočku.

Tato zadávací část nebyla vytvořena žádným externím subjektem, tedy neexistuje žádný oficiální zadávací dokument. Požadavky byly vytvářeny na základě konzultace s vedoucím práce a na základě zjištěných informací o reálných knihovních procesech.

6.2 Použité nástroje pro návrh a implementaci

Následující podkapitoly představí nástroje a platformy, na základě kterých je aplikace vyvíjena.

6.2.1 Enterprise Architect

Pro zpracování a hlavně dokumentaci požadavků, analýzy a návrhu tříd byl použit nástroj Enterprise Architect, a to konkrétně ve verzi 11. Tento program nabízí zachycení této analýzy pomocí modelovacího jazyka UML. Dále nabízí i modelování business procesů pomocí BPMN, modelování podnikové architektury pomocí jazyka Archimate a další. Výhodou nástroje je, že podporuje generování zdrojových kódů z návrhových tříd. Dále také podporuje reverse engineering, tedy naopak generování UML diagramů ze zdrojových kódů [39].

Jako modelovací jazyk byl zvolen jazyk UML, který je v dnešní době považován za standard.

6.2.2 Java, Spring a Hibernate Framework

Aplikace je vyvíjena v jazyce Java, a to ve verzi 7. Implementovaný IS běží na frameworku Spring verze 4 a frameworku Hibernate v téže verzi.

6.2.3 MySQL databáze

Pro persistenci dat byla zvolena databáze MySQL 5.6.19. Hlavním důvodem volby právě této databáze bylo, že lze využívat její bezplatné licence. Další důvod byl ten, že je nejrozšířenější na poli databázových systémů, které jsou zdarma a že pro účely této aplikace není potřeba žádného robustního databázového systému jako např. Oracle [29].

6.2.4 Apache Tomcat

Aby bylo možné aplikaci zprovoznit, je potřeba běhového prostředí pro Javu. To poskytují webové servery. Těch je v dnešní době více, nicméně zvolen byl Apache Tomcat ve verzi 8.0.9. Ten je již po instalaci integrován do IDE Netbeans, není třeba ho tedy ručně doinstalovávat.

6.2.5 Maven

Maven je nástroj pro správu a sestavování aplikací. Umožňuje například celou robustní aplikaci rozdělit do několika celků (modulů), z nichž se celý projekt skládá. Nicméně jednou z hlavních vlastností je to, že nám dokáže stáhnout veškeré závislosti na knihovnách třetích stran (zde v aplikaci například Spring a Hibernate frameworku). Jediná věc, kterou pak musíme udělat, je vytvořit tzv. POM (Project Object Model). Ten je uložen v souboru pom.xml a obsahuje soubor knihoven, na kterých je projekt závislý. Činnost Mavenu je pak taková, že si přečte tento soubor a veškeré definované knihovny stáhne přímo do projektu. Navíc pro práci s Mavenem nemusíme užívat příkazové řádky, ale je již dnes plně integrován do každého IDE. To pak práci s Mavenem automatizuje [27].

Pro Maven je typická struktura projektu shrnutá v tabulce č. 6.

Tabulka 6 – Adresářová struktura Mavenu [18], [27]

Adresář	Popis
/src/main/java	zdrojové kódy, .java soubory
/src/main/resources	zdroje, další soubory
/src/test/java	testové třídy
/src/test/resources	testové zdroje
/src/main/webapp	zdroje webové aplikace
/target	distribuovatelné JAR soubory
/target/classes	zkompilovaný byte kód

Jak již bylo zmíněno, Maven pracuje na základě definování POM. Příklad toho, jak se v souboru pom.xml definují závislosti, je na následujícím obrázku (Obrázek č. 22).

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
    <version>4.1.5.RELEASE</version>
    <type>jar</type>
  </dependency>
  <dependency>
    <groupId>com.librarysystem</groupId>
    <artifactId>librarySystem-model</artifactId>
    <version>1.0</version>
    <type>jar</type>
  </dependency>
</dependencies>
```

Obrázek 22 – Definice závislostí v souboru pom.xml

Další obrázek (Obrázek č. 23) pak obsahuje definici modulů, ze kterých se celý projekt skládá.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.librarysystem</groupId>
  <artifactId>librarySystem</artifactId>
  <version>1.0</version>
  <packaging>pom</packaging>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>
  <modules>
    <module>librarySystem-model</module>
    <module>librarySystem-web</module>
    <module>librarySystem-be</module>
  </modules>
</project>
```

Obrázek 23 – Definice modulů v souboru pom.xml

6.2.6 Netbeans IDE

Jako vývojové prostředí byl zvolen Netbeans 8.0.2. Vývojové prostředí je zdarma a jedná se o produkt Oracle Corporation. IDE podporuje hlavně jazyk Java, nicméně pomocí přídatných modulů lze psát i jazycích jako PHP, C, C++, Groovy, HTML, CSS a JavaScript [30].

IDE nabízí optimální rozložení ovládacích prvků, debugger, syntaktickou analýzu kódu a nápovědu. Dále lze spravovat webové servery a připojit se do databáze. Pro tu má i grafické rozhraní, kde lze data spravovat a editovat [30].

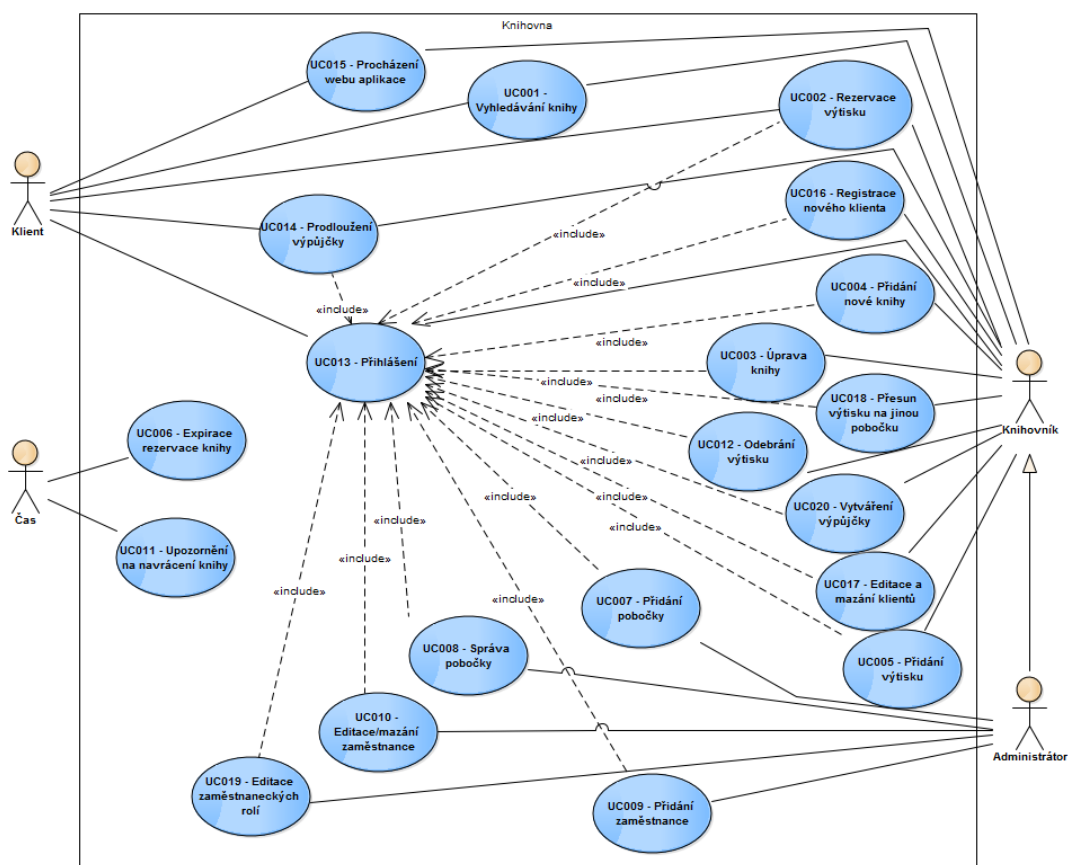
6.2.7 Technologie použité na webové vrstvě

Ve frontend části aplikace, tedy části webové, je použita technologie JSP v kombinaci s klasickým HTML. O stylování se stará CSS framework Bootstrap a o dynamické části webu JavaScript a jQuery. O zpracování požadavků se samozřejmě starají controllery Spring MVC modulu.

6.3 Analýza a návrh

Každý vývoj aplikace by měl začít analýzou a návrhem, který nám poskytne základní představu o tom, z jakých komponent se bude daná aplikace skládat a jaké na ni budou požadavky. V této podkapitole budou tedy zobrazeny jednotlivé případy užití a následně rozdělení aplikace do logických celků a tříd, se kterými se bude dále pracovat.

Souhrn rolí a toho, co jaká role může v systému využívat je zobrazeno na následujícím obrázku (Obrázek č. 24).



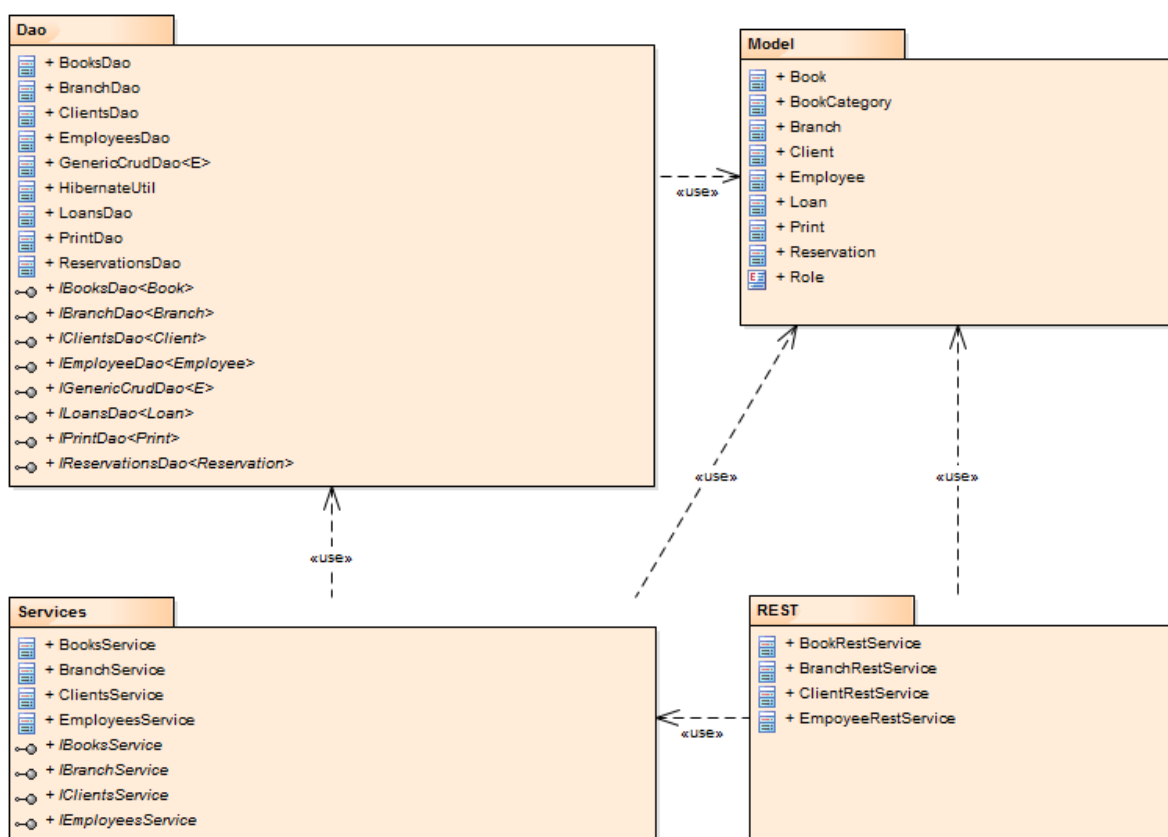
Obrázek 24 – Use case diagram knihovního IS

Jak je patrné, v aplikaci přibyla další role, a to čas. Čas v aplikaci hlídá datum rezervace knihy, a tedy to, jestli rezervace neexpirovala. Dále klienta upozorňuje na navrácení knihy v případě nevrácení do stanoveného termínu.

Jediná role klienta může v aplikaci vystupovat i bez přihlášení. V tomto případě lze o této roli hovořit jako o klasickém uživateli, který má ale jen omezená práva. Práva jsou tedy procházení dostupného obsahu webu, a to vyhledávání knížek, zobrazení detailu knížek, jejich stavů a zobrazování informací o knihovně.

Přihlašování do aplikace probíhá na základě emailu a hesla. Pro zaměstnance je logika aplikace postavena tak, že se přihlašují svým emailem končícím „@knihovna.cz“. Tím je rozpoznáno, že se do aplikace přihlašuje zaměstnanec. Ostatní emaily jsou brány jako emaily klientů.

Na dalším obrázku (Obrázek č. 25) je zobrazeno rozdělení tříd do logických balíčků. Část aplikace zobrazená na obrázku se týká backendu, tedy části spolupracující s databází a poskytující REST API. Je zde tedy balíček Model, sdružující modelové třídy a dále balíček Dao, který má za úkol persistenci dat. Dalším balíčkem je Service, jedná se o servisní vrstvu, která implementuje business logiku aplikace a spolupracuje s Dao balíčkem. Posledním balíčkem je REST. Jedná se o samotné REST API a spolupracuje se servisní vrstvou.



Obrázek 25 – Rozdělení tříd do balíčků

UML diagram persistentních tříd není v této dokumentační části zahrnut, ale je přiložen na CD v Enterprise Architektu v modelu tříd. Diagram databázového schématu zde také není shrnut, protože ho odráží tento objektový model (Hibernate si automaticky vytvoří schéma na základě těchto tříd). Popis jednotlivých tříd a výtčů shrnuje následující seznam:

- Book – tato třída má za úkol shrnovat informace o samotné knize, tedy její ISBN kód, název, jméno autora, nakladatelství, počet stran, číslo edice, datum publikování, datum přidání do systému, kategorii knihy, její obrázky a nakonec množinu výtisků této knihy.
- BookCategory – třída pro uchovávání kategorie knihy. Požadavek na správu kategorií není, takže číselník kategorií je v databázi uložen napevno. Číselník odborné literatury se pak drží MDT.
- Branch – třída pro jednotlivou pobočku dané knihovny. Třída obsahuje následující atributy: název pobočky, město, kde se nachází, ulice, email, telefonní číslo, fotku pobočky, množinu zaměstnanců a množinu výtisků, které se na pobočce nachází.
- Client – uchovává informace o klientovi. Persistentní atributy jsou následující: jméno a příjmení klienta, město, ulice, datum narození, jeho přihlašovací email, heslo (uložené v hashi) a přihlašovací token. Dále má tato třída odkaz na množinu rezervací a výpůjček klienta.
- Employee – třída logicky uchovává informace o zaměstnanci. Jde tedy o jméno, příjmení, přihlašovací email, heslo (v hashi), jeho fotografie, role zaměstnance (jestli jde o administrátora nebo klasického zaměstnance) a jeho přihlašovací token. Další atributy jsou množina výpůjček, které zaměstnanec zpracoval a odkaz na pobočku, na které pracuje.
- Loan – tato třída dědí od třídy Reservation a má za úkol ukládat jednotlivé výpůjčky. Navíc obsahuje informaci o zaměstnanci, který výpůjčku vytvořil, datum vrácení knihy a to, jestli je možné výpůjčku prodloužit.
- Print – uchovává informace o id výtisku, má odkaz na titul, kterým je výtiskem. Další atributy jsou: pobočka, na které se výtisk nachází, klient, který má daný výtisk půjčený nebo rezervovaný a odkaz na množinu rezervací, kterou je nebo byl součástí.
- Reservation – třída Reservation, jak název napovídá, má za odpovědnost ukládat jednotlivé rezervace. Atributy rezervace jsou datum jejího vytvoření a expirace, výtisk, kterého se rezervace/výpůjčka týká, klient, který rezervaci/výpůjčku vytvořil a příznak valid, který určuje, zda je rezervace/výpůjčka validní. Valid atribut je zde tedy z důvodu historie, tedy jen ty, s příznakem valid nastaveným na true jsou výpůjčkami/rezervacemi aktuálními. Ostatní jsou brány jako výpůjčky/rezervace vytvořené v minulosti.
- Role – tento výčet určuje roli zaměstnance v systému. Možné hodnoty výčtu jsou CASUAL (běžný zaměstnanec) a ADMINISTRATOR (administrátor s rozšířenými právy).

Některé atributy těchto tříd by se mohly zdát redundantní, jsou zde ale z důvodu Hibernate mapování (obousměrných asociací).

Třídy, starající se o samotnou práci s databází, jsou programovány pomocí návrhového vzoru DAO. Funkčnost těchto tříd je shrnuta v následující kapitole.

Dále je zde speciální balíček Exceptions, kde jsou uživatelsky definované výjimky. Název výjimky spolu se stavem, kdy je výjimka vyvolána shrnuje následující tabulka (Tabulka č. 7).

Tabulka 7 – Uživatelsky definované výjimky

Název výjimky	Důvod vyvolání
CannotCreateReservationException	Nelze vytvořit rezervaci z důvodu překročení počtu rezervovaných výtisků.
CannotCreateLoanException	Nelze vytvořit výpůjčku z důvodu překročení počtu vypůjčených výtisků.
InsufficientRightsException	Výjimka nastává v případě, že by zaměstnanec volal metodu, na kterou má právo pouze administrátor.
NotValidSignToken	Vzniká při nevalidním přihlašovacím tokenu, tedy při neoprávněném přístupu k API.
UserAlreadyExistsException	Nastává v případě, když chceme přidat uživatele, který v databázi již existuje.

Posledním balíčkem je balíček Rest, jehož třídy tvoří samotné REST rozhraní.

6.4 Implementace

V implementační části bylo postupováno tak, že byly z nástroje Enterprise Architect vygenerovány modelové třídy, a ty pak rozšířeny o příslušné anotace kvůli Hibernate persistenci. Aplikace tedy byla vyvíjena stylem top-down a databázové schéma bylo vygenerováno nástrojem Hibernate. Je potřeba zmínit, že vzhledem k dědičnosti tříd Reservation a Loan musela být zvolena příslušná strategie pro generování tabulek. Nakonec bylo rozhodnuto ve prospěch strategie table per class, tedy tabulka pro každou třídu zvlášť, a to z důvodu přehlednosti v databázi. Dále byly vygenerovány příslušné DAO a Service třídy pro každou doménu a patřičně doimplementována těla metod. DAO vrstva pracuje s generickou abstraktní třídou GenericCrudDao, která implementuje rozhraní IGenericCrudDao a má za odpovědnost CRUD operace. Implementuje tedy příslušné metody

pro mazání, editaci, přidání a získání všech objektů daného typu. Třída spolupracuje s třídou HibernateUtil, což je třída navrhnutá podle návrhového vzoru Singleton a vytváří objekt typu SessionFactory. Tento objekt pak vytváří jednotlivá sezení (Session) s databází. Při vytváření tohoto objektu je čtena konfigurace ze souboru hibernate.cfg.xml. Vytváření SessionFactory shrnuje obrázek č. 26.

```
package com.librarysystem.dao;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.AnnotationConfiguration;

/**
 * @author Richard
 * @version 1.0
 * @created 24-XI-2015 18:18:44
 */
public class HibernateUtil {

    private static final SessionFactory sessionFactory;

    static {
        try {
            // Create the SessionFactory from standard (hibernate.cfg.xml) Config file.
            sessionFactory = new AnnotationConfiguration()
                .configure().buildSessionFactory();
        } catch (Throwable ex) {
            // Log the exception.
            System.err.println("Initial SessionFactory creation failed." + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

Obrázek 26 – Třída HibernateUtil

Jak lze vidět podle komentáře v kódu, třída čte konfiguraci z XML souboru hibernate.cfg.xml. V tomto souboru jsou definovány základní údaje k přístupu k databázi, tedy dialekt SQL jazyka (zde MySQL dialekt), zaregistrování řadiče pro připojení do databáze, URL adresa, na které běží databázový server a přihlašovací údaje k databázi. Dalšími vlastnostmi jsou pak logování sql dotazů do server logu (show_sql), nastavení, co se má s databázovým schématem dít po jeho změně (vlastnost hbm2ddl.auto) a dále mapování jednotlivých persistentních tříd. Tento soubor je zobrazen na obrázku č. 27.

Třída GenericCrudDao poskytuje tedy obecné rozhraní pro práci s databází. Ostatní třídy, jako například BooksDao potom dědí od této generické třídy a specializovaná třída pak přebírá konkrétní typ, se kterým pracuje. Třída BooksDao má potom stejné metody, jen s konkrétním typem. Implementuje ale dále i rozhraní s metodami navíc, které jsou specifické pro danou doménu.

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
    <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/librarySystemDb</property>
    <property name="hibernate.connection.username">richard</property>
    <property name="hibernate.connection.password">password</property>

    <property name="show_sql">true</property>
    <property name="hbm2ddl.auto">update</property>

    <mapping class="com.librarysystem.model.Book" />
    <mapping class="com.librarysystem.model.BookCategory" />
    <mapping class="com.librarysystem.model.Branch" />
    <mapping class="com.librarysystem.model.Client" />
    <mapping class="com.librarysystem.model.Employee" />
    <mapping class="com.librarysystem.model.Loan" />
    <mapping class="com.librarysystem.model.Print" />
    <mapping class="com.librarysystem.model.Reservation" />
  </session-factory>
</hibernate-configuration>

```

Obrázek 27 – Konfigurační soubor Hibernate (hibernate.cfg.xml)

Dalším bodem byla implementace servisních tříd. Servisní třídy jsou ty, označené anotací `@Service`. Tyto třídy implementují business logiku aplikace a spolupracují s DAO třídami. Některé z metod těchto tříd jen převolávají metody DAO tříd, některé mají ale složitější logiku. Například při vytváření výpůjčky se kontroluje počet výpůjček daného klienta, nastavuje se datum expirace, atd. Expirace výpůjčky je pak v aplikaci hlídána aktuální datem, kde se v aplikaci zobrazí jak klientovi, tak zaměstnanci expirace dané výpůjčky. Kniha se musí při vypůjčení fyzicky vrátit, tzn. zaměstnanec pak výpůjčku v aplikaci ruší sám. Rezervace jsou ale implementované jinak, protože rezervace expiruje daným datem. Pro rušení rezervace je při jejím vytváření naplánována pomocí metody `schedule` třídy `Timer` úloha, která se spouští v době expirace rezervace a zařídí tak automaticky její zrušení.

Nakonec bylo naimplementováno samotné REST API, tedy třídy, které tvoří hranici backend aplikace a zpracovávají požadavky pomocí REST dotazů. Pro komunikaci s REST rozhraním byl zvolen jazyk JSON.

Aby byla aplikace zabezpečena, co se týče k přístupu ke zdrojům, je implementováno zabezpečení na dvou úrovních. První je zabezpečení na straně front-endu, tedy k přístupu ke stránkám. Na úrovni jednotlivých controllerů je kontrolováno, zda se jedná o přihlášeného klienta nebo zaměstnance a následně je pak rozhodnuto, jestli má tento přístup k danému zdroji. Dalším, a mnohem důležitějším zabezpečením, je autorizace k REST API. Je zde z toho důvodu, že REST API je vystaveno veřejně a každý, kdo by znal adresu a parametry například namapované metody pro mazání uživatelů, by mohl těchto údajů zneužít.

Zabezpečení je docíleno tak, že při úspěšném přihlášení uživatele (nezáleží, jestli klienta nebo zaměstnance) je vygenerován a uložen jeho přihlašovací token. Ten je pak uložen trvale v databázi do té doby, než se uživatel odhlásí. Ke generování tohoto tokenu je použita třída `java.util.UUID`, konkrétně pak metoda `randomUUID()`. Metody backend části projektu (REST API), pro jejichž přístup je potřeba se autorizovat, pak tento přihlašovací token ve svém těle přijímají a kontrolují, zda se jedná o přihlášeného uživatele. V případě, že by tomu tak nebylo, je generován HTTP status 401 (unauthorized).

Další úroveň zabezpečení je hashování hesel. Ta pak nejsou v databázi uložena jako čistý text, ale jako hash, z důvodu možnosti zneužití přihlašovacích údajů. Hashování je implementováno pomocí třídy `org.springframework.util.DigestUtils` a konkrétně její metody `md5DigestAsHex`.

Na obrázku č. 28 je vidět kód metody pro získání všech zaměstnanců, která zpracovává dotaz. Další obrázek (Obrázek č. 29) pak ilustruje volání REST dotazu (i s přihlašovacím tokenem) pomocí URL zadané do webového prohlížeče spolu s JSON výstupem.

```
@RequestMapping(value = "getEmployees")
@ResponseBody
public List<Employee> getAllEmployees(
    @RequestParam(value = "signToken", required = true) String signToken)
    throws NotValidSignToken, InsufficientRightsException {

    if (!employeeService.validateSignToken(signToken)) {
        LOGGER.warning("employees not obtained");
        throw new NotValidSignToken(signToken);
    }

    checkAdminRole(signToken);

    return employeeService.getAllEmployees();
}
```

Obrázek 28 – REST metoda pro získání všech zaměstnanců



```
[{"id":1,"name":"Pavla","surname":"Jaklová","email":"Pavla.Jaklova@knihovna.cz","password":"6572bdaff799084b973320f43f09b363","employeePhoto":null,"employeeRole":"CASUAL","branch":{"id":1,"name":"Pobočka Tabulový vrch","city":"Olomouc","street":"Karafiátova 10","branchPhoto":null,"email":"tabulovy.vrch@knihovna.cz","telephone_number":"585 555 555"},"signToken":"9e623436-6f39-407d-abf6-6bfc09cfc765"}, {"id":4,"name":"Karel","surname":"Hnát","email":"Karel.Hnat@knihovna.cz","password":"6572bdaff799084b973320f43f09b363","employeePhoto":null,"employeeRole":"CASUAL","branch":{"id":2,"name":"Pobočka Neředín","city":"Olomouc","street":"Tererovo náměstí 15","branchPhoto":null,"email":"neredin@knihovna.cz","telephone_number":"585 111 333"},"signToken":null}, {"id":101,"name":"Martin","surname":"Novák","email":"administrator@knihovna.cz","password":"6572bdaff799084b973320f43f09b363","employeePhoto":null,"employeeRole":"ADMINISTRATOR","branch":{"id":3,"name":"Hlavní pobočka","city":"Olomouc","street":"Náměstí republiky 805/15","branchPhoto":null,"email":"hlavni@knihovna.cz","telephone_number":"585 000 111"},"signToken":"b215e765-ed3b-46bb-a74c-94ac19af522b"}]
```

Obrázek 29 – Výstup REST dotazu na získání všech zaměstnanců

REST API není dokumentováno klasickými Java komentáři v kódu, a to z důvodu, že kód není přístupný. Naopak samotné rozhraní pro komunikaci přístupné je a pro externí subjekt je potřeba toto rozhraní zdokumentovat. Část API je zdokumentovaná v následujícím textu.

V API jsou čtyři kořenové adresy, a to pro každou doménu, se kterou se pracuje (kniha, pobočka), nebo která chce vyvolat nějaký požadavek (zaměstnanec, klient). Kořenová relativní URL adresa je /rest, dále za ní následuje jedna z následujících: /book, /branch, /employee, /client. Následně už je pak volán název metody spolu s parametry. U všech metod jsou všechny parametry povinné. Popis názvu metod a jejich funkcionality, včetně vyžadovaných parametrů s jejich typem a návratovým typem, popisují tabulky uvedené v příloze A. Metody, které jsou určeny pro zaměstnance a přihlášené klienty, vyhazují v případě nevalidního přihlašovacího tokenu výjimku `NotValidSignToken`. Metody určené pro administrátora vyhazují v případě volání metody s přihlašovacím tokenem zaměstnance bez role administrátor výjimku `InsufficientRightsException`.

Pro kontrolu toho, co se na webovém serveru děje nebo jestli někde nevznikla nějaká chyba či aplikace úplně nespádla, je v aplikaci implementováno i logování. Logování je implementováno pomocí třídy `Logger` z balíčku `java.util.logging`. Ta poskytuje hned několik metod pro logování informací, které pak můžeme najít v logu serveru.

Celý hotový projekt se tedy skládá ze tří modulů, a to `librarySystem-model` obsahující modelové třídy, `librarySystem-be`, což je backendová část projektu a nakonec `librarySystem-web`, tedy uživatelské rozhraní knihovního systému. Tyto moduly jsou součástí jednoho Maven projektu `librarySystem`.

Webová část byla založena jako klasická Java Web aplikace. Do této aplikace byl přidán Spring framework a ten nakonfigurován, aby si sám naskenoval controllery a mohl používat zdroje jako CSS, JavaScript a obrázky. Tato konfigurace je uložena v souboru `dispatcher-servlet.xml`. Poté byly implementovány samotné controllery a mapovány URL na příslušné metody controllerů. Nakonec byly vytvořeny příslušné .jsp soubory, tedy pohledová stránka projektu.

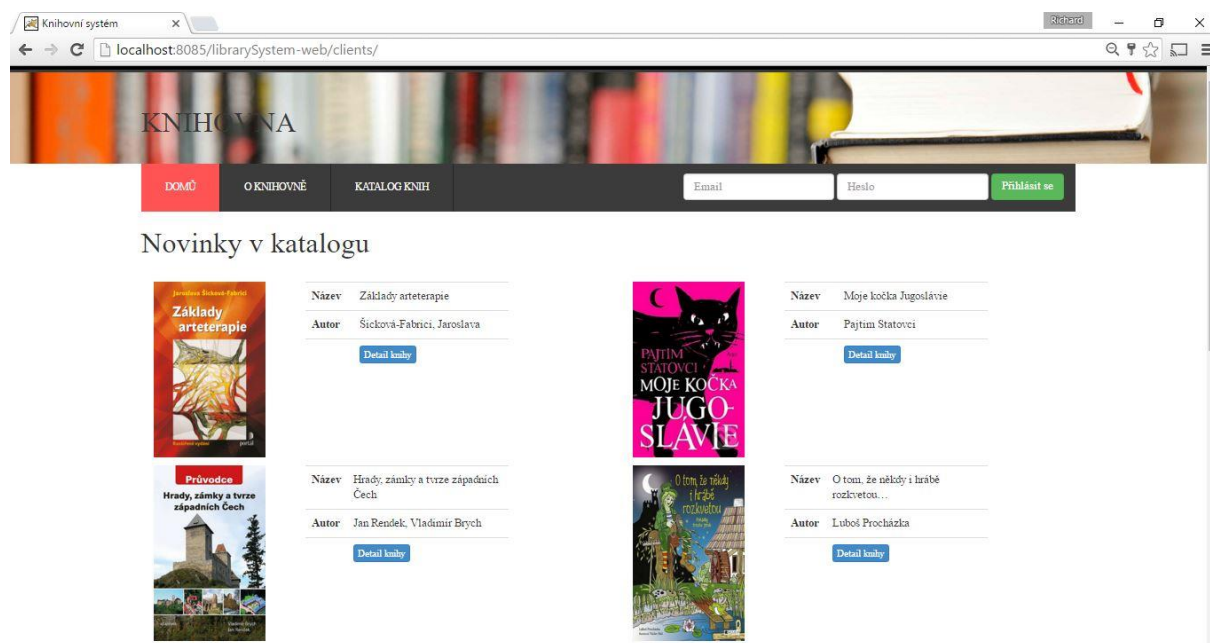
Front-end část aplikace je rozdělena do dvou balíčků, a to `Controllers` a `Utils`. V balíčku `Utils` jsou pomocné nástroje. Dále balíček obsahuje třídu `SessionAttributes`, ve které jsou pak nadefinované názvy session atributů.

Po této implementační části byla aplikace na závěr otestována.

6.5 Představení hotové aplikace

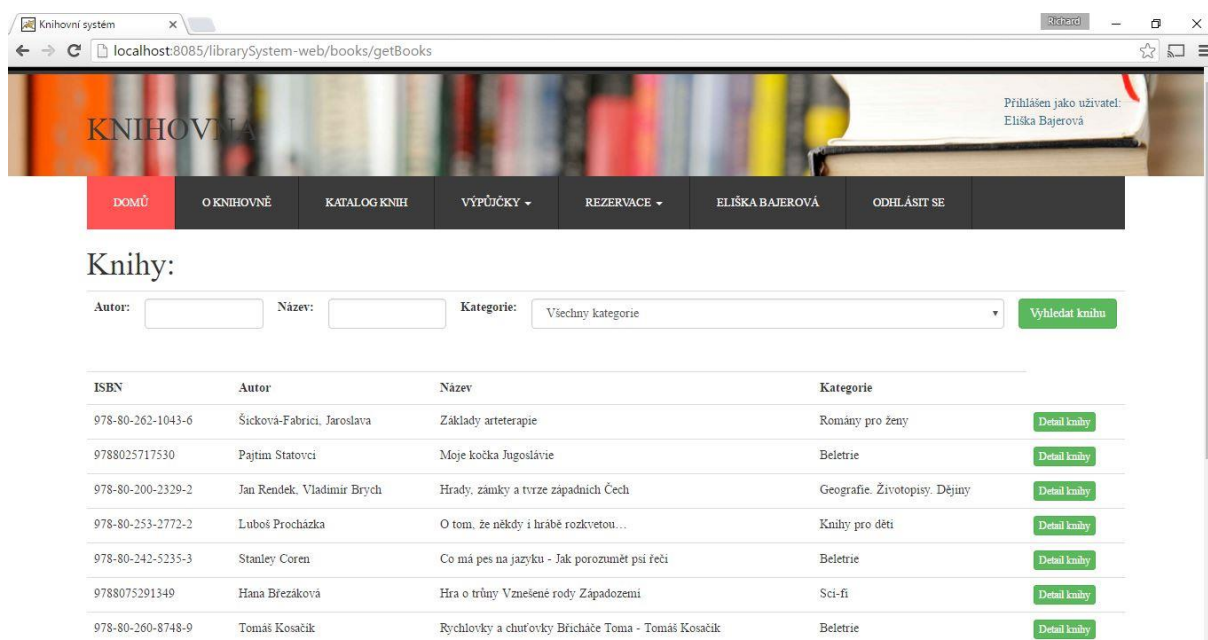
Tato kapitola by měla představit hotovou aplikaci. Bude tedy předvedeno pár obrazovek hotové aplikace. Na obrázku č. 30 lze vidět úvodní obrazovku aplikace s přihlašovacím

formulářem. Je to tedy část aplikace pro nepřihlášené uživatele. Na úvodní obrazovce si lze všimnout knižních novinek. Dále aplikace nabízí záložku „O knihovně“ se základními informacemi o knihovně a záložku „Katalog“ s veřejným katalogem knih dostupných v knihovně.



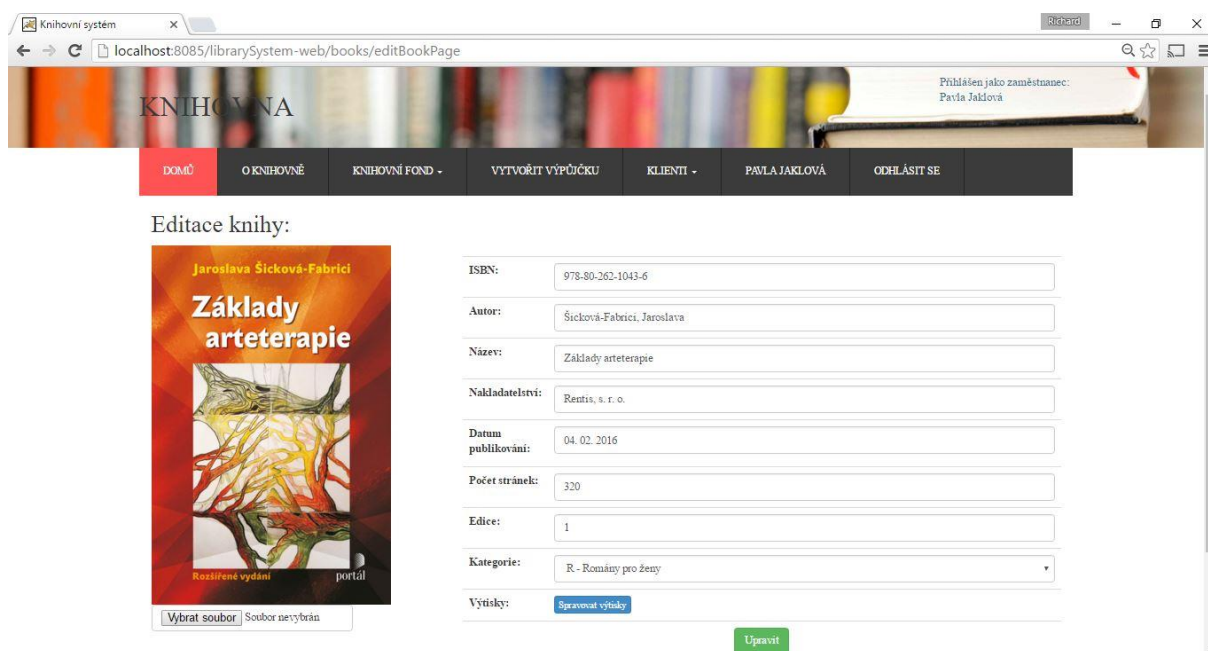
Obrázek 30 – Úvodní obrazovka aplikace

Další obrázek (Obrázek č. 31) zobrazuje obrazovku s filtrací knih v přihlášeném režimu klienta. Záložky „O knihovně“ a „Katalog knih“ zůstávají, přibyla mu navíc záložka „Výpůjčky“ s aktuálními výpůjčkami a historií výpůjček, záložka „Rezervace“ s aktuálními rezervacemi a historií rezervací a záložka ve tvaru „Jméno Příjmení“, kde jsou pak zobrazeny jeho základní údaje.



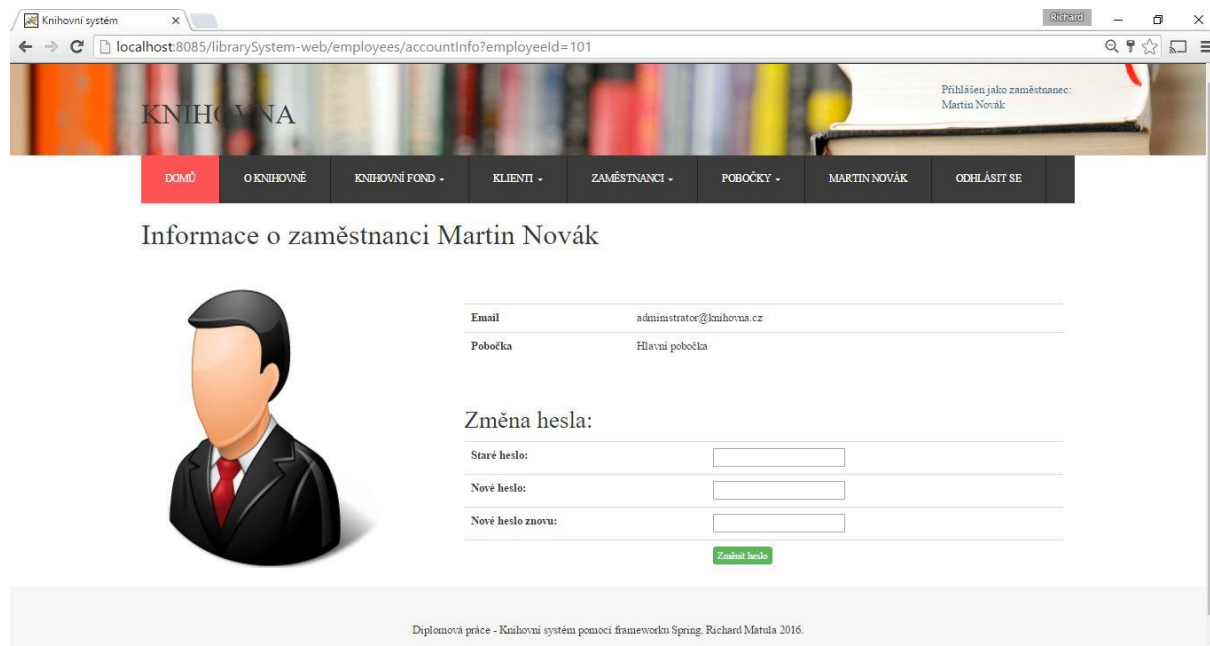
Obrázek 31 – Seznam knížek s vyhledáváním

Obrazovka s formulářem na editaci knihy a zaměstnaneckým rozhraním je na obrázku č. 32. V levé části obrazovky je obrázek knížky a lze jej také editovat. Kniha musí spadat pod některou z kategorií, ty jsou pak k zobrazení v seznamu kategorií (včetně MDT číslování). Je patrné, že zaměstnanecké rozhraní má jiné rozložení záložek. Přibyly tři nové záložky: „Knihovní fond“, kde je skryt v seznamu katalog knih a přidávání knihy, dále „Vytvořit výpůjčku“ a nakonec záložka „Klienti“, která shrnuje informace o klientech, jejich výpůjčkách, rezervacích a možnostích klienta zaregistrovat.



Obrázek 32 – Formulář na editaci knihy

Záložka s informacemi o přihlášeném zaměstnanci je zobrazena na následujícím obrázku (Obrázek č. 33). V rámci této záložky si může zaměstnanec nebo klient také měnit své heslo.



Obrázek 33 – Obrazovka se zaměstnaneckým profilem

Další obrázek (Obrázek č. 34) pak ukazuje obrazovku zaměstnance při vytváření výpůjčky. Na tomto místě v podstatě spáruje klienta s daným výtiskem. Má k dispozici dva seznamy, na prvním jsou klienti seřazení podle příjmení (i se svým id a datem narození) a na druhém pak nevypůjčené výtisky dané pobočky (na které zaměstnanec pracuje) seřazené podle názvu knihy.



Obrázek 34 – Obrazovka s vytvářením výpůjčky

Na posledním obrázku (Obrázek č. 35) je náhled obrazovky, jak v aplikaci vypadá seznam všech zaměstnanců s filtrem (uživatel přihlášen jako administrátor). Administrátor pak má

také jinak rozložené záložky. Nevytváří výpůjčky ani rezervace, ale má k dispozici záložky na správu poboček knihovny a jejich zaměstnanců.

Knihovní systém

localhost:8085/librarySystem-web/employees/getEmployees

Přihlášen jako zaměstnanec: Martin Novák

DOMŮ O KNIHOVNĚ KNIHOVNÍ FOND KLIENTI ZAMĚŠTNANCI POBOČKY MARTIN NOVÁK ODHLÁSIT SE

Seznam zaměstnanců:

Filtr: Píšte sem...

Id	Jméno	Příjmení	Email	Pobočka		
1	Pavla	Jaklová	Pavla.Jaklova@knihovna.cz	Pobočka Tabulový vrch	Vymazat zaměstnance	Editovat zaměstnance
4	Karel	Hnat	Karel.Hnat@knihovna.cz	Pobočka Nefedín	Vymazat zaměstnance	Editovat zaměstnance
101	Martin	Novák	administrator@knihovna.cz	Hlavní pobočka	Vymazat zaměstnance	Editovat zaměstnance

Diplomová práce - Knihovní systém pomocí frameworku Spring. Richard Matula 2016.

Obrázek 35 – Obrazovka se seznamem zaměstnanců

Pro umožnění čtenáři si aplikaci vyzkoušet a otestovat je popsán návod v příloze B této práce.

Závěr

V práci byla provedena rešerše vybraných knihovních IS a byl vysvětlen princip programování webových aplikací na platformě Java. Byl tedy představen návrhový vzor MVC, a to pak i v konkrétní implementaci pomocí Spring frameworku. Dalším bodem práce bylo popsat ORM nástroj Hibernate. Je tedy popsáno, v jakých případech bychom měli zvolit ORM a jak ho nakonfigurovat na reálném projektu. V práci je ORM vyzdvihováno, nicméně nebyl zmíněn fakt, že jeho použitím se mnohdy i snižuje výkon aplikace. Čtvrtá kapitola uzavírá teoretickou část a vysvětluje principy webových služeb, popisuje tedy SOAP protokol, ale více se zaměřuje na REST webové služby, které jsou pak součástí i praktické části.

V praktické části práce je za pomoci těchto nástrojů navrhnut a naimplementován knihovní informační systém. V poslední kapitole jsou tedy popsány nástroje, které byly při analýze, návrhu a implementaci použity. Dále je zde zdokumentována samotná analýza a návrh a popsána struktura projektu.

Diplomová práce splnila veškeré požadavky, které na ni byly kladeny, a to jak po teoretické, tak po praktické stránce. Praktická část práce samozřejmě nedosahuje takových rozměrů jako již existující automatizované knihovní systémy. V práci jsou nicméně základní knihovní procesy automatizovány. Do budoucna by bylo možné práci rozšířit například o modul akvizice, meziknihovní výpůjční systém, dokonalejší knihovní katalog nebo tisk klientských kartiček a čárových kódů výtisků. Pro lepší uživatelský prožitek by bylo možné do knihoven implementovat skenery čárových kódů pro interaktivnější vytváření výpůjček klientům.

Dále je potřeba zmínit, že pro vypracování této práce je v dnešní době dost kvalitní literatury, a to nejen knih, ale i mnoho tutoriálů a článků na internetu, protože oba zde popsané frameworky jsou vyvíjeny již více než deset let. Pro pokročilého Java programátora, nebo toho, kdo by se chtěl naučit programovat moderní webové aplikace na platformě Java, by práce mohla být dobrým zdrojem pro čerpání informací.

Literatura a použité zdroje

- [1] AMUTHAN, G. *Spring MVC Beginner's Guide*. Packt Publishing. June 2014. ISBN 1-78328-488-9.
- [2] BARRY, Douglas K. *Comparison of Object and Relational Terminology* [online]. [cit. 2016-3-22]. Dostupné z: http://www.service-architecture.com/articles/database/comparison_of_object_and_relational_terminology.html
- [3] BARRY, Douglas K. *Service Architecture: Representational State Transfer (REST)* [online]. [cit. 2016-4-7]. Dostupné z: http://www.service-architecture.com/articles/web-services/representational_state_transfer_rest.html
- [4] BARRY, Douglas K. *Service Architecture: Web Services Explained* [online]. [cit. 2016-4-9]. Dostupné z: http://www.service-architecture.com/articles/web-services/web_services_explained.html
- [5] BAUER, Christian a Gavin KING. *Java persistence with Hibernate*. rev. ed. Greenwich: Manning Publications, c2007, xxiii, 408 s. ISBN 19-323-9488-5.
- [6] BERNARD, Emmanuel. *Hibernate Annotations – Reference Guide* [online]. Red Hat Inc. September 15, 2010 [cit. 2016-4-5]. Dostupné z: https://docs.jboss.org/hibernate/stable/annotations/reference/en/html_single/
- [7] Class RestTemplate. *Spring Framework Documentation* [online]. [cit. 2016-4-8]. Dostupné z: <https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/client/RestTemplate.html>
- [8] DEMICHIEL, Linda a SHANNON, Bill. *Java™ Platform, Enterprise Edition (Java EE) Specification, v7* [online]. ORACLE. 7. 3. 2013 [cit. 2016-1-22]. Dostupné z: https://java.net/downloads/javaee-spec/JavaEE_Platform_Spec_PFD_candidate.pdf
- [9] EVANS, Ian. *Java Platform, Enterprise Edition: Your First Cup: An Introduction to the Java EE Platform* [online]. ORACLE. September 2014 [cit. 2016-1-22]. Dostupné z: <http://docs.oracle.com/javaee/6/firstcup/doc/firstcup.pdf>
- [10] Hibernate – Annotations. *Tutorialspoint.com* [online]. [cit. 2016-4-3]. Dostupné z: http://www.tutorialspoint.com/hibernate/hibernate_annotations.htm
- [11] Hibernate – Architecture. *Tutorialspoint.com* [online]. [cit. 2016-3-25]. Dostupné z: http://www.tutorialspoint.com/hibernate/hibernate_architecture.htm
- [12] Hibernate – Configuration. *Tutorialspoint.com* [online]. [cit. 2016-4-4]. Dostupné z: http://www.tutorialspoint.com/hibernate/hibernate_configuration.htm

- [13] Hibernate – Mapping Files. *Tutorialspoint.com* [online]. [cit. 2016-4-3]. Dostupné z: http://www.tutorialspoint.com/hibernate/hibernate_mapping_files.htm
- [14] Hibernate – ORM Overview. *Tutorialspoint.com* [online]. [cit. 2016-3-22]. Dostupné z: http://www.tutorialspoint.com/hibernate/orm_overview.htm
- [15] Hibernate – Sessions. *Tutorialspoint.com* [online]. [cit. 2016-4-14]. Dostupné z: http://www.tutorialspoint.com/hibernate/hibernate_sessions.htm
- [16] Integrovaný knihovní systém Verbis. *KPSYS.cz* [online]. [cit. 2016-04-28]. Dostupné z: <http://kpsys.cz/verbis/index.php/cs/ct-menu-item-16/ct-menu-item-17>
- [17] Interface Session. *Hibernate JavaDoc (4.0.1.Final)* [online]. [cit. 2016-4-14]. Dostupné z: <https://docs.jboss.org/hibernate/orm/4.0/javadocs/org/hibernate/Session.html>
- [18] Introduction to the Standard Directory Layout. *Apache Maven Project* [online]. [cit. 2016-3-28]. Dostupné z: <http://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html>
- [19] iOS Design Patterns: Model View Controller (Part 3). *Treehouse Island, Inc.* [online]. 29 November 2011 [cit. 2016-1-18]. Dostupné z: <http://blog.teamtreehouse.com/ios-design-patterns-model-view-controller-part-3>
- [20] JANKOVSKÁ, Blanka. *Univerzitní knihovna – Současnost* [online]. 4. 9. 2015 [cit. 2016-04-28]. Dostupné z: <http://www.upce.cz/knihovna/o-knihovne/uk-soucasnost.html>
- [21] JENDROCK, Eric et al.. *Java Platform, Enterprise Edition: The Java EE Tutorial* [online]. ORACLE. September 2014 [cit. 2016-1-18]. Dostupné z: <https://docs.oracle.com/javaee/7/JEETT.pdf>
- [22] JOHNSON, Rod et al. *Professional Java development with the Spring Framework* [online]. Indianapolis, IN: Wiley Pub., 2005 [cit. 2016-1-18]. ISBN 0764574833. Dostupné z: <https://iamgodsom.files.wordpress.com/2014/08/wrox-professional-java-development-with-the-spring-framework.pdf>
- [23] JOHNSON, Rod et al. *Web MVC framework - Introduction to Spring Web MVC framework* [online]. 2010 [cit. 2016-3-21]. Dostupné z: <http://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/mvc.html>
- [24] JOHNSON, Rod et al. *Introduction to the Spring Framework* [online]. Spring Framework Reference Documentation. 2015 [cit. 2016-2-10]. Dostupné z: <http://docs.spring.io/spring-framework/docs/current/spring-framework-reference/html/overview.html>

- [25] KUČEROVÁ, Helena. *Automatizovaný knihovní systém*. In: KTD: Česká terminologická databáze knihovnictví a informační vědy (TDKIV) [online]. Praha: Národní knihovna ČR, 2003 [cit. 2016-04-26]. Dostupné z: http://aleph.nkp.cz/F/?func=direct&doc_number=000000085&local_base=KTD
- [26] MATULÍK, Petr. *SPRING FRAMEWORK I – ÚVODNÍ POJMY* [online]. 14. 4. 2005 [cit. 2016-2-8]. Dostupné z: <http://vsadnaju.cz/2005-04/odborne/spring-framework/spring-framework-i-uvodni-pojmy/>
- [27] Maven – Overview: What is Maven? *Tutorialspoint.com* [online]. [cit. 2016-3-28]. Dostupné z: http://www.tutorialspoint.com/maven/maven_overview.htm
- [28] Model-View-Controller Explained. *Mobilefish.com* [online]. [cit. 2016-01-18]. Dostupné z: <http://www.mobilefish.com/tutorials/mvc/mvc.html>
- [29] MySQL. *MySQL.com* [online]. [cit. 2016-3-28]. Dostupné z: <http://www.mysql.com/>
- [30] NetBeans IDE. *Netbeans.org* [online]. [cit. 2016-3-28]. Dostupné z: <https://netbeans.org/>
- [31] OPÁLKOVÁ, Markéta. *OPACy nové generace III – Evergreen a Koha*. Ikaros [online]. 2009, ročník 13, číslo 12 [cit. 2016-04-28]. urn:nbn:cz:ik-13291. ISSN 1212-5075. Dostupné z: <http://ikaros.cz/node/13291>
- [32] PICHLÍK, Roman. *Spring Framework – představení J2EE lightweight kontejneru* [online]. 21. 10. 2005 [cit. 2016-2-8]. Dostupné z: <https://www.interval.cz/clanky/spring-framework-predstaveni-j2ee-lightweight-kontejneru/>
- [33] PUNČOCHÁŘ, Jan. *Přehled možností novějších českých a slovenských automatizovaných knihovních systémů* [online]. 2008 [cit. 2016-04-28]. Bakalářská práce. Masarykova univerzita, Filozofická fakulta. Vedoucí práce Zdeněk Kadlec. Dostupné z: http://is.muni.cz/th/181037/ff_b/
- [34] RŮŽIČKA, Jan. *Vliv informatizace na knihovnictví* [online]. 6. 6. 2010 [cit. 2016-04-26]. Dostupné z: <http://www.inflow.cz/vliv-informatizace-na-knihovnictvi>
- [35] SANDOVAL, José. *RESTful Java web services: master core REST concepts and create RESTful web services in Java*. Birmingham, U.K.: Packt Pub., 2009, v, 241 s. ISBN 978-1847196460.
- [36] Spring Framework – Overview. *Tutorialspoint.com* [online]. 2015 [cit. 2016-2-4]. Dostupné z: http://www.tutorialspoint.com/spring/spring_overview.htm

- [37] STÖCKLOVÁ, Anna. *Automatizace v knihovnách České republiky*. Ikaros [online]. 2006, ročník 10, číslo 5 [cit. 2016-04-28]. urn:nbn:cz:ik-12088. ISSN 1212-5075. Dostupné z: <http://ikaros.cz/node/12088>
- [38] The Java EE 6 Tutorial. *Oracle* [online]. January 2013 [cit. 2016-3-26]. Dostupné z: <https://docs.oracle.com/javaee/6/tutorial/doc/bnbqa.html>
- [39] Ultimate Modeling Power. *Sparxsystems.com.au* [online]. [cit. 2016-3-28]. Dostupné z: <http://www.sparxsystems.com.au/products/ea/>
- [40] WALLS, Craig. *Spring in action*. Fourth Edition. Shelter Island, NY: Manning, 2015, xxiv, 600 s. ISBN 161729120x.
- [41] What is Object/Relational Mapping? *Hibernate.org* [online]. [cit. 2016-3-22]. Dostupné z: <http://hibernate.org/orm/what-is-an-orm/>

Přílohy

Příloha A – <i>Dokumentace REST API</i>	74
Příloha B – <i>Postup nasazení aplikace</i>	81

Tabulka 8 – Book REST API

/book			
Název metody	Popis	Parametry	Návratová hodnota
addBook	přidání nové knihy	signToken (String) – přihlašovací token book (Book) – kniha, která má být přidána	Integer – id nově přidané knihy.
editBook	editace knihy	signToken (String) – přihlašovací token book (Book) – kniha, která má být editována	Book – nově zeditovaná kniha.
deleteBook	smazání knihy	signToken (String) – přihlašovací token bookId (int) – id knihy, která má být smazána	Boolean – příznak, jestli se smazání podařilo nebo ne.
getBooks	získání všech knih	bez parametrů	List<Book> – seznam všech knih.
getBookByISBN	získání knihy na základě ISBN	ISBN (String) – ISBN knihy	Book – kniha se zadaným ISBN.
getBookById	získání knihy na základě jejího id	id (int) – id knihy	Book – kniha se zadaným id.
getBooksByCategory	získání knih na základě kategorie	bookCategory (BookCategory) – kategorie knihy	List<Book> – seznam knih na základě zadané kategorie.
getBookCategories	vrací seznam všech kategorií	bez parametrů	List<BookCategory> – seznam všech dostupných kategorií knih.
movePrint	přesune výtisk na jinou pobočku	signToken (String) – přihlašovací token printId (int) – id výtisku newBranchId (int) – id nové pobočky	Print – výtisk se změněnou pobočkou.
getNewBooks	získání knižních novinek	bez parametrů	List<Book> – seznam 10 knižních novinek.

/book			
Název metody	Popis	Parametry	Návratová hodnota
findBook	metoda pro nalezení knížky podle názvu, autora nebo kategorie (kombinace)	name (String) – název knížky author (String) – autor bookCategory (String) – kategorie	List<Book> – seznam nalezených knih.
getAllNotLoanedPrintsByBranch	vrátí všechny nevypůjčené výtisky knih podle pobočky	signToken (String) – přihlašovací token branchId (int) – id pobočky	List<Print> – seznam výtisků.
addPrint	přidá výtisk dané knížce	signToken (String) – přihlašovací token print (Print) – výtisk, který se má přidat	Print – nově přidáný výtisk.
deletePrint	vymaže výtisk dané knihy	signToken (String) – přihlašovací token printId (int) – id výtisku, který se má vymazat	Boolean – příznak, jestli se vymazání podařilo nebo ne.
getBookPrints	vrátí všechny výtisky dané knihy	bookId (int) – id knížky	List<Print> – seznam všech výtisků knížky podle jejího id.

Tabulka 9 – Branch REST API

/branch			
Název metody	Popis	Parametry	Návratová hodnota
addBranch	přidání nové pobočky	signToken (String) – přihlašovací token branch (Branch) – pobočka, která má být přidána	Integer – id nově přidané pobočky.
editBranch	editace pobočky	signToken (String) – přihlašovací token branch (Branch) – pobočka, která má být editována	Branch – nově zeditovaná pobočka.

/branch			
Název metody	Popis	Parametry	Návratová hodnota
deleteBranch	smazání pobočky	signToken (String) – přihlašovací token branchId (int) – id pobočky, která má být smazána	Boolean – příznak, jestli se smazání podařilo nebo ne.
getBranches	získání všech poboček	bez parametrů	List<Branch> – seznam všech poboček.
getBranch	získání pobočky podle jejího id	branchId (int) – id pobočky	Branch – pobočka podle zadaného id.

Tabulka 10 – Client REST API

/client			
Název metody	Popis	Parametry	Návratová hodnota
signIn	přihlášení klienta	email (String) – email klienta password (String) – heslo uživatele (v hashi)	Client – přihlášený klient.
logOut	odhlášení klienta	signToken (String) – přihlašovací token klienta clientId (int) – id klienta	Boolean – příznak, jestli se odhlášení podařilo nebo ne.
addClient	přidání nového klienta (registrace)	signToken (String) – přihlašovací token client (Client) – klient, která má být přidán	Integer – id nově přidaného klienta.
editClient	editace klienta	signToken (String) – přihlašovací token client (Client) – klient, která má být editován	Client – nově zeditovaný klient.
deleteClient	smazání klienta	signToken (String) – přihlašovací token clientId (int) – id klienta, který má být smazán	Boolean – příznak, jestli se smazání podařilo nebo ne.
getClients	získání všech klientů	signToken (String) – přihlašovací token	List<Client> – seznam všech klientů.

/client			
Název metody	Popis	Parametry	Návratová hodnota
getClientById	získání klienta podle jeho id	signToken (String) – přihlašovací token clientId (int) – id klienta	Client – klient podle zadaného id.
getClientByEmail	získání klienta podle jeho emailu	signToken (String) – přihlašovací token email (String) – email klienta	Client – klient podle zadaného emailu.
createReservation	vytvoření rezervace	signToken (String) – přihlašovací token printId (int) – id výtisku clientId (int) – id klienta	Integer – id nově vytvořené rezervace. V případě null hodnoty se rezervaci nepodařilo vytvořit.
cancelReservation	zrušení rezervace	signToken (String) – přihlašovací token reservationId (int) – id rezervace	Boolean – příznak, jestli se podařilo rezervaci zrušit nebo ne.
prolongateLoan	prodloužení výpůjčky o 1 měsíc	signToken (String) – přihlašovací token loanId (int) – id výpůjčky	Boolean – příznak, jestli se prodloužení podařilo nebo ne (prodloužit lze jen jednou).
getClientReservations	vrátí aktuální rezervace daného klienta	signToken (String) – přihlašovací token clientId (int) – id klienta	List<Reservation> – seznam rezervací daného klienta.
getClientReservationsHistory	vrátí historii rezervací klienta	signToken (String) – přihlašovací token clientId (int) – id klienta	List<Reservation> – seznam historie rezervací daného klienta.
getClientLoans	vrátí aktuální výpůjčky daného klienta	signToken (String) – přihlašovací token clientId (int) – id klienta	List<Loan> – seznam výpůjček daného klienta.
getClientLoansHistory	vrátí historii výpůjček daného klienta	signToken (String) – přihlašovací token clientId (int) – id klienta	List<Loan> – seznam historie výpůjček daného klienta.

/client			
Název metody	Popis	Parametry	Návratová hodnota
getClientExpiredLoans	vrátí všechny expirované výpůjčky daného klienta	signToken (String) – přihlašovací token clientId (int) – id klienta	List<Loan> – seznam expirovaných výpůjček daného klienta.
changeClientPassword	změní danému klientovi heslo	signToken (String) – přihlašovací token password (String) – nové heslo (v hashi)	Client – klient s nově změněným heslem.

Tabulka 11 – Employee REST API

/employee			
Název metody	Popis	Parametry	Návratová hodnota
signIn	přihlášení zaměstnance	email (String) – email zaměstnance password (String) – heslo zaměstnance (v hashi)	Employee – přihlášený zaměstnanec.
logout	odhlášení zaměstnance	signToken (String) – přihlašovací token zaměstnance employeeId (int) – id zaměstnance	Boolean – příznak, jestli se odhlášení podařilo nebo ne.
addEmployee	přidání nového zaměstnance (registrace)	employee (Employee) – zaměstnanec, který má být přidán	Integer – id nově přidaného zaměstnance.
editEmployee	editace zaměstnance	signToken (String) – přihlašovací token employee (Employee) – zaměstnanec, který má být editován	Employee – nově zeditovaný zaměstnanec.

/employee			
Název metody	Popis	Parametry	Návratová hodnota
deleteEmployee	smazání zaměstnance	signToken (String) – přihlašovací token employeeId (int) – id zaměstnance, který má být smazán	Boolean – příznak, jestli se smazání podařilo nebo ne.
getEmployees	získání všech zaměstnanců	signToken (String) – přihlašovací token	List<Employee> – seznam všech zaměstnanců.
getEmployee	získání zaměstnance podle jeho id	signToken (String) – přihlašovací token employeeId (int) – id zaměstnance	Employee – zaměstnanec podle zadaného id.
getEmployeesByBranch	získání zaměstnanců podle pobočky	signToken (String) – přihlašovací token branchId (int) – id pobočky	List<Employee> – seznam všech zaměstnanců podle pobočky.
createLoan	vytvoření výpůjčky	signToken (String) – přihlašovací token clientId (int) – id klienta employeeId (int) – id zaměstnance, který výpůjčku vytváří printId (int) – id výtisku	Integer – id nově vytvořené výpůjčky. V případě null hodnoty se výpůjčku nepodařilo vytvořit.
getLoans	získání všech aktuálních výpůjček	signToken (String) – přihlašovací token	List<Loan> – seznam všech aktuálních výpůjček.
getReservations	získání všech aktuálních rezervací	signToken (String) – přihlašovací token	List<Reservation> – seznam všech aktuálních rezervací.
cancelLoan	zrušení dané výpůjčky	signToken (String) – přihlašovací token loanId (int) – id výpůjčky	Boolean – příznak, jestli se zrušení podařilo nebo ne.
prolongateLoan	prodloužení výpůjčky	signToken (String) – přihlašovací token loanId (int) – id výpůjčky	Boolean – příznak, jestli se prodloužení výpůjčky podařilo nebo ne.

/employee			
Název metody	Popis	Parametry	Návratová hodnota
cancelReservation	zrušení dané rezervace	signToken (String) – přihlašovací token reservationId (int) – id rezervace	Boolean – příznak, jestli se zrušení podařilo nebo ne.
createReservation	vytvoření rezervace	signToken (String) – přihlašovací token	Integer – id nově vytvořené rezervace. V případě null se rezervaci nepodařilo vytvořit.
changeEmployeePassword	změna hesla zaměstnance	signToken (String) – přihlašovací token password (String) – nové heslo (v hashi)	Employee – zaměstnanec s nově editovaným heslem.

Příloha B – *Postup nasazení aplikace*

Pro umožnění nasazení a testování hotové aplikace by mělo být postupováno podle následujících kroků. Nejdříve je potřeba na webový server nainstalovat MySQL databázi. V databázi je pak nutné si vytvořit účet a databázové schéma (nejlépe s UTF-8 kódováním pro správné zobrazování češtiny).

Pro vytvoření databázového schématu s UTF-8 kódováním byl použit následující příkaz:

```
CREATE DATABASE `librarysystemdb` CHARACTER SET utf8 COLLATE utf8_general_ci;
```

Po zkopírování zdrojových kódů z příloženého CD se v adresáři projektu librarySystem-be bude po kompilaci nacházet v adresáři target WAR archiv librarySystem-be-1.0. Tento archiv obsahuje funkční zdrojové kódy připravené k nasazení na Java webový server. Po nasazení tohoto archivu na server nástroj Hibernate automaticky vytvoří z modelových tříd tabulky, které jsou pak připraveny k použití (projekt lze také spustit a nasadit i přes některé z dostupných IDE). Po tomto kroku pak lze použít skripty přiložené na CD pro naplnění dat do databáze – tabulky jsou již vytvořeny nástrojem Hibernate. CD obsahuje jak skript pro naplnění každé tabulky zvlášť, tak skript, který naplní všechny tabulky najednou (data.sql).

Aplikaci lze samozřejmě spouštět na svém schématu se svým účtem. To ale vyžaduje spuštění aplikace v některém z IDE se správnou konfigurací Hibernate v souboru hibernate.cfg.xml, následné sestavení aplikace a její nasazení.

Po těchto krocích je připraven běžící backend projekt s databází a stačí tak nasadit frontend část aplikace. Ta se bude po kompilaci nacházet v adresáři librarySystem-web/target a bude zde dostupná také jako sestavený WAR archiv. Je potřeba si dát pozor, na které adrese běží backend část projektu. Aplikace byla testována na localhostu s portem 8085, tedy součástí tohoto WAR archivu je tato konfigurace a nebude v případě backendu běžícího na jiném portu fungovat. Při spouštění aplikace v IDE jsou pak adresy REST API nastaveny v kódu ve třídě WebUtils jako statická konstanta BE_ENDPOINT.

Po tomto kroku by měla již běžet celá funkční aplikace. Hesla jsou v databázi zašifrovaná, ale pro účely testování jsou po importu dat všude „asdf123“ (hesla pak lze změnit). Administrátorský účet má email administrator@knihovna.cz. Zaměstnanecké a klientské emaily lze zjistit podle importovaných dat v databázi.