Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informatiky a kvantitativních metod

Backendové technologie

Bakalářská práce

Autor: Artur Hamza

Studijní obor: Aplikovaná informatika

Vedoucí práce: doc. Ing. Ph.D. Filip Malý

Hradec Králové srpen 2020

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

vlastnoruční podpis

V Hradci Králové dne Artur Hamza

Poděkování:

Děkuji vedoucímu bakalářské práce doc. Ing. Ph.D. Filipovi Malému za metodické vedení práce.

Anotace

Tato bakalářská práce se zabývá porovnáním backendových technologií v Javě. V průběhu bakalářské práce jsou v teoretické části uvedeny a popsány vybrané backendové technologie. Ty jsou následně v praktické části implementovány a porovnány.

Tato práce může mít určitý význam pro zájemce o poznání backendových technologií Javy.

V této bakalářské práci jsou popsány pojmy „frontend“ a „backend“ a rozdíl mezi nimi. Následně je popsán programovací jazyk Java a podrobně jeho varianta Java EE, včetně části jejích technologií. Poté jsou uvedeny a popsány vybrané technologie, konkrétně modulární Spring framework, MVC framework Struts, ORM framework Hibernate, specifikace Java Persistence API, technologie MyBatis, která se zaměřuje na použití SQL dotazovacího jazyka, specifikace Java Data Objects, Data Nucleus, implementující specifikace JPA a JDO, JDBC a frameworky zabezpečení Apache Shiro, Java authentication and authorization service a OACC. Také je s nimi porovnán framework Spring Security.

Tyto technologie jsou následně implementovány v aplikaci ztrát a nálezů a podobné se porovnávají. Konkrétně v kategorií MVC frameworků, databázových technologií a frameworků zabezpečení. Na konci srovnání technologií dané kategorie je také uveden teoretický výběr nejlepší technologie pro aplikaci. Konkrétně jsou vybrány Spring Web, Spring Security a Hibernate.

Annotation

Title: Backend technologies

This Bachelor´s thesis deals with comparison of backend technologies in Java. During the Bachelor´s thesis in theoretic part there are introduced and described chosen backend technologies. Then in practical part they are implemented and compared.

In this Bachelor´s thesis are described terms “frontend” and “backend” and differences between them. Then is described programming language Java and in detail its variant Java EE, with a part of its technologies included. Then there are introduced and described chosen technologies, specifically modular Spring framework, MVC framework Struts, ORM framework Hibernate, specification Java Persistence API, technology MyBatis, which focuses on using the query language SQL, specification Java Data Objects, DataNucleus, implementing specifications JPA and JDO, JDBC and security frameworks Apache Shiro, Java authentication and authorization service and OACC. Spring Security framework is compared with them too.

These technologies are then implemented in the application of found and lost items and similar ones are compared. Specifically in the category of MVC frameworks, database technologies and security frameworks. At the end of the comparisons of the technologies for each category there is also stated theoretical choice of the best technology for the application. Which are Spring Web, Spring Security and Hibernate.

Obsah

Seznam obrázků

Rejstřík

1 Úvod

1.1 Popis

Tato bakalářská práce se zabývá problematikou backendových technologií v Javě. V teoretické části jsou uvedeny pojmy backend a frontend a rozdíl mezi nimi, informace o objektově orientovaném programovacím jazyce Javě. Jeho varianta Java EE je následovně podrobněji popsána, včetně jejích určitých technologií, které mohou následné backendové technologie využívat (a zároveň Java EE také patří mezi backendové technologie).

Dále jsou vybrané backendové technologie popsány, aby bylo jasné, na co se zaměřují a jak fungují. Poté následuje praktická část, ve které jsou dané technologie implementovány, jejích implementace je popsána v bakalářské práci. A následně jsou mezi sebou podobné technologie porovnány.

1.2 Cíl práce

Smyslem a účelem práce je prozkoumat backendové technologie v programovacím jazyce Javě, vybrané technologie popsat, implementovat je v různých variantách aplikace, následně je porovnat a na implementacích ukázat mezi nimi rozdíly.

Výzkumné otázky mohou být v rámci této bakalářské práce následující: Co je to backend a frontend? Co je to Java, a hlavně jak funguje Java EE? Jaké backendové technologie jsou vhodné k výběru a porovnávání? Na co se každá z vybraných backendových technologií zaměřuje a jaké jsou mezi nimi rozdíly? Jak je implementovat do aplikací? Budou muset být ty různé varianty aplikace moc rozdílné? Budou spolu backendové technologie při jejich společné implementaci navzájem kompatibilní?

Vhodné je se také zamyslet nad tím, co je při této bakalářské práci nevhodné.: Je vhodné při práci s technologií čerpat jenom z jednoho druhu zdroje, jako je například tutoriál, namísto použití více druhů zdroje, pokud jsou k dispozici, jako je třeba dokumentace API technologie a dokumentace přímo dané technologie? Je vhodné implementovat více technologií do různých částí aplikace bez ověření, zda jsou spolu dané technologie navzájem kompatibilní?

Je dobrým nápadem také porovnávat úplně odlišné technologie, jako například MVC framework s technologií ORM, kde tudíž není co porovnávat?

1.3 Metodika zpracování

Cílem bakalářské práce se prozkoumat backendové technologie v programovacím jazyce Javě, vybrané technologie popsat, porovnat a ukázat rozdíly mezi nimi v různých variantách aplikace s implementovanými porovnávanými backendovými technologiemi.

Jak už bylo zmíněno v cíli této bakalářské práce, tak výzkumné otázky mohou být následující: Co je to backend a frontend? Co je to Java, a hlavně jak funguje Java EE? Jaké backendové technologie jsou vhodné k výběru a porovnávání? Na co se každá z vybraných backendových technologií zaměřuje a jaké jsou mezi nimi rozdíly? Jak je implementovat do aplikací? Budou muset být ty různé varianty aplikace moc rozdílné? Budou spolu backendové technologie při jejich společné implementaci navzájem kompatibilní?

K zodpovězení těchto výzkumných otázek a během samotné bakalářské práce byly analyzovány odborné texty, dokumentace a tutoriály. Implementací v Javě potom dále byly prozkoumány a porovnány vybrané backendové technologie.

1.4 Literární rešerše

V následující části bakalářské práce se nacházejí literární rešerše.:

1.4.1 A Review on Java Frameworks for Web Applications

Článek se zabývá problematikou použití vybraných frameworků v Javě v rámci tvorby webových aplikací. Je zde popsána architektura frameworku Springu, frameworku Strutsu, frameworku Hibernate a technologie JSP. Nakonec jsou ve článku zmíněny výhody využití těchto vybraných frameworků v Javě. (1)

1.4.2 Review on JPA Based ORM Data Persistence Framework

Autoři Neha Dhingra, Emad Abdelmoghith, and Hussien T. Mouftah, se v článku Review on JPA Based ORM Data Persistence Framework zabývají poskytovateli techniky zvané ORM neboli tzv. Object/Relational mapping. (2)

Autoři se v článku zaměřují hlavně na ORM frameworky podporující Java Persistence API, ve zkratce JPA. Autoři v článku diskutují o porovnání zaměřeném na samotný JPA a jeho implementace. (2)

Nejdříve jsou v článku uvedeni tzv. JPA poskytovatelé, a to Hibernate, EclipseLink, OpenJPA a DataNucleus. Tyto technologie jsou mezi sebou následně porovnány v jednotlivých částech, zaměřených na porovnání spojení a prvků konfigurace, vyrovnávací paměti, dotazů a transakcí, automatického vložení, JPA objekt, niťování, mapování a distribuované transakce a optimalizaci výkonu. (2)

V závěru autoři článku uvádí, že poskytovatelé JPA persistence mohou vytvořit efektivní API a být orientované na výkon. Následovně uzavírají porovnání technologií ve prospěch frameworku Hibernate, který zhodnocují, jako nejvýjimečnější v případě podpory dokumentace a knihoven ke správě komplikovaných odpovědností. Dále zmiňují, že v budoucnosti, optimální řešení vylepšující škálovatelnost API skrze vhodné dokumentace komplexních úkolů se zlepšením mechanismu vyrovnávací paměti, je OpenJPA. (2)

1.4.3 Espresso: Brewing Java For More Non-Volatility with Non-volatile Memory

Autoři Mingyu Wu, Ziming Zhao, Hayou Li, Heting Li, Haibo Chen, Binyu Zang a Haibing Guan, ve článku Espresso: Brewing Java For More Non-Volatility with Non-volatile Memory zmiňují problematiku většího využití nevolatilní paměti v Javě. Jako řešení autoři nabízí framework Espresso, sjednocený perzistentní framework, který podporuje zprávu persistence dobře zrněné a hrubě zrněné, zatímco je kompatibilní s datovými strukturami v existujících programech Javy a značně zrychlí výkon zprávy perzistence. Framework ukládá persistentní objekty Javy prostřednictvím tzv. Persistent Java Heap (PJH), což je tzv. heap neboli hromada založená na nevolatilní paměti. (3)

Autoři v článku dále popisují framework Espresso. V druhé sekci článku jsou přezkoumány dva hlavní přístupy pro zprávu perzistence, a to přístup hrubě zrněné perzistence v JPA a přístup dobře zrněné persistence v PCJ (Persistence Collections for Java). (3)

Ve třetí sekci článku autoři uvádí přehled PJH a jazykových rozšíření k manipulaci persistentních datových objektů. Ve čtvrté sekci autoři hlouběji popisují mechanismy frameworku k zaručení konzistence havárie PJH. V páté sekci dále autoři prezentují abstrakci Persistentního Javovského Objektu (PJO) s perzistentním programujícím modelem. V šesté sekci článku autoři vyhodnocují svůj vlastní design. Následovně, v sedmé sekci autoři popisují související práci. (3)

V závěru článku autoři článek uzavírají tím, že propagoval framework Espresso, aby programátoři Javy mohli využit nevolatilní paměť pro zjednodušení zprávy perzistence. Dále Espresso se skládá z PJH a PJO. V závěru autoři také zmiňují výsledek vyhodnocení, který ukázal, že zjednodušená zpráva perzistence značně zvýšila výkon. (3)

1.4.4 Database Design and Implementation – JDBC

Autor Edward Sciore, v knize Database Design and Implementation, v části JDBC, popisuje API JDBC a jeho implementaci. Konkrétně popisuje jeho základní rozhraní, připojování se k databázi a odpojování se z ní, výjimky SQL, práci s SQL výroky a výsledky přijatými z databáze a využití dotazovacích metadat. V této časti knihy autor také dále popisuje pokročilejší práci s JDBC, například práci s transakcemi a rozhraní PreparedStatements. (4)

V závěru části této knihy je celá kapitola shrnuta. Například JDBC metody zpravují přesun dat mezi klientem Javy a databází. Dále sady výsledků s pojení drží prostředky, které můžou ostatní klienti potřebovat. Proto by je měl JDBC klient uzavřít co nejdříve, jak je to možné. A základní JDBC klient ignoruje existence transakcí. Databáze provádí tyto klienty v režimu automatického potvrzení, což znamená, že každý SQL výrok je zároveň i transakcí. (4)

1.4.5 Extended Role-Based Access Control with Context-Based Role Filtering

Autoři Gang Liu, Runnan Zhang, Bo Wan, Shaomin Ji a Yumin Tian se ve článku Extended Role-Based Access Control with Context-Based Role Filtering zabývají modelem uživatelských rolí tzv. přístupové ovládání založené na rolích neboli „role-based access control“, ve zkratce RBAC. V modelu se autoři konkrétně zabývají problémem ve zprávě uživatelských roli, aktivace role sezení neboli „session role activation (SRA)“. Tento problém znamená, že uživatelé znají úkoly pro sezení, ale nejsou schopni popsat požadovaná oprávnění přesně, což vede ke komplikaci aktivace rolí pro sezení. (5)

Jako řešení tohoto problému autoři navrhují rozšířený RBAC model, který používá kontextové informace pro filtrování rolí a získává kandidátní role související se sezením. Ve článku je nejdříve analyzován vztah mezi znalostí uživatelského zabezpečení a problémem zprávy rolí. Je navrhován rozumnější předpoklad uživatelského dokonalených znalostí zabezpečení. SRA problém je definován na základě tohoto předpokladu. (5)

Dále autoři v článku navrhuji rozšíření RBAC model s kontextově založenou funkcí filtrování role k řešení problému SRA. Používá kontextové podmínky k výběru sezení, které je víc související s uživatelskými rolemi, což značně snižuje číslo kandidátních rolí. Rozšířený RBAC model pomáhá uživatelům, kteří nemají povědomí o zabezpečení, aktivovat vhodné role pro sezení. (5)

Tudíž, dynamická zpráva vztahu sezení a role je realizována, což splňuje princip minimálního oprávnění. Autoři v této části také k ukázce efektivity modulu používají metodu simulace. Poté je rozšířený RBAC model implementován v rámci frameworku Shiro. (5)

V závěru jednak autoři zmiňují výsledky simulace, které ukazují, že mechanismus kontextu sezení rozšířeného RBAC modulu rozumně vybírá kandidátní role prostřednictvím automatického vnímání změn v sezení. Model také efektivně snižuje počet kandidátních rolí jejích filtrováním. (5)

Autoři také ale docházejí k závěru, že tato studie má pořád omezení v aplikačním prostředí. Zde, kromě použití kontextových informací, jako omezení, další informace související s uživatelem nebo sezením mohou být použity k filtrování rolí. Přesto ale autoři v závěru věří, že jejich rozšířený RBAC model používající kontextová omezení jsou efektivním řešením SRA problému. (5)

1.4.6 Solution to cubic equation using Java programming

Autoři Shouthiri Partheepan a Disne Sivalignam, ve článku Solution to cubic equation using Java programming se zabývají vývoje programujícího řešení kubických výpočtů s využitím Javy. Ve článku konkrétně navrhují počítačové programy řešící kubické výpočty založené na metodách diskriminantu a generaci kubických výpočtů založených na reálných řešeních. (6)

Během vývoje řešení autoři používají přístup diskriminantu. Algoritmus založen na tomto přístupu je implementován s využitím programovacího jazyku Javy a jeho kroky jsou následně v článku popsány. Následovně autoři testují aplikaci například použitím různých kubických výpočtů k nalezení reálných kořenů. Stejné výpočty jsou také provedeny ručně. (6)

V závěru autoři rekapitulují kroky při návrhu řešení. (6)

1.4.7 A Comprehensive Study of Bloated Dependencies in the Maven Ecosystem

Autoři César Soto-Vatero, Nicolas Harrand, Martin Monperrus a Benoit Baundry se ve článku A Comprehensive Study of Bloated Dependencies in the Maven Ecosystem zabývají problémem tzv. nafouklých závislostí. (7)

Tento problém se vyskytuje ve zprávě závislostí. Nafouklé závislosti jsou balíčky, které jsou deklarované, jako závislosti pro aplikaci, ale nejdou důležité pro její sestaveni nebo běh. Hlavním problémem s nafouknutými závislostmi je zbytečné množství kódu navíc ve finální verzi nasazeného binárního souboru, než je nutné. Což je problémem v případě, kdy se aplikace posílá přes síť, jako jsou například webové aplikace, anebo pokud je nasazena v malých zařízeních. Nafouknuté závislosti navíc také mohou obsahovat zranitelný kód, který může být zneužit, zatímco je zbytečný pro aplikaci. Tyto problematické závislosti také zbytečně komplikují zprávu a vylepšování softwarových aplikací. (7)

Autoři v článku nabízí novou, unikátní a ve velkém rozsahu analýzu o nafouknutých závislostech. Autoři se zde zaměřují na Maven. Pro analýzu tisíců artefaktů v repositáři Maven Central, která je v případě ruční analýzy problematická, autoři vytvořili a použili nástroj, který nazvali, jako tzv. DepClean, který tuto analýzu dělá automaticky. (7)

Autoři dále ve druhé sekci článku uvádí klíčové koncepty zprávy závislostí v Mavenu a prezentuje ilustrativní vzor. Dále autoři, ve třetí sekci článku, uvádí nové terminologie a popisuje implementaci DepCleanu. (7)

Ve čtvrté sekci autoři prezentují výzkumné otázky, které se týkají této studie a následující metodologie. (7)

Otázky, které pro výzkum zde autoři položili, se například ptají na výskyt nafouknutých závislostí, zda praktiky znovupoužití ovlivňují nafouknuté závislosti a do jakého měřítka jsou vývojáři ochotni odstranit jednotlivé druhy nafouknutých závislostí. V páté sekci článku se nachází experimentální výsledky pro každou výzkumnou otázku. Autoři v šesté a sedmé sekci článku poskytují vyčerpávající diskusi o výsledcích, které obdrželi, a také v těchto sekcích autoři popisují hrozby validity studie. (7)

V závěru například autoři zmiňují výsledek analýzy DepCleanu, kterým zanalyzovali 722 444 závislostí 9 639 artefaktu z Maven Central. Výsledek ukázal, že 2,7 % z těchto závislostí jsou nafouknuté. Za základě tohoto výsledku autoři dále určili dvě možné příčiny problému, a to kaskádu nechtěných transitivních závislostí navozených přímými závislostmi. A mechanismus dědění závislosti vícemodulových projektů Mavenu. (7)

Autoři dále dochází například k výsledku, že vývojáři jsou ochotni odstranit nafouknuté přímé závislosti. Ale v případě vyloučení nafouknutých transitivních závislostí jsou vývojáři skeptičtí. (7)

1.4.8 The Comparison Firebase Realtime Database and MySQL Database Performance Using Wilcoxon Signed-Rank Test

Autoři Margaretha Ohyver, Jurike V. Moniaga, Iwa Sungkawa, Bonifasius Edwin Subagyo a Ian Argus Chandra, ve článku The Comparison Firebase Realtime Database and MySQL Database Performance Using Wilcoxon Signed-Rank Test porovnávají výkon technologií Firebase Realtime Database a MySQL, sloužících, jako systém zprávy databáze pro jejích mobilní aplikaci pro denní potřeby stravy batolat. (8)

Autoři testují výkon technologií formou testu, který je proveden, jako tzv. CRUD (create, read, update and delete) operace v obou databázích, v rámci mobilní aplikace. V každé databázi jsou zmíněné operace provedené padesát krát. Dále autoři popisují a diskutují výsledky testů jednotlivých CRUD operací. (8)

V závěru článku autoři zmiňují, že Firebase Realtime Database má lepší výkon v případě odpovědi než MySQL. Navíc také uvádím, že Firebase Realtime Database je na základě jejích klíčových prvků vhodnější pro jejich aplikaci. (8)

2 Popis backendových technologií

V teoretické části práce bude uveden rozdíl mezi frontendem a backendem, popsána Java a Java EE a její vybrané technologie, poté budou popsány vybrané backendové technologie. Ty budou na konci této části porovnány.

2.1 Rozdíl mezi tzv. backendem a frontendem

Pro porozumění problematice nyní bude popsán rozdíl mezi backendem a frontendem. Příklad si popíšeme na webové aplikaci, kde platí vztah mezi klientem a serverem, kde klient je osobní počítač, na kterém prostřednictvím webového prohlížeče pracuje uživatel s webovou aplikací. A server, což je počítač, který reaguje na dotazy klienta, zpracovává data a posílá odpověď a zpracovaná data klientovi.

Frontendové technologie neboli tzv. „frontend“ jsou technologie, které zpracovávají data na straně klienta neboli to, co uživatel vidí na svém počítači. Mezi frontend v případě webové aplikace patří jazyky, jako je HTML, CSS, Javascript apod. Frontend sice není hlavním tématem této bakalářské práci, ale v praktické části se s ním také bude muset pracovat.

Dále backendové technologie neboli tzv. „backend“, jsou technologie, které zpracovávají data na straně serveru. Sem patří obvykle dvě vrstvy, jednak samotná webová aplikace, která běží na serveru, a která může být naprogramována v různých programovacích jazycích, jako je například Java, C# apod. Další vrstvou je databázový systém, jelikož webová aplikace si obvykle ukládá data do určité databáze. Zde mohou být například využiti SQL databázové systémy, jako je SQL Server od Microsoftu, Oracle SQL a MySQL.

Tato bakalářská práce se bude zabývat backendovými technologiemi v Javě, jelikož kromě základního programovacího jazyka lze využít také různé další technologie.

2.2 Java

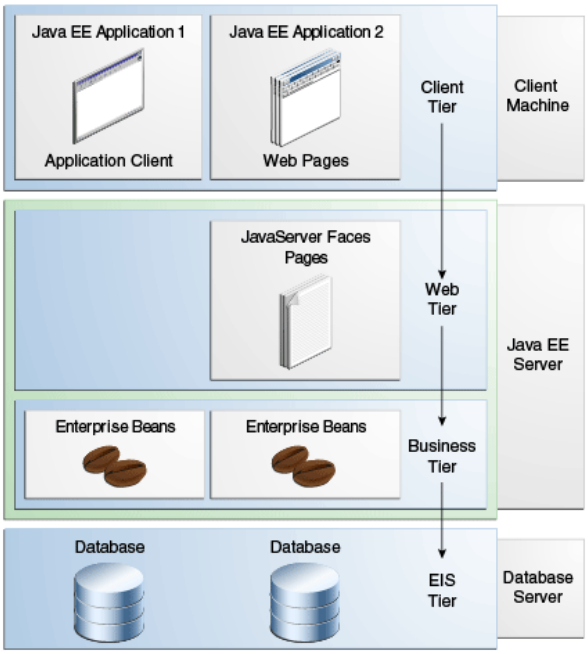
Java je objektově orientovaný programovací jazyk, jehož vývoj začal v roce 1991 pod společností Sun Microsystems. Java byla vydána v roce 1995. V současnosti patří společnosti Oracle, která tento programovací jazyk dále vyvíjí. (9)

Java má více variant. Základní variantou je Java SE, která slouží k vývoji Javovských aplikací. (10) Pro tuto bakalářskou práci podstatnější je ale Java EE, která slouží k vývoji softwaru pro podniky. (11)

Java EE používá zjednodušený model programování. Nevyžaduje využití popisků XML nasazení. Namísto nich lze dané informace vložit, jako anotaci přímo do souboru se zdrojovým kódem v Javě a server Javy EE nakonfiguruje komponentu během nasazení a za běhu. V Javě EE lze také použít injekci závislosti neboli tzv. dependency injection, a to ve všech prostředcích potřebujících komponentu. Lze to využít v kontejnerech JavaBeans (EJB), webových kontejnerech a aplikačních klientech. (12)

2.2.1 Vícevrstevná aplikace

Aplikace tvořené v Javě EE jsou rozděleny do následujících vrstev.:



Obrázek 1– Vrstvy aplikace tvořené v Javě EE (12)

V klientské vrstvě běží buď normální aplikace, anebo webová aplikace tvořená dynamickými webovými stránkami, které jsou vykreslovány ve webovém prohlížeči. Ve webové vrstvě jsou komponentami buď tzv. servlety nebo webové stránky vytvořeny s pomocí technologie JavaServer Faces anebo s tzv. technologií JSP. Servlety jsou třídy Javovského programovacího jazyka, které dynamicky zpracovávají požadavky a vytváří odpovědi. JSP stránky jsou textové dokumenty, které se spouští, jako servlety, ale dovolují víc přirozený přístup k vytváření statického obsahu. Technologie JavaServer Faces je vytváří na servletech a JSP technologii a poskytují framework komponenty uživatelského rozhraní pro webové aplikace. (12)

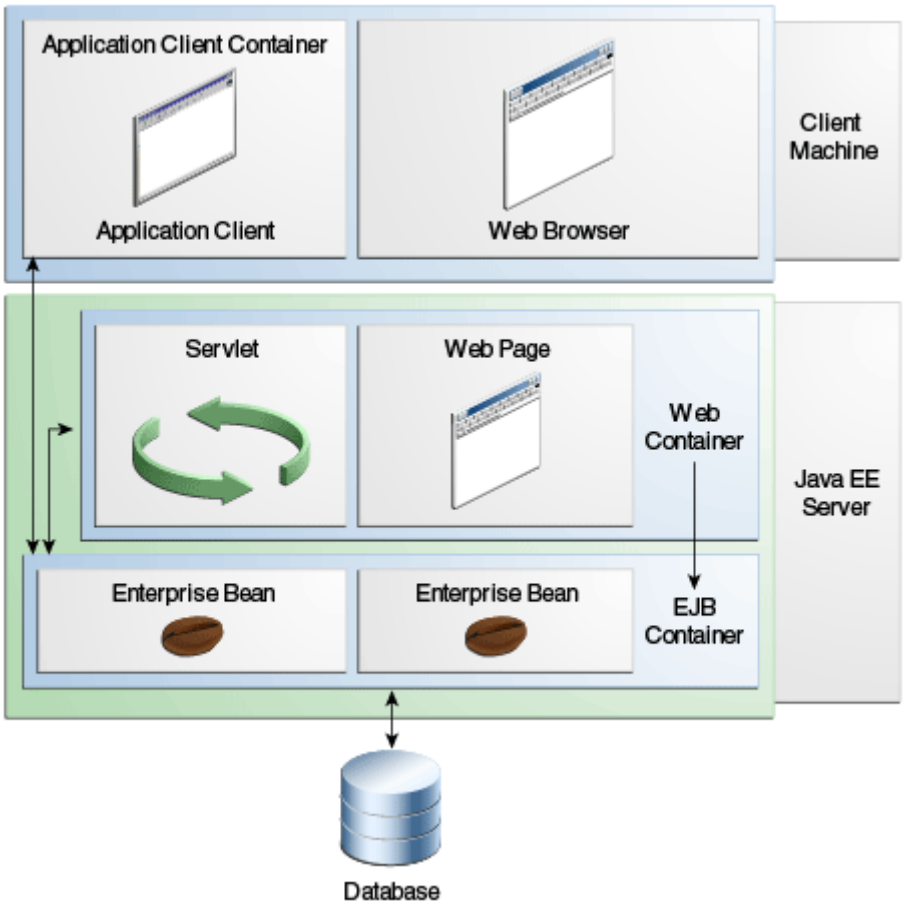
Pro zajímavost, statické HTML stránky a aplety jsou sice zahrnuty během složení aplikace, ale nejsou považovány, jako webové komponenty specifikace Javy EE. (12)

Obchodní vrstva je zpracovávána tzv. podnikovými fazolkami, které běží v této, nebo ve webové vrstvě. A ve vrstvě podnikového informačního systému neboli enterprise information systém (EIS) je například spravováno spojení s databází. (12)

2.2.2 Kontejnery

Kontejnery jsou rozhraním mezi komponentou a nízko úrovňovou, platformě specifickou, funkcionalitou, která podporuje komponentu. Před spuštěním musí být daná komponenta složena do modulu Javy EE a nasazena do jejího kontejneru. (12).

Jednotlivé kontejnery a jejich komponenty jsou ukázány v obrázku níže.:



Obrázek 2 – Kontejnery a jejich komponenty (12)

Java EE server je běhovou částí produktu. EJB kontejner spravuje spouštění podnikových fazolek pro aplikace Javy EE. Podnikové fazolky a jejich kontejner běží na serveru Javy EE. Dále webový kontejner spravuje spouštění webových stránek, servletů a některých EJB komponent pro aplikace Javy EE. Webové komponenty a jejich kontejner běží na Javě EE. Kontejner aplikačního klienta spravuje spuštění jeho komponent. Aplikační klienti a jejich kontejner běží na klientovi. (12)

2.2.3 XML

Extensible Markup Language neboli XML, je přechozí platforma, konkrétně rozšiřitelný, textový standard pro reprezentaci dat. (12)

Strany, které si vyměňují XML data si mohou vytvořit jejich vlastní tagy pro popis dat, nastavit schémata pro specifikaci, jaké tagy mohou být použity v daném druhu XML dokumentu, a použít stylové listy XML pro zprávu zobrazení a manipulace s daty. (12)

2.2.4 Technologie (API) v Javě EE

V následující části budou popsány některé technologie neboli API (Application Programming interface neboli rozhraní programování aplikace) (13) obsažené v Javě EE. Jednou z technologií je tzv. Enterprise JavaBeans Technology. Komponenta Enterprise JavaBeans (EJB), neboli podniková fazolka je tělo kódu, které má pole a metody pro implementaci modulů obchodní logiky. (12)

Podnikové fazolky mohou být fazolkami sezení, které reprezentují přechodnou konverzaci s klientem. Poté, co klient skončí danou operaci, fazolka sezení a její data jsou smazány. Podnikové fazolky mohou být také fazolky založené na zprávě. Ty kombinují prvky fazolky sezení a naslouchače zprávy, což umožňuje obchodní komponentě obdržet zprávy asynchronně. (12)

Další technologií je tzv. Java servlet, který umožňuje definovat třídy HTTP-specifického servletu. Servletová třída rozšiřuje možnosti serverů, ke kterým hostující aplikace přistupují způsobem programovacího modelu požadavku-odpovědi. (12)

Také existuje JavaServer Faces technologie, která slouží, jako framework uživatelského rozhraní webových aplikací. Dále JavaServerPages (JSP) je technologie, která umožňuje umístit útržky servletového kódu přímo do textového dokumentu. JSP stránka je textový dokument, obsahující statická data, ve formě například HTML nebo XML, a JSP elementy, definující dynamický obsah ve stránce. (12) K té lze například využít knihovnu tagů neboli JavaServer Pages Standard Tag library, která zabaluje základní funkcionality JSP aplikací do jedné sady tagů. (12)

Prací s databází se například zabývá technologie Java Persistence API (JPA), která se zabývá, jak už to vyplývá z názvu, persistencí. (12)

Persistence v tomto případě používá přístup
objektově-relačního mapování (ORM) k propojení objektově orientovaného modelu a relační databáze. Java Persistence API se skládá ze samotné technologie, dotazovacího jazyka a dat objektově relačního mapování. (12)

Další technologie pracující s databází je Java Transaction API (JTA), která poskytuje rozhraní pro vymezení transakcí. Architektura Javy EE při výchozím nastavení poskytuje automatický commit (potvrzení a uložení transakce) pro práci s transakcemi a rollbacky (zrušení transakce a jejích změn). Auto commit znamená, že jakákoliv aplikace, která v databázi přistoupí k datům, je uvidí aktualizovaná po každé operaci čtení nebo psaní. Lze ale v této technologii vymezit, kdy daná transakce, včetně jejích operací začne, potvrdí změny, nebo je zruší. (12)

Java Database Connectivity (JDBC) API je technologie umožňující volat databázové SQL příkazy přímo z metod v Javě. Technologie se používá v podnikovém semínku během přístupu k databázi semínkem sezení. Lze tuto technologii také použít ze servletu nebo JSP stránky k přímému přístupu k databázi bez podnikového semínka. Tato technologie se dělí na dvě části. První částí je rozhraní na aplikační úrovni, které je použito aplikačními komponentami k přístupu k databázi. A druhou částí je rozhraní poskytovatele služby k přiložení JDBC ovladače k platformě Javy EE. (12)

Dále Java EE obsahuje technologii Managed Beans Jsou to lehké, kontejnerově spravované objekty, s minimálními požadavky. Managed Beans podporuje malou sadu základních služeb, jako injekce prostředků, zpětné volání životního cyklu a přerušitele. Je to v podstatě generalizace zpravovaných fazolek specifikována technologií JavaServerFaces. (12)

Dále existuje tzv. Dependency Injection for Java neboli injekce závislosti pro Javu. Tato technologie definuje standartní sadu anotací a jednoho rozhraní pro použití ve vhodných třídách. V Javě EE tuto technologii implementuje technologie
Context and Dependency Injection for Java EE (CDI), která definuje sadu kontextuálních služeb poskytnutými kontejnery Javou EE, která zjednodušuje použití podnikových semínek s technologií JavaServer Face ve webových aplikacích. (12)

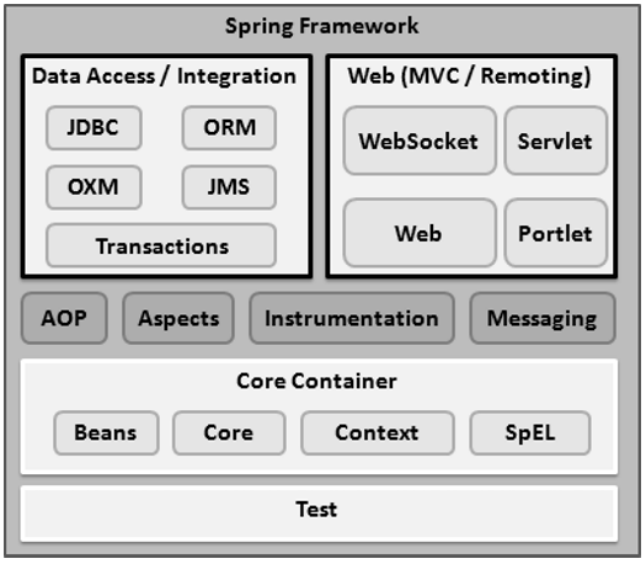
Java EE má také technologie zabývající se tzv. JSONem. JSON znamená JavaScript Object Notation neboli Notace JavaSkriptového objektu. Je to formát výměny textových dat, který je odvozen z JavaScriptu a je obvykle používán například ve webových službách. (12)

2.3 Uvedení vybraných technologií

V následující části budou uvedeny a popsány vybrané backendové technologie v jazyce Javě.

2.3.1 Spring framework

Spring je modulární framework, který je flexibilní a lze s ním vytvořit víc architektur, podle potřeby aplikace. Obrázek níže ukazuje jeho architekturu.:



Obrázek 3 – Architektura Springu (14)

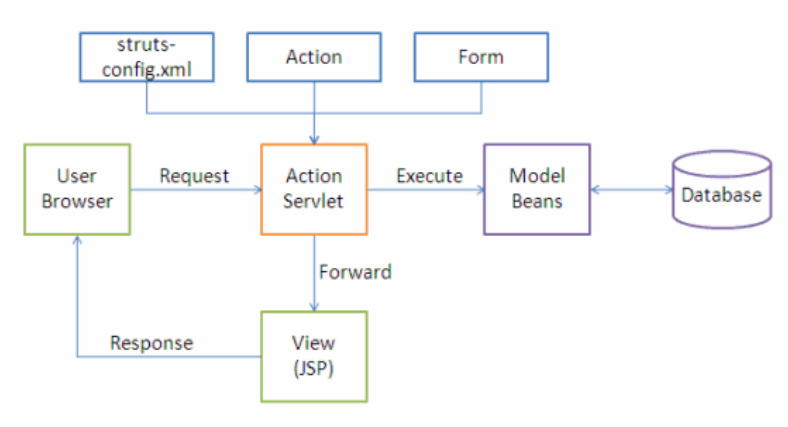
Jeho základním konceptem je tzv. Bean Factory neboli fazolová továrna, která poskytuje vypouštění tzv. Factory Pattern neboli továrního vzorce, který vytváří fazolky podle přání vývojáře, zadaného v souboru formátu „.xml.“ Vývojáři tak mohou například přidělit přerušovače metody podle aspektově orientovaného programování. Zajímavé je, že kompilace, která v Javě obvykle probíhá před spuštěním programu, je zde přeskočena a namísto toho probíhá za běhu. Což znamená, že je jednodušší ji udržovat, protože cílí na deklarativní transakční management. Modul Data Access/Integration (DAO), česky modul přístupu k datům nebo integrace dat, poskytuje nízko úrovňový úkol práce se spojením, jako jeho vytvářením, ukončením apod. Také si dále udržuje hodnocení nezbytných výjimek, namísto vyhazování chybových kódů přímo z databáze. Modul také využívá už zmíněné aspektově orientované programování ke zprávě transakcí. (1)

Spring sice neobsahuje svoji vlastní ORM aplikaci, ale nabízí různé kombinace se známými ORM nástroji, jako je například Hibernate, Oracle TopLink, IBATIS SQL apod. (1)

2.3.2 Struts framework

Struts framework rozšiřuje Java Servlet API a zabývá se architekturou Model, View, Controller (MVC). Pro zajímavost, tato architektura rozděluje práci s aplikací na model, ve kterém se pracuje s daty aplikace, obvykle se spojením s databází, dále Controller, ve kterém je backendová část aplikace a View, ve kterém se pracuje s frontendem. V tomto případě modelová část obsahuje JavaBeans, EJB, pohledová část obsahuje JSP soubory a ovladač obsahuje tzv. akce. Framework umožňuji vytvářet webové stránky založené na JSP stránkách, Java Beans a XML. (1)

Na obrázku níže je znázorněna architektura tohoto frameworku.:



Obrázek 4 – Architektura Struts frameworku (15)

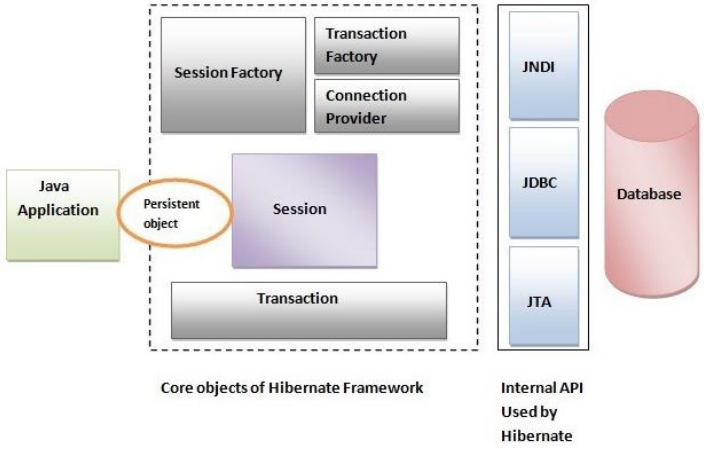
Struts framework funguje následovně.: Přijde mu tzv. http požadavek od webového prohlížeče, klienta. Požadavek přijme servlet akce. Akce obsluhující požadavky jsou uloženy v souboru „Struts-config.xml“. V tomto souboru jsou také zadána mapování akce a přesměrování akcí. Nejdříve si servlet akcí přečte soubor struts-config.xml a vybuduje si databázi kompozičních objektů. Akční servlet následně provede rozhodnutí odkazováním k danému objektu při jeho zpracovávání. Když servlet akcí obdrží požadavek, tak shromáždí všechny jeho hodnoty do třídy JavaBeanu. Ta se rozhodne, která akční třída vyhovuje procesu požadavku, schválí data poslaná uživatelem. Dále s pomocí modelové komponenty třída akcí zpracuje požadavek. (1)

Model dále zpracuje požadavek interakcí s databází. Akční třída vrátí přesměrování akce ovladači po dokončení zpracování požadavku. (1)

2.3.3 Hibernate framework

Hibernate je ORM framework, který lze využít při řešení komunikace aplikace s databázovým serverem. Mapuje Javovské třídy do databázových tabulek. To znamená, že Hibernate pracuje s transakcemi v databázích, které umí spustit a ukončit. Pro práci s transakcemi Hibernate používá JTA. (1)

Obrázek níže znázorňuje architekturu Hibernate frameworku.:



Obrázek 5 – Architektura frameworku Hibernate (1)

Hibernate se skládá z několik částí. Například z tzv. „Session Factory“, která se stará o data ve vyrovnávací paměti. Session factory má své rozhraní, prostřednictvím kterého získává objekt sezení (Session). Rozhraní v sobě obsahuje faktorní metodu. (1)

Dále Hibernate obsahuje tzv. Session object neboli objekt sezení, který poskytuje rozhraní mezi aplikací a daty uloženými v databázi. Vytváří transakci. Drží data vyrovnávací paměti první úrovně. V rámci sezení Hibernate dále obsahuje Session Interface neboli rozhraní sezení, které poskytuje metodu pro vklad, aktualizaci a odstranění objektu. Toto rozhraní také poskytuje faktorní metody pro transakci. (1)

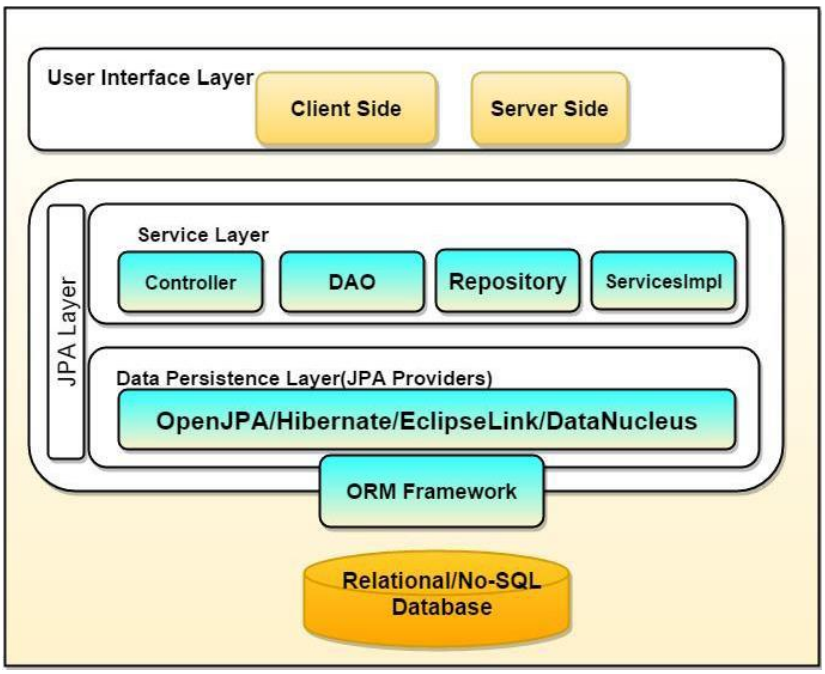
Další věci se týkají transakce. Zde Hibernate obsahuje objekt transakce, který specifikuje atomickou jednotku práce. Objekt transakce nemusí být při implementaci v aplikaci využit, jelikož je volitelný. Transakce ale kromě objektu také obsahuje své vlastní rozhraní neboli Transaction interface, které poskytuje metody pro zprávu transakce. Dále Hibernate obsahuje tzv. connection provider, který vytváří spojení s databází. Odděluje aplikaci od manažeru ovladače nebo datového zdroje. Je volitelný. Volitelným je také tzv. Transaction factory, které vytváří transakci. (1)

2.3.4 Java Persistence API (JPA)

Java Persistence API neboli ve zkratce JPA, je rozhraní, které ukládá entitu vytvořenou v Javě do relační databáze. JPA specifikace se skládá z prázdných metod a kolekce rozhraní která pouze popisuje metodologie persistence Javy a poskytuje standardizované programování skrze implementaci JPA. JPA má několik implementací od různých poskytovatelů JPA perzistence, a to jak komerčních, tak open source. JPA je například implementováno frameworky Hibernate od JBOSSe, EclipseLink od firmy Oracle, OpenJPA od IBM a Data Nucleus od JPOX a Tapestry. (2)

Pro zmapování map do databáze JPA provádí metadatové modelování a vytváření schématu, což je dosaženo buď skrze příslušné anotace, anebo xml soubory s příslušnými tagy. Typické API JPA definuje objekty běhového rozhraní pro zachování dat a pro vytvoření spojení. Pro vytvoření spojení mezi objekty Javy a databází, JPA definuje objekt rozhraní, EntityManagerFactory. Tento objekt provádí alokace a de-alokace prostředků. Jakmile je proces mapování hotový, tak je manažer zničen a prostředky jsou uvolněny. Objekt EntityManagerFactory je volán objektem rozhraní EntityManager. EntityManager pak provádí operace vytvoření, přečtení, aktualizace a smazání operací entit. V angličtině shrnuto zkratkou CRUD (create, read, update and delete). Proces persistence v JPA zpravuje SQL operace skrze transakce a dotazovací objekty pro získání a provádění operací jazyka definice dat (DDL – data definition language). (2)

Obrázek níže ukazuje běžnou architekturu JPA.:



Obrázek 6 Běžná architektura JPA (2)

Obrázek výše zobrazuje tři vrstvy vysokoúrovňové architektury. Nejvyšší vrstva představuje grafické uživatelské rozhraní, které komunikuje s klientem a serverem za účelem provedení operací na front-endu. Střední, JPA vrstva, je rozdělena na služební vrstvu, která používá kontrolery, DAO objekty, repositáře a implementaci služeb. Druhou částí JPA vrstvy je vrstva datové persistence, která definuje proceduru mapování mezi relační databází a entitami Javy. (2)

Nejnižší vrstva určuje typ relační databáze, aby zachovala data v tabulkovém formátu. (2)

2.3.5 MyBatis

MyBatis je open source, lehký a persistentní framework. Zaměřuje se na řešení komunikace aplikace s databází. Automatizuje mapování mezi SQL databázemi a objekty v Javě. Mapování jsou oddělena od aplikační logiky zabalením SQL kódu do konfiguračních souborů ve formátu XML. (16)

MyBatis je odvozen od JDBC kódu a také podporuje vlastní SQL, uložené procedury a pokročilé mapování. Jedním z hlavních rozdílů tohoto frameworku oproti ostatním frameworkům je důraz na použití dotazovacího jazyka SQL, oproti jiným podobným frameworkům (jako je například Hibernate), které používají svůj vlastní dotazovací jazyk. (16)

2.3.6 JDO – Java Data Objects

Java Data Objects (JDO) je framework, který slouží k přístupu k perzistentním datům v databázi s využitím tzv. plain old Java objects (POJO). Což odděluje manipulaci s daty od manipulace s databází, kde manipulace s daty se provádí prostřednictvím Javovských datových členů v doménových objektech Javy. A s databází se manipuluje prostřednictvím metod rozhraní frameworku JDO. (17)

JDO má například v uživatelově pohledu persistence uvedena tři rozhraní. A to PersistenceManager, který je zodpovědný za životní cyklus persistentních instancí, Query factory a přístup transakce. Dále rozhraní dotazu (Query), které je zodpovědné za tázání se databáze a návrat perzistentních instancí nebo hodnot. A poslední, rozhraní transakce, které je zodpovědné za zahájení a ukončení transakcí. (17)

2.3.7 Data Nucleus

DataNucleus je přístupová platforma, která umožňuje perzistenci a získání dat z databází, prostřednictvím implementace různých Api, a to konkrétně JDO, JPA a REST (Representational state transfer) (18). (19)

DataNucleus vyžaduje, aby všechny třídy spojené s perzistentními daty v databázi implementovaly tzv. Persistable rozhraní. Což znamená, že programátoři musí tyto třídy označit anotací „@persistable“. DataNucleus poskytuje instrumentor pojmenovaný jako „enhancer“, který transparentně přemění třídy anotované touto anotací do tříd s implementovaným zmíněným rozhraním. Poté, implementace API, které jsou vyžadovány rozhraním Persistable, jsou také automaticky vygenerované tímto enhancerem. Enhancer také vloží kontrolní pole (odpovídající datovým polím, která ukládají daná data) do Persistable objektů a nástrojové pomocné metody pro zjednodušení správy. (3)

Rozvržení dat v databázích typu RDBMS (Relational Database Management System neboli systém zprávy relačních databází) (20) není kompatibilní s tím v objektech Javy. Pro práci s daty DataNucleus musí provést transformační fázi pro překlad všech aktualizací dat do syntaxe SQL. Poté tyto syntaxe pošle do databáze prostřednictvím technologie JDBC. (3)

2.3.8 Java database connectivity (JDBC)

Java Database connectivity neboli JDBC, je API, které používají aplikace v Javě k interakci s databází. Základní funkcionalita této technologie je zprostředkována pěti rozhraními. A to rozhraním Driver, Connection, Statement, ResultSet a ResultSetMetadata. (4)

Jelikož každá databáze má svůj mechanismus pro navázání spojení s klientem a klient by měl být co nejméně závislý na serveru, tak každá databáze poskytuje svůj vlastní ovladač neboli driver, což je třída, kterou volá klient. Tato třída implementuje rozhraní Driver. Driver má metodu connect, kterou zavolá klient při připojování k databázi. Tato metoda přijímá dva argumenty, a to odkaz URL (Uniform Resource Locator) (21), který identifikuje ovladač, server a databázi. Toto URL je textový řetězec String. Druhým argumentem je objekt typu Properties, který umožňuje poslat nastavení vlastností spojení databázi. Pokud jsou ale tyto vlastnosti už součástí prvního argumentu, tak druhý argument není nutno zadávat, a může se nastavit jako null. (4)

Během interakce mezi klientem a databází může dojít k různým výjimkám. Například kvůli špatně zadanému SQL dotazu, nebo kvůli selhání připojování klienta k databázi. Pro tyto výjimky, které mohou být u různých databází zpracovávány různě, používá JDBC svoji vlastní výjimkovou třídu, SQLException. (4)

Prostřednictvím metody CreateStatement u objektu Connection lze získat objekt Statement. Tímto objektem lze provádět dotazy jazyka SQL. A to metodami executeQuery a executeUpdate. Objekt také obsahuje metodu close pro uvolnění prostředků objektu. Metoda executeQuery vrací výsledek ve formě objektu ResultSet. Z něj mohou být dále získána požadovaná data, která tento objekt získal z databáze. ResultSet obsahuje metodu next, která se posune na další řádek získaný z tabulky v databázi a vrátí pravdivý boolean, pokud se posunutí zdařilo a nepravdivý, pokud už další řádky dat výsledku nenásledují. (4)

Podstatné je, že každý nový objekt ResultSet se vždycky nachází před prvním řádkem dat, takže se musí zavolat metoda next před prací s prvním řádkem získaných dat. Při práci s daty se pak používají metody, jako getInt a getString, kterými se získávají proměnné z dat. (4)

K práci se schématem získaného výsledku slouží rozhraní ResultSetMetaData. Pro volání dotazu k databázi kromě Statementu lze také použít třídu PreparedStatement. Tato třída umožňuje vytvářet parametrizované dotazy, kde se na místo parametrů napíše otazník. Každý parametr má svoji indexovou pozici podle pořadí v dotazu. První parametr má index číslo 1. Metodami, jako například setInt a setString lze poté přiřadit hodnoty parametrům. Po vytvoření dotazu a přiřazení hodnot parametrům se poté daný dotaz provede. (4)

2.3.9 Apache Shiro

Apache Shiro je flexibilní open-source bezpečnostní framework, který obstarává autentizaci, autorizaci, management podnikového sezení a kryptografii. Framework je hodně aplikovaný díky jednoduchosti jeho použití a porozumění, zejména v síťových prostředích. Poskytuje čistou a intuitivní API, což zjednodušuje práci zabezpečení aplikací pro vývojáře. Autorizační modul tohoto frameworku je realizací standardního RBAC (role-based access control neboli kontrola přístupu založeného na rolích) modelu. (5)

Autorizace frameworku poskytuje jednoduché rozhraní ovládání přístupu, které v případě přidělených uživatelů a jejich rolí a povolení používá metodu hasRole pro určení, zda má uživatel danou roli a zda mu spíše udělit povolení než používat sadu uživatelsky aktivované role. Tento autorizační proces je jednoduchý, ale ne dostatečně bezpečný a flexibilní, protože se vzdává sezení modelu RBAC. (5)

2.3.10 Java authentication and authorization service (JAAS)

Java authentication and authorization service, ve zkratce JAAS, je nízko úrovňový bezpečnostní framework Javy SE, který rozšiřuje model zabezpečení že zabezpečený založeného na kódu na uživatelsky založené zabezpečení. JAAS lze použít k autentizaci, která identifikuje entitu, která používá daný kód, a poté autorizaci, která zajišťuje, aby entity měla požadovaná oprávnění nebo povolení přístupové kontroly k provedení určitého kódu. (22)

JAAS má tři základní rozhraní. CallbackHandler, které se používá k získání přihlašovacích údaj. Dále Configuration, která načítá implementace rozhraní LoginModule. A již zmíněné LoginModule, které je použito pro autentizaci uživatele. (22)

2.3.11 OACC

OACC je API, které se zabývá autentizací, autorizací a jejich správou. API obsahuje své vlastní implementace hesla ve formě přihlašovacích údajů. Případně lze implementovat AuthenticationProvider rozhraní. Framework poskytuje delegaci identity povolením autorizovaného subjektu „napodobit“ jiný subjekt. (23)

OACC pro svůj model zabezpečení používá plně implementované RDBMS datové úložiště, které následně zpravuje. K tomu obsahuje SQL skripty pro konfiguraci databáze. Bezpečnostní model OACC je založený na povoleních, která zpravuje mezi prostředky. Prostředky v API reprezentují jak zabezpečené objekty, tak jejich aktéry neboli uživatele. Povolení jsou založena na textovém řetězci Stringu. (23)

OACC také podporuje RBAC. Role a skupiny mohou být za tímto účelem modelovány skrze dědění povolení, což umožňuje hierarchické role, prostřednictvím kterých může subjekt získat přidělená povolení. (23)

3 Popis aplikace a jejích základních technologií

V následující části bude popsána aplikace, která byla rámci této bakalářské práce vyvíjena, technologie, které byly v rámci aplikace využity a struktura aplikace.

3.1 Popis technologií použitých během vývoje aplikace

Před popisem samotných backendových technologií nejdříve budou uvedeny technologie, které byly použity během vývoje aplikace.

3.1.1 Eclipse IDE

Aplikace byla vyvíjena v integrovaném vývojářském prostředí (v angličtině integrated development environment, zkráceně nazýváno, jako IDE) Eclipse. Eclipse obsahuje základní pracovní prostor a rozsáhlý systém rozšíření pro upravení vývojového prostředí. Sada vývoje softwaru Eclipse, v angličtině Eclipse software development kit (SDK), který obsahuje vývojářské nástroje Javy, je bezplatný a open-source. (6)

Během vývoje byl použit balíček Eclipse IDE for Enterprise Java Developers, který obsahuje platformu datových nástrojů, integraci Gitu pro Eclipse, vývojářské nástroje Eclipse Javy a Eclipse Javy EE, integraci Mavenu pro Eclipse, Mylyn seznam úkolů a vývojářské prostředí pro doplňky Eclipsu. (24)

3.1.2 Apache Tomcat

Apache Tomcat je open source implementace technologií Java Servletu, JavaServer Pages (JSP), Java Expression Language a Java WebSocket. (25) Aplikace je spouštěna na technologii Apache Tomcat, verzi 9, kde v době implementace nejnovější verzí byla 9.0.37.

3.1.3 Apache Maven

Apache Maven je populární správce balíčku a nástroj automatizace sestavování pro projekty Javy a další jazyky, které se kompilují ve virtuálním stroji Javy. Maven závisí na specifickém konfiguračním souboru ve formátu xml, který se nazývá, jako „POM“ („Project Object Model“, neboli model objektu projektu), který umožňuje úpravu všech životních cyklů sestavení. (7)

Tento soubor obsahuje informace o projektu a všech jeho konfiguračních detailech použitých Mavenem během jeho různých fázích sestavování. Stejně jako v objektově orientovaném programování, soubory POM mohou dědit od základního POM, tzv. rodičovský POM Mavenu. (7)

Maven obsahuje tzv. artefakty. Artefakty Mavenu jsou zkompilované projekty Mavenu, které byly nasazeny na repositář externího binárního kódu pro pozdější znovupoužití. V Mavenu jsou artefakty typicky zabalené, jako soubory ve formátu JAR, které obsahují bytecode Javy. Každý artefakt je unikátně identifikovaný tzv. trojicí údajů, a to údajem groupId, který identifikuje organizaci, která vytvořila tento artefakt, dále artifactId, který identifikuje jméno knihovny a version, který navazuje na verzovací schéma Mavenu. V současnosti nejpopulárnější veřejný repositář, který je hostitelem artefaktů Mavenu, je centrální repositář Mavenu neboli Maven Central repository. Artefakty v Mavenu jsou tzv. immutable neboli neměnné, což znamená, že jakmile jsou nasazeny, tak nesmí být upraveny. Což znamená, že všechny verze dané knihovny zůstanou trvale dostupné v repositáři. (7)

Jedním ze základních prvků Mavenu je jeho mechanismus tzv. dependency resolution, který určí sadu závislostí nutných k vybudování jednotlivého artefaktu, a pak získá závislosti, které nejsou dostupné lokálně, z externích repositářů, jako Maven Central. Maven si vytvoří strom závislostí, který reprezentuje vztahy závislostí mezi POM soubory vyřešených závislostí. Závislosti v Mavenu mohou být přímé, zděděné a tranzitivní. Přímé závislosti jsou deklarovány přímo v souboru POM daného projektu. Zděděné závislosti jsou deklarované v rodičovském POM souboru. Transitivní závislosti jsou získané z tranzitivního uzavření přímých a nepřímých závislostí. (7)

3.1.4 Databáze MySQL

Aplikace využívá relační databázový server MySQL. MySQL podporuje databázový jazyk SQL (Structured Query Language). Podle Dataconomy, poslední verze tohoto serveru je jednou z nejpopulárnějších databází, protože je to databáze typu open source s kompatibilitou se všemi hlavními hostitelskými poskytovateli. Je to také cenově efektivní databáze, kterou je také jednoduché spravovat. MySQL má ale nevýhody ve škálování, například dlouhý čas vývoje a ceny logování databáze. (8)

3.2 Popis aplikace

Aplikace, která byla v rámci implementací backendových technologií vyvíjena, se jmenuje Ztráty a nálezy, její projektové jméno je „ZtratyANalezy“. Je to webová aplikace zaměřující se na hlášení a zprávu seznamu ztrát a nálezů neboli předmětů, jaké mohl někdo ztratit v budově firmy, nejspíše zákazník firmy, anebo její zaměstnanec. Aplikaci mohou používat pouze zaregistrovaní uživatelé. V aplikaci jsou uživatelské role rozdělené na tzv. „ROLE_ADMIN“, neboli administrátora a „ROLE_USER“, zaregistrovaný zákazník. V aplikaci si může uživatel vytvořit účet zákazníka formou registrací.

Všichni uživatelé si mohou upravovat údaje svého účtu. Dále se všichni uživatelé mohou hlásit k nálezům, které patří jim, odhlásit se od nich, pokud se k nim přihlásili omylem, anebo uložit do databáze ztrátu neboli nahlásit předmět, který v budově firmy ztratili. A samozřejmě si mohou zobrazit své předměty, jaké vložili do databáze. A upravit si vlastní ztráty.

Administrátoři mohou navíc vytvářet nálezy neboli vkládat do databáze věci zákazníků, co se našly v budově firmy. Mohou je také upravovat. Také mohou přeměnit ztrátu na nález, pokud se našla, a tím pádem označit ztracená předmět za nalezený. A nakonec mohou administrátoři přidělit nalezenou ztrátu neboli předmět, který byl vytvořen, jako ztráta, ale později kvůli jeho nálezu převeden na nález uživateli, který nalezenou ztrátu původně vytvořil. Anebo předmět, který byl vytvořen, jako nález, mohou přidělit uživateli, co se hlásí k vlastnictví danému předmětu. Což znamená převedení předmětu na předmět odevzdaný majiteli. Pro přidělování předmětů jejich vlastníkům si může administrátor zobrazit uživatele hlásící se k předmětům a nalezené ztráty.

Administrátoři si mohou kromě vlastních předmětů také zobrazit předměty všech uživatelů, ze všech kategorií, anebo ze zvolené kategorie. A zákazníci si mohou zobrazit všechny nálezy, které byly vytvořeny v databázi. U vyhledávání předmětů si uživatelé mohou vyhledávat také předmět podle jeho jména a pokud na to mají oprávnění, tak také podle její kategorie.

3.3 Struktura aplikace

Jak už je zřejmé z předchozích částí, tak tato aplikace je vyvíjena v programovacím jazyce Java. Pří vývoji byla, jako základní jazyk, použita Java SE 14 a v jednom případě Java SE 13, kvůli technologii, která Javu SE 14 nepodporovala.

Jelikož aplikace využívá Maven, tak projekt obsahuje soubor „pom.xml“, kde jsou nastaveny údaje projektu, jeho vlastnosti a závislosti. Co se týče formátu případné kompilace projektu, tak oproti normálním aplikacím, které se v případě Javy kompilují ve formátu „.JAR“, tak tato aplikace bude kompilována ve formátu „WAR“, neboli tzv. Web Archive (Webový archiv), což je koncovka souboru, která zabalí hierarchii webové aplikace ve formátu zip. (26) Webové aplikace Javy se obvykle ve formátu WAR kompilují pro nasazení. (26) Z vlastností, v tagu „properties“, jsou nastaveny verze Javy, jakou projekt využívá a verzi Javy, ve které má Maven kompilovat projekt, tagy „maven.compiler.source“ a „maven.compiler.target“.

Snímek níže zobrazuje základní nastavení Mavenu u jedné verze projektu.:

```
<?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>cz.uhk</groupId>
    <artifactId>ZtratyANalezyStruts</artifactId>
    <version>1.0.0</version>
    <packaging>war</packaging>
    <name>ZtratyANalezyStruts</name>
    <description>Systém evidence ztrát a nálezů v budově firmy</description>

    <properties>
        <java.version>13</java.version>
        <maven.compiler.source>13</maven.compiler.source>
        <maven.compiler.target>13</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
    </properties>
```

Obrázek 7 Základní nastavení projektu v Mavenu

Ve složce projektu „src/main“ se dále aplikace dělí na tři části: složku resources, která obsahuje konfigurace implementovaných technologií, které to vyžadují. Dále složku webapp, kde se nachází část pohledu aplikace, a to ve formě složek využívaných technologií.

Hlavní složkou, která se ve složce webapp nachází, je „WEB-INF“, která kromě případných konfigurací webové aplikace a některých implementovaných technologií obsahuje složku .jsp, kde se nachází .jsp soubory pohledu a dále složku. tags, která obsahuje soubor ve formátu .tag, což je soubor, který v pohledu slouží, jako základní layout, který sdílí soubory v “jsp“ složce.

Třetí část aplikace je ve složce „src/main/java“, kde se nachází soubory backendu, a to Javy. V jednotlivých verzích aplikace se mohou balíčky lišit, ale balíčky, které obvykle obsahují všechny projekty společně jsou „cz.uhk.actions“ v případě implementovaného MVC frameworku Struts 2, anebo „cz.uk.controller“, které obsahují třídy sloužící, jako kontrolery aplikace. Dále balíček „cz.uhk.dao“, který obsahuje třídy umožňující komunikaci aplikace s databází. A nakonec balíček „cz.uhk.model“, který obsahuje třídy modelových dat aplikace.

V případě ukázání rozdělení kontrolerů na projektu ZtratyANalezyStruts, tak v balíčku cz.uhk.actions jsou následující třídy.:

- IndexAction – obsahuje akce prováděné indexem webové aplikace, což znamená zobrazení předmětů uživateli a detailní zobrazení jednoho předmětu.

- LoginAction – obstarává přihlášení uživatele a jeho registraci.

- PictureAction – Obstarává stažení obrázku předmětu. Využívána detailním zobrazením předmětu v Indexu.

- ReportedItemOwnershipAction – Obstarává zobrazení a práci s nalezenými ztrátami a nálezů, k jejichž vlastnictví se hlásí uživatelé.

- UserAction – Obstarává zobrazení a úpravu informací účtu přihlášeného uživatele.

- ZtrataAction – Obstarává práci se samotnými předměty, a to jejich přidání, úpravu a smazání.

Balíček cz.uhk.dao obsahuje třídu DaoBase, která je abstraktní, což znamená, že nemůže být instancována a nemusí implementovat metody rozhraní, které implementuje. Tato třída obsahuje společné metody pro všechny ostatní dao třídy.

Například vytvoření nového objektu v databázi, získání všech objektů z databáze, úprava a smazání objektů v databázi. Všechny tyto metody jsou implementovány z rozhraní IDaoBase, které slouží k definici těchto metod a jejich popisu ve formě dokumentace.

Třída DaoBase také obvykle obsahuje instanci spojení s databází, která je definováno, jako statická proměnná, což znamená, že je stejná ve všech instancích tříd, které dědí z třídy DaoBase. Což slouží k optimalizaci aplikace, aby se spojení nemuselo otvírat pro každou instanci dao tříd zvlášť. Třída DaoBase také obsahuje konstruktor, ve kterém volá metody pro vytvoření instance spojení s databází.

Pro jednoduchou možnost využití třídy DaoBase pro ostatní dao třídy je třída DaoBase a rozhraní IDaoBase generické, a to parametrizované třídou, jejíž výchozí název je „T“. Tato třída je implementována u dědících dao tříd a jako parametr se dosazuje modelová třída, ke které daná dao třída patří.

Aplikace se připojená ke schématu „ztraty_a_nalezy“ v databází MySQL. Při konfigurace databáze se vyskytl problém s časovou zónou. Jelikož při výchozím nastavení databáze využívá systémovou časovou zónu, kterou ale v případě Windows 10 nepodporuje, tak pokus aplikace o spojení s databází selhal s výjimkou o chybném nastavení časové zóny. V souboru nastavení instance databáze musel být povolen parametr „default-time-zone“, s nastavením času UTC (Coordinated Universal time) (27) „+02:00“.

Každé tabulce v databázi připadá jedna modelová třída v aplikaci a každé modelové třídě v aplikaci připadá jedna dao třída, která dědí ze třídy DaoBase a obsahuje tak základní metody pro práci s danou instancí v databázi a případně své vlastní metody, pokud jsou u dané třídy potřebné.

4 Implementace vybraných technologií a jejich porovnání

V následující části bakalářské práce bude popsána implementace vybraných technologií a jejích následné srovnání. Technologie budou rozděleny na čtyři části: MVC frameworky, databázové technologie a technologie zabezpečení, převážně zajišťující autentizaci a autorizaci.

4.1 MVC frameworky

Nyní budou popsány implementace MVC frameworků.

4.1.1 Spring Web

Jako první MVC framework byl implementován Spring Web ze skupiny frameworků Spring, v projektu ZtratyANalezy. Spring Web byl implementován s využitím technologie Spring Boot, která usnadňuje konfiguraci Springu a v některých případech ji i automatizuje. (28) V Mavenu byl implementován rodič skupiny „org.springframework.boot“, s identifikací artefaktu „spring-boot-starter-parent“, verze „2.3.0.RELEASE“. Tento rodič poskytuje výchozí konfigurace pro aplikaci a také kompletní strom závislostí pro projekt vyvíjený ve Spring Bootu. Tento rodič například u závislostí Springu konfiguruje jejich verzi, takže není nutné je zvlášť konfigurovat u jednotlivých závislostí zvlášť.

Dále jsou prostřednictvím Mavenu implementovány závislosti skupiny „org.springframework.boot“, artefakty „spring-boot-starter-data-jpa“, „spring-boot-starter-security“, „spring-boot-starter-web“, „spring-boot-starter-tomcat“, „spring-boot-starter-test“.

Základní konfigurace Springu je v souboru „application.properties“, který se nachází ve složce projektu „src/main/resources“. Zde jsou nakonfigurovány základní vlastnosti, jako jméno aplikace, cesta, kde se nachází soubory pohledu a koncovka souborů pohledů (.jsp).

Spring podporuje své propojení s ORM frameworky, jako je například Hibernate. Toto propojení bylo experimentálně také nakonfigurováno ve vlastnostech Springu, ale nakonec byl Hibernate implementován zvlášť, nezávisle na Springu, pro zjednodušení úpravy aplikace v různých verzích při implementacích jiných technologií. Proto toto propojení nebylo plně implementováno.

Snímek níže ukazuje nastavení Springu.:

```
spring.application.name=ZtratyANalezy
spring.mvc.view.prefix=/WEB-INF/jsp/
spring.mvc.view.suffix=.jsp
spring.datasource.url=jdbc:mysql://localhost:3306/ztraty_a_nalezy
spring.datasource.username=root
spring.datasource.password=123
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=create-drop
spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect
logging.level.web=debug
```

Obrázek 8 Nastavení vlastností Springu

Poznámka: Jak lze na snímku vidět, tak přihlašovací údaje k databázi jsou ve formě jména, jako root, což je podobně, jako v operačním systému Linux, uživatel se všemi oprávněními a heslo „123“. Heslo „123“ bude v této práci zmíněno vícekrát. Toto heslo bylo použito především pro zjednodušení ukázek funkcí technologie, ale v praxi se samozřejmě využití takového hesla silně nedoporučuje, jelikož by mělo být složitější, delší a s více znaky.

Co se týče struktury části aplikace v Javě, tak pro Spring je zde vytvořen balíček „cz.uhk.webapp“ a v něm třídy „ServletInitializer“ a „ZtratyANalezyApplication“, které obstarávají spuštění aplikace. Na třídu „ZtratyANalezyApplication“ je také nutné umístit anotace, a to hlavně „@ComponentScan“, která řekne Springu, v jakých balíčcích má Spring hledat své komponenty, jako například konfigurační soubory a kontrolery. V případě této aplikace je to konkrétně zmíněný balíček „cz.uhk.controler“, a také balíček „cz.uhk.config“, který obsahuje konfiguraci frameworku Spring Security, kterému se bude práce věnovat zvlášť). A také zde musí být umístěna anotace „SpringBootApplication“.

Kontrolery jsou implementovány v balíčku cz.uhk.controller. Na úrovni třídy se musí nacházet anotace „Controler“, která tak danou třídu Springu označí. Jednotlivé metody potom, kde každá metoda představuje jednu akci, musí vracet buď String s názvem souboru pohledu, anebo může také vracet ModelAndView, což je třída Springu, která umí pracovat s modelovými daty a také při vytváření jí lze nastavit název pohledu, který pak má uživateli vrátit.

Metodou „addObject(název atributu, hodnota atributu)“ lze uložit modelová data, ke kterým potom lze přistupovat přímo z pohledu (například prostřednictvím JSP tagů). Každý atribut lze také pojmenovat určitým Stringem. Což je vhodné, pokud autor aplikace nechce, aby atribut měl stejný název, jako proměnná a také v některých případech je to nutné i k následné funkčnosti přístupu k atributu v pohledu.

Každá metoda akce je v aplikaci označena atributem „@GetMapping(cesta)“, který očekává požadavek ve formátu „GET“, což znamená, že veškerá příchozí data jsou viditelná v URL. Tato anotace očekává atribut „path“, neboli URL, jakým se tato metoda akce zavolá. Například v případě metody index je to cesta „/“.

V případě nutnosti použití metody požadavku „POST“, například v případě přijetí citlivých dat od uživatele z formuláře pohledu, lze metodu akce také označit anotací „@RequestMapping(value, method)“, která přijímá cestu URL, jako hodnotu, a enumerací „RequestMethod“ lze nastavit metodu požadavku na POST, anebo na GET.

Každá akce může přijímat parametry, a to jednak parametry od uživatele z odkazů a formulářů, a také parametry servletu a Springu, jako například HttpServletResponse pro nastavení typu odpovědi na dotaz (to konkrétně využívá PictureController k vracení obrázku uživateli namísto pohledu) a také rozhraní Springu, jako například „RedirectAttributes“, které pracuje s atributy, které se zachovají při přesměrování na jinou akci.

Parametry očekávané od uživatele je vhodné označit anotací „@RequestParam“, kde lze nastavit, zda je parametr povinný pro danou akci, jeho výchozí hodnotu v případě, že nebude obdržen v požadavku (což je nutné použít v akcích s atributy, které se neočekávají, jako například v akci index atributy označující název předmětu a identifikační číslo jeho kategorie).

Dále jelikož aplikace v modelových třídách využívá validaci balíčku „javax.validation“, tak pokud metoda akce očekává, jako atribut modelovou třídu, tak je tato třída označena anotací „@Valid“, kterou se řekne Springu, aby provedl na této modulové třídě validace podle validačních anotací jednotlivých proměnných třídy. Pro validaci je nutné, jako další parametr metody akce přidat rozhraní „BindingResult“ a do metody následně ověření s využitím metody tohoto rozhraní „.hasErrors“, která ve forme booleanu vrátí výsledek, zda instance validované třídy obsahuje chyby, nebo ne. V případě výsledku booleanu „true“ je v rámci této aplikace uživatel vrácen zpátky do pohledu s formulářem dané akce, aby si mohl opravit chyby, které během vyplňování udělal.

V případě použití anotace „@Valid“ u parametru potom není vhodné u daného parametru používat anotaci „@RequestParam“. Jelikož použití anotací od různých technologií může způsobit nefunkčnost aplikace, anebo vyhodit vyjímku, jako se to stalo při implementaci Springu.

Přesměrování se ve Springu provádí prostřednictvím vrácení URL ve formě „redirect:/[odkaz]“, například vrácením Stringu „redirect:/login“ se přesměruje uživatel na akci očekávající volání URL „/login“, což je v případě aplikace metoda akce „index“ třídy „LoginController“. Pro zachování parametrů, například zprávy, která se uživateli později zobrazí, o úspěchu nebo neúspěchu akce, se využívá rozhraní „RedirectAttributes“, s metodou „addFlashAttribute(název atributu, hodnota atributu)“, prostřednictvím které se uloží do paměti daný atribut, který v ní zůstane zachován i po přesměrování, v další akci.

Rozhraní například Springu a HttpServletu není nutné zvlášť inicializovat. Při jejich použití je stačí umístit do metody akce, jako parametr a Spring se automaticky postará o zbytek.

4.1.2 Apache Struts

Na implementaci frameworku Apache Struts se zaměřil projekt ZtratyANalezyStruts. Prostřednictvím Mavenu byly staženy závislosti ze skupiny „org.apache.struts“, konkrétně artefakt „struts2-core“, který obsahuje samotný Apache Struts. Dále „struts2-convention-plugin“, který obsahuje anotace Strutsu 2 a jejich podporu v aplikaci. A „struts2-bean-validation-plugin“, který přidává podporu Strutsu 2 pro bean validation modulů.

Struts 2 musel být nakonfigurován v souboru „web.xml“, ve složce „src/main/webapp/WEB-INF/“. A to konkrétně jeho filtr. Snímek níže ukazuje jeho konfiguraci.:

```
<filter>
    <filter-name>struts2</filter-name>
    <filter-class>org.apache.struts2.dispatcher.filter.StrutsPrepareAndExecuteFilter
    </filter-class>
</filter>
<filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
    <dispatcher>FORWARD</dispatcher>
    <dispatcher>REQUEST</dispatcher>
    <dispatcher>INCLUDE</dispatcher>
    <dispatcher>ERROR</dispatcher>
</filter-mapping>
```

Obrázek 9 Konfigurace filtru Strutsu 2

I když byl Struts 2 převážně implementován s využitím anotací, tak základní konfigurace, která byla nezbytná, se nachází v souboru „struts.xml“, který se nachází ve složce „src/main/resources“. Zde jsou nastaveny věci, jako vývojářský režim, který v případě vyjímky na straně klienta zobrazí podrobnější popis vyjímky (mimo vývoj, ve verzi aplikace, která je následně nasazena, je vhodně tento vývojářský režim zakázat). Dále je zde nastavena bean validation, kterou poskytuje Hibernate. A vlastní balíčky tzv. interceptorů neboli zachycovačů pro validaci.

Kontrolery, nazývány ve Struts 2, jako akce, se nachází v balíčku cz.uhk.actions. Každá třída musí dědit ze třídy ActionSupport a mít anotací „@ResultPath“ na úrovni třídy nastavenou složku, kde se nachází pohledy, což je v tomto případě cesta „/WEB-INF/jsp“.

Oproti Springu, Struts 2 nemůže přímo přijímat data od uživatele do svých metod akcí, jako parametry. Proto je nutné veškerá očekávaná data definovat, jako proměnné třídy akce. Dále jim vytvořit metodu Get, pokud chce autor aplikace umožnit, aby k dané proměnné mohl přistupovat pohled a metodu „Set“, pokud autor chce, aby danou proměnnou mohl poslat uživatel aplikaci, jako část URL odkazu, anebo, jako data z formuláře. S proměnnými se pak dále pracuje v jednotlivých metodách akce. Struts 2 prázdné proměnné automaticky nastaví na hodnotu null.

Každá metoda akce obvykle vrací String a musí být označená anotací „@Action“, s parametry „value“, kterým se nastaví, jaké URL volání má daná akce očekávat, dále jaké má akce využívat zachycovače a jaký má provést výsledek.

Zachycovače jsou jedním z prvků frameworku Struts 2, které provádí při zavolání metody akce své operace, předtím, než je provedena samotná metoda akce. Pokud není žádný zachycovač nastaven, tak Struts 2 použije výchozí tzv. zásobník zachycovačů, který se jmenuje „defaultStack“. Tento zásobník obsahuje zachycovače, jako je například „fileUpload“, který zpracuje a pošle přijatý soubor z formuláře dané akci, aby s ním mohla dále pracovat, params, který je nezbytný pro přijetí parametrů pro danou akci od uživatele, a validation, která se stará o validaci v rámci frameworku Struts 2. Zachycovač Validation ale nepodporuje validaci beanu. (29)

Aplikace ve většině případů používá výchozí zásobník zachycovačů, který stačí pro implementaci jejích akcí. V případě některých zachycovačů z výchozího zásobníku je také nutné implementovat jeho rozhraní, jako v případě modelově založeného zachycovače, kde je nutné, aby třída dané akce implementovala rozhraní ModelDriven<>, které je generické a jako parametr přijímá modelovou třídu, se kterou metody akce následně pracují. Tato modelová třída musí být následně deklarována, jako proměnná, hned s novou vytvořenou instancí (například v případě třídy ZtrataAction, kde modelová třída je Ztrata, tak deklarace její instance je ve formě „private Ztrata ztrata=new Ztrata();“. Tato proměnná musí samozřejmě také mít vlastní metodu get a set, ale navíc je zde metoda getModel(), která je implementována v rámci rozhraní ModelDriven<>, a ta vrací instanci modelové třídy, kterou přijalo toto generické rozhraní, jako parametr.

Například v případě modelové třídy Ztrata potom implementace tohoto rozhraní vypadá následovně.:

```
    @Override
    public Ztrata getModel() {
        return ztrata;
    }
}
```

Obrázek 10 Implementace metody getModel() ve Struts 2

Dále v případě zachycovače validace bylo nutné, aby třída využívající daný zachycovač implementovala rozhraní ValidationAware. Pro podporu tzv. bean validation byla ale také nutná konfigurace v souboru struts.xml, a to nastavení třídy poskytující daný validátor, což je v případě této aplikace validátor org.hibernate.validator.HibernateValidator, a také implementace zachycovače Strutsu 2 beanValidation. Jelikož tento zachycovač není ve výchozím zásobníku nastaven, tak musel být nastaven ručně. K tomu byly nastaveny vlastní zásobníky zachycovačů, které byly vytvořeny v novém balíčku „beanValidationPackage“, který dědí z konfiguračního souboru „struts-default.xml“, což znamená, že v sobě tento balíček také obsahuje výchozí zachycovače a jejích zásobníky, jaké Struts 2 obsahuje.

V tomto balíčku byly vytvořeny dva zásobníky zachycovačů, a to „beanValidationStack“, který volá zásobník modelDrivenStack, který v sobě obsahuje modelDriven interceptor pro zpracování modelu a zásobník basicStack, který v sobě obsahuje, podobně, jako default-stack, zachycovače zpracovávající základní věci, včetně parametrů v požadavku. (29) Podstatné je, že oproti zásobníku „defaultStack“, „basicStack“ neobsahuje v sobě zásobník validace a workflow. (29) Po zásobníku „modelDrivenStack“ je následně v zásobníku „beanValidationStack“ zavolán samotný zachycovač „beanValidation“, který provede validaci modelových dat. A pote zásobník „workflow“, který v případě nalezených chyb předchozím zachycovačem namísto provedení metody akce zavolá výsledek „INPUT“, v rámci, kterého je uživatel vrácen zpátky na webovou stránku formuláře se zobrazenými chybami.

Druhý zásobník zachycovačů, který byl vytvořen, je „beanValidationAndPictureUploadStack“, který volá zachycovače „modelDriven“, „fileUpload“ pro zpracování přijatého souboru, kde je nastaveno, že smí přijímat pouze soubory ve formátu „.jpg“, „.png“. Poté následuje volání zásobníku „basicStack“, a přímo zachycovačů „beanValidation“ a „workflow“.

Tyto vlastní zásobníky zachycovače výše byly použity například ve třídě „ZtrataAction“ a „UserAction“. Dále metoda akce musí mýt v anotaci „@Action“ nastavený výsledek. Do výsledku se nastavuje jeho typ, jméno a lokace. V aplikaci byl použit typ „dispatcher“, který vrací pohled a jako jeho lokace byl nastaven soubor pohledu, jako například v případě metody upravitPredmet() třídy ZtrataAction, byl, jako lokace, nastaven soubor „upravitPredmet.jsp“. Dále byl v aplikaci, například u metody upraveniPredmetu() stejné třídy, použit výsledek typu „redirect“, který po dokončení akce ji přesměruje na jinou akci.

Jako lokaci se v tomto případě nastavuje URL volání akce, tedy v případě zmíněné metody je to například indexové „/“. Také v případě poslání proměnných dané akci je nutné použít parametr „params“ a zde deklarovat proměnné, které se pošlou další akci. Parametry se píšou ve dvojici, a to název proměnné a její hodnota, kde se daná hodnota, podobně jako v .jsp pohledu, zadává prostřednictvím dolaru. Například v případě zmíněného příkladu, kde se, jako parametr nastavuje proměnná Stringu success, která oznamuje úspěch provedení dané akce, tak v tomto případě se nastaví následovně.: „params= {„success", "${success}“}“, kde první „success“ je název proměnné a druhý znamená její hodnotu.

Podstatnou částí výsledku je ale jeho jméno. Protože každá metoda akce vrací textový řetězec String. Struts 2 podle tohoto řetězce vyhledá výsledek s odpovídajícím jménem a ten následně provede. Struts 2 obsahuje čtyři výchozí názvy výsledků, a to „SUCCESS“ pro úspěšné provedení metody, „INPUT“, v případě chybě ve validaci dat, které vyplnil uživatel ve formuláři a obvykle se tento název používá v případě výsledku, kdy se uživatel vrací zpátky do stejného formuláře, aby si mohl své chyby opravit.

Dále „ERROR“, „LOGIN“, který se obvykle používá k přesměrování uživatele na přihlašovací stránku, pokud není přihlášen, a „NONE“, který lze použít v případě, že uživatel bude přesměrován na jinou akci. Tyto výchozí názvy může programátor implementovat po svém, například název výsledku „SUCCESS“ byl použit jak pro vrácení pohledu, tak pro přesměrování uživatele. Lze také použít vlastní názvy výsledků, jako například „error2“ v případě metody „reportOfOwnership“, třídy ReportedItemOwnershipAction.

Ve třídě „PictureAction“, kde metoda „getPicture()“ namísto pohledu nebo přesměrování vrací obrázek pohledu detailního zobrazení předmětu, tak zde je také využit výsledek typu „stream“, který vrací proud dat neboli tzv. stream a očekává parametry nastavující typ jeho obsahu, název jeho vstupu a samotný stream. Tento typ výsledku přijímá ale stream typu InputStream.

4.1.3 Porovnání MVC frameworků

Oba frameworky mají svoji vlastní konfiguraci, kterou je nutno provést proto, aby fungovaly. Jak ve Springu, tak ve Struts 2 lze většinu věcí nastavit s pomocí anotací, což ulehčuje práci, jelikož většinu věcí lze provést přímo ve třídách kontrolérů bez nutnosti se zdržovat s prací s jinými soubory.

Hlavní rozdíl mezi těmito frameworky však je v implementacích parametrů metod akcí a rozhraní například servletu nebo daného frameworku. Zatímco ve Springu lze parametry a rozhraní nastavit přímo, jako parametry metody a Spring se postará o zbytek, ve Struts 2 je nutné tyto parametry deklarovat zvlášť, jako proměnné, jejich get a set metody a implementace rozhraní a funkcí, jako například Bean validation, vyžaduje implementaci určitých zachycovačů a rozhraní a jejích metod. Což zbytečně komplikuje práci a dělá i samotný kód méně přehledným.

Výhoda Struts 2 je hlavně v jeho zachycovačích, které umí provádět své operace ještě před samotnou akcí. Další jeho výhodou je implementace výsledků, kde stačí definovat například jeden výsledek pro úspěch nebo chybu a pak lze vracet pouze jeho název, bez toho, aby se u každého výsledku muselo zvlášť deklarovat, jaký pohled nebo jaké přesměrování má vracet.

Spring má přehlednější posílání proměnných pohledu, nebo jiné akci v rámci přesměrování, protože proto vše má rozhraní s metodami, kam se vkládá přímo daná proměnná. Což opět dělá jednak přehlednější kód, je jasně vidět jaké proměnné se posílají pohledu, a také v případě přesměrování se nemusí pracně deklarovat proměnná, kterou chceme poslat jiné akci, jako nějaký String v dolarových závorkách, ale namísto toho se tam přímo zadá ta proměnná, což zajišťuje i větší přesnost v zadání názvu proměnné, jelikož to kompilátor zkontroluje, zda je zadána proměnná správně, kdežto zadanou hodnota existující proměnné ve formě Stringu v závorkách kompilátor nezkontroluje, a tudíž je větší šance na výskyt chyb, například v rámci překlepu.

Na závěr, s těchto dvou frameworků by definitivně vybrán pro tvorbu aplikace byl vybrán Spring, jelikož je přehlednější, jednodušší na implementaci, a hlavně také obsahuje další technologie své značky, jako například Spring Security, které jsou s ním kompatibilní a frameworku Spring Security se bude tato práce ještě věnovat.

4.2 Databázové technologie

Následující část práce se bude zabývat technologiemi zabývajícími se komunikací aplikace s databází.

4.2.1 Hibernate

Implementace frameworku Hibernate bude popsána na projektu ZtratyANalezy. Do projektu se nejdříve musel framework stáhnout prostřednictvím Mavenu, ze skupiny „org.hibernate“, artefact „hibernate-core“.

Hibernate je nakonfigurován v souboru „hibernate.cfg.xml“, který se nachází ve složce „src/main/resources“. Zde je nakonfigurován Hibernate a informace nutné pro připojení k databázi. Také jsou zde nastaveny soubory, ve kterých byly namapovány tabulky, s odpovídající třídou. Snímek níže ukazuje konfiguraci v tomto souboru.:

```
<?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>
        <!-- Configuration of the connection -->
        <property name="hibernate.dialect"> org.hibernate.dialect.MySQL5Dialect</property>
        <property name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property>
        <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/ztraty_a_nalezy</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password">123</property>
        <!-- List of XML mapping files -->
        <mapping resource="mapping/ztrata.hbm.xml"
            class="cz.uhk.model.Ztrata" />
        <mapping resource="mapping/city.hbm.xml" class="cz.uhk.model.City" />
        <mapping resource="mapping/rights.hbm.xml"
            class="cz.uhk.model.Rights" />
        <mapping resource="mapping/user.hbm.xml" class="cz.uhk.model.User" />
        <mapping resource="mapping/lossState.hbm.xml"
            class="cz.uhk.model.LossState" />
        <mapping resource="mapping/reportingUser.hbm.xml"
            class="cz.uhk.model.ReportingUser" />
        <mapping resource="mapping/itemReturned.hbm.xml"
            class="cz.uhk.model.ItemReturned" />
        <mapping resource="mapping/picture.hbm.xml"
            class="cz.uhk.model.Picture" />
    </session-factory>
</hibernate-configuration>
```

Obrázek 11 Konfigurace frameworku Hibernate

Jednotlivé soubory mapování modelových tříd s odpovídajícími tabulkami v databázi se nacházejí ve složce src/main/resources/mapping. Podstatné je, že každý soubor musí mít koncovku „.hbm.xml“.

V těchto souborech se mapuje název dané třídy a tabulky, která k ní patří. Dále se v nich deklaruje primární klíč tabulky neboli v Hibernate tzv. „id“ a způsob jeho generace. V aplikaci byl nastaven generátor třídy „native“, což znamená, že Hibernate použije automaticky generátor primárního klíče, jaký používá daná databáze. Což je v případě MySQL tzv. „Auto Increment“, kde v případě číselných primárních klíčů databáze, jako první číslo, použije hodnotu 1. A další vytvořené řádky v tabulce mají automaticky primární klíč vygenerovaný o jednu hodnotu výše než poslední vytvořený řádek. Dále se zde mapují proměnné modelové třídy, odpovídající sloupec v tabulce a jejich datový typ. Také se zde konfigurují vazby k jiným tabulkám.

Snímek níže ukazuje mapování třídy a tabulky Ztrata.:

```
<?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="Ztrata" table="ztrata">
        <id name="id" type="int" column="id">
            <generator class="native" />
        </id>
        <property name="nazev" column="nazev" type="string" />
        <property name="popis" column="popis" type="string" />
        <property name="timeCreated" column="timeCreated" type="Date" />
        <property name="timeChanged" column="timeChanged" type="Date" />
        <property name="wasCreatedAsLoss" column="wasCreatedAsLoss" type="boolean"/>
        <many-to-one name="lossState" column="loss_state_id"
            foreign-key="id" />
        <many-to-one name="user" column="user_id" foreign-key="id" />
        <many-to-one name="itemReturned" column="returned_id"
            foreign-key="id" />
    </class>
</hibernate-mapping>
```

Obrázek 12 Mapování třídy a tabulky Ztrata v Hibernatu

Dále v modelových třídách, kde byly využity anotace v rámci validace a JPA mapování, tak pro funkčnost Hibernatu bylo nutno vygenerované primární klíče označit mj. anotací „@GenericGenerator“, což je anotace Hibernatu, ve které je nastaveno jméno a strategie generování na nativní.

Dále, v Javě se musí nejdříve implementovat vytváření instance tzv. SessionFactory. Což je rozhraní Hibernatu, která se stará o vytvoření instance rozhraní Session, prostřednictvím

kterého se vytváří spojení s databází a provádí se v ní operace. Vytvoření SessionFactory je implementováno ve třídě „HibernateUtil“, která se nachází v balíčku „cz.uhk.util“.

Samotný Hibernate je pak implementován v balíčku „cz.uhk.dao“, a to ve všech Dao třídách, které dědí od třídy „DaoBase“. Tato třída v konstruktoru ověří, zda je session neboli spojení s databází navázáno, a pokud ne, tak ho naváže.

Vzhledem k tomu, že otevření tohoto spojení trvá déle a Hibernate je implementován ve webové aplikaci, která neustále přistupuje k databázi, tak z důvodu optimalizace je instance session neustále otevřená a v jednotlivých metodách v Dao třídách není uzavřena. V případě, pokud by aplikace nepotřebovala neustály přístup k datům, tak je ale vhodné po skončení operace session ukončit, aby se uvolnila z paměti.

Třída DaoBase dále obsahuje metody pro práci s databází, konkrétně pro vytváření, mazání a aktualizaci dat v databázi. Pro tento účel je použita vytvořená instance session. V těchto metodách je tato instance prováděna během vytvoření transakce, která se získává také ze session. Samotná práce s daty a metoda transakce „commit()“, která potvrdí změny v databázi a uloží je v ní, jsou zabaleny do bloku try/catch. Kde, jelikož například potvrzení transakce může vracet vyjímky, tak v případě detekce výjimky se ověří, zda je transakce aktivní a pokud je, tak se zruší tzv. metodou „rollback()“, kterou se zruší prováděné změny. Transakci je také nutné zrušit v případě chyby, jinak zůstane stále otevřená a nepůjdou v databázi provádět žádné jiné změny bez restartu aplikace.

Vyhledávání dat v databázi bylo implementováno ve formě tzv criteria query, prostřednictvím specifikace JPA. Hibernate sice obsahuje také své vlastní API pro criteria query, ale jelikož že označeno, jako tzv. „deprecated“, tedy že autoři frameworku nedoporučují, aby toto API bylo použito, tak bylo využito API JPA namísto toho.

Criteria query bylo vybráno z toho důvodu, že oproti vyhledávání dat v databázi s využitím syntaxe nativní SQL, nebo jazyku daného frameworku, jako například HQL v případě Hibernate (30) , které se píšou ve Stringu, tak v tzv. kriteriálním dotazu se dotazy generují prostřednictvím rozhraní dané API, v tomto případě prostřednictvím JPA. V těchto rozhraních se používají metody, které jako parametr využívají proměnné z modelových tříd. Což zmenšuje pravděpodobnost překlepových chyb, jelikož názvy těchto proměnných kontroluje kompilátor.

A také zde není nutné vytvářet tzv. dynamický SQL kód neboli kód prostřednictvím sčítání Stringů, což není vhodné mj. u očekávaných parametrů od uživatele, kvůli tzv. SQL Injection, což je metoda útoku, kterou může útočník propašovat do SQL dotazu svůj vlastní, a tak například smazat tabulky apod.

Jako příklad kriteriálního dotazu si ukážeme metodu getById() ve třídě DaoBase. Nejdříve se přes CriteriaBuilder, který se získá od instance rozhraní Session, vytvoří instance rozhraní CriteriaQuery<>, které je generické a parametrizované zde generickou třídou T, pro účely dědění. Poté se prostřednictvím criteriaQuery vytvoří instance rozhraní Root, která reprezentuje hlavní třídu, kterou chceme získat z databáze. Pak se prostřednictvím metod instance criteriaQuery vytvoří samotný dotaz prostřednictvím jeho metod. Poté se prostřednictví vytvořeného dotazu v criteriaQuery a session vytvoří instance rozhraní Query, prostřednictvím kterého se následně dotaz vykoná a vrátí se výsledek, v tomto případě jeden metodou „uniqueResutl()“.

Snímek níže ukazuje kód metody „getById()“.:

```
@Override
public T getById(final int id, final Class<T> clas) {
    final CriteriaBuilder builder = session.getCriteriaBuilder();
    CriteriaQuery<T> criteriaQuery = session.getCriteriaBuilder().createQuery(clas);
    final Root<T> root = criteriaQuery.from(clas);
    criteriaQuery = criteriaQuery.select(root).where(builder.equal(root.get("id"), id));
    final Query<T> query = session.createQuery(criteriaQuery);
    return query.uniqueResult();
}
```

Obrázek 13 Ukázka získání jednoho řádku z databáze prostřednictvím specifikace JPA v Hibernate

4.2.2 DataNucleus a JDO

DataNucleus byl implementován v projektu ZtratyANalezyDataNucleusJDO. Byl stažen prostřednictvím Mavenu, a to konkrétně závislosti ze skupiny org.datanucleus. Artefakty, které byly staženy, jsou datanucleus-core, poté datanucleus-rdbms, který zajišťuje kompatibilitu DataNucleusu s relačními databázemi, jako je MySQL, a nakonec datanucleus-api-jdo a javax.jdo, což jsou knihovny potřebné pro implementaci JDO specifikace. Dále byly staženy artefakty datanucleus-xml, datanucleus-mavenu-plugin, datanucleus-jdo-query a datanucleus-enhancer.

Specifikace JDO využívá pro práci s databází tzv. PersistenceManager, který je vytvořen prostřednictvím PersistenceManagerFactory. O vytvoření instance PersistenceManagerFactory se stará v balíčku cz.uhk.dao, ve třídě DaoBase metoda createPersistenceManagerFactory().

V této metodě je také provedena základní konfigurace metadat persistence, která se normálně musí nastavit zvlášť v souboru ve formátu „.xml“, ale jednou z funkcí technologie DataNucleus je, že umí tato metadata vygenerovat přímo sám, prostřednictvím své třídy PersistenceUnitMetaData.

V rámci této konfigurace jsou nastaveny třídy modelových dat, a také vlastnosti spojení s databází a vlastnosti samotého DataNucleusu. Snímek níže zobrazuje tuto konfiguraci.:

```
protected PersistenceManagerFactory createPersistenceManagerFactory() {
    final PersistenceUnitMetaData persistenceMetaData = new PersistenceUnitMetaData("dynamic-unit",
            "RESOURCE_LOCAL", null);
    persistenceMetaData.addClassName("cz.uhk.model.City");
    persistenceMetaData.addClassName("cz.uhk.model.ItemReturned");
    persistenceMetaData.addClassName("cz.uhk.model.LossState");
    persistenceMetaData.addClassName("cz.uhk.model.Picture");
    persistenceMetaData.addClassName("cz.uhk.model.ReportingUser");
    persistenceMetaData.addClassName("cz.uhk.model.Rights");
    persistenceMetaData.addClassName("cz.uhk.model.User");
    persistenceMetaData.addClassName("cz.uhk.model.Ztrata");
    persistenceMetaData.setExcludeUnlistedClasses(true);

    persistenceMetaData.addProperty("javax.jdo.option.ConnectionDriverName", "com.mysql.cj.jdbc.Driver");
    persistenceMetaData.addProperty("javax.jdo.option.ConnectionURL",
            "jdbc:mysql://localhost:3306/ztraty_a_nalezy");
    persistenceMetaData.addProperty("javax.jdo.option.ConnectionUserName", "root");
    persistenceMetaData.addProperty("javax.jdo.option.ConnectionPassword", "123");
    persistenceMetaData.addProperty("datanucleus.autoCreateSchema", "true");
    persistenceMetaData.addProperty("datanucleus.rdbms.mysql.characterSet", "utf8_bin");
    return new JDOPersistenceManagerFactory(persistenceMetaData, null);
}
```

Obrázek 14 Vytvoření PersistenceManagerFactory v DataNucleus JDO

Mapování modelových tříd a jejích tabulek se kromě ve vygenerovaných metadatech muselo také provést přímo v modelových třídách, ve formě anotací. Každá modelová třída je označená anotací „@PersistenceCapable“, ve které je prostřednictvím parametrů definován název tabulky a typ identity, který byl zvolen jako aplikační. V případě aplikační identity se předává kontrola nad specifikací primárních klíčů DataNucleusu. (31) Dále byly prostřednictvím anotaci namapovány proměnné a vazby. V případě vazeb je zajímavé, že cizí klíče stačilo namapovat anotací „@Column“ stejně, jako proměnné dané třídy.

Snímek níže ukazuje mapování třídy ReportingUser.:

```
@PersistenceCapable(table = "reporting_user", identityType = IdentityType.APPLICATION)
public class ReportingUser {

    @PrimaryKey(column = "id")
    @Persistent(valueStrategy = IdGeneratorStrategy.NATIVE)
    private int id;
    @Column(name = "time_reported", allowsNull = "false")
    private Date timeReported;
    @Column(name = "user_id", allowsNull = "false")
    private User user;
    @Column(name = "ztrata_id", allowsNull = "false")
    private Ztrata ztrata;
```

Obrázek 15 Mapování modelové třídy ReportingUser ve specifikaci JDO, implementaci DataNucleus

Pro funkčnost DataNucleusu je nutné u všech těchto modulových tříd, aby implementovaly rozhraní „Persistable“. Toto zajišťuje tzv. enhancer neboli vylepšovač, který upravuje zkompilované třídy, aby implementovaly toto rozhraní a všechny jeho metody. (32)

Implementace tohoto vylepšování však byla komplikovaná. Jelikož první pokus o implementaci vylepšovače formou Mavenu, kdy se do souboru pom.xml nastavilo, aby se vylepšovač spustil při kompilaci, skončil neúspěšně. Vylepšovač byl tedy implementován formou doplňku do Eclipse. (33) V tomto doplňku lze pak povolit projektu podporu DataNucleusu, jako v tomto případě projekt „ZtratyANalezyDataNucleusJDO“ a následně provést „vylepšení“. Nutné však je v konfiguraci tohoto doplňku, která se nachází v preferencích Eclipsu, zda má využívat API persistence JDO nebo JPA. Což je nutné, pokud je tento doplněk použit u více projektů.

Data se do databáze ukládají a mažou prostřednictvím instance JDO rozhraní persistenceManager. Ukládají se metodou „makePersistent()“, která, jako parametr, očekává instanci třídy, která se má uložit. Mažou se metodou „deletePersistent()“, která také očekává instanci třídy, která se má vymazat. Tyto základní metody obsahuje třída DaoBase a ostatní třídy je od ní dědí.

Stejně, jako v Hibernate API, tak jsou také operace persistenceManagera provedeny, jako součást transakce, která se získává také od persistenceManagera. Transakce se navíc ale musí před prací s daty v databázi začít metodou begin(). Stejně, jako v Hibernate, tak i zde může transakce vyhodit v případě chyby vyjímku, a proto je provedení operace a potvrzení transakce zabaleno v try/catch bloku a v případě nezdaru je transakce zrušena, pokud je aktivní.

Změna dat v databázi je ale v případě specifikace JDO problematická. Jelikož během implementace metoda makePersistent() vytvářela duplikát, namísto aktualizace řádku v databázi, tak metodu update() si musí každá modelová třída implementovat zvlášť. Proto tato metoda byla označena, jako abstraktní, což znamená, že nemusí být implementována přímo v dané třídě a její implementace se přenechává dědicům.

Implementace aktualizace dat v databázi je v případě JDO zbytečně složité. Například v implementaci metody update() ve třídě Ztrata, je nutné nejdříve si od persistenceManager získat transakci a tu zahájit. Poté se musí stejná instance třídy, jako už je přijata v metodě, jako parametr, se musí stáhnout z databáze, v tomto případě je to instance updateZtrata. Potom se musí té instanci zvlášť změnit její proměnné. A pak potvrdit transakci, čímž se uloží změny v databázi.

K vyhledávání dat v databázi, podobně, jako v JPA jsou CriteriaQuery, tak v případě JDO je to tzv. JDOQLTypedQuery, což je rozhraní, které s použitím svých metod generuje dotazy v syntaxi JDOQL, což je dotazovací jazyk specifikace JDO. K funkčnosti této specifikace musí mít každá třída vytvořenou svoji dotazovací neboli tzv. query třídu, která má stejné jméno, jako její modelová třída, kromě „Q“ na začátku, takže například tázací třída třídy Ztrata se jmenuje „QZtrata“.

Tyto dotazovací třídy generuje DataNucleus u všech tříd s anotací @PersistenceCapable, ale pro funkčnost tohoto generování je to nutné nastavit v IDE, v tomto případě v Eclipse. Ve vlastnostech projektu, konkrétně v Java Compiler/Anotation Processing se musí povolit specifická nastavení projektu, pak zpracovávání anotací. Dále se do části vlastností „Factory Path“, která se nachází v Annotation Processing, se musí přidat knihovny zpracovávající tyto anotace, konkrétně knihovna „datanucleus-jdo-query-5.0.9.jar“, která se v případě této aplikace nachází v „C:\Users\Jméno uživatele\.m2\repository\org\datanucleus\“.

Dále je nutné v možnostech Factory Path také přidat knihovnu „javax.jdo-3.2.0-m13.jar“, která se nachází v „C:\Users\Jméno uživatele\.m2\repository\org\datanucleus\javax.jdo\3.2.0-m13\javax.jdo-3.2.0-m13.jar“. Po tomto nastavení se při kompilaci budou automaticky generovat dotazovací třídy u těch s příslušnou anotací.

Psaní dotazů pak je jednodušší. Například ve třídě ZtrataDao, v metodě „getItemsForUserByName(name, id)“ se nejdříve musí s pomocí instance persistenceManager vytvořit nová instance generického rozhraní JDOQLTypedQuery, kde se jak do vytvářející metody „newJDOQLTypedQuery“, tak do parametru generického rozhraní vkládá třída, na níž se bude aplikace v databázi dotazovat.

Následně se vytvoří instance dotazovací třídy QZtrata zavoláním statické metody stejné třídy „.candidate()“. Následně se s pomocí proměnných této třídy vytvoří dotaz, který se v tomto případě poté provede metodou „.executeList()“. Snímek níže ukazuje celou metodu „getItemsForUserByName“.:

```
public List<Ztrata> getItemsForUserByName(final String name, final int userId) {
    final JDOQLTypedQuery<Ztrata> query = persistenceManager.newJDOQLTypedQuery(Ztrata.class);
    final QZtrata candidate = QZtrata.candidate();
    return query
            .filter(candidate.user.id.eq(userId)
                    .and(candidate.user.id.eq(userId).and(candidate.nazev.matches(".*" + name + ".*"))))
            .executeList();
}
```

Obrázek 16 Příklad dotazovacího volání JDO

Výjimka, kde takto dotazy nešlo implementovat, byly dotazy v metodách „getAll()“ a „getById()“, ve třídě DaoBase. V případě getAll() stačilo v JDOQLTypedQuery vytvořit pouze tento typovaný dotaz s generickou třídou T a její instancí clas, která se očekává, jako parametr v metodě getAll(). Poté lze hned dotaz provést bez nutnosti dalšího nastavení.

V případě getById bylo použito API JDOQL, kde se musel definovat filtr neboli deklarovat proměnnou, jako filtr a ten pak umístit, jako parametr do dotazu. Snímek níže ukazuje průběh celé metody.:

```
@Override
public T getById(final int id, final Class<T> clas) {
    final Query<T> q = persistenceManager.newQuery(clas);
    q.setFilter("id==paramId");
    q.declareParameters("int paramId");
    return q.setParameters(id).executeUnique();
}
```

Obrázek 17 Získání instance podle Id v JDO

4.2.3 Implementace JPA v Hibernate

Implementace specifikace JPA ve frameworku Hibernate bude ukázána na projektu „ZtratyANalezyHibernateJPA“. Nejdříve se musela nastavit metadata persistence. Jejich nastavení se nachází ve složce „src/main/resources/META-INF/“, v souboru persistence.xml. Zde jsou nastaveny modelové třídy a vlastnosti spojení s databází. Název jednotky persistence byl zadán, jako „ZtratyANalezy“.

Mapování tříd stejně, jako u JDO, proběho prostřednictvím anotací. Snímek níže zobrazuje mapování třídy ReportingUser.:

```
@Entity
@Table(name = "reporting_user")
public class ReportingUser {

    @Id
    @Column(name = "id", nullable = false)
    @GeneratedValue(strategy = GenerationType.AUTO, generator = "native")
    @GenericGenerator(name = "native", strategy = "native")
    private int id;
    @Column(name = "time_reported", nullable = false)
    private Date timeReported;
    @ManyToOne
    @JoinColumn(name = "user_id", nullable = false)
    private User user;
    @ManyToOne
    @JoinColumn(name = "ztrata_id", nullable = false)
    private Ztrata ztrata;
```

Obrázek 18 Mapování třídy ReportingUser v Hibernate JPA

Ve třídě DaoBase se v případě JPA s databází pracuje prostřednictvím instancí rozhraní EntityManagerFactory a EntityManager. Při vytváření entityManagerFactory se v metodě createEntityManagerFactory(), třídy Persistence, musí, jako parametr uvést název jednotky metadat, který byl použit v konfiguraci persistence. Následně se prostřednictvím instance entityManagerFactory vytvoří entityManager.

Při práci s daty se používá entity manager, a to jeho metoda persist() pro uložení dat do databáze, merge() pro aktualizaci dat v databázi a remove() pro smazání dat z databáze. Získávání dat z databáze implementuje kriteriální dotazování.

Snímek níže ukazuje získání instance podle jejího primárního klíče.:

```
@Override
public T getById(final int id, final Class<T> clas) {
    final CriteriaBuilder builder = entityManager.getCriteriaBuilder();
    CriteriaQuery<T> criteriaQuery = builder.createQuery(clas);
    final Root<T> root = criteriaQuery.from(clas);
    criteriaQuery = criteriaQuery.select(root).where(builder.equal(root.get("id"), id));
    final TypedQuery<T> query = entityManager.createQuery(criteriaQuery);
    try {
        return query.getSingleResult();
    } catch (NoResultException e) {
        return null;
    }
}
```

Obrázek 19 Získání instance podle jejího primárního klíče v Hibernate JPA

Byla snaha také implementovat specifikaci JPA v DataNucleusu, v projektu ZtratyANalezyDataNucleusJPA, ale z důvodu problémů při implementaci aktualizaci dat v databázi a výjimek ze strany DataNucleusu v případě některých dotazů byla implementace namísto toho provedena v Hibernate, kde JPA mnohem lépe funguje.

4.2.4 MyBatis

MyBatis persistentní framework je implementován v projektu „ZtratyANalezyMyBatis“. Jeho knihovny byly staženy prostřednictvím Mavenu, a to skupiny „org.mybatis“, artefaktu „mybatis“.

MyBatis byl nakonfigurován v souboru myBatis.xml, který se nachází ve složce „src/main/resources/“. V tomto souboru jsou nakonfigurovány modelové třídy neboli tzv. typeAliases, kde se nastaví celá cesta třídy ve formě Javy, což znamená, že například třída City, nacházející se v balíčku „cz.uhk.model“, se musí nastavit u vlastnosti typeAlias, jako „cz.uhk.model.City“ a její alias pro jednodušší volání je v aplikaci nastaven, jako „City“. Dále jsou v této konfiguraci nastavené vlastnosti pro připojení k databázi a soubory, které mapují jednotlivé modelové třídy a jejich tabulky a které se nacházejí ve složce „src/main/resources/mapping/“.

V souborech mapování se nachází jednak mapování výsledků neboli tzv. „resultMap“, kde jsou namapovány proměnné třídy se sloupci v databázi a vazby na jiné tabulky a třídy. Každé toto mapování výsledků má svůj název neboli tzv. „id“, takže lze napsat více variant mapování pro různé potřeby aplikace. Zajímavé je, že zde se vztahy neboli asociace se mapují na mapování výsledku jiné třídy.

Každá namapovaná proměnná zde má atribut property, což je název proměnné ve třídě, a column, což je název sloupce v databázi. V případě využití tzv. aliasů neboli přidělení jiného jména sloupci v rámci dotazu select, lze tyto aliasy použít, jako název sloupce. Je nutné ale pak na ně nezapomenou v případě implementace samotných dotazů.

Snímek níže například ukazuje v mapujícím souboru ztrata.xml mapování výsledků třídy Ztrata.:

```
<resultMap id="ztrataResult" type="Ztrata">
    <id property="id" column="ztrata_id" />
    <result property="nazev" column="nazev" />
    <result property="popis" column="popis" />
    <result property="timeCreated" column="timeCreated" />
    <result property="timeChanged" column="timeChanged" />
    <result property="wasCreatedAsLoss" column="wasCreatedAsLoss" />
    <association property="itemReturned"
        resultMap="ItemReturned.itemReturnedResult" />
    <association property="lossState"
        resultMap="LossState.lossStateResult" />
    <association property="picture"
        resultMap="Picture.pictureResult" />
    <association property="user" resultMap="User.userResult" />
</resultMap>
```

Obrázek 20 Mapování výsledku třídy Ztrata v MyBatis

V každém souboru mapování kromě mapování proměnných a vazeb třídy jsou také napsány operace práce s databází a vyhledávání dat v ní, pod xml elementy insert, update, delete a select, a to ve formě SQL kódu. V případě insertu neboli vložení dat je nutné použít nastavení „useGeneratedKey=“true““, kterým se nastaví frameworku, aby používal primární klíče vygenerované databází. Každá operace v mapujícím souboru má svůj název neboli tzv. „id“, kterým se následně volá z Javy. Každý select dotaz musí mít také nastavenou odpovídající resultMap, v rámci, které je namapován.

MyBatis za autora aplikace oproti ORM frameworkům, jako například Hibernate, neřeší problémy, jako stejné názvy sloupců u různých tabulek, ani, co má vrátit v případě, že má jedna tabulka vazby na jiné tabulky. Tyhle všechny problémy se proto musí řešit v daném dotaze v jazyce SQL. Výhodou zde ale je, že framework z databáze vrátí pouze ty proměnné, které se v SQL syntaxi určí, oproti ORM frameworkům, které vrací všechny proměnné dané třídy a jejích vazeb, alespoň v případě implementace v rámci této práce.

Parametry se do SQL kódu v MyBatis vkládají prostřednictvím mřížky a následně složené závorky, do které se uvádí název parametru. Jako příklad vkládání parametrů do SQL dotazu poslouží snímek níže, zobrazující operaci insert neboli vložení nového řádku do databáze, v tabulce Ztrata.:

```
<insert id="insert" useGeneratedKeys="true">
    insert into ztrata (nazev,
    popis, user_id,
    loss_state_id, returned_id,
    timeCreated,
    timeChanged,
    wasCreatedAsLoss,
    picture_id)
    values(#{nazev},#{popis},#{user.id},#{lossState.id},#{itemReturned.id},#{timeCreated},
    #{timeChanged}, #{wasCreatedAsLoss}, #{picture.id});
</insert>
```

Obrázek 21 Vložení nového řádku do databáze tabulky Ztrata, ve frameworku MyBatis

V Javě MyBatis provádí spojení s databází prostřednictvím rozhraní SqlSession, jehož instanci vytváří rozhraní SqlSessionFactory. Instance SqlSessionFactory je vytvořena v metodě buildSessionFactory() třídy DaoBase, ve které je vytvořena prostřednictvím třídy SqlSessionFactoryBuilder a vstupního proudu dat, který přijímá, jako data, soubor myBatis.xml.

Snímek níže ukazuje celý průběh metody.:

```
protected SqlSessionFactory buildSessionFactory() {
    final SqlSessionFactory sessionFactory;
    try {
        final InputStream inputStream = Resources.getResourceAsStream("/myBatis.xml");
        final SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();
        sessionFactory = builder.build(inputStream);
        inputStream.close();
        return sessionFactory;

    } catch (IOException e) {
        e.printStackTrace();
    }
    return null;
}
```

Obrázek 22 MyBatis – vytvoření SqlSessionFactory

Prostřednictvím SqlSessionFactory se následně v konstruktoru třídy otevře instance SqlSession, pojmenovaná jako session, prostřednictvím které se poté volají metody pro práci s databází. V případě změn dat v databázi je nutné změny následně potvrdit metodou commit().

Každá metoda instance session, která pracuje s daty v databázi, očekává, jako parametr, volání dané operace prostřednictvím Stringu, kterým se zavolá daná akce z mapovacího souboru. Což znamená, že například, pokud chceme získat všechna data třídy Ztrata, tak jako parametr vložíme String „Ztrata.getAll“. Do těchto metod také lze kromě tohoto Stringu vložit parametr pro danou akci.

Problém ale nastává v případě zájmu poslat do volání akce více parametrů, jako například v případě metody getFound LossessOfUserByName, třídy ZtrataDao. Tato metoda očekává dva parametry, a to jméno a primární klíč ztráty. V takovém případě je nutno použít HashMap, ve kterém se vytvoří mapování ve formě dvojic: „String, Object“, neboli textový název proměnné a její hodnota. A dané parametry se poté uloží do dané instance HashMapu a tato instance se poté s parametry pošle metodě instance session pro práci s databází.

4.2.5 JDBC

JDBC byl implementován v projektu ZtratyANalezyJDBC. JDBC není třeba stahovat z externích zdrojů, jelikož se nachází v Javě, v balíčku „Java.sql“.

V JDBC nebylo třeba provádět moc konfigurace, tudíž nastavení spojení s databází, což je jediné, co JDBC požaduje nastavit, je nastaveno ve třídě DaoBase, v metodě createConnection(). Tato metoda ve formě textového řetězce nastavuje parametry připojení k databázi, a pak je vkládá do metody „getConnection“, třídy DriverManager, která očekává, jako parametry, textový řetězec URL spojení s databází, jméno a heslo uživatele.

Je nutné také zmínit, že samotný JDBC je jenom specifikace, což znamená, že ke své funkčnosti potřebuje technologii, která ji implementuje. K tomu v případě JDBC slouží ovladače od výrobců dané databáze. JDBC si tento ovladač umí nastavit sám, ale v případě neschopnosti automatického nastavení ovladače tak, jako se to stalo v případě projektu ZtratyANalezyOACC, tak je nutné také zavolat metodu „Class.forName(„com.mysql.cj.jdbc.Driver“)“, kterou se zavede v případě této aplikace ovladač MySQL do aplikace.

JDBC využívá rozhraní Connection pro vytvoření spojení s databází. Tímto rozhraním se také následně vytváří instance rozhraní Statement nebo PreparedStatement, kterými se následně provádí práce s databází, ve formě SQL kódu. Statement i PreparedStatement jsou proudy dat, což znamená, že po ukončení práce s nimi musí být ručně uzavřeny, aby se uvolnily z paměti. Jelikož v případě chyby tato rozhraní mohou vyhodit různé vyjímky, tak práce s nimi v Dao třídách byla provedena v rámci try/catch bloku. Jelikož tato rozhraní také podporují automatické uzavření po dokončení try/catch bloku, tak lze instance těchto rozhraní vytvořit přímo pro daný try blok.

Rozdíl mezi Statement a PreparedStatement je, že PreparedStatement podporuje práci s parametry. Jelikož není vhodné parametry vkládat ve formě sčítání textových řetězců kvůli hrozícímu SQL injection útoku, tak namísto toho lze v SQL kódě, který se následně vloží, jako textový řetězec do instance PreparedStatement, parametry vložit, jako otazníky. Následně pak každý parametr má svoje číslo pořadí, od 1, a na ta pořadí pak lze vložit jednotlivé parametry.

Snímek níže ukazuje práci s PreparedStatementem v metodě delete(), třídy DaoBase.:

```
@Override
public void delete(final String className, final int id) {
    final String sql = "delete from " + className + " where id=?;";
    try (final PreparedStatement statement = connection.prepareStatement(sql)) {
        statement.setInt(1, id);
        statement.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Obrázek 23 Smazání dat z databáze v JDBC

Poznámka: V případě metody delete, jelikož nešlo vložit název třídy, jako parametr, tak musel být přičten, jako String. Což se samozřejmě v žádném případě nedoporučuje a v rámci aplikace to bylo implementováno pouze v pevně daném názvu třídy v kódě. Tudíž proměnná „className“ nepřijímá žádné informace od uživatele.

Jelikož SQL syntax pro každou třídu je jiný, a i následné zpracování přijatých dat od databáze, tak většina metod pro práci s databází ve třídě DaoBase jsou abstraktní, jelikož si je každá třída musí implementovat sama.

Kromě rozhraní posílajících SQL syntaxe do databáze je zde také rozhraní ResultSet, které pracuje s daty přijatými z databáze. V takovém případě je v případě volání SQL kódu v například Statementu zavolat metodou „executeQuery()“, která vrátí instanci rozhraní ResultSet s daty. Hlavní problém v JDBC je, že nepodporuje žádné mapování, takže poté se musí veškerá data z databáze ručně převést do instance modelové třídy metodami get ResultSetu. ResultSet sice obsahuje metodu „getClass()“, která by zřejmě mohla převést celý jeden řádek v ResultSetu na instanci modelové třídy, ale ovladač MySQL bohužel tuhle metodu nepodporuje. Tudíž přijímání dat z databáze je v JDBC hodně komplikované.

Snímek níže ukazuje přijetí dat pro třídu Ztrata v metodě getAll(), ve třídě ZtradaDao.:

```
    try (final Statement statement = connection.createStatement();
            final ResultSet result = statement.executeQuery(sql)) {
        while (result.next()) {
            final User user = new User(result.getInt("user_id"), result.getString("user_name"),
                    result.getString("surname"));
            final LossState lossState = new LossState(result.getInt("loss_state_id"),
                    result.getString("loss_state_name"), result.getString("loss_state_description"));
            final ItemReturned itemReturned = new ItemReturned(result.getInt("item_returned_id"),
                    result.getDate("timeOfReturn"));
            final Picture picture = new Picture(result.getInt("picture_id"), result.getString("picture_name"),
                    result.getString("type"), result.getShort("size"), result.getBytes("file"));
            ztraty.add(new Ztrata(result.getInt("id"), result.getString("nazev"), result.getString("popis"),
                    result.getDate("timeCreated"), result.getDate("timeChanged"),
                    result.getBoolean("wasCreatedAsLoss"), lossState, user, itemReturned, picture));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return ztraty;
}
```

Obrázek 24 Zpracování přijatých dat z databáze v JDBC

Podstatné při práci s ResultSetem je také brát v úvahu jeho tzv. kurzor, který se při jeho vytvoření nachází před prvním řádkem přijatým z databáze. Což znamená, že před zpracováváním dat se musí zavolat metoda next(), která posune kurzor na další řádek, pokud možno a vrátí boolean true, pokud se posunul na další řádek a false, pokud se posunul za poslední řádek. Proto je vhodné práci s daty provést v rámci cyklu while, který volá tuto metodu.

ResultSet, stejně, jako Statement a PreparedStatement, je proud dat, vyžadující své ruční uzavření po skončení práce s ním. Také podporuje své automatické uzavření v bloku try, ale také je vázán. ResultSet je ale také vázán na instanci rozhraní, například Statement, kterou byl vytvořen. Což znamená, že po ukončení této instance je instance ResultSetu také automaticky uzavřena, a tudíž se nemusí zavírat ručně. (34)

4.2.6 Porovnání databázových technologií

Technologie, která se nejhůře implementovala, je JDBC, jelikož neobsahuje žádné mapování, takže se musí u každé metody při přijetí dat ručně převádět každý sloupec řádku dat na proměnnou instance. Což je zbytečně zdlouhavé a zpomaluje to práci, zvlášť u větších tříd a jejích tabulek.

MyBatis, jako technologie pro psaní SQL Syntaxí je díky tomu oproti JDBC lepší, kromě podpory mapování také podporuje víc verzí mapování pro danou třídu, takže si lze vytvořit více mapování podle potřeby. Problémem ale je, že nativní syntax SQL podporují také ostatní porovnávané technologie, jejichž práce s nativním SQL kódem by teoreticky mohla být i lepší než v případě MyBatis, pokud by tato práce porovnávala implementaci nativního jazyku SQL u všech porovnaných technologií.

Dále je vhodné porovnat Hibernate a DataNucleus v jejich implementaci. Zde je lepší jednoznačně Hibernate, protože nevyžaduje implementaci žádného speciálního rozhraní nebo jeho generací prostřednictvím komplikovaného „vylepšovače“ a i když oproti DataNucleusu neumí generovat metadata persistence pro JPA implementaci přímo v Javě tak, jak to umí DataNucleus, tak metadata persistence lze jednoduše nakonfigurovat v jejích xml souboru. Navíc v Hibernate proběhla implementace JPA bez problémů, kdežto DataNucleus při stejné implementaci Dao tříd vyhazoval v některých případech nepřehledné vyjímky, konkrétně null pointer vyjímky, bez vysvětlení, co je špatně, a tudíž to celou práci komplikovalo.

Při samotné práci s daty s databází, tedy jejích ukládání, vytváření nebo mazání nejlepší při implementaci bylo API Hibernate, konkrétně jeho Session. Které podporuje vytváření nových řádků v tabulce, jejích změny a mazání. Jediný problém může nastat, pokud aplikace obsahuje dvě instance jednoho řádku v tabulce, což může při snaze prací v databázi, například aktualizaci jednou instancí daného řádku v tabulce, způsobit vyjímky z důvodu snahy o práci s instancí, kterou už má Hibernate uloženou v paměti v jiné podobě. Ale to Hibernate řeší metodou v Session „merge“, která v paměti tyto dvě instance spojí dohromady a poté aktualizuje data v databázi na základě té instance, která se do metody merge vložila.

Metoda merge pro aktualizaci v případě EntityManageru v JPA také umí aktualizovat existující data v databázi, ale v některých implementacích (například DataNucleus) nemusí aktualizovat všechno.

Nejhorší při prací s databází byl EntityManager v JDO, jelikož ten nepodporuje aktualizaci existujících dat v databázi, takže jejích aktualizace je zbytečně zdlouhavý a komplikovaný proces.

V případě posílání dotazů databázi pro získání dat byla nejlepší implementace JDO, JDOQL typed query, protože i když vyžadovala konfiguraci anotací, tak vytváření syntaxe JDOQL prostřednictvím dotazových tříd a jejích metod byl jednodušší a navíc díky ověření kompilátorem i spolehlivější, než složitější CriteriaQuery v JDA, kde se sice nemusí dělat žádná konfigurace navíc, ale zase se musí komplikovaně používat více rozhraní a část parametrů se píše, jako textový řetězec, což znamená, že tuto část kompilátor nezkontroluje.

Pro tvorbu nějaké definitivní aplikace by v rámci práce byla ideální technologie, která by měla implementované Session z Hibernate a JDO, ale protože taková technologie mezi srovnávanými není, tak by byl vybrán Hibernate se Session pro práci s daty v databázi a JPA specifikace pro jejích hledání, implementována v Hibernate.

4.3 Frameworky zabezpečení

V následující části budou popsány frameworky zabezpečení. Byly převážně vybrány technologie zaměřující se na autentizaci a autorizaci.

4.3.1 Spring Security

Framework Spring Security byl implementován v projektu ZtratyANalezy a byl stažen prostřednictvím Mavenu, konkrétně skupina „org.springframework.security“, artefakty „spring-security-test“, „spring-security-web“, „spring-security-config“, „spring-security--jwt“ a „spring-security-taglibs“.

V balíčku „cz.uhk.authenticationImplementation“ jsou implementována rozhraní frameworku pro autentizaci a autorizaci, a to UserDetails ve třídě UserDetailsImplementation, která implementuje metody pro získání přihlašovacího jména uživatele, jeho hesla, získávání celé instance třídy uživatele User a získání uživatelova oprávnění. Uživatelská oprávnění ve frameworku jsou uložena ve formě kolekce rozhraní GrantedAuthority, do které, jelikož aplikace používá pouze jednu roli uživatele, je uživatelova role uložena ve formě instance třídy SimpleGrantedAuthority, což je implementace tohoto rozhraní.

Ve stejném balíčku je také implementováno rozhraní frameworku UserDetailsService třídou UserDetailsServiceImplementation, které implementuje metodu loadUserByUsername(), která získává třídu uživatele podle uživatelského jména, které metoda přijímá, jako parametr. V případě implementující aplikace uživatele získává třída UserDao. S instancí uživatele pak metoda vytvoří a vrátí novou instanci třídy UserDetailsImplementation.

V balíčku „cz.uhk.config“ se nachází třída WebSecurityConfig, která obsahuje základní konfigurace frameworku. Dědí ze třídy WebSecurityConfigurerAdapter, ze které následně implementuje konfigurační metody.

Tato třída nastavuje implementaci rozhraní UserDetailsService třídou UserDetailsServiceImplementation(), BCryptPasswordEncoder, jako kódovač hesel, který hashuje hesla algoritmem BCrypt. Pro zajímavost, hashování hesla znamená jednosměrné převedení hesla do směsice znaků a číslic. Dále třída nastavuje implementaci rozhraní DaoAuthenticationProvider.

Kromě metod implementujících rozhraní frameworku, je zde podstatná poslední metoda, configure(). Zde se nastavují přes třídu HttpSecurity vlastnosti přihlašování, odhlašování uživatele a požadovaná oprávnění k přístupu k daným URL. Přihlašování a odhlašování zpracovává framework Spring Security přímo po přijetí požadavku na nastavené URL, což znamená, že se to neprovádí prostřednictvím akce v ovladači. Důležité proto také je v přihlašovacím formuláři nastavit názvy parametrů uživatelského jména a hesla stejné, jako jsou v nastavení Spring Security. Výsledek přihlášení a odhlášení, a to jak úspěšný, tak neúspěšný, se už ale zpracovává na úrovni MVC části Springu, v akcích kontroleru LoginController.

V případě odhlášení je problém, že v případě povolené csrf ochrany, kterou Spring Security podporuje a v rámci jeho implementace byla zapnuta, tak přijímá požadavek na odhlášení pouze v typu požadavku POST. Jelikož aplikace využívá odkaz v navigaci pro odhlášení, což znamená typ požadavku GET, tak se tento typ požadavku musí zvlášť nastavit v konfiguraci.

U většiny metod je umístěna anotace „@Bean“, která označuje danou metodu jako metodu produkující bean neboli fazolku, kterou bude následně zpravovat Spring. Nejdůležitější jsou ale anotace na úrovni třídy, a to „@Configuration“, která označuje danou třídu jako konfigurační

třídu Springu, „@EnableWebSecurity“, která povoluje zabezpečení webové aplikace Springem Security a v případě využití anotací „@Secured“ je třeba na úroveň třídy také umístit anotaci „@EnableGlobalMethodSecurity(secureEnabled=true)“, kterou se povolí použití této anotace.

U kontrolerů, v balíčku cz.uhk.controller, lze následně jak na úrovni třídy, tak na úrovni metody použít anotaci „@Secured“, do které se, jako textový řetězec, umístí parametr, kterým se nastaví, jaké role mají k dané části webové aplikace přístup. Zároveň se touto anotací vyžaduje, aby uživatel přistupující k dané části webové aplikace byl přihlášen.

V ovladači LoginController jsou zpracovány výsledky přihlášení a také registrace. V případě registrace je nutné zahashovat heslo před jeho uložením do databáze. K tomu Spring Security v případě implementovaného algoritmu BCrypt obsahuje třídu BCryptPasswordEncoder, kterou lze k tomu využít. Tato třída je také využita v ovladači UserController, při změně uživatelských údajů. Data přihlášeného uživatele se následně získávají prostřednictvím jeho přihlašovacího jména, které se získává voláním „SecurityContextHolder.getContext().getAuthentication().getName()“.

Spring Security také obsahuje svůj vlastní tag do jsp pohledů, umožňující identifikace uživatelů přímo v pohledu a podle toho jeho generování.

4.3.2 Apache Shiro

Apache Shiro byl implementován v projektu ZtratyANalezyStruts. Technologie byla stažena prostřednictvím Mavenu, ze skupiny „org.apache.shiro“, artefakty „shiro-core“, „shiro-web“, „shiro-servlet-plugin“. Kvůli kompatibilitě s hesly uloženými v databázi byly také staženy šifrovací knihovny z frameworku Spring Security, konkrétně ze skupiny V Mavenu „org.springframework.security“, artefact „spring-security-crypto“, jelikož Apache Shiro nepodporuje hashování algoritmem BCrypt.

Třída HibernateRealm, v balíčku cz.uhk.realm, implementuje vlastní tzv. realm, neboli autorizaci a autentizaci s využitím tříd UserDao a User. Třída HibernateRealm implementaci provádí prostřednictvím děděním ze třídy frameworku AuthorizingRealm, ze které následně implementuje metody doGetAuthorizationInfo() a doGetAuthenticationInfo.

Metoda doGetAuthorizationInfo přijímá, jako parametr od frameworku kolekci principálů neboli údajů přihlášeného uživatele. Z kolekce metoda získá tzv. primární principál neboli přihlašovací jméno uživatele, na základě, kterého může následně získat uživatele z databáze. Jméno role uživatele následně uloží a vrátí ve formě třídy SimpleAuthorizationInfo, což je implementace rozhraní AuthorizationInfo, které metoda vrací. Pokud by metoda z nějakého důvodu nemohla získat uživatele z databáze, tak vyhodí výjimku neznámého účtu.

Metoda doGetAuthenticationInfo, která vrací implementaci rozhraní AuthenticationInfo, je implementována podobně. Jako parametr očekává od frameworku instanci rozhraní AuthenticationToken, která v sobě obsahuje přihlašovací údaje uživatele neboli přihlašovací jméno a heslo. Metoda z tohoto tokenu tyto přihlašovací údaje získá a vloží je do instance třídy SimpleAuthenticationInfo, kterou následně vrací. Kromě toho také tato instance v sobě obsahuje jméno realmu, které je prostřednictvím konstruktoru nastavené, jako „HibernateRealm“. V případě nenalezení účtu tato metoda vyhodí příslušnou vyjímku.

Apache Shiro, jak už bylo zmíněno, nepodporuje hashovací algoritmus BCrypt. Což znamená, že musel být nastaven zvlášť, a to ve formě implementace rozhraní frameworku PasswordService. Implementace byla učiněna ve třídě BCryptPasswordService, která se nachází v balíčku cz.uhk.passwordService. Z tohoto rozhraní třída implementuje metodu encryptPassword, která hashuje přijaté heslo třídou BCrypt ze Spring Security. A metodu passwordMatch, která porovnává přijaté heslo od uživatele, v tzv. plain textu, se zaheshovaným heslem v databázi a booleanem vrací buď true, pokud jsou obě hesla totožná, anebo false, pokud ne.

Tyto vlastní implementace, s implementacemi od Shira, jsou nastaveny v konfiguračním souboru shiro.ini, který se nachází ve složce „src/main/webapp/WEB-INF/“. Kromě hlavní části konfigurace se v tomto souboru také nachází část „[urls]“, ve které se v případě webové aplikace nastavují požadovaná oprávnění pro URL částí webové aplikace. Anon nastavení znamená, že všichni návštěvníci dané části webové aplikace k ní mají přístup. Authc vyžaduje, aby byl uživatel přistupující k dané části aplikace autentizován. A roles nastavení nastavuje požadavek uživatele mít určitou roli při přístupu k dané části aplikace.

Ve stejné složce jsou také v souboru web.xml nakonfigurovány filtry frameworku.

Apache Shiro má podobně, jako Spring Security, také anotace, které lze implementovat přímo do ovladačů a jejích metod v MVC frameworku, bohužel ale tyto anotace nefungovaly a Apache Shiro nikde nemá implementaci těchto anotací k MVC frameworku Struts 2, oproti například implementace k frameworku Spring.

Přihlašování uživatele se následně provádí ve třídě akcí LoginAction, v metodě doLogin(), ve které se vytvoří instance třídy UserNamePasswordToken, do které se vloží přihlašovací jméno a heslo uživatele. Následně se vytvoří instance třídy Subject, v této metodě pojmenovaná, jako currentUser, prostřednictvím třídy Shira SecurityUtils.

Následně se v instanci currentUser zavolá metoda login(), do které se vloží token, který byl předtím vytvořen s přihlašovacími údaji uživatele. Tato metoda následně provede přihlášení uživatele. V případě neúspěchu tato metoda vyhodí různé výjimky, například výjimku pro chybně vložené heslo.

Odhlašování poté provádí metoda logout() ve stejné třídě, která získá přihlášeného uživatele, jako instanci třídy Subject od frameworku a následně zavolá metodu instance logout(), která uživatele odhlásí.

Údaje přihlášeného uživatele lze v aplikaci získat, jako instanci třídy Subject, prostřednictvím metody getSubject() třídy SecurityUtils. Následně z této instance lze získat přihlašovací jméno uživatele metodou getPrincipal() u instance Subjectu a následně lze přihlašovacím jménem například získat data uživatele z databáze.

4.3.3 JAAS

JAAS je implementován v projektu ZtratyANalezyJAAS. Framework je obsažen v Javě, v balíčku „javax.security“.

Přihlašování je v Javě implementováno v balíčku cz.uhk.loginModule, ve třídě LoginModuleImplementation, která implementuje rozhraní JAASu LoginModule. Třída z tohoto rozhraní implementuje jeho 5 metod, což je metoda initialize, která podobně, jako konstruktor, inicializuje proměnné v instanci třídy.

Podstatné proměnné, které tato metoda nastavuje, je instance třídy Subject, která reprezentuje informace o uživateli a instance rozhraní CallbackHandler, která se stará o získávání přihlašovacích údajů od uživatele, například při přihlášení.

Další implementovanou metodou je metoda login(). Tato metoda je zavolána frameworkem v případě přihlašování uživatele. V metodě jsou získány prostřednictvím tříd NameCallback a PasswordCallback přihlašovací jméno a heslo uživatele. Prostřednictvím UserDao třídy je pak následně na základě přihlašovacího jméno získána instance třídy User z databáze s informacemi o uživateli. Následně jsou porovnány zadané přihlašovací údaje přijaté od uživatele s těmi uloženými v databázi. Heslo je porovnáno prostřednictvím třídy BCrypt ze Spring Security.

V případě úspěšného ověření přihlašovacích údajů metoda vrátí hodnotu booleanu true a framework následně zavolá metodu „commit()“. V případě neúspěchu přihlášení uživatele však musí metoda vyhodit výjimku. Implementace využívá výjimky od JAASu.

Metoda commit() následně po úspěšném ověření uživatele v předchozí metodě uloží údaje uživatele ve formě tzv. principálů. Aplikace využívá dva principály, a to UserPrincipal přímo od JAAS, do kterého je uloženo přihlašovací jméno uživatele. A vlastní implementaci ve třídě RolePrincipal, do které je uložen název role přihlášeného uživatele. Po uložení uživatelských údajů metoda commit() vrací boolean, v hodnotě true. Pro zajímavost, tato třída se nachází v balíčku „cz.uhk.principal“ a implementuje rozhraní Principal. Z něj implementuje metodu getName(), která musí vracet proměnnou názvu principálu, k tomu musí obsahovat danou proměnnou ve formě textového řetězce, kterou inicializuje ve svém konstruktoru.

Po dokončení přihlašovacího úspěchu je frameworkem zavolána metoda abort(), která se stará o čištění dat, vzniklých během přihlašování. Jelikož většina proměnných vzniklých pro přihlášení je vytvořena přímo v předchozích metodách, tak se o toto čištění postará Garbage collector, a tudíž tato metoda není v implementaci využita. Po dokončení vrací boolean v hodnotě false.

V případě odhlašování uživatele je frameworkem zavolána poslední metoda, kterou implementuje, logout(). Tato metoda vymaže všechny údaje přihlášeného uživatele z paměti a vrací boolean v hodnotě true.

Přihlašování uživatele v běžící aplikaci provádí přímo JAAS, nezávisle na MVC frameworku. Přihlašování uživatele je implementováno v pohledu login.jsp. Formulář v pohledu odkazuje na akci „j_security_check“. Parametr přihlašovacího jména musí být pojmenován, jako „j_username“ a hesla „j_password“. Třida ovladače LoginAction pouze obstarává přihlášení v případě chyby, a to v metodě akce „loginError()“.

Odhlašování uživatele je ve stejné třídě provedeno metodou logout(). K odhlášení se musí v rámci instance rozhraní HttpServletRequest zavolat metoda „getSession().invalidate“, kterou se ukončí platnost sezení a framework JAAS na to následně zareaguje zavoláním metody odhlášení uživatele v implementovaném přihlašovacím modulu. V rámci MVC frameworku Struts 2 HttpServletRequest byl získán s implementací rozhraní „ServletRequestAware“, ze kterého se následně implementovala metoda „setServletRequest“, která nastavuje HttpServletRequest, jako proměnnou třídy.

K údajům přihlášeného uživatele se v aplikaci přistupuje prostřednictvím jeho přihlašovacího jména, které je opět získáno prostřednictvím HttpServletRequest, a to jeho metodou getRemoteUser().

Samotná konfigurace frameworku JAAS mimo Javu je komplikovaná. Pro implementaci vlastního přihlašovacího modulu musí být ve složce „src/main/resources“ vytvořen soubor „jaas.config“, ve kterém je přihlašovací modul nakonfigurován. Aby ale framework používal tento soubor, tak musí být externě nakonfigurován. V případě této aplikace ve spouštěcích konfiguracích serveru „Apache Tomcat v9.0.37 JAAS“, na kterém je aplikace spuštěna, musel být přidán argument virtuálního stroje při spuštění „Djava.security.auth.login.config="C:\Users\jmenoUzivatele\workspace\ZtratyANalezyJAAS\src\main\resources\jaas.config"“, kterým se nastaví přesná pozice daného souboru.

Následně v souboru context.xml, ve složce src/main/webapp/META-INF/ je nakonfigurován JAASRealm servletu Apache Tomcat, který tato aplikace využívá. Přesná lokace tohoto realmu je „org.apache.catalina.realm.JAASRealm". Také zde musí být vloženo jméno aplikace stejné, jako v konfiguraci JAASu, „JaasLogin“. Následně pro tento realm musí být nakonfigurovány třídy, principály, pro uživatelské jméno a název role.

V souboru web.xml ve složce src/main/webapp/WEB-INF jsou nastavena přístupová oprávnění pro uživatele k jednotlivým URL částem aplikace. Také je zde nastaveno URL akce zobrazující přihlašovací pohled a akci v případě chyby při přihlašování.

JAAS neobsahuje tagy pro ověření role a autentizace uživatele v jsp pohledu, a proto musela tato ověření být napsána v pohledech ručně.

4.3.4 OACC

OACC je implementován v projektu ZtratyANalezyOACC. Protřednictvím Mavenu byl stažen artefakt acciente-oacc, ze skupiny com.acciente.oacc.

Jelikož OACC využívá vlastní schéma databáze s vlastními daty, tak před samotnou implementací frameworku bylo nutné implementovat toto schéma do databáze aplikace, v případě této práce do databáze MySQL. K tomu autoři frameworku vytvořili skripty, které se postarají o vytvoření schématu v databázi „oaccdb“, jeho tabulek a následného uživatele databáze. (35) Do skriptu vytvářejícím uživatele databáze,“create_user.sql“, je nutné doplnit vlastní heslo, jakým se uživatel přihlašuje k databázi.

Po vytvoření požadovaného schématu s daty a uživatelem v databázi se následně musí framework dále inicializovat prostřednictvím třídy
„SQLAccessControlSystemInitializer“ v Javě. Tato inicializace je implementována ve třídě OACCSystemInit v balíčku cz.uhk.oaccInit. U této třídy se počítá, že bude využita pouze jednou, při zavádění OACC frameworku.

První metodou v této třídě je metoda „initializeOACC“, která inicializuje framework metodou „.initializeOACC()“. Tato metoda očekává URL schématu databáze oaccdb, uživatelské jméno a heslo dříve skripty vytvořeného databázového uživatele v rámci tohoto schématu, poté název schématu, který je zde nastaven na hodnotu null, jelikož ke schématu oaccdb se už aplikace připojuje, poté heslo tzv. superuživatele a šifrovač hesla, kde je použit hashovací algoritmus BCrypt.

V dalších metodách je využívána instance rozhraní AccessControlContext, se jménem accessControlContext, v rámci, které framework přistupuje ke svým datům a ukládá si v ní potřebné údaje, jako například přihlášeného uživatele.

Prostřednictvím této instance jsou v metodě createSerouceClassessAndDomains vytvořeny prostředkové třídy reprezentující role uživatelů. Metoda createResourceClass očekává název dané prostředkové třídy, poté boolean, zda daná třída reprezentuje uživatele a zda daná třída může být vytvořena i uživatelem, který není autentizován. V metodě je také vytvořena doména ZtratyANalezyDomain.

V rámci této metody se aplikace autentizuje, jako systémový superuživatel, který byl vytvořen v rámci inicializace frameworku a který má přístup ke všem funkcím frameworku.

Další metodou v této třídě je migrateUsersToOACCSchema, což je spíše v rámci implementace v této aplikaci snaha jednoduše převést všechny uživatele v databázi ze schématu aplikace do schématu frameworku. Jak lze v implementaci vidět, je to spíše jenom narychlo udělané převedení účtů pro funkčnost frameworku, ale normálně by tato migrace musela být udělána přesněji, například v případě hesel by musela řešit problém přehashování hesel tak, aby byla kompatibilní s tímto frameworkem.

Další třídou implementující framework je třída OACCDao v balíčku cz.uhk.oaccDao. Tato třída vytváří JDBC spojení se schématem databáze frameworku, které poté využívá při vytváření instance rozhraní AccessControlContext. Tuto instanci poté může metodou get předat jiné třídě, která s ní pracuje.

Přihlašování uživatele je následně implementováno ovladačem metodou doLogin v ovladači LoginAction. Zde se nejdříve třídou OACCDao získá nová instance accessControlContext. V této instanci se následně zavolá metoda authenticate, které se, jako parametry, nastaví prostředek uživatele, jehož instance se získá prostřednictvím přihlašovacího jména a heslo uživatele. Metoda autentizace může při selhání přihlášení uživatele vrátit vyjímky. Ty jsou zachyceny v blocích try/catch. Následně je tato instance, s přihlášeným uživatelem, uložena do sezení aplikace.

Sezení je v rámci MVC frameworku Struts 2 implementováno prostřednictvím rozhraní SessionAware, které musí implementovat každá ovládací třída pracující se sezením. Sezení je následně v aplikaci definováno, jako instance mapy se jménem userSession, a to s dvojící mapování ve formě textového řetězce, který složí, jako název, a samotného objektu.

Instance userSession se v případě potřeby automaticky nastavuje implementovanou metodu setSession.

Odhlášení uživatele je provedeno v metodě logout, která získá accessControlContext ze sezení, metodou unauthenticate uživatele odhlásí a poté jsou všechna data ze sezení userSession vymazána.

V případě registrace uživatele, prováděné v metodě registration, je nutné kromě vlastního schématu nového uživatele také vytvořit v rámci schématu OACC frameworku. Prostřednictvím nejdříve metody getInstance, ze třídy frameworku Resources, je prostřednictvím přihlašovacího jména uživatele pokus o získání prostředku uživatele z databáze pro ověření, zda už uživatel s daným přihlašovacím jménem existuje v databázi, nebo ne. Ověření se provádí na primárním klíči daného prostředku, který je v longu. V případě existence účtu se stejným přihlašovacím jménem je uživatel vrácen zpátky do registračního formuláře a musí zadat jiné přihlašovací jméno.

Poté prostřednictvím accessControlContext, metody createResource je uživatel vytvořen v databázi frameworku, jako nový prostředek, s názvem ROLE_USER neboli jeho rolí obyčejného uživatele, dále v doméně ZtratyANalezyDomain a s nastaveným přihlašovacím jménem a heslem, jaké si uživatel nastavil.

K přihlášenému uživateli se v databázi přistupuje prostřednictvím instance accessControlContext získané ze sezení, ze kterého získává jeho přihlašovací jméno kombinací metod getAuthenticatedResource, která získá prostředek přihlášeného uživatele a následně metodou getExternalId získá dané přihlašovací jméno uživatele.

Při změně uživatelského hesla se musí v accessControlContext zavolat metoda setCredentials, které se jako parametry předává autentizovaný prostředek neboli přihlášený uživatel a poté nové

heslo uživatele. Bohužel framework OACC nepodporuje změnu přihlašovacího jména, jelikož je neměnné neboli tzv. immutable.

Framework dále nepodporuje nastavení omezení přístupu k URL aplikace, a tudíž ověření přihlášení role uživatele a jeho přihlášení musí být uděláno ručně. V případě samotného přihlášeného uživatele stačí ověřit, zda je accessControlContext uložený v sezení.

Název role uživatele se získává v accessControlContextu metodami „getResourceClassInfoByResource.getResourceClassName()“. Pro zajímavost, v rámci tohoto projektu je v případě nedostačující role uživatele ve frameworku Struts 2 použit typ výsledku httpheader, který je v tomto případě nastaven, aby vracel uživateli http chybu 403 forbidden.

Framework neobsahuje tagy pro ověření role a autentizace uživatele, a proto tato ověření musela být také napsána v pohledech ručně.

4.3.5 Porovnání technologií zabezpečení

Kromě OACC, která vyžaduje svoji specifickou implementaci, tak ostatní technologie převážně obsahují rozhraní, která se v aplikaci následně implementují podle jejích potřeb, případně se použijí výchozí implementace daného frameworku.

Spring Security je nejvhodnější pro aplikaci s implementovanými technologiemi Springu, konkrétně například jeho MVC frameworku, jelikož je s ním kompatibilní a lze v případě nastavení omezení k přístupu k částem aplikace použít anotace, což zjednodušuje práci,
a hlavně lze přímo ve ovládacích třídách přehledně vidět, jací uživatelé mají přístup k daným částem aplikace.

V případě implementace jiných technologií, jako například MVC frameworku, v případě této aplikace, v jejích verzích, kde byl implementován MVC framework Struts 2, je mnohem lepší Apache Shiro, jelikož v případě nefunkčnosti svých anotací má mnohem jednodušší a přehlednější nastavení autorizace než Spring Security, ve kterém se autorizace uživatele musí nastavovat složitě prostřednictvím spousty metod, pokud se proto nepoužívají anotace.

Nejhorším frameworkem v případě implementace je framework JAAS, jelikož v případě jeho implementace pro webovou aplikaci je závislý na běhovém prostředí, jako například Apache Tomcat. A hlavně přihlašovací modul se zde musí také nastavovat externě, nikoliv přímo v Javě nebo konfiguračních souborech, oproti ostatním frameworkům.

Framework OACC oproti ostatním frameworkům nepodporuje jednoduché nastavení autorizace ve webové aplikaci. Tento framework navíc využívá svoje vlastní schéma databáze, což je zbytečné pro aplikaci, která pro veškeré údaje o uživateli obsahuje svoji vlastní databázi, a tudíž není pro aplikaci, ve které byl implementován, vhodná volba.

5 Shrnutí výsledků

V průběhu bakalářské práce byl rozpoznán rozdíl mezi frontendem a backendem, následně byla poznána Java a prozkoumána Java EE, hlavně její API technologie, z nichž se některé budou dát porovnávat z konkrétními backendovými technologiemi. Například technologie pracující s databází se budou dát porovnávat s technologií Hibernate, JDO apod. Také byla zanalyzována část vybraných technologií.

Následně byly vybrané technologie implementovány v různých verzích aplikace zabývající se evidencí ztrát a nálezů v budově firmy. Implementace byly následně v bakalářské práci popsány a podobné technologie, jako například MVC frameworky, databázové frameworky apod., byly mezi sebou porovnány.

Na konci každého srovnání byla vybrána nejvhodnější technologie pro teoretickou implementaci finální verze aplikace. Jedná se konkrétně o technologie Spring Web a Hibernate, jako ORM framework. V případě technologie autorizace a autentizace je to však složitější. Jelikož k aplikaci implementující Spring Web se nejlépe hodí Spring Security a k aplikaci implementující, jako MVC framework, Struts 2, se nejlépe hodí Apache Shiro.

6 Závěry a doporučení

V této bakalářské práci byla prozkoumána problematika backendových technologií. Oproti očekáváním, že nejlépe se budou implementovat technologie Springu a Hibernate, tak je zajímavé, že z technologií zabezpečení se dobře implementoval také Apache Shiro.
A DataNucleus v případě jeho JDO implementace.

Je nutné vzít v úvahu, že srovnávané backendové technologie byly hodnoceny na základě implementace do webové aplikace Ztrát a nálezů. Pokud by byly implementovány v jiném druhu aplikace, tak by mohla implementace a porovnání technologií mohlo dopadnout jinak. Například podle jiných potřeb aplikace. Zvlášť se to týká technologií, jako je například Apache Shiro, které mají své specifické funkce pouze pro určitý typ aplikace, jako je například webová aplikace.

V budoucnu, pokud by se v tématu této bakalářské práce dále pokračovalo, tak by bylo zajímavé porovnat backendové technologie zaměřující se na jiné věci, než ty, které v této práci byly již porovnány. Případně by se mohly porovnané technologie hlouběji prozkoumat. V případě implementace nových technologií je však nutné brát v úvahu jejích vzájemnou kompatibilitu.

7 Použitá literatura

1. **Raghavendra, G. a Pushpanjali, P.** A Review on Java Frameworks for Web Applications. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology.* 2018, Sv. III, 3.

2. **Dhingra, Neha, Abdelmoghith, Emad a Mouftah, Hussien T.** Review on JPA Based ORM Data Persistence Framework. *International Journal of Computer Theory and Engineering.* Říjen 2017, Sv. 9, 5.

3. *Espresso: Brewing Java For More Non-Volatility with Non-volatile Memory.* **Wu, Mingyu, a další.** New York : Association for Computing Machinery, 2018. ASPLOS '18: Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems. stránky 70-83. 978-1-4503-4911-6.

4. **Sciore, Edward.** JDBC. *Database Design and Implementation.* Cham : Springer, 2020, stránky 15-47.

5. **Liu, Gang, a další.** Extended Role-Based Access Control with Context-Based Role Filtering. *KSII Transactions on Internet and Information Systems.* Měsíc, 31. Březen 2020, Sv. 14, 3, stránky 1263-1279.

6. **Partheepan, Shouthiri a Sivalingam, Disne.** SOLUTION TO CUBIC EQUATION USING JAVA PROGRAMMING. *European Journal of Mathematics and Computer Science.* Deník, 2020, Sv. 7, 1, stránky 20-28.

7. **Soto-Valero, César, a další.** *A Comprehensive Study of Bloated Dependencies in the Maven Ecosystem.* Department of Software and Computer Systems, KTH Royal Institute of Technology. Stockholm : ArXiv, 2020. Manuscript.

8. **Ohyver, Margaretha, a další.** The Comparison Firebase Realtime Database and MySQL Database Performance using Wilcoxon Signed-Rank Test. *Procedia Computer Science.* Deník, 2019, Sv. 157.

9. **Oracle.** Java Timeline. *Oracle.* [Online] Oracle, 2019. [Citace: 3. Únor 2019.] http://oracle.com.edgesuite.net/timeline/java/.

10. —. Java SE at a Glance. *Oracle.* [Online] Oracle, 2019. [Citace: 7. Únor 2019.] https://www.oracle.com/technetwork/java/javase/overview/index.html.

11. —. Java™ EE at a Glance. *Oracle.* [Online] Oracle, 2017. [Citace: 7. Únor 2019.] https://www.oracle.com/technetwork/java/javaee/overview/index.html.

12. —. *Java Platform, Enterprise Edition (Java EE) 8 The Java EE Tutorial.* [Html stránky] místo neznámé : Oracle, 2017.

13. **MuleSoft LLC.** What is an API? (Application Programming Interface). *MuleSoft.* [Online] MuleSoft LLC, 2020. [Citace: 21. Červenec 2020.] https://www.mulesoft.com/resources/api/what-is-an-api.

14. **LifeXasy.** Spring Framework - Architecture. *Lifexasy.* [Online] LifeXasy, 2019. [Citace: 9. Únor 2019.] http://lifexasy.com/Spring-Framework-Architecture.html.

15. **Muthuraman, Meyyappan.** Struts MVC Architecture Tutorial. *DZone.* [Online] DZone, 13. Červen 2012. [Citace: 9. Únor 2019.] https://dzone.com/tutorials/java/struts/struts-tutorial/struts-mvc-architecture-tutorial.html.

16. **Tutorials Point.** Tutorials point. *MYBATIS Overview.* [Online] Tutorials point, 2019. [Citace: 9. Únor 2019.] https://www.tutorialspoint.com/mybatis/mybatis_overview.htm.

17. **The Apache Foundation – JDO Java Data Objects.** About Apache JDO. *JDO Java Data Objects.* [Online] JDO Java Data Objects, 2015. [Citace: 9. Únor 2019.] http://db.apache.org/jdo/index.html.

18. **Codecademy.** What is REST? *codecademy.* [Online] Codecademy, 2020. [Citace: 21. Červenec 2020.] https://www.codecademy.com/articles/what-is-rest.

19. **DataNucleus.** DataNucleus Products. *DataNucleus.* [Online] DataNucleus, 2020. [Citace: 21. Červenec 2020.] http://www.datanucleus.org/documentation/products.html.

20. **Sharpened Productions.** RDBMS. *TechTerms.* [Online] Sharpened Productions, 16. Prosinec 2017. [Citace: 21. Červenec 2020.] https://techterms.com/definition/rdbms.

21. **Oracle.** What Is a URL? *The Java™ Tutorials.* [Online] Oracle, 2019. [Citace: 22. Červenec 2020.] https://docs.oracle.com/javase/tutorial/networking/urls/definition.html.

22. **baeldung.** Guide To The Java Authentication And Authorization Service (JAAS). *Baeldung.* [Online] Baeldung SRL., 14. Březen 2020. [Citace: 23. Červenec 2020.] https://www.baeldung.com/java-authentication-authorization-service.

23. **Acciente, LLC.** Documentation Features. *OACC.* [Online] Acciente, LLC., 2018. [Citace: 23. Červenec 2020.] http://oaccframework.org/oacc-features.html.

24. **Eclipse Foundation, Inc.** Eclipse IDE for Enterprise Java Developers. *Eclipse foundation.* [Online] Eclipse IDE for Enterprise Java Developers, 2020. [Citace: 25. Červenec 2020.] https://www.eclipse.org/downloads/packages/release/2020-09/m1/eclipse-ide-enterprise-java-developers.

25. **The Apache Software Foundation .** Apache Tomcat. *Apache Tomcat®.* [Online] The Apache Software Foundation , 2020. [Citace: 25. Červenec 2020.] http://tomcat.apache.org/.

26. **baeldung.** How to Deploy a WAR File to Tomcat. *Baeldung.* [Online] Baeldung SRL., 12. Únor 2020. [Citace: 26. Červenec 2020.] https://www.baeldung.com/tomcat-deploy-war.

27. **Buckle, Anne a Hocken, Vigdis.** UTC – The World's Time Standard. *timeanddate.com.* [Online] Time and Date AS, 2020. [Citace: 26. Červenec 2020.] https://www.timeanddate.com/time/aboututc.html.

28. **VMware, Inc.** Spring Boot. *Spring.* [Online] VMware, Inc., 2020. [Citace: 7. Srpen 2020.] https://spring.io/projects/spring-boot.

29. **The Apache Software Foundation.** Interceptors. *STRUTS.* [Online] SoftwareMill, 2018. [Citace: 28. Červenec 2020.] https://struts.apache.org/core-developers/interceptors.html.

30. **Mihalcea, Vlad, a další.** Hibernate. *Hibernate ORM 5.4.19.Final User Guide.* [Online] 27. Červenec 2020. [Citace: 29. Červenec 2020.] https://docs.jboss.org/hibernate/orm/5.4/userguide/html_single/Hibernate_User_Guide.html#hql.

31. **DataNucleus.** JDO Mapping Guide (v5.2). *DataNucleus v5.2.* [Online] DataNucleus, 2020. [Citace: 30. Červenec 2020.] http://www.datanucleus.org/products/accessplatform/jdo/mapping.html#application_identity.

32. —. JDO Enhancement Guide (v5.2). *DataNucleus v5.2.* [Online] DataNucleus, 2020. [Citace: 30. Červenec 2020.] http://www.datanucleus.org/products/accessplatform/jdo/enhancer.html.

33. —. JDO Tools Guide (v5.2). *DataNucleus v5.2.* [Online] DataNucleus, 2020. [Citace: 30. Červenec 2020.] http://www.datanucleus.org/products/accessplatform/jdo/tools.html#eclipse.

34. **Oracle.** Interface ResultSet. *Java® Platform, Standard Edition & Java Development Kit Version 14 API Specification.* [Online] Oracle, 2020. [Citace: 1. Srpen 2020.] https://docs.oracle.com/en/java/javase/14/docs/api/java.sql/java/sql/ResultSet.html#close().

35. **Acciente, LLC.** Getting Started Tutorial. *OACC by acciente.* [Online] Acciente, LLC., 2018. [Citace: 3. Srpen 2018.] http://oaccframework.org/getting-started-tutorial.html.

36. **JavaJee.com.** Introduction and History of the Spring Framework. *JavaJee.com.* [Online] JavaJee.com, 2018. [Citace: 5. Únor 2019.] https://javajee.com/introduction-and-history-of-the-spring-framework.

37. **NHibernate Community.** Home. *NHibernate.* [Online] NHibernate Community, 2019. [Citace: 5. Únor 2019.] https://nhibernate.info/.

38. **Oracle.** Chapter 22 Enterprise Beans. *The Java EE 6 Tutorial.* [Online] Oracle, 2013. [Citace: 6. Únor 2019.] https://docs.oracle.com/javaee/6/tutorial/doc/gijsz.html.

39. **Ligios, Andrea.** Blade – A Complete Guidebook. *Baeldung.* [Online] Baeldung SRL., 6. Září 2019. [Citace: 11. 1 2020.] https://www.baeldung.com/blade.

40. **Hunt, John.** Play Framework. *A Beginner's Guide to Scala, Object Orientation and Functional Programming.* Cham : Springer, 2018.

41. **Lightbend.** Accessing an SQL database. *Play Framework.* [Online] 2018. [Citace: 18. Únor 2020.] https://www.playframework.com/documentation/2.8.x/AccessingAnSQLDatabase.

42. **Tutorials Point.** Apache Tapestry - Overview. *tutorialspoint.* [Online] Tutorials Point, 2020. [Citace: 26. Únor 2020.] https://www.tutorialspoint.com/apache_tapestry/apache_tapestry_overview.htm.

43. —. Apache Tapestry - Architecture. *tutorialspoint.* [Online] Tutorials Point, 2020. [Citace: 26. Únor 2020.] https://www.tutorialspoint.com/apache_tapestry/apache_tapestry_architecture.htm.

44. **The Apache Software Foundation.** Wicket 9.x Reference Guide. *Apache Wicket 8.* [Online] The Apache Software Foundation, 21. Únor 2020. [Citace: 7. Březen 2020.] https://ci.apache.org/projects/wicket/guide/9.x/single.html.

45. **baeldung.** The Spring Boot Starter Parent. *Baeldung.* [Online] Baeldung SRL., 20. Květen 2020. [Citace: 28. Červenec 2020.] https://www.baeldung.com/spring-boot-starter-parent.

8 Přílohy

8.1 Zkomprimovaný soubor

1) ProjektyASchemaDatabaze.zip

8.2 Projekty Javy ve zkomprimovaném souboru

1) ZtratyANalezy
2) ZtratyANalezyDataNucleusJDO
3) ZtratyANalezyDataNucleusJPA
4) ZtratyANalezyHibernateJPA
5) ZtratyANalezyJAAS
6) ZtratyANalezyJDBC
7) ZtratyANalezyMyBatis
8) ZtratyANalezyOACC
9) ZtratyANalezyStruts

8.3 Schéma MySQL databáze ve zkomprimovaném souboru

1) ZalohaDatabaze.sql

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor:	**Artur Hamza**
Studium:	I1600532
Studijní program:	B1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název bakalářské práce:	**Backendové technologie**
Název bakalářské práce AJ:	Backend technologies

Cíl, metody, literatura, předpoklady:

Cíl práce: Práce se zabývá backendovými technologiemi v Javě. Cílem je vybrané technologie popsat, implementovat je v různých variantách aplikace a následně je porovnat.

Osnova:

1. Úvod
2. Cíle práce
3. Popis backendových technologií
4. Popis aplikace a jejích základních technologií
5. Implementace vybraných technologií a jejích porovnání
6. Závěr
7. Literatura

Raghavendra, G. a Pushpanjali, P. A Review on Java Frameworks for Web Applications. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology.* 2018, Sv. III, 3.

Dhingra, Neha, Abdelmoghith, Emad a Mouftah, Hussien T. Review on JPA Based ORM Data Persistence Framework. *International Journal of Computer Theory and Engineering.* Říjen 2017, Sv. 9, 5.

Sciore, Edward. JDBC. *Database Design and Implementation.* Cham : Springer, 2020, stránky 15-47.

Garantující pracoviště:	Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu
Vedoucí práce:	doc. Ing. Filip Malý, Ph.D.
Oponent:	prof. RNDr. PhDr. Antonín Slabý, CSc.
Datum zadání závěrečné práce:	14.1.2015