Java frameworky a vývoj rozsáhlých web aplikací pro podnikovou sféru

Java frameworks and enterprise-grade applications development

Bc. Martin Mihál

Diplomová práce
2011

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2010/2011

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Martin MIHÁL**
Osobní číslo:	**A09710**
Studijní program:	**N 3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Téma práce:	**Java frameworky a vývoj rozsáhlých web aplikací pro podnikovou sféru**

Zásady pro vypracování:

1. **Prostudujte architekturu Java EE 6.**
2. **Na příkladu vývoje menší aplikace (max. 20 tabulek, z toho 30% s vazbami M:N, víceúrovňová uživatelská práva) srovnejte standardní komponenty Java EE 6 s alternativními Java frameworky, např. s těmito: http://en.wikipedia.org/wiki/Comparison_of_web_application_frameworksJava.**
3. **Použijte srovnávací kritéria: rychlost vývoje aplikace, bezpečnost – ošetření SQL injection, XSS a dalším běžným útokům, podpora víceúrovňové autorizace, zpětná kompatibilita API (kritérium velmi důležité a zároveň velmi pracně dohledatelné), možnost definice constraints, které framework dle použité databáze implementuje buď přímo v DB, nebo v modelu, nezávislost logického a fyzického modelu (možnost přejmenovat tabulky i sloupce v DB bez nutnosti přepisovat kód), časová/paměťová složitost vybraných operací, kvalita dokumentace.**

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

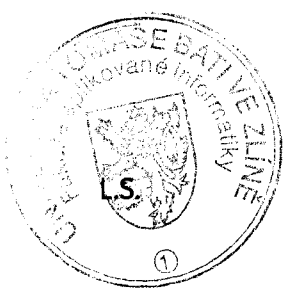
1. **TAYLOR, Art; BUEGE, Brian; LAYMAN, Randy. Hacking bez tajemství: Java a J2EE. vydání první. Brno : Computer Press, 2003. 440 s. ISBN 80-7226-868-6.**
2. **BÖCK, Heiko. Platforma NetBeans : Podrobný průvodce programátora. vydání první. Brno : Computer Press, 2010. 320 s. ISBN 978-80-251-3116-9.**
3. **BURNS, Ed; SCHALK, Chris. JavaServer Faces 2.0 : The Complete Reference. [s.l.] : McGraw-Hill Companies, 2010. 752 s. ISBN 978-0-07-162510-4.**
4. **MULARIEN, Peter. Spring Security 3. First published. Birmingham : Packt Publishing, 2010. 396 s. ISBN 978-1-847199-74-4.**
5. **GONCALVES, Antonio. Beginning Java EE 6 Platform with GlassFish 3. 2nd Edition. [s.l.] : Springer, 2010. 537 s. ISBN 978-1-4302-2890-5.**
6. **KUMMEL, Bart. Apache MyFaces 1.2 Web Application Development. [s.l.] : Packt Publishing, 2010. 408 s. ISBN 978-1-847193-25-4.**

Vedoucí diplomové práce: **Ing. Tomáš Dulík**
Ústav informatiky a umělé inteligence

Datum zadání diplomové práce: **24. února 2011**

Termín odevzdání diplomové práce: **18. května 2011**

Ve Zlíně dne 24. února 2011



prof. Ing. Vladimír Vašek, CSc.
děkan

doc. Mgr. Roman Jašek, Ph.D.
ředitel ústavu

ABSTRAKT

Cieľom práce je porovnať štandardné vývojové nástroje Java EE 6 s alternatívnymi frameworkami tejto platformy na základe stanovených kritérií. Hľadajú sa riešenia spĺňajúce tieto požiadavky v jednotlivých frameworkoch, ktoré sú demonštrované na praktických príkladoch. Následne je vykonaný záťažový test, preverujúci časové a pamäťové nároky jednotlivých frameworkov. V závere práce je predstavená webová aplikácia, demonštrujúca praktické použitie týchto nástrojov.

Kľúčové slová: Java EE, MVC, JSF, Spring, Struts, Stripes, framework.

ABSTRACT

The aim of dissertation is to compare standard developmental implements Java EE 6 with the alternative frameworks of this platform on the basis of given standards. The solutions, which are verifying these standards in individual frameworks are demonstrated on practical models and are in the process of searching. Consequently is carried out a load test verifying time and memory claims of individual frameworks. At the end of dissertation the web application is presented and this application demonstrates the practical use of these implements.

Keywords: Java EE, MVC, JSF, Spring, Struts, Stripes, framework

Poďakovanie

Ďakujem vedúcemu diplomovej práce, Ing. Tomášovi Dulíkovi, za poskytnutie cenných rád a osobný prístup pri komunikácii a vedení diplomovej práce.

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně

…….

podpis diplomanta

OBSAH

ÚVOD

Ak by sme sa sami seba spýtali, čo považujeme za technický fenomén súčasnosti, mnohých z nás by bez pochyby ako prvé napadlo slovo internet, ktorý za krátku dobu vyrástol zo skromného zdroja informácii do najväčšieho komunikačného prostriedku v histórii ľudstva. Jednoduchou odpoveďou na otázku, ako je možné, že za tak krátku dobu svojej existencie prekonal internet svojim rozsahom v každom smere všetky doterajšie formy komunikácie spoločnosti, akými sú listová zásielka, telefón, ale aj rozhlas a televízia, by bola jeho interaktivita.

Technické zabezpečenie týchto internetových služieb kladie vysoké nároky na objemnosť a zložitosť prostriedkov, pomocou ktorých vznikajú. To si vyžaduje existenciu vývojových nástrojov pre konkrétne programovacie jazyky, ktorých úlohou je poskytnúť hotové riešenia pre rutinné a režijné činnosti aplikácie. Medzi takéto nástroje patria takzvané frameworky (aplikačné rámce), ktorých kvalita v zmysle efektívnosti, spoľahlivosti, jednoduchosti a zároveň robustnosti, zvyšuje u vývojárov popularitu a uplatniteľnosť tých programovacích jazykov, pre ktoré sú tieto frameworky určené.

Práca v teoretickej časti predstavuje architektúru JavaEE, popisuje význam a úlohu frameworkov vo všeobecnej rovine. Následne sú jednotlivo predstavené vybrané frameworky a ich spoločná architektúra návrhového vzoru.

Praktická časť je venovaná porovnaniu vlastností konkrétnych frameworkov. Vzhľadom na rozsah nástrojov a funkcionalít jednotlivých frameworkov sú porovnávané tie vlastnosti, ktoré riešia požiadavky definované v zadaní. Tieto vlastnosti sú prezentované na praktických ukážkach. V závere praktickej časti je predstavená výsledná aplikácia internetového obchodu, zostavená vybranými nástrojmi prezentovaných frameworkov.

S ohľadom na rozsah vlastností nielen vybraných frameworkov, ale aj JavaEE technológie samotnej sú u čitateľa predpokladané základné znalosti tejto technológie a programovacích jazykov Java, XML, HTML, SQL a HTTP protokolu.

I. TEORETICKÁ ČASŤ

1 ARCHITEKTÚRA JAVA EE

1.1 Java EE platforma

Komunikácia všetkých služieb (aplikácii) využívajúcich k prenosu dát infraštruktúru internetu, je založená na architektúre klient-server. *Architektúra klient-server zhromažďuje počítačové služby do centrálnych procesov, ktoré klientom na požiadanie poskytujú prostriedky ako pamäťový priestor, správu databáz, alebo prenos súborov.*[1] Táto architektúra bola uplatňovaná v dátovej komunikácii ešte predtým, než začali vznikať počítačové služby v podobe, v aké ich dnes poznáme. A teda na jej základe vznikali protokoly ako HTTP, FTP, atď., ktoré riešia spôsob komunikácie počítačových služieb na aplikačnej vrstve.

Spočiatku počítačové služby boli poskytované tzv. „bezstavovými" HTTP servermi. To znamená, že počas jedného HTTP spojenia server obdržal požiadavku od klienta, spracoval ju a vrátil odpoveď bez toho, aby si priebežne uchovával údaje o tejto transakcii. Klientske požiadavky vyžadujúce spustenie nejakého programu alebo prácu s databázou obslúžil program CGI (*„Common Gateway Interface"*, protokol prepojujúci externú aplikáciu s HTTP serverom).

Tento spôsob komunikácie bol nepostačujúci, keďže akékoľvek prerušenie spojenia viedlo k ukončeniu transakcie. Programovací jazyk Java ponúkol riešene týchto nedostatkov prostredníctvom technológie servletov, ktoré boli základom pre vznik JavaEE (*„Java Enterprise Edition",* rozsiahla rodina nástrojov určených pre vývoj webových služieb). *Servlety sú odpoveďou technológie Java na programovanie pomocou CGI. Jedná sa o program, ktorý beží na webovom serveri a pôsobí ako stredná vrstva medzi požiadavkou prichádzajúcou z webového prehliadača alebo od iného HTTP klienta a databázou, či aplikáciou na serveri HTTP.*[2]

---

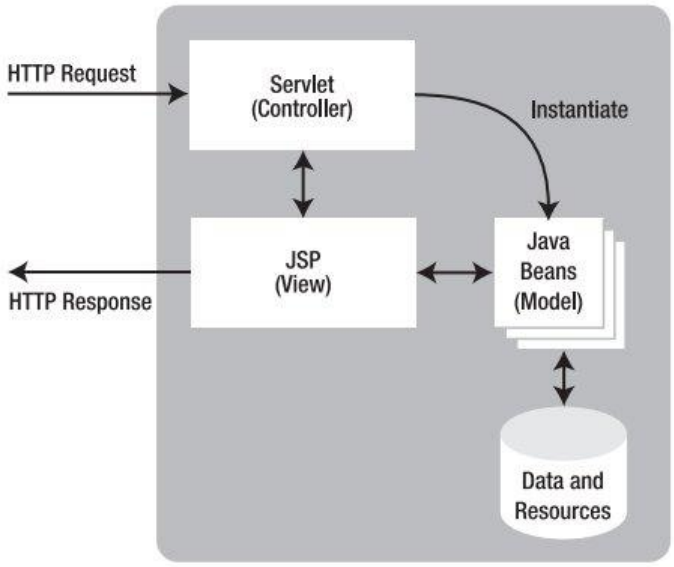
[1] BOLLINGER, Gary; NATARAJAN, Bharathi. *JSP - Java Server Pages : podrobný průvodce začínajícího tvůrce webu.* str. 15

[2] HALL, Marty. *JAVA servlety a stránky JSP.* str. 5

Tvorenie webových stránok len s pomocou servletov je značne neefektívne. Servlety sú určené hlavne k nízkoúrovňovej obsluhe HTTP protokolu. Nadstavbou servletov sú JSP (*„JavaServer Pages“*, textový dokument, podobný HTML, ktorý sa pri prvom volaní prevedie na servlet).

Vkladanie Java kódu do JSP súborov prinieslo veľa pozitív, ale zároveň sa stával kód značne neprehľadným a miešala sa prezentačná logika s aplikačnou.

Lepším riešením, vhodným pre veľké aplikácie, je oddeliť aplikačnú logiku od prezentačných stránok. Toto riešenie prichádza v podobe JSP Model 2 architektúry, taktiež známej ako Model-View-Controller (MVC) návrhový vzor[3].



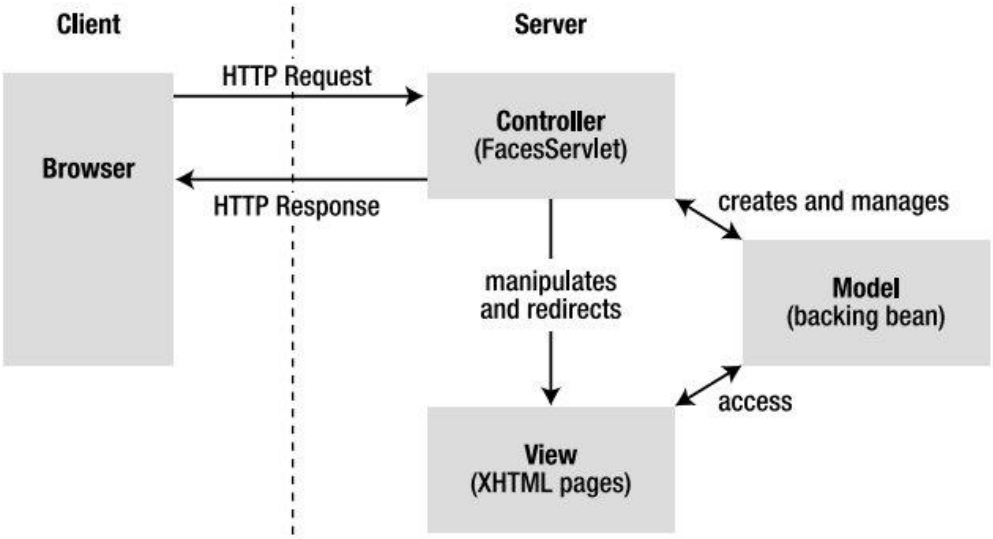
Obr. 1. Architektúra JSP Model 2 [3]

Nová vlastnosť v JSP Model 2 podporila vznik prvých JavaEE frameworkov postavených na MVC návrhovom vzore.

[3] GIULIO, Zambon; SEKLER, Michael. *Beginning JSP, JSF and Tomcat Web Development : From Novice to Professional.* str. 10

1.2 MVC

MVC je architektonický vzor, slúžiaci na oddelenie obchodnej logiky od užívateľského rozhrania. Obchodná logika sa nemieša s kódom užívateľského rozhrania. Ak sú tieto dve rozhrania zmiešané, aplikácia je oveľa ťažšie udržiavateľná a menej škálovateľná.[4]



Obr. 2. MVC návrhový vzor [4]

MVC nie je doménou výhradne webových aplikácii, ale všeobecnou štruktúrou návrhu a tvorby softvéru ako takého.

MVC návrhový vzor sa skladá z troch častí:

- Model – predstavuje dátovú vrstvu, nad ktorou pracuje View
- View – prevádza Model do prezentačnej vrstvy
- Controller – je prostredníkom medzi vrstvami Model a View

Vďaka tomuto rozdeleniu do jednotlivých vrstiev sa predchádza ich vzájomnému vmiešavaniu a tesnej väzbe. Taktiež každý z modelov preberá zodpovednosť za svoju časť práce. Životný cyklus jednej požiadavky vypadá nasledovne. Prichádzajúca požiadavka je

[4] GONCALVES, Antonio. *Beginning Java EE 6 Platform with GlassFish 3 : From Novice to Professional*. str. 345

obslúžená Controllerom (Servlet), ten z Modelu získa potrebné dáta obsiahnuté v požiadavke a odošle ich na zobrazenie časti View.

1.3 Frameworky

Jedným z hlavných kritérií efektívnej práce vo všeobecnosti je neplytvať čas a energiu na činnosti, ktoré už vykonal niekto druhý a navyše tieto riešenia ponúka k širokému využitiu.

Pre obľúbené a široko používané programovacie jazyky často vznikajú programové balíky obsahujúce knižnice, návrhové vzory, prípadne pomocné programy a vývojové postupy nazývané frameworky. Vzhľadom na skutočnosť že JavaEE bezpochyby takýmto jazykom je, tak aj pre túto platformu existujú rôzne frameworky. Úlohou frameworkov je zefektívňovať softvérový vývoj a ponúknuť nástroje, prípadne hotové riešenia na najčastejšie požadované funkcionality, akými sú napríklad práca s databázou, bezpečnosť.

Značný počet takýchto frameworkov je vyvíjaných aj pre platformu JavaEE. Ide prevažne o nástroje rôznych nezávislých výrobcov, ale medzi nimi sa nachádza aj framework JSF, ktorý je vyvíjaný tvorcami samotnej Javy. Medzi ďalšie najrozšírenejšie frameworky, ktoré sú obsiahnuté v tejto práci, patria Spring, Apache Struts a Stripes.

1.3.1 JSF – JavaServer Faces

JSF je komponentovo orientovaný UI framework. JSF je súčasťou Java EE, takže je štandardom tejto platformy a umožňuje vytvárať aplikácie bez nutnosti pridávať do projektu ďalšie knižnice. Podstatou JFS je naplnenie dvoch základných úloh:

1. Generovanie užívateľského rozhrania (najčastejšie doručením HTML odpovede) do prehliadača a zobrazením web stránky ako celku. Toto užívateľské rozhranie je reprezentované na serveri stromom komponent. To znamená že dochádza k tzv. „jedna k jednej“ mapovaniu medzi elementmi v strome komponent a elementmi v užívateľskom rozhraní. Toto oddelené užívateľské rozhranie a strom komponent umožňujú podporu vzájomne odlišným znakovým jazykom (HTML naproti XUL) alebo podporu prehliadačom pre rôzne platformy (desktop naproti smartphone). Taktiež je tým umožnený oddelený vývoj aplikácie, pričom vývoj na prezentačnej vrstve (užívateľské rozhranie) môže prebiehať nezávisle od vývoja na aplikačnej vrstve.

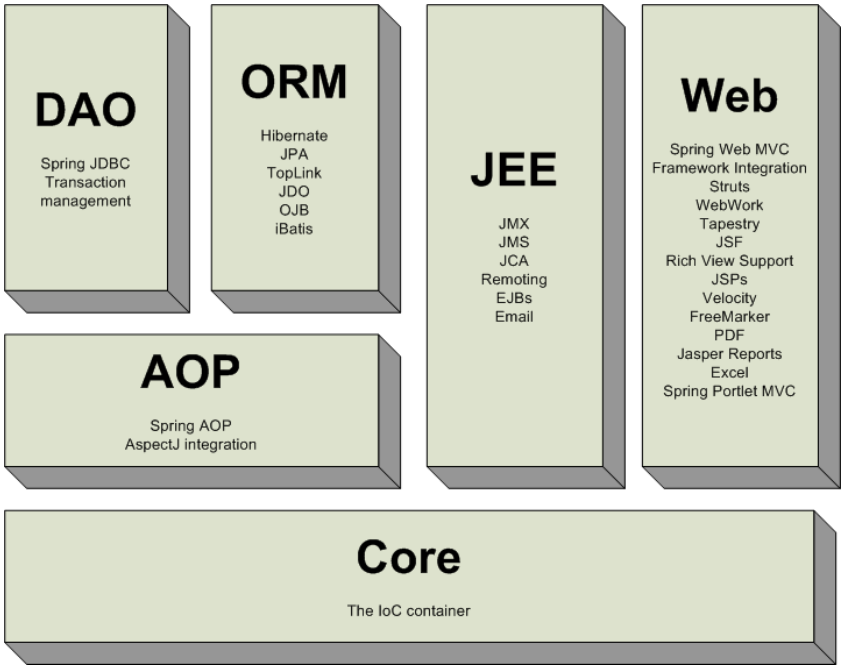
2. Obsluha užívateľom generovanej udalosti (požiadavky) na web stránke volaním server-side listeners, ktoré následne generuje ďalšie zobrazenie web stránky alebo aktualizáciu existujúceho zobrazenia web stránky. V tomto ohľade je JSF udalosťami riadený framework.

JSF framework je založený na MVC návrhovom vzore a v tomto prípade sa to odzrkadľuje na lepšom riadení aplikácie, lebo užívateľské rozhranie (View) je oddelené od aplikačných dát a logiky (Model). Dôraz na obsluhu užívateľského rozhrania ako celku významne uľahčuje prácu s HTTP dotazmi a odpoveďami. Užívateľské rozhranie je definované vlastnými XML značkami, ktorými sa špecifikujú odkazy na Java beans umiestnené v aplikačnom serveri. Všetky užívateľské požiadavky sú obsluhované cez FacesServlet rozhranie (Controller).

V prvej verzii JSF bola prezentačná vrstva tvorená JSP, ale ukázalo sa, že JSP má výrazne odlišnú koncepciu naproti JSF. Životný cyklus JSP je podstatne jednoduchší, keď HTML odpovede na požiadavky zo strany užívateľa sú tvorené v servlete a jeho abstrakciou je JSP súbor. Naproti tomu životný cyklus JSF je podstatne zložitejší (generovanie stromu komponent, validácia užívateľských vstupov...). Táto skutočnosť viedla k vzniku technológie „Facelets“ a je priamo implementovaná v JSF od verzie 2.0. Základom je štandard XML, čo vedie k rýchlemu oboznámeniu sa s technológiou a nevyžaduje sa znalosť JSP a Servlety.

1.3.2 Spring

Spring je rozsiahly framework, pozostávajúci z modulov, vďaka čomu je možné v projekte implementovať len potrebné časti, prípadne ich kombinovať s nástrojmi mimo frameworku. Architektúra Springu pozostáva zo šiestich modulov. Niektoré zdroje uvádzajú sedem modulov, kedy modul „Web“ a modul “Spring Web MVC“ sa uvádzajú zvlášť, keďže modul „Spring Web MVC“ je možné nahradiť aj iným webovým frameworkom (viď. Obr. 3).



Obr. 3. Štruktúra Spring frameworku[5]

Štruktúra Spring framework:

- Core – je fundamentálnou časťou celého frameworku poskytujúci IoC *("Inversion of Control"*, návrhový vzor architektúry softvéru, obdobne ako MVC) a „Dependency Injection“ vlastnosti. Základom modulu je „BeanFactory“, obsluhujúca POJO objekty (*"Plain Old Java Object"*, bežné objekty Javy, ktoré nie sú Java Beany) v celom ich životnom cykle.
- JEE (Context) – Poskytuje podporu pre „BeanFactory“. Je prostredníkom medzi klientským kódom a „BeanFactory“.
- DAO – Poskytuje abstrakciu nad JDBC API, čo odstraňuje zdĺhavé a chybové programovanie opakujúceho sa kódu, pri práci s JDBC API.
- ORM – Zabezpečuje integráciu nástrojov objektovo-relačného modelovania ORM API (*"Object Relational Mapping"*, konverzia dát medzi relačnou databázou a objektovým jazykom). Obsahuje podporu pre JPA, JDO, Hibernate a iBatis.

[5] *Spring Framework, Reference Documentation* [online].

- AOP – Je podporou pre „aspektovo orientované programovanie“, čo znamená možnosť odčleniť časť kódu do tzv. aspektu a následne ho aplikovať na ľubovoľný objekt.
- Web – Vytvára webovo-orientované integračné rozhranie, ktoré poskytuje napríklad file-upload funkcionality, inicializáciu IoC kontajneru. V tomto členení je zahrnutý aj Web MVC modul, ktorý bude aj predmetom testu.

Dôvodom pre vznik Spring frameworku bolo uľahčiť vývoj aplikácii na Java EE platforme so zameraním sa na nasledovné aspekty:

- Odstránenie úzkych väzieb POJO objektov a vrstiev k čomu slúži IoC návrhový vzor (kontajner). To znamená, že zodpovednosť za vytvorenie a previazanie objektu (bean) je presunutá z aplikácie na framework. Vytvorený objekt je následne získaný tzv. vsadením závislosti (dependency injection), čo je jednou z metód IoC.
- Podpora nástrojov obsluhujúcich prístup k dátam (JDBC, ORM),
- možnosť voľby business vrstvy pre aplikačnú architektúru (EJB, POJO),
- zjednodušené používanie rôznych Java EE API (JavaMail, JDBC, ...).

1.3.3 Apache Struts

Je najstarším z rodiny frameworkov zastrešovaných organizáciou Apache Software Foundation. Struts pozostáva z balíčkov rôznych úrovní, ktoré obsahujú množstvo tried a rozhraní. Spolu s utilitami a pomocnými triedami poskytuje triedy a rozhrania pre prácu s controller-om a prezentačnou vrstvou , pomocou vlastnej knižnice tagov.

Paradigmou Struts frameworku je taktiež MVC návrhový vzor:

- Controller komponenty – Každú požiadavku od užívateľa vybavuje ActionServlet. Ten príjme dotaz, zachytí URL na základe konfiguračného súboru a prenechá obsluhu žiadosti triede Action. Trieda Action je súčasťou controller-a a je zodpovedná za komunikáciu s vrstvou Model.
- View komponenty – Tak ako v samotnom MVC návrhovom modeli, aj tu sú View komponenty zodpovedné za zobrazenie informácií užívateľovi a taktiež prevzatie užívateľových požiadaviek, teda zodpovedajú za zobrazenie informácií

poskytnutých Model komponentami. Pre prezentačnú vrstvu sa najčastejšie používajú JSF, ale je možné použiť aj vlastné tagy, JavaScript, atď..

- Model komponenty – sú to Java triedy a zabezpečujú business logiku, teda poskytujú rozhranie databázam alebo back-end systémom. Tieto komponenty nemajú definovaný štandardný formát, preto je možné použiť ľubovoľné triedy, napríklad použité v iných projektoch.

1.3.4 Stripes

Je MVC framework, využívajúci technológiu anotácii a dodržiavajúci koncept "konvencie nad konfiguráciou" (convention-over-configuration), čo znamená: "schopnosť využívania programovacích konvencií na dosiahnutie rovnakej funkcionality, bez použitia externej konfigurácie. Používanie konfiguračných súborov (hojne využívaných napríklad vo frameworku Apache Struts) je eliminované na nevyhnutné minimum, teda na jeden jediný konfiguračný súbor "web.xml", vyžadovaný samotnou Java technológiou.

Aplikácia pozostáva z klasických Java tried, ktoré implementujú "ActionBean" rozhranie, ktoré príjme dáta na vstupe a spracováva užívateľské požiadavky. Taktiež je možné dediť z ktorejkoľvek inej triedy s plným využitím objektového návrhu. Funkcionalitu konfiguračných súborov nahradzuje samotný framework, ktorý vyhľadá všetky triedy implementujúce "ActionBean" a väzbu na URL vytvorí z názvu samotnej triedy alebo z jej anotácie s názvom "@UrlBinding".

II. PRAKTICKÁ ČASŤ

2 TESTOVANIE FRAMEWORKOV

V prvej kapitole bola popísaná JavaEE architektúra s dôvodmi a prínosmi pre oblasť programovania webových aplikácii. Taktiež boli predstavené vybrané framewokry, ktoré budú v tejto kapitole podrobené záťažovým testom a vzájomným porovnávaniam s dôrazom na požadované funkcionality. V závere kapitoly bude predstavená webová aplikácia zostavená vybranými nástrojmi, testovaných frameworkov.

2.1 Vývojové nástroje

Nakoľko v tejto časti práce sú výsledky porovnania aplikované na praktickej ukážke, neoddeliteľnou súčasťou vývoja ľubovoľného softvéru je aj vývojový nástroj podporujúci daný jazyk, teda takzvané IDE (*„Integrated Development Environment“*, softvér s integrovanými vývojovými nástrojmi). IDE môže obsahovať rôzne vývojové nástroje ako napríklad editor, kompilátor, interpreter, debuger a mnohé ďalšie nástroje zefektívňujúce softvérový vývoj v danom jazyku. IDE nástrojov podporujúcich Java EE platformu je nespočetné množstvo a medzi najznámejšie patria napríklad Eclipse, IntelliJ IDEA, Jbuilder, JDeveloper, NetBeans.

2.1.1 NetBeans

Zdrojové ukážky príkladov použité v teoretickej časti popisujúce jednotlivé frameworky, ale aj samotné testy frameworkov a aj záverečná webová aplikácia sú demonštrované v NetBeans IDE. Dôvody ovplyvňujúce túto voľbu sú nasledovné:

- NetBeans je open-source vyvíjaný spoločnosťou, ktorá stojí za samotným programovacím jazykom Java.
- Je robustný, teda jeho súčasťou sú rozsiahle vývojové nástroje, ktoré nepotrebujú dodatočnú implementáciu v podobe prídavných balíkov, ktorých inštalácia nie vždy prebehne podľa očakávaní. Medzi takéto nástroje patria napríklad:
    - aplikačný server GlassFish a Apache Tomcat,
    - frameworky: JavaServer Faces, Spring Web MVC, Struts, Hibernate
- Implementuje zaužívané nástroje pre tvorbu, správu a riadenie aplikácie: Apache Ant a Apache Maven.

- Obsahuje optimalizačné nástroje ako napríklad Refactor, Call Hierarchy, Built-in Profiler.

2.2 Efektívnosť vývoja aplikácie pomocou frameworkov

V nasledujúcich príkladoch bude demonštrovaný príklad aplikácie (formulár obsahujúci textové pole a tlačidlo). Po vyplnení textového poľa a odoslaní formulára sa zobrazí krátka správa, zobrazujúca obsah textového poľa. Účelom je demonštrovať náročnosť na vytvorenie jednoduchej aplikácie v jednotlivých frameworkoch a z hľadiska rozsiahlosti aplikácie a rozsiahlosti požadovaných znalostí rôznych technológií.

2.2.1 JSF aplikácia

Obsluha JSF stránky je v konfiguračnom súbore `/WEB-INF/web.xml`, kde sa deklaruje a mapuje servlet (FacesServlet). Po príchode požiadavky webový kontajner spúšťa tento servlet, ktorý už sám riadi beh aplikácie.

```
1  <servlet>
2      <servlet-name>Faces Servlet</servlet-name>
3      <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
4      <load-on-startup>1</load-on-startup>
5  </servlet>
6  <servlet-mapping>
7      <servlet-name>Faces Servlet</servlet-name>
8      <url-pattern>/faces/*</url-pattern>
9  </servlet-mapping>
10  ..
11 <welcome-file-list>
12     <welcome-file>faces/index.xhtml</welcome-file>
13 </welcome-file-list>
```

Konfiguračný súbor aplikácie mapuje úvodné zobrazenie aplikácie na súbor `index.xhtml`, obsahujúci formulár s textovým poľom a tlačidlom. Volanie JavaBeanu je zabezpečené EL tagom, volajúcim príslušnú metódu `#{uzivatelBean.uzivMeno}`.

```
1.  <h:form>
2.      <h4>JSF 2 - príklad</h4>
3.      Zadajte Vaše meno:
4.      <h:inputText value="#{uzivatelBean.uzivMeno}" />
```

```
5.     <h:commandButton value="Submit" action="odpoved" />
6.  </h:form>
```

Po odoslaní obsahu formulára (stlačení tlačidla "Submit") hodnota zadaná do textového poľa, volá metódu `setUzivMeno` (setter) triedy `UzivatelBean` (managed bean) a zároveň je volaný súbor `odpoved.xhtml`. Triede `UzivatelBean` predchádzajú anotácie (nahrádzajúce konfiguračný súbor `faces-config.xml`), ktorými je táto trieda registrovaná ako managed bean class.

```
1.  @ManagedBean
2.  @SessionScoped
3.  public class UzivatelBean implements Serializable {
4.     private String uzivMeno;
5.     public String getUzivMeno() {
6.        return uzivMeno;
7.     }
8.     public void setUzivMeno(String uzivMeno) {
9.        this.uzivMeno = uzivMeno;
10. }  }
```

Súbor `odpoved.xhtml` zobrazí správu spolu so zadanou hodnotou v textovom poli súboru `index.xhtml`, ktorú získa z prístupových metód triedy, volanej EL výrazom `#{uzivatelBean.uzivMeno}`.

```
1.  <h:body>
2.     <h4>JSF 2 - príklad</h4>
3.     Vaše meno je: <h:outputText value="#{uzivatelBean.uzivMeno}" />
4.  </h:body>
```

2.2.2 Spring aplikácia

Taktiež v prípade Springu je vstupnou bránou pre implementáciu frameworku do aplikácie konfiguračný súbor `web.xml`. Centrálnym servletom Springu je tzv. "front controller", (trieda `DispatcherServlet`) a v nasledovnej ukážke je pomenovaný ako `dispatcher`, ktorý obsluhuje všetky požiadavky prichádzajúce z URL `../main/*`. Počiatočným súborom je `redirect,jsp`, nastavujúci úvodnú stránku na `index.jsp`.

```
1.   <servlet>
2.      <servlet-name>dispatcher</servlet-name>
3.      <servlet-class>org.springframework.web.servlet.DispatcherServlet
</servlet-class>
4.      <load-on-startup>2</load-on-startup>
5.   </servlet>
6.   <servlet-mapping>
7.      <servlet-name>dispatcher</servlet-name>
8.      <url-pattern>/main/*</url-pattern>
9.   </servlet-mapping>
10.  <welcome-file-list>
11.     <welcome-file>redirect.jsp</welcome-file>
12.  </welcome-file-list>
```

Ďalší konfiguračný súbor využívaný front controllerom pre vytvorenie väzieb medzi vzájomne komunikujúcimi objektmi aplikácie je `dispatcher-servlet.xml`. Slovo "dispatcher" v jeho názve nie je náhodné a musí korešpondovať s pomenovaním front controllera vo `web.xml`. V konfigurácii je spresnené, že každá URL požiadavka z `http://.../uzivatel/` bude posielaná do triedy `UzivatelController` a výsledná URL pre zobrazenie, po úprave je `http://../uzivatel/odpoved.jsp`.

```
1.  <bean name="/uzivatel/*" class="controller.UzivatelController"
p:zobrazUzivMeno="uzivatel/odpoved"/>
```

Vo formulári je vstupné textové pole previazané s atribútom príslušnej triedy v modeli, teda textové pole `uzivMeno` je previazané s atribútom `uzivMeno` inštancie `uzivatel` triedy `Uzivatel`.

```
1.  <form:form modelAttribute="uzivatel">
2.     <table>
3.     <tr><td>Zadajte Vaše meno:</td><td><form:input path="uzivMeno" />
</td></tr>
4.     </table>
5.     <div><input type="submit" value="Submit"/></div>
6.  </form:form>
```

O samotné spracovanie HTTP požiadaviek sa postará trieda `UzivatelController`. Anotácie `@RequestMapping` slúžia pre mapovanie business metód. Metóda `formulár()` je

volaná pri požiadavke zobrazenia formulára (úvodnej stránky) a metóda `spacovanieFormDat()` je volaná po odoslaní formulára. Jej návratová hodnota sa nastaví v metóde `setZobrazUzivMeno` na logickú podobu výstupu zobrazenia, následne sa preloží v `dispatcher-servlet.xml` a zostaví vo `web.xml`, do fyzickej podoby výstupu `../uzivatel/odpoved.jsp`.

```
1.  @Controller
2.  public class UzivatelController {
3.     private String zobrazUzivMeno;
4.     public void setZobrazUzivMeno(String zobrazUzivMeno) {
5.        this.zobrazUzivMeno = zobrazUzivMeno;
6.     }
7.     @RequestMapping(method=RequestMethod.GET)
8.     public Uzivatel formular() {
9.        return new Uzivatel();
10.    }
11.    @RequestMapping(method=RequestMethod.POST)
12.    public String spacovanieFormDat(Uzivatel uzivatel) {
13.       return zobrazUzivMeno;
14. }  }
```

Trieda `Uzivatel` obsahuje prístupové metódy poskytujúce služby patriace do vrstvy - Model v MVC návrhovom vzore. Napĺňa atribút `uzivMeno` hodnotou zadanou vo formulári a taktiež vracia túto hodnotu, ak o to požiada kontrolér.

```
1.  public class Uzivatel {
2.     private String uzivMeno;
3.     public Uzivatel() {
4.     }
5.     public Uzivatel(String uzivMeno) {
6.        this.uzivMeno = uzivMeno;
7.     }
8.     public String getUzivMeno() {
9.        return uzivMeno;
10.    }
11.    public void setUzivMeno(String uzivMeno) {
12        this.uzivMeno = uzivMeno;
13     }
14.    @Override
15.    public String toString() {
```

```
16.       return uzivMeno;
17. }  }
```

Na záver Controller odošle na zobrazenie súbor `odpoved.jsp` s EL výrazom, obsahujúcim hodnotu atribútu `uzivMeno`, triedy `Uzivatel`.

```
1.  <body>
2.     <h4>SPRING - príklad</h4>
3.     <p>Vaše meno je: ${uzivatel}</p>
4.  </body>
```

2.2.3 Apache Struts aplikácia

Súbor web.xml v Apache Struts taktiež slúži konfigurácii vlastností servlet kontajnera.

```
1.   <filter>
2.      <filter-name>struts2</filter-name>
3.      <filter-class>org.apache.struts2.dispatcher.ng.filter.
StrutsPrepareAndExecuteFilter</filter-class>
4.   </filter>
5.   <filter-mapping>
6.      <filter-name>struts2</filter-name>
7.      <url-pattern>/*</url-pattern>
8.   </filter-mapping>
9.   <welcome-file-list>
10.     <welcome-file>index.jsp</welcome-file>
11.  </welcome-file-list>
```

Posledná verzia Apache Struts je odľahčená o konfiguračný súbor `struts-config.xml`. Ten bol nahradený jednoduchším a modulárnejším konfiguračným súborom `struts.xml`, ktorý inicializuje vlastné zdroje frameworku obsahujúce napríklad triedy volajúce business logiku. V nasledujúcom príklade je to trieda `Uzivatel` z balíka `utb` a ňou spracované dáta sú zobrazené súborom `odpoved.jsp`.

```
1.  <struts>
2.     <package name="default" extends="struts-default">
3.        <action name="Uzivatel" class="utb.Uzivatel">
4.           <result name="SUCCESS">/odpoved.jsp</result>
5.        </action>
```

```
6.      </package>
7.  </struts>
```

Úvodné zobrazenie stránky je vo `web.xml` aj v tomto prípade nastavené na súbor `index.jsp`, kde pomocou UI tagov je vytvorený formulár s textovým poľom a tlačidlom. Pre spracovanie obsahu textového poľa je volaná metóda `uzivMeno`, triedy `Uzivatel`.

```
1.  <%@taglib uri="/struts-tags" prefix="s" %>
2.  <body>
3.      <s:form action="Uzivatel" >
4.      <s:textfield name="uzivMeno" label="Zadajte Vaše meno" />
5.      <s:submit />
6.  </s:form>
7.  </body>
```

Riadiaci servlet nájde požadovanú triedu (na základe konfiguračného súboru `struts.xml`) a zavolá jej metódu `execute()`. Metóda spracuje prichádzajúce dáta od `index.jsp` a jej návratová hodnota je identifikátor, na základe ktorého riadiaci servlet rozhodne, ktorý súbor zobrazí svoj obsah.

```
1.  package utb;
2.  public class Uzivatel {
3.      private String message;
4.      private String uzivMeno;
5.
6.      public Uzivatel() {
7.      }
8.      public String execute() {
9.          setMessage("Vaše meno je: " + getUzivMeno());
10.         return "SUCCESS";
11.     }
12.     public String getMessage() {
13.         return message;
14.     }
15.     public void setMessage(String message) {
16.         this.message = message;
17.     }
18.     public String getUzivMeno() {
19.         return uzivMeno;
```

```
20.     }
21.     public void setUzivMeno(String uzivMeno) {
22.         this.uzivMeno = uzivMeno;
23. }   }
```

Návratové dáta sú konfiguračným súborom odoslané na zobrazenie súboru `odpoved.jsp`.

```
1.  <body>
2.      <h4>STRUTS - príklad</h4>
3.      <s:property value="message" />
4.  </body>
```

2.2.4 Stripes aplikácia

Ako už bolo spomenuté, Stripes pozná len jeden XML konfiguračný súbor `/WEB-INF/web.xml`, v ktorom sú deklarované štandardné nastavenia, ako napríklad Stripes Filter a Dispatcher Servlet (jedná sa o MVC model). Dôležitým parametrom je deklarácia `ActionResolver.Packages`, ktorý slúži na nasmerovanie k triede implementujúcej ActionBean. Hodnota parametra je cesta k balíku tried. Na konci konfiguračného súboru sa deklaruje úvodný súbor web aplikácie.

```
1.  <filter>
2.      <filter-name>StripesFilter</filter-name>
3.                                                              <filter-
class>net.sourceforge.stripes.controller.StripesFilter</filter-class>
4.      <init-param>
5.          <param-name>ActionResolver.Packages</param-name>
6.          <param-value>utb</param-value>
7.      </init-param>
8.  </filter>
9.  <welcome-file-list>
10.     <welcome-file>index.jsp</welcome-file>
11. </welcome-file-list>
```

Na začiatku úvodného súboru `index.jsp` je anotovaná knižnica tagov (Stripes Tag Library). Nasleduje formulár s textovým polom a tlačidlom. Tag tlačidlo obsahuje atribút `name`, tento atribút je dôležitý pre inicializáciu správnej metódy vo volanej triede. V tomto

prípade atribút `name` môže byť aj prázdny, lebo volaná metóda je zároveň aj preddefinovanou návratovou metódou s anotáciou `@DefaultHandler`.

```
1.  <%@ taglib prefix="stripes"
uri="http://stripes.sourceforge.net/stripes.tld"%>
2.  <stripes:form beanclass="utb.UzivatelBean" focus="">
3.
4.     <table>
5.        <tr><td>Zadajte Vaše meno:</td>
6.            <td><stripes:text name="uzivMeno"/></td>
7.            <td><stripes:submit name="vaseMeno" value="Submit"/></td>
8.        </tr>
9.     </table>
10. </stripes:form>
```

Po odoslaní formulára je volaná metóda `vaseMeno` triedy `utb.UzivatelBean`, implementujúcej ActionBean. Handler metóda môže mať ľubovoľný názov, no musí byť `public` a musí mať návratový typ `Resolution`.

```
1.  public class UzivatelActionBean implements ActionBean {
2.     private ActionBeanContext context;
3.     private String uzivMeno;
4.
5.     public ActionBeanContext getContext() {
6.        return context;
7.     }
8.     public void setContext(ActionBeanContext context) {
9.        this.context = context;
10.    }
11.    public String getUzivMeno() {
12.       return uzivMeno;
13.    }
14.    public void setUzivMeno(String uzivMeno) {
15.       this.uzivMeno = uzivMeno;
16.    }
17.
18.    @DefaultHandler
19.    public Resolution vaseMeno() {
20.        return new ForwardResolution("/odpoved.jsp");
21. }  }
```

Metóda vracia súbor `odpoved.jsp` obsahujúci EL výraz `${actionBean.uzivMeno}`.

```
1.  <body>
2.      <h4>STRIPES - príklad</h4>
3.      Vaše meno je: ${actionBean.uzivMeno}
4.  </body>
```

2.3 Bezpečnosť

Tak ako vo všeobecnosti problematika bezpečnosti informačných technológií sa dotýka rôznych oblastí v celom ich životnom cykle, ani v prípade webových aplikácií sa tento problém nijako zásadne nezužuje. Bezpečnosť informačných technológií, a teda aj webových aplikácií je veľmi zložité (a snáď aj nemožné) riešiť komplexne ako jeden problém, a preto je vhodné ho rozdeliť na oblasti riešenia bezpečnosti počas tvorby, poskytovania a užívania produktu. Keďže posledné dve spomenuté oblasti sa týkajú správy existujúceho systému a jeho užívateľov, táto podkapitola sa zameriava na počiatočnú oblasť, a to oblasť riešenia bezpečnosti počas vývoja webových aplikácií.

2.3.1 JSF

JSF nemá implementovaný bezpečnostný model, ale skôr sa spolieha na bezpečnostný štandard JavaEE. Napríklad medzi bezpečnostné modely riadenia prístupu patria JAAS (Java Authentication and Authorization Service) a ACL (Access Control List), ktoré je možné implementovať bez väčšej námahy.

Bezpečnosti na aplikačnej vrstve, explicitne venujú pozornosť projekty budované na JSF, akým je napríklad Apache Myfaces a ich SecurityContext.

2.3.2 Spring

Časť venujúca sa bezpečnosti aplikáciám na JavaEE platforme je natoľko rozsiahla a komplexná, že sa stala samostatným frameworkom pod súhrnným názvom "Spring Security" a je možné ho považovať za bezpečnostný štandard Spring aplikácii. Rámec poskytuje komplexnú sadu služieb, pre autentifikáciu a autorizáciu:

- HTTP autorizácia a autentifikácia
- Bezpečnosť na servisnej vrstve (napríklad EJB anotácie)

- Rozšírený ACL balík
- Podpora OpenID
- LDAP autentifikácia

Tento rámec je možné využívať v aplikáciách aj samostatne, alebo je možné ho implementovať do iných frameworkoch mimo Spring.

2.3.3 Apache Struts

Riešenie bezpečnosti v Spring frameworku je možné rozdeliť do troch stupňov podľa samotného MVC návrhového vzoru.

Primárna metóda zabezpečujúca validáciu aplikácie v časti - Model, sa nazýva "Struts Validation framework", vyžadujúca existenciu niekoľkých konfiguračných súborov (`validator.xml`, `commons-validator.xml`, `validator-rules.xml`), ďalej ich deklaráciu vo všetkých formulároch, využívajúcich toto rozšírenie a taktiež deklaráciu v samotnom `struts-config.xml` konfiguračnom súbore.

V časti - View sa využíva proces „výstupnej hygieny“ (Output Sanitation), ktorý zabezpečuje, že výstup neobsahuje HTML a XML špecifické znaky, ktoré by mohli byť zneužité napríklad k XSS útokom, ale sú definované vlastné tagy (`bean:Write`, `html:Hidden`, `html:Messages`, `html:Multibox`, `html:OptionsCollection`, `html:Options`, `html:Option`, `html:Radio`, `html:TextArea`, `html:File`, `html:Hidden`, `html:Password`, `html:Text`).

V časti - Controller je umožnené v konfiguračnom súbore `struts-config.xml` špecifikovať atribút `role` a uviesť zoznam užívateľských rolí s povoleným prístupom k objektom `ActionMapping`.

2.3.4 Stripes

Aj keď Stripes framework neobsahuje konzistentnú knižnicu nástrojov zaoberajúcich sa bezpečnosťou, využíva nástroje štandardu JEE, ktoré obohacuje o vlastné metódy. Ako napríklad v prípade prevencie pred XSS útokom, kedy JSP metóda ošetrujúca užívateľom zadané hodnoty `fn:escapeXml()` je rozšírená o Stripes metódu `HtmlUtil.encode()`,

filtrujúcu hodnoty pred ich zobrazením. Taktiež existujú knižnice, zaoberajúce sa touto problematikou. Sú nimi napríklad Spripes-security a StripesStuff.

- Stripes-security: je knižnica založená na ACL štandarde. Využíva užívateľské role spolu s preddefinovanými anotáciami tried a metód.
- StripesStuff: SecurityManager plne podporuje JEE anotácie (`@DenyAll`, `@PermitAll`, `@RolesAllowed`) a podporuje kontrolu JSP tagov (správnosť vykonania akcie).

2.4 Viacúrovňová autorizácia

Autorizácia a autentifikácia (overovanie) sú dva pojmy ktorých význam sa často zamieňa, prípadne sa im prisudzuje spoločný význam, čo nie je správne. *Overovanie je proces určujúci, či je entita (užívateľ) tým, za koho sa vydáva. Autorizácia je proces určujúci akcie, ktoré je užívateľ v systéme počítača alebo v sieti oprávnený vykonávať*[6].

Platforma JavaEE poskytuje modulárne riešenie autentifikácie a autorizácie na úrovni aplikačného servera. Modulárne vo význame, že bez zmeny kódu samotnej aplikácie, teda len zmenou v nastaveniach konfiguračných súborov, je možné meniť napríklad úložisko overovaných dát (LDAP, databáza, ...) alebo voľba rozhrania autentifikácie (HTTP autentifikácia, alebo HTML formulár). Tieto služby zabezpečuje modul JAAS, ktorý je súčasťou Java SDK štandardu od verzie 1.4.

Nasledovná ukážka kódu demonštruje spôsob autorizácie užívateľa v aplikácii. Všetky potrebné nastavenia sa vykonávajú v konfiguračnom súbore `web.xml`. V tomto súbore definujeme cestu, teda adresár `admin/*`, do ktorého bude mať prístup len užívateľ s rolou `spravca`. Následne definujeme žiadosť o autentifikáciu, pokiaľ je volaný zabezpečený obsah. Tag `auth-metod` definuje formu žiadosti o autentifikáciu (BASIC - HTTP žiadosť o meno a heslo) a `realm-name` je doména (databáza, súbor,...) užívateľských mien, hesiel a rolí.

```
1.  ..
```

[6] TAYLOR, Art; BUEGE, Brian; LAYMAN, Randy. *Hacking bez tajemství: Java a J2EE.* str. 19

```
2.  <security-constraint>
3.     <web-resource-collection>
4.     <web-resource-name>Admin webu nieco.org</web-resource-name>
5.        <url-pattern>/admin/*</url-pattern>
6.     </web-resource-collection>
7.     <auth-constraint>
8.       <role-name>spravca</role-name>
9.     </auth-constraint>
10. </security-constraint>
11. ..
12. <login-config>
13.    <auth-method>BASIC</auth-method>
14.    <realm-name>eshop</realm-name>
15. </login-config>
16. ..
```

Na nižšej vrstve sa pomocou takzvanej deklaratívnej bezpečnosti zabezpečí prístup k metóde pomocou anotácie. V nasledovnom príklade je pomocou anotácie povolený prístup k triede len užívateľovi s pridelenou rolou `spravca`.

```
1.  ..
2.  @RolesAllowed("spravca")
3.  public void setUzivMeno(String uzivMeno) {
4.     this.uzivMeno = uzivMeno;
5.  }
6.  ..
```

Pre viacúrovňovú autorizáciu (hierarchické členenie užívateľských rolí) má Spring k dispozícii riešenie, ktoré je súčasťou projektu Spring Security. Ako už bolo spomenuté, každý Spring projekt, predstavuje samostatný balík, ktorý je možné implementovať do ľubovoľnej aplikácie, a tak je to aj v tomto prípade. Trieda „RoleHierarchyVoter“ umožňuje nastaviť dedenie rolí. Implementuje sa spolu s triedou „RoleHierarchy“, od ktorej získava „oprávnenia“ priradené užívateľom.

```
1. <bean id="roleVoter"
class="org.springframework.security.access.vote.RoleHierarchyVoter">
2.    <constructor-arg ref="roleHierarchy" />
3.     <bean id="roleHierarchy"
class="org.springframework.security.access.hierarchicalroles.RoleHierarch
yImpl">
```

```
4.      <property name="hierarchy">
5.          ROLE_ADMIN > ROLE_USER
6.          ROLE_USER > ROLE_GUEST
7.      </property>
8.      </bean>
9.      ..
```

Hierarchia rolí v uvedenom príklade znamená že `ROLE_ADMIN` zahrňuje `ROLE_USER` a `ROLE_USER` zahrňuje `ROLE_GUEST`.

2.5 Spätná kompatibilita API

Rozhranie pre programovanie aplikácie (API) je možné považovať za základné stavebné prvky každej aplikácie. Sú to objekty (funkcie, procedúry, triedy) združené do knižnice, prípadne inej programovej formy, ktoré sú k dispozícii programátorovi. Programátor pomocou týchto programových celkov, ich implementáciou a vzájomným volaním vytvára samotnú aplikáciu. Ich účelom je minimalizácia písania kódu samotným programátorom, čo výsledkom je viacero pozitívnych aspektov.

- Prítomnosť často a opakovane používaných funkcionalít priamo v programovom prostredí,
- eliminácia chybného kódu na minimum, keďže API je možné považovať za takmer bezchybný kód s maximálnym efektom,
- minimalizácia napísaného kódu, čo vedie k úspore času a zdrojov (programátorov) a zároveň minimalizuje chybovosť (Pravidlo: Dĺžka zdrojového kódu je priamo úmerná jeho chybovosti).

Každý uvedený framework má prístupnú API dokumentáciu, ktorá obsahuje nielen zoznam a popis všetkých existujúcich balíčkov a tried, ale aj ich väzby na predchádzajúce verzie frameworkov. Taktiež sú tu uvedené funkcie z predchádzajúcich verzií API, ktoré sú označené ako "Deprecated" s definovaním ich náhrady v aktuálnej verzii API. Táto dokumentácia je výsledkom zdrojovej dokumentácie, vytváranej v priebehu programovania samotných API (viď. kapitola Dokumentácia).

Webové odkazy na API dokumentáciu uvádzaných frameworkov:

- JSF - http://java.sun.com/javaee/javaserverfaces/reference/api/index.html

- Spring - http://static.springsource.org/spring/docs/3.0.x/api/
- Apache Struts - http://struts.apache.org/2.0.14/struts2-core/apidocs/index.html
- Stripes - http://stripes.sourceforge.net/docs/current/javadoc/

Spätná kompatibilita starších verzií API je významná z dlhodobého hľadiska používania nástrojov, a preto jej podpora je nezanedbateľným prínosom pre daný framework.

- JSF – zásadné zmeny vo verzii 2.x neumožňujú spätnú kompatibilitu s verziou 1.x
- Spring – sa snaží zachovať spätnú kompatibilitu na základe doporučení pre verzovanie projektu „Apache Portable Runtime“
- Apache Struts – fundamentálnou zásadou tohto frameworku je veľký dôraz spätnej kompatibility, čo zároveň spôsobuje pomalú implementáciu zmien v novších verziách.

2.6 Integritné obmedzenia

Integrita databázy definuje pravidlá pre zabezpečenie konzistentnosti dát. To znamená, aby do databázy vstupovali len dáta, ktoré majú byť uchované a aby nedošlo k nekontrolovanej strate, prípade pozmeneniu uchovávaných dát v tejto databáze.

Integritné obmedzenia:

- Entitné obmedzenie je v relačnom modeli povinné. Definuje povinnosť definovania primárneho kľúča tabuľky. Ide o atribút jednoznačne špecifikujúci n-ticu relácie.
- Doménové obmedzenie určuje povinnosť definovať obmedzenia stĺpcov tabuľky (dátový typ, rozsah hodnôt).
- Referenčné obmedzenie sa vzťahuje na väzby medzi tabuľkami, a to s použitím cudzích kľúčov.
- Obmedzenia definované užívateľom sa týkajú rozsahu platnosti dát špecifických pre danú povahu definovaného údaju (0 < vek < 100; 1 <= známka <= 5; ...).

Bežnou praxou v iných programovacích jazykoch je vytváranie databázových spojení priamo v aplikácii, čo je samozrejme možné aj v JavaEE, ale JavaEE ponúka iné riešenie; a to ponechať túto režijnú činnosť (nadviazanie spojenia, identifikácia) na aplikačnom serveri.

Zámerom je poskytovať unifikovaný prístup k rôznym databázam, čo umožňuje programátorovi nezaoberať sa špecifickým API konkrétnej databázy, ale využívať jednotné rozhranie pre prístup do ľubovoľnej databázy (podmienkou je, aby daný typ databázy bol podporovaný). Tento koncept zdôrazňuje dôležitosť implementácie integritných obmedzení priamo v návrhu konkrétnej databázy. Ďalšie výhody implementácie integritných obmedzení už počas návrhu databázy sú nasledovné:

- Interoperabilita - Ak s danou databázou komunikujú ďalšie systémy, čo je v prípade rozsiahlych riešení na platforme JavaEE dosť časté, je vhodné zabezpečiť integritu dát na jednom mieste (teda v databáze), vzhľadom k faktu, ak nie všetky implementované systémy môžu vlastniť funkcionality zabezpečenia integrity dát. Ďalej sa tým predchádza aj potenciálnemu vnášaniu chybovosti implementácii týchto funkcionalít v kooperovaných systémoch.
- Centralizácia - Konzistentnosť dát je centralizovaná na jednom mieste, v databáze. Ak dochádza k nekonzistentnosti dát, pozornosť je sústredená na problém v databáze (nesprávne integritné obmedzenia, zle navrhnutý model).
- Výkon a spoľahlivosť - Funkcionality, napríklad integritné obmedzenia obsiahnuté v databázových systémoch, sú navrhnuté s dôrazom na správnosť a efektivitu ako napríklad v prípade rôznych API.

Súčasťou JavaEE platformy sú nasledovné API definujúce prácu s databázou:

- JDBC API - poskytuje rozhranie pre prístup k relačným databázam,
- JPA - poskytuje nástroje objektovo-relačného mapovania ORM.

Objektovo-relačné mapovanie je technika zaisťujúca konverziu dát medzi objektovým a dátovým modelom. To znamená, že mapuje Java objekty na entity v relačnej databáze. K tomu sa používajú tzv. mapovacie súbory, obsahujúce informácie, ako transformovať objekt do databázy a naopak. To umožňuje v aplikácii narábať s objektovým modelom dát, pristupujúcich do relačnej databázy. Namiesto mapovacích súborov je možné použiť anotácie. Vďaka týmto technikám je možné vykonávať určité integritné obmedzenia priamo v Modeli, prípadne ich implementovať do databázy (ako v prípade Hibernate frameworku).

2.6.1 Hibernate

Najznámejším Java frameworkom ORM implementujúcim JPA rozhranie, je Hibernate. Hibernate je samostatný framework, ktorý je možné implementovať do väčšiny JavaEE frameworkov. Z frameworkov uvedených v tejto práci, má Hibernate priamu podporu u Spring a Stripes. Taktiež je súčasťou základnej inštalácie NetBeans IDE.

Používanie JPA v aplikácii si vyžaduje nastaviť typ používanej databázy, prihlasovacie parametre a mapovanie perzistentných objektov (triedy, ktoré sú mapované na tabuľky). Tieto nastavenia sa vykonávajú v konfiguračnom súbore `persistence.xml`.

```
1.  ..
2.  <properties>
3.    <property name="hibernate.connection.driver_class"
value="com.mysql.jdbc.Driver" />
4.    <property name="hibernate.connection.url"
value="jdbc:mysql://localhost:80/" />
5.    <property name="hibernate.connection.username" value="meno" />
6.    <property name="hibernate.connection.password" value="heslo" />
7.  </properties>
8.  ..
```

Ďalej je potrebné vytvoriť perzistentný objekt, prezentujúci tabuľku v databáze. Pre mapovanie ich atribútov a metód sa používajú anotácie (prípadne XML mapovacie súbory).

```
1.    @Entity
2.    public class Uzivatel {
3.
4.        @Id
5.        @GeneratedValue
6.        private Integer id;
7.
8.       @Column(length=30, nullable=false)
9.       private String meno;
10.
11.    @OneToOne
12.    private Kontakty kontakty;
13.    ...
14.  }
15.
16.    // dalsia trieda
```

```
17.   @Entity
18.   public class Kontakty {
19.
20.         @Id
21.         @GeneratedValue
22.         private Integer id;
23.
24.         @Column(length=100, nullable=false)
25.         private String adresa;
26.     ...
27.  }
```

Ak máme k dispozícii identifikátor objektu `Uzivatel` (teda `id`), celý objekt užívateľa môžeme získať z databázy nasledovným spôsobom:

```
1.   ...
2.   Uzivatel uzivatel = (Uzivatel) session get(Uzivatel class, id);
3.   ...
```

Anotácie triedy z predchádzajúceho príkladu poukazujú aj na ďalšiu vlastnosť JPA - možnosť implementácie integritných obmedzení.

- Entitné obmedzenie: na riadku 4., anotácia `@id` označuje atribút `private Integer id` ako primárny kľúč a na riadku 5., anotácia `@GeneratedValue` špecifikuje, že tomuto atribútu bude automaticky pridelená hodnota.
- Doménové obmedzenie: na riadku 8. je anotácia `@Column(length=30, nullable=false)` definujúca počet znakov a nenulovosť atribútu `private String meno;`
- Referenčné obmedzenie: na riadku 11. je anotácia `@OneToOne` mapujúca asociáciu od objektu `Kontakty` k objektu `Uzivatel`.

Bežnou praxou pri vývoji aplikácii je vytváranie objektového modelu z vopred vytvorených databázových tabuliek. Hibernate umožňuje tento postup zameniť, teda vytvárať databázové tabuľky z existujúceho objektového modelu a ďalej ich dynamicky modifikovať podľa tohto modelu. Túto funkcionalitu je možné dosiahnuť nastavením príslušnej vlastnosti v konfiguračnom súbore `persistence.xml`. Integritné obmedzenia tabuliek sú definované anotáciami a niektoré z nich sú uvedené aj v predchádzajúcom príklade.

```
1.  ...
2.  <property name="hibernate.hbm2ddl.auto" value="update" />;
3.  ...
```

2.7 Nezávislosť logického a fyzického modelu databázy

Zriedkavo dochádza k zásadným zásahom do funkčného systému napríklad v rozsahu zmien v štruktúre databázy (zmena počtu alebo názvu atribútov tabuľky), a ak sa tak deje, tak sú najčastejšie dva dôvody. Nesprávna funkčnosť systému, vyplývajúca z nesprávnej analýzy, návrhu, vývoja a implementácie systému. Druhým dôvodom môžu byť nové požiadavky (rozšírenie, zmena, nové uplatnenie) na existujúci systém, s ktorými sa neuvažovalo pri jeho návrhu.

Konceptom JDBC API je Driver Manager, ktorý je prostredníkom medzi aplikačnou vrstvou a dátovou, v tomto prípade databázou. Driver Manager umožňuje vkladať inštancie tried do SQL dotazov, platných pre konkrétny typ databázy. Použitie JDBC API v aplikácii nijako zásadne nezjednodušuje nutnosť zásahu do kódu aplikácie, ale napríklad Spring framework obsahuje triedu JdbcTemplate implementujúcu rozhranie ParameterizedRowMapper, ktorý sa stará o naplnenie inštancie požadovanej triedy hodnotami z SQL výsledkovej sady. Výhodou je, že pokiaľ vykonáme zmeny v tabuľke (pridanie atribútu, zmena, názvu atribútu), je potrebné vykonať zmenu v dátovej vrstve len na jednom mieste, teda v príslušnom RowMapperi. Každej tabuľke prislúcha jeden RowMapper.

```
1.  ..
2.  public class ObchodnikRowMapper implements
ParameterizedRowMapper<Podnik> {
3.      public Obchodnik mapRow(ResultSet rs, int i) throws SQLException {
4.          Obchodnik ob = new Obchodnik();
5.          ob.setId(rs.getInt("id"));
6.          ob.setMeno(rs.getString("meno"));
7.          return ob;
8.  }  }
9.  ..
```

Jednou zo základných vlastností ORM je nezávislosť logického a fyzického modelu databázy, čo je zreteľné aj z predchádzajúcej kapitoly. Ďalším príkladom tejto vlastnosti, možnosť použitia jedného perzistentného modelu na komunikáciu s viacerými databázami

pomocou vytvorenia samostatných konfiguračných, pre každú databázu. Následne v triede volajúcej tieto databázy je vytvorený samostatný konštruktor pre každú databázu zvlášť. Ak sa volajú dve databázy definované v konfiguračných súboroch napríklad `oracle.cfg.xml` a `mysql.cfg.xml`, potom deklarácia týchto konštruktorov v triede volajúcej obe databázy je nasledovná.

```
1.   ...
2.   SessionFactory sessionFactoryOracle = new
3.   Configuration().configure("oracle.cfg.xml").buildSessionFactory();
4.   SessionFactory sessionFactoryMysql = new
5.   Configuration().configure("mysql.cfg.xml").buildSessionFactory();
6.   ...
```

2.8 Časová a pamäťová zložitosť vybraných operácii

Pre objektívne vyhodnotenie hardvérovej náročnosti aplikácie, zostavenej s použitím konkrétneho frameworku, boli testy vykonané v dvoch odlišných rovinách simulujúcich reálne nasadenie aplikácie s použitím testovacích nástrojov JMeter a MetBeans Profile.

2.8.1 Podmienky testovania

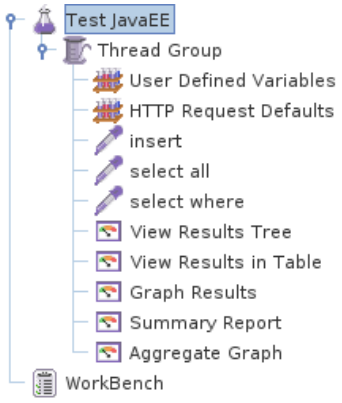
Testy časovej a pamäťovej zložitosti boli uskutočnené na notebooku ASUS UL80V s parametrami:

- Procesor: 1.30GHz (dual core)
- Operačná pamäť: 4 GB
- Operačný systém: Linux Debian (kernel 2.6.32-5-686)
- Aplikačný server - GlassFish Server 3.0.1
- Databáza – MySQL 5.1.49-3

2.8.2 JMeter

Je open-source softvér, ktorý sa používa ako nástroj pre záťažové testovanie, analýzu a meranie výkonnosti rôznych služieb, so zameraním na webové aplikácie. Testovanie bolo uskutočnené zo vzdialenej stanice, umiestnenej v lokálnej sieti, to znamená, že Jmeter bol umiestnený na stanici, z ktorej moli vykonávané dotazy pre tú konkrétnu aplikáciu, spustenú

na aplikačnom serveri. Dôvodom sú hardvérové nároky aplikačného servera, ale aj samotného softvéru Jmeter.



Obr 4. Testovací scenár JMeter

Samotný test prebiehal nasledovne. JMeter simuloval prístup 10 užívateľov. Užívatelia pristupovali postupne v sekundových intervaloch. Každý užívateľ mal nasledovné požiadavky o ktoré požiadal aplikačný server trikrát za sebou.

- vloženie 20 záznamov do tabuľky. Každý záznam obsahuje 7 atribútov (pomenované v nasledovných tabuľkách ako „insert“),
- výpis všetkých záznamov z tabuľky (pomenované v nasledovných tabuľkách ako „select all“),
- výpis všetkých záznamov z tabuľky s požadovaným atribútom (pomenované v nasledovných tabuľkách ako „select where“).

Celý test sa zopakoval trikrát pre každý framework a výsledky testov sú zobrazené v nasledovných tabuľkách.

Tab. 1. JSF: výsledky testov - JMeter

	Priemer [ms]	Min [ms]	Max [ms]	Priepustnosť [kB/s]
Prvý test				
insert	11	6	61	0,92
select all	19	3	47	133,64
select where	3	2	10	3,03
Celkom	11	2	61	136,96
Druhý test				
insert	12	7	61	0,92
select all	17	4	32	133,48
select where	3	2	5	3,03
Celkom	11	2	61	137,12
Tretí test				

insert	9	6	14	0,92
select all	18	4	33	133,30
select where	3	2	13	3,02
Celkom	10	2	33	136,92

Tab. 2. SPRING: výsledky testov - JMeter

	Priemer [ms]	Min [ms]	Max [ms]	Priepustnosť [kB/s]
Prvý test				
insert	10	6	58	0,92
select all	18	4	40	133,14
select where	3	2	7	3,02
Celkom	11	2	58	136,84
Druhý test				
insert	10	6	56	0,92
select all	17	4	54	132,72
select where	3	2	8	3,01
Celkom	10	2	56	136,38
Tretí test				
insert	10	6	59	0,92
select all	18	3	51	133,34
select where	3	2	4	3,03
Celkom	11	2	59	137,00

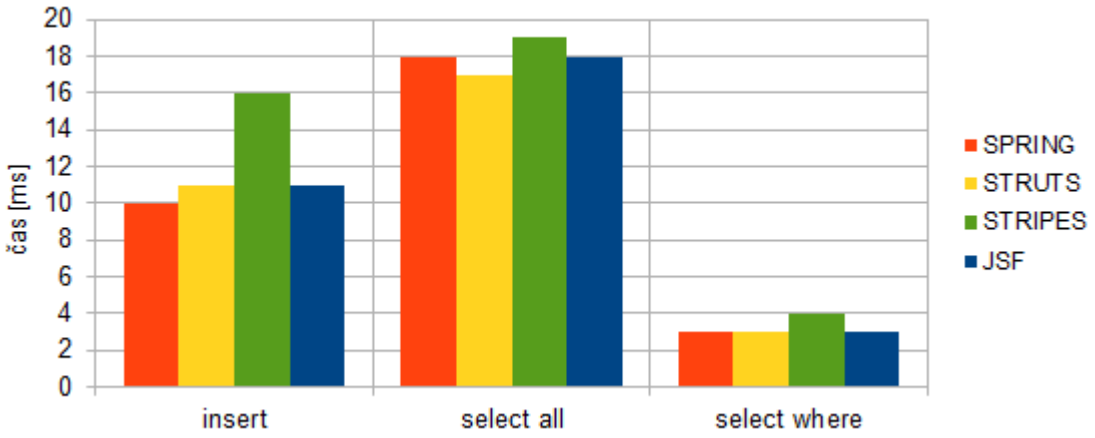
Tab. 3. APACHE STRUTS: výsledky testov - JMeter

	Priemer [ms]	Min [ms]	Max [ms]	Priepustnosť [kB/s]
Prvý test				
insert	13	6	66	0,92
select all	18	4	53	133,02
select where	3	2	7	3,02
Celkom	11	2	66	136,69
Druhý test				
insert	12	6	63	0,91
select all	17	4	57	132,20
select where	3	2	7	3,00
Celkom	11	2	63	135,77
Tretí test				
insert	9	6	14	0,92
select all	16	4	59	132,91
select where	3	2	8	3,02
Celkom	9	2	59	136,50

Tab. 4. STRIPES: výsledky testov - JMeter

	Priemer [ms]	Min [ms]	Max [ms]	Priepustnosť [kB/s]
Prvý test				
insert	15	9	61	0,91
select all	19	5	39	132,85
select where	4	3	9	3,02
Celkom	13	3	61	136,19
Druhý test				
insert	16	9	149	0,90
select all	19	4	45	130,96
select where	3	3	7	2,97
Celkom	13	3	149	134,33
Tretí test				
insert	16	7	60	0,91
select all	19	5	41	132,37
select where	4	3	9	3,01
Celkom	13	3	60	135,86

Na základe získaných údajov z vykonaných meraní je zostavený výsledný graf, zobrazujúci výsledky priemerných časových odoziev na jednotlivé požiadavky vo všetkých testovaných frameworkoch.



Obr. 5. Graf časovej odozvy vybraných funkcií v testovaných frameworkoch

Z uvedeného grafu je zrejmé, že medzi vybranými frameworkami nie sú zásadné rozdiely časovej náročnosti na vybrané funkcie, a teda by nemali mať zásadný vplyv na výber konkrétneho frameworku.

2.8.3 NetBeans Profiler

Je modulom NetBeans IDE poskytujúcim komplexné profilovanie vyvíjanej aplikácie. Funkcie obsiahnuté v tomto module obsahujú profilovanie CPU, RAM, vlákien a taktiež základný monitoring JVM. Nástroj je veľkým pomocníkom pri optimalizácii pamäťovej náročnosti a celkového výkonu vyvíjanej aplikácie.

Nasledujúci test vyhodnocuje vyťaženosť pamäte, teda veľkosť alokovanej pamäte a veľkosť použitej pamäte využívanej webovou aplikáciou.

Tab. 5. JSF: výsledky testov - Profiler

JSF	Pamäť max [MB]	Pamäť použitá [MB]
Test 1	85	53
Test 2	85	47
Test 3	85	48
Priemer	85	49

Tab. 6. SPRING: výsledky testov - Profiler

SPRING	Pamäť max [MB]	Pamäť použitá [MB]
Test 1	88	61
Test 2	86	42
Test 3	86	44
Priemer	87	49

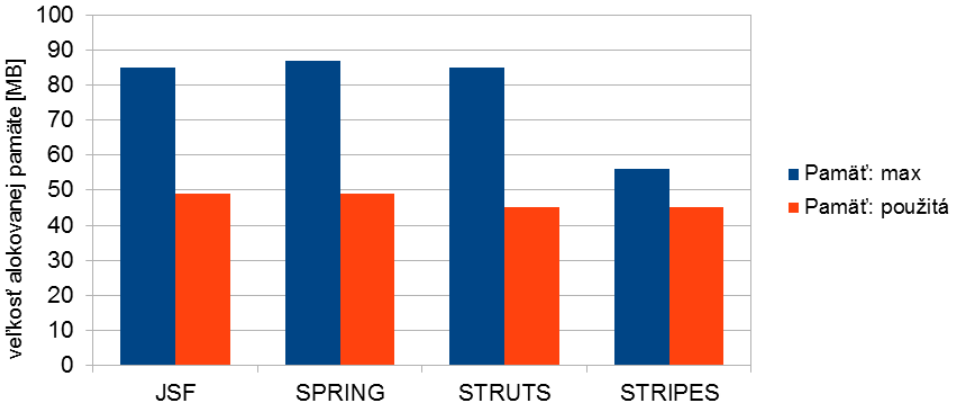
Tab. 7. STRUTS: výsledky testov - Profiler

STRUTS	Pamäť max [MB]	Pamäť použitá [MB]
Test 1	83	49
Test 2	85	45
Test 3	86	42
Priemer	85	45

Tab. 8. STRIPES: výsledky testov - Profiler

STRIPES	Pamäť max [MB]	Pamäť použitá [MB]
Test 1	56	38
Test 2	56	34
Test 3	56	34
Priemer	56	35

Taktiež aj v prípade druhých testov je z nameraných hodnôt zostavený graf. Hodnoty v grafe sú výsledkami priemerných hodnôt z jednotlivých meraní vyťaženia pamäte testovaných frameworkov.



Obr. 6. Graf pamäťovej náročnosti testovaných frameworkoch

2.8.4 Vyhodnotenie testov

Na základe výsledkov získaných z predchádzajúcich testov je možné vyvodiť záver, že ani v prípade časovej náročnosti a taktiež vyťaženosti pamäte nepredurčujú žiadny z vybraných frameworkov k jednoznačnému odporučeniu.

Malou výhodou môže byť menšia alokácia pamäte v prípade Stripes frameworku v porovnaní s ostatnými, ale rozdiel 10 MB sa dá považovať za zanedbateľný, vzhľadom k veľkosti operačnej pamäte v súčasných webových serveroch.

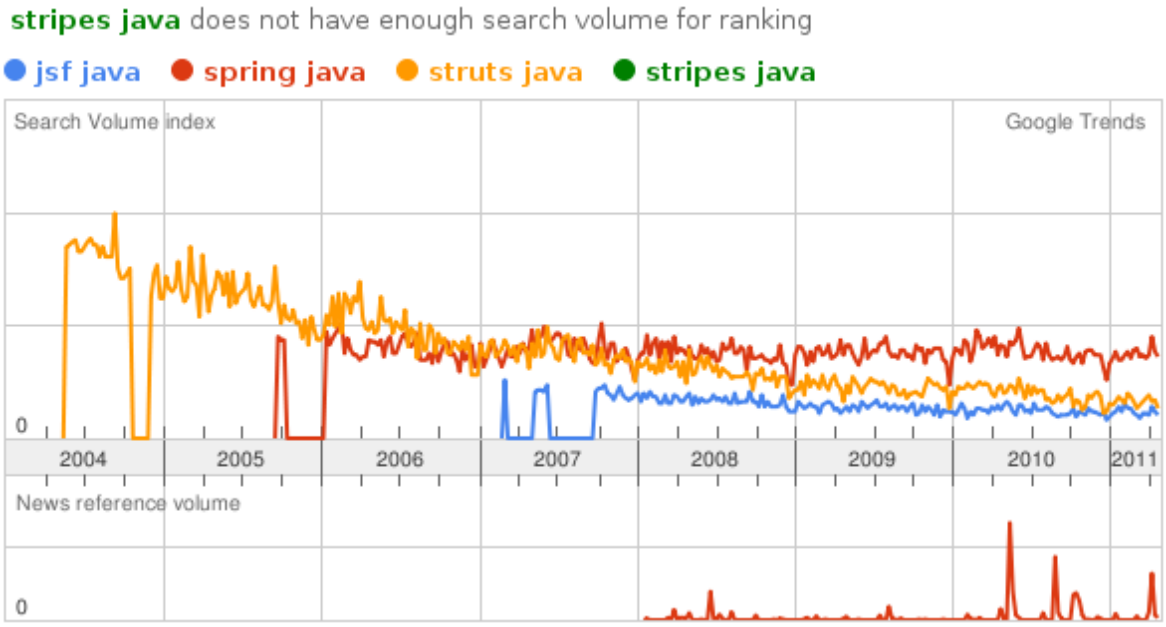
2.9 Dokumentácia

Jednou z množstva pozitívnych vlastností programovacieho jazyka Java je tvorba dokumentácie priamo počas tvorby, teda písania programu. Táto dokumentácia je vlastne komentárom v zdrojovom kóde (komentár začínajúci sadou znakov: `/**`) a jeho súčasťou môžu byť špeciálne značky (`@author`, `@version`, `@param`, `...`), ktoré slúžia pre formátovanie a navigáciu v dokumentácii.

Nakoľko triedy v Java EE, známe ako Beans, sú plnohodnotnými triedami jazyka Java, je možné týmto spôsobom tvoriť dokumentáciu príslušných tried aj v JavaEE. Týmto spôsobom je vytvorená dokumentácia API príslušných frameworkov, ktorá je dostupná na internete viď kapitola 2.5 Spätná kompatibilita API.

Ďalším neodmysliteľným zdrojom informácii je samotný internet, ktorého kvalita, ale aj kvantita získaných informácii je priamo úmerná schopnostiam ich vedieť vyhľadávať. A preto nie je možné objektívne zhodnotiť, ako sú na tom jednotlivé frameworky z hľadiska voľne dostupných zdrojov na internete. No jednoduchá štatistika vyhľadávania kľúčových slov na internete môže napovedať o množstve dostupných informácii týkajúcich sa vyhľadávaných kľúčových slov.

Pre túto štatistiku bol použitý nástroj Google Trends.



Obr. 7. Výsledok vyhľadávaných kľúčových slov na Google Trends.

Z grafu je možné usúdiť, že najvyhľadávanejším frameworkom je Spring. Stripes sa do tejto štatistiky ani nedostal pre jeho nízku mieru vyhľadávania.

Hlavným zdrojom relevantných informácii, a teda aj dokumentácie stále zostávajú knihy, či už v tlačenej alebo elektronickej podobe. Na českom a slovenskom knižnom trhu sa momentálne nevyskytuje žiadna literatúra týkajúca sa problematiky JavaEE, a preto prehľad dostupnej literatúry k jednotlivým framevorkom je čerpaný z knižného servera www.amazon.com.

Prehľad dostupnej svetovej literatúry k jednotlivým frameworkom:

- JSF - 271 titulov, z toho 103 titulov venujúcich sa JSF 2.0
- Spring - 164 titulov

- Apache Struts - 119 titulov
- Stripes - 8 titulov

2.10 Záverečné vyhodnotenie

Zo zistení získaných počas porovnávania jednotlivých frameworkov a taktiež z výsledkov testov je možné vyvodiť záver, že žiadny z testovaných frameworkov nie je možné charakterizovať ako ideálny a všestranný nástroj.

Pre nižšie požiadavky na znalosť problematiky JavaEE platformy (nevyžaduje sa znalosť JSP a servlets) a zároveň širokej podpore vizuálnych nástrojov prezentačnej vrstvy je pre menšie projekty vhodné použitie JSF frameworku. Veľkým množstvom nezávislých nástrojov na rôzne oblasti problémov (bezpečnosť, mobilné aplikácie...) disponuje Spring, vďaka čomu je možné ho implementovať aj len za účelom riešenia konkrétnych požiadaviek v celom projekte.

Vhodným a v praxi často preferovaným riešením voľby vhodného vývojového nástroja, býva ich kombinácia. Toto riešenie bolo aj voľbou pre zostavenie prezentačnej webovej aplikácie, v ktorej boli použité vybrané nástroje z JSF, Spring a Hibernate frameworkov.

3 PREZENTÁCIA VÝSLEDNEJ APLIKÁCIE

Posledná časť tejto práce prezentuje jednoduchú web stránku pozostávajúcu z vybraných vlastností testovaných frameworkov, predstavujúcu internetový Bazár s nasledovnými funkciami.

Užívateľské rozhranie:

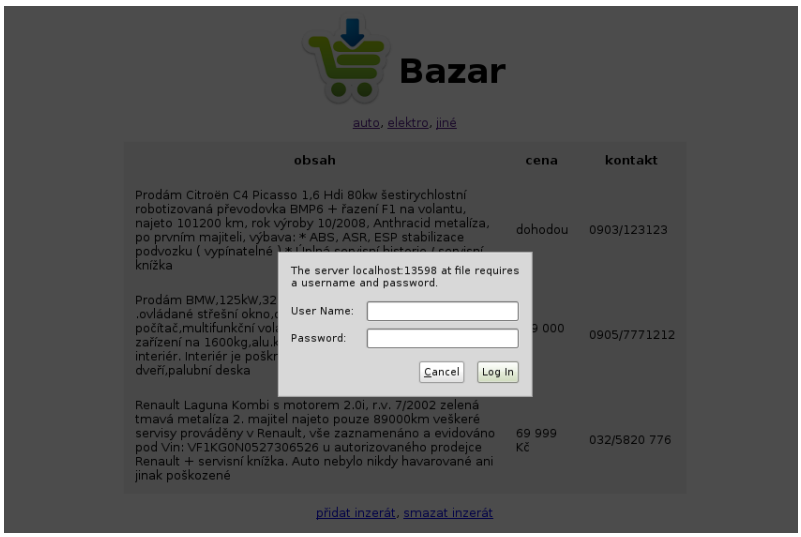
- prezeranie inzerátov radených v kategóriách,
- pridanie inzerátu,
- zmazanie inzerátu.

Obr. 8 Úvodná stránka webovej prezentácie.

Inzeráty sú radené podľa kategórií, ktorých menu je umiestnené nad zobrazenými inzerátmi. Ak chce užívateľ pridať inzerát, zvolí odkaz „přidat inzerát“ v spodnom menu pod zobrazenými inzerátmi. Následne sa mu zobrazí formulár pre vloženie inzerátu.

Obr. 9 Formulár pre vloženie nového inzerátu.

Ak chce užívateľ zmazať existujúci inzerát a zvolí v spodnom menu položku „smazat inzerát“, automaticky je vyzvaný na zadanie mailu a hesla, ktoré zadal pri uverejňovaní svojho inzerátu.



Obr. 10 Prihlasovacie okno pre zmazanie inzerátu.

Na základe overovacích údajov sa mu zobrazia ním zadané inzeráty, ktoré môže následne zmazať. V prípade, ak sa užívateľ prihlási s prístupovými právami ako správca, zobrazia sa mu všetky inzeráty, ktoré môže následne zmazať.

Obr. 11 Všetky inzeráty prihláseného užívateľa.

Táto webová stránka slúži na demonštráciu užitočnosti JavaEE nástrojov umožňujúcich rýchlu a efektívnu tvorbu webov.

ZÁVER

Komplexné porovnanie existujúcich vývojových nástrojov pre daný programovací jazyk je hodnotným zdrojom informácii pre každého záujemcu v danej oblasti. Napriek širokej popularite JavaEE platformy, takýto ucelený test medzi českou a slovenskou Java komunitou absentuje, a to bolo dôvodom pre vykonanie tohto testu.

Pre rozsiahlosť a veľký počet v súčasnosti existujúcich frameworkov boli pre test vybrané najpoužívanejšie z nich, teda tie, o ktorých sa najčastejšie vedú diskusie v komunitných fórach. Ďalším kritériom bola ich spoločná vlastnosť - MVC návrhový model.

Z poznatkov v teoretickej časti vyplýva, že vzájomná podobnosť vlastností a funkcionalít testovaných frameworkov nie je náhodná. Jej základom je úzka väzba na API samotnej JavaEE architektúry, ale aj historický vývoj týchto nástrojov, pričom vývoj jedného bol inšpirovaný iným, už existujúcim. Ako napríklad v prípade Spring a Struts.

V úvode praktickej časti boli prezentované funkčné príklady všetkých testovaných frameworkoch, z ktorých je možné usúdiť stupeň náročnosti na programátorské znalosti. Zároveň tieto ukážky môžu poslúžiť ako referenčné návody pre začínajúcich programátorov.

Na základe výsledkov získaných z porovnaní a funkčných testov som dospel k záveru, že kombinácia viacerých komponent od rôznych dodávateľov je väčším prínosom pre výsledný produkt, než striktné používanie nástrojov jedného z nich.

Ako ukážka možností testovaných frameworkov a vhodnosť ich vzájomne kombinovať, bola v závere praktickej časti prezentovaná webová aplikácia. Táto aplikácia bola zostavená kombináciou JSF, Spring Secure a Hibernate komponent, čo dokazuje možnosť a vhodnosť kombinácie rôznych nástrojov.

Napriek rozsiahlosti tejto platformy a krátkosti času, kedy som mal možnosť sa oboznámiť s JavaEE platformou, dúfam že ucelený test nástrojov určených tejto technológii spolu s praktickými príkladmi a ukážkovou aplikáciou presvedčili čitateľa o potrebe existencie tejto platformy a rozličných frameworkov pre ňu určených.

ZÁVER V ANGLIČTINE

A comprehensive comparison of existing developmental implements for given programming language is a valuable source of information for every interested person in a given field. This kind of test is missing between the Czech and Slovak Java community and that was the reason for the test execution, despite the extensive popularity of Java EE platform.

Because of the extensiveness and huge number of existing frameworks were chosen the most used ones, which are the most often discussed in community forum. Another standard was their common quality, a MVC proposal model.

From the knowledge of theoretical section follows, that a mutual similarity of qualities and functions of tested frameworks is not coincidental. Its foundation is close bond on API of Java EE architecture, but also historical development of these implements, when the development of one of them was inspirited by the existing one. As for example in a case of

Spring and Struts.

At the beginning of practical section were presented functional models of all tested frameworks, from which it is possible to conclude the degree of demand at programmer knowledge. These presentations could help as the reference instructions for beginning programmers.

I came to the conclusion, that combination of several components from different suppliers is bigger contribution for resulting product, than strict use of implements from one producer according to results obtained from comparisons and functional tests.

The web application was presented at the end of practical section as the illustration of possibilities of tested frameworks and their suitability for their mutual combination. This application was assembled by combination of JSF, Spring Secure and Hibernate component, which does prove the possibility and suitability of combination of several implements.

Despite the extensiveness of this platform and shortness of time, when I had possibility to acquaint with Java EE platform, I do hope that compact test of implements given to this technology together with practical examples and sample application convinced the reader about the necessity of existence of this platform and different frameworks to be intended for her.

ZOZNAM POUŽITEJ LITERATÚRY

Monografia:

[1] BOLLINGER, Gary; NATARAJAN, Bharathi. *JSP - Java Server Pages : podrobný průvodce začínajícího tvůrce webu.* Praha : Grada Publishing a.s., 2003. 420 s. ISBN 8024703408.

[2] HALL, Marty. *JAVA servlety a stránky JSP.* 1. Praha : Neokortex, 2001. 586 s. ISBN 80-86330-06-0.

[3] GIULIO, Zambon; SEKLER, Michael. *Beginning JSP, JSF and Tomcat Web Development : From Novice to Professional.* 1. [s.l.] : Apress, 2007. 275 s. ISBN 978-1-59059-904-4.

[4] GONCALVES, Antonio. *Beginning Java EE 6 Platform with GlassFish 3 : From Novice to Professional.* 1. [s.l.] : Apress, 2009. 500 s. ISBN 978-1-4302-1954-5.

[5] TAYLOR, Art; BUEGE, Brian; LAYMAN, Randy. *Hacking bez tajemství: Java a J2EE*. Brno : Computer Press, 2003. 440 s. ISBN 80-7226-868-6.

[6] PECINOVSKÝ, Rudolf. *Myslíme objektově v jazyku Java 5.0.* První vydání. Praha : Grada Publishing a.s., 2004. 604 s. ISBN 80-247-0941-4.

[7] GONCALVES, Antonio. *Beginning Java EE 6 Platform with GlassFish 3 : From Novice to Professional*. Second Edition. [s.l.] : Apress, 2010. 500 s. ISBN 978-1-4302-2890-5.

[8] LINWOOD, Jeff; MINTER, Dave. *Beginning Hibernate : From Novice to Professional*. [s.l.] : Apress, 2006. 280 s. ISBN 978-1-59059-693-7.

Internetové zdroje:

[9] *Spring Framework, Reference Documentation* [online]. [cit. 2011-05-02]. Dostupný z WWW: <http://static.springsource.org/spring/docs/current/spring-framework-reference/html/index.html>

[10] *Stripes documentation* [online]. [cit. 2011-05-02]. Dostupný z WWW: <http://www.stripesframework.org/ display/stripes/Documentation>

[11] *Apache Struts 2 Documentation* [online]. [cit. 2011-05-03]. Dostupný z WWW: < http://struts.apache.org/2.2.3/docs/home.html>

[12] *Java EE & Java Web Learning Trail* [online]. [cit. 2011-05-05]. Dostupný z WWW: <http://netbeans.org/kb/trails/java-ee.html>

[13] *Category:OWASP Java Project* [online]. [cit. 2011-05-05]. Dostupný z WWW: <https://www.owasp.org/index.php/Category:Java>

[14] *Comparison of Web application frameworks* [online]. [cit. 2011-05-03]. Dostupný z WWW: http://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks

ZOZNAM POUŽITÝCH SYMBOLOV A SKRATIEK

ACL Access Control List - zoznam pre riadenie prístupu k danému objektu, napríklad súboru

AOP Aspect Oriented Programming - aspektovo orientované programovanie

API Application Programming Interface - knižnica tried určených pre konkrétny programovací jazyk

CGI Common Gateway Interface

DAO Data Access Object

EJB Enterprise Java Beans - serverové komponenty vychádzajúce z klasických Java Beans

EL Expression Language - skriptovací jazyk umožňujúci prístup k Java Beans z JSP

FTP File Transfer Protocol - protokol pre prenos súborov v sieti

HTTP Hypertext Transfer Protocol - protokol definujúci výmenu dokumentov HTML formátu

IDE Integrated Development Environment - softvér s integrovanými vývojovými nástrojmi

IoC Inversion of Control - návrhový vzor architektúry softvéru

JAAS Java Authentication and Authorization Service - prihlasovací systém Javy

JavaEE Java Enterprise Edition - Java platforma pre vývoj rozsiahlych informačných systémov

JDO Java Data Objects - aplikačné rozhranie pre perzistenciu dát v Jave

JDBC Java DataBase Connectivity - rozhranie pre prístup k relačným databázam

JPA Java Persistence API - API umožňujúce objektovo relačné mapovanie ORM

JSF JavaServer Faces - skriptovací jazyk XML formátu, umožňujúci prístup k Java Beans; je náhradou za JSP

JSP JavaServer Pages - technológia umožňujúca implementáciu servletov v HTML stránkach

LDAP	Lightweight Directory Access Protocol - protokol pre prístup k dátam na aplikačnom serveri
MVC	Model View Controller - návrhový model pre tvorbu softvéru
ORM	Object Relational Mapping - konverzia dát medzi relačnou databázou a objektovým jazykom
POJO	Plain Old Java Object - bežné objekty Javy, ktoré nie sú Java Beany
UI	User Interface - užívateľské rozhranie
URL	Uniform Resource Locator - adresa zdroja v sieti internet
XML	eXtensible Markup Language - značkovací jazyk; je zovšeobecnením jazyka HTML
XUL	XML User Interface Language - formát pre tvorbu multimediálneho grafického rozhrania
XSS	Cross-Site Scripting - metóda prieniku do web stránky cez jej neošetrené vstupy

ZOZNAM OBRÁZKOV

ZOZNAM TABULIEK

ZOZNAM PRÍLOH

PI: Disk CD s diplomovou prácou a zdrojovými kódmi aplikácie