VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Systém pre management úloh

Task management system

2015

Tomáš Rybnický

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Tomáš Rybnický**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Systém pro management úkolů
Task Management System

Zásady pro vypracování:

Cílem práce je vytvořit systém pomocí technologie JavaEE umožňující správu úkolů a projektů ve spolupráci se softwarem pro správu verzí.

Program bude umožňovat:
1. Evidenci projektů, úkolů a uživatelů.
2. Rozdělení uživatelů do rolí a přidělování oprávnění editace a viditelnosti mezi uživateli a úkoly.
3. Evidenci vztahů mezi úkoly (číselník typů vztahů bude možno rozšiřovat o nové typy).
4. Přidávat komentáře k jednotlivým úkolům.
5. Přiřazování typu, kategorie a stavu úkolům podle typu (číselník typů i kategorií bude možno rozšiřovat).
6. Evidence odpracovaného času k jednotlivým úkolům.
7. Zobrazení přehledu odpracovaného času podle nejrůznějších kritérií.
8. Propojení systému se systémem pro správu verzí. Systém bude sledovat stav systému pro správu verzí a podle předem definovaných komentářů v nových přírůstcích bude aktualizovat stavy úkolů.

Práce bude obsahovat:
1. Implementaci výše popsaného systému.
2. Programátorskou dokumentaci řešení s využitím diagramů jazyka UML.
3. Uživatelskou dokumentaci aplikace.

Seznam doporučené odborné literatury:

[1] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Gang of Four): Návrh programů pomocí vzorů. Grada. Praha 2003. ISBN 8024703025
[2] DARWIN, Ian F. Java cookbook. 2nd ed. Sebastopol, CA: O'Reilly, c2004, xxiv, 829 p. ISBN 05-960-0701-9. Dostupné z: http://it-ebooks.info/book/2249/

Dále podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. David Ježek, Ph.D.**

Datum zadání: 01.09.2014

Datum odevzdání: 07.05.2015

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prehlásenie študenta

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne. Uviedol som všetky literárne pramene a publikácie z ktorých som čerpal.

V Ostrave dňa: *30. júla 2015*

.....
podpis študenta

Poďakovanie

Rád by som poďakoval Ing. Davidovi Ježekovi, PhD. za odbornú pomoc a konzultácie pri vytváraní tejto bakalárskej práce.

Abstrakt

Bakalárska práca sa zaoberá problematikou efektívneho rozdelenia práce pracovníkom, ktorý robia na projektoch. Existuje niekoľko softvérových produktov, ktoré sa tejto problematike venujú, no väčšinou sa jedná o niekoľko programov, ktoré sa musia používať súčasne aby pokryli potreby projektu.

Cieľom mojej práce bolo vytvoriť informačný systém pomocou technológie Java EE, ktorý zastrešuje všetky potrebné nástroje pre projektovú prácu najmä v oblasti IT technológií. Informačný systém obsahuje nástroje pre správu projektov a vytváranie jednotlivých úloh týkajúcich sa projektu, evidenciu odpracovaného času, správa verzií zdrojového kódu a to všetko v jednej aplikácii. Poskytuje plnohodnotný nástroj pre prácu v projektovom tíme. Pre implementáciu som sa rozhodol použiť súčasť platformy Java EE a štruktúru JSF s jeho populárnou nadstavbou Primefaces.

Kľúčové slová

Informačné systémy, podnikové aplikácie, webové aplikácie, server, klient, správa verzií, programovanie, projektový management, MySQL, Java EE, JSF, Primefaces

Abstract

Bachelor thesis if focused to problematics of efficient dividing of workload among team workers working on projects. There are already several software products, but mostly need to be used as multiple programs running so as to cover all project work requirements.

Object of my work is to create system using Java EE technology, which is covering all project management needs mostly within IT industry. Information system contains tools for project management, task creating, evidence of work time and code versioning management and all within one application. It provides useful tool for project tem work. I decided to use part of Java EE platform, framework JSF and its popular extension Primefaces.

Key words

Information systems, enterprise systems, company applications, web applications, server, klient, versioning systems, programming, project management, MySQL, Java EE, JSF, Primefaces

Zoznam použitých skratiek

Skratka	Význam v angličtine	Význam v slovenčine
API	Application Programming Interface	Rozhranie pre programovanie aplikácií
DB	Database	Databáza
IT	Information techologies	Informačné technológie
Java EE	Java Enterprise Edition	
JSF	Java Server Faces	
HTML	Hypertext Markup Language	Značkovací jazyk
CSS	Cascading Style Sheets	Kaskádové štýly
XHTML	Extensible Hypertext Markup Language	Rozšíriteľný značkovací jazyk
OS	Operation system	Operačný systém
JVM	Java Virtual Machine	Java virtuálny stroj
HTTP	Hypertext Transfer Protocol	Protokol pre prenos hypertextových dát
HTTPS	Hypertext Transfer Protocol Secure	Bezpečný protokol pre prenos hypertextových dát
SSL	Secure Sockets Layer	Vrstva bezpečných zásuviek
IDE	Integrated Development Environment	Vývojové prostredie

Zoznam použitých termínov

Termín	Význam termínu
Objektovo relačné mapovanie	Programovacia technika, ktorá zabezpečuje konverziu dát medzi relačnou databázou a objektovo orientovaným programovacím jazykom
Uložené procedúry	Databázový objekt, ktorý obsahuje kód, ktorý sa má nad dátami vykonať
Trigger	Definuje činnosť, ktorá sa má vykonať na základe určenej udalosti nad databázovou tabuľkou
InnoDB	Formát uloženia dát v databázovom systéme MySQL
Java Persistence API	Framework programovacieho jazyka Java, ktorý poskytuje objektovo relačné mapovanie
Framework	Softwarová štruktúra, ktorá poskytuje základné komponenty pre tvorbu ďalších projektov
The Java Database Connectivity API	Java databázová technológia umožňujúca konexiu klienta na databázový server
open sourcee	Vo všeobecnosti akákoľvek informácia voľne dostupná verejnosti za predpokladu že jej voľné šírenie ostane zachované.
Cloud computing	Riešenia v IT, kde sú dáta a programy zdieľané na internete a užívatelia k nim pristupujú pomocou prehliadačov alebo na to určených aplikácií

Obsah

1 Platforma Java EE a jej technológie ... 2

1.1 Viac vrstvové aplikácie ... 2

1.2 Klientská vrstva ... 3

1.3 Webová vrstva ... 3

1.4 Vrstva aplikačnej logiky ... 4

1.5 Dátová vrstva ... 4

2 Relačná databáza ... 5

2.1 Databázový systém MySQL ... 5

3 Použité API a štruktúry ... 6

3.1 JavaServer Faces ... 6

3.1.1 Komponenty užívateľského rozhrania ... 7

3.1.2 Navigácia medzi stránkami ... 9

3.1.3 Triedy ManagedBeans ... 9

3.2 Primefaces ... 12

3.2.1 Nastavenie pred použitím ... 12

3.2.2 Ukážka použitia na stránkach ... 13

3.3 Systém pre správu verzií ... 14

3.3.1 Lokálne systémy ... 15

3.3.2 Centralizované systémy ... 15

3.3.3 Distribuované systémy ... 16

4 Požiadavky na funkčnosť aplikácie ... 17

4.1 Správa projektov ... 17

4.2 Vytváranie a prideľovanie úloh ... 18

4.3 Evidencia odpracovaného času ... 19

4.4 Prepojenie so systémom pre správu verzií ... 20

4.5 Bezpečnosť ... 21

5 Implementácia ... 23

5.1 Databáza a evidované objekty ... 23

5.1.1 Databázová štruktúra ... 23

5.1.2 Objektovo-relačné mapovanie ... 24
5.2 Aplikačná logika ... 26
5.2.1 Enterprise JavaBeans ... 26
5.2.2 ManagedBeans ... 28
5.2.3 Prepojenie so systémom pre správu verzií ... 30
5.3 Užívateľské rozhranie ... 34
5.3.1 Základné komponenty ... 34
5.3.2 Mapy a grafy ... 37
Záver ... 40
Použitá literatúra ... 41
Zoznam príloh ... 42

Úvod

Témou tejto bakalárskej práce bolo navrhnúť a naprogramovať webovú aplikáciu pre správu projektov a s nimi súvisiacich úloh a ich následné rozdelenie medzi pracovníkov. Aplikácia by mala byť predovšetkým zameraná na projektovú prácu v oblasti informačných technológií, no je v podstate jej použitie je možné v akomkoľvek projektovo orientovanom odvetví.

Jednou z najdôležitejších požiadaviek pri vývoji v oblasti IT je vypracovať zadaný projekt v čo najkratšom čase, ale zároveň aby práca na projekte obsahovala všetky potrebné fázy. Pre dosiahnutie týchto hodnôt pri vytváraní produktu je potrebná značná synchronizácia práce medzi pracovníkmi zahrňujúca paralelne prebiehajúce práce na projekte. Avšak človek zodpovedný za projekt musí mať o prebiehajúcej práci dostatočný prehľad, aby mohol rozhodovať a následných krokoch a potrebných veciach pre chod a celistvosť projektu. Pre pracovníkov na projekte by jednotlivé úlohu, ktoré im boli priradené mali dávať dostatočné informácie pre vykonanie požadovanej práce a zároveň zbytočne nezaťažovať náročnou réžiou ich vypracovania v projektovom systéme. Jedným z nepriaznivých faktorov na prácu na projekte je potreba vyhľadávania informácií na viac miestach, čím sa taktiež zvyšuje réžia potrebná pre plnenie úloh. Preto som sa rozhodol vytvoriť túto aplikáciu a poskytnúť tak všetky potrebné funkcie pre správu projektovej práce na jednom mieste.

Práca je rozdelená celkom do piatich kapitol v ktorých popisujem nasledujúce veci. V prvej kapitole sa venujem platforme Java EE, ktorej použitie pre aplikáciu vyplýva zo zadania tejto práce, táto kapitola je teoretická.. V druhej kapitole sa venujem stručnému popisu databázového systém, ktorý je neoddeliteľnou súčasťou každej podnikovej aplikácie. Tretia kapitola je venovaná štruktúram a API, ktoré som sa pre implementáciu aplikácie rozhodol použiť. V štvrtej kapitole rozoberám funkčné požiadavky, ktoré by mala moja aplikácia spĺňať. V poslednej kapitole popisujem samotnú implementáciu programu pri pomocí technológii na platforme Java EE. Venujem sa hlavne špecifickým častiam systému, ktoré by ho mali odlišovať od ostatných systémov venujúcim sa tejto problematike a popisujem problémy na ktoré som pri implementácii narazil. Pri implementácii nepopisujem základné nastavenia a postupy pri vytváraní Java EE aplikácie, pretože rozsahom prevyšuje zadanie bakalárskej práce.

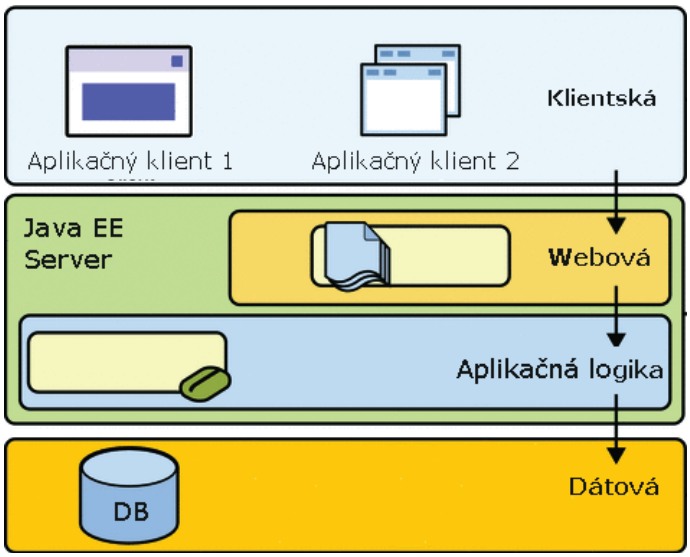
1 Platforma Java EE a jej technológie

Použitá platforma Java EE vyplýva zo zadania tejto bakalárskej práce. Použité súčasti sú ale jednou z kombinácií technológii, ktoré môžu byť pre vývoj webovej aplikácie použité. Ale základné technológie pri webových aplikáciách sú v podstatnej miere zhodné a všetky aplikácie vychádzajú z rovnakých princípov.

Platforma Java EE je rozšírením štandardnej edície platformy Java, pridáva technológie pre viacvrstvové serverové aplikácie. Je navrhnutá tak aby pokryla požiadavky pre vývoj mohutných, škálovateľných, spoľahlivých a bezpečných aplikácií. Tieto aplikácie sú vyžadované najmä pri väčších firemných riešeniach a preto sa nazývajú podnikové aplikácie. Možnosti, ktoré sú od týchto aplikácií vyžadované robia ich vývoj a aplikácie samotné zložitými. Platforma Java EE robí vývoj menej zložitý tým že poskytuje model aplikácií, API, prostredie pre beh aplikácií a umožňuje tak vývojárom sústrediť sa na funkcionalitu a danú problematiku [1].

1.1 Viac vrstvové aplikácie

Základnou myšlienkou je rozvrstvenie aplikácie na menšie celky takzvané vrstvy, takéto aplikácie sa nazývajú aj vrstvené alebo viacvrstvové. Typicky sa viacvrstvové aplikácie skladajú z troch základných vrstiev. Klientská vrstva obsahuje aplikačných klientov, ktorý komunikujú s aplikačným serverom. Aplikačná, ktorá sa rozdeľuje na ďalšie dve pod vrstvy webovú a vrstvu obsahujúcu aplikačnú logiku. A posledná dátová vrstva, ktorá obsahuje dátové zdroje. Platforma Java EE obsahuje technológie, ktoré pokrývajú potreby každej spomenutej vrstvy. Príklad vrstiev podnikovej aplikácie môžeme vidieť na **Chyba! Nenašiel sa žiaden zdroj odkazov.**.



Obrázok 1.1: Štruktúra vrstiev Java EE aplikácií

Takéto rozdelenie aplikácie umožňuje taktiež rozdelenie záťaže na viac serverov, tým že každá vrstva bude bežať na osobitnom serveri a komunikácia medzi jednotlivými vrstvami bude zabezpečená. V nasledujúcom texte popisujem jednotlivé vrstvy a technológie, ktoré sa v nich používajú. Špecifikácie jednotlivých technológií sú dostupné k stiahnutiu z webových stránok korporácie Oracle, jedná sa o rozsiahle dokumenty podrobne popisujúce každú technológiu [4].

1.2 Klientská vrstva

Táto vrstva obsahuje aplikačných klientov, ktorý zvyčajne bežia na inom serveri ako Java EE server. Klienti posielajú na server dotazy, ten ich spracuje a posiela naspäť ako odpoveď klientovi, ktorý dotaz poslal. Aplikačný klienti nemusia byť len Java EE klienti, ale môže to byť webový prehliadač alebo iný server či zariadenie.

1.3 Webová vrstva

Táto vrstva obsahuje komponenty, ktoré sa spracúvajú interakcie medzi klientskym rozhraním a vrstvou aplikačnej. Bežné úlohy komponentov vo webovej vrstve sú napríklad zber dát z klientskej vrstvy a spätné odoslanie správnej odpovede z vrstvy aplikačnej, kontrola navigácie medzi stránkami, alebo udržiavanie dát v rámci užívateľskej relácie [1].

Medzi najčastejšie Java EE technológie používané v rámci aplikačnej logiky viacvrstvovej aplikácie patria:

- **JavaServer Faces (JSF)** - je technológia obsahujúca štruktúru komponentov pre Java aplikácie so špecifikáciou užívateľského rozhrania na strane serveru. Obsahuje API pre reprezentáciu komponentov užívateľského rozhrania, spravovanie ich stavu, reakcie na udalosti, konverziu dát, validáciu dát a navigáciu medzi jednotlivými stránkami, podporuje internacionalizáciu použitých statických textov na stránkach.
- **JavaServer Pages (JSP)** - technológia pre vývoj dynamických webových stránok, ktorá je svojou architektúrou podobná programovaciemu jazyku PHP. Generovanie obsahu stránok funguje na základe odpovede zo Servletov vo webovom kontajnery aplikácie.
- **Expression Language (EL)** - je nástroj pre dynamické vkladanie dát z objektov na strane servera do stránok, taktiež umožňuje volať metódy z JavaBean tried a vykonávať aritmetické operácie.
- **Servlet** - je program napísaný v jazyku Java. Jeho funkcia spočíva v prijímaní dotazov zo strany klienta, ich následné spracovanie na základe aplikačného kódu a odoslanie odpovede späť klientovi, ktorý ich vykreslí užívateľovi.
- **JavaBeans** - sú to triedy napísané v jazyku Java splňujúce dané požiadavky. Medzi tieto požiadavky patri to že trieda musí mať konštruktor bez parametra a k jej vlastnostiam sa pristupuje len pomocou špeciálnych prístupových metód pre získanie a nastavenie hodnoty vlastnosti. Sú používané aj v platforme Java SE, z ktorej Java EE vychádza a rozširuje ju. JavaBeans sú základnými prvkami každej aplikácie pracujúcej s objektami napísanej v jazyku Java.

1.4 Vrstva aplikačnej logiky

Táto vrstva obsahuje komponenty ktoré poskytujú základnú aplikačnú logiku pre danú problematiku. V správne navrhnutej aplikácii je základná funkcionalita systému obsiahnutá v tejto vrstve.

Nasledujúce Java EE technológie patria medzi najčastejšie používané v rámci aplikačnej logiky viacvrstvovej aplikácie.

- **Enterprise JavaBeans (EJB)** - vychádzajú s JavaBeans, úlohou EJB je oddeliť základnú logiku aplikácie od webovej a klientskej vrstvy a vytvoriť znovu použiteľné komponenty zastrešujúce základnú funkcionalitu pracujúcu s problematikou, ktorej sa aplikácia venuje. Sú určené pre beh v oddelenom kontajneri v JVM tzv. EJB kontajneri s ktorým základnú funkcionalitu poskytuje iným komponentom aplikácie.
- **Java Persistence API Entita (JPA Entita)** - je to trieda definovaná na základe dát z databázy, inštancia tejto triedy môže byť uložená do databázy a následne z nej spätne načítaná. JPA Entita reprezentuje tabuľku v databázy. Slúži pre reprezentáciu objektov s ktorými aplikácia pracuje a potrebuje ich ukladať vo forme riadkov to tabuliek do databázy.

1.5 Dátová vrstva

Táto vrstva pozostáva s prostriedkov poskytujúcich dáta pre aplikáciu, ako napríklad databázové servery. Tieto zdroje zvyčajne bežia na oddelenom serveri. V rámci dátovej vrstvy sú použité technológie, ktoré zabezpečujú prístup k týmto zdrojom, a správne spracovanie dát. Tieto technológie nie sú špecifické len pre platformu Java EE, ale používajú sa aj v rámci štandardnej platformy Java SE.

Nasledujúce Java SE/Java EE technológie patria medzi najčastejšie používané v rámci dátovej vrstvy:

- **Java Database Connectivity API (JDBC)** - je to štandard umožňujúci komunikáciu programu napísaného v programovacom jazyku Java rôznymi databázovými a tabuľkovými systémami pre trvalé ukladanie dát. Pomocou JDBC umožňuje spojenie so zdrojom dát, posielanie dotazov na databázový server a spracovanie výsledkov [2].
- **Java Persistence API (JPA)** - je to rozhranie programovacieho jazyka Java pre prácu s objektovo-relačným mapovaním objektov z databázy. Dáta mapuje pomocou entít, ktoré sú reprezentáciou tabuliek v databázy. Zahrňuje aj jazyk pre dotazy tzv. *JQPL (Java Persistence Query Language)*, pomocou ktorého je možné komunikovať s databázovým serverom.
- **Java Transaction API (JTA)** - je to rozhranie komunikácie medzi správcom transakcií a jednotlivými časťami systému distribuovaných transakcií.

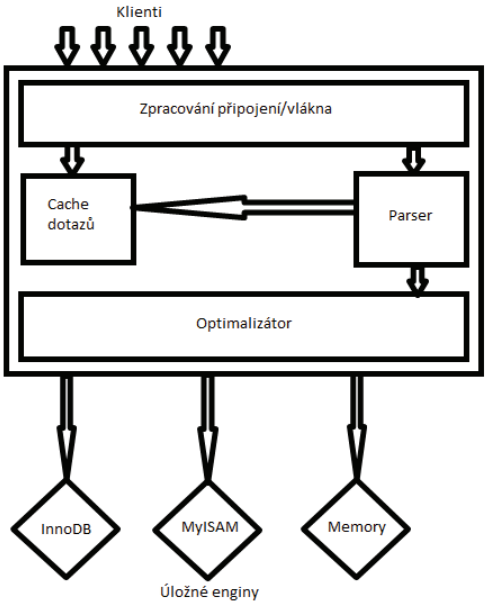
2 Relačná databáza

Relačná databáza slúži ako trvalé úložisko pre dáta s ktorými systém pracuje a potrebuje ich uchovávať pre svoj beh. Dáta sú v databázy uložené v tabuľkách, ktoré sú brané ako entity a jeden riadok v tabuľke je chápaný ako jeden objekt. Medzi entitami môžu byť definované vzťahy. Dáta v uložené v databázy musia byť konzistentné a spĺňať integritné obmedzenia [5].

2.1 Databázový systém MySQL

Jedná sa o multiplatformný databázový systém napísaný v jazyku C a C++, ktorý je orientovaný hlavne na výkon rýchlosť. Pri migrácií databázového serveru je schopný bežať na rôznych platformách OS. V súčasnosti je jeden z najrozšírenejších databázových systémov.

Od verzie 5.0 podporuje aj funkcionalitu známu z iných databázových systémov ako sú uložené procedúry a funkcie, trigger, pohľady a kurzory. Je dostupný ako pod bezplatnou tak ak komerčnou licenciou. MySQL ponúka niekoľko typov úložísk, ktoré sa líšia svojimi možnosťami a spôsobom ukladania dát, pre môj projekt potrebujem podporu transakcií a cudzích kľúčov, ktorú poskytuje formát InnoDB [5]. Jednoduchý popis architektúry MySQL serveru môžeme vidieť na obrázku 2.1.



Obrázok 2.1*: Architektúra MySQL databázového serveru*

So systémom bude databázový server komunikovať prostredníctvom JDBC a objektovo-relačné mapovanie bude zabezpečené pomocou **Java Persistence API** (JPA). Transakcie budú spravované prostredníctvom Java Transaction API (JTA), spomenuté v kapitole *1.5* Dátová vrstva.

3 Použité API a štruktúry

3.1 JavaServer Faces

JavaServer Faces (JSF) technológia je štruktúra obsahujúca komponenty pre Java aplikácie so špecifikáciou užívateľského rozhrania na strane serveru. Obsahuje API pre reprezentáciu komponentov užívateľského rozhrania, spravovanie ich stavu, reakcie na udalosti, konverziu dát, validáciu dát a navigáciu medzi jednotlivými stránkami. Framework podporuje internacionalizáciu použitých statických textov na stránkach. Taktiež poskytuje rozšíriteľnosť pre všetky vyššie spomenuté vlastnosti.

Dobre navrhnutý programový model JSF a knižnica značiek pre komponenty značne uľahčujú namáhavú prácu pri budovaní a spravovaní komplexných aplikácií s prvkami užívateľského rozhrania na strane serveru.

Pomocou JSF môžete jednoducho a s minimálnou námahou uskutočňovať nasledujúce koky v rámci vývoja aplikácie:

- Vkladanie komponentov na stránky pomocou pridávania značiek
- Napájanie udalostí z komponentov na aplikačný kód na strane serveru
- Spájať komponenty užívateľského rozhrania s dátami na strane serveru
- Vytvárať užívateľské rozhranie pomocou znovu použiteľných a rozšíriteľných komponentov

Jedna z najväčších výhod JSF spočíva v prehľadnom oddelení prezentácie od správania aplikácie. Týmto umožňuje vytvárať aplikácie so striktne oddelenými prvkami prezentácie a správania, ako to je známe skôr u aplikácii s architektúrou užívateľského rozhrania na strane klienta. Oddelenie aplikačnej logiky od prezentácie dát umožňuje taktiež rozdeliť prácu na aplikácii medzi dvoch programátorov, ktorý sa môžu sústrediť na svoju časť práce. Alebo umožňujú vytváranie stránok bez znalosti programovania v aplikačnej časti, na stránkach je potrebné len nalinkovať elementy na objekty na strane serveru a to bez písania skriptov.

Typická JSF aplikácia pozostáva s nasledujúcich častí, ktoré môžu byť rozšírené o špecifické prvky pre danú aplikáciu:

- Množina XHTML stránok reprezentujúcich dáta v prehliadači
- Množina špeciálnych tried jazyka Java nazývaných ManagedBeans ktoré definujú vlastnosti a funkcie komponentov na stránke.
- Súbor s konfiguráciou aplikácie, v ktorom sa definujú triedy a navigačné pravidlá
- Popisovač nasadenia na aplikačný server (súbor *web.xml*)
- Objekty vytvorené programátorom aplikácie, môže obsahovať, validátory, konvertory a iné
- Množinu značiek, ktoré reprezentujú komponenty vytvorené ako rozširujúce alebo špecifické pre aplikáciu

Každá JSF aplikácia musí obsahovať mapovanie FacesServlet inštancie. FacesServlet je program napísaný v jazyku Java, prijíma dotazy, ktoré predáva procesorom životného cyklu a inicializuje zdroje pre vypracovanie odpovede na dotaz a následne generuje HTML kód pre stránky. Mapovanie FacesServlet inštancie musí byť špecifikované v popisovači nasadenia, ktorým je súbor web.xml. v následnej ukážke mapovania sa FacesServlet mapuje na všetky URL adresy XHTML stránok prezentujúcich dáta.

```
<servlet>
  <display-name>FacesServlet</display-name>
  <servlet-name>FacesServlet</servlet-name>
  <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>FacesServlet</servlet-name>
  <url-pattern>*.xhtml </url-pattern>
</servlet-mapping>
```

3.1.1 **Komponenty užívateľského rozhrania**

Nasledujú ukážka základu dokumentu stránky XHTML pre použitie s JSF, keďže som pre stavbu užívateľského rozhrania použil nadstavbu Primefaces popis jednotlivých komponentov sa nachádza v kapitole *3.2* Primefaces. Jedná sa o XHTML stránky obsahujúce komponenty z JSF API ktoré sú volané pomocou XML návestí špecifikovaných v základnej značke html. Všetky funkčné prvky užívateľského rozhrania musia byť uzavreté medzi značkami formuláru, ktoré reprezentujú formulár v rámci ktorého môže klient odosielať dáta na server.

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
    <h:body>
        <f:view>
            <h:form> ...  </h:form>
        </f:view>
    </h:body>
</html>
```

Technológia JSF podporuje rôzne knižnice značiek pre vkladanie elementov n webovú stránku. Pre podporu týchto elementov sa používajú deklarácie XML návestí ako to vidíme v ukážke na základnej štruktúry JSF stránky. Deklarácie sa uvádzajú v základnom elemente html dokumentu. Tabulka 1.1: popisuje hlavné knižnice používané pre vytváranie stránok pomocou technológie JSF.

Tabulka 1.1: *Knižnice značiek podporované technológiou JSF*

Knižnica	URI	Prefix	Príklady	Obsah
JSF značky Facelets	http://java.sun.com/jsf/facelets	ui:	ui:component ui:insert	Značky pre vytváranie šablón
JSF značky HTML	http://java.sun.com/jsf/html	h:	h:head h:body h:outputText h:inputText	Všetky značky pre vytváranie komponentov užívateľského rozhrania
JSF základné značky	http://java.sun.com/jsf/core	f:	f:ajax f:attribute	Značky správania a pomocné prvky.

Vkladanie komponentov z JSF API je možné prostredníctvom predefinovaných značiek, a ich atribútov špecifikovať vlastnosti a správanie komponentu. Následne sa komponenty môžu previazať s objektami na strane serveru pomocou EL. Ukážka použitia základných elementov obsahujú len značky bez základnej štruktúry dokumentu popísanej vyššie.

```
<h:panelGroup style="width: 100px">
    <h:outputLabel value="Enter name" styleClass="css-class-
output"/>
    <h:inputText value="#{managedBean.name}" styleClass="css-
class-input"/>
    <h:commandButton value="OK" action="managedBean.method()"/>
</h: panelGroup>
```

Ukážka reprezentuje jednoduchý formulár na zadanie mena na textového poľa. Ako vidíme každý element môže mať nalinkovaný vlastný štýl pomocou triedy v kaskádových štýloch, alebo mať vlastnosti štýlu definované priamo na stránke. Nalinkovanie na objekty v triedach v kóde prebieha pomocou EL, pomocou týchto výrazov môže byť dynamicky zobrazená vlastnosť daného objektu alebo volaná metóda.

Výsledná stránka, ktorá vnikne preložením kódu z ukážky XHTML dokumentu je na obrázku**Chyba! Nenašiel sa žiaden zdroj odkazov.**

Obrázok 1.3 : Ukážka jednoduchého JSF formulára

Ako bolo spomenuté stránka zložená z JSF komponentov je stromová štruktúra. JSF podporuje vytváranie šablón a vkladanie elementov z iných s XHTML dokumentov

```
<ui:include src="another_doc.xhtml" />
```

Takto môžeme vytvárať komplexné prvky na stránkach zložené z menších celkov a udržovať tak kód stránky prehľadný. V tomto prípade bude na miesto kde je značka použitá vložený obsah dokumentu *another_doc.xhtml*.

3.1.2 Navigácia medzi stránkami

Po vytvorení stránok je potrebné vytvoriť pravidlá pre navigáciu medzi nimi. Navigačné pravidlá je možné špecifikovať v konfiguračnom súbore *faces-config.xml*. Zostavovanie navigačných pravidiel spočíva v špecifikovaní zdrojovej stránky a cieľovej stránky na základe výstupu. V nasledujúcom príklade navigačných pravidiel v konfiguračnom súbore dochádza k navigácii zo stránky *login.xhtml* na stránku *home.xhtml* v prípade výstupu s hodnotou "ok".

```
<navigation-rule>
    <from-view-id>login.xhtml</from-view-id>
    <navigation-case>
        <from-outcome>ok</from-outcome>
        <to-view-id>home.xhtml</to-view-id>
    </navigation-case>
</navigation-rule>
```

3.1.3 Triedy ManagedBeans

Sú to triedy napísané v jazyku Java vo svojej podstate sa jedná bežné triedy spĺňajúce definíciu JavaBeans, spomínané v kapitole *1.3* Webová vrstva. Oproti bežným JavaBeans objektom sa líšia tým že sú registrované v JSF. Inými slovami, ktoré vystihujú ich názov sa jedná a JavaBeans manažované štruktúrou JSF. Tvoria model pre komponenty užívateľského rozhrania na XHTML stránkach. Rovnako ako JavaBean obsahujú metódy pre prístup k ich vlastnostiam, metódy pre reakcie na udalosti z užívateľského rozhrania, metódy pre validáciu dát a jednoduchú logiku správania sa komponentov na JSF stránkach.

Aby boli ManagedBeans brané ako triedy pre JSF musia byť zaregistrované. Triedy môžu byť zaregistrované dvoma spôsobmi.

- Pomocou XML súboru *faces-config.xml*
- Pomocou anotácii - základná anotácia registrujúca triedu ako ManagedBean triedu je **@ManagedBean**, kde prostredníctvom atribútu **name** definuje meno pod ktorým je trieda registrovaná v kontexte a môže byť volaná pomocou EL v kóde XHTML stránok. Ďalším dôležitým atribútom registrovanej triedy je **eager**. V prípade že je atribút eager nastavený na hodnotu *true* objekt definovaný triedou je vytvorený bez ohľadu na to či je pre chod aplikácie potrebný, v opačnom prípade je vytvorený až keď je prvý krát použitý.

Nasledujúca ukážka je príklad registrácia ManagedBeans v súbore *faces-config.xml*. Registrácia musí obsahovať názov pod ktorým sa trieda registruje, cesta k samotnej triede a špecifikácia rozsahu, ktorý je popísaný v pokračovaní tejto podkapitoly.

```
<managed-bean>
  <managed-bean-name>helloWorld</managed-bean-name>
  <managed-bean-class>com.test.HelloWorld</managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
<managed-bean>
  <managed-bean-name>message</managed-bean-name>
  <managed-bean-class>com.test.Message</managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

Rozsah triedy registrovanej voči JSF určuje rozsah, v ktorom je daný objekt aktívny v rámci aplikácie. Tabulka 1.2: opisuje jednotlivé rozsahy definované v štandarde technológie JSF. Každý rozsah má svoje špecifické využitie a poskytuje isté výhody. Vhodnou kombináciou rozsahov ManagedBeans je možné dosiahnuť výkonovú optimalizáciu aplikácie, a vhodne oddeliť životnosť dát v rámci objektov vytvorených a uchovávaných v pamäti serveru. Ďalej možno rozsahy využiť napríklad pre oddelenie dát v rámci jednotlivých užívateľských relácií.

Tabulka 1.2: *Rozsahy tried ManagedBeans v JSF technológii*

Názov	Anotácia	Popis
Request	@RequestScoped	Objekt vzniká s HTTP dotazom a zaniká s v momente ukončenia HTTP odpovede týkajúcej sa tohto dotazu
None	@NoneScoped	Objekt vzniká a zaniká v rámci vyhodnotenia jedného EL dotazu v rámci volania dát na XHTML stránkach.
View	@ViewScoped	Objekt pretrváva po dobu čo užívateľ pracuje v rámci jedného JSF pohľadu. Vzniká s prvým HTTP dotazom a zaniká pri presmerovaní na nový pohľad.
Session	@SessionScoped	Objekt pretrváva po celú dobu relácie. Vzniká s prvým HTTP dotazom a zaniká pri rušení relácie. Vhodný pre udržanie dát pre prihláseného užívateľa, napr. doba prihlásenia.
Application	@ApplicationScoped	Objekt pretrváva po celú dobu života aplikácie. Vzniká po nasadení aplikácie (v prípade nastavenia atribútu eager na hodnotu true). Vhodný na udržovanie dát pre nutných pre beh aplikácie a zdieľané dáta.

Nasleduje jednoduchý príklad triedy ManagedBean, v ktorej budú uložené dáta, ktoré sa dynamicky zobrazujú na XHTM stránkach JSF aplikácie. Táto trieda bude len jednoduchou demonštráciou známeho príkladu vo svete programovacích jazykov *Hello world!*. Bez parametrický konštruktor triedy je vytvorený automaticky.

```
@ManagedBean(name = "helloWorld", eager = true)
@RequestScoped
public class HelloWorld implements Serializable {
   private String message;
   public String getMessage() {
      if(messageBean != null){
         message = "Hello world!";
      }
      return message;
   }
   public void setMessageBean(Message message) {
      this.messageBean = message;
   }
}
```

Napojenie demonštrujem jednoduchou XHTML stránkou, ktorá obsahuje dáta s ManagedBean triedy s názvom *HelloWorld*, ktorá je registrovaná pod menom *helloWorld*. Táto jednoduchá stránka zobrazí v prehliadači správu *Hello world!*.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
   <title>JSF Tutorial!</title>
  </head>
  <body>
   #{helloWorld.message}
  </body>
</html>
```

Táto demonštrácia fungovania JSF je veľmi jednoduchá a dostatočne popisuje prepojenie a použitie ManagedBeans na stránkach XHTML. Kombinovaním rôznych komponentov užívateľského rozhrania a dát z ManagedBeans v rôznych rozsahoch životnosti v rámci aplikácie je možné dosiahnuť elegantne a prehľadne komplexné riešenia.

3.2 Primefaces

Je to knižnica voľne verejnosti dostupných komponentov, ktoré rozširujú zmienený framework JSF. Táto knižnica bola vyvinutá tureckou spoločnosťou *PrimeTek*, za priameho autora tohto rozšírenia je považovaný Çağatay Çivici, ktorý je členom *JavaServer Faces Experts Group* a dlhoročným lektorom v oblasti vývoja v programovacom jazyku Java zameranom na vývoj webových aplikácií a aplikácií v JSF. Primefaces ponúka širokú paletu komponentov, ktoré majú predpripravené API pre stavbu zložitých užívateľských rozhraní. Obsahuje viac ako 100 komponentov, API s podporou zobrazenia na mobilných zariadeniach, vstavané API postavené na základe JSF AJAX štandardu, viac ako 35 vstavaných šablón štýlu a širokú komunitu vývojárov združených okolo tohto produktu [7]. Pre použitie tejto knižnice rozširujúcej JSF som sa rozhodol na základe analýzy komponentov ktoré poskytuje sama o sebe a navyše poskytuje možnosť definovať vlastné komponenty, ktoré môžu byť zložené z už veľmi komplexných komponentov poskytovaných touto knižnicou.

3.2.1 Nastavenie pred použitím

Primefaces je obsiahnuté v jednom JAR súbore, ktorý je pomenovaný podľa aktuálnej verzie **primefaces-{číslo verzie}.jar**. Aktuálna verzia tejto knižnice má označenie 5.2. Tento

súbor je stiahnuteľný na oficiálnych stránkach Primefaces produktu, alebo pomocou nástroja pre stavbu Java aplikácií *Maven* [8] a to pomocou špecifikovania závislosti v súbore *pom.xml* ako v nasledujúcej ukážke kódu.

```
<dependency>
  <groupId>org.primefaces</groupId>
  <artifactId>primefaces</artifactId>
  <version>5.2</version>
</dependency>
```

Komponenty z tejto knižnice sprístupníme v dokumentoch stránok XHTML nasledujúcou špecifikáciou pridaním k návestiam XML v základnom elemente dokumentu.

```
xmlns:p="http://primefaces.org/ui"
```

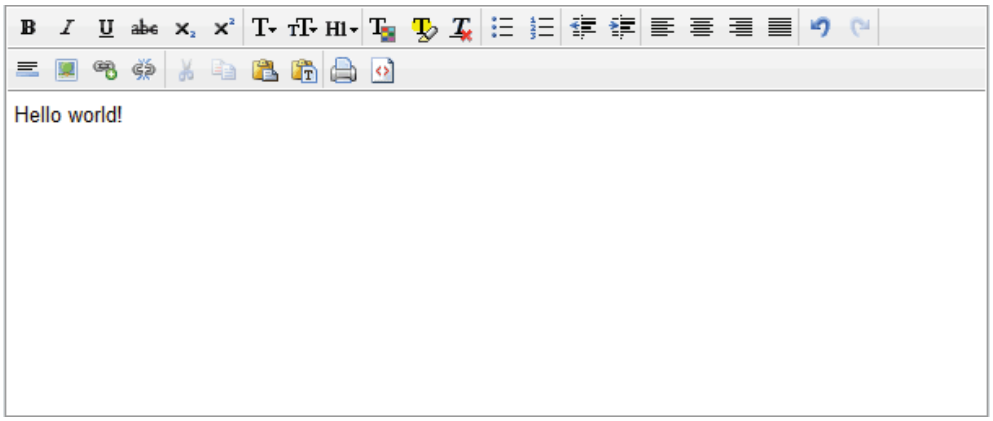
Komponenty sú následne dostupné pod značkami začínajúcimi sa písmenom p nasledovaným dvojbodkou. Príklad textového poľa s hodnotou z ManagedBean objektu registrovaného pod názvom *managedbean*.

```
<p:inputText value="#{managedbean.value}" />
```

3.2.2 Ukážka použitia na stránkach

Po pridaní JAR súboru do Java EE aplikácie a následné pridanie návestí do XHTML súborov s otvárajú možnosti použitia komplexných komponentov obsiahnutých v Primefaces. Komponenty ponúkané knižnicou, poskytujú jednoduché a výkonné riešenie bežných problémov a potrieb pri vývoji webových aplikácií a to jednoduchým vložením komponentu a jeho následným prepojením s objektami na strane serveru. V jednoduchej demonštrácii použitia Primefaces na stránkach XHTML som sa rozhodol ukázať vloženie značne komplexnej komponenty ako je textový editor. Nasleduje ukážka jednoduchého XHTML dokumentu, ktorý na stránke zobrazí textový editor, výstup je na obrázku 3.1.

```
<!DOCTYPE html>
<html xmlns="http://www.w3c.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:p="http://primefaces.org/ui">
<h:head></h:head>
  <h:body>
    <p:editor />
  </h:body>
</html>
```



Obrázok 3.1: Textový editor z knižnice Primefaces

Komplexný komponent textového editoru bol na stránku vložený jednou značkou reprezentujúcou tento komponent v knižnici Primefaces (ukážka neobsahuje previazanie s dátami na strane serveru), avšak to zďaleka nie je všetko, táto knižnica ponúka viac ako 100 komponentov, ktoré sú do XHTML dokumentov vkladané obdobe jednoducho. Jedná sa o komplexné riešenia napr.: galérie obrázkov, nahrávanie súborov, export dát do známych formátov a mnoho iných. Kompletná ukážka komponentov obsiahnutých v knižnici Primefaces je prehľadne zdokumentovaná na webových stránkach tzv. výkladnej skrine knižnice [9]. Na spomínaných stránkach nájdeme ukážku funkčnej komponentu spolu s ukážkou jeho aktuálneho zdrojového kódu na strane XHTML stránok, a zdrojového kódu v jazyku Java na strane serveru v ManagedBeans triedach.

3.3 Systém pre správu verzií

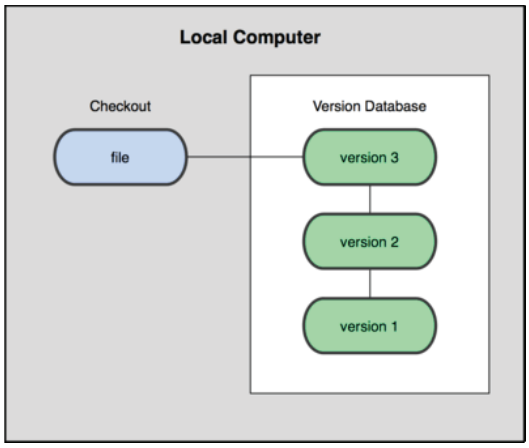
Správa verzií je vo svojej podstate uchovávanie histórie zmien týkajúcich nejakej digitálnej informácie. Potreba správy verzií vyplýva z uľahčenia práca v prípadoch ako je zistenie chyby a následná potreba vrátenia sa k stavu informácií keď neobsahovali chybu. Tento princíp podporujú databázové systémy tým že prebieha záloha dát, v dnešnej dobe riešenia zdieľania dát na internete tzv. *Cloud Computing* a ich poskytovatelia už v základnom balíčku poskytujú ukladanie histórie zmien súborov. Ale toto je správa verzií vo všeobecnosti, avšak najčastejšie využitie nachádza práve v oblasti IT a to pri práci so zdrojovými kódmi. Častým scenárom pri vývoji je úprava kódu, ktorá ho urobí nefunkčným. V množstve zdrojových kódov, s ktorými musia programátori pracovať je priam žiadúce nasadenie nástroja, ktorý by pomohol vrátiť kroky pri zmene kódu späť do stavu v ktorom bol funkčný. Ďalším scenárom je práca v tíme, kde na jednej aplikácii pracuje viac programátorov, v tom prípade nastáva potreba zdieľania hotového kódu pre všetkých členov tímu a z toho vyplývajúce možné konflikty v jednotlivých verziách zdrojového kódu od jednotlivých členov.

Pre spomínané prípady bolo vyvinutých niekoľko systémov pre správu verzií. Tieto systémy nepracujú len so zdrojovými kódmi, ale môžu pracovať s akýmkoľvek digitálne uloženým súborom a evidovať jeho zmeny. Zmeny sú chovávané v repozitároch.

Súbory môžu byť uložené na lokálnom zariadení s ktorým užívateľ pracuje, alebo na centrálnom severy a špeciálnym prípadom je riešenie, pri ktorom sú súbory na servery aj na užívateľských staniciach.

3.3.1 Lokálne systémy

Bývajú väčšinou zabudované priamo v OS (nemusia byť však povolené). Fungujú na princípe vedenia databázy verzií v lokálnom systéme. Užívateľ je tak schopný vrátiť sa k verzii podľa dátumu, alebo poznámky, ktorá bola pri vytváraní jednotlivej verzie uložená. Obrázok 3.2 zachytáva princíp lokálneho systému správy verzií.



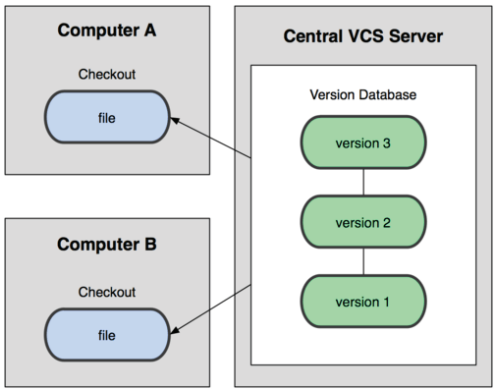
Obrázok 3.2: Lokálna správa verzií

Tieto systémy neriešia problematiku potrieb spolupráce s viacerými užívateľmi na jednom projekte, kde je potreba spravované súbory a ich verzie zdieľať.

3.3.2 Centralizované systémy

Systém obsahuje server, na ktorom sa verzované súbory uchovávajú a z tohto miesta si ich klienti sťahujú do svojich lokálnych systémov. Tieto systémy tak poskytujú prehľad o práci jednotlivých členov tímu, a poskytujú prehľad o zmenách vykonaných jednotlivými pracovníkmi. Centralizovaný systém poskytuje podporu pre riešenie problematiky práce v tíme viacerých pracovníkov.

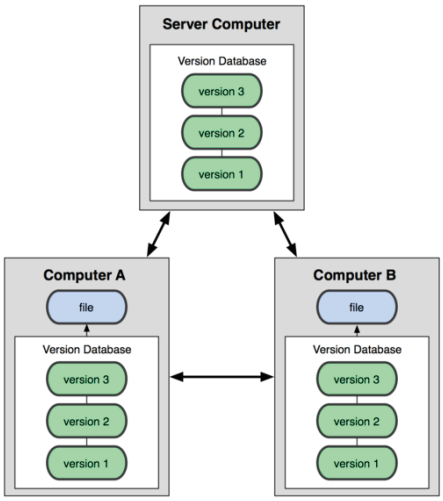
Napriek svojim nepopierateľným výhodám, majú centralizované systémy jednu veľkú slabinu a to je práve centrálny server, ktorý v prípade jeho nedostupnosti neumožňuje ukladať jednotlivým pracovníkom zmeny v súboroch. Dokonca v prípade kolapsu pevných diskov na serveri, kde sú centralizované dáta uložené môže dôjsť k úplnej strate histórie verzií. Obrázok 3.3 popisuje princíp centralizovaného systému správy verzií.



Obrázok 3.3: Centrálna správa verzií

3.3.3 Distribuované systémy

Pre doladenie nedostatku spomínaných centralizovaných systémov boli vyvinuté distribuované systémy. Medzi distribuované systémy patrí napríklad *GIT* [10], ktorý som si zvolil aj pre svoj projekt. V tomto prípade si užívatelia z centrálneho serveru nesťahujú len verzované súbory ale aj celý repozitár obsahujúci aktuálnu históriu zmien vykonaných na verzovaných súboroch. V prípade výpadku centrálneho serveru je v tomto prípade možné obnoviť centrálny repozitár z akejkoľvek užívateľskej stanice. Obrázok 3.4 popisuje princíp distribuovaných systémov pre správu verzií.



Obrázok 3.4: Distribuovaný systém správy verzií

4 Požiadavky na funkčnosť aplikácie

Ako už bolo spomenuté v úvode, jednou z najpodstatnejších vecí pri riadení projektov v oblasti IT a to najmä pri implementácii softwarových riešení v rámci projektu je implementácia v čo najkratšom čase. Časový rámec zohráva dôležitú rolu, hlavne kvôli potrebám zákazníka alebo cieľovej skupiny užívateľov. Konkurenčná firma by mohla software dodať v kratšom čase, alebo zákazník, pre ktorého je software určený môže byť nespokojný. V texte naďalej rozoberám systém ako zameraný na správu projektov v softwarových firmách.

Pre efektívne vedenie projektu je potrebné mať k dispozícii informácie týkajúce sa celého projektu a jednotlivých úloh a pod úloh. Medzi najpodstatnejšie informácie patria:

- Zadanie projektu, obsahuje aj údaje ako rozpočet a predpokladaný čas
- Pridelenie pracovníkov na projekte a prerozdelenie úloh
- Aktuálny stav s popisom a prehľadom kompletnosti partikulárnych úloh
- Čas strávený na implementácii
- Časový predpoklad
- Prehľad financií
- Prístup k dokumentácii

Pre vyššie zmienené potreby riadenia projektu, existuje na trhu už niekoľko riešení, ale žiadne z nich neposkytovalo komplexné riešenie všetkých požiadavkou, čiže samotný pracovník musí pracovať s viacerými nástrojmi aby našiel potrebné informácie týkajúce sa jeho úloh, alebo odpracovaného času. Moja aplikácia je navrhnutá tak aby pokryla všetky potreby riadenia projektov na jednom mieste. S týmito potrebami súvisia nasledujúce funkčné požiadavky na systém správy projektov.

4.1 Správa projektov

Samotná evidencia projektov nezahŕňa len evidenciu samotných projektov, ale taktiež zákazníkov pre ktorých sa projekty implementujú, jednotlivých zamestnancov, ktorý na projektoch pracujú alebo ich vedú. Vlastník firmy, alebo projektový vedúci chce mať prehľad o odpracovanom čase na jednotlivých projektoch pre kontrolu financií a trendov v implementácii v rámci tímu. V rámci správy projektov musí systém splňovať nasledujúce úlohy:

- Evidencia zákazníkov
- Evidencia projektov a úloh (aj správa verzií)
- Evidencia pracovníkov
- Prideľovanie úloh pracovníkom
- Evidencia odpracovaného času
- Prehliadanie štatistických údajov
- Zhromažďovanie dokumentácie na jednom mieste

Projekty v rámci prechádzania fázami projektu, prechádzajú stavmi, ktoré jednoznačne aktuálnu fázu projektu určujú a podávajú tak informáciu a životnom cykle daného projektu. Jednotlivé fázy projektu pri vývoji softwaru sú:

- Plánovanie
- Analýza
- Návrh
- Implementácia
- Testovanie
- Údržba

Stavy projektov by mali byť jednoducho rozšíriteľné v rámci administrácie systému, čiže užívateľ s dostatočnými právami môže základné stavy meniť či rozširovať o nové stavy špecifické v rámci tímu alebo firmy. Základná kolekcia stavovo projektov pozostáva z nasledujúcich stavov, ktoré sú zoradené podľa životného cyklu projektu a majú príznak či sú konečným stavom projektu:

- vytvorený
- otvorený
- analýza
- implementácia
- testovanie
- kompletný
- vyradený
- zamietnutý

4.2 Vytváranie a prideľovanie úloh

Pre samotnú implementáciu riešenia je základným prvkom úloha, ktorá by mala byť jednoznačne zadaná a týkajúca sa jednotného celku. Výsledkom spracovania úlohy v prípade jej splnenia je výstup, ktorý sa líši v závislosti na fázy projektu. Môže to byť napríklad vypracovaná analýza trhu, funkčná časť logickej vrstvy projektu alebo správa a úspešnom/neúspešnom nasadení. Systém by mal preto splňovať potreby pre špecifikáciu zadania úlohy v ktorejkoľvek fázy projektu.

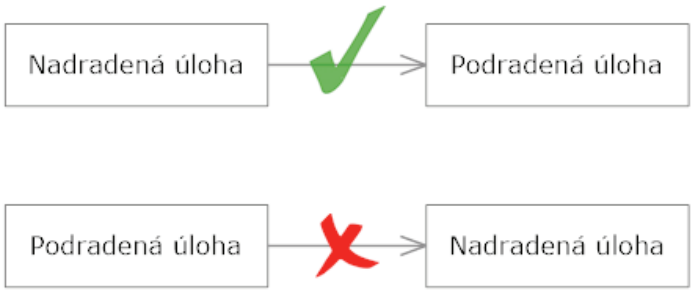
Jednotlivé úlohy sú v rám ci projektu vypracovávané členmi tímu, ktorý za ich riešenie a výsledný výstup nesú zodpovednosť. Za projekt je zodpovedný vedúci projektu, ktorý rozhoduje o chode projektu, vytvára úlohy týkajúce sa zadania a prideľuje ich svojim kolegom alebo podriadeným. Vedúci projektu by mal byť jediná osoba, ktorá má práva upravovať zadanie projektu a úlohy týkajúce sa tohto projektu.

Pri riešení úloh môžu úlohy prechádzať rôznymi stavmi v závislosti na ich percentuálnej kompletizácii, alebo nadväznosti na iné úlohy. Pracovník, ktorý na úlohe pracuje by si mal viesť dokumentáciu o stave projektu a tým podať informáciu zadávateľovi úlohy a takisto aj uľahčovať si prácu pri znovu otvorení úlohy. Do poznámok k úlohe môžu prispievať aj ostatný členovia tímu

a tým môžu prispieť vhodnými informáciami, ktoré napomôžu pri riešení úlohy. Stavy úloh by mali byť jednoducho rozšíriteľné v rámci administrácie systému, takže rovnako ako pri stavoch projektu bude užívateľ s dostatočnými právami schopný základnú sadu stavov upraviť alebo rozšíriť. Základná kolekcia stavov úloh pozostáva z nasledujúcich stavov:

- vytvorená
- priradená
- spracovávaná
- testovanie
- vypracovaná
- zavretá
- zamietnutá

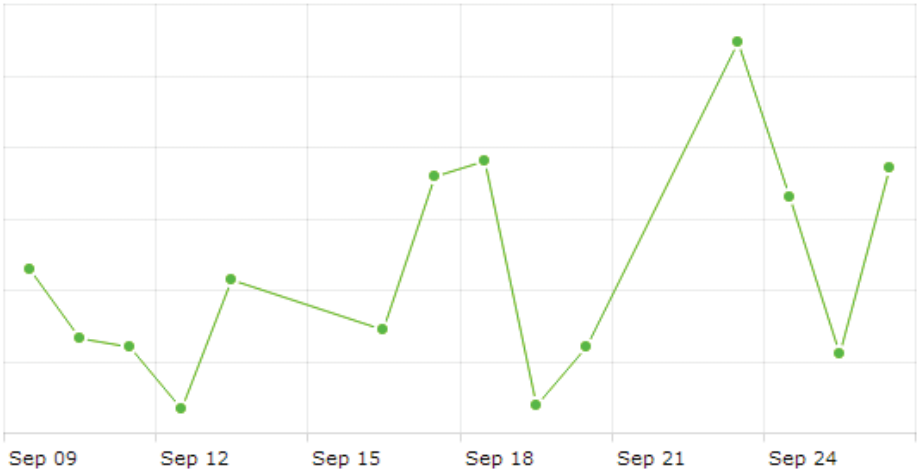
Medzi jednotlivými úlohami môžu vznikať nadväznosti, ktoré môžu byť buď podradené alebo nadradené. Pri podradených nadväznostiach sa daná úloha môže uzavrieť bez ohľadu na úlohy s ktorými je vo vzťahu. Pri nadradených úlohách naopak úloha nemôže byť uzavretá pokiaľ nie sú úlohy, s ktorými je vo vzťahu kompletné. Vzťahy medzi jednotlivými úlohami môžu byť špecifické v pre danú firmu, zákazníka, tím či dokonca projekt. Preto musia byť jednotlivé vzťahy, ktoré môžu byť medzi úlohami nadviazané v systéme voľne rozšíriteľné o nové typy.



Obrázok 4.1: Postup plnenia úloh

4.3 Evidencia odpracovaného času

Novodobým trendom v oblasti IT je takzvaný flexibilný pracovný čas, ktorý je pre zamestnancov veľkou výhodou. Z pohľadu zamestnávateľa, alebo osoby na vedúcej funkcii je ale naopak o to ťažšie kontrolovať, koľko hodín bolo na implementácii projektu či danej úlohy strávených, a keďže zamestnanci môžu pracovať z domu nie je to nekontrolovateľné ani fyzickou prítomnosťou človeka na pracovisku. Práve pre uľahčenie evidencie odpracovaného času by mal systém poskytnúť prostriedky na jeho meranie, prehľad a štatistiky. Pre meranie času bude k dispozícii jednoduchý časovač, ktorý bude spustiteľný v rámci otvorenej úlohy a v reálnom čase môže bežať len jedna inštancia tohto objektu. Po ukončení práce pracovník zastaví časovač a odošle nameranú hodnotu do trvalého úložiska dát, odkiaľ sa namerané hodnoty následné spracujú pre potreby jednotlivých reportov alebo štatistík.



Obrázok 4.2: Príklad reportu odpracovaných hodín

Tento systém evidencie odpracovaného času musí byť podložený aj nejakým výsledkom práce, ktorú zamestnanec vykázal, toto by mal kontrolovať vedúci projektu, ktorý na jednotlivé úlohy stanoví predbežný čas vypracovania, ktorý by mal byť určený na základe trendov a môže byť prekročený len prípade schválenia vedúcou osobou a s opodstatnením uvedeným v poznámkach k úlohe. Štatistiky o odpracovanom čase sú k dispozícii užívateľovi systému, kde si môže sledovať pomer času, ktorý bol prihlásený s časom ktorý odpracoval. Užívatelia s vyššími právami si môžu vygenerovať report pre projekt, úlohu, zamestnanca alebo zákazníka.

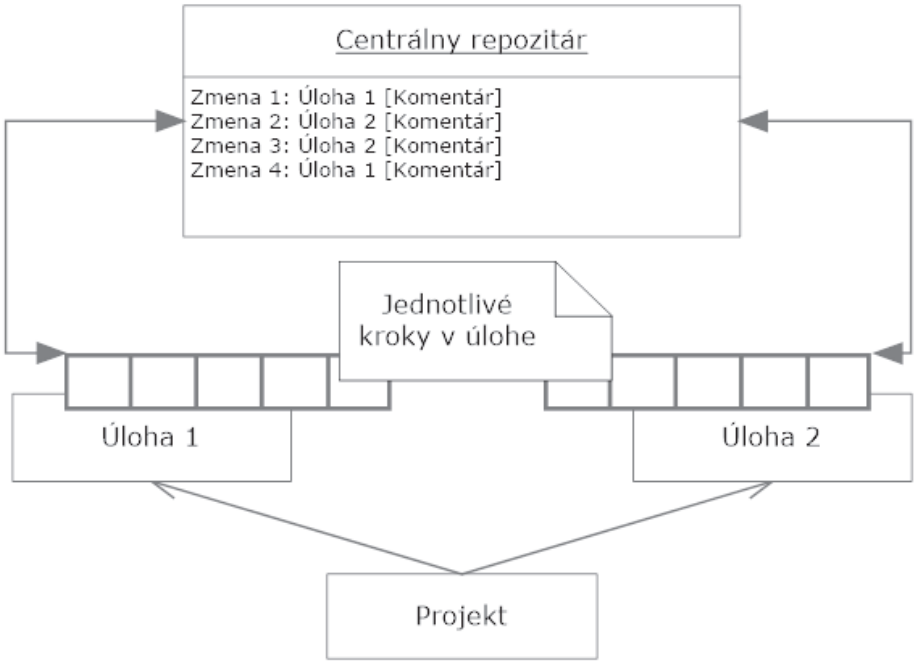
4.4 Prepojenie so systémom pre správu verzií

Kontrola stavu projektu je úzko prepojená s aktuálnym stavom zdrojového kódu, alebo rôznych súčastí projektu ako dokumentácia, či konfiguračné súbory. Plnenie úlohy pozostáva s vytvorenia výsledného produktu, ktorým môže byť funkčná časť projektu, alebo časť dokumentácie. Pri práci na projekte je nutné zdieľať prevedené zmeny s ostatnými členmi tímu a súčasne udržiavať aktuálnosť projektu. Potreba zdieľať svoju prácu, či uložiť jej aktuálny stav vyvstáva pri každom významnejšom kroku, ktorý by mal byť naviazaný na výsledok práce na úlohe. Pre zdieľanie kódu a súčasti projektu je vhodné použiť systém pre správu verzií, popis týchto systémov je zmienený v kapitole *3.3* Systém pre správu verzií. Systém pre správu verzií poskytuje aj mechanizmus prehliadania jednotlivých zmien vykonaných pracovníkmi na projekte, a navyše spätné vrátenie zmien ku stavu projektu pred nimi. Tieto možnosti správy verzií sú užitočné pri prípadnom odhalení chyby v zdrojovom kóde.

Pre prácu so systémom verzií existuje mnoho aplikácií, ktoré sú buď nainštalované do operačného systému, alebo dostupné ako webová aplikácia. Pre obsluhu správy verzií však musíme pristupovať do osobitnej aplikácie, kde pri odoslaní zmeny do centrálneho repozitára stručne popíšeme čoho sa zmena, ktorú odosielame týka. Následne sa vrátiť späť do systému, v ktorom máme zadanú úlohu a uzavrieť ju s obdobným komentárom v poznámke alebo v riešení

úlohy. Toto odporuje základnej myšlienke, ktorej som sa pri navrhovaní aplikácie držal a tou je mať všetky potrebné nástroje na jednom mieste.

Prepojenie môjho systému so systémom pre správu verzií bude je mocnou prednosťou umožňujúcou jednoduchú kontrolu nad aktuálnym stavom úlohy, či projektu. Na obrázku 4.3 je načrtnuté prepojenie jednotlivých zmien vykonaných na projekte v rámci jednej úlohy a zmien v centrálnom repozitári projektu. Každá zmena týkajúca sa uceleného kroku v rámci vypracovania úlohy je naviazaná a zmenu odoslanú do centrálneho repozitára systému správy verzií. Mechanizmus správy verzií podporuje aj riešenie konfliktov pri možnom prepise kódu ktorý zmenil iný pracovník. Prepojením teda získam možnosť vytvorenia a odoslania zmeny len na jednom mieste namiesto minimálne dvoch.



Obrázok 4.3: Prepojenie úloh s jednotlivými zmenami

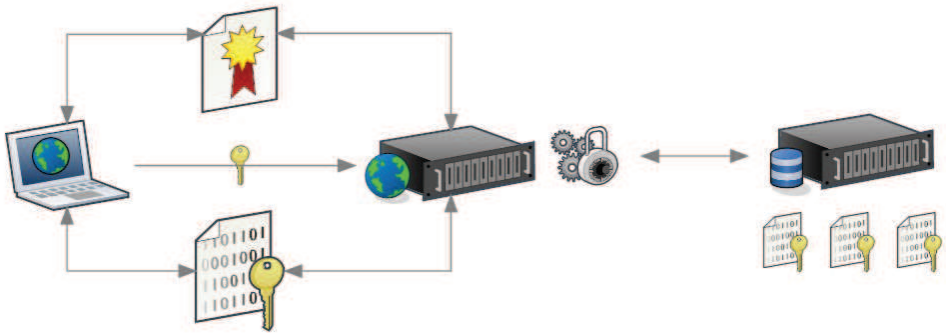
4.5 Bezpečnosť

Práca na projektoch pre zákazníka zahrňuje aj spracovanie rôznych údajov týkajúcich sa osôb na strane zákazníka, alebo celej firmy, taktiež to môžu byť prístupové údaje k systémom. Uchovanie týchto dát pred nepovolanými osobami je jednou z najdôležitejších vecí, ktoré musia byť splnené pre dôveryhodný kontakt so svojimi zákazníkmi. Môj systém by teda mal poskytovať vhodný mechanizmus pre následné bezpečnostné prvky týkajúce sa bezpečnosti pri uchovávaní a spracovaní dát vo webových aplikáciách:

- Šifrovanie prenášaných dát
- Šifrovanie citlivých dát uložených v databáze
- Autentifikácia a autorizácia užívateľov

Šifrovanie prenášaných dát je dôležité pre zabránenie odchytenia dát na ceste medzi webového prehliadača na klientskom počítači a serverom. Možno dosiahnuť použitím vhodného komunikačného protokolu, ktorý dáta šifruje a umožňuje komunikáciu len overeným účastníkom komunikácia, najčastejšie na základe certifikátu. Takýmto protokolom je HTTPS [11], ktorý šifruje dáta pomocou SSL, je nadstavbou bežného protokolu HTTP, ktorý sa používa pre komunikáciu a prenos dát prostredníctvom internetu. Systém musí vhodnou implementáciou podpory tohto protokolu vyhovovať požiadavkám pre bezpečný prenos dát.

Dáta s ktorými systém pracuje sú uložené v databáze odkiaľ môžu byť prečítané neoprávneným prístupom alebo podsunutím príkazov do dotazov smerujúcich do databáze. Preto je vhodné citlivé dáta uložené v databáze zašifrovať algoritmom, prostredníctvom ktorého ich bude vedieť dešifrovať len aplikácia pre ktorú je databáza určená. Po šifrovaní sú dáta uložené ako zdanlivo nezmyselný reťazec, ktorý je možné preložiť do zrozumiteľného textu len na základe daného pravidla - kľúča pomocou ktorého bol zašifrovaný.



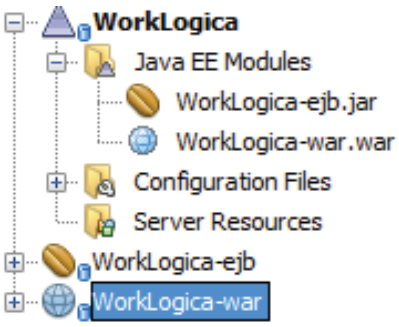
Obrázok 4.4: Schéma šifrovania dát v systéme

Autentifikácia a autorizácia užívateľov pokrýva overovanie identity užívateľov a udeľovania oprávnení prístupu k častiam alebo funkciám systému na základe pridelených práv v rámci systému. Tento proces prebieha pri prihlásení do systému, kde užívateľ používa svoje prihlasovacie údaje, ktoré mu boli pridelené spolu s právami pre jeho profil. Prístup do systému majú teda len povolané osoby, ktoré vlastnia správne prihlasovacie údaje. Samozrejme je možné že prihlasovacie údaje boli užívateľovi odcudzené, vtedy je vhodné užívateľa na prihlásenie sa z inej stanice buď vhodne upozorniť, alebo samotný užívateľ túto udalosť nahlási administrátorovi.

5 Implementácia

V tejto kapitole sú popísané implementácie základných častí aplikácie, ktoré boli implementované v závislosti na funkčných požiadavkách na systém. Kapitola je rozdelená do podkapitol podľa vrstiev trojvrstvovej architektúry webových aplikácii. V kapitole rozoberám len postupy špecifické pre môj projekt. Detailné postupy pre vývoj webových aplikácii pomocou technológie sú oveľa obšírnejšie a nie sú zadaním tejto práce.

Pre implementáciu projektu som zvolil populárne programovacie IDE NetBeans, v ktorom som založil nový projekt ako Java EE Enterprise Aplikáciu, ktorá v sebe obsahuje moduly pre objekty EJB a pre webový kontajner.



Obrázok 5.1: Vytvorený projekt v IDE NetBeans

5.1 Databáza a evidované objekty

Základom každej aplikácie je miesto pre ukladanie dát s ktorými pracuje, a tým je databáza. Databázová štruktúra musí svojim návrhom splňovať základné pravidlá návrhu databáz a potreby aplikácie pre ktorú je navrhnutá. Základný návrh databáze môže byť v priebehu implementácie ostatných vrstiev aplikácie jemne upravený, ale väčšie zmeny si vyžadujú zložité zásahy, ktoré v pokročilej fázy implementácie projektu môžu spôsobiť problémy a spomaliť tak vývoj. Pri návrhu databázovej štruktúry som dôkladne zanalyzoval potreby aplikácie a dodržal pravidlá návrhu databáz a to špeciálne pre použitý databázový systém MySQL [12].

5.1.1 Databázová štruktúra

Návrh mojej štruktúry vychádza z potrieb evidencie objektov popísaných v kapitole *4* Požiadavky na funkčnosť aplikácie a objektov súvisiacich s nimi. Ďalej návrh štruktúry obsahuje tabuľky, ktoré boli vytvorené na základe normalizácie databázy. Stručný popis účelu jednotlivých tabuliek a ich názov v databáze je v tabuľkeTabulka 1.3:.

Tabulka 1.3: *Stručný popis databázových tabuliek*

Názov	Účel
configuration	Konfiguračné dáta systému
credential	Uložené prihlasovacie údaje
customer	Objekty zákazníkov
git_repository	Údaje o repozitároch verzovacích systémov
news_message	Zdieľané správy v systéme
notification	Notifikácie užívateľom
payment	Platobné profily
permission	Oprávnenia v rámci systému
permission_category	Kategórie oprávnení
project	Objekty projektov
project_note	Poznámky k projektom
project_state	Stavy projektov
rating	Hodnotenia užívateľov
relation_type	Typy vzťahov medzi úlohami
role	Role užívateľov
role_to_permission	Vzťah medzi rolami a oprávneniami
salaryticket	Výplatný lístok
sessiontime	Evidencia času prihlásenia v systéme
ticket	Objekty úloh
ticket_note	Poznámky k úlohám
ticket_relation	Vzťahy, ktoré môžu vzniknúť medzi úlohami
ticket_state	Stavy úloh
ticket_to_ticket	Vzťahy medzi úlohami
ticketcategory	Kategórie úloh
tracing	Logovanie operácií nad databázou
user	Objekty užívateľov
user_group	Skupina užívateľov systému
user_to_role	Priradenie role užívateľovi
usercategory	Kategórie užívateľov
worktime	Evidencia odpracovaného času

Kompletný dátový slovník a ER diagram vypracované pomocou dostupných modelovacích nástrojov pre návrh databáz sú súčasťou prílohy tejto práce.

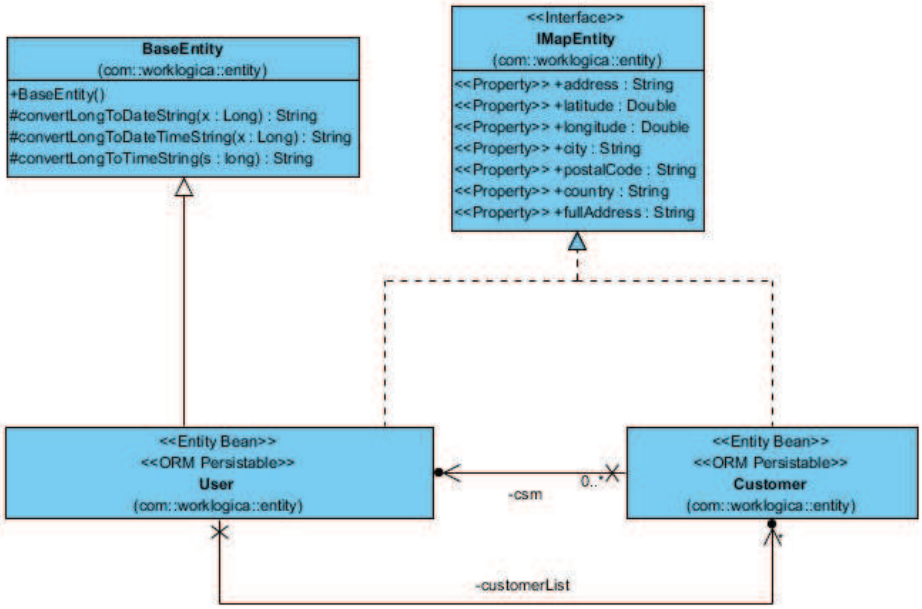
5.1.2 Objektovo-relačné mapovanie

Pre uľahčenie komunikácie medzi aplikáciou databázou sa dáta uložené v tabuľkách mapujú a s prislúchajúcimi vzťahmi na objekty. Objekty slúžia ako reprezentácia dát v databázy. Mapovanie objektov som zvolil podľa potrieb spracovávania údajov a ich následné prezentovanie v prezentačnej vrstve.

Pre mapovanie som zvolil technológiu JPA spomenutú v kapitole *1.5* Dátová vrstva. JPA mapuje tabuľky na entity. Moderné programovacie IDE poskytujú mechanizmy na generovanie základných štruktúr kódu medzi jeden z mechanizmov rozšíreného programovacieho IDE

NetBeans je aj generovanie mapovania entít. Vygenerovaný kód som následne upravil pre potreby mojej aplikácie. Entity som rozšíril o dedičnosť a rozhrania pre uľahčenie práce pri vytváraní komplexných prvkov. Ukážka kódu obsahuje príklad JPA anotácií, ktoré som pri mapovaní entít v dátovej vrstve použil.

```
@Entity
@Table(name = "usercategory")
@XmlRootElement
public class Usercategory implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "id_user_category")
    private Integer id;
    @Basic(optional = false)
    @NotNull
    @Size(min = 1, max = 150)
    @Column(name = "name")
    private String name;
    @OneToMany(mappedBy = "category", fetch = FetchType.LAZY)
    private List<User> userList;

    public Usercategory() {
        this.name = "";
    }
}
```



Obrázok 5.2: Ukážka mapovania entít pomocou JPA

Triedny diagram na obrázku 5.2 zachytáva časť entít ktoré využívajú dedičnosť a rozhrania. Nie je možné zachytiť kompletný triedny diagram, kvôli jeho rozsiahlosti, ale táto ukážka prezentuje základné princípy použité pre mapovaní entít pomocou JPA.

5.2 Aplikačná logika

V aplikačnej vrstve je implementovaná aplikačná logika na strane serveru, ktorá je rozdelená do dvoch pod vrstiev. Základných operácii nad entitami, ktoré som zaobalil do EJB kontajneru vo forme Enterprise JavaBeans komponentov. EJB kontajner môže bežať na separátnom servery v oddelenom JVM. Druhá pod vrstvu aplikačnej logiky je umiestnená v archíve webovej aplikácie tzv. WAR balíčku vo forme špeciálnych Java tried ManagedBeans spomínaných v kapitole *1.4* Vrstva aplikačnej logiky.

5.2.1 Enterprise JavaBeans

V Java triedach EJB je umiestnená základná business logika aplikácie. Základnou logikou myslím operácie nad databázou a s nimi spojené kontroly dát a nastavovanie vlastností entít ako napríklad úpravy kolekcií a objektov v nich obsiahnutých. Metódy implementované sú volané z Managed Beans tried a poskytujú zdroj dát vo forme entít a operácie s entitami pri ukladaní do trvalého úložiska dát.

```
@Stateless
public class CustomerSession extends AbstractSessionClass<Customer> {

    @PersistenceContext(unitName = "WorkLogica-ejbPU")
    private EntityManager em;

    @Override
    protected EntityManager getEntityManager() {
        return em;
    }

    public CustomerSession() {
        super(Customer.class);
    }

}
```

Ukážka kódu obsahuje príklad bez stavovej EJB, ktorá používa tzv. *Persistence Unit* [13] ako prepojenie s dátovým zdrojom nakonfigurovaným na strane webového serveru. Persistence Unit je v kóde reprezentovaný inštanciou objektu manažéra entít. Pomocou metód dostupných z objektu manažéra entít je možné komunikovať cez dátový zdroj s databázou. V ukážke kódu je metóda pre uloženie novej inštancie entity do databázy.

```
@Override
public void create(Customer entity) {
    if (!constraintValidationsDetected(entity)) {
        em.persist(entity);
        entity.getCsm().getCustomerList().add(entity);
        em.merge(entity.getCsm());
        em.flush();
        em.refresh(entity);
        em.refresh(entity.getCsm());
    }
}
```

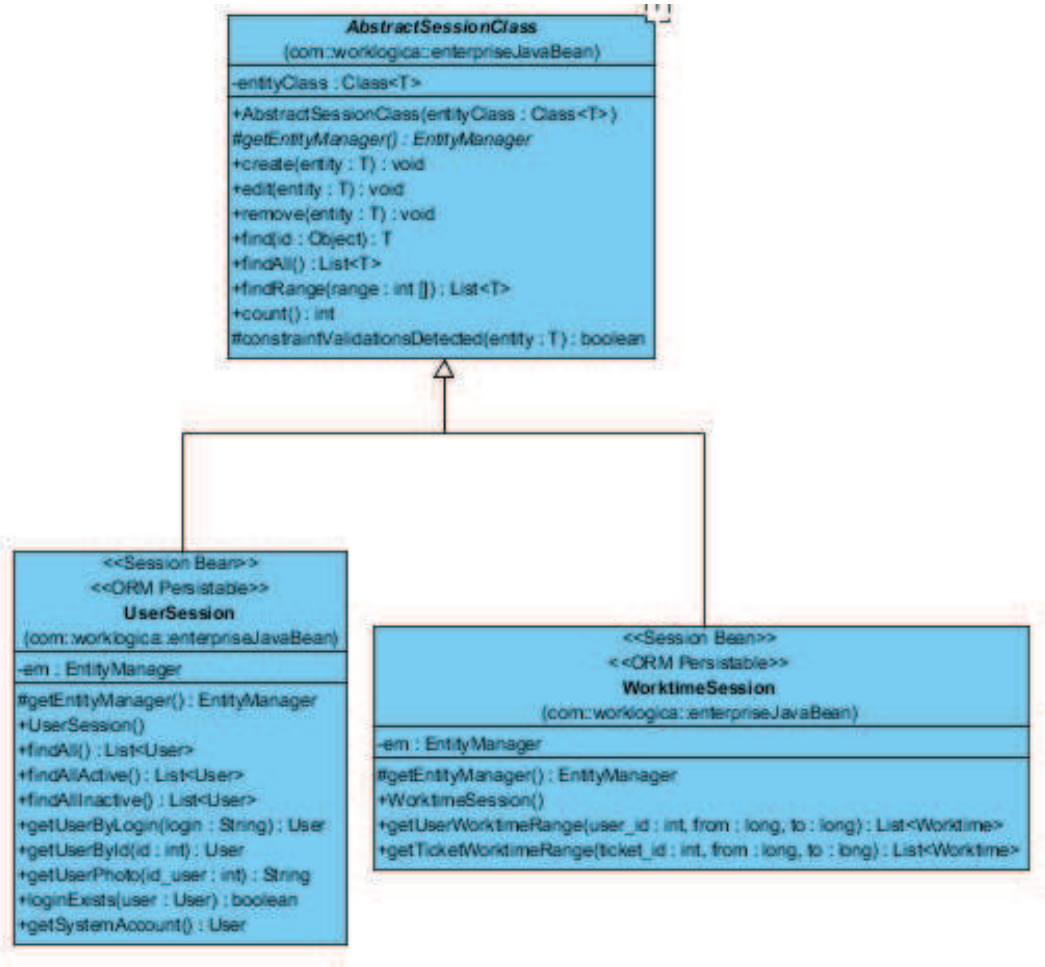
Definíciu Persistence Unit pre dátový zdroj, ktorý som nakonfiguroval vidíme v nasledujúcom kóde, ktorý je uložený v konfiguračnom súbore *persistence.xml* v EJB kontajnery.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
                http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
    <persistence-unit name="WorkLogica-ejbPU" transaction-type="JTA">
        <jta-data-source>jdbc/WorkLogica</jta-data-source>
    <exclude-unlisted-classes>false</exclude-unlisted-classes>
    <properties/>
  </persistence-unit>
</persistence>
```

Pri implementácii som použil abstraktnú triedu, ktorá určuje metódy pre triedy ktoré ju rozširujú. Triedny diagram na obrázku 5.3. Diagram ukazuje príklad dvoch EJB tried, ktoré rozširujú základnú abstraktnú triedu a doplňujú jej metódy o metódy špecifické pre danú entitu.

Ukážky volania EJB v ManagedBeans triedach je v nasledujúcej podkapitole. EJB sú volané v kóde pomocou špeciálnych anotácií @EJB, ktorých požitie je v nasledujúcej ukážke kódu z ManagedBean.

```
@ManagedBean(name = "customerController")
@ViewScoped
public class CustomerBean extends GeolocateEntityBean {

    @EJB
    private CustomerSession customerSession;
    @EJB
    private UserSession userSession;
```



Obrázok 5.3: Triedny diagram pre EJB pre entity

5.2.2 ManagedBeans

Java triedy typu ManagedBeans ako jeden zo základných stavebných jednotiek aplikácií postavených na technológii JSF, plnia funkciu logiky na strane serveru, ktorá sa schováva za akciami z webovej vrstvy. Pripravuje dáta pre zobrazenie a spracúva dáta, ktoré po validácii prišli z dátovej. ManagedBeans podporujú definovanie rozsahu tzv. *Scopes* platnosti pre triedu v rámci životného cyklu aplikácie zmienené v kapitole *3.1.3* Triedy ManagedBeans. Vybrané rozsahy, ktoré vyhovovali môjmu prípadu užitia som vo svojej aplikácii využil nasledovne:

- **Request** - dáta, ktoré potrebujem len pre daný dotaz z užívateľského rozhrania (overenie prihlasovacích údajov, výpočet geolokačných súradníc, odosielanie emailov)
- **View** - udržanie dát v rámci jednej stránky (oprávnenia v rámci stránky, spracovanie formulárov a tabuliek s dátam, štatistické dáta)
- **Session** - udržiavanie dát v rámci prihlásenia užívateľa (počítadlá času, kolekcie objektov týkajúcich sa aktivít užívateľa rámci jedného prihlásenia)

- **Application** - udržiavanie dát po dobu spustenia aplikácie (prihlásený užívatelia, doba spustenia aplikácie, konfigurácia systému)

ManagedBeans som využil ako kontrolné aplikačné prvky pre entity, s ktorými potrebujem pracovať v rámci užívateľského rozhrania, spracovávať dáta z formulárov, zobrazovanie v rámci tabuliek pre zobrazovanie dát. Všetky entity mapované pomocou JPA v dátovej vrstve majú svoju ManagedBean triedu ktorá reprezentuje kontrolný prvok pre entitu.

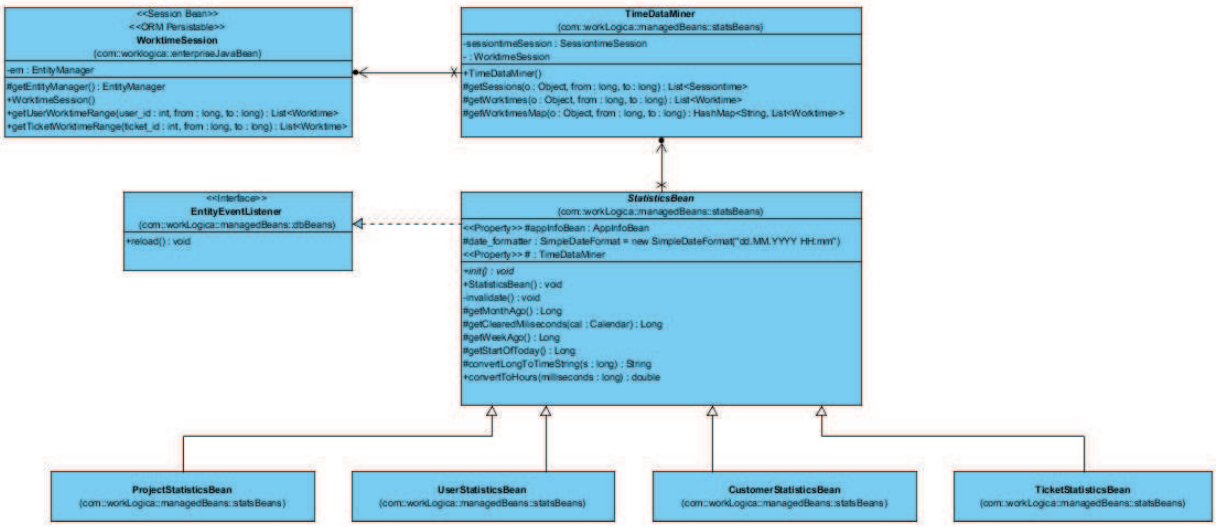
Triedy sú pomocou *@ManagedBean* anotácie naviazané na názov pomocou ktorého pristupujem k vlastnostiam a metódam tried na stránkach v prezentačnej vrstve, kde slúžia na zobrazovanie dát alebo vyvolávanie akcií na základe interakcie s užívateľským rozhraním. Ako bolo spomenuté v predchádzajúcom texte ManagedBeans majú definovaný rozsah platnosti v závislosti na účelu ich použitia v rámci životného cyklu aplikácie. Pomocou EJB, ktoré je mapované pomocou anotácie *@EJB* a vytvorením inštancie EJB príslušnej entity sprístupňujem komunikáciu medzi aplikačnou vrstvou a dátovou vrstvou. Nasledujúca ukážka kódu zobrazuje hlavičku a časť triedy použitej v mojej aplikácii.

```
@ManagedBean(name = "paymentController")
@ViewScoped
public class PaymentBean extends AbstractBasicEntityBean {

    @EJB
    private PaymentSession paymentSession;
    private Payment payment;
    private List<Payment> paymentsAll;
    private List<Payment> paymentsEnabled;
    private boolean isUpdating;

    public PaymentBean() {
        isUpdating = false;
    }

    @Override
    public void reload() {
        this.paymentsAll = paymentSession.findAll();
        this.paymentsAll = paymentSession.getEnabledPayments();
    }
```

Volanie vlastností a metód z ManagedBeans tried je demonštrované v kapitole *5.3.1* Základné komponenty, kde sú ukážky použitia vlastností pre zobrazenie dát a vyvolanie funkcií na základe akcií užívateľa (napr. stlačenie tlačidla vo formulári).

Špecifické využite v rámci mojej aplikácie je v rámci kontrolných tried pre štatistické dáta, ktoré sú spracovávané na základe rôznych kritérií a predávané do prezentačnej vrstvy vo forme vhodnej pre zobrazenie v rôznych grafoch popísaných ďalej v kapitole *5.3.2* Mapy a grafy.

Na obrázku Na obrázku 5.4 je triedny diagram ManagedBeans, ktoré slúžia na spracovanie štatistických dát.

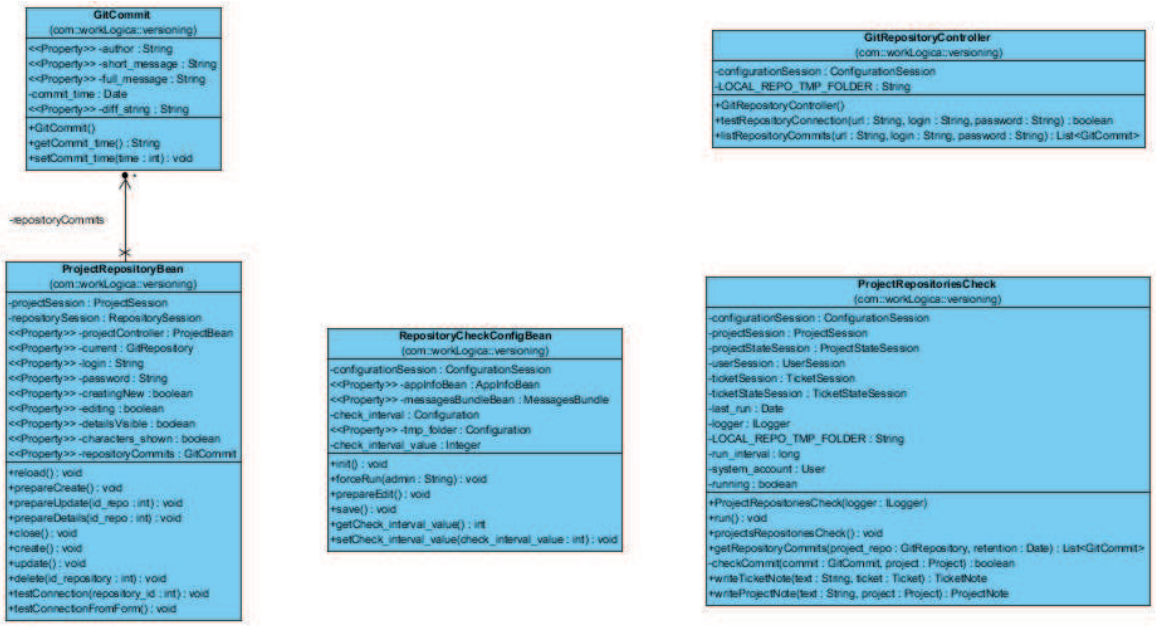


Obrázok 5.4: Triedny diagram ManagedBeans pre štatistiku

5.2.3 Prepojenie so systémom pre správu verzií

Nakoľko prepojenie aplikácie so systémom pre správu verzií vyplýva zo zadania práce venujem tejto podkapitole viac priestoru. Ja som sa kvôli predošlým skúsenostiam rozhodol použiť systém *GIT*. Git je voľne šíriteľný open source distribuovaný systém správy verzií určený pre projekty malého rozsahu až po veľké projekty na ktorých pracuje veľa zamestnancov. Existuje niekoľko implementácií API pre komunikáciu so systémami správy verzií v programovacom jazyku Java. Líšia sa svojou obšírnosťou, možnosťami a cieľovým systémom pre správu verzií. Ako vhodné API som vybral JGIT, je to knižnica ktorá implementuje GIT v jazyku Java.

Pre pripojenie knižnice JGIT som si stiahol JAR súbor, ktorý som pridal medzi knižnice mojej webovej aplikácie. Pri programovaní som využil oficiálnu užívateľskú príručku [14] a prispôsoboval som triedy z knižnice pre moje použitie. Nasledujúci triedny diagram popisuje triedy, ktoré som pre kontrolu repozitárov vytvoril.



Obrázok 5.5: Triedy vytvorené pre komunikáciu s GIT

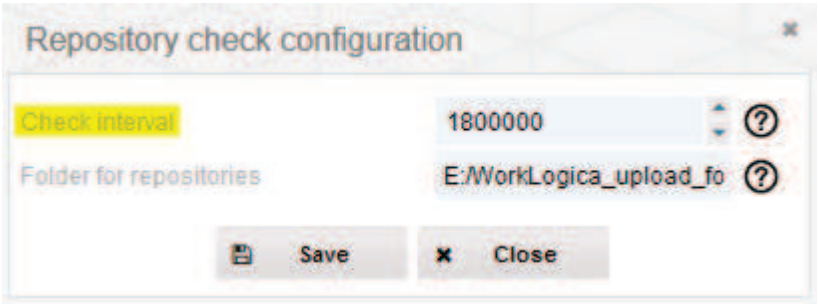
Základom komunikácie je pripojenie sa na repozitár a prechádzať jednotlivé zmeny odoslané užívateľmi. Pre abstrakciu jednej zmeny som vytvoril triedu *GitCommit*, ktorej inštancie reprezentuje jednu zmenu z repozitára.

Pre rýchlejšie prehliadanie repozitárov som sa rozhodol vytvárať kópie v lokálnej zložke v súborovom systéme na servery, na ktorom beží webová aplikácia. Týmto je prehliadanie repozitárov odľahčené tým že pri každom dotaze na obsah sa len aktualizujú zmeny prevedené od posledného dotazu. Taktiež replikami repozitárov vytváram zálohu celého repozitáru.

```
/**
 * Initiate all tasks that need to be run regularly in some schedule
 */
public void startRegularRoutines() {
    if (this.repo_check_fesible) {
        if (this.app_timer != null) {
            this.app_timer.cancel();
        }
        this.app_timer = new Timer();
        ProjectRepositoriesCheck repo_check = new ProjectRepositoriesCheck(this);
        app_timer.schedule(repo_check, 0, this.repository_check_interval);
    }
}
```

Základné využitie systému pre správu verzií som v aplikácii použil pre automatické aktualizácie úloh v systéme na základe kľúčových slov. Pre automatické aktualizácie je potreba spúšťať kontrolu repozitárov v pravidelných intervaloch. Pre túto potrebu som pri inicializácii

ManagedBean, ktorá udržiava existuje po celú dobu životnosti aplikácie vytvoril metódu *startRegularRoutines*. Spomínaná metóda vytvára inštanciu časovača zo základnej triedy jazyka Java *java.util.Timer*, ten spúšťa danú rutinu v pravidelných intervaloch. Interval kontroly repozitárov je uložený v databázy v tabuľke nastavení systému a je možné ho meniť v administrácii aplikácie - obrázok 5.6.



Obrázok 5.6: Ukážka z administrácie systému

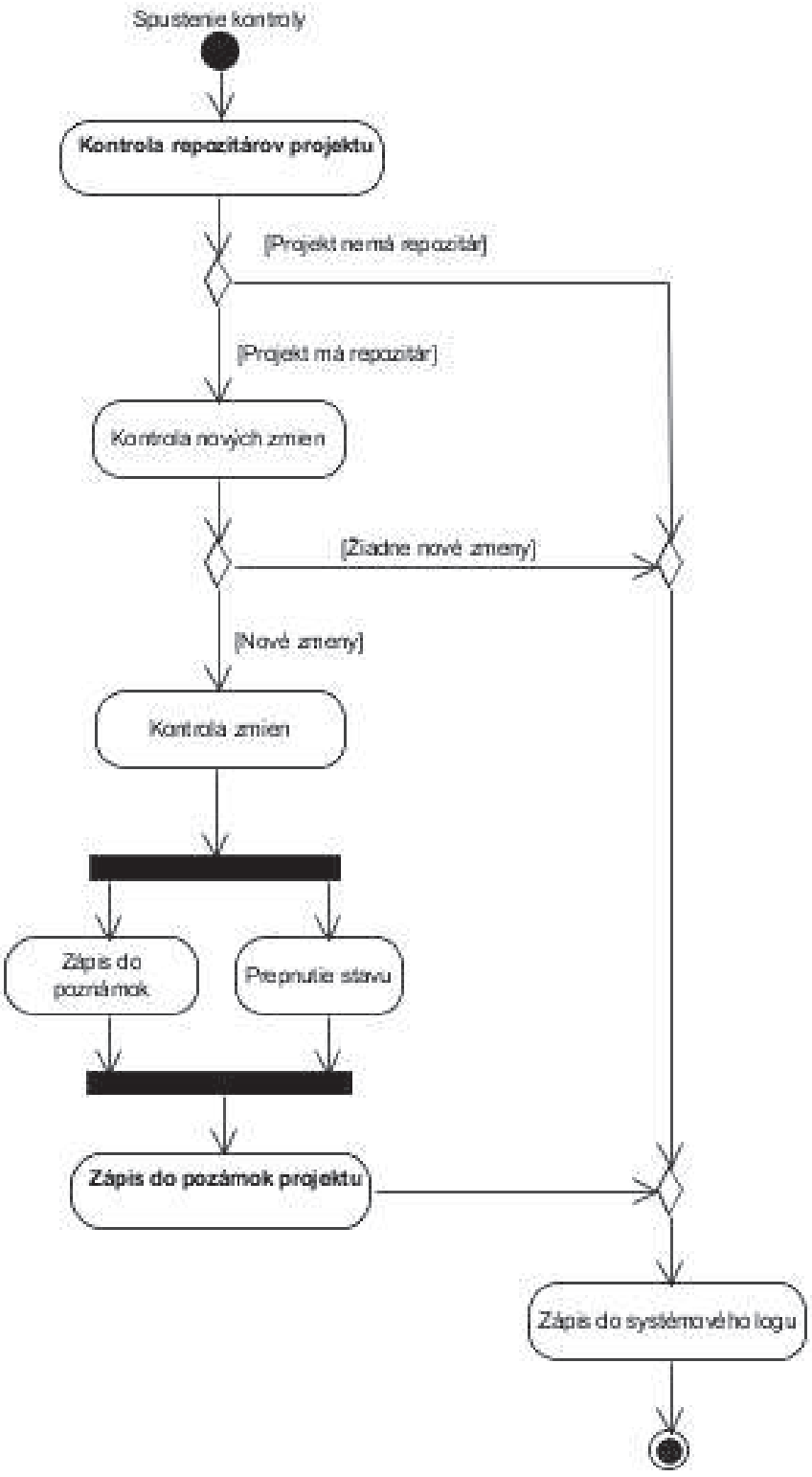
Pre každý projekt je možné definovať jeden a viac repozitárov, ktoré sa následne pravidelne kontrolujú. Postup kontroly repozitárov je popísaný diagramom aktivít na obrázku 5.8. Každá kontrola je zapísaná do systémového logu aj s detailmi ako napríklad počet prejdených repozitárov alebo počet aktualizovaných objektov úloh. Pri zmene stavov úloh sa zapisujú automaticky aj poznámky k projektu a k jednotlivým úlohám.

```
INFO: 17.07.2015 18:47:17 Initial check of repositories started. It will take more time because of fullscan required
INFO: 17.07.2015 18:47:21 Local repository for https://github.com/rybnitom/WorkLogica.git found, new commits w
WARN: 17.07.2015 18:47:25 TCK0000074 - task not found in system. Inform user Jambo Tester - tomas.rybnicky@
INFO: 17.07.2015 18:47:25 Repositories scanned - 1, new commits found - 24 This run of repository check updat
INFO: 17.07.2015 18:47:25 Repository check complete. Next run: Fri Jul 17 19:17:25 CEST 2015
INFO: 17.07.2015 18:48:20 Tomas Rybnicky log on
INFO: 17.07.2015 18:48:41 Tomas Rybnicky entered dashboard
```

Obrázok 5.7: Ukážka logu systému

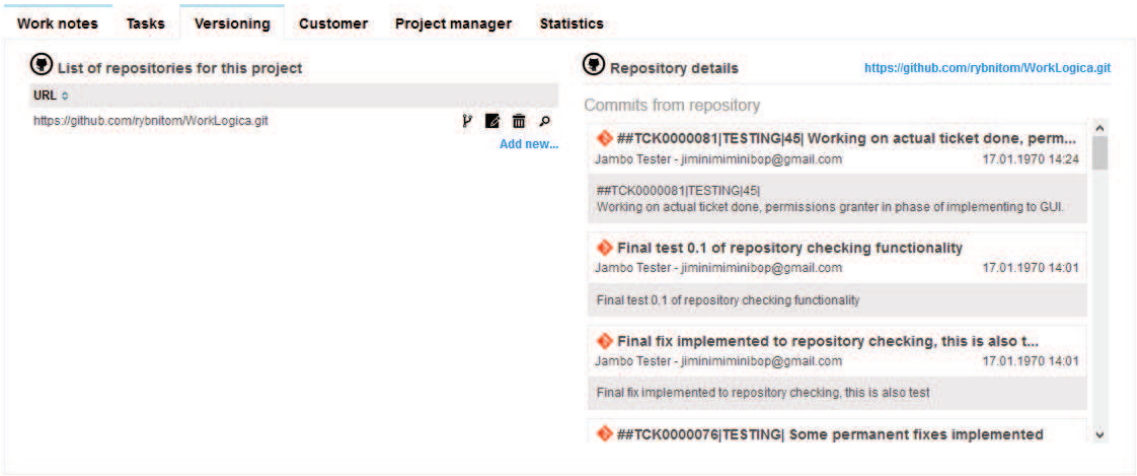
Jednotlivé úlohy a zmeny ich stavov sa kontrolujú na základe predom definovaných hlavičiek správ, ktoré pri odosielaní zmien musia užívatelia zadať. Je možné zadať bežnú správu, v tom prípade sa daná zmena pri kontrole neberie do úvahy. Prvá časť hlavičky je referenčné číslo tiketu v systéme, druhá časť označuje aktuálny stav a tretia voliteľná časť je aktuálna percentuálna kompletizácia úlohy. Nasleduje ukážka hlavičky odoslanej zmeny, za hlavičkou nasleduje bežný text stručne popisujúci zmenu.

$$\#\#TCK0000081|TESTING|45|$$



Obrázok 5.8: Kontrola repozitárov - diagram aktivít

Takto označené zmeny sú kontrolované v pravidelných intervaloch, a užívatelia systému tak môžu zadávať zmeny do systému automaticky spolu s odosielaním zmien v kóde. V zmenách je možné následne aj listovať v užívateľskom rozhraní aplikácie, obrázok 5.9. Toto prehliadanie zmien môže byť užitočné pre spätnú kontrolu manažérom projektu, alebo pre dohľadanie správnej zmeny ku ktorej sa treba v rámci odstránenia zmien zamerať.



Obrázok 5.9: Listovanie v zmenách v repozitári

5.3 Užívateľské rozhranie

Pre implementáciu prezentačnej vrstvy som sa rozhodol použiť knižnicu komponentov Primefaces, ktorá rozširuje základné komponenty JSF. Knižnica komponentov je popísaná v kapitole *3.2* Primefaces. V tejto podkapitole by som sa chcel stručne venovať komponentom, ktoré som pri stavbe užívateľského rozhrania využil. Ďalej poukázať na možnosti využitia knižnice pre vykresľovanie grafov, ktorá je v Primefaces priamo podporovaná a základným prvkom designu, ktorým som chcel aplikáciu urobiť príjemnou pre použitie.

5.3.1 Základné komponenty

Medzi základné komponenty užívateľského rozhrania každej aplikácie patria nasledujúce prvky, ktoré sú aj s bohatými rozšíreniami zahrnuté v základnej knižnici Primefaces:

- dispozícia
- panely
- tabuľka
- navigácia
- textové polia a oblasti
- ostatné vstupné prvky

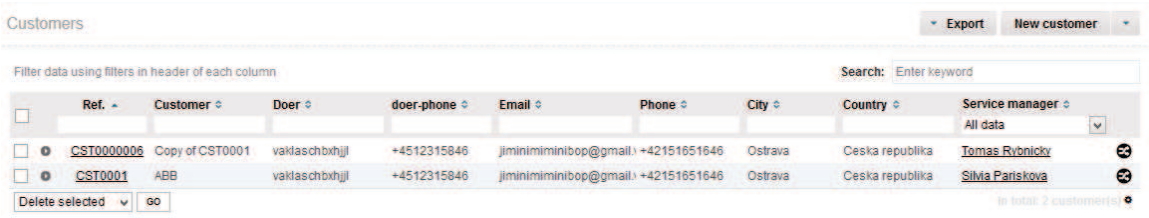
Všetky základné komponenty som pri zostavovaní užívateľského rozhrania mojej aplikácie využil, či už v ich základnej podobe alebo rozšírené o moje vlastné prvky. Najviac ma pri práci z Primefaces zaujali bohaté možnosti komponent tabuľky a panelov, preto tieto prvky a ich použitie v mojej aplikácii priblížim v nasledujúcom texte. Nastavenia potrebné pre zakomponovanie knižnice Primefaces sú popísané v kapitole *3.2* Primefaces.

Nasledujúca definícia tabuľky a jej hlavičky prepája kolekciu dát, v tomto prípade kolekciu objektov zákazníka s tabuľkou a poskytuje jednoduché ale mocné nástroje ako je filtrovanie, zoradenie alebo presúvanie stĺpcov len jednoduchý definovaním základných atribútov komponentu. Prepojenie prebieha na základe volania mena triedy ktorá je na strane serveru reprezentovaná triedou ManagedBean s anotáciou *customerController*.

Výslednú tabuľku doplňuje definícia stĺpcov, ktorá nasleduje po definícii hlavičky. Pri definovaní dát v stĺpcoch sa jedná o prepojenie atribútov objektu *customer* so základnými vykresľovaniami komponentami JSF a Primefaces.

```
<p:dataTable id="customersTable" widgetVar="customersTable"
             var="customer" value="#{customerController.customers}"
             styleClass="contentPanel" style="width: 100% !important; border: none !important;"
             emptyMessage="#{messager.empty_filter_result}" draggableColumns="true"
             rendered="#{not customerController.creatingNew and not customerController.customerEditing}"
             rowExpandMode="single" sortBy="#{customer.referenceNo}"
             rowKey="#{customer.id}" selection="#{customerGroupOperationsBean.selectedObjects}">

    <f:facet name="header">
        <h:panelGrid columns="2" style="width: 100%;" styleClass="no-padding">
            <p:outputLabel value="#{messager.filter_table_message}" styleClass="table-labels"/>
            <h:panelGrid columns="2" styleClass="table-search_panel">
                <h:outputText value="#{messager.search}:  "/>
                <p:inputText id="customer_global_filter" onkeyup="PF('customersTable').filter();"
                             styleClass="table-search-input-field" placeholder="#{messager.enter_keyword}"/>
            </h:panelGrid>
        </h:panelGrid>
    </f:facet>
```

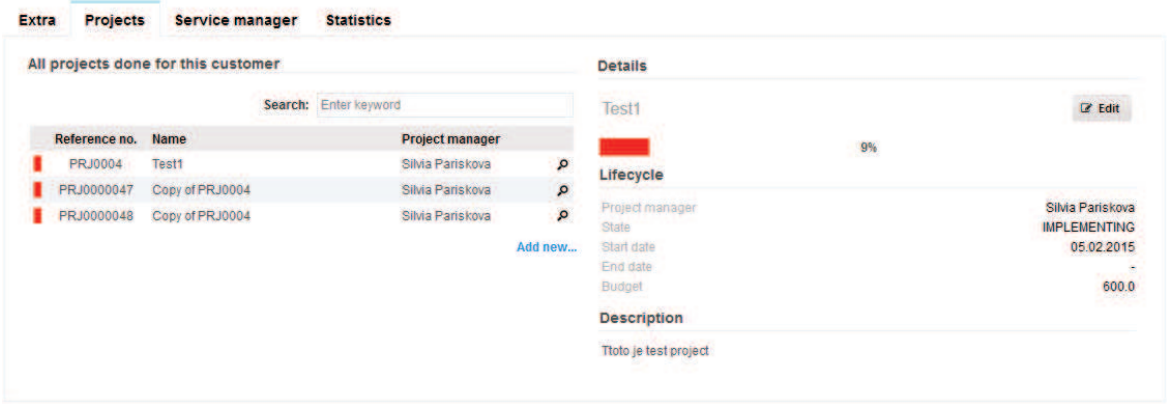
Výsledná tabuľka vyzerá v užívateľskom rozhraní nasledovne, víz obrázok 0. V celku komplexný prvok rozhrania pri využití jednoduchej definície použitia komponenty tabuľky z knižnice Primefaces.



Obrázok 5.10: Ukážka komponenty tabuľky z Primefaces

Ďalším komponentom, ktorý ma zaujal je záložkový panel, ktorý podobne ako pri definovaní tabuľky zahŕňa množstvo užitočných funkcií a rôzne animácie, ktoré použijeme jednoduchým nastavením jeho atribútov. Nasledujúca ukážka kódu zobrazuje definíciu panela záložiek na obrázku 5.11. Obdobný panel som využil pre zobrazenie vlastností všetkých objektov v aplikácií, pre toto zobrazenie som sa rozhodol, kvôli jeho kompaktnosti a prehľadnosti tým že poskytuje možnosť oddelenia dát podľa domén.

```
<p:tabView style="width: 100%;" id="customer_form_tabs" orientation="top" styleClass="horizantal-tabs"
        onTabChange="#{appInfoBean.removeContextMessages()}" activeIndex="0">
    <p:tab title="Extra">
        <ui:include src="customer_form_extra.xhtml" />
    </p:tab>
    <p:tab title="#{messager.projects}" rendered="#{not customerController.creatingNew}">
        <ui:include src="customer_form_projects.xhtml" />
    </p:tab>
    <p:tab title="#{messager.customer_service_manager}" rendered="#{not customerController.creatingNew}">
        <ui:include src="customer_form_csm.xhtml" />
    </p:tab>
    <p:tab title="#{messager.stats}" rendered="#{not customerController.creatingNew}">
        <ui:include src="customer_form_stats.xhtml" />
    </p:tab>
</p:tabView>
```



Obrázok 5.11: Ukážka panelu záložiek

Ako je možné vidieť v ukážke kódu Primefaces podporuje šablónovanie, tým robí výslednú aplikáciu prehľadnejšou a pri implementácii a úpravách kódu sa môžem zamerať len na súbor, ktorý vytvára obsah danej záložky.

Knižnica Primefaces obsahuje veľké množstvo zaujímavých komponentov, ktorých popisom by som značne prekročil rozsah práce. Použitiu a predvedeniu nasadených komponentov sa venuje oficiálna stránka ukážok [9].

5.3.2 Mapy a grafy

Zaujímavým prvkom mojej aplikácie sú mapy a grafy, ktoré som použil napríklad pre zobrazenie geografických údajov o zamestnancoch, alebo štatistické údaje o projektoch. Knižnica podporuje vykresľovanie máp a grafov už v základe ja som ale svoju aplikáciu rozšíril o výpočet súradníc na základe adresy s použitím API od spoločnosti Google, toto API je voľne dostupné na adrese:

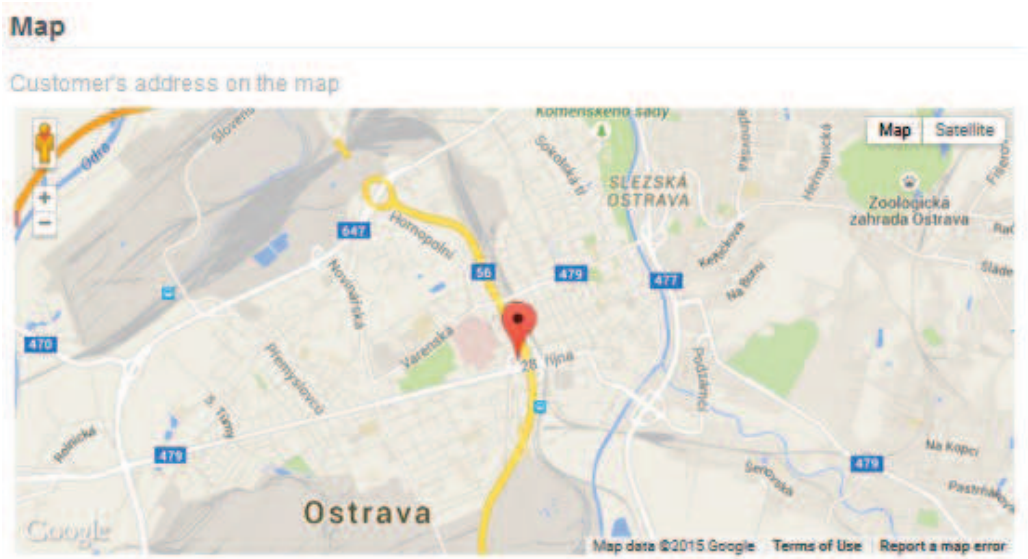
$$http://maps.googleapis.com/maps/api/geocode/xml?$$

Po zadaní adresy v preddefinovanom formáte na koniec adresy, API odosiela XML súbor, ktorý obsahuje základné geo-lokačné informácie ako aj súradnice na mape. Tento súbor následne v triedach ManagedBean, ktoré obsahujú logiku práce s tými to súbormi spracúvam a výsledné informácie sú ukladané do databáze.

Nasledujúci kód ukazuje použitie komponenty mapy z knižnice Primefaces jej prepojenie s objektami obsahujúcimi geo-lokačné údaje na strane serveru.

```
<p:gmap center="#{mapBean.latitude},#{mapBean.longtitude}" zoom="10" type="ROADMAP"
        styleClass="map" model="#{mapBean.simpleModel}"/>
```

Vykreslená mapa je na obrázku05.12, použitie a definícia tejto komponenty je taktiež veľmi jednoduché, preto som mal priestor na rozšírenie funkcionality o automatické vyhľadávanie adresy.



Obrázok 5.12: Ukážka mapy Primefaces

Knižnica Primefaces v základe poskytuje bohaté možnosti zobrazenia dát pomocou rôznych typov grafov, ktoré sú prepojené s inštanciami predpripravených tried. Pre svoju aplikáciu som grafy využil pre zobrazenie štatistických dát spomínaných v kapitole *4.3* Evidencia odpracovaného času. Grafy využívajú zásuvný modul jazyka jQuery pre framework Javascript nazývaný *jqPlot* [15]. JqPlot je testovaný vo všetkých populárnych internetových prehliadačoch a poskytuje jednoduchú konfiguráciu a definovanie štýlu vykresľovaných grafov.

Grafy sú prehľadné a pekne vykreslené už v základnej konfigurácii dodanej knižnicou Primefaces. Pre ich kustomizáciu stačí definovať Javascript funkcie tzv. *extendery* v Javascript súbore, ktorý je importovaný v hlavičke html súboru. Definícia funkcie špecifikujúcej štýl zobrazenia grafu a to vykreslenie mriežky a použitých farieb je v nasledujúcej ukážke kódu.

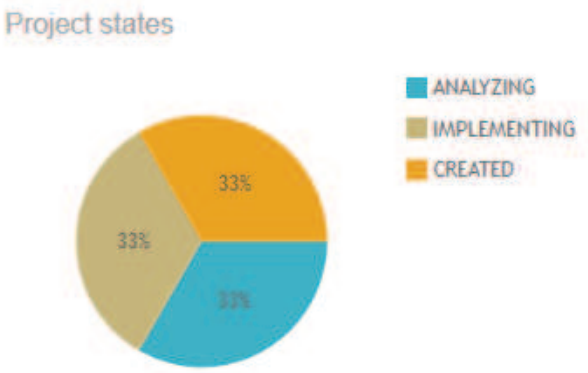
```
function ext() {
    this.cfg.grid = {
        background: 'transparent',
        drawBorder: false,
        shadow: false
    };

    this.cfg.seriesColors = ["#4bb2c5", "#c5b47f", "#EAA228",
        "#579575", "#839557", "#958c12", "#953579", "#4b5de4",
        "#d8b83f", "#ff5800", "#0085cc"];
}
```

Takto definovaný extender je naviazaný na model dát, ktorý je zdrojom pre vykreslené grafy. Modely sú definované v triedach ManagedBean na strane serveru a nastavovaním ich vlastností je možné upravovať výsledné vykreslenie grafu. Následná ukážka kódu je z metódy inicializácie volanej po vytvorení objektu inštancie triedy ManagedBean. V tejto metóde nastavujem vlastnosti kruhového grafu, vlastnosti môžu byť nastavované na základe vlastností iných objektov či kolekcií v rámci aplikačnej vrstvy a zobrazovať tak grafy dynamicky v závislosti na daných objektoch.

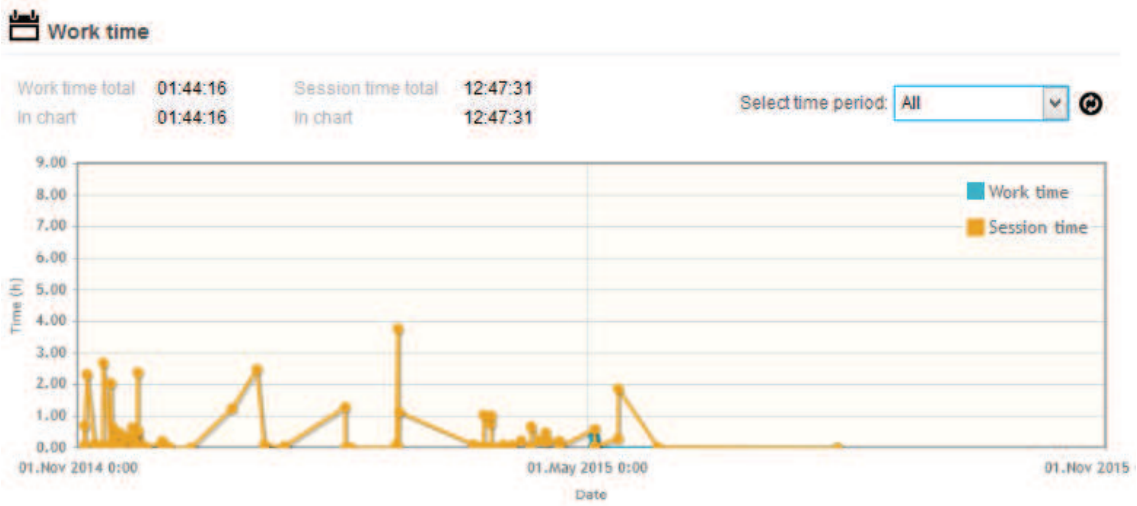
```
@PostConstruct
@Override
public void init() {
    this.customerProjectsChart = new PieChartModel();
    this.customerProjectsChart.setExtender("ext");
    this.customerProjectsChart.setShadow(false);
    this.customerProjectsChart.setMouseoverHighlight(false);
    this.customerProjectsChart.setLegendPosition("ne");
    this.customerProjectsChart.setShowDataLabels(true);
    this.customerProjectsChart.setDiameter(180);
}
```

Prepojenie vyššie zmieneného modelu s prvkami v užívateľskom rozhraní prebieha pomocou Expression Language, v elemente grafu z knižnice Primefaces je nutné zadať typ grafu a na základe použitého typu priradiť model z ManagedBean. Pri nesprávnom modely, ktorý napríklad nepodporuje definíciu priemeru grafu, ktorá je pri vykresľovaní kruhových grafov potrebná hlási framework chybu a graf v užívateľskom rozhraní nevykreslí. Podrobné informácie o chybe môžeme následne nájsť v logu webového servera. Výsledný graf je na obrázku 5.13. Ukážka lineárneho grafu, ktorý vykresľuje štatistické dáta týkajúce sa odpracovaného času je na obrázku 5.14.

```
<p:chart type="pie"
         rendered="#{customerStatisticsBean.customerHasProjects}"
         model="${customerStatisticsBean.customerProjectsChart}"/>
```



Obrázok 5.13: Ukážka kruhového grafu Primefaces



Obrázok 5.14: Ukážka lineárneho grafu Primefaces

Záver

Cieľom tejto práce bolo vytvoriť systém pre správu projektov a úloh pomocou technológií Java EE. Podobných softwarov exituje na trhu mnoho a mojou hlavnou úlohou bolo zapracovať do výsledného riešenia prvky, ktoré ho budú odlišovať. Základným odlíšením systému vyplýva zo zadania, prepojenie so systémom pre správu verzií.

Implementáciu aplikácií pomocou technológii Java EE som si naštudoval a zvolil vhodnú kombináciu technológií na základe požiadaviek na výsledný systém a implementáciu. Základnou technológiou pri vytváraní systému bol framework Java Server Faces a jeho nadstavba knižnica Primefaces. Primefaces sa stáva stále obľúbenejším pre vývojárov podnikových aplikácií, tak som si implementáciu systému chce vyskúšať práve s použitím komponentov z tejto knižnice.

V práci popisujem funkčné požiadavky, ktoré vyplývajú zo zadania a požiadavky, ktoré som v rámci analýzy vyhodnotil. Systém je vytvorený pre prostredia menších softwarových firiem, preto som zohľadnil hlavne potreby pracovníkov v oblasti IT a to najmä vo vývoji software. Ďalej popisujem základné nastavenia, ktoré je potreba vykonať pre použitie zvolených technológií a samotnú implementáciu systému. Pri implementácii som sa zameral hlavne na zaujímavé prvky zvolených technológií, ktoré som si prispôsobil pre môj systém.

Výsledkom mojej práce je systém pre správu verzií, ktorý spĺňa všetky funkčné požiadavky a je užívateľsky prijemný. Jednotlivé prvky systému som pri implementácii testoval sám na webovom serveri, ktorý bežal na mojom počítači spolu s databázovým systémom MySQL. V produkčnom prostredí by aplikácia mala byť rozvrstvená na viac serverov. Systém plánujem priebežne aktualizovať a ladiť chyby. Do budúcna chcem rozširovať funkcionalitu a ďalšie prvky, ktoré by v rámci vývoja software mohli zjednodušiť prácu.

Použitá literatúra

[1] EVANS, Ian. Java Platform, Enterprise Edition: Your First Cup: An Introduction to the Java EE Platform. [online]. [cit. 2015-04-25]. Dostupné z: https://docs.oracle.com/javaee/7/JEEFC.pdf

[2] CRAWFORD, William, Jim FARLEY a David FLANAGAN. Java Enterprise in a Nutshell. 2nd Edition. O'Reilly Media, April 2002. ISBN 978-0-596-00152-0.

[3] JENDROCK, Eric a Jennifer BALL. The Java EE5 Tutorial. [online]. [cit. 2015-04-26]. Dostupné z: http://docs.oracle.com/javaee/5/tutorial/doc/javaeetutorial5.pdf

[4] ORACLE. Java EE 7 Technologies. [online]. [cit. 2015-04-27]. Dostupné z: http://www.oracle.com/technetwork/java/javaee/tech/index.html

[5] ORACLE CORPORATION. MySQL Documentation: MySQL Reference Manuals [online]. [cit. 2015-04-25]. Dostupné z: http://dev.mysql.com/doc/

[6] MAHMOUD, Qusay H. Developing Web Applications with JavaServer Faces. [online]. [cit. 2015-04-26]. Dostupné z: http://www.oracle.com/technetwork/articles/javase/javaserverfaces-135231.html

[7] ÇIVICI, Çağatay. Primefaces user guide 5.2. [online]. [cit. 2015-05-03]. Dostupné z: http://www.primefaces.org/docs/guide/primefaces_user_guide_5_2.pdf

[8] Maven - Introduction. [online]. [cit. 2015-05-03]. Dostupné z: https://maven.apache.org/what-is-maven.html

[9] Primefaces Showcase. [online]. [cit. 2015-05-03]. Dostupné z: http://www.primefaces.org/showcase/index.xhtml

[10] CHACON, Scott a Ben STRAUB. Pro GIT [online]. [cit. 2015-05-03]. Second edition. Dostupné z: https://progit2.s3.amazonaws.com/en/2015-05-02-7ddee/progit-en.478.pdf

[11] HTTPS [online]. [cit. 2015-07-14]. Dostupné z: https://cs.wikipedia.org/wiki/HTTPS

[12] SCHNEIDER, Robert D. MySQL: oficiální průvodce tvorbou, správou a laděním databází. 1. vyd. Praha: Grada, 2006, 372 s. ISBN 80-247-1516-3.

[13] Oracle GlassFish Server Online Documentation Library: GlassFish Server Application Development Guide [online]. 2015, 2 [cit. 2015-07-15]. Dostupné z: http://docs.oracle.com/cd/E26576_01/doc.312/e24930/jpa.htm#GSDVG00008

[14] JGit/User Guide [online]. [cit. 2015-07-17]. Dostupné z: http://wiki.eclipse.org/JGit/User_Guide

[15] JqPlot [online]. [cit. 2015-07-27]. Dostupné z: http://www.jqplot.com/

Zoznam príloh

Príloha A: Dátový slovník databázy navrhnutej pre aplikáciu

Príloha B: ER diagram databázy navrhnutej pre aplikáciu

Príloha C: Užívateľská príručka aplikácie

Súčasťou BP je CD.

Adresárová štruktúra priloženého CD:

\app

    \source

        \worklogica

         \config

    \deploy

    \database

\prílohy