VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMÚ

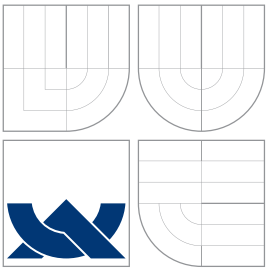
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYSTÉM MONITOROVÁNÍ STAVU PLÁNOVACÍCH ÚLOH

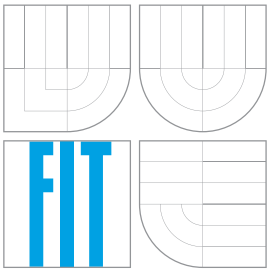
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE MARTIN MAGA
AUTHOR

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMÚ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYSTÉM MONITOROVÁNÍ STAVU PLÁNOVACÍCH ÚLOH

PLANNING TASK MONITORING SYSTEM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE MARTIN MAGA
AUTHOR

VEDOUCÍ PRÁCE ZDĚNEK LETKO SUPERVISOR.TITLE.P
SUPERVISOR

BRNO 2014

Abstrakt

Záverečná práca prezentuje Systém monitorovania stavu plánovacích úloh, ktorý umožňuje nájdené optimálneho riešenia pre NP-problém vzhľadom na dostupný čas a dostupné algoritmy. V práci analyzujeme technológie k tvorbe užívateľského rozhrania pre tento systém s využitím open-source technológií, rovnako analyzujem systém Optaplanner, ktorý vykonáva riešenie plánovacích problémov použitím rôznych konfiguračných súborov, ktoré definujú zadanie daného problém a použitie algoritmov. Vypracovali sme návrh, ktorý je užívateľský intuitívny a jednoduchý na pochopenie s pomerne strmou učiacou sa krivkou. Tento návrh sme predložili užívateľom, ktorý na základe vyplnenia dotazníka poskytli spätnú väzbu na overenie formálnosti a validity všetkých akcií. Zistili sme, že užívateľské rozhranie by mohlo obsahovať radu rozšírení, ktoré umožňia užívatelovi jednoduchšiu orientáciu v prostredí. Predpokladá sa použitie tohto projektu v rámci firemných požiadovok, rovnako aj pre komnunitné potreby, ktoré ho môžu ľubovoľne upravovať. Výsledok je riešenia problematiky Systému monitorovania stavu plánovacích úloh je užívateľské rozhranie je užívateľské rozhranie, ktoré intuitívne umožžnuje pracovať v rámci organizácie, rovnako aj sledovať stav a vytvárať nové úlohy. Rovnako je možné definovať vlastné úlohy a overiť si riešenia rôznych NP problémov, ktoré sú známe.

Abstract

Výtah (abstrakt) práce v anglickém jazyce.

Klíčová slova

Java EE 6, Java, Java Beans,Java Server Faces, Monitorovanie, Twitter, Bootstrap, Optaplanner, Webová služba, Enterprise Java Bean, JBoss, Rich Faces, Model, Komponenta, Maven, Arquillian, Plánovanie, MySQL, Užívateľ, Užívateľská rola, Obmedzenie, Plánovací problém, Úloha, Martin Večera, Zděnek Letko, Red Hat .

Keywords

Java EE 6, Java, Java Beans,Java Server Faces, Monitoring, Twitter, Bootstrap, Optaplanner, Web Services, Enterprise Java Bean, JBoss, Rich Faces, Model, Component, Maven, Arquillian, Planning, MySQL, User, User Role, Constraint, Planning problem, Martin Večera, Zděnek Letko, Red Hat .

Citace

Martin Maga: Systém monitorování stavu plánovacích úloh, bakalářská práce, Brno, FIT VUT v Brnì, 2014

Systém monitorování stavu plánovacích úloh

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatne pod vedením pana Zděnka Letka a Martina Večeřu

.....
Martin Maga
25. dubna 2014

Poděkování

Veľmi rád by som poďakoval za vedenie mojej bakalárskej práce pánovi Zděnkovi Letkovi a pánovi Martinovi Večeřovi, ktorý mi poskytli rady a podali pomocnú ruku vždy, keď som narazil na problém.

Obsah

Kapitola 1

Úvod

V úvode by som Vás rád krátko zoznámil s témou svojej bakalárskej práce, ktorá sa venuje téme Systému monitorovania stavu plánovacích úloh. Tento systém sa skladá z užívateľského rozhrania, ktoré je vytvorené prostredníctvom Java open-source technológií, ktoré bežia na javovskom serveri. Preto sa zameráme na všetky technológie, ktoré potrebujeme pre správne pochopenie a následnú implementáciu užívateľského rozhrania pre tento systém. Rovnako bližšie vysvetlím použitý open-source java server, ktorý je nevyhnutý pre beh aplikácie. Rovnako bude treba správne pochopiť celý plánovací open-source plánovací systém Optaplanner, pre ktoré je užívateľské rozhranie určené. Tento systém umožňuje spúšťať definované užívateľsky definované problémy, ktoré systém prostredníctvom správnych algoritmov naplánuje a dospeje k správnemu riešeniu vhľadom na dostupný čas a dostupné algoritmy. Rovnako sa budem venovať testovaniu a vyhodnoteniu užívateľského rozhrania z hľadiska intuitívnosti, jednoduchosti a splnenia všetkých formálnych požiadavok. Rovnako uvediem testy potrebné k overeniu správnej činnosti aplikácie a použitý framework. Toto téma bolo vybraté z dôvodu môjho osobného záujmu o open-source technológie,rovnako o možnosti ich využitia a veľmi ma zaujala možnosť verejná zdieľania projektu medzi open-source komunitov, ktorá mi môže poskytnúť spätnú vazu, resp. môže túto prácu využívať v praxi, čo cieľ, ktorý by som rád prostredníctvom tejto práce dosiahol.

Dopísať podľa vyhodnotnenie podľa testovania a dotazníka.

Kapitola 2

Java Enterprise edition 6

2.1 Motivácia

V posledných rokoch prevláda tendencia tvorby komplexných informačných systémov, ktoré spracovávajú veľké množstvo dát. Preto sa zvyšuje tlak na vývojárov na tvorbu prostriedkov, ktoré dokáže takéto systémy ľahko a rýchlo vytvárať. Jedným z takýchto prostriedkov je platforma Java Enterprise Edittion (Java EE), ktorá použijeme vo verzii 6, ktorá nám postačuje pre implementáciu aplikácie. Java EE je platformou, ktorá rozširuje základné možnosti jazyka Java o enterprise technológie, ktoré umožňujú tvorbu komplexnejší systémov, ktoré bežia na rôznych aplikačných serveroch. Jazyk Java je open source, rovnako ako aj všetky poskytnuté technológie, preto som sa rozhodol využívať tento programovací jazyk. Platforma Java EE je ďalej tvorená špecifikáciami pre podporu webových technológí, webových aplikácií, podnikovej logiky a .... Nám budú postačovať prvé 3 špefikácie tejto platformy, ktoré rozobereme v nasledujúcej časti spolu s technológiami, ktoré ich reprezentujú. Na základe Javy boli implementované boli implementované rôzne java EE kontajnery, ktoré sú potrebné pre správu a beh aplikácie. My sa zameriame na open-source riešenia z dôvodou šírenia projektu ako open-source. Ďalšou výhodou použitia tejto platformy je použitie anotácií, ktoré zjednodušujú implementáciu výslednej aplikácie a spôsobia konfiguráciu danej komponenty pri nasadzovaní a za behu. Rovnako je zdôraznení princíp POJO(Plain Old Java Objects)[3] a zjednodušenie tvorba balíkov. V poslednom rade musí spomenúť princíp „Convetion over configuration“, ktorý minimalizuje počet konfigurácií pre daný projekt. V nasledujúcej časti rozoberiem všetky potrebné špecifikácie doplnené o rôzne frameworky, bez ktorých by sa vývojový cyklus aplikácie nezaobišiel.

2.2 Špecifikácia platformy

Java EE predstavuje platformu určenú na vývoj webových a podnikových aplikácií[7]. Tieto aplikácie sú viacvrstvové z dôvodu lepšej prenositeľnosti, nasaditeľnosti a modifikovateľnosti. Frontend, predstavujúci užívateľské rozhranie a logiku na jeho ovládanie, pozostáva z webových frameworkov, stredná vrsta poskytuje bezpečnosť a transakcie.

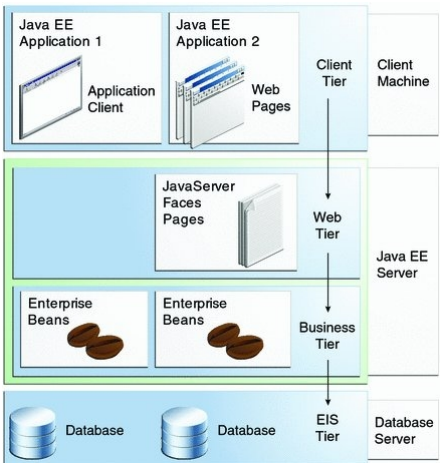
Najnižšia vrstva poskytuje pripojenie k databázam. Java EE je platformou, ktorá poskytuje širokú škálu aplikačných programových rozhraní(API), ktoré zjednodušujú, zkracujú a znižujú komplexnosť vývoja a nasadenia výslednej aplikácie. Jej vývoj neustále napreduje a je spravovaný Java Comunnity process(JCP). Aplikácie pre platformu Java EE sú vyvíjané prostredníctvom API, ktoré táto plaforma poskytuje. Medzi tieto API patrí napríklad: Java Server Faces, Java Persistence API, Enterprise Java Bean, ... Behovým prostredím sú aplikačné servery, ktoré pozostávajú zo servletov, JavaServer Pages, EnterpriseJavaBeans a iných technológií, ktoré sa starajú o správu aplikácie a jej nasadenie. Keďže je Java označovaná ako multiplatformovaná musí poskytovať prostriedky, ktoré je možné nasadiť naprieč rôznymi aplikačnými serverami. Medzi takýto prostriedok patrí bezpečnosť, ktorá je v riešená pomocou prístupových pravidiel, ktoré sú interpretované za behu aplikácie. V ďalších kapitolách si rozoberieme aplikaný model jazyka, ktoré je veľmi dôležitý pre pochopenie princípu činnosti aplikácií vyvinutých touto platformou. V ďalšej kapitole rozobereme aplikačný model platformy Java EE.

2.3 Aplikačný model

Java EE definuje aplikácie, ktoré sú viacvrstvové(multitier). Aplikačná logika je rozdelená medzi komponenty podľa ich funkcie[2]. Jednotlivé komponenty sa následne rôzne inštalujú na rôzne zariadenia v závislosti, do ktorého stupňa patria(Keďže každý stupeň môže byť fyzicky na inom aplikačnom serveri). Jednotlivé stupňe sa skladajú z rôznych komponent, pričom stupne sú rozdelené nasledovne:

- Klientský stupeň sa skladá z klientských komponenent, ktoré bežia na klientskom počítači
- Java EE server sa skladá z webových a podnikových komponent, ktoré bežia na Java EE serveri
- Databázový server ktorý sa skladá z enterprise information system komponent

Typicky beží medzi klientskom a databázou častou viac-vláknový Java EE server, ktorý býva označovaný skratkou EIS. Viacstupňovérozloženie môžete názorne vidieť na obrázku č. 2.1. Java EE aplikácia beží na klientskej stanici, býva obykle reprezentovaná tenkým klientom(webovým prehliadačom), nazývaným „thin client“(pretože sa nedotazuje priamo na databázový server), alebo hrubým klientom, do ktoré je čiastočne vložená logika aplikácia. Klient môže byť reprezentovaný ako webový alebo aplikačný. Typický webový klient pritom pozostáva z: Webové prehliadača, ktorý zobrazuje stránky a dynamických webových stránok pozostávajúceho z rôzneho značkovaciehojazyka(HTML,XHTML), ktoré sú generované webovými komponentami. Zložitá logika je vykonávaná strednou vrstvou, pričom klient len posiela požiadavky na Java EE server a ten prípadne sa dotazuje databázové servera a následne predáva výsledok. Klient môže poskytovať aj bohatšie užívateľské rozhranie, ktorá býva vytvárané technológiou Swing alebo Abstract Window Toolkit[10], po prípade sa vyskytuje aj prístup prostredníctvom príkazového riadku. V



Obrázek 2.1: Model Java EE [http://docs.oracle.com/javaee/6/tutorial/doc/]

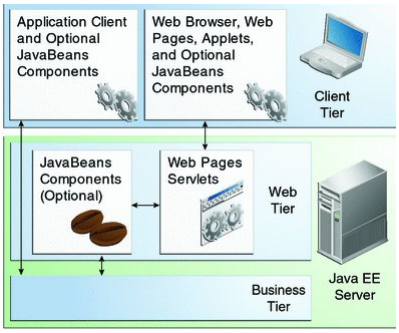
strednej časti obrázku sa nachádza Java EE server, na ktorom môžu bežať rôzne technológie v závislosti od požiadavky výslednej aplikácie a možností daného servera. Stredná vrstva sa ešte delí na webový stupeň, ktorý je prezentovaný technológiami JavaServer Faces a Pages. Druhá časť strednej vrstvy takzvaná podniková vrstva býva reprezentovaná technológiu EnterpriseJava Beans, ktoré vytvárajú logiku aplikácie. Java EE server môže by reprezentovaný, ešte okrem spomenutých technológií, rôznymi inými dostupnými technológiami, v závislosti od možnosti aplikačného servera, ktorý môže byť open-source (JBoss, Tomcat, GlassFish) alebo komerčný (IBM WebSphere,BEA WebLogic), ten obsahuje rôzne komponenty, ktoré so sebou rôzne komunikujú a interagujú na požiadavky klienta a na druhej strane komunikujú s databázovým systémom a starajú sa o beh aplikácie a jej nasadenie. Posledná časť predstavuje databázový server, ktorý obsahuju dáta, ktoré klient požaduje pri svojom požiadavku, tento server sa nazýva "EIS". Pre prístup k nemu sa používa buď nový prístup, ktorý sa nazýva objektovo-relačné mapovanie, ktoré využíva rozličné ovládače pre prístup k databázovému systému(napr. JBDC).

V nasledujúcej kapitole sa zameriame na technológie strednej vrstvy, ktoré sú nevyhnuté pre tvorbu a pochopenie činnosti navrhnutej aplikácie.

2.4 Webové komponenty

Java EE webové komponenty sú softwarové komponenty, ktoré spracovávajú prichádzajúci HTTP požiadok a poskytujú naň odpoveď. Všetky Java EE webové komponenty sú postavané na servletoch. ervlety sú javovské triedy , ktoré dynamicky spracovávajú požiadavky a tvoria odpovede.Súčasťou servletov alebo webových stránok, ktoré sú technológie JavaServer Faces technológiu(JSF) and JavaServer pages(JSP). Servlety podporujú automatickú správu sedenia, prostriedky pre vytváranie a ničenie servletov. Technológie JavaServer Faces a JavaServer Pages podporujú spracovanie užívateľských vstupov a ich

predanie a spracovanie podnikovou logikou. Pre implementáciu výslednej aplikáciu bola použitá JavaServer Faces technológia, ktorá poskytuje dostatočné možnosti pri tvorbe webových stránok. V rámci webových komponent spomeniem technológiu, ktorá je potrebná pre pochopenie funkčnosti aplikácie. Ide o technológiu Web Service.



Obrázek 2.2: Webové komponenty [http://docs.oracle.com/javaee/6/tutorial/doc/]

Na nasledujúcom obrázku č.2.2 je ukázaný princíp fungovania webových komponent. V hornej časti obrázku sa nachádza klientská vrstva, ktorá obsahuje buď len webový prehliadač po prípade Applety alebo JavaBean komponenty, ktoré čiastočne obsahujú logiku aplikácie. Na druhej strane môže byť klient reprezentoný aplikačným klientom, ktorý obsahuje obsahuje úplnú prezentačnú logiku aplikácie a teda v tom prípade, odpadá potreba spracovania vstupov po prípade nejaké generovania html stránky. Takýto klient komunikuje už len priamo s Java EE serverom, konkrétne podnikovým stupňom, ktorý implementuje zvyšnú logiku aplikácie a je reprezentovaný technológiou Enterprise Java Beans. V prípade, že máme k dispozícií tenkého klienta, klient komunikuje prostredníctvom webové prehliadača s HTML alebo XHTML stránky, ktoré sú vytvorené technológiou, ktoré spracovávajú požiadavok od klienta(vstupy) a následne komunikuje s podnikovým stupňom, ktorý obsahuje logiku reprezentovanú Enterprise Java Beans technológiou, ktorý následne môže komunikovať s databázovým serverom. Odpoveď je následne „predaná“ stránkám vytvorené prostredníctvom JavaServer Faces alebo JavaServer Pages technológiou a následne zobrazená užívatelovi v podobe výstupu na webovú stránku. V nasledujúcich dvoch podkapitolách sa bližšie pozreme na technológie JavaServer Faces a JavaServer Pages.

2.4.1 JavaServer Faces

JavaServer Faces(JSF) je framework pre tvorbu užívateľských rozhraní webových aplikácií. Tento framework beží na Java EE serveri. Tento framework poskytuje sa skladá z ďalšieho frameworku, ktorý obsahuje rôzne komponty pre zobrazenie informácií, užívateľských vstupov, spracovanie udalostí, navigáciu medzi stránkami. JSF vytvára aplikácie na základe MVC - Model-View Controller. Aplikácia, ktorá je vytvára týmto frameworkom pozostáva z webových stránok, grafických komponent, sadou komponent naviazané

Knižnica	URI	Zavedený prefix	Popis.
JSF Facelets Tag Library	http://java.sun.com/jsf/facelets	Tagy pro tvorbu šablon	
JSF HTML Tag Library	http://java.sun.com/jsf/html	Tagy komponent uživatelského rozhraní	
JSF Core Tag Library	http://java.sun.com/jsf/core	Tagy s funkcionalitou nezávislou na RenderKitu	
JSTL Core Tag Library	http://java.sun.com/jstl/core	Tagy jádra JSTL – cykly, podmínky atd.	
JSTL Functions Tag Library	http://java.sun.com/jstl/functions	Tagy JSTL pro funkce – např. toUpperCase atd.	

Tabulka 2.1: Tagy knižníc [http://docs.oracle.com/javaee/6/tutorial/doc]

na serverovú časť. Môže obsahovať rôzne desktriptory a konfiguračné súbory, ktoré nám pomáhujú pri nasadzovaní aplikácie. Základom JavaServer Faces je Facelets, čo je vlastne framework , ktorý slúži k tvorbe prezentačnej vrstvy webových aplikácií. Tento jazyk pomáha vytvárať JSF pohľady prostredníctvom HTML a buduje strom komponent. Facelets využíva XHTML pre vytváranie stránok, rovnako podporuje Facelets tag library, ktoré obsahujú obsahujú rôzne deklaracie komponenty, ktorých použitie je nevyhnuté pokiaľ chceme použiť nejakú komponenty z danej knižnice. Ttagy jednotlivých knižníc sú rozlišované na základe menných priestorov a to nasledovne:

Facelets stránky používajú XHTML 1 a CSS. Facelets od JSF 2.0 nahradzuje pôvodne používanú technológiu JavaServer Pages, ktorá sa pôvodne starala o tvorbu prezentačnej vrstvy aplikácií. Neoddeliteľnou súčasťou technológiu je Expression Language(EL), ktorý umožňuje dynamicky pristupovať k metódam javovských tried, rovnako dokáže získať a nastaviť hodnotu danej komponenty.Pri preklade sa vygenerejú z facelets a EL html komponenty, ktoré sú viazané na javovské triedy, ktoré sa nazývaju „Managed Bean“. Managed Bean sú javovský trieda, ktoré zabezpečujú predávanie údajov medzi rôznymi podnikovými komponentami(tvorenými EnterpriseJava Bean technológiou) a webovou stránkou. Takáto trieda dokáže za behu spracovávať údaje zadané na webovú stránku, rovnako dokáže obstarať validáciu vstupov, následne metódy a vlastnosti, ktoré sú volané alebo sú predávané údaje z vygenerovavnej stránky(HTML alebo XHTML) do managed bean-y alebo opačne. Aby bola v aplikácií „známa“ daná managed bean-a je potrebnú ju zaregistrovať v súbore faces-config.xml. Faces-config-xml je vlastne konfiguračný súbor, ktorý obsahuje zoznam managed bean a cestu k ním v rámci balíkovania, rovnako aj typ managed beany. Typ managed beany môže byť:

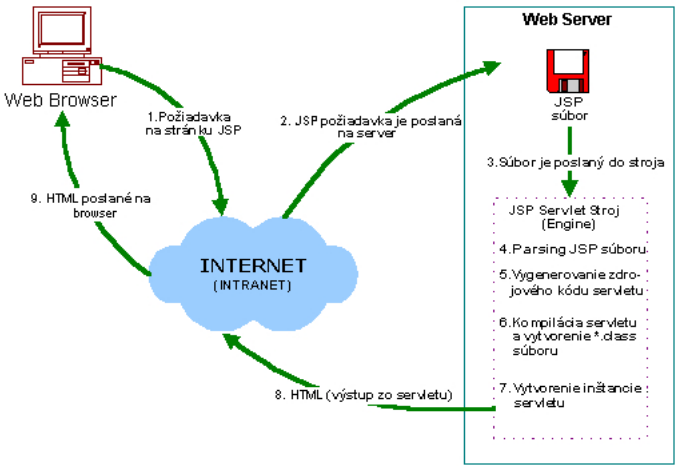
- @RequestScoped Managed beana pokiaľ prežíva HTTP požiadavok. Vytvára sa pri vytvorený požiadavku a zaniká pri zrušení HTTP požiadavku
- @NoneScoped Managed Beana existuje tak dlho ako existuje vyhodnotenie Facelets na stránky, po vyhodnotení zaniká
- @ViewScoped Managed beana prežíva pokiaľ existuje interakcia s danou JSF stránkou. Vytvára sa pri žiadosti o danú stránku a zaniká pokiaľ užívateľ prejde na inú JSF stránku

- @SessionScoped Managed bean prežíva tak dlho pokiaľ existuje HTTP sedenie. Vytvára sa pri 1.požiadavke o danú stránku a zaniká pri invalidácií daného HTTP sedenia
- @ApplicationScoped Managed Bean prežíva dokiaľ existuje aplikácia. Je vytvorená pri prvej interakcii s aplikáciou a zaniká pri ukončení aplikácie
- @CustomScoped Managed Bean existuje dokiaľ existuje záznam o bean-e v v custom Map, ktorá je vytvorená pre existenciu danej beany

Rovnako je možné v tomto konfiguračnom súbore nastaviť validačné triedy, reakcie na chyby a iné. Rovnako každá JSF aplikácia môže obsahovať konfiguračný súbor web.xml, ktorý je webový aplikačný deskriptov nasadenia. Tento súbor definuje všetky informácie, ko ktorých musí server vedieť, napr. použité servlety, inicializačné parametre, uvítacie stránky, ...

2.4.2 JavaServer Pages

JavaServer Pages technológia je jazyk, ktorý umožňuje priamo vkladanie Java kódu do HTML kódu. Pre vloženia ohraničenie java kódu v HTML stránke sa používajú nasledujúce značky: <% %> medzi, ktoré sa vloží príslušný java kód. Takéto časti v html stránky sa nazývaju skriptlety. Tieto skriplety sú dynamické, to znamená, že sú vykonávané za behu aplikácie. Výhodou tejto technológie, že je napísaná v Java a tento jazyk je komplexnejší a bezpečnejší. Rovnako pri žiadosť o JSP stránku(html stránka s príponou JSP, ktorá obsahuje nejaký skriptlet) je, že pri zmene sa nemení celý obsah stránky ale len jej časť, ktorá bola zmenená. Takže takéto JSP stránky sú dynamické a umožňujú zmenu obsahu za behu.



Obrázek 2.3: JSP architektúra [http://interval.cz/clanky/javaserver-pages-pro-vsechny/]

Na nasledujjúcom obrázku č.2.3 Užívateľ je reprezentovaný webový prehliadačom, ktorý zažiada o JSP stránku. Táto požiadavka je predaná Java EE serveru, ktorý zistí, že sa jedná o požiadavku na JPS stránku. Preto je požiadavok spracovaný JSP servlet strojom. Ten skontroluje početnosť požiadavok na danú stránku, pokiaľ ide o 1. tak stránku vygeneruje v opačnom prípade ju prekontroluje. Následne sa vygeneruje špeciálny servlet, ktorý ako základ použije JSP súbor. Kód tohto servletu je skompilovaný a je vytvorená jeho inštancia. Výstupotom zo servletu, ktorý vznikol ako požiadavka o JSP stránku je html stránka, ktorá je predaná užívateľovi, ktorý si ju zobrazí.

2.4.3 Web Service

Web Service sú klientské a serverové aplikácie, ktoré komunikujú prostredníctvom HTTP protokolu vymenieňaním XML správ. Tieto aplikácie poskystujú interoperabilitu medzi rôznymi platformami naprieč počítačou sieťou. Web Service umožňuje komunikáciu medzi rôznymi aplikáciami, ktoré bežia na rôznych platformách, napr. Java aplikácie založené na Windowse komunikujú s Net, aplikáciami bežiaci na Linuxe. Tento aspekt je aspekt je umožnený tým, že aplikácie komunikujú prostredníctov HTTP protokolu. Na konceptuálnej úrovni môžme web service chápať ako softwarové komponenty poskytujúje prístup ku koncovému bodu. Týmto koncovým bodov môžme rozumieť systém inej organizácie, od ktorej potrebuje získať nejaké dáta. Rovnako sa môže jednať aj o systém v rámci organizácie. Komunikácia prostredníctvom web service sa delí na 2 učastníkov. Prvý účastník produkovateľ(producer), ktorý vytvára požiadavok a spotrebiteľ(consumer), ktorý prijíma požiadavok. Komunikácia prebieha medzi týmto dvoma učastníkmi výmenov správ. Web service môže byť technicky implementovaný rôznymi možnosťami a prostredníctvom Big Web Service alebo Restful WebService.

"Big"Web Services

V Jave EE 6 existuje API, ktoré sa nazýva JAX-WS, ktoré umožňuje vytvorenie práve tohto typu web servici.[7] "Big"web service využíva XML správy, spolu so SOAP a XML jazykom, ktorý definuje architekrúru a formát správ. Tento typ Web Service obsahuje definíciu pre Web Service vo formáte WSDL, ktorý je čitatelný aj počítačom. Formát SOAP správ a definíciu jazyka WSDL rozhrania môže znížiť zložitosť vývoja aplikácií, webových služieb.

RESTful Web Service

V Java EE 6 existuje pre druhý typ web service API, ktoré sa nazýva JAX - RS. Tento typ web serrvice je vhodný pre základné , ad hoc integračné scenáre. REST webové služby, často lepšie integrované s HTTP ako službami založenými na SOAP je , nevyžadujú XML správ alebo definície WSDL služby - API.

2.5 Java Persistence API

Java Persistence API(JPA) je špecifikácia jazyku Java, ktorý poskytuje objektovo relačné mapovanie. Java Persistence API využíva pre mapovanie entity, ktoré reprezentujú dáta v databázi. Typicky teda reprezentuju tabuľku databáze a jej každá inštacia riadok tabuľky. Entitná trieda má teda atribúty, ktoré priamo odpovedajú názvu stĺpcov v databázovej schéme. To značne uľahčuje a zprehľadňuje prácu s databázou. Pre prístup k databáze sa používa trieda EntityManager, ktorá spracováva transakcie a zabezpečuje aby boli spĺňali podmienky ACID. Java Persistence API rovnako definuje vlastný jazyk Java Persistence Query Language, čo je jazyk podobný jazyku SQL, ktorý využíva trieda EntityManager pri svojej práci. Výhodou je, že tento jazyk je nezávislý na zvolenej databázovej technológií a má objektové vlastnosti, teda pri dotazoch nepoužívame konkrétne názvy tabuliek ale názvy entitný tried a jej vlastností.

Medzi konkrétne implementácie JPA patrí: Hibernate, Oracle TopLink, OpenJPA.

2.6 Enterprise Bean

Enterprise Bean(WB) sú Java EE komponenty, ktoré implementujú Enterprise JavaBeans(EJB) technológie. Enterprise bean beží v kontajneri EJB. EB je server-side komponentov, ktorá zapuzdruje enterprise logiku aplikácie. Obchodná logika je kód, ktorý spĺňa účel použitia. Z niekoľkých dôvodov, EB zjednodušuhe vývoj rozsiahlych, distribuovaných aplikácií. Po prvé, pretože kontajner EJB poskytuje služby na úrovni systému a vývojár sa môže sústrediť na riešenie obchodných problémov. Kontajner EJB, je zodpovedný za služby na úrovni systému, ako je riadenie transakcií a autorizácie zabezpečenia.

EB sa delia na 2 kategórie:

- Message-driven - Vykoná úloha pre klienta; voliteľne môže implementovať webové služby
- Session - Pôsobí ako poslucháča pre určitý typ správ, ako je API Java Message Service

2.7 Convetion over Configuration

Maven je založený na centrálnej konceptu životného cyklu zostavenie. Čo to znamená, že proces budovania a distribúciu určitého artefaktu(projektu) je jasne definovaná. Depency management je jedným z rysov Maven-u, ktorý je najlepšie známy pre užívateľa a je jednou z oblastí, kde Maven vyniká. Maven sa používa hlavne pre multimodulové aplikácie pre zachovanie vysokého stupňa kontroly a stability.

Životný cyklus pozostáva z fáz. Každá z týchto zostavenie životný cyklus je definovaný iným zoznamu zostavenie fáz, pričom fáza zostavenie predstavuje fázu životného cyklu. Maven môže generovať dokumentáciu projektu a vytvoriť stavbu. Maven má niekoľko správ, ktoré sa môžu pridať na webové stránky a zobraziť aktuálny stav projektu. Tieto správy majú formu pluginov, rovnako ako tie používané na zostavenie projektu.

Internacionalizácia v Maven je veľmi jednoduché.[5]

2.8 Testovanie

Arquillian je integračne a funkčne testovacia platforma, ktorá môže byť použitá pre testovanie middleware Java. Jej hlavným cieľom je urobiť integráčné(a funkčné) testy tak jednoduché písať unit testy, čo umožňuje vývojárom riadenie behu v rámci testu. Arquillian podporuje integráciu s Java EE kontajnermi, ako je JBoss AS a GlassFish a servlet kontajnerov, ako sú Tomcat a Jetty, a podporuje vykonávanie testov v cloudových službách. Podpora kontajnerov umožňuje vývojárom zamerať sa na rad technologických platforiem, vrátane Java EE 5 a 6, servlet prostredie, OSGI, Embedded EJB a samostatné CDI.[1]

Kapitola 3

JBoss Aplication Server

JBoss, čo je vlastne skratka pre JavaBeans Open Source Applicatom Server, v súčastní nazývaný WildFly je aplikačný server, ktorý je založený na platforme Java a Java Enterprise Edition.[6] Aplikačný server tvorí vrstvu medzi aplikáciami a operačným systémom, pričom rovnako poskytuje aplikácia často využívané funkcie, napr. spracovanie transakcií, výmená správ, ... Aplikačné servery podobne ako JBoss Aplication Server(JBoss AS) je open source. JBoss je Java EE platforma pre vývoj a nasadzovanie Java aplikáciu, webových aplikácií, služieb a portálov.

JBoss AS je napísaný v Jave, preto existuje možnosť ho používať naprieč rôznym platformám. Tento server slúži na vývoj a nasadzovanie podnikových aplikácií, webových aplikácií, služieb a portálov. Tento server je licensovaný pod GNU Lesser General Public License(GNU PL).

3.1 História JBoos-u

Všetko naštartoval v roku 1999 Marc Fleury. Pre podporu vývoja middleware InterBohemia sebe rozhodol implementovať jeden zo štandardov J2EE , EJB kontajner. Tým sa zrodil prvý projekt - EJBoss, ktorý sa neskôr premenoval na JBoss. TO niekoľko rokov neskôr sa stal prvým certifikovaným J2EE open source aplikačným serverom. Ďalej by som rád ukázal na vývoj rôznych verzií JBossu:[13]

- JBoss AS 4.0 , Java EE aplikačný server 1.4, je vybavený vloženým Apache Tomcat 5.5 servlet kontajnerom. JBoss môže bežať na mnohých operačných systémoch, vrátane mnohých POSIX platformách (ako GNU/Linux , FreeBSD a Mac OS X) , Microsoft Windows a ďalšie
- JBoss AS 5.1 , povolený v roku 2009, pracuje ako Java EE 5 aplikačný server. Je to menšia aktualizácia hlavnej verzie JBoss AS 5.0, ktorý bol vo vývoji po dobu najmenej troch rokov a bol postavený na vrchole novej JBoss microcontainer.
- JBoss AS 6.0, bol neoficiálne implementáciou Java EE 6, vydané 28. decembra 2010 .

- JBoss AS 7, bola vydané 12. júla 2011, len šesť mesiacov po poslednej hlavnej verzie , JBoss AS 6. JBoss AS 7 podporuje rovnaké špecifikáciu Java EE ako posledná verzia, a to Java EE 6. Java EE profil je iba čiastočne implementovaný v JBoss AS 7. Hlavné zmeny viditeľné pre užívateľa sú: oveľa menšiu veľkosť (menej než polovica z JBoss AS 6) a násobné zníženie v čase spustenia.

- JBoss AS 7. , aktuálna stabilná verzia bola vydaná vo februári 2012 . Zostávajúce časti EE špecifikácie boli realizované , a táto verzia bola certifikovaná pre EE plnom profile.

- WildFly 8 je priamym pokračovaním na JBoss AS projektu .

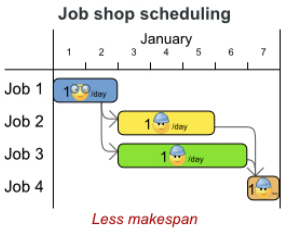
V ďalšej časti sa zameriame na OptaPlanner, čo je vlastne systém, pre ktorý je rozhranie navrhované.

Kapitola 4

OptaPlanner

OptaPlanner je odľahčený open source software a ďalšie prokračovanie frameworku JBoss Drools, ktorý optimalizuje plánovacie problémy.[11] Tieto plánovacie problémy môžu byť nasledujúceho charakteru:

- Plánovanie agendy: plánovanie schôdzok, vymenovanie, práca údržby
- Plánovanie vzdelávania: plánovanie lekcie, kurzov



Obrázek 4.1: Job, Shop scheluding, prevzaté z http://www.optaplanner.org/

Obrázok č. 4.1 zobrazuje typické použitie OptaPlanner-u. Môžme vidieť v nasledujúcom obrázku vystupú 4 osoby, ktoré vykonávajú nejakú činnosť. Ich činnosť je špecifická a silne závisí od práce predchádzajúcich. Optaplanner sa snaží ich činnosti maximálne optimalizovať a jednotlivé činnosti zvoliť v následnosti tak, aby výsledná práca bola spravená za najkratší možný čas vzhľadom na činnosť, ktorá sa optimalizuje.

4.0.1 NP-úplný problém

Každý plánovací problém je NP-úplný problém.[8] NP-úplné problémy sú nedeterministicky polynomiálne problémny, ktoré nie sú riešitelné v dostupnom čase, pretože sa nepodarilo nájsť deterministický algoritmus. Príkladom NP úloh môžme považovať: problém obchodné cestujúceho, ...

Riešenia poskytnutým týmto frameworkom, ktorý využíva pokročilé optimalizačné algoritmy, sú dosiahnuteľné v reálnom čase. Dosiahnutie v reálnom čase znamená nájdenie 1 alebo viacerých riešení, alebo nenájdenie žiadneho riešenia vzhľadom na poskytnutý čas a optimalizačné algoritmy, ktoré sú implementované.

Každý plánovací problém je definovaný na základe obmedzení, ktoré musia minimálne spĺnať: [4]

- Negatívne "hard"obmedzenie, ktoré nesmie byť porušené
- Negatívne "soft"obmedzenie, ktoré by nemali byť porušené pokiaľ sa dá tomu vyhnúť.

Niektoré problémy môžu obsahovať aj pozitívne podmienky alebo odmeny, ktoré by mali byť splnené pokiaľ je možné ich splniť.

Tieto podmienky definujú skóre plánovacieho problému. Tieto podmienky môžu byť zapísané v Jave alebo v Drools pravidlách, ktoré značne zjednodušujú kód.

Vytvorenie je pomocou pravidiel môže robiť oveľa jednoduchšie spájať mnoho pravidiel s mnohými akciami. Tieto pravidlá bývajú typicky definované pomocou XML súboru.

OptaPlanner pomáha programátori riešiť obmedzenie problémov spokojnosti efektívne. Pod kapotou sa kombinuje optimalizačné heuristiky na výpočet skóre.

4.0.2 Výsledky plánovacieho problému

Tieto obmedzenia definujú výpočetné skóre problému plánovania. Každé riešenie problému plánovanie môže byť odstupňovaná so skóre.

Plánovanie problému má niekoľko riešení. Existuje niekoľko kategórií riešení:

- Možným riešením je nejaké riešenie, či je alebo nie je ľubovoľný počet obmedzení. Problémy plánovanie mávajú neuveriteľne veľké množstvo možných riešení. Mnoho z týchto riešení sú bezcenné.
- Uskutočniteľným riešením je riešenie, ktoré neporušuje žiadne (negatívne) tvrdé obmedzenia. Niekedy nie sú realizovateľné riešenie. Každý uskutočniteľné riešenie je možné riešenie.
- Optimálnym riešením je riešenie s najvyšším počtom bodov. Problémy plánovanie mávajú jedno alebo niekoľko optimálnych riešení. K dispozícii je vždy aspoň 1 optimálnym riešením, a to aj v prípade , že neexistujú žiadne uskutočniteľné riešenie, a optimálne riešenie nie je možné .
- Najlepším riešením je nájsť riešenie s najvyšším skóre zistené implementáciou v danom čase.

OptaPlanner podporuje niekoľko optimalizačných algoritmov ako efektívne prehrýzť týmto neuveriteľne veľkým množstvom možných riešení. V závislosti na prípade použitia, niektoré optimalizačné algoritmy dosahujú lepšie výsledky ako ostatné, ale to je nemožné povedať dopredu. Pri plánovaní , je ľahké prepnúť algoritmus optimalizácie, zmenou konfigurácie Solver na niekoľkých riadkov XML alebo kódu.

4.0.3 Ukážka XML configuračného súboru

V nasledujúcom obrázku by som rád ukázal príklad XML configuračného súboru pre OptaPlanner.

Listing 4.1: Test

```
<?xml version="1.0" encoding="UTF-8"?>
<solver>
  <!--<environmentMode>FAST_ASSERT</environmentMode>-->
  <!-- Domain model configuration -->
  <solutionClass>org.optaplanner.examples.cloudbalancing.domain.
      CloudBalance</solutionClass>
  <planningEntityClass>org.optaplanner.examples.cloudbalancing.domain.
      CloudProcess</planningEntityClass>
  <!-- Score configuration -->
  <scoreDirectorFactory>
    <scoreDefinitionType>HARD_SOFT</scoreDefinitionType>
    <simpleScoreCalculatorClass>org.optaplanner.examples.cloudbalancing
        .solver.score.CloudBalancingSimpleScoreCalculator</
        simpleScoreCalculatorClass>
    <!--<scoreDrl>/org/optaplanner/examples/cloudbalancing/solver/
        cloudBalancingScoreRules.drl</scoreDrl>-->
  </scoreDirectorFactory>
  <!-- Optimization algorithms configuration -->
  <termination>
    <maximumSecondsSpend>120</maximumSecondsSpend>
  </termination>
  <constructionHeuristic>
    <constructionHeuristicType>FIRST_FIT_DECREASING</
        constructionHeuristicType>
    <!--forager-->
      <pickEarlyType>FIRST_NON_DETERIORATING_SCORE</pickEarlyType>
    <!--/forager-->
  </constructionHeuristic>
  <localSearch>
    <acceptor>
      <entityTabuSize>7</entityTabuSize>
    </acceptor>
    <forager>
      <acceptedCountLimit>1000</acceptedCountLimit>
    </forager>
  </localSearch>
</solver>
```

Konfigurácie solveru pozostáva z 3 častí:

- Domail model configuration: Musíme Optaplanneru uviesť hlávnú triedu.
- Configurácia skóre, ktorá hovorí Optaplanneru ako ma optimalizovať premenné. Pokiaľ používame hard a soft obmedzenia, použijeme "HardSoftScore". Musíme tiež uviesť ako vypočítať také skóre, v závislosti na našich požiadavkách. Ďalej sa, sme musíme pozrieť do 2 alternatívy pre výpočet skóre: pomocou jednoduchej implementácie Java, alebo pomocou Drols DRL.Optaplanner bude hľadať riešenie s najvyšším skóre. Budeme používať HardSoftScore, čo znamená, plánovač bude hľadať riešenie s žiadnymi tvrdými obmedzeniami členenie (spĺňajú požiadavky na hardvér) a pokiaľ možno čo najmenej mäkkých obmedzenia členenie (minimalizovať náklady na údržbu).
- Konfigurácia optimalizačných algoritmov

4.0.4 Optimalizačné algoritmy

V nasledujúcej časti textu si ukážeme optimalizačné algoritmy, ktoré používa optaplanner.

- First FIT - To je veľmi jednoduché greedy algoritmus aproximácie. Algoritmus spracováva položky v ľubovoľnom poradí. Pre každú položku, pokúsi sa umiestniť na položku v prvej priehradke, ktorý sa môže ubytovať položku. Ak nie je nájdený žiadna priehradka, otvára novú priehradku a kladie položku v rámci nového zásobníka.
- Firt FIT Decreasing
- Firt FIT Decreasing + heurestic local search
  - Hill Climbing - hill climbing je matematická optimalizácia technika, ktorá patrí do rodiny miestneho vyhľadávania. Jedná sa o iteratívny algoritmus, ktorý začína s ľubovoľným riešenie problému, potom sa pokúsi nájsť lepšie riešenie tým, že postupne mení jeden prvok riešenia. Ak zmena vytvára lepšie riešenie zmeny sa opakujú až žiadne ďalšie zlepšenie nie je možno nájsť.[9]
  - Tabu Search - Tabu search používa miestny alebo susedcký postup vyhľadávanie, tak že iteratívne presuvá z jedného možného riešenia x k lepšiemu riešeniu x v susedstve x, kým sa niektoré kritérium zastavenia splní. Miestne postupy vyhľadávania sa často uviaznu v zle bodovaných oblastiach. V snahe vyhnúť sa týmto nástrahám a preskúmať oblasti hľadaného miesta, ktoré by mali zostať bez prehliadky inými miestnymi postupmi vyhľadávania, tabu search starostlivo skúma okolí každého toku ako hľadanie postupuje. Riešenie prijatí do novej štvrti sú určené pomocou pamäťových štruktúr.

– Simulated Annealing - Simulované žíhanie je všeobecný algoritmus pre globálne optimalizačné problémy lokalizovať dobré priblíženie k globálnej optimálnej danej funkcie vo veľkom vyhľadávacieho priestoru. To sa často používa pri hľadaní priestorov diskrétnych (napr. všetky výlety, ktoré navštevujú danú množinu miest). U niektorých problémov, môže simulované žíhanie byť účinnejšie ako vyčerpávajúci zoznam - za predpokladu, že cieľom je iba nájsť prijateľne dobré riešenie v stanovenú dobu, skôr ako tým najlepším možným riešením .

V ďalšej kapitole by som rád uviedol problematiku užívateľského rozhrania.

Kapitola 5

Grafické užívateľské rozhranie

V tejto kapitole sa zameráme na problematiku užívateľského rozhrania, ktoré vlastne je reprezentované prostredíctvom technológie Java Server Faces v kombinácií frameworkov Rich Faces a Twitter Bootstrap-u. Toto rozhranie bude umožňovať nahrávať pravidlá, zobrazovať výsledky, spúšťať, pozastovať a zobrazovať detaily úloh. Keďže táto výsledná aplikácia by mala byť použíteľná aj na mobilnom telefóne bol vybratý štýlovací framework Twitter Bootstrap, ktorý značne uľahčenie tvorbu takéhoto rozhrania.

5.1 Twitter Bootstrap

Twitter Bootstrapje veľmi jednoduchý a voľne dostupný súbor nástrojov pre vytváranie moderného webu a webových aplikácií.[12] Ponúka podporu najrôznejších webových technológií HTML , CSS , JavaScript a mnoho prvkov , ktoré je možné ľahko implementovať do svojej stránky. Pre použitie Twitter Bootstrap sú nutné základné znalosti HTML a CSS. Interaktívne prvky ako sú tlačidlá, boxy , menu a ďalšie kompletne nastavené a graficky spracované elementy je možné vložiť iba pomocou HTML a CSS .

Výhodou tohto súboru nástrojov je jednoduché spracovanie akéhokoľvek používateľského rozhrania vo webovej aplikácii a nerozhoduje , či to je napríklad používateľské rozhranie v administrácii back-endových alebo front-endových aplikácií.

Podrobné vysvetlenie jednotlivých komponent nájdete na nasledujúcej adrese http://getbootstrap.com/, rovnako aj s príkladmi použitia. V nasledujúcej časti prejdem na samotný návrh užívateľského rozhrania.

5.2 Rich Faces

Rich faces predstavuje open-source Ajax knižnicu, ktorá predstavuje rozšírenie pre JavaServer Faces. Umožňuje integráciu schopností ajaxu do enterprise aplikácií. RichFaces obohacuje framework Ajax4jsf v dvoch dôležitých ohľadoch. Po prvé, sa rozširuje množstvo vizuálnych pripravených komponent. Po druhé, plne implementuje funkciu skinnability rámca Ajax4jsf vrátane veľkého množstva preddefinovaných vzhľadov. Pomocou skinnability, že je oveľa ľahšie riadiť vzhľad aplikácie.

5.3 Rozbor aplikácie

Výsledná aplikácia bude zobrazovať priebežné výsledky výpočtu frameworku optaplanner. Tento framework bude pre túto prácu optimalizovaný pre úlohu N Dám, ktorú bude schopný riešiť. Bude sa deliť na aplikáciu, ktorá predstavuje užívateľské rozhranie vytvorené prostredníctvom technológie Java Server Faces v kombinácií s Rich Faces nastylované prostredníctvom frameworku Twitter Boostrap, ktoré zároveň zabezpečuje prenositeľnosť rozhrania na mobilné telefóny. Užívateľské umožňuje zobrazovanie a spravovanie úloh, organizácií, do ktorých náležia jednotliví užívatelia a užívateľov. Každá časť systému bude sprístupnená podľa príslušnosti užívatela k užívateľskej role. Z tohto rozhrania bude môcť užívateľ pokiaľ mu to užívateľská rola dovoľuej pridať úlohu, ktorú môžme následne spustiť alebo pozastaviť. Spustenie prebehia zavolaním služby web service, ktorá obsahuje potrebné prostriedky na spustenie úlohy. Web service následne zavolá enterprise bean, v ktorej prebieha spracovanie danej úlohy. Výsledok výpočtu sa priebežne ukladá do databáze. Výsledku sa následne priebežne zobrazuje v užívateľskom rozhraní. Užívatelia majú prístup k obmedzenému počtu úloh, zároveň môžu vykonávať obmedzené akcie a to nasledovne:

- Administrátor - má prístup ku všetkým úlohám v systéme, úlohy môže editovať vytvárať, mazať, publikovať a odpublikovať , môže vytvárať, mazať a editovať užívateľov, rovnaké môžnosti má aj s organizáciami
- Plánovač - má prístup k úlohám v rámci svojej organizácie, môže vytvárať, editovať, mazať úlohy, publikovať a odpublikovať
- Čitateľ - úlohy môže len zobrať v rámci svojej organizácie, publikovať , odpublikovať

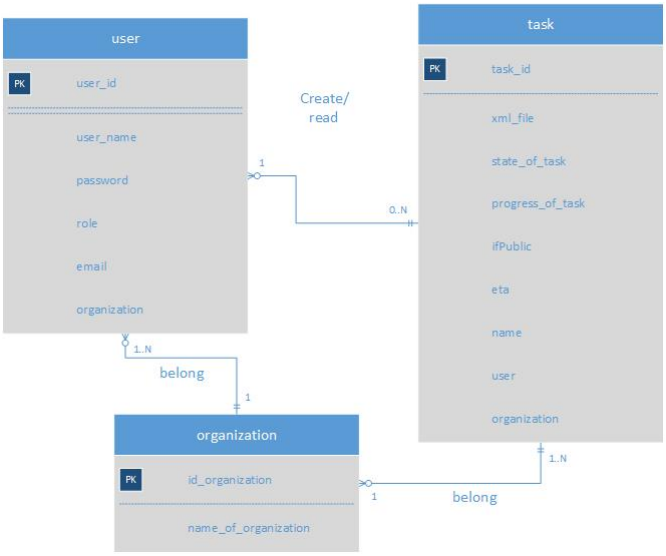
Výsledná aplikácia bude nasadená na aplikačný server JBoss.

5.3.1 Databázová technológia

Aplikácia potrebuje pre spracovanie úloh, organizácií a užívateľov databázu. Pre potreby bakalárskej práce bol vybratý relačný open-source databázový model MySQL. MySQL databázová technológia je veľmi vhodná pre malé a stredne veľke aplikácie, čo tá naša je, rovnako poskytuje dobrý výkon pri vykonávaní transakcií, umožňuje vytvárať procedúry, databázové triggere a jej inštalácia je pomerne jednoduchá a nezaberá veľa diskové priestoru, rovnako je MySQL multiplaformová, keďže je možné ju nasadiť na systémy s operačným systémov windows, linux, max os. Medzi nevýhody tejto technológie patrí neefektívna práca s databázovými transakciami, neefektívne ukladanie veľkého množstva dát.

5.3.2 Návrh modelu databáze

Na nasledujúcom obrázku je ukázaný ER diagram, ktorý bol použitý pre dtabázu:



Obrázek 5.1: ER diagram

Tento obrázok zobrazuje jednotlivé entity, ktoré sú potrebné na uloženie v databáze, každá z nich ma určité položky. ER diagrama sa skladá z 3 entít: user - entita, ktorá reprezentuje užívateľ, task - entita, ktorá reprezentuje úlohu a organization - entita, ktorá reprezentuje organizáciu. Výsledný návrh odpovedá skutočnosti, že každý užívateľ musí byť súčašťou organizácia, rovnako môže mať vytvorené 0 až N úloh. Taktiež pre zjednodušenie je každa úloha priradená priamo organizácií pre zlepšenie rýchlosti získania výsledkou a zjednodušenia ich nájdenia. Každá entita obsahuje primárny kľúč(jedná sa o silné entitné množiny), ktorý je odvodený od názvu a začína predponou „id_“ a pokračuje názvom entity. Poďme sa pozrieť bližšie na jednotlivé entity. Entitná množina organization obsahuje 2 položky jednou z nich je primárny klúč a ďalšou názov organizácia podľa, ktorej sú zaraďovaný jednotlivý užívatelia. Ďalej prejdime k entitnej množine user. Táto entita má rovnako primárny kľúč. Ďalej obsahuje položku pre užívateľské meno(user_name), heslo(password), email, užívateľskú rolu(role) a cudzí kľúčč organization, ktorý obsahuje na organizáciu. Nakoniec prejdime k entitnej množine task. Táto entitná množina obsahuje primárny kľúč, ďalej obsahuje xml súbor, ktorý reprezentuje danú úlohu(v našom prípade N dám), stav úloh(state_of_task, ktorý reprezentuje rôzne stavy úlohy), ktorý si podrobnejšie rozobereme. Úloha sa môže nachádzať v jednom z nasledujúcich stavov:

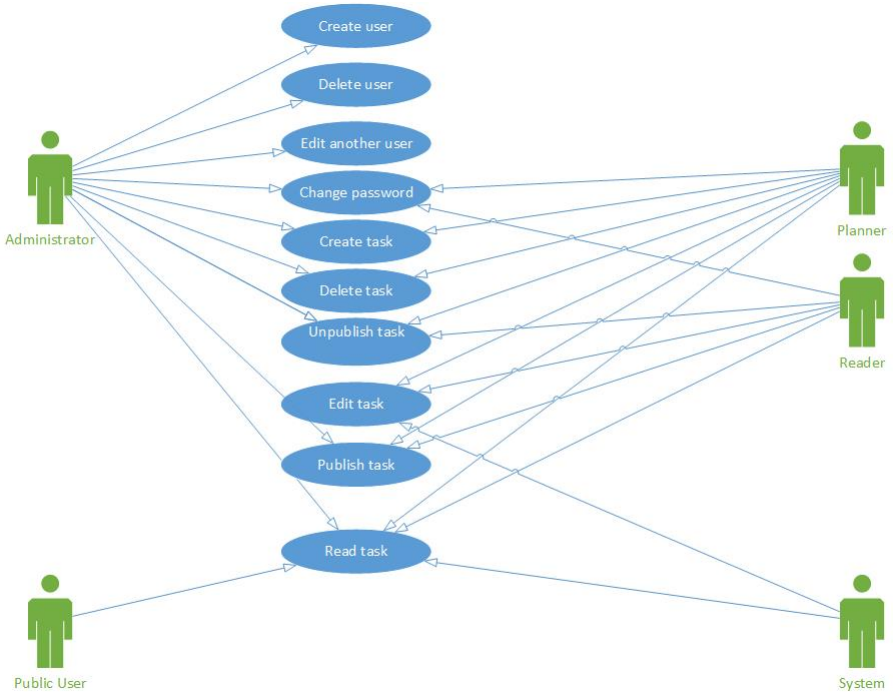
- NEW - úloha bola vytvorená
- MODIFIED - xml súbor bol modifikovaný
- WAITING - úloha čaká na spracovanie
- IN_PROGRESS - práve prebieha výpočet

- PAUSED - úloha je pozastavená
- COMPLETE - úloha je dokončená

Entitná množina task ďalej obsahuje položku, ktorá percentuálne hodnotí stav výpočtu úlohy(progress_of_task), čas do skončenia výpočtu úlohy(eta), nastavenie úlohy na privátnu alebo verejnú(ifPublic), názov úlohy(name) a cudzie kľúce user, ktorý odkazuje na užívateľa, ktorým bola úloha vytvorená a organization, ktorá odkazuje na organizáciu užívateľa, ktorým bola vytvorená. V ďalšej kapitole sa pozrieme na use case diagram.

5.3.3 Diagram užitia

Nasledujúci obrázok ukazuje príprady užitia systému: Na nasledujúcom obrázku č.5.2



Obrázek 5.2: UseCase diagram

môžme identifikovať 4 užívateľov systém. Public user je verejný úžívateľ, ktorý môže čítať úlohy, ktoré sú nastavené ako verejné(public). Systém môže načítať úlohy z databáze pre potreby výpočtu, z ktorými následne pracuje, a potom výsledky ukladá do databáze, teda edituje úlohy(tasky). Čitateľ(Reader) môže úlohy čítať, publikovať a odpublikovať. Plánovač môže úlohy čítať, vytvárať v databáze, mazať, editovať už vytvorené úlohy v rámci svojej organizácie,publikovať a odpublikovať úlohy a meniť vlastné heslo. Administrátor môže vytvárať úlohy, mazať úlohy , editovať úlohy, publikovať a odpublikovať úlohy. Rovnako si môže meniť heslo, vytvárať,mazať editovať organizácie a užívateľov.

5.4 Návrh rozhrania

Výsledné rozhranie kladie dôraz na jednoduchosť a prehľadnosť zobrazených úloh. Z tôhto dôvodu boli implementované mechanizmy vyhľadávanie úloh, organizácií a užívateľov. Rovnako možnosti lexikografického triedenia. Po prihlásení do systému Jednotlivé môžnosti sú následe zakompotované do záložiek, v ktorých je sprístupné príslušná funkčnosť. Výsledné rozhranie je prenositeľné aj na mobilné zaradenie, čo je spôsobené použitím frameworku Twitter Boostrap.

5.5 Implementácie

Aplikácie bola rozložená do viacerých tried podľa zodpovednosti daných komponent. Pri implementácií bolo použité JBoss Developer studo 7.1.1 GA spolu s JBoss AS 7.1.1 Final. Celý projekt boli založený na technológií maven, ktorú uľahčovala celý process vývoja a jeho následne nasadenie na JBoss server. Pre databázovú technológiu bol nainštalovaný mysql server nakonfigurovaný s príslunými prihlasovacími údajmi. Celý vývoj začal tvorbou balíka pre entitné triedy, ktoré pristupovali k databázami. Vývoj ďalej pokračoval tvorbou triedy, ktorá implementuje všetky potrebné operácie nad dátami. Následne treba správne nakonfigurovať datasource v JBoss AS, aby mohol správne pristupovať k databázmi vrátane prihlasovacích údajov. Následne pre správne priprájanie nasadeného maven projektu trebalo nastaviť súbor persistence.xml, do ktorého sa definovali entitné triedy a odkaz na datasource umiestneného na JBoss AS. Následne som sa sústredil na vývoj samotného užívatelského rozhrania. Užívateľské rozhranie som rozdelil do do 4 Managed bean, ktoré sa starajú o funkčnosť implementovaných xhtml stránok. Stránky sú celkom 4 , 3 pre užívatelské role, 4. pre prihlasovanie do systému. Každej stránke pritom odpovedá managed beana. Prihlasovacia stránka je upravená prostredníctvom Twitter Bootstrapu, pričom obsahuje zabezpečenie prostredníctvom uživatelských rol. Rovnako obsahuje validáciu pred nekorektným prihlasovaním, ktoré môže byť spôsobené neznámym užívateľským meno alebo heslom. Následne je užívateľ presmerovaný na základe svojej role na príslušnú stránka. Každá stránka obsahuje záložky a podla svojej role dovodluje užívatelia vykonávať akcie. Základom každého zobrazenie je dátová tabulka h:datatable, ktorá zodpovedá príslušnému modelu(užívateľ, úloha, organizácia s príslušnými vlastnosťami), ktorá zobrazuje údaje na stránku a neustále obnovuje svoj obsah podľa obsahu v databázi. Príslušné riadky odpovedajú položkám v databázi pričom pre každú položku sa vzťahuje nejaká akcia, ktorá je reprezentovaná prislušným tlačidlom. Po stlačení tlačidla sa prislušna zmena prejaví na stránke rovnako ako aj na stránke. Vytváranie novej úlohy je reprezentované komponentov rich:upload, ktorá pochádza z frameworku Rich Faces. Táto komponenta zabezpečuje nahrávanie xml súborov, ktoré reprezentujú danú úlohu, celé je to zaobalené do formuláru, ktorý obsahuje položku na zadanie mena. Po stlačení tlačidla „Create Task“, dôjde k zavolaniu methódy z managed beany príslušnej triedy, ktorá sa postará o vytvorenie úlohy v databázi a jej zobrazenie na stránke. V každú tabulku je možné zoraďovať podľa stĺpcov, deje sa to kliknutím na príslušný stĺpec, to spôsobí zavolanie metódy v managed bean-e, ktorá prejde všetky položky daného

stĺpca, ktoré sú uložené v zozname a zoradí ich podľa stĺpcov. Obnovanie obsahuje sa realizuje prostredníctvom komponenty aj4:poll, ktorá je súčasťou frameworku Rich Faces. Táto komponenta neustála volá metódu, ktorá získava všetky položky z databáze, čím je zabezpečený aktuálny obsah tabuliek. Vyhľadávanie položiek v danej tabulke sa deje prostredníctvom formulára, do ktorého sa zadá vyhľadávaný reťazec a zvolí sa z vyskakovacie zoznamu, následna sa zavolá metóda, ktorá nastaví obsah h:datable na nájdené položky. Tento postup je rovnaký pre manažovanie organizácií a užívateľov. Jednotlivé možnosti sú zakomponované od záložiek, ktoré sú implementované prostredníctvom twitter boostrap-u. Jednotlivé metódy sú pritom identické vo všetkých managed bean podľa užívatelskej role, až na to, že sú obmedzené podľa užívatelské role. Spustenie výpočtu sa deje prostredníctvom tlačidla „Run Task“, to zavolé metódu „Run Task“ web service prostredníctvom http protokolu a predá mu ako parameter ID úlohy, ktorú má spustiť. Tá následne získa data z databáze a zaradí úlohu do fronty. Z fronty si odoberajú enterprise beany, ktoré sa môžu nachádzať kvôli rozprestrenie záťaže na viac serveroch. Tie si úlohu vyberú z actiemsq fronty a spustia výpočet. Priebežne pritom ukladajú údaje o stavu úlohy, času do ukončenia úlohy a percentuálnom ohodnotení úlohy. Po ukončení úlohy je uložené najlepšie možné riešenie do databáze. Užívateľ môže úlohy pozastaviť a to po stlačení tlačidla „Stop Task“, ktoré zavolá metódu web service prostredníctvom protokolu http. Následne je možné úlohu opäť spustiť. Výsledky spracovania sú priebežne zobrazované(každé 2 sekundy) na užívateľské rozhranie. Rozhranie obsahuje validáciu, ktorá kontroluje všetky užívateľské vstupy.

5.6 Testovanie

Testovanie prebiehalo na servery JBoss AS 7.1.1 Final najprv prostredníctvom jednodúch JUnit testov, ktoré malo overiť komplikovanú fukčnosť metód. Následne sa pre overenie fukčnosti databáze použil framework Arqullian, ktorý umožňuje nasadenie tried priamo do Java EE kontajneru, čo zjednodušuje testovanie. Prostredníctvom tohto frameworku sa testovala celková fukčnosť aplikácie.

V ďalšej častie prebiehalo testovanie medzi konkrétnymi užívateľmi. Užívatelia testovali aplikáciu a hľadali buggy, ktoré neodhalilo predošlé testovanie.

5.7 Vyhodnotenie aplikácie

Po testovacej fáze nasledovala fáza vyhodnotenia aplikácie. Cielovým užívatelom bol poskytnutý prístup k aplikácií. Následne po vyskúšaní aplikácie vyplnili dotáznik, ktorá poskytoval celkový pohľad na aplikáciu.

Kapitola 6

Záver

Literatura

[1] Ament, J. D.: *Arquillian Testing Guide*. Packt Publishing, 2013, iSBN 978-1782160700.

[2] Arun Gupta: *Java EE 6 Pocet Guide*. O'REILLY, 2012, iSBN 978-1-449-33668-4.

[3] Ashmore, D. C.: *The Java EE Architect's Handbook, Second Edition: How to be a successful application architect for Java EE applications*. Packt Publishing, 2014, iSBN 978-0972954884.

[4] Bali, M.: *Drools JBoss Rules 5.X Developer's Guide*. Packt Publishing, 2013, iSBN 978-1782161264.

[5] Company, S.: *Maven: The Definitive Guide*. O'Reilly Media, 2008, iSBN 978-0596517335.

[6] Davis, T. M. S.: *JBoss at Work: A Practical Guide*. O'Reilly Media, 2006, iSBN 978-596-00734-8.

[7] Jendrock, E.; Cervera-Navarro, R.; Evans, I.; aj.: The Java EE 6 Tutorial [online]. `http://docs.oracle.com/javaee/6/tutorial/doc/`, 2013-11-03 [cit. 2013-10-25].

[8] Johnson, M. R. G. D. S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. Macmillan Higher Education, 1979, iSBN 978-0716710455.

[9] Michiels, J. K. E. A. W.: *Theoretical Aspects of Local Search*. Springer, 2007, iSBN 978-3540358534.

[10] Piroumian, V.: *Java GUI Development (Sams Professional)*. Sams, 1999, iSBN 978-0672315466.

[11] Thilo, M. O. J. T. C. R. J.: Twitter Bootstrap [online]. `hhttp://getbootstrap.com/`, 2013-12-16 [cit. 2013-12-27].

[12] Thilo, M. O. J. T. C. R. J.: Twitter Bootstrap [online]. `hhttp://getbootstrap.com/`, 2013-12-16 [cit. 2013-12-27].

[13] WWW stránky: JBoss.
`https://docs.jboss.org/author/display/AS71/Documentation`, 2013-12-16 [cit. 2013-12-27].