

# VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ  
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY  
DEPARTMENT OF INTELLIGENT SYSTEMS

## SYSTÉM MONITOROVÁNÍ STAVU PLÁNOVACÍCH ÚLOH

BAKALÁŘSKÁ PRÁCE

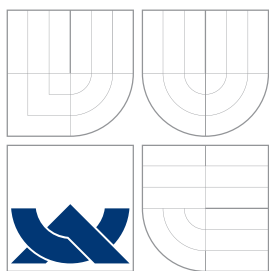
BACHELOR'S THESIS

AUTOR PRÁCE

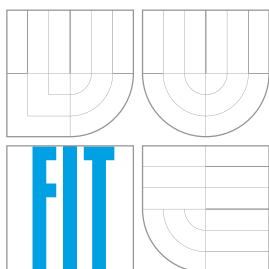
AUTHOR

MARTIN MAGA

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ  
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ  
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY  
DEPARTMENT OF INTELLIGENT SYSTEMS

# SYSTÉM MONITOROVÁNÍ STAVU PLÁNOVACÍCH ÚLOH

PLANNING TASK MONITORING SYSTEM

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE

AUTHOR

MARTIN MAGA

VEDOUCÍ PRÁCE

SUPERVISOR

ZDĚNEK LETKO, Ph.D.

BRNO 2014

## Abstrakt

Závěreční práce prezentuje Systém monitorování stavu plánovacích úloh, který umožňuje nalezení optimálního řešení pro rozličné plánovací problémy, přičemž některé mohou mít charakter NP-problém, přičemž řešení může být nalezeno vzhledem na dostupný čas a dostupné algoritmy. V práci analyzujeme technologie k tvorbě uživatelského rozhraní pro tento systém s využitím open-source technologií, rovněž analyzujeme systém Optaplanner. Vypracovali jsme návrh, který je intuitivní a jednoduchý na pochopení s poměrně strmou učicí se křivkou. Tento návrh jsme předložili uživatelům, kteří na základě vyplnění dotazníka poskytly zpětnou vazbu. Výsledek řešení této problematiky je uživatelské rozhraní, které umožňuje pracovat v rámci organizace, rovněž sledovat stav a vytvářet nové úlohy, přičemž je možné porovnat výsledky se známými řešeními.

## Abstract

Thesis presents planning task monitoring system, which allows to determine optimal solutions for a variety of scheduling problems, some of them can have NP-problem character, so solution can be found according to given time and available algorithms. The thesis analyzes the technologies to create a user interface for the system which uses open-source technologies, also analyzes Optaplanner system. We develop a design that is intuitive and easy to understand with a fairly steep learning curve. The proposal we have presented to users, which based on the completed questionnaire provide feedback. The result of solving this issue is user interface that allows you to work within organization as well and to monitor the state and create new jobs, while it is possible to compare the results with known solutions.

## 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, Plánování, MySQL, Uživatel, Uživatelská role, Obmedzení, Plánovací problém, Úloha 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, Zdeněk Letko, Red Hat .

## Citace

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

# System monitorování stavu plánovacích úloh

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Zdeňka Letka a Martina Večeře

.....  
Martin Maga  
7. května 2014

## Poděkování

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

© Martin Maga, 2014.

*Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.*

# Obsah

# Kapitola 1

## Úvod

V úvode by som Vás rád krátko oboznámil 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 správne riešeniu vhladom na dostupný čas a dostupné algoritmy, pričom existuje možnosť nenájdenia riešenia. 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 budúcej spolupráce s open-source komunitou, ktorá mi môže poskytnúť prácu na zaujímavom projekte. Rovnako by som medzi ciele svojej práce zaradil splnenie intuitívnosti rozhrania, splnenie formálnych požiadavok, rovnako objasnenie použitých technológií a dôraz na to, prečo boli tieto technológie vybraté. V nasledujúcich kapitolách sa budem postupne venovať použitým technológiám pre implementáciu a testovanie, rovnako aj vysvetlenie systému Optaplanner, aplikačnému serveru JBoss a nakoniec sa budem venovať aplikácií, jej analýze, návrhu, imlementácií, testovaniu a vyhodnoteniu.

## 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 dokážu takéto systémy ľahko a rýchlo vytvárať. Jedným z takýchto prostriedkov je platforma Java Enterprise Edition (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 komplexnejších 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 technológií, webových aplikácií, podnikovej logiky a . . . . Nám budú postačovať prvé 3 špecifikácie tejto platformy, ktoré rozobereme v nasledujúcej časti spolu s technológiami, ktoré ich reprezentujú. Na základe Java boli implementované boli implementované rôzne Java EE kontajnery, ktoré sú potrebné pre správu a beh aplikácie. My sa zameriame na open-source riešenia. 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 zdôraznený princíp POJO (Plain Old Java Objects) [?] a zjednodušenie tvorby balíkov. V poslednom rade musí spomenúť princíp „Convention 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 [?]. 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 vrstva poskytuje bezpečnosť a transakcie. Najnižšia vrstva poskytuje pripojenie k databázam. Java EE je platformou, ktorá posky-

tuje 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 Community process(JCP). Aplikácie pre platformu Java EE sú vyvíjané prostredníctvom API, ktoré táto platforma poskytuje. Medzi tieto API patrí napríklad: Java Server Faces, Java Persistence API, Enterprise Java Bean, . . . . Behovým prostredím sú aplikačné servere, 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 nasaďovať naprieč rôznymi aplikačnými serverami. Medzi takéto prostriedky patrí komponenty na zabezpečenie aplikácie, ktoré riešia bezpečnosť pomocou prístupových pravidiel, ktoré sú interpretované za behu aplikácie. V ďalšej kapitole rozobereme aplikačný model platformy Java EE.

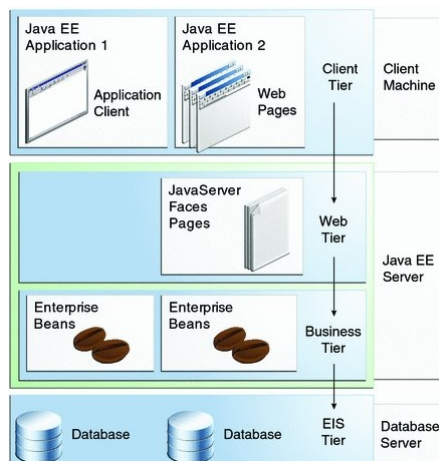
## 2.3 Aplikačný model

Java EE definuje aplikácie, ktoré sú viacvrstvové(multitier). Aplikačná logika je rozdelená medzi komponenty podľa ich funkcie[?]. 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 stupne sa skladajú z rôznych komponent, pričom stupne sú rozdelené nasledovne:

- Klientský stupeň sa skladá z klientských komponent, 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. ?? . Java EE aplikácia beží na klientskej stanici, býva obvykle reprezentovaná tenkým klientom(webovým prehliadačom), nazývaným „thin client“(pretože sa nedotazuje priamo na databázový server), alebo hrubým klientom, do ktorého 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 dynamické webové stránky pozostávajúce z rôzneho značkovacieho jazyka(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 vytváraná technológiou Swing alebo Abstract Window Toolkit[?], po prípade sa vyskytuje aj prístup prostredníctvom príkazového riadku. V 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





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

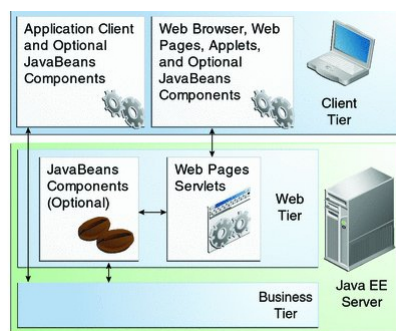
Faces a JavaServer Pages. Druhá časť strednej vrstvy takzvaná podniková vrstva býva reprezentovaná technológiu EnterpriseJava Beans, ktoré vytvárajú logiku aplikácie. Java EE server môže byť reprezentovaný, ešte okrem spomenutých technológií, rôznymi inými dostupnými technológiami, v závislosti od možnosti 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 obsahuje 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. JDBC).

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. Servlety sú javovské triedy, ktoré dynamicky spracovávajú požiadavky a tvoria odpovede. Súčasťou servletov alebo webových stránok 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 aplikácie 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.?? je ukázaný princíp fungovania webových komponent. V hornej ľavej časti obrázku sa nachádza klientská vrstva, ktorá obsahuje buď len webový prehliadač po prípade Applety alebo JavaBean komponenty, ktoré čiastočne obsahujú logiku aplikácie. V hornej pravej časti môže byť klient reprezentovaný aplikačným klientom, ktorý obsahuje úplnú prezentačnú logiku aplikácie a teda v tom prípade, odpadá potreba spracovania vstupov po prípade nejake 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 dispozícii tenkého klienta, klient komunikuje prostredníctvom webového prehliadača s HTML alebo XHTML stránky, ktoré sú vytvorené technológiou JavaServer Faces alebo JavaServer Pages, 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“ stránkam vytvorené prostredníctvom JavaServer Faces alebo JavaServer Pages technológiou a následne zobrazená užívateľovi v podobe výstupu na webovú stránku. V nasledujúcich dvoch podkapitolách sa bližšie pozremo 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 sa skladá z ďalšieho frameworku, ktorý obsahuje rôzne komponenty 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 na serverovú časť. Môže obsahovať rôzne deskriptory a konfiguračné súbory, ktoré nám pomáhajú pri nasadzovaní aplikácie. Základom JavaServer Faces je Facelets, čo je vlastne

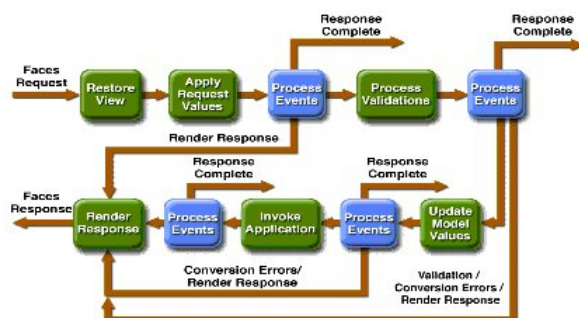
framework , ktorý slúži k tvorbe prezentačnej vrstvy webových aplikácií. Tento pomáha vytvárať JSF pohľady prostredníctvom HTML a buduje strom komponent. Facelets využíva XHTML pre vytváranie stránok, rovnako podporuje Facelets tag library, ktoré obsahujú obsahujú rôzne deklarácie komponent, ktorých použitie je nevyhnuté pokiaľ chceme použiť nejakú komponenty z danej knižnice. Tagy jednotlivých knižníc sú rozlišované na základe menných priestorov.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 vygenerujú z facelets a EL html komponenty, ktoré sú viazané na javovské triedy(ich vlastnosti a metódy), ktoré sa nazývajú „Managed Bean“. Managed Bean sú javovské triedy, 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 obstaráť validáciu vstupov, následne metódy a vlastnosti, ktoré sú volané alebo sú im predávané údaje z vygenerovanej 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 balíkovaní, rovnako aj typ managed beany. Typ managed beany môže byť:

- @RequestScoped Managed beana prežíva pokiaľ existuje HTTP požiadavok. Vytvára sa pri vytvorení požiadavku a zaniká pri zrušení HTTP požiadavku
- @NoneScoped Managed Beana existuje tak dlho ako existuje vyhodnotenie Facelets na stránke, 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 invalidácii 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 deskriptor nasadenia. Tento súbor definuje všetky informácie,

ko ktorých musí server vedieť, napr. použité servlety, inicializačné parametre, uvítacie stránky, .... V poslednom rade treba uviesť životný cyklus JSF aplikácie.

Celý štandardný cyklus spracovania požiadavky a následne generovania odpovedi je popísaný na nasledujúcom obrázku. Na obrázku č.?? môžeme vidieť životný



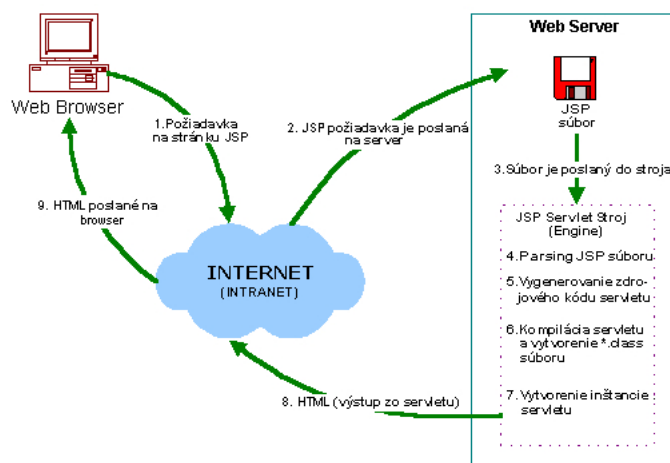
Obrázek 2.3: JSF životný cyklus [<http://docs.oracle.com/javaee/1.4/tutorial/doc/>]

cyklus JSF aplikácie. Počas fázy Restore View, keď je kliknuté na tlačidlo alebo na link sa vytvorí náhľad stránky, spoja sa všetky spracovania udalostí, validátory a komponenty a uložia sa do inštancie FacesContext. V ďalšej fáze Apply Request Values nové hodnoty sú získané použitím metódy decode. Hodnoty sú potom uložené lokálne do komponenty. Pokiaľ nastane chyba, tak je propagovaná a generovaná do FacesContext-u. Na konci tejto fázy sa vykoná znova dekodovanie pokiaľ stály nejaké nové hodnoty vo fronte na spracovanie. Vo fáze Process Validations spracuje všetky registrované validátory ku komponentám. Pokiaľ nastala chyba tak je táto informácia uložená do FacesContext-u. Počas ďalšej fázy Update Model Values nastaví do komponent lokálne nové hodnoty. Počas predposlednej ázy Invoke Application je spracované rozličné žiadosti ako potvrdenie formulára alebo link na inú stránku. V poslednej fáze Render Response dôjde k renderu stránku s novými hodnotami v kotajnety.

## 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 nazývajú skriptlety. Tieto skriptlety 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 žiadost 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.

Na nasledujúcom obrázku č.?? je užívateľ reprezentovaný webový prehliadačom, ktorý zažiada o JSP stránku. Táto požiadavka je predaná Java EE serveru, ktorý zistí,



Obrázek 2.4: JSP architektúra [http://interval.cz/clanky/javaserver-pages-pro-vsechny/]

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. Výstupom zo servletu, ktorý vznikol ako požiadavka o JSP stránku je html stránka, ktorá je predaná užívateľovi, ktorý si ju zobrazí. V poslednom rade by som uviedol životný cyklus JavaServer Faces stránky, ktorý začína generovaním HTTP požiadavku klienta o stránky a končí odpoveďou servera stránkou, ktorá je preložená do HTML kódu. Životný cyklus generovania odpovedi na strane servera je oveľa zložitejší, keďže sa berú do úvahy rôzne interakcie medzi komponentami na stránke a iné závislosti.

### 2.4.3 Web Service

Web Service je softwarový systém navrhnutý na podporu interoperability medzi rôznymi zariadeniami prostredníctvom počítačovej siete. Komunikácia prebieha prostredníctvom HTTP protokolu vymenianím XML správ. Tieto aplikácie poskytujú interoperabilitu medzi rôznymi platformami naprieč počítačovou sieťou. Web Service umožňuje komunikáciu medzi rôznymi aplikáciami, ktoré bežia na rôznych platformách, napr. Java aplikácie bežiacie na linuxe komunikujú s Net aplikáciami na Windowse. Tento aspekt je umožnený tým, že aplikácie komunikujú prostredníctvom HTTP protokolu. Na konceptuálnej úrovni môžeme web service chápať ako softwarové komponenty poskytujúce prístup ku koncovému bodu. Týmto koncovým bodom môžeme rozumieť systém inej organizácie, od ktorej potrebujeme získať nejaké dáta. Rovnako sa môže jednať aj o systém v rámci organizácie. Komunikácia prostredníctvom web service sa delí na 2 účastníkov. Prvý účastník produkovateľ(producer), ktorý vytvára požiadavok a spotrebiteľ(consumer), ktorý prijíma

požiadavok. Komunikácia prebieha medzi týmito dvoma účastníkmi výmenov správ. Web service môže byť technicky implementovaný rôznymi možnosťami a prostredníctvom Big Web Service alebo Restful WebService. Tento typ technológie bol zvolený z dôvodu potreby komunikácie užívateľského rozhrania so systémom na výpočet, ktorý sa nachádza na inom výpočte zariadení a tieto zariadenia si vymieňajú informáciu prostredníctvom HTTP protokolu. Umožňuje nám teda zdieľať funkčnosť kódu prostredníctvom siete a jej vzdialené volanie. Ďalšou výhodou tejto technológie využívajú štandardizované protokoly komunikácie, čo umožňuje jednoduchú interoperability medzi inými systémami. Keďže táto technológia používa pre komunikáciu HTTP protokol využíva pre komunikáciu prostredníctvom štandardnej počítačovej siete, čo znižuje náklady na prevádzkovanie na rozdiel od iných proprietárnych technológií.

## "Big" Web Services

Big Web Service je druh web service, ktorá pre svoju implementáciu používa API JAX-SW[?] "Big". Tento typ web service umožňuje vytvárať web service orientované na správy alebo web service, ktoré využívajú vzdialené volanie procedúr(RPC). Tento typ web service využíva XML správy, spolu so Simple Object Access Protocol(SOAP) a XML jazykom, ktorý definuje architektúru a formát správ. Nasledujúci obrázok č. ?? ukazuje



Obrázok 2.5: Big Web Service [<http://docs.oracle.com/javase/6/tutorial/doc/>]

spôsob komunikácie medzi klientom, ktorý sa nachádza v ľavej časti obrázku a web service, ktorá sa nachádza vpravej časti obrázku. Komunikácia prebieha prostredníctvom vymienania SOAP správ. Rovnako ako na klientovi tak aj web service obsahuje potrebné API, ktoré spracováva SOAP správy a predáva ich ďalej.

Tento typ Web Service obsahuje definíciu pre Web Service vo formáte WSDL, ktorý je čitateľný aj počítačom vo formáte xml. Formát SOAP správ a definíciu jazyka WSDL rozhrania môže znížiť zložitosť vývoja aplikácií, webových služieb, pričom toto rozhranie je definované prostredníctvom XML. SOAP definuje štruktúru kódovania správ, konvenciu pre reprezentáciu web service, rovnako aj spôsob volania a odpovedí. Správy volania a odpovedí web service sú vymieňané prostredníctvom SOAP správ prostredníctvom HTTP protokolu. JAX-WS API je pomerne komplikované, preto celá komplexnosť je vývojárovi zakrytá a je jediné, čo definuje vývojár sú metódy, ktoré je možné vzdialene volať. Rovnako vývojár nespracováva SOAP správy, ale celá táto problematika je riešená prostredníctvom prostredníctvom API. Veľká výhoda je platformová nezávislosť, ktorá je dosiahnutá prostredníctvom Javy. Tak isto toto API umožňuje prístup k ne-Javovským web service, čo prináša veľkú flexibilitu. Čo sa týka vývoja web service, tak sa jedná o jednoduchú Java triedu, ktorá používa anotáciu javax.jws.WebService, konkrétne anotáciu @WebService, ktorá označuje, že sa jedná o web service endpoint. Táto trieda následne

definuje metódy, ktoré môžu byť vzdialené volané. Aby mohla byť metóda metódou web service musí byť anotovaná prostredníctvom anotácie `javax.jws.WebMethod @WebMethod`. API ponúka aj ďalšie možnosti ako ovplyvňovať životný cyklus web service. Dôvod vybratia tohto druhu web service je ten, pretože umožňuje využívať mechanizmy pre QoS a poskytuje štandardy na zabezpečenie spoľahlivosti.

## 2.5 Java Persistence API

Java Persistence API(JPA) je špecifikácia jazyku Java, ktorá poskytuje prístup, spravovanie dát medzi Java objektmi a triedami a relačnými databázami. Základnou jednotkou JPA je entita, čo je vlastne odľahčený perzistentný doménový objekt, ktorý typicky reprezentuje tabuľku v relačnej databáze a každá jej inštancia je riadkom v tabuľke. Základný artefaktom v programovaní je pre entity entitná trieda, ktorá obsahuje vlastnosti, ktoré priamo odpovedajú stĺpcov v databáze. Každá entitná trieda priamo musí byť anotovaná `javax.persistence.Entity` anotáciou. Rovnako musí mať parametrický konštruktor, aby bolo možné vytvárať nové entity. Každá vlastnosť musí pritom spĺňať princíp POJO(Plain Old Java Object), čo znamená, že pre každú metódu musí existovať metóda `get` a `set`. Jednotlivé hodnoty môžu byť priamo validované prostredníctvom API `JavaBeans Validation` prostredníctvom, ktoré je možné definovať anotácie `avax.validation.constraints`, ktoré definujú aby daná položka spĺňala rôzne vlastnosti(nenulovosť, požiadavky na špeciálny formát vlastnosti, ...). Každá entitná trieda má unikátny identifikátor. Týmto identifikátor sa chápe primárny kľúč, ktorý môže byť, môže byť jednoduchý alebo môže byť zložený. Jednoduchý primárny kľúč sa v entitnej triede uvádza prostredníctvom anotácie `javax.persistence.Id`. V poslednom rade môže byť každá entita vo vzťahu s inými entitami. Na označenej danej vlastnosti, že sa súčasťou vzťahu s nejakou entitou používame anotácie `One-to-one`, `One-to-many`, `Many-to-one`, `Many-to-Many`, pričom prostredníctvom anotácie `javax.persistence.JoinColumn` nám zabezpečuje naviazanie vzťahu s inou entitou. JPA ponúka aj iné, pokročilé možnosti mapovania, pre naše potreby nám budú stačiť nasledujúce informácie. Spravovanie entít je zabezpečované triedou `EntityManager`, ktorá je reprezentovaná triedou `javax.persistence.EntityManager`. Každá inštancia tejto triedy je spojená s perzistentným kontextom a určitou množinou entitných tried. Entity manager vytvára a odstraňuje perzistentne entity, umožňuje vyhľadávať entity, rovnako aj vytvárať SQL dotazy nad databázou. Entity manager pracuje s persistence unit, čo odpovedá definícii entitných tried, s ktorými entity manager pracuje, teda definuje databázové tabuľky, s ktorými entity manager pracuje. Persistence unit sú definované v súbore `persistence.xml`. JPA rovnako definuje vlastný jazyk Java Persistence Query Language(JPQL), čo je jazyk podobný jazyku SQL, ktorý využíva trieda `EntityManager` pri svojej práci. Je to jednoduchý reťazcovo založený jazyk na spravovanie entít a vzťahou. Výhodou je, že tento jazyk je nezávislý na zvolenej databázovej technológii a má objektové vlastnosti, teda pri dotazoch nepoužívame konkrétne názvy tabuliek ale názvy entitných tried a jej vlastností. Problém JPQL je typová nebezpečnosť, čo vyžaduje pretypovanie výsledkov dotazu z entity manager-a. To môže spôsobiť chyby, ktoré nemusia byť odchytené počas kompilácie. JPA definuje ešte Criteria API, ktoré je využívané k vytvá-

raniu dotazovou nad entitami a vzťahy, ktoré sú typovo bezpečné. Výhodou tohto API, pre použitie na dotazovanie, je rovnako možnosť vytvárať dynamické dotazy, ktoré majú lepšiu výkonnosť ako JPQL. Pre naše potreby budeme používať obe varianty dotazovania nad dátami v databázy. V poslednom rade treba spomenúť, že JPA je nezávislé nad použitou databázovou technológiou, je možné vytvárať dotazy nad MySQL, SQL databázou . . . . Nakoniec treba zdôrazniť, že JPA nie je konkrétna implementácia ale špecifikácie, implementácie JPA môže byť: Hibernate, TopLink, OpenJPA.

## 2.6 Enterprise JavaBeans

EnterpriseJavaBeans(EJB) je technológia, ktorá vytvára Enterprise Beans(EB), ktoré predstavujú Java EE komponenty. EJB ďalej predstavuje spravovateľnú, server-side, komponentnú architektúru pre modulárne konštruovanie enterprise aplikácií. Podstatnou úlohou tejto komponenty je zapuzdrenie podnikovej logiky aplikácie. EB pomáha programátorom pri riešení často opakovaných problémov, napríklad perzistencia, bezpečnosť . . . . V prípade škálovateľnej aplikácie ponúka EB možnosť behu na viacerých zariadeniach, rovnako pre väčší počet užívateľov môže existovať viacero EB, ktoré môžu byť rozlične veľké a mať rozličné inštancie v závislosti od požiadavky klienta. EB sa môžu rôzne nachádzať na zariadeniach, ktoré sú prepojené prostredníctvom počítačovej siete a môžu byť volané prostredníctvom štandardného klient/server modelu. Také aplikácie môže byť distribuované a teda poskytovať obsluhu veľkého množstva užívateľov.

EB sa delia na 2 kategórie:

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

### 2.6.1 Session Bean

Session bean je typ EB, ktorá zapúzdruje podnikovú logiku, ktorá môže byť vyvolaná lokálne, vzdialene alebo prostredníctvom web service klienta. Prístup k session bean sa deje prostredníctvom volania metód bean-y. Bean-a následne vykoná podnikový kód na serverovej strane. Takýto typ beany nie je perzistentný. Táto beana môže byť ešte 3 typov:

- Stateful Session Bean - beany udržiava hodnoty premených, každá beana reprezentuje unikátny stav klienta/bean sedenia. Stav komunikácie beany a klienta sa často nazýva „conversational state“, môže mať len 1 klienta, rovnako sedenie môže interagovať len s 1 klientom. Stav je zachovávaný počas klientského/bean sedenia. Pokiaľ sa sedenie odstráni stav zmizne.
- Stateless Session Bean - Neudržiava conversational state s klientom. Počas invokácie metódy takejto beany môže inštancia obsahovať premenné, ktoré môžu obsahovať



špecifický stav vzhľadom na klienta, alebo len po počas invokácie metódy. Stav beanu špecifický pre klienta nepretrváva, ale tento stav môže pretrvať v poll stateless bean počas ďalšej invokácie metódy. Rovnako jednotlivé inštancie stateless bean sú ekvivalentné takže môžu byť priradené ľubovoľnému klientovi. Z toho vyplýva, že podporuje viacnásobný prístup klientov a lepšiu škálovateľnosť. Tento typ session beanu môže implementovať web service.

- Singleton Session Bean - Tento typ beanu je inštanciovaný len raz a pretrváva počas celého životného cyklu aplikácie. Využíva sa pri zdieľaní a súčasnom prístupe viacerých užívateľov. Rovnako aj tento typ session beanu môže implementovať web service, rovnako udržiava stav medzi invokáciami metód klienta. Tento typ beanu môže byť využitý pre špecifické akcie aplikácie, napr. ukončenie aplikácie, štart aplikácie, . . . , pretože pretrváva počas celého životného cyklu aplikácie.

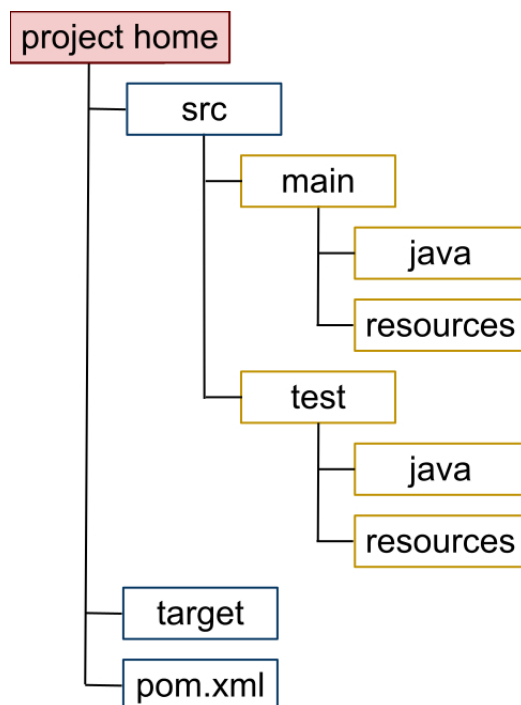
Tento typ beanu môžeme použiť pokiaľ potrebuje udržať stav medzi klientskými volaniami metód, rovnako pokiaľ potrebuje odľahčiť aplikáciu a zvýšiť výkonnosť použijeme tento typ beanu konkrétne stateless session bean. Tieto beanu dokážu implementovať web service z tohto dôvodu sme sa rozhodli pre použitie tohto typu EB pre implementáciu web service.

### 2.6.2 Message-driven Bean

Message-driven bean je typ EB, ktorá umožňuje Java EE aplikáciám asynchrónne spracovanie správ. Tento typ beanu sa správa rovnako ako Java Message Service(JMS)[?] naslúchavanie správ, ale narozdiel od JMS beana prijíma JMS správy. Tieto správy môžu byť poslané rôznymi Java EE komponentami, alebo JMS aplikáciou alebo aj iným systémom, ktorý nepoužíva Java technológiu. Tieto beanu nespracovávajú len JMS správy ale aj iné typy správ. Zásadný rozdiel je medzi message-driven bean a session bean v zásade v tom, že sa k takému to typu beanu neprístupuje prostredníctvom rozhrania a invokácie metód. Tento typ beanu obsahuje len bean triedy. Message-driven bean neudržiava dáta alebo conversational state pre klienta. Všetky inštancie takého typu beanu sú ekvivalentné, to umožňuje EJB kontajneru priradiť správy ľubovoľne message-driven bean inštancii. Jedna inštancia beanu môže spracovávať rozličné správy od klientov. Inštancia premenných môže udržiavať stav spracovania klientských správ napr. JMS API connection, databázové pripojenie, . . . . Klienti pristupujú k message-driven bean, napr. zasielaním správ do cieľa pre message-driven beanu je MessageListener. Message-driven bean má ďalšie zaujímavé vlastnosti a to, že môžu byť vyvolané asynchrónne, žijú relatívne krátko a sú bezstavové. Pokiaľ správa dorazí do message-driven bean kontajner zavolá metódu onMessage, ktorá pretypuje JMS správy na jeden typ správy a naloží s ňou podľa podnikovej logiky. Výhodou tejto beanu je posielanie asynchrónnych správ, ktoré nevyťažujú tak prostriedky servera. Tento typ beanu bol použitý pre prijatie požiadavok na spustenie plánovacích úloh a pozastavenie spracovania vzhľadom na možnosť asynchrónneho zasielania a zaradzovania požiadavkov do fronty.

## 2.7 Convetion over Configuration

Convetion over Configuration je sotwarový design, ktoré znižuje počet rozhodnutí, ktoré musí urobiť vývojár a zároveň zvyšuje jednoduchnosť. Po zjednodušení môže tento princíp chápať v zmysle, že vývojár nakonfiguruje len „nezvyklé“ aspekty aplikácie. Tento princíp sa snaží odstrániť množstvo konfiguračných súbor, ktoré robia daný projekt neprenositelný. Jedným z prostriedkov, ktorý implementuje daný princíp je Maven. Maven je stavebný automatizačný nástroj, ktorý je primárne používaný pre Java projekty. Maven definuje spôsob ako bude software zostavený, rovnako aj definuje závislosti (Dependency Management). Celý obsah je definovaný v XML súbore, v ktorom sa rovnako definujú závislosti na externé moduly, poradie zostavovania komponent a požadované pluginy. Obsahuje predefinované úlohy ako kompilácia, testovanie, balíkovanie a nasadzovanie. Maven dynamicky sťahuje Javovské knižnice a maven pluginy z jedného alebo viacerých repozitárov ako napr. Maven 2 Central Repository a ukladá ich v lokálnej cache. Maven projekty sú konfigurované prostredníctvom xml súbor, ktorý využíva Project Objec Model a nazýva sa pom.xml, pričom sa nachádza v koreňovom adresári projektu. Nasledujúci obrázok ukazuje štandardnú adresárovú štruktúru maven projektu:



Obrázek 2.6: Maven adresárová štruktúra [<http://maven.apache.org/>]

obrázok č. ?? ukazuje základnú adresárovú štruktúru maven projektu. Každý maven projekty sa skladá z project home, ktorý obsahuje súbor pom.xml a všetky ostatné podadresáre. Ďalej sa skladá z priečinkov src, kde sa nachádzajú zdrojové kódy a target, kde

sa ukladajú preložené triedy[?]. Adresár src sa ďalej skladá z adresáru main, ktorý ešte obsahuje adresára java, ktorý obsahuje java zdrojový kód pre daný projekt a resources, ktorý obsahuje prostriedky pre daný projekt ako sú rôzne súbore, ktoré obsahujú nastavenie prostriedkov pre daný projekt. Podadresár src sa skladá z adresára test, ktorý rovnako ako src obsahuje podadresár java, ktorom je umiestnený java zdrojový kód pre testovanie projektu. Podadresár test obsahuje ďalší adresár resources, ktorý obsahuje prostriedky potrebné pre testovanie. Celá táto štruktúra predstavuje základne adresáru štruktúru pre maven projekt a tak to robí viac prenositeľný. Jednotlivé závislosti pre projekt jednoducho definuje v súbore pom.xml. Preložením projektu sa preložia všetky triedy a uložia do adresára tagert. Celý projekt môže byť pre väčšiu modularitu rozdelený na moduly, pričom každý modul rovnako splňuje základnú Maven adresárovú štruktúru. Každý maven projekt je možné dať ako cieľ jednu z fáz životného cyklu. Životný cyklus môže byť z jednej fáz: 1. validate - validácia korektnosti projektu a kontrola dostupnosti potrebných informácií pre projekt ,2.kompilácia - kompilácia zdrojového kódu projektu, 3.test - testovanie zkompilevaného zdrojového kódu, táto fáza nie je vyžadovaná 4.package - zabalenie zkompilevaného projektu do balíku, napríklad jar, 4. integration-test - spracovanie a nasadenie balíku pokiaľ je to potrebné do prostredia, kde môžu bežať integračné testy, 5.verify-run - beh a overenie balíku, že spĺňa všetky kritéria pre spustenie, 6. install - inštalácia balíku do lokálneho repozitára, v prípade, že potrebuje použiť balík ako závislosť 8.deploy - nasadenie projektu do kontajneru a spustenie. Jednotlivé fázy môže spustiť príkazom „mvn názov životného cyklu“, napríklad mvn package. Na porovnanie výhod Maven som použil nástroj Apache Ant. Ant nástroj nedefinuje presnú adresárovú štruktúru a podporované úlohy narozdiel od maven-u. Definovanie úloh Ant-u sa môžu nachádzať vo viacerých súboroch, pričom každý súbor môže definovať jednu úlohu. Výhodou Ant-u je optimalizácie XML jazyka pre potreby Ant-u pre jasnu definíciu, čo robí a od čoho závisí. Laik je schopný okamžite pochopiť, čo vykonáva xml súbor ant-u.Na druhej strane výhodou Maven-u je, že vývojár, ktorý má skúsenosti s maven-om pochopí okamžite pri neznámom projekte štruktúru projektu a je schopný spustiť štandardné úlohy maven a pozná očakávaný výstup. Z tohto dôvodu som sa pochopil využiť Maven pre svoj projekt, ktorý má rovnako výhodu použitým Dependency Management.

## 2.8 Bezpečnosť

Na zabezpečenie aplikácie bol vybraný framework Seam. Seam je aplikačný framework pre enterprise Javu, ktorý definuje uniformný komponentný model pre podnikovú logiku aplikácie. Takéto komponenty môžu byť stavové, perzistentné a môžu udržiavať konverzačný kontext naprieč viacerými webovými žiadosťami. Seam rovnako rieši integráciu EJB a JSF spolu. Rovnako umožňuje písať Seam aplikácie, kde je všetka funkčnosť ukrytá v EJB. Medzi ďalšie výhodné vlastnosti tohto frameworku patrí integrácia Ajax-u, rovnako aj vstavaná podpora javascriptu a efektívne spracovanie webových dotazov. Seam obsahuje množstvo rôznej inej funkcionality, čím uľahčuje písanie enterprise aplikácií, rovnako aj ich udržiavanie a efektivity pri vykonávaní definovanej činnosti. My sa zameriame na modul Seam security, ktorý obsahuje množstvo mechanizmov na zabezpečenie

našej enterprise aplikácie. Základom každej bezpečnosti je autentifikácia, čo je process vytvorenia alebo potvrdenia identity užívateľa. Užívateľ potvrdzuje svoju identitu prostredníctvom užívateľského mena a hesla, ktoré sa nazýva „credentials“. Seam security poskytuje API prostredníctvom, ktorého je možné sa autentizovať z rozličných zdrojov (databáza, LDAP, ...). Ďalšou vlastnosťou je Identity Management, ktoré je množina API na správu užívateľov, skupín a užívateľských rol. Identity Management je poskytovaný v Seam komponentou PicketLink IDM, ktorá spravuje uloženie užívateľov v rozličných bezpečnostných úložiskách. Poslednými dvoma funkciami tohto modulu frameworku Seam je externá autentifikácia prostredníctvom externých autentifikačných služieb a autorizácia užívateľov na akcie, ktoré môžu v aplikácii vykonávať. Seam security je k dispozícii v prostredníctvom nástroja Maven. Základom autentifikácie je Identity Bean, ktorá reprezentuje identitu užívateľa a pri úspešnej autentifikácii je identita vložená do životného cyklu aktuálneho sedenia. V rámci autentifikácie sú definované metódy Login a Logout. Ďalšou podstatnou časťou je Credentials bean. Táto bean-a udržiava užívateľské informácie (meno, heslo) predtým než sa užívateľ prihlási. Seam security modul poskytuje v rámci autentifikácie rôzne autentifikačné triedy. Základom každej autentifikačnej triedy je metóda authenticate, ktorá autentifikuje užívateľa. Podstatná je metóda setStatus, v ktorej sa nastaví úspech (SUCCESS) alebo neúspech (FAILURE) pri overení zadaných údajov. Základom obmedzenia prístupu užívateľov je nainjektovanie triedy Identity, a to môže byť urobené nasledovne, napr. `@Inject Identity identity;` keď tento kód vložíme do managed beanu je možné potom v rámci managed beanu vytiahnuť informáciu o užívateľovi (užívateľskú rolu, užívateľské meno). Seam security modul poskytuje spôsob akým zabezpečiť svoje triedy a metódy anotovaním typesafe security binding. Každé typesafe security binding má autorizačnú metódu, ktorá je zodpovedná za to, že určí či užívateľ má odpovedajúce privilégia k invocácii danej metódy. Základom tvorby typesafe security binding je tvorba rozhrania, napríklad `public @interface Admin`. Následne je potrebné pre dané rozhranie vytvoriť autorizačnú metódu. Autorizačná metóda je anotovaná tagom `@Secures @Admin` (záleží od názvu rozhrania, ktoré môže reprezentovať skupinu užívateľov, užívateľskú rolu alebo môže ísť aj o konkrétneho užívateľa). Následne v nej sa overí prostredníctvom Identity triedy, že obsahuje správne údaje alebo mohol byť daný užívateľ priradený k danému rozhraniu. Tým pádom je umožnený prístup k špecifickým akciám, metódam, ... Pre potreby bezpečnosti bol použitý modul Seam Faces, ktorý je rovnako súčasťou frameworku Seam. Jeho úlohou je integrácia context and dependency injection (CDI) do JSF frameworku. Funkčnosť, ktorá nás zaujíma je Faces View Configuration. Ten nám umožňuje spojenie s Seam Security modulom na obmedzenie/povolenie prístupu pre danú užívateľskú rolu, prepisovanie URL, ... My sa zameriame na povolenie/obmedzenie prístupu k JSF stránkam. JSF stránky sú anotované prostredníctvom vlastností a umiestnené v Java enum-e. Základom je anotovanie rozhrania tagom `@ViewConfig` a následne tvorba statického enum-u. V ňom sú obsiahnuté anotácie na obmedzenie prístupu k JSF stránkam. Základom je tvorba rozhraní, ktoré obsahujú autorizačné metódy. Následne je možné názvy týchto rozhraní používať k povoleniu prístupu. Pokiaľ máme definované rozhranie Admin (môžeme ho používať anotáciou `@Admin`) tak pridružením tagu `@ViewPattern()`, ktorý odkazuje na nejakú

stránku hovoríme, že povolíme užívateľskej role Admin prístup len k stránke definovanej obsahuje tagu `@ViewPattern`. Takto môže definovať viaceré role. Rovnako môžeme definovať ako sa aplikácia zachová v prípade, že užívateľ sa snaží prístupit tam, kde nemá prístup. Tento prístup vykonáme použitím tagu `@FacesRedirect`, ktorý hovorí, že presmeruje `@ViewPattern("/*)`, teda z ktorejkoľvek stránky `@AccessDeniedView()` na obsahu daný tagom `@AccessDeniedView`. Týmto zabezpečíme pokiaľ užívateľ z role Admin sa snaží prístupit na stránku, kde má mať prístup len User, tak ho presmeruje na obsah tagu `@AccessDeniedView`. Seam framework patrí pod divíziu JBoss, takže je pomerne jednoduché ho integrovať pod aplikačný server JBoss-u. Rovnako je tento framework open-source, čo je jeho ďalšou výhodou.

## 2.9 Testovanie

V poslednom rade musí technológia, ktoré budem používať pre testovanie. Základom testovania sú 2 technológie, ktoré sme použili jednouchou z nich je technológia JUnit a druhá z nich je technológia Arquillian. V prvom rade by som sa rád venoval technológiám JUnit. JUnit je unit testovací framework pre programovací jazyk Java. JUnit sa používa pre typ testovania, ktorý sa nazýva test-driven development a je jedným z kolekcie unit testovacích frameworkov. JUnit je linkovaný ako jar súbor pri kompilácii a býva súčasťou balíku `org.junit`. [?]. JUnit testovací inventár je javovský objekt. Testovacie metódy sú anotované prostredníctvom `@Test` anotácie. JUnit rovnako umožňuje vykonať kód pred spustením testu, to docielime anotovaním metód `@Before` anotáciou alebo po sputení testu, to docielime anotáciou `@After`. V testovacej metóde potom vykonáme nejaký kód a očakávaný výstup porovnáme s nami očakávaným výsledkom prostredníctvom `Assert`. JUnit testy sú písané pre otestovanie konkrétnej funkčnosti kódu. Cieľom testovania prostredníctvom JUnit sú malé kúsky kódu, ako metódy alebo triedy. JUnit testy môžu použiť rovnako pri písaní kódu alebo pri refaktORIZácii na overenie požadovanej funkčnosti kódu. Všetky tieto výhody som využil pri testovaní malých kúskov kódu, ktoré vykonávali požadovanú funkčnosť.

Nakoniec by som rád spomenul technológiu Arquillian. Arquillian je testovací framework, ktorý vykonáva testy vo vnútri vzdialeného alebo vstavaného kontajneru alebo nasadí archív na kontajner, tak aby test mohol interagovať so vzdialeným klientom. Arquillian integruje aj ďalšie testovacie frameworky, napr. JUnit 4, TestNG 5, ..., rovnako aj iné frameworky Ant, Maven a iné. Musím zdôrazniť, že na rozdiel od JUnit testov umožňuje testovanie v java EE kontajnery (GlassFish, JBoss) [?]. Tento framework má zásadnú výhodu na prenositeľnosť testov na rôzne podporované kotajnery. Testy môžu byť spustené rovnako z IDE tak aj zostavovacím nástrojom. Framework rozširuje alebo integruje existujúce testovacie frameworky. Framework automaticky zabalí do testovacej platformy všetky potrebné prostriedky. V poslednom rade treba spomenúť, že spustenie arquillian testu je veľmi jednoduché. Použitie arquillianu je dané použitím anotácie `@RunWith Arquillian`, ktoré zabezpečí spustenie testov po spustení. Následne arquillian sputí kontajner nasadí testovací archív, ktorý je daný anotáciou `@Deployment`. Archív obsahuje testy so špecifickými triedami a knižnicami. Testy sa následne vykonajú vo vn-

útri kotajneru. Čo znamená, že sa použijú všetky dependency a resource inject do testov, takže môžeme pristupovať k EJB, rovnako môže pristupovať k databáze, . . . . Na prvý pohľad vyzerajú arquallian testy ako štandardné JUnit testy. Arquallian umožňuje testovať reálne komponenty, ktoré interagujú s java ee kontajnerom. Arquallian umožňuje spustiť testy vo viacerých módov. Jedným z nich je mód In-cotainer, kde testovanie prebieha priamo v java ee kontajnery pre interakciu s java ee komponentami, druhá možnosť je Client mode, kde testovanie prebieha u klienta u ukazuje ako klient využíva aplikáciu, posledna možnosť je použiť Mixed mode, ktorý kombinuje obe predchádzajúce možnosti. V poslednom rade je potrebné nastaviť kontajner, ktorý sa použije na testovanie. Konfigurácia sa deje prostredníctvom xml súboru arquillian.xml.

## Kapitola 3

# JBoss Application Server

Aplikačný server(AS) je software, ktorý poskytuje vrstvu medzi operačným systémom a java ee aplikáciami. AS poskytuje základnú funkciu programom(prístup k súborovému systému, posielanie správ, ...), konkrétne enterprise aplikáciám. Vytvára vrstvu, ktorá zjednodušuje vývoj enterprise aplikácie. Dôvod použitia pre enterprise aplikácie je ten, že tieto aplikácie sú robustné a komplexné a spracovávajú súčasne veľké množstvo požiadavkou od klientov, pričom typickou aplikáciou môže byť webová aplikácia. Pomerne veľká skupina AS je vyvíjaná v jazyku Java. Dôvodom pre tento jazyk existencia štandardu pre enterprise aplikácie a to je Java EE. Nás bude zaujímať implementácia Java EE, ktorá sa nazýva JBoss AS.

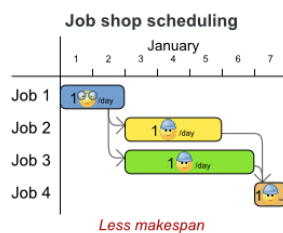
JBoss, čo je vlastne skratka pre JavaBeans Open Source Applicatom Server, je aplikačný server, ktorý je založený na platforme Java a Java Enterprise Edition.[?]. Tento typ AS je open-source, preto je možné jeho stiahnutie spolu so zdrojovými kódmi. Používanie aplikačného servera JBoss je veľmi jednoduché jeho spustenie môžete vykonať ručne prostredníctvom konzoly a nájdeným inštalačného adresára JBoss-u a následne adresára bin, ktorý obsahuje skript run.sh, ktorý spustí AS. Druhou možnosťou je spustenie prostredníctvom IDE. Po spustení serveru je možné k nemu implicitne pristupovať na localhost na porte 8080 počítača. Základným stavebným kameňom JBoss AS je JBoss Microcontainer. JBoss Microcontaijner je refaktORIZÁCIA JBoss JMX Microkernel aby podporoval POJO nasadzovanie a samostatné použitie mimo aplikačného servera. Microcontainer plní funkciu jadra, do ktorého sa registrujú všetky služby. Služby, ktoré majú byť prístupné sa registrujú v podobe managed beans. Microcontainer spravuje a riadi beh týchto služieb. Do AS sa registruje všetko jeho funkcionality, vrátane Java Management Extention, ktorý umožňuje správu zaregistrovaných služieb. Prostredníctvom tohto rozhranie môžeme spravovať všetky zaregistrované služby. JBoss implicitne podporuje databázový server Hypersonic SQL, ktorý má ale obmedzené možnosti, ktorý je určený len na testovanie. Do tohto databázove servera si služby ukladajú informácie. Tento server je licenovaný pod GNU Lesser General Public License(GNU PL). Dôvod výberu tohto aplikačného servera je integrácia open-source technológií, ktoré sú založené na Jave. Rovnako ako JBoss poskytuje implementáciu širšej škály Java EE technológií, ktoré potrebuje pre vývoj svojej práce. Aplikácie, ktoré bežia na JBoss sú oveľa komplexnejšie.

## Kapitola 4

# OptaPlanner

OptaPlanner je odľahčený open source software a ďalšie pokračovanie frameworku JBoss Drools, ktorý vykonáva a optimalizuje plánovacie problémy. Tento framework je určený pre programovací jazyk java a pomáha programátorom riešiť problémy s obmedzeniami efektívne. Optaplanner kombinuje optimalizačné heuristiky a metaheuristické metódy prostredníctvom kalkulácie skóre. Skóre je hodnota, ktorá reprezentuje bodové hodnotenie riešenia daného problému. Daný plánovací problém môže byť riešený rôznymi optimalizačnými algoritmami. Lepším riešením daného problému je to, ktoré má vyššie skóre. Optaplanner je možné použiť k optimalizácií plánovacích problémov. V bežnom živote, rovnako ako ja v podnikových sférach sa stretávame s rôznymi plánovacími problémami. Môže ísť o problémy ako správne naplánovať cestu vozidiel(aút, lodí, ...), aby sme ju spravili za čo najkratší čas, rovnako môžeme požadovať aby cesta bola, čo finančne najefektívnejšia. Rovnako môžeme plánovanie rozvrhu práce zamestnancov vo firme, aby zbytočne nespomalovali chod a ostatných zamestnancov, ktorí sú na ich práci závislí nemuseli zbytočne čakať. Plánovať môžeme spúšťanie testovania aplikácií v rámci vývojarskej firmy, aby niektoré úlohy boli otestované skôr ako iné no musí byť čo najefektívnejšie vyvážené a zbytočne nemrhali časovým kvantom. Plánovacích problémov existuje celá rada a každý deň prichádzame s novými. Pokiaľ je problem dostatočne komplexný potom je veľmi vhodné použiť Optaplanner. Optaplanner môžeme používať prostredníctvom maven-u, ktorý poskytuje potrebné triedy na výpočet a konfiguráciu systému, rovnako ako aj nastavenie definičného súboru problému vo formáte xml. Optaplanner je možné použiť aj k riešeniu nedeterministicky polynomiálnych problémov. Na nasledujúcom obrázku môžeme vidieť príklad plánovacieho problému





Obrázek 4.1: Job, Shop scheduling [<http://www.optaplanner.org/>]

Obrázok č. ?? zobrazuje typické použitie OptaPlanner-u. Môžeme 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 Princíp

Riešenie problému je postavené na práci s definícou problému, ktorý je reprezentovaný xml súborom. Ako sa má daný problém riešiť je dané kódom v Jave. Riešenie je možné ovplyvnoť konfiguráciou Optaplanner-u, v ktorom môžeme nastaviť optimalizačné algoritmy plus ďalšie heuristiky, rovnako aj spôsob kalkulácie skóre. 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. Najlepším riešením je riešenie s najvyšším skóre.

## 4.0.2 Ukážka XML configuračného súboru

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

Listing 4.1: Test

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <solver>
3   <!--environmentMode>FAST_ASSERT</environmentMode-->
4   <!-- Domain model configuration -->
5   <solutionClass>org.optaplanner.examples.cloudbalancing.domain.
      CloudBalance</solutionClass>
6   <planningEntityClass>org.optaplanner.examples.cloudbalancing.domain.
      CloudProcess</planningEntityClass>
7   <!-- Score configuration -->
8   <scoreDirectorFactory>
9     <scoreDefinitionType>HARD_SOFT</scoreDefinitionType>
10    <simpleScoreCalculatorClass>org.optaplanner.examples.cloudbalancing
        .solver.score.CloudBalancingSimpleScoreCalculator</
        simpleScoreCalculatorClass>
11    <!--<scoreDrl>/org/optaplanner/examples/cloudbalancing/solver/
        cloudBalancingScoreRules.drl</scoreDrl-->
12  </scoreDirectorFactory>
13  <!-- Optimization algorithms configuration -->
14  <termination>
15    <maximumSecondsSpend>120</maximumSecondsSpend>
16  </termination>
17  <constructionHeuristic>
18    <constructionHeuristicType>FIRST_FIT_DECREASING</
        constructionHeuristicType>
19    <!--forager-->
20    <pickEarlyType>FIRST_NON_DETERIORATING_SCORE</pickEarlyType>
21    <!--/forager-->
22  </constructionHeuristic>
23  <localSearch>
24    <acceptor>
25      <entityTabuSize>7</entityTabuSize>
26    </acceptor>
27    <forager>
28      <acceptedCountLimit>1000</acceptedCountLimit>
29    </forager>
30  </localSearch>
31 </solver>
```

Konfigurácie solveru pozostáva z 3 častí:

- Domain model configuration (ja umiestnené medzi tagom `<solutionClass>`): Definuje hlavnú triedu pre riešenie. Ďalej sa skladá z triedy, ktorá zaobstaráva plánovanie (planningEntityClass).
- Konfigurácia skóre, ktorá hovorí Optaplanneru ako ma optimalizovať premenné (tag `<scoreDirectorFactory>` a podtag `<scoreDefinitionType>`). 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 jednoduchéj implementácie Java (umiestnime triedy to tagu `<simpleScoreCalculatorClass>`, alebo pomocou Drols DRL (umiestnime do tagy `<scoreDrl>`). Optaplanner bude hľadať riešenie s najvyšším skóre. Budeme používať HardSoftScore, čo znamená, plánovač bude hľadať riešenie s žiadnymi tvrdými obmedzeniami členenie (splňajú požiadavky na hardvér) a pokiaľ možno čo najmenej mäkkých obmedzenia členenie (minimalizovať náklady na údržbu).
- Konfigurácia optimalizačných algoritmov (tag `<constructionHeuristic>` a podtag `<constructionHeuristicType>`). V našom prípade použijeme optimalizačný algoritmus FIRST\_FIT DECREASING
- Voliteľne je možné definovať aj metaheuristiky (umiestnené v tagy `<localSearch>`). V našom prípade bola použitá metaheuristika TABU SEARCH

#### 4.0.3 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 sa pokúsi umiestniť na položku v prvej priehradke, kde ju môže vložiť. Ak nie je nájdený žiadna priehradka, otvára novú priehradku a kladie položku v rámci nového zásobníka.
- First FIT Decreasing - Tento algoritmus na rozdiel predchádzajúceho funguje nasledovne: Položky sú zoradené podľa ne-rastúcej veľkosti. Ďalším bodom je vždy zabalená položka do prvého koša, kde sa hodí.
- First FIT Decreasing + heuristic local search - Algoritmus funguje nasledovne: Položky sú zoradené podľa ne-rastúcou veľkosti. Ďalším bodom je vždy zabalené do prvého koša, kde sa hodí. Local search je metaheuristická metóda pre riešenie výpočtovo zložitých optimalizačných úloh. Miestne vyhľadávanie je možné použiť na problémy, ktoré môžu byť formulované ako hľadanie riešení maximalizuje kritérium medzi niekoľkými kandidátnymi riešeniami. Miestne algoritmy sa pohybujú z roztoku do roztoku v priestore kandidátnych riešení (priestor hľadania) za použitia

lokálnych zmien, kým je roztok považovaný za optimálny je nájdený, alebo uplynul čas.

- Hill Climbing - hill climbing je matematická optimalizácia technika, ktorá patrí do rodiny miestneho vyhľadávania. Jedná sa o iteratívny algoritmus, ktorý začína s ľubovoľným riešením 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.[?]
- Tabu Search - Tabu search používa miestny alebo susedský postup vyhľadávania tak, že iteratívne presúva z jedného možného riešenia  $x$  k lepšiemu riešeniu  $x$  v susedstve  $x$ , kým sa niektoré kritérium zastavenia splní. Miestne postupy vyhľadávania často uviaznu v zle ohodnotených 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 Annealing je všeobecný algoritmus pre globálne optimalizačné problémy lokalizovať dobré priblíženie k globálnej optimálnej danej funkcii vo veľkom vyhľadávacom priestore. To sa často používa pri hľadaní priestorov diskretných. U niektorých problémov, môže simulované žiňanie byť účinnejšie ako vyčerpávajúci zoznam - za predpokladu, že cieľom je iba nájsť prijateľne dobré riešenie v stanovenú dobu, skôr ako tým najlepším možným riešením.

#### 4.0.4 Výsledky plánovacieho problému

Každý plánovací problém je definovaný na základe obmedzení, ktoré musia minimálne spĺňať: [?]

- 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 pomocou pravidiel môže robiť spájanie s inými akciami jednoduchším. Tieto pravidlá bývajú typicky definované pomocou XML súboru. Optaplanner spája pri riešení skóre a plánovacie algoritmy.

Obmedzenia definujú výpočetné skóre problému plánovania. Každé riešenie problému plánovania 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 plánovania majú 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 majú 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 nájsť tieto veľké množstvá 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-u.

## Kapitola 5

# Aplikácie

V tejto kapitole postupne rozobereme problematiku grafického užívateľského rozhrania, použité technológie Rich Faces a Twitter Bootstrap v kombinácii s JavaServer Faces. Výsledné rozhranie bude môcť nahrávať definície problému, zobrazovať výsledky, spúšťať, pozastovať a zobrazovať detaily úloh a nakoniec vyhľadávať úlohy. Keďže táto výsledná aplikácia by mala byť použiteľná aj na mobilnom telefóne bol vybratý štýlovací framework Twitter Bootstrap, ktorý značne uľahčuje tvorbu takéhoto rozhrania. V nasledujúcich kapitolách rozobereme aplikáciu postupne od analýzy až po vyhodnotenie.

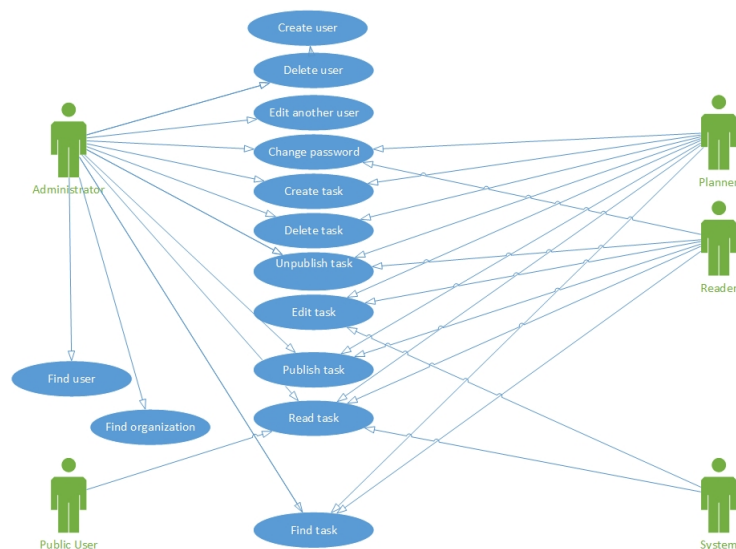
### 5.1 Špecifikácia požiadavkov

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 kombinácii s Rich Faces nastýlované prostredníctvom frameworku Twitter Bootstrap, 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 prístupná podľa príslušnosti užívateľa k užívateľskej role. Z tohto rozhrania bude môcť užívateľ pokiaľ mu to užívateľská rola dovoľuje pridať úlohu, ktorú môžeme 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 možnosti 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. Nasledujúci obrázok ukazuje príklady užívania systému: Na nasledujúcom obrázku č.?? môžeme identifikovať 4



Obrázek 5.1: UseCase diagram

užívateľov systém. Public user je verejný užívateľ, ktorý môže čítať úlohy, ktoré sú nastavené ako verejné(public) a sú publikované. 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, viac akcií mu nie je umožnené. Plánovač môže úlohy čítať, vytvárať úlohy v databázi, 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 rovnaké ackie môže vykonávať pre užívateľov. Zároveň môžu Administrátor, plánovač a čitateľ vyhľadávať v daných úlohách. Administrátor môže následne vyhľadávať užívateľov a organizácie.

## 5.2 Analýza

Výsledné aplikácia by mohla byť rozdelená na 3 časti. Výsledné rozhranie by mohlo byť vytvorené prostredníctvom technológie Javaserer Faces, ktoré podporuje množstvo komponent na spracovanie a zobrazenie informácií. Pre spracovanie informácií by sme mohli

použiť managed beany, rovnako by mohla byť použitá technológia JavaServer Pages, ktorá podporuje skripty. Druhou časťou aplikácie by bola Web Service rozhranie, ktoré poskytuje metódy na spustenie a pozastavenie úlohy, ktoré bude volané z užívateľského rozhrania. Výpočet by prebiehal v enterprise beane rovnako by bola možnosť výpočet nechať na metódu web service. Rovnako trebalo zvoliť použitú databázovú technológiu existuje veľa možnosti medzi relačnými a nerelačnými technológiami. Medzi 2 zvolené technológie bola relačná technológia MySQL a nerelačná technológia MongoDB. V poslednom rade trebalo zvoliť správny technológiu pre nastylovanie a zabezpečenie prenositeľnosť na mobilný telefón. Existuje komplikovaná možnosť vytvorenia CSS štýlu alebo zvolenia dostupných frameworkov, ktoré túto prácu vyriešia za nás. Jedným z takýchto frameworkov je Twitter Bootstrap. Aplikácia potrebuje pre spracovanie úloh, organizácií a užívateľov databázu. Pre potreby bakalárskej práce bol vybraný relačný open-source databázový model MySQL. MySQL databázová technológia je veľmi vhodná pre malé a stredne veľké 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 diskového priestoru, rovnako je MySQL multiplatformová, keďže je možné ju nasadiť na systémy s operačným systémom windows, linux, max os. Medzi nevýhody tejto technológie patrí neefektívna práca s databázovými transakciami, neefektívne ukladanie veľkého množstva dát. Pre prenositeľnosť na mobilný telefón, rovnako aj pre rýchlu implementáciu riešenia spolu s jeho rozšírením „Awesome Font“. Pre tvorbu rozhrania a spracovanie a zobrazovanie údajov sme použili technológiu JavaServer Faces, ktorá je v porovnaní s JavaServer Pages oveľa jednoduchšia, keďže použitie skriptov by bolo veľmi komplikované. Z aplikačných serverov bol použitý JBoss, keďže ide o open-source riešenie a implementuje plne platformu Java EE narozdiel od Tomcatu, ktorý predstavuje len Java EE servlet kontajner. Aplikácia bola rozdelená na 2 časti rozhranie, web service + enterprise bean-a. Pre bezpečnosť bol zvolený open-source framework Seam, konkrétne sme z neho použili moduly Seam Security a Seam Faces. Do úvahy pripadol aj framework OWASP, keďže je Seam súčasťou platformy JBoss, tak sme sa rozhodli pre Seam. Pre implementáciu Web Service bola použitá API JAX-WS, ktoré je oveľa jednoduchšie od JAX-RS. V poslednom ohľade sa uvažovalo o možnosti pravidelné obnovovania obsahu tabuľky úloh, ktorá zobrazuje výsledok spracovania úlohy. Pre túto možnosť bola uvažovaná možnosť použitia Ajaxu. Samotné použitie Ajaxu v JSF aplikácií zbytočne komplikované, preto bola zvolená možnosť použitia Rich Faces, ktorý podporuje Ajax komponenty a poskytuje jednoduchú integráciu s JSF aplikáciou.

## Twitter Bootstrap

Twitter Bootstrap je veľmi jednoduchý a voľne dostupný súbor nástrojov pre vytváranie moderného webu a webových aplikácií.[?] 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 používa-



teľského rozhrania vo webovej aplikácii a nerozhoduje , či to je napríklad použivateľské rozhranie v administrácii back-endových alebo front-endových aplikácií. Tak isto podporuje zobrazenie aj na mobilných telefónoch.

Podrobné vysvetlenie jednotlivých komponent nájdete na nasledujúcej adrese <http://getbootstrap.com/>, rovnako aj s príkladmi použitia. Výhodou a dôvodom použitia tohto frameworku je ľahká integrácia do webovej aplikácie, rovnako aj open source charakter a v poslednom rade množstvo príkladom použitia. V poslednom rade bolo použité rozšírenie Font Awesome, ktoré obsahuje framework, ktorý obsahuje ikonové fonty určené pre twitter bootstrap.

## 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. Jej zásadnou a už spomenutou výhodou je implementácie množstvo ajax komponent, ktoré sa jednoducho používajú a integrujú do JSF aplikácie. Použitie tohto frameworku je možné prostredníctvom maven-u.

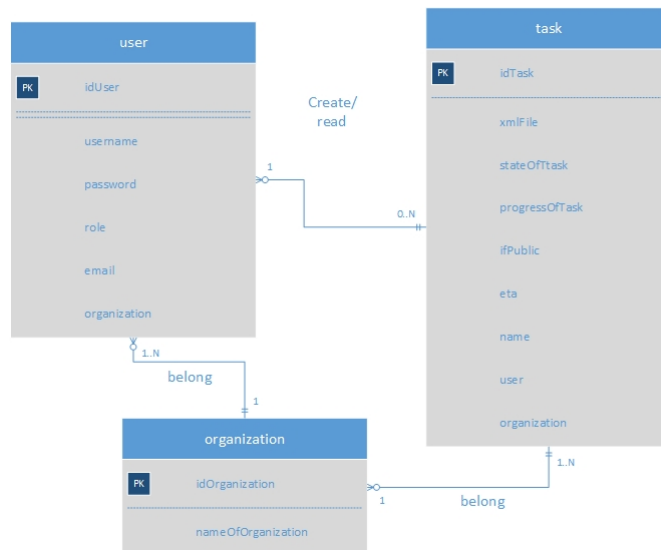
## 5.3 Návrh aplikácie

Aplikácia bola rozdelená na 2 časti. Jednou časťou je samotné grafické užívateľské vytvorené technológiou JSF skombinované s rôznymi frameworkami. Táto aplikácia komunikuje s databázou prostredníctvom JPA s databázou technológiou MySQL. Rovnako zobrazuje informácie úlohách a ďalšie informácie podľa užívateľskej role, rovnako spracováva vstupy zadané užívateľov. Spustenie a zastevenie úloh je vykonáva prostredníctvom Message-driven beany. Signál daný message-driven beany je daný cez web service, ktorá dostane požiadavku od JSF aplikácie. Tá následné predá požiadavok message-driven beane. Požiadavky sa radia do activemsq fronty. Z tejto fronty sú postupne odoberané požiadavky message-driven beanami, ktorá spustia plánovanie(výpočet úlohy). Do fronty sú umiestňované ID úlohy, ktoré sa majú spustiť. Výpočet potom prebieha vytiahnutím xml súboru a spustenie výpočtu, pričom sa priebežne ukladajú informácie o skončení úlohy, rovnako aj informácie percentuálnom pokroku úlohy. Po skončení úlohy sa zmení stav úlohy a uložia sa informácie o dokončení úlohy. JSF aplikácia každých 4 sekúnd aktualizuje stav úloh z databáz a zobrazuje ich užívateľovi v prehliadači. Úlohu je možné aj pozastaviť.

### 5.3.1 Návrh modelu databáze

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

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



Obrázek 5.2: ER diagram

reprezentuje organizáciu. Výsledný návrh odpovedá skutočnosti, že každý užívateľ musí byť súčasťou organizácie, rovnako môže mať vytvorené 0 až N úloh. Taktiež pre zjednodušenie je každá 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 s Camel notáciou. Poďme sa pozrieť bližšie na jednotlivé entity. Entitná množina organization obsahuje 2 položky jednou z nich je primárny kľúč a ďalšou názov organizácie podľa, ktorej sú zaraďovaný jednotliví užívatelia. Ďalej prejdime k entitnej množine user. Táto entita má rovnako primárny kľúč. Ďalej obsahuje položku pre užívateľské meno(username), heslo(password), email, užívateľskú rolu(role) a cudzí kľúč 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 dām), stav úloh(stateOfTask, ktorý reprezentuje rôzne stavy úlohy), ktorý si podrobnejšie rozobereme. Úloha sa môže nachádzať v jednom z nasledujúcich stavov:

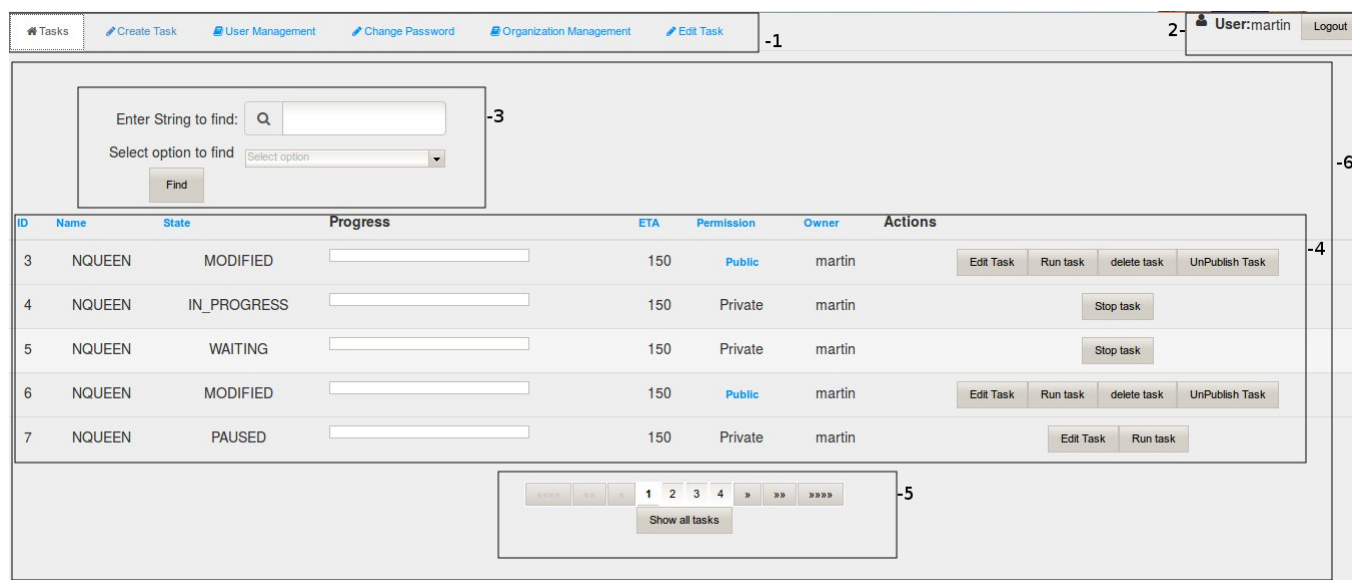
- NEW - úloha bola vytvorená
- MODIFIED - xml súbor bol modifikovaný
- WAITING - úloha čaká na spracovanie
- IN\_PROGRESS - práve prebieha výpočet
- PAUSED - úloha je pozastavená

- COMPLETE - úloha je dokončená

Entitná množina task ďalej obsahuje položku, ktorá percentuálne hodnotí stav výpočtu úlohy(progressOfTask), čas do skončenia výpočtu úlohy(eta), nastavenie úlohy na privátnu alebo verejnú(ifPublic), názov úlohy(name) a cudzie kľúče 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.

### 5.3.2 Návrh užívateľského rozhrania

Výsledné rozhranie kladie dôraz na jednoduchosť a prehľadnosť zobrazených úloh. Z tohto dôvodu boli implementované mechanizmy vyhľadávania úloh, organizácií a užívateľov. Rovnako možnosti lexikografického triedenia. Po prihlásení do systému Jednotlivé možnosti sú následne zakomponované do záložiek, v ktorých je sprístupnená príslušná funkčnosť. Výsledné rozhranie je prenositeľné aj na mobilné zariadenie, čo je spôsobené použitím frameworku Twitter Bootstrap. Je ešte lepšiu prívetivosť bolo použité rozšírenie Font Awesome. Užívateľské rozhranie je popísané na nasledujúcom obrázku: Na obrázku č.??



Obrázek 5.3: Návrh užívateľského rozhrania

môžeme vidieť návrh užívateľského rozhrania. Rozhranie je rozdelené do 6 častí, ktoré môžeme rozoznať na obrázku číslami od 1 do 6, ktoré sú aj ohraničené. Celé rozhranie môžeme rozdeliť do nasledujúcich častí:

- Oblasť č.1 predstavuje navigačné menu, kde sú jednotlivé akcie rozdelené do záložiek podľa ich názvu. Pre kliknutie na príslušnú záložku sa zmení aj obsah na stránke.

- Oblasť č.2 obsahuje informáciu o prihlásenom užívateľovi , rovnako obsahuje aj tlačidlo „Logout“, prostredníctvom ktorého sa môže užívateľ z aplikácie odhlásiť
- Oblasť č.6 predstavuje funkčnú oblasť. Táto oblasť je špecifická pre každú záložku, ktorá reprezentuje jej obsah. V tej oblasti sú umiestnené typicky obsahy databázových tabuliek, nástroje na vyhľadávanie, rôzne akcie, ktoré je možné vykonávať s dátami, rovnako aj možnosti na vytváranie entít
- Oblasť č.3 predstavuje jednu z funkčných možností. Jedná sa o vyhľadávanie, ktoré je zložené zo vstupného prvku, do ktorého zadame vyhľadávaný reťazec a druhá časť predstavuje menu, z ktorého zvolíme stĺpec na vyhľadávanie. Následne je možnosť realizovať tlačidlom Find, ktoré prekreslí obsah tabulky nižšie a naplní ju nájdenými výsledkami.
- Oblasť č.4 predstavuje tabulky, ktorá je dynamicky obnovená a reaguje na asynchronné ukladanie dát z web service, ktoré sa dynamicky obnovujú každé 4 sekundy. Tabuľka je rozdelená do stĺpcov. Názvy stĺpcov, ktoré sú označené modrou farbou sú zároveň odkazy, na ktoré je možné kliknúť. Po kliknutí na daný odkaz dôjde k lexicografickému zoradeniu obsahu tabuľky podľa daného stĺpca striedavo vzostupne alebo zostupne. Rád by som upozornil na stĺpec progress, ktorý pre každú úlohu zobrazuje stav spracovania úlohy. Rovnako musím zdôrazniť stĺpec Permission, ktorý zobrazuje, či je úloha verejná alebo privátna. Pokiaľ je úloha verejná(Public), tak je tento odkaz zobrazený modrou farbou, čo znamená, že je odkaz preto je možné naň ho kliknúť. Po kliknutí sa zobrazí stránka s informáciami o názve úlohy a obsahuje výsledného xml súboru. Tento odkaz je možné následne ľubovoľne preposlať a pristupovať k nemu. V poslednom rade treba zdôrazniť stĺpec „Actions“, ktorý je najdôležitejší pre každú úlohu povoľuje sadu akcií. Jednotlivé akcie sú reprezentované tlačidlami, pritom odrážajú aktuálny stav spracovania úlohy spolu s ďalšími informáciami o úlohe.
- Oblasť č.5 predstavuje komponentu na stránkovanie, aby pri rozsiahlom obsahu sa nezväčšoval neúmerne veľkosť stránky.

Zvyšné návrhy rozhrania je možné dohľadať v prílohe.

## 5.4 Implementácie

Aplikácie bola rozložená do viacerých tried podľa zodpovednosti daných komponent. Pri implementácii bolo použité JBoss Developer studio 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 príslušnými údajmi.

### 5.4.1 Rozdelenie aplikácie

Aplikácia bola rozdelená na 2 časti:

- JSF aplikácia s užívateľským rozhraním, ktorá zobrazuje výsledky, umožňuje spravovanie databáze, rovnako aj prístupovanie k službám web service, je pritom rozdelená na hlavný modul s celou funkčnosťou a modul s entitami, ktoré zdieľa s PlannerService - optaplanner.controller
- Web service + EJB aplikácia, ktorá poskytuje web service koncový bod spolu s metódami na spustenie a pozastavenie úlohy. Spustenie úlohy volá Message-driven Beanu, ktorá vykonáva spravenie(plánovanie) úlohy, využíva pritom modul Enties, ktorý obsahuje entitné triedy - PlannerService

### 5.4.2 JSF aplikácia

JSF aplikácia bola postavená na technológií JavaServer Faces v kombinácii s frameworkami Rich Faces, Twitter Bootstrap a Font Awesome. Aplikácia zabezpečuje prihlasovanie užívateľov v kombinácii s bezpečnosťou, ktorá je implementovaná s frameworkom Seam. Je rozdelená na .xhtml stránky podľa užívateľskej role. Na každej stránke pritom môže nájsť komponenty, ktoré dovoľujú užívateľovi príslušné akcie. Na .xhtml stránkach sa následne zobrazuje aj obsah spracovania, ktorý už bol prezentovaný tak aj entity užívateľov a organizácií spolu s ich spravovaním. Tieto komunikujú prostredníctvom expression language s managed beanami na pozadí, ktoré rovnako vykonávajú aj validáciu jednotlivých komponent(od nepovolených hodnôt, až po prázdne vstupné polia). Stlačenie príslušného tlačidla spôsobí zavolanie metódy z managed beany a vykonanie akcie, pričom jej obsah môže byť ale nemusí byť zobrazený na .xhtml stránku.

### 5.4.3 Prihlasovanie

Prihlasovanie je realizované prostredníctvom frameworku Seam, z ktorého sme využili moduly Seam Faces a Seam Security. Pre každú užívateľskú rolu bolo vytvorené rozhranie a metódy, ktoré overovali identity užívateľa. Pre prístup musela mať užívateľ danú užívateľskú rolu, pričom prístup bol implementovaný enum-om, v ktorom každej role, ktorá bola reprezentovaná anotáciou(@Administrator,@Reader, @Planner) boli povolené len im prislúchajúce stránky a bolo deklarované, čo sa má vykonať pri po pokuse o nepovolený prístup(presmerovanie na prihlasovací formulár). Prihlasovanie rovnako obsahovalo validáciu a informovalo užívateľa o nevalidnom hesle, alebo mene. Po úspešnom prihlásení bola do životného cyklu vložená Identita a užívateľ bol presmerovaný na jemu prislúchajúcu stránku. Vloženie identity malú tú výhodu, že bolo možné z managed beany zistiť aký je prihlásený užívateľ. Prihlasovanie overovalo údaje z databáze, pričom pre overenie hesla používalo triedy ShaEncoder, ktorá obsahuje hash funkcia prostredníctvom, ktorej sú zabezpečené všetky heslá v databáze.

### 5.4.4 Logika JSF aplikácie

V prvom rade trebalo správne nakonfigurovať súbor persistence.xml, aby ukazoval na nami definovaný datasource v rámci aplikačného servera. To umožňuje prístupovanie k dátam v databáze. Celá logika aplikácie bola sústredná bola sústredená do managead

bean, pričom 1 managed beana podľa prihláseného užívateľa (napr. užívateľská rola administrátor na stránke `Administrátor.xhtml` bola spravovaná managed beanov `AdministratorBean`). Tie obsahovali metódy a vlastnosti, ktoré bola zobrazované/prevzaté z komponent na `.xhtml` stránke. Vlastnosti museli spĺňať princíp POJO. Metódy následne volali podľa potreby databázové operácie, ktoré boli sústredené v balíku `databaseOP` v triede `Operation`. Tá obsahovala metódy na mazanie, update, vytváranie entít. Predpisy entitných tried sa nachádzali v module `Entites`. V balíku `service` sa nachádzali triedy, ktoré umožňujú volanie metód web service.

#### 5.4.5 Implementácia rozhrania

Pre implementáciu rozhrania som využil technológiu `xhtml` stránok. Pre každú užívateľskú rolu som vytvoril `xhtml` stránku identitickú s názvom užívateľskej role. Pre prihlasovanie bola použitá `Login.xhtml` stránka. Na `Login.xhtml` boli umiestnené komponenty na zadanie užívateľského mena a hesla vrátane skrytých validačných komponent. Pre implementáciu `xhtml` pre užívateľské role sa zameriam na užívateľskú rolu Administrátor, keďže rola Plánovač a Čitateľ prevzali všetku implementáciu a komponenty práve od Administrátor, ale len v obmedzenom množstve, teda komponenty vrátane akcií, ktoré mohli vykonávať. XHTML stránka sa skladá v hornej časti z menu, ktoré je implementované ako záložky prostredníctvom Twitter Bootstrap-u. V pravej hornej časti sa nachádza informácia o prihlásenom užívateľovi vrátane tlačidla na odhlásenie. Pri kliknutí na záložky sa zobrazí obsah, ktorý odpovedá názvu záložku. Záložky „user management, task, organization management“ obsahujú komponenty `h:datatable` z knižnice JSF pre zobrazenie dát. Tieto dáta sú pravidelné obnovované komponentov z knižnice Rich Faces `a4j:poll`, ktorý využíva Ajax pre obnovenie obsahu. Každá z tých záložiek obsahuje pole pre vyhľadávanie pričom je možné zvoliť podľa, ktorého stĺpca sa bude vyhľadávať. Výsledky sa zobrazia do tabuľky (`h:datatable`) pričom zobrazené položky budú odpovedať nájdeným výsledkom. Pri každej položke v tabuľke je možné vykonávať isté akcie ako je vymazať danú entitu, po prípade ju editovať, alebo vykonávať množstvo iných akcií. Akcie pritom reflektujú individuálny stav danej entity. Pri každej z tých záložiek okrem task (ktorú v zápati rozoberem) je možné entity aj vytvárať. Vytváranie je veľmi jednoduché, keď užívateľ vyplní všetky polia, ktoré musí mať daná entita. Každú tabuľku je možné aj radiť. Radenie prebieha kliknutím na názov stĺpca tabuľky, pričom daný stĺpec implementuje funkciu radenia pre daný stĺpec len v prípade, že názov je vyznačený modrou farbou. Vytváranie úloh (taskov) je zaradené do samostatnej záložky kvôli lepšej prehľadnosti. Užívateľ vyplní meno a prostredníctvom komponenty na nahrávanie súboru z knižnice Rich Faces nahrá obsah do databáze. Ďalej rozoberem záložku change password, ktorá umožňuje si pre daného užívateľa zmeniť heslo, vyplní pritom heslo a potvrdenie heslo a heslo sa zmení. Nakoniec rozoberem záložku edittask, táto záložka je pri bežnom prehliadaní prázdna je to spravené kvôli bezpečnosti. Táto záložka sa aktivuje editovaním úlohy v záložke task, ktorá nás prepne do záložky edittask, v ktorej sa už aktivuje obsah a užívateľ vyplní názov úlohy, vlastníka úlohy a nakoniec edituje xml súbor úlohy. Potvrdením sa vytvorí úloha so stavom "MODIFIED".

### 5.4.6 Web Service, EJB aplikácia

Pri implementácii web service bola využitá už čiastočne implementovaná Web service, do ktorej boli následne doplnená funkčnosť spustenia úlohy a pozastavenia úlohy. Jej časť spustenia výpočtu už nebola cieľom bakalárskej práce a bola teda implementovaná mojim vedúcim Martinom Večeřom. Táto aplikácia je pomenovaná PlannerService a obsahuje ako som spomínal Big Web servica s 2 metódami. Tieto metódy sú metóda runTask s jedným parametrom a to ID úlohy, ktorá sa má spustiť. Pričom dôjde k vytvoreniu spojenia s Message-driven bean, ktorá vykonáva výpočet. Najprv sa do fronty zaradí ID úlohy, ktorá sa má spustiť. Následne si Message-driven beana získa z fronty ID úlohy a následne spustí výpočet. Tieto beany sa kvôli rozloženiu záťaže môžu nachádzať na viacerých serveroch. Priebežne pritom aktualizuje stav spracovania úlohy. Priebežne pritom ukladajú údaje o stavu úlohy, času do ukončenia úlohy a percentuálnom ohodnotení úlohy. Po ukončení úlohy je uložené najlepšie možné riešenie do databázy. Na skončení spracovania nastaví príslušný stav úlohy a skončí. Web service obsahuje aj metódu na pozastavenie výpočtu s 1 parametrom a to je ID úlohy, ktoré sa má pozastaviť. Tá následne zavolá Message-driven bean-u, ktorá si z fronty získa úlohu a pozastaví jej výpočet. Tie si úlohu vyberú z actiemsq fronty a spustia výpočet.

## 5.5 Testovanie

Testovanie prebiehalo na servere JBoss AS 7.1.1 Final najprv prostredníctvom jednoduchých JUnit testov, ktoré malo overiť komplikovanú funkčnosť metód. Následne sa pre overenie funkčnosti databázy použil framework Arquillian, ktorý umožňuje nasadenie tried priamo do Java EE kontajneru, čo zjednodušuje testovanie. Prostredníctvom tohto frameworku sa testovala celková funkčnosť aplikácie. Jednoduchšie časti boli otestované pomocou JUnit testov. Postupným budovaním aplikácie sa pristupovalo k testovaniu navrhnutých častí. JUnit boli postupne skonštruované pre jednoduchšie metódy, ako je overenie funkčnosti vyhľadávania entít, mazanie entít, pridanie entít do zoznamu úloh. Pomocou arquillian-u bolo následne otestované prihlasovanie, databázové operácie, rovnako bola otestovaná bezpečnosť aplikácie.

V ďalšej časti prebiehalo testovanie medzi konkrétnymi užívateľmi. Užívatelia testovali aplikáciu a hľadali buggy, ktoré neodhalilo predošlé testovanie. Rovnako overovali, či boli splnené formálne požiadavky. Skupine užívateľov bol predložený odkaz na nasadenú aplikáciu a prihlasovacie údaje. Užívatelia mohli následne sa prihlasovať pod rôznymi užívateľskými rolami, tí následne testovali vytváranie užívateľov, organizácií, úloh. Následne mohli sledovať stav spracovania plánovacích úloh. Aplikáciu otestovali pod 2 prehliadačmi a to Google Chrome vo verzii 34.0 a Mozilla Firefox verzie 28.0. Bol použitý operačný systém linux 3.13.0-24-generic s operačným systémom Kubuntu 14.04. Aplikácia sa správala pod oboma rovnako a korektne. Po odhalení chýb boli chyby ohlásené a odstránené a aplikácia bola následne opäť nasadená. Tento postup sa opakoval až dokým neboli odhalené všetky chyby.

## 5.6 Vyhodnotenie aplikácie

Po testovacej fáze nasledovala fáza vyhodnotenia aplikácie. Cieľovej skupine bol po opravení chýb aplikácie predložený dotazník, do ktoréh výplňami rôzne informácie, kde dávali spätnú väzbu, chyby v návrhu, rovnako aj v intuitívnosti ovládania. Cieľovou skupinou bolo 7 eventuálnych používateľov tejto aplikácie. Z dotazníka nám vyplynuli nasledujúce názory a pohľady na aplikáciu.



## Kapitola 6

### Záver

Plánovanie s ním spojené problémy narážame v bežnom živote čoraz častejšia. Ešte väčšie problémy tohto typu majú organizácie, ktoré musia dennodenne riešiť ako naplánovať efektívnu prácu svojich zamestnancov, ako správne komunikovať so zákazníkom a mnoho iných problémov. Riešenie klasickým prístupom a to využitím ľudskými zdrojmi je časovo neefektívne, rovnako treba brať do úvahy ľudský faktor. Preto vzniklo riešenie, ktoré odbremeňuje organizácie od riešenia komplikovaných plánovacích úloh. Taký software je šírený pod licenciou open-source a nazýva sa Optaplanner. Tento systém je následne možné využívať pre akúkoľvek oblasť plánovania, aká len nás napadne. Jediné obmedzenie tohto systému sú použité plánovacie algoritmy kombinovaný s rôznymi heuristikami. Užívateľ je schopný definovať definíciu problému, pričom sa môžeme inšpirovať verejne dostupnými príkladmi, vytvoriť si pravidlá a nechať systém nech nájde optimálne riešenie pre daný problém. Vytvorená aplikácia predstavuje jedným zo spôsobov ako daný systém využiť pre plánovanie. Aplikácia je intuitívna, formálne spĺňa požiadavky, rovnako sú predstavené možnosti rozšírenia rozhrania a urobenie tohto rozhrania oveľa užívateľsky prívetivejšie. Rovnako ukazuje akým spôsobom bol systém navrhnutý z implementačného hľadiska, sú vysvetlené technológie potrebné pre implementáciu so zreteľom na výhody použitia. Pre systém bol použitý aplikačný server JBoss, ktorý predstavoval medzi dostupnými riešeniami najlepší java EE kontajner. Pre lepší návrh by mohla byť aplikácia rozšírená na použitie ich iných plánovacích úloh, rovnako môže byť užívateľské rozhranie rozdelené do viacerých samostatných sekcií kvôli lepšej prehľadnosti. V poslednom rade kvôli lepšej pochopiteľnosti aplikácie by mohla byť JSF aplikácia rozdelená do viacerých balíkov.

# Příloha A

## Inštalácia

V tejto kapitole by som Vád objasnil postup inštalácie aplikácie. V prvom rade uviediem potrebné prostriedky pre beh aplikácie. Pre správny beh aplikácie potrebujeme nasledovné prostriedky:

- JBoss aplikačný server najmenej vo verzii 7.1.1.Final. Je možné ho zdarma stiahnuť z [www.jboss.org](http://www.jboss.org)
- Nástroj Maven, ktorý je možné stiahnuť vo verzii 3.2.1 z [www.maven.apache.org](http://www.maven.apache.org), kde sa nachádza aj manuál na inštaláciu
- MySQL Connector/J minimálne vo verzii 5.0.8, ktorý môžete stiahnuť z <https://dev.mysql.com/downloads/connector/j/>
- MySQL databázový server najmenej vo verzii 5.5.37
- Webový prehliadač Mozilla Firefox najmenej vo verzii 29.0 alebo Google Chrome najmenej vo verzii 34.0

V prvom rade je potrebné nainštalovať MySQL server nakonfigurovať databázu s názvom „optaplanner“ s užívateľským menom „root“ a heslom „root“. Následne je potrebné rozbaľiť JBoss aplikačný server na súborový systém. V ďalšom kroku nastaví náš aplikačný server pre správne použitie MySQL databázy. V tomto kroku je potrebné nakonfigurovať súbor *standalone.xml* v adresári `JBOSS_HOME/standalone/configuration/`. Tam správne nastavíme datasource(zdroj databázových dát pre našu aplikáciu)(môžeme použiť online dokumentáciu <https://docs.jboss.org/author/display/AS71/DataSource+configuration>, kde je celý postup zdokumentovaný). Následne treba stiahnutý MySQL Connector/J skopírovať do adresára `JBOSS_HOME/standalone/deployments`. Následne treba vytvoriť potrebnú databázu to a naplniť ju dátami :

- zadaním príkazu `mysql -u root -p optaplanner < cesta/k/suboru/create.sql`(bude od nás vyžadované heslo root, ktoré zadáme)
- zadaním príkazu `mysql -u root -p optaplanner < cesta/k/suboru/insert.sql`(bude od nás vyžadované heslo root, ktoré zadáme)

Aplikáciu môžeme spustiť nasledovne:

- Skopírujeme maven projekt optaplanner.controller do adresára JBOSS\_HOME/standalone/deployments
- Skopírujeme maven projekt PlannerService do adresára JBOSS\_HOME/standalone/deployments
- Prejdeme do zložky JBOSS\_HOME/standalone/bin a spustíme skript *standalone.sh*

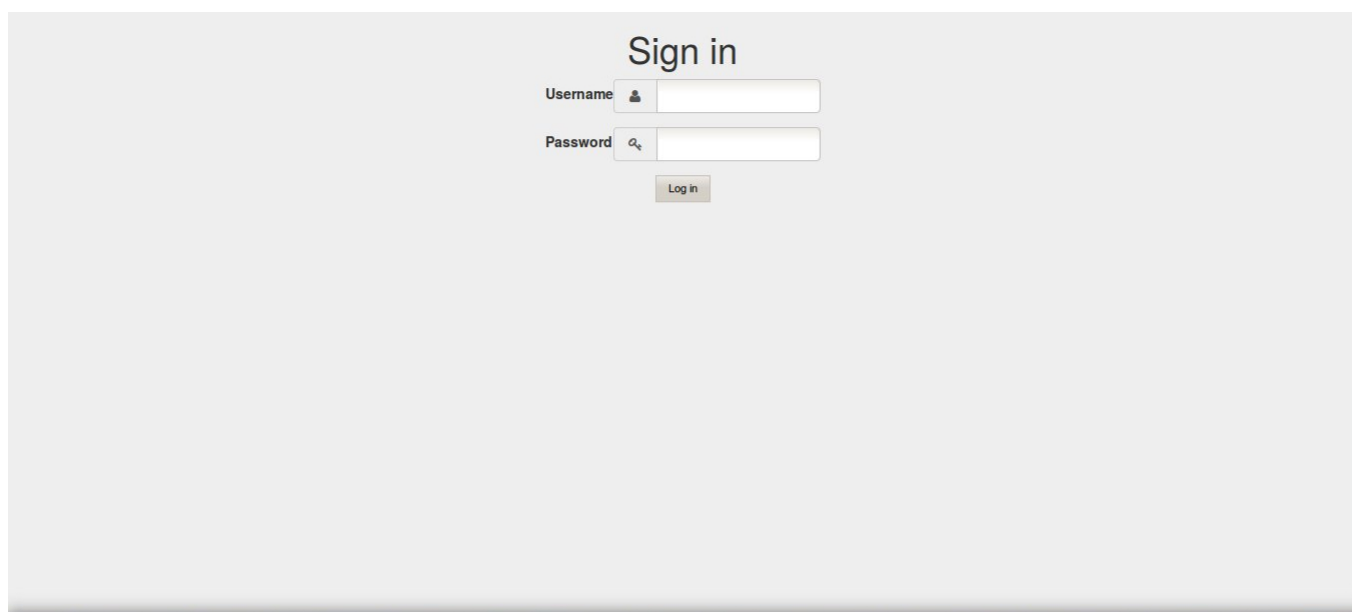
Aplikáciu je možné spustiť aj pomocou nástroju maven a to nasledovne:

- Nastavíme sa na projekt optaplanner.controller a zadáme príkaz mvn package
- Následne zadáme príkaz mvn compile
- A nakoniec príkaz mvn jboss as:deploy

Postup uvedený vyššie vykonáme aj pre projekt Planner.Service. Aplikáciu spustíme zadáním príkazu „<http://localhost:8080/optaplanner.controller/>“ do webového prehliadaču.

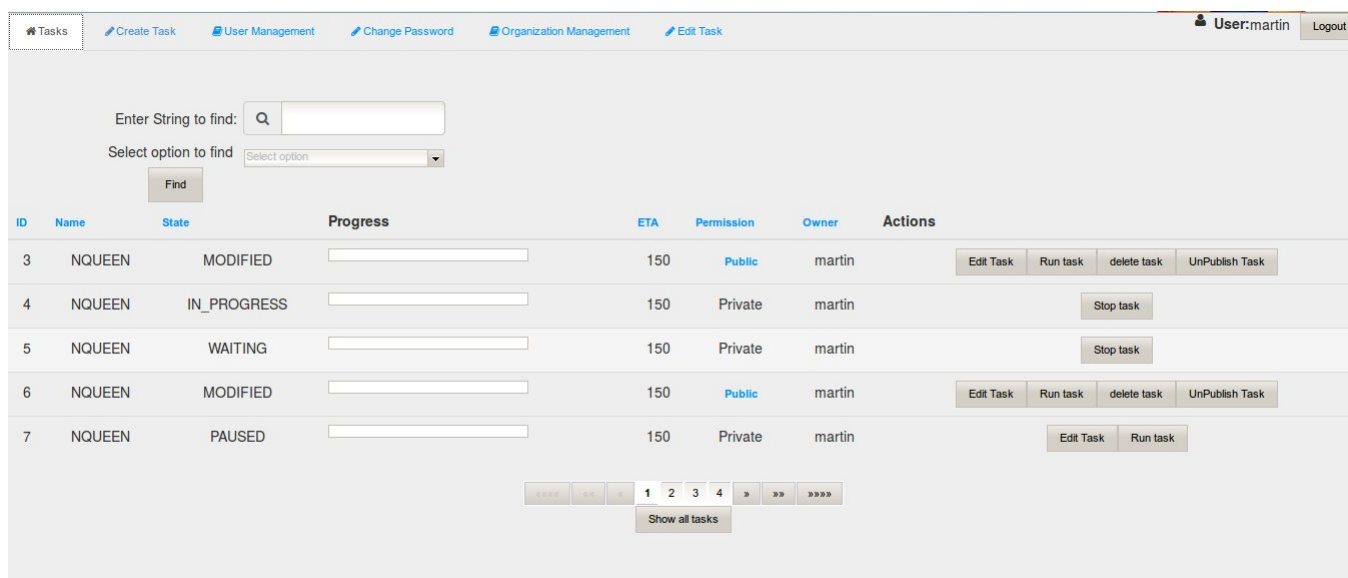
## Příloha B

# Uživatelské rozhraní

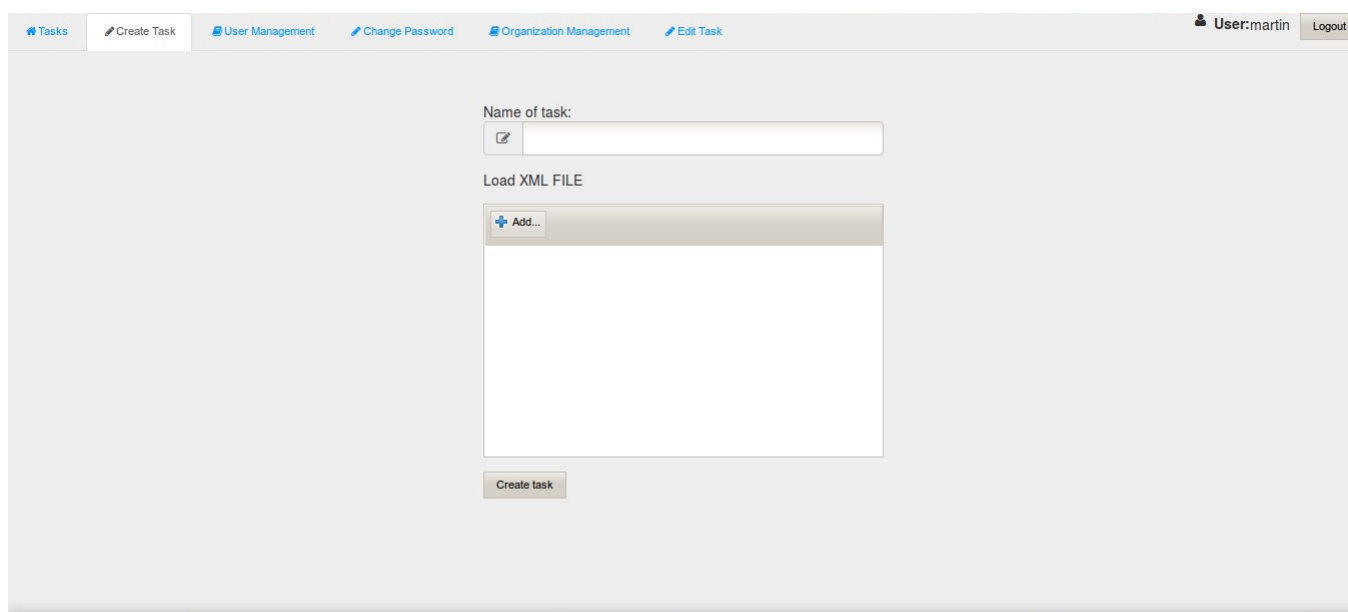


A screenshot of a web application's sign-in interface. The background is a light gray. In the upper right quadrant, the text "Sign in" is displayed in a dark gray, sans-serif font. Below this text, there are two input fields. The first field is labeled "Username" and has a small icon of a person inside a square box to its left. The second field is labeled "Password" and has a small icon of a key inside a square box to its left. Both input fields are white with a thin gray border. Below the password field, there is a small, rectangular button with the text "Log in" in a dark gray font.

Obrázek B.1: Prihlasovacia obrazovka



Obrázek B.2: Zobrazenie úloh spolu s akciami



Obrázek B.3: Nahrávanie novej úlohy

Tasks Create Task **User Management** Change Password Organization Management Edit Task User:martin Logout

Username:

Password:

Re-type password:

Email:

Organization:

Role:

Create User

Enter String to find:

Select option to find:

Find

Username	Email	Role	Organization	Action
martin	martin@martin.cz	Administrator	Red Hat	<input type="button" value="Edit User"/> <input type="button" value="Delete User"/> <input type="button" value="Change Password"/>
david	david@david.cz	Reader	IBM	<input type="button" value="Edit User"/> <input type="button" value="Delete User"/> <input type="button" value="Change Password"/>
peter	peter@peter.cz	Planner	Oracle	<input type="button" value="Edit User"/> <input type="button" value="Delete User"/> <input type="button" value="Change Password"/>

Obrázek B.4: Spravovanie užívateľov

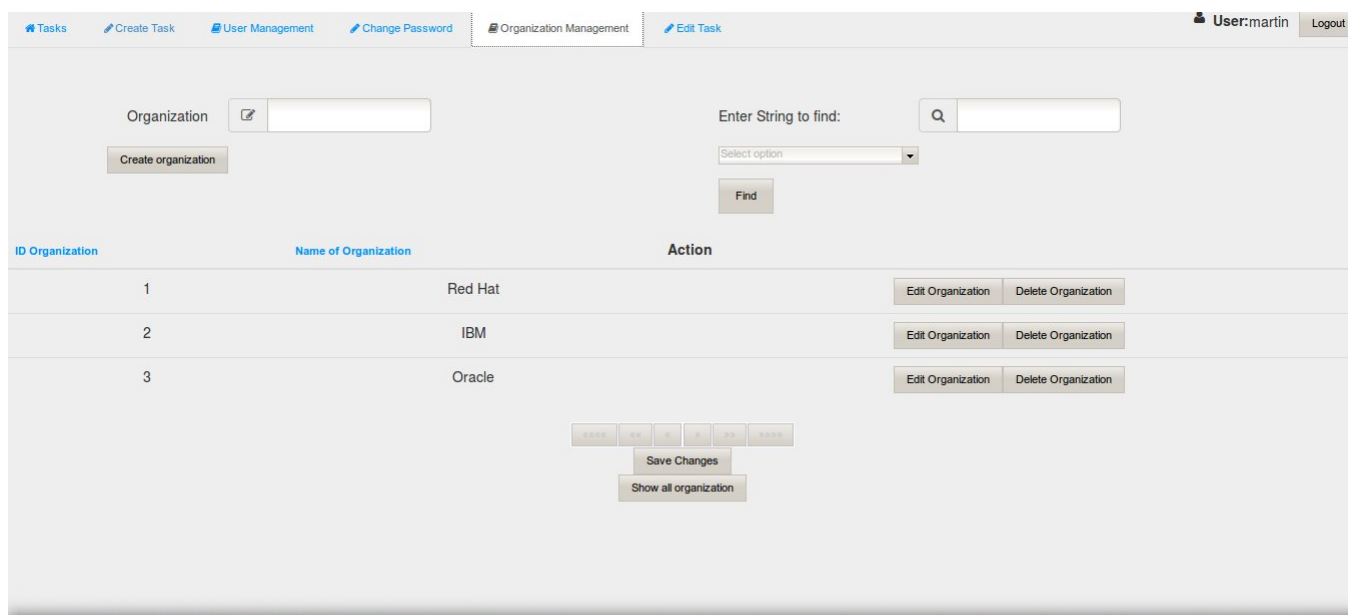
Tasks Create Task User Management **Change Password** Organization Management Edit Task User:martin Logout

Password :

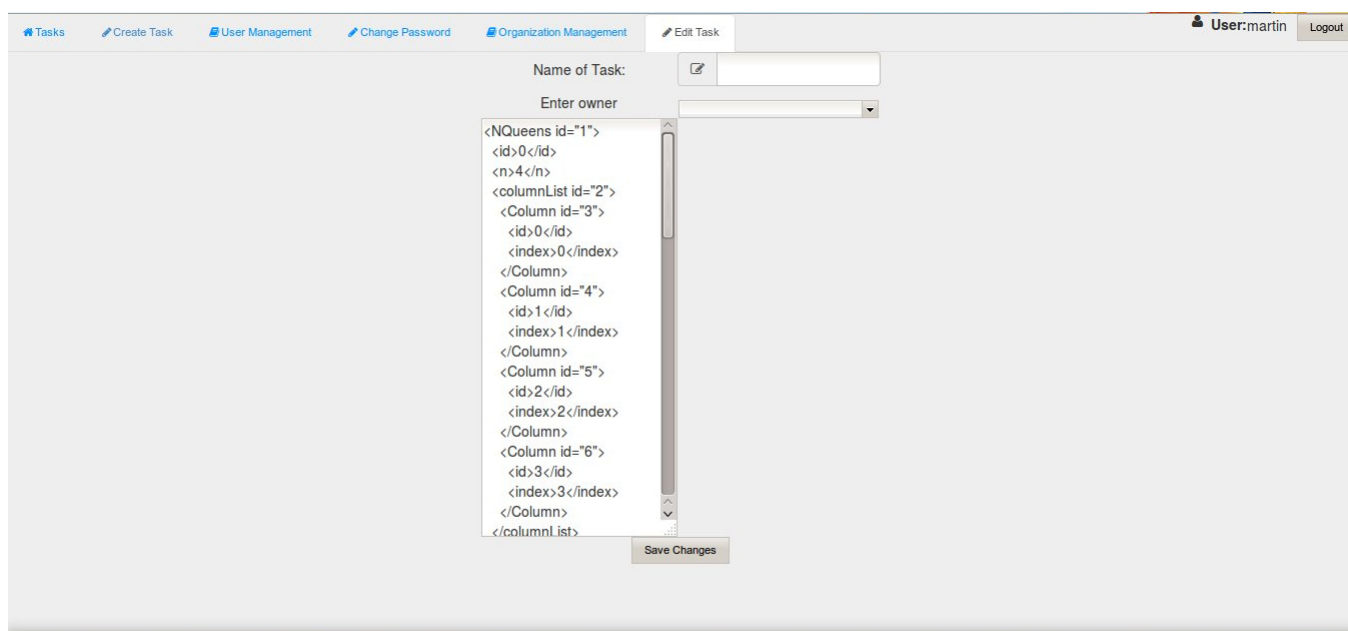
Confirm password :

Change password

Obrázek B.5: Zmena hesla



Obrázek B.6: Spravovanie organizácií



Obrázek B.7: Editovanie úloh podľa predlohy

# Příloha C

## Dotazník

### C.1 Obsah dotazníka

**Grafické uživatelské rozhranie pre systém Optaplanner**

Som študentom tretieho ročníka FIT VUT v Bme. Mojou bakalárskou prácou je tvorba užívateľského rozhrania pre Systém monitorovania plánovacích úloh nazývaný Optaplanner. Ide o aplikáciu, ktorá dokáže podľa príslušnej užívateľskej role užívateľa spracované úlohy v rámci systému Optaplanner, spravovať užívateľov, spravovať organizácie a úlohy. Tento software je optimalizovaný na prístup z rôznych zariadení, či sa už jedná o mobilné telefón ale pc, notebook.

Všetky potřebné informace sú uložené na 1 mieste v databázi.

V rámci svojej aplikácie môžeme vykonávať nasledujúce činnosti:

- vytvárať, mazať, editovať, publikovať, odpublikovať a zobrazovať stav spracovania úloh
- vytvárať, mazať, editovať a zobrazovať užívateľov
- zmeniť si heslo
- vytvárať, mazať, editovať, zobrazovať organizácie

Chceli by sme Vás požiadať o vyplnenie jednoduchého dotazníku, na základe ktorého môžeme svoju prácu vylepšiť.

**Koľko máte rokov ?**

☐ Menej ako 15 rokov

☐ 15 až 25 rokov

☐ 25 až 40 rokov

☐ Nechcem odpovedať

**Stretli ste sa už niekedy s frameworkom Optaplanner ?**

☐ Áno

☐ Nie

Obrázek C.1: Všeobecné informácie

### C.2 Grafické vyhodnotenie



**Vaše preferencie**

**Aké informácie by ste chceli zobrazovať pri správe užívateľov ?**

☐ Užívateľské meno

☐ Posledný čas prístupu do systému

☐ Užívateľské heslo

☐ Other:

**Aké informácie by ste chceli zobrazit' pri správe organizácií ?**

☐ Názov organizácie

☐ Jednoznačný identifikátor organizácie

☐ Čas vytvorenia organizácie

☐ Užívateľ, ktorý vytvoril organizáciu

☐ Other:

**Aké informácie by ste chceli zobrazovať pri úloh, ktorú sú spracované nejaké serverom ?**

☐ Názov úlohy

☐ Jednoznačný identifikátor úlohy

☐ Čas do konca spracovania

☐ Percentuálne hodnotenie spracovania

☐ Kedy bola úloha spustená

☐ Stav úlohy

☐ Definíčný súbor danej úlohy

☐ Other:

Obrázek C.2: Preferencie

**Aký systém triedenia úloh by Vám najviac vyhovoval ?**

☐ Všetky úlohy na jednom mieste a možnosť lexikografického zoradenia

☐ Zobrazenie častí úloh podľa určitého kritéria(napr stavu,...)

☐ Rozdelenie do kategórií podľa stavu spracovania

☐ Triedenie podľa numericky podľa ich jednoznačného číselného identifikátora

☐ Other:

**Aké dôležité sú pre Vás nasledujúce funkcie ?**

	Nedôležité				Dôležité
Možnosť zoradovať úlohy podľa rôznych stĺpcov	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Možnosť vyhľadávať úlohy podľa viacerých kritérií	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Možnosť vyhľadávať úlohy	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Možnosť pristupovať k systému z tabletu	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Možnosť vyhľadávať organizácie	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Možnosť kategorizovať zobrazované úlohy podľa určitého kritéria	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Obrázek C.3: Hodnotenie

**Hodnotenie nášho riešenia**

**Ako hodnotíte systém pridávania nových úloh ?**

	1	2	3	4	5
Veľmi nepraktické - Veľmi praktické	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

**Ako hodnotíte vyhľadávanie úloh, užívateľov, organizácií ?**

	1	2	3	4	5
Veľmi nepraktické - Veľmi praktické	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

**Aký dojem máte s prehliadania úloh, užívateľov, organizácií ?**

	1	2	3	4	5
Veľmi nepraktické - Veľmi praktické	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

**Ktorá funkcionality bola príliš skýta ?**  
 Napíšte, akú funkciu by ste očakávali na inom mieste, viac viditeľnú a podobne.

**Aká funkcionality vám chýbala?**

**Čo by ste spravili určili (rozmiestnenie, funkčnosť, ktorá Vás brzdila) ?**

Obrázek C.4: Závěrečné hodnocení

## Příloha D

# CD so zdrojovými kódy

Priložené CD obsahuje nasledujúce súbory:

- optaplanner.controller - adresár JSF aplikácie obsahujúci potrebné zdrojové kódy a .xhtml stránky vrátane konfiguračných súborov
- PlannerService - adresár s EJB a web service, ktorý obsahuje zdrojové kódy vrátane konfiguračných súborov
- install.txt - súbor s popisom inštalácie
- create.sql - sql súbor s definíciami tabuliek
- insert.sql sql súbor s naplnením dát tabuliek
- bachelor\_thesis.pdf - elektronická verzia textovej Časti bakalarskej práce