

# 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

**Ing. ZDENĚK LETKO, Ph.D.**

BRNO 2014

## Abstrakt

Úkolem této bakalářské práce je dle konkrétních požadavků zadavatele společnosti Red Hat vytvořit systém monitorování stavu plánovacích úloh. Hlavním cílem je pochopit, co je to plánovací problém, jako ho můžeme definovat, uložit do systému, spustit jeho plánování a monitorovat průběh. Systém plánování byl rozdělen na části uživatelského rozhraní vytvořeného pomocí technologií JavaServer Faces, Rich Faces a Twitter Bootstrap, které umožňují nahrávání, spouštění a pozastavení běhu úloh a druhou část reprezentovanou webovou službou v kombinaci s technologií Enterprise JavaBeans, která zpracovává požadavky na spouštění/pozastavení běhu plánování, vykonává je s využitím frameworku OptaPlanner a průběh plánování ukládá do MySQL databáze. Z této databáze jsou uživatelským rozhraním získávány informace o plánovacích problémech a průběžně zobrazovány v uživatelském rozhraní. Pro implementaci obou částí byla použita platforma Java EE 6 a nasazena v Java EE kontejneru JBoss. Systém byl odzkoušen na plánovacím problému N Dam na platformě UNIX uživateli prostřednictvím cloudové služby OpenShift, kteří si zkusili vyřešit problém N Dam pro různá rozmístění.

## Abstract

The object of this thesis is based on specific company requirements Red Hat to create planning task monitoring system. The main objective is to understand what a planning problem is, as it can be defined, stored in the system, start and monitor its progress. Planning system has been divided into parts of the user interface created with technology JavaServer Faces, Rich Faces and Twitter Bootstrap that allows recording, starting and suspension of the tasks and the second part represented by the Web service in combination with Enterprise JavaBeans technology that processes requests for start/pause run of planning, executes them using the framework OptaPlanner and store them in a MySQL database. From this database user interface gain information about planning problems and continuously displayed in the user interface. To implement both parts were used Java EE 6 platform and deployed in Java EE container JBoss. The system was tested on the planning problem N Queen on UNIX based system users through cloud service OpenShift, who tried to solve the problem for different location of Queens.

## Klíčová slova

Java EE 6, JavaServer Faces, Monitorovací systém, Twitter, Bootstrap, OptaPlanner, Webová služba, Enterprise JavaBean, JBoss, Rich Faces, Arquillian, Plánování, MySQL, Plánovací problém

## Keywords

Java EE 6, JavaServer Faces, Monitoring system, Twitter, Bootstrap, OptaPlanner, Web Service, Enterprise JavaBean, JBoss, Rich Faces, Arquillian, Planning, MySQL, Planning problem

## 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. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....

Martin Maga

1. júla 2014

## Poděkování

Veľmi rád by som poďakoval za vedenie mojej bakalárskej práce pánovi Zdeňkovi Letkovi a firme Red Hat, s ktorou som komunikoval prostredníctvom externého konzultanta Martina Večeře, ktorému tiež patrí moja vďaka.

© 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

<b>1</b>	<b>Úvod</b>	<b>3</b>
<b>2</b>	<b>Java Enterprise edition 6</b>	<b>4</b>
2.1	Špecifikácia platformy . . . . .	4
2.2	Trojvrstvový model . . . . .	4
2.2.1	Princíp webových komponent . . . . .	6
2.2.2	JavaServer Faces . . . . .	7
2.3	Twitter Bootstrap . . . . .	8
2.4	Rich Faces . . . . .	8
2.5	Webová služba . . . . .	9
2.5.1	Big webová služba . . . . .	9
2.5.2	RESTful webová služba . . . . .	10
2.6	Java Persistence API . . . . .	10
2.7	Enterprise JavaBeans . . . . .	11
2.7.1	Session Bean . . . . .	12
2.7.2	Message-driven Bean . . . . .	12
2.8	MySQL . . . . .	12
2.9	Seam . . . . .	13
2.10	Testovanie . . . . .	13
2.11	JBoss Aplikačný server . . . . .	14
<b>3</b>	<b>OptaPlanner</b>	<b>15</b>
3.1	Plánovací problém . . . . .	15
3.2	Princíp . . . . .	16
3.3	Konfiguráciu OptaPlanneru . . . . .	18
3.4	Výsledky plánovacieho problému . . . . .	20
<b>4</b>	<b>Aplikácia</b>	<b>21</b>
4.1	Špecifikácia požiadavkov . . . . .	21
4.2	Analýza . . . . .	22
4.3	Návrh aplikácie . . . . .	23
4.3.1	Návrh modelu databáze . . . . .	24
4.3.2	Návrh užívateľského rozhrania . . . . .	26
<b>5</b>	<b>Implementácie</b>	<b>28</b>
5.1	Aplikácie pre užívateľské rozhranie . . . . .	28
5.1.1	Prihlasovanie . . . . .	28
5.1.2	Zabezpečenie . . . . .	29

5.1.3	Komunikácie s PlannerService . . . . .	29
5.1.4	Logika aplikácie . . . . .	30
5.1.5	Implementácia rozhrania . . . . .	30
5.1.6	Publikovanie úloh . . . . .	31
5.1.7	Validácia . . . . .	32
5.2	PlannerService . . . . .	32
5.3	Testovanie . . . . .	33
5.4	Vyhodnotenie aplikácie . . . . .	34
<b>6</b>	<b>Záver</b>	<b>36</b>
<b>A</b>	<b>Inštalácia</b>	<b>39</b>
<b>B</b>	<b>Užívateľské rozhranie</b>	<b>41</b>
<b>C</b>	<b>Dotazník</b>	<b>44</b>
C.1	Obsah dotazníka . . . . .	44
<b>D</b>	<b>CD so zdrojovými kódmi</b>	<b>47</b>

# Kapitola 1

## Úvod

V poslednej dobe sa pre tvorbu rozsiahlych aplikácií, ktoré kladú dôraz na rýchlosť, bezpečnosť a transakčné spracovanie používa trojvrstvový model. Komplexným riešením je platforma Java EE, ktorá zahŕňa množstvo technológií pre prístup k relačným databázam, pre podporu webových služieb, pre vývoj zdieľanej podnikovej logiky, . . . .

Cieľom tejto práce bolo vytvorenie systému monitorovania stavu plánovacích úloh, ktorý bol zadaný spoločnosťou Red Hat. Tento systém je schopný riešiť rozličné plánovacie problémy. Riešenie je realizované frameworkom OptaPlanner, ktorý na základe pravidiel a definičného súboru pre danú úlohu sa pokúsi nájsť najlepšie riešenie, ktorý poskytne ako výsledok. Rovnako je process analýzy a vývoja výsledného systému obsahom tejto správy.

Druhá kapitola 2 sa venuje Java EE platforme spolu s použitými technológiami. Kapitola predstavuje trojvrstvový model, rovnako aj použité technológie. Na záver kapitoly 2.11 je uvedený aplikačný server JBoss.

V tretej kapitole 3 je vysvetlený plánovací framework OptaPlanner. V kapitole sa pojednáva postupne od základov problematiky plánovania až po bližšie vysvetlenie pojmu plánovací problém. V tejto kapitole sa ešte oboznámime s princípom plánovania prostredníctvom tohto frameworku, rovnako aj jeho konfiguráciou.

V štvrtej kapitole 4 je prezentovaná špecifikácia požiadavkov, rovnako ako aj analýza technológií spolu s návrhom užívateľského rozhrania a databázovej schémy. V piatej kapitole 5 je uvedená implementácia výsledného systému. Na záver kapitoly sú uvedené metódy a postupy testovania. V záverečnej kapitole 6 je zhrnutý obsah celej práce, sú zhodnotené jej prínosy a možnosti ďalšieho rozšírenia.

V sekcii príloh nájdeme postup na inštaláciu a spustenie aplikácie rovnako ako aj kompletný prehľad navrhnutého rozhrania.

## Kapitola 2

# Java Enterprise edition 6

Táto kapitola poskytuje prehľad o platforme Java EE 6, rovnako ako o technológiách, ktoré sú súčasťou tejto platformy a sú používané pri implementácii systému monitorovania. Kapitola rovnako predstavuje trojvrstvový model pre tvorbu aplikácií a použité technológie z platformy Java EE.

V závere kapitoly je rozobratý aplikačný server JBoss, ktorý je použitý pre nasadenie výslednej aplikácie. Dôvodom použitia tohoto servera je možnosť použitia pokročilých testovacích nástrojov a nástroja na správu projektu, ktoré sú určené pre jazyka Java.

### 2.1 Špecifikácia platformy

Java EE je platforma, ktorá zastrešuje viaceré technológie a definuje prostriedky určené pre zjednodušenie vývoja komplexných podnikových aplikácií [12]. Tieto aplikácie sú rozsiahle, komplexné a kladú dôraz na bezpečnosť a spoľahlivosť. Z dôvodu prehľadnejšieho návrhu, implementácie a jednoduchšej údržby sú tieto aplikácie rozdelené do vrstiev. Týmto vrstvami sú: klient, Java EE Server a databázový server. Rovnako je súčasťou tejto platformy kolekcia špecifikácií od Sun/Oracle pre vývoj webových aplikácií, podporu webových služieb, . . . , ktoré zjednodušujú a zefektívňujú výsledný vývoj. Základom Java EE je špecifikácia Java SE, ktoré sú vyvíjané tzv. Java Community Process [14]. Ten predstavuje spoluprácu viacerých firiem, ktoré sa podieľajú na jej výsledku.

### 2.2 Trojvrstvový model

Java EE definuje rozdelenie aplikácie do vrstiev, ktoré medzi sebou komunikujú. Toto rozdelenie zprehľadňuje a uľahčuje vývojové cykly aplikácie. Každá vrstva je reprezentovaná inými technológiami z platformy Java EE. Jednotlivé vrstvy sú reprezentované komponentami, ktoré odpovedajú zodpovednosti danej vrstvy:

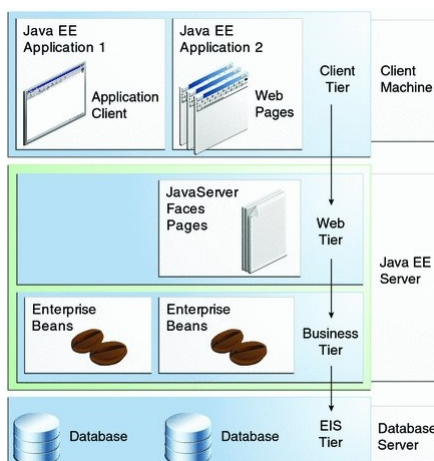
- Klientská vrstva sa skladá z klientských komponentov, ktoré bežia na klientskom počítači. Táto vrstva sa stará o spracovanie užívateľských vstupov a ich poslanie na spracovanej strednej vrstve
- Stredná vrstva sa skladá z webových a podnikových komponentov, ktoré bežia na Java EE serveri, ktorý predstavuje prostredie pre nasadenie, spravovanie a beh podnikových a webových komponentov Java EE aplikácie. Táto vrstva definuje logiku systému, keď



na jednu stranu pracuje s užívateľskými vstupmi a na strane druhej ukladá/získava informácie z najnižšej vrstvy.

- Najnižšia vrstva predstavuje externé systémy využívané Java EE aplikáciou. Typicky sa jedná o databázový server a externé systémy, ktoré označujeme názvom „Enterprise Information System(EIS)“. Úlohou tejto vrstvy je ukladať a sprístupňovať dát.

Na obrázku č. 2.1 môžeme vidieť viacvrstvové rozdelenie Java EE aplikácie: Klient prístu-



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

puje k Java EE aplikácií prostredníctvom webového prehliadača, alebo klientskej aplikácie. Klient môže byť 2 typov:

- Tenký klient - pozostáva z webové prehliadača, ktorý zobrazuje stránky pozostávajúce z rôzneho značkovacieho jazyka. Tenký klient sa dotazuje prostredníctvom Hypertext Transfer protokolu(HTTP), čo je internetový protokol pre výmenu hypertextových dokumentov, na webové komponenty na Java EE serveri.
- Hrubý klient - môže byť reprezentovaný rozličnými Java technológiami pre tvorbu užívateľských rozhraní, sa môže priamo dotazovať podnikových komponent a preskočiť tak komunikáciu s webovými komponentami. Úlohou tejto vrstvy je zobrazenie dát pre klienta.

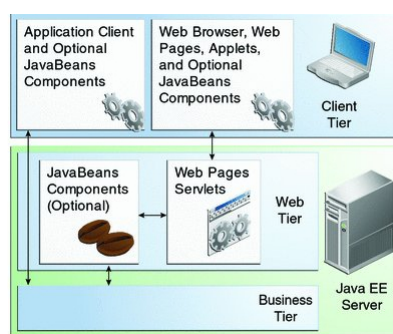
Táto vrstva taktiež zabezpečuje spracovanie užívateľských vstupov a ich validáciu.

Stredná vrstva sa delí na webovú vrstvu, ktorá je prezentovaná technológiami JavaServer Faces a JavaServer Pages. Druhá časť strednej vrstvy, takzvaná podniková vrstva, býva reprezentovaná technológiu Enterprise JavaBeans. Webová vrstva je reprezentovaná webovými komponentami, ktoré spracovávajú požiadavky od užívateľa a generujú odpoveď, ktorú posielajú naspäť užívateľovi. Môžu pritom komunikovať aj s podnikové komponenty pre zistenie dodatočných informácií(typicky informácií z databáze). Podniková vrstva je reprezentovaná podnikovými komponentami(beanami), ktoré tvoria logiku aplikácie. Tieto komponenty môžu prijímať požiadavky priamo od klienta klienta alebo webovej vrstvy a následne generujú odpovede, pričom môžu komunikovať s najnižšou vrstvou(napríklad komunikovať s databázovým serverom). Celá vrstva beží na Java EE serveri, kde sa teda sústreďuje logika špecifická pre danú aplikáciu.

Najnižšia vrstva predstavuje rozličné externé systémy, ktoré aplikácia môže využívať, či už sa jedná o databázový systém, alebo iné systémy. Vrstva býva označovaná skratkou EIS(External Information System). Úlohou tejto vrstvy je uchovanie, ukladanie a sprístupňovanie dát.

### 2.2.1 Princíp webových komponent

Java EE webové komponenty sú softwarové komponenty, ktoré spracovávajú prichádzajúci HTTP požiadok a poskytujú naň odpoveď. Všetky Java EE webové komponenty sú postavené na servletoch. Servlety sú javovské triedy, ktoré dynamicky spracovávajú požiadavky a tvoria odpovede. Súčasťou servletov alebo webových stránok sú technológie JavaServer Faces technológiu(JSF) 2.2.2 and JavaServer pages(JSP) [2]. Technológie JavaServer Faces a JavaServer Pages podporujú spracovanie užívateľských vstupov a ich predanie a spracovanie podnikovou vrstvou.



Obr. 2.2: Webové komponenty prevzate z <http://docs.oracle.com/javaee/6/tutorial/doc/bnaay.html>.

Na nasledujúcom obrázku č.2.2 je ukázaný princíp fungovania webových komponent. V hornej ľavej časti obrázku sa nachádza klientská vrstva, ktorá obsahuje buď len webový prehliadač po prípade Applety. Applet je aplikácia, ktorú spúšťa užívateľ prostredníctvom webového prehliadača a je vykonávaná virtuálnym strojom. 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 webovými komponentami. Takýto klient komunikuje už len priamo s Java EE serverom, konkrétne podnikovou vrstvou, ktorá implementuje zvyšnú logiku aplikácie. V prípade, že máme k dispozícii tenkého klienta, klient komunikuje prostredníctvom webového prehliadača s HTML alebo XHTML stránkami, ktoré sú vytvorené technológiou JavaServer Faces?? alebo JavaServer Pages [2], ktoré spracovávajú požiadavky od klienta(vstupy) a následne komunikuje s podnikovým stupňom, ktorý obsahuje logiku aplikácie, ktorá následne môže komunikovať s databázovým serverom. Odpoveď je v prípade tenkého klienta následne „predaná“ stránkám vytvoreným prostredníctvom JavaServer Faces alebo JavaServer Pages technológiou a následne zobrazená užívateľovi v podobe výstupu webovej stránky. V prípade hrubého klienta sa výstup zobrazí v aplikačnom klientovi.

## 2.2.2 JavaServer Faces

JavaServer Faces(JSF) je framework určený pre tvorbu užívateľských rozhraní webových aplikácií v jazyku Java [6]. Výsledné užívateľské rozhranie stavia zo základaných grafických komponent, ktoré framework obsahuje alebo pomocou vlastne definovaných komponent.

Komponenty si udržujú svoj stav, napriek tomu, že tento framework pracuje s bezstavovým protokolom HTTP. Tieto požiadavky sú obsluhované štandardnou cestou, ktorú nazývame životný cyklus spracovania požiadavky. Framework vytvára aplikácie na základe návrhového vzoru Model-View-Controller(MVC), ktorý oddeľuje logiku aplikácie od prezentačnej časti. MVC pracuje s nasledujúcimi princípmi:

- Model - špecifická reprezentácia dát, s ktorými pracuje aplikácia
- View - prevádza dáta aplikácie vhodné do podoby prezentácie užívateľa
- Controller - reaguje na udalosti, typicky od klienta a zabezpečuje zmeny v model alebo view

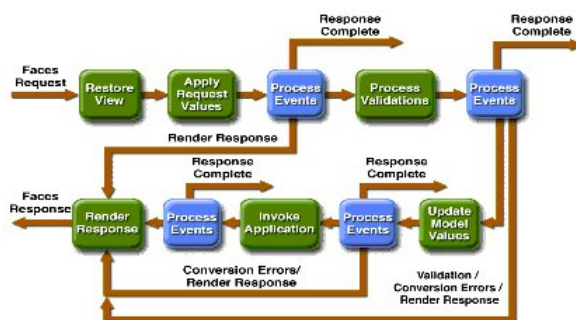
V JSF je controller implementovaný triedou FacesServlet, ktorý je súčasťou frameworku. Ten zabezpečuje spracovanie jednotlivých požiadavok, ktoré prichádzajú z URL adresy stránky aplikácie. Spracovanie požiadavky je daný životným cyklom spracovania požiadavky.

Model tvoria triedy, ktorých premenné a metódy sú zviazané s komponentami v prezentačnej časti. Tieto triedy majú meno prostredníctvom, ktoré sú adresované a rozsah ich životnosti v aplikácii.

Prezentačná vrstva je zložená zo stránok, ktoré sa označujú ako facelets. Tie určujú použité komponenty rozhrania a rovnako definujú mapovanie premenných a metód z modelu.

### Životný cyklus spracovania požiadavky

Celý štandardný cyklus spracovania požiadavky a následne generovania odpovedi je popísaný na nasledujúcom obrázku: Na obrázku č.2.3 môžeme vidieť životný cyklus JSF



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

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 sú snové hodnoty 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. 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 fázy Invoke Application sú spracované rozličné žiadosti ako potvrdenie formulára alebo link na inú stránku. V poslednej fáze Render Response dôjde k renderu stránky s novými hodnotami v kotajnery.

## Facelets

Facelets označuje deklaratívny jazyk pre tvorbu prezentačnej časti. Táto technológia nahradila staršiu technológiu JavaServer Pages [2]. Výhodou je oddelenie prezentačnej časti od aplikačnej logiky. Pre tvorbu stránok je použitá technológia XHTML [8].

Pri preložení aplikácie sa vytvorí zo stránky strom komponent, nad ktorými sú následne vykonávané operácie. Tieto komponenty sú typicky rozdelené podľa ich špecifickej funkcie do knižníc. Pred použitím komponenty z knižnice je potrebné definovať menný priestor knižnice, z ktorého komponenta pochádza.

## Managed Beans

Managed Beans sú triedy, ktoré sú definované v súbore faces-config.xml alebo sú anotované anotáciou @ManagedBean. Tieto triedy zhromažďujú dáta z prezentačnej časti. Managed Bean sú definované ich menom a rozsahom platnosti. ManagedBean je spravovaná automaticky, teda v prípade, že na stránke nachádza výraz požadujúci hodnotu z modelu JavaServer Faces automaticky zabezpečí priradenie konkrétnej Managed Bean-e. Pre prepojenie prezentačnej časti a Managed Bean sa používa špeciálny jazyk, ktorý sa nazýva Expression Language(EL). Tento jazyk zabezpečuje obojsmerné viazanie hodnôt medzi komponentami a Managed Beanami. JSF zabezpečuje jej pravidelné aktualizácie hodnôt v Managed Bean-e v rámci fázy aktualizácie hodnôt.

## 2.3 Twitter Bootstrap

Twitter Bootstrap je framework, ktorý obsahuje súbor nástrojov pre vytváranie webových stránok a webových aplikácií [20]. Ponúka podporu webových technológií HTML, CSS, JavaScript a mnohých prvkov, ktoré je možné ľahko integrovať do stránky. Twitter Bootstrap implementuje interaktívne prvky ako sú tlačidlá, boxy, menu a ďalšie grafické elementy. Pre použitie Bootstrap-u je potrebné vložiť do HTML kódu odkaz na kaskádové štýly a javascriptový súbor.

Výhodou týchto nástrojov je jednoduché používanie a možnosť použitia aj na mobilných telefónoch.

Bootstrap obsahuje rozšírenie Font Awesome, čo je CSS framework, ktorý obsahuje rôzne grafické ikony, ktoré je možné integrovať do HTML kódu [9].

## 2.4 Rich Faces

Rich faces je open-source framework s podporou Asynchrons Javascript and XML(AJAX)[10], ktorý predstavuje rozšírenie JSF frameworku 2.2.2. Rich Faces obsahuje

API, ktoré obsahuje grafické komponenty s podporou Ajax-u. RichFaces podporuje množstvo preddefinovaných vzhľadov. Rovnako umožňuje definovať, ktoré JSF komponenty budú invokované na základe Ajax požiadavky, vrátane spôsobu invokácie a odpovede. Rovnako podporuje validáciu na strane klientskeho prehliadača.

## 2.5 Webová služba

Webová služba je softwarový systém navrhnutý na podporu interoperability medzi rôznymi zariadeniami prostredníctvom počítačovej siete [12]. Komunikácia medzi zariadeniami prebieha prostredníctvom HTTP protokolu vymenovaním Extensible Markup language(XML) správ. XML je značkovací jazyk, ktorý definuje sadu pravidiel pre kódovanie dokumentu vo formáte porozumiteľnom človeku prostredníctvom ľubovoľných značiek. Webové služby poskytujú interoperabilitu medzi rôznymi platformami naprieč počítačovou sieťou. Tento aspekt je umožnený tým, že aplikácie komunikujú prostredníctvom HTTP protokolu. Komunikácia prostredníctvom webovej služby sa delí na 2 účastníkov. Prvý účastník producent(producer), ktorý vytvára požiadavok a spotrebiteľ(consumer), ktorý prijíma požiadavok. Komunikácia prebieha medzi týmito dvoma účastníkmi výmenou správ. Webová služba môže byť technicky implementovaná rôznymi možnosťami a to prostredníctvom Big Web Service alebo Restful WebService, pričom v princípe ide o Java triedy, ktoré obsahujú špeciálne definície triedy a metód a pri nasadení na Java EE server môžu byť vzdialene(po sieti) zavolané ich metódy.

### 2.5.1 Big webová služba

Big webová služba je druh webovej služby, ktorý pre svoju implementáciu používa API JAX-WS [12]. Tento typ webovej služby umožňuje vytvárať webové služby orientované na správy alebo na techniku vzdialeného volania procedúr(RPC). RPC je technológia, ktorá umožňuje volanie metód, ktoré sa nachádzajú na inom mieste, typicky inom mieste počítačovej siete. Tento typ webovej služby využíva XML správy, spolu so Simple Object Access Protocol(SOAP) a XML jazykom. SOAP definuje protokol pre výmenu správ založených na jazyku XML prostredníctvom siete prostredníctvom HTTP protokolu. SOAP správy sa skladajú z hlavičky a tela správy, ktoré obsahuje odpoveď webovej služby alebo požiadavok na vyvolanie akcie webovej služby. Nasledujúci obrázok č.2.4 ukazuje spôsob komunikácie



Obr. 2.4: „Big“ webová služba prevzaté z  
[<http://docs.oracle.com/javaee/6/tutorial/doc/bnayl.html>].

medzi klientom, ktorý sa nachádza v ľavej časti obrázku a webovou službou, 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 webovej služby obsahuje definíciu vo formáte Web Service Description Language(WSDL). WSDL je definícia vo formáte XML, ktorá popisuje aké akcie webová služba poskytuje a spôsob ich invokácie, rovnako aj odpoveď. 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. 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 konvový bod webovej služby. Táto trieda následne definuje metódy, ktoré môžu byť vzdialené volané. Aby mohla byť metóda metódou webovej služby musí byť anotovaná prostredníctvom anotácie `javax.jws.WebMethod @WebMethod`.

### 2.5.2 RESTful webová služba

RESTful webová služba je druh webovej služby, ktorý pre svoju implementáciu používa API JAX-RS [12]. Tento druh webovej služby nevyžaduje striktné používanie XML formátu a doručovanie správ vo formáte SOAP ako Big webová služba. Tento typ webovej služby je prístupovaný na základe Uniform Resource Identifier(URI), ktorý predstavuje textový reťazec, ktorý slúži k špecifikácii zdroja. K tomuto je používaná anotácia `@Path()`, ktorej hodnota zabezpečí namapovanie a teda pomocou nej môžeme prístupovať k RESTful webovej službe a to zadani „ hodnota anotácie `@Path`“ . Keďže táto služba nemá presne stanovený formát správ, môžeme zvoliť z formátov ako HTML, JSON, PDF, .... Tento typ služby je bezstavový, takže každý prístup musí obsahovať všetky potrebné informácie, pričom je možné ich označiť ako cachovateľné(uchávajúce sa vo vyrovnávanej pamäti) kvôli zvýšeniu výkonosti. Nevýhodou je, že pri vytváraní klienta a služby musí byť použité rovnaké rozhranie z dôvodu explicitnej nepodporovateľnosti jednoznačného formátu správy pre komunikáciu. Výhodou použitia tohto typu webovej služby je jednoduchosť implementácie producenta a klienta.

## 2.6 Java Persistence API

Java Persistence API(JPA) je framerok jazyku Java, ktorá poskytuje prístup a spravovanie relačných dát v databáze pomocou prístupu objektovo relačné mapovanie [16]. Princíp objektovo relačného mapovania predstavuje namapovanie javovskej triedy, ktorú nazývame entita na databázovú tabuľku. Základnou jednotkou, s ktorou pracuje tento framework je entita, ktorá predstavuje perzistentný doménový objekt, ktorého inštancia reprezentuje riadok v databázovej tabuľke. Základný artefaktom v programovaní je pre entity povinnosť obsahovať vlastnosti, ktoré priamo odpovedajú schéme vytvorenej databáze. Táto entitná trieda musí spĺňať nasledujúce vlastnosti:

- Entitná trieda musí byť anotovaná `javax.persistence.Entity` anotáciou
- Entitná trieda musí mať parametrický konštruktor, aby bolo možné vytvárať nové entity(riadky v tabuľke)
- Každá vlastnosť(položka) entitnej triedy musí spĺňať princíp Plain Old Java Object(POJO), čo znamená, že pre každú vlastnosť existuje metóda v tvare `getNázovVlastnosti`, ktorá získa hodnotu vlastnosti a metóda v tvare `setNázovVlastnosti`, ktorá nastaví danú hodnotu vlastnosti. Jednotlivé vlastnosti môžu byť dodatočne anotované napr. kvôli kontrole na hodnotu konkrétneho typu alebo vlastnosti(nenulovosť, špeciálny formát, ...)

- Každá entitná trieda musí mať unikátny identifikátor. Týmto identifikátorom chápeme primárny kľúč, čo je vlastnosť, ktorá dokáže v databáze jednoznačne identifikovať záznam. Primárny kľúč býva anotovaný prostredníctvom anotácie `javax.persistence.Id`

JPA obsahuje API, ktoré je nezávislé nad použitou databázovou technológiou, preto je možné vytvárať dotazy nad ľubovoľnou databázovou technológiou. Preto je pomerne jednoduché preniesť vytvorenú aplikáciu na iný typ databázovej technológie.

Jednotlivé entity môžu byť vo vzťahu s inými entitami. Vo vzťahu s databázovými tabuľkami je možné ho analogicky popísať tak, že nejaká tabuľka je závislá na inej. V prípade, že vlastnosť entity je súčasťou vzťahu s inou entitou používame niektorú z nasledujúcich anotácií podľa násobnosti vzťahu: `@One-to-one`, `@One-to-many`, `@Many-to-one`, `@Many-to-many`. Následne je uvedená vlastnosť/vlastnosti druhej entity, ktoré sa podieľajú na vzťahu. Tieto vlastnosti anotujeme anotáciou `javax.persistence.JoinColumn`, v ktorej parametroch uvedieme názvy vlastností druhej entity, ktoré sú súčasťou vzťahu.

Pre prácu z jednotlivými entitami sa používa `javax.persistence.EntityManager`. `EntityManager` je trieda, ktorá dokáže vytvárať a odstraňovať entity, umožňuje ich vyhľadávať, rovnako aj vytvárať dotazy nad databázou. Dotazy, ktoré môžeme vytvoriť pomocou JPA sa podobajú klasickému jazyku Structured Query Language(SQL), ktorý dokáže vytvárať dotazy nad databázou, avšak dotazovací jazyk JPA má niekoľko rozdielov. Tento jazyk sa nazýva Java Persistence Query Language(JPQL), čo je ako bolo spomenuté jazyk podobný SQL, pričom tento jazyk je reťazcovo založený a je nezávislý na zvolenej databázovej technológii a má objektové vlastnosti, čo znamená, že pri tvorbe dotazov používame názvy vlastností entitných tried a názvy entitných tried. Problém JPQL je typová nebezpečnosť, čo vyžaduje pretypovanie výsledkov dotazu z entity manager-a a to môže spôsobiť chyby, ktoré nemusia byť odchytené počas kompilácie.

JPA definuje ešte spôsob dotazovania, ktorý sa nazýva Criteria API, ktoré je využívané k vytváraniu dotazov nad entitami a vzťahom, ktoré sú typovo bezpečné. Výhodou tohto API, pre použitie na dotazovanie, je rovnako možnosť vytvárať dynamické dotazy, ktoré majú lepšiu výkonnosť ako JPQL.

Pre určenie, s ktorými entitami má `EntityManager` pracovať je používaný XML súbor `persistence.xml`. Tento súbor obsahuje perzistentnú jednotku(persistence unit), čo je XML predpis, do ktorého uvedieme entitné triedy, odkaz na databázu po prípade ďalšie vlastnosti. Tento súbor predstavuje konfiguráciu, ktorá obsahuje okrem názvu entitných tried, s ktorými má `EntityManager` pracovať aj rôzne iné vlastnosti, napr. je automatické vytvorenie schémy databáze z entitných tried. V tomto súbore sa rovnako nachádza dôležitá položka a to je `datasource`, ktorý definuje odkaz na databázu, s ktorou pracujeme.

## 2.7 Enterprise JavaBeans

Enterprise JavaBeans(EJB) je technológia, ktorá umožňuje vytvárať komponenty, ktoré v strednej podnikovej vrstve trojvrstvového aplikačného modelu 2.2 [7]. Tieto komponenty sú modulárne, keďže je možné ich vytvoriť a spravovať viac ich inštancií. Cieľom týchto komponent je uchovávanie aplikačnej logiky.

Takéto komponenty komunikujú s klientom alebo webovými komponentami a na druhej strane môžu komunikovať s EIS vrstvou a vykonávajú/predávajú získané informácie. Na EJB sa môžeme pozeráť aj ako na API platformy Java EE, prostredníctvom, ktorého môžeme

vytvárať triedy, ktoré sú špeciálne anotované a obsahujú podnikovú logiku a sú nasadené na Java EE server. Triedy vytvorené týmto API sa nazývajú Enterprise Bean-y(EB).

EB sa delia na 2 kategórie:

- Message-driven bean - Komponenta pôsobí v roli poslucháča určitého typu správ, na ktorých príjem reaguje vykonaním určitých akcií [12]
- Session bean - Komponenta vykoná úlohy pre klienta. Môže implementovať webové služby [12]

### 2.7.1 Session Bean

Session bean(SB) je typ EB, ktorá zapúzdruje podnikovú logiku, pričom môže byť vyvolaná lokálne alebo vzdialene. Prístup k session bean je realizovaný prostredníctvom volania metód SB. SB následne vykoná kód metódy, po prípade vráti nejaký výsledok

SB delíme na 3 typy:

- Stateful Session Bean - Udržiava hodnoty premených, každá beana reprezentuje unikátny stav klienta/sedenia. Pokiaľ sa sedenie odstrániť stav zmizne.
- Stateless Session Bean - Neudržiava stav komunikácie s klientom. Počas invokácie metódy takejto beany môže inštancia obsahovať premenné, ktoré môžu obsahovať špecifický stav vzhľadom na klienta. Po ukončení stav zmizne, rovnako tento typ SB je možné použiť k implementácii webovej služby.
- Singleton Session Bean - Tento typ beany je inštanciovaný len raz a pretrváva počas celého životného cyklu aplikácie. Využíva sa pri zdieľaní a súčasnom prístupe viacerých užívateľov.

### 2.7.2 Message-driven Bean

Message-driven bean(MB) je typ EB, ktorá umožňuje aplikáciám asynchrónne spracovanie správ. Tento beany prijíma správy z JMS fronty, ktoré následne analyzuje a vykonáva akcie [15]. JMS je technológia, ktorá umožňuje komunikovať komponentám prostredníctvom správ. JMS fronty sú obyčajné fronty, do ktorých sa na jednom konci pri zavolaní MB vloží špecifická JMS správa a na druhej strane sú MB postupne tieto správy odoberané a spracované len raz.

Zásadný rozdiel je oproti session bean v zásade v tom, že sa k takému typu beany nepristupuje prostredníctvom rozhrania a invokácie metód. Prístup k takému typu EB sa deje prostredníctvom vytvorenia spojenia s JMS frontou a vložení správou do fronty. Správy sú následne spracované na strane MB metódou `onMessage`, ktorá vyberá z JMS fronty správu po správe. Výhodou MB je ekvivaletnosť MB, to znamená že správy môže byť ľubovoľnej inštancii. Výhodou je asynchrónne vyvolanie, ktoré nevyťažuje prostriedky servera.

## 2.8 MySQL

MySQL je open source relačná databázová technológia, ktorá je vhodná pre malé a stredne veľké aplikácie, pričom je vyvíjaná spoločnosťou Sun Microsystems. Výhodou tejto technológie je dobrý výkon pri vykonávaní transakcií, vytváranie databázových procedúr, data-



bázových triggerov a jednoduchá inštalácie. Táto technológia je multiplatformová, preto je možné ju nasadiť na systémy s operačným systémom Windows, Linux, Mac Os.

Medzi nevýhody tejto technológie patrí neefektívna práca s databázovými transakciami, neefektívne ukladanie veľkého množstva dát.

## 2.9 Seam

Seam je aplikačný framework pre Java EE, ktorý definuje uniformný komponentný model pre podnikovú logiku aplikácie [17]. Seam rieši integráciu EJB 2.7 a JSF 2.2.2 spolu. Medzi ďalšie výhodné vlastnosti tohto frameworku patrí integrácia Asynchronous JavaScript and XML(Ajax) [10], rovnako aj vstavaná podpora javascriptu a efektívne spracovanie webových dotazov.

Tento framework obsahuje množstvo modulov od zabezpečenia aplikácie až prácu s emailovou komunikáciou. My sa zameriame na modul Seam Security. Tento modul obsahuje mechanizmy na zabezpečenie enterprise aplikácie a overovanie užívateľa.

Základom bezpečnosti tohto modulu je autentifikácia, čo je process vytvorenia alebo potvrdenia identity užívateľa. Užívateľ potvrdzuje svoju identitu prostredníctvom užívateľského mena a hesla. Seam security poskytuje API prostredníctvom, ktorého je možné sa autentizovať z rozličných zdrojov(databáze, ...).

Ďalšou vlastnosťou je Identity Management, ktoré je množina API na správu užívateľov, skupín a užívateľských rol, ktorá je súčasťou Seam Security API. Identity Managent je poskytovaný Seam komponentou PicketLink IDM, ktorá spravuje uloženie užívateľov v rozličných bezpečnostných úložiskách.

Základom autentifikácie je Identity Bean, čo je java trieda, ktorá reprezentuje identitu užívateľa a pri úspešnej autentifikácii je identita vložená do životného cyklu aktuálneho sedenia. Týmto spôsobom(prítomnosťou triedy Identity Bean) sa overuje užívateľ. V rámci autentifikácie sú definované v rámci API metódy Login(prihlásenie) a Logout(odhlásenie). Potvrdenie identity užívateľa je realizovaná metódou authenticate, v ktorej prebieha autentifikácia užívateľa.

Počas autentifikácie sa overí pravosť užívateľa a prostredníctvom metódy setStatus sa nastaví úspech(SUCCESS) alebo neúspech(FAILURE) pri overení zadaných údajov. Po autentifikácii dôjde k vloženiu identity užívateľa do životného cyklu aplikácie, ktorú je možné získať z triedy Identity prostredníctvom anotácie @Inject triedy Identity.

Na záver spomenieme modul Seam Faces. Tento modul obsahuje API na zabezpečenie prístupu k HTML a XHTML stránkam. Túto funkčnosť nazývame *Faces View Configuration*. Ide vlastne o súbor, v ktorom je uvedené, ktorý užívateľ môže pristupovať, ku ktorej stránke. Z dôvodu overenia užívateľa existuje pre Seam Security modulom, ktorý vloží identity do životného cyklu aplikácie.

## 2.10 Testovanie

Základom testovania je nástroj JUnit a nástroj Arquillian. V prvom rade sa budem venovať nástroju JUnit. JUnit je unit testovací nástroj pre programovací jazyk Java. JUnit sa používa pre typ testovania, ktorý sa nazýva test-driven development a je jedným z kolekcie unit testovacích nástrojov [18]. JUnit býva súčasťou balíku org.junit [13]. JUnit testy sú písané pre otestovanie konkrétnej funkčnosti kódu. Cieľom testovania prostredníctvom JUnit sú malé kúsky kódu, ako metódy alebo triedy.

Testovacie metódy sú anotované prostredníctvom @Test anotácie. JUnit rovnako umožňuje vykonať kód pred spustením testu, to docielime anotovaním metód @Before anotáciou alebo po spustení testu, to docielime anotáciou @After. V testovacej metóde potom vykonáme nejaký kód a očakávaný výstup porovnáme s nami očakávaným výsledkom prostredníctvom metódy Assert.

Nakoniec spomeniem nástroj Arquillian. Arquillian je testovací nástroj, ktorý vykonáva testy vo vnútri vzdialeného alebo vstavaného kontajneru alebo nasadí archív (obsahujúci java triedy spolu s testovacími triedami) na Java EE kontajner (JBoss, Tomcat, ...). Arquillian integruje aj ďalšie testovacie nástroje, napr. JUnit 4, TestNG 5, .... Treba zdôrazniť, že na rozdiel od JUnit testov umožňuje testovanie v java EE kontajnery (GlassFish, JBoss) [1]. Tento framework má zásadnú výhodu v prenositeľnosti testov na rôzne podporované Java EE kontajnery. Nástroj pri spustení automaticky zabalí do archívu všetky potrebné prostriedky pre platformu. Pre správny beh testov je potrebné nakonfigurovať XML súbor arquillian.xml. V tomto súbore sa nastaví kontajner, na ktorý budú testy nasadené a spôsob spustenie testov.

Písanie testov s nástrojom Arquillian začína tvorbou javovskej triedy, ktorá vyzerá ako štandardná testovacia trieda vytvorená nástrojom JUnit.

Použitie nástroja Arquillian sa deje použitím anotácie @RunWith. Táto anotácia zabezpečí spustenie testov v testovacej triede v Java EE kontajnery. Následne tento nástroj spustí kontajner a nasadí testovací archív, ktorý obsahuje anotáciu @Deployment. Archív obsahuje testy so špecifickými triedami a knižnicami, ktoré potrebuje. Testy sa následne vykonávajú vo vnútri kontajneru, preto je možné otestovať podnikové a webové komponenty za behu.

## 2.11 JBoss Aplikačný server

Aplikačný server (AS) je software, ktorý poskytuje vrstvu medzi operačným systémom a Java EE aplikáciami. AS poskytuje funkcionality aplikáciám (prístup k súborovému systému, ...), konkrétne enterprise aplikáciám. Vytvára vrstvu, ktorá zjednodušuje vývoj enterprise aplikáciám. Pomerne veľká skupina AS je vyvíjaná v jazyku Java. Dôvodom pre tento jazyk je existencia štandardu Java EE.

JBoss (JavaBeans Open Source) je aplikačný server, ktorý je založený na platforme Java a Java Enterprise Edition [11]. Tento typ AS je open-source, preto je možné jeho stiahnutie spolu so zdrojovými kódmi. Základným stavebným kameňom JBoss AS je JBoss Microcontainer. JBoss Microcontainer je refaktORIZÁCIA JBoss JMX Microkernel aby podporoval POJO nasadzovanie a samostatné použitie mimo aplikačného servera. Microcontainer registruje všetky použité služby. Služby, ktoré majú byť prístupné sa registrujú v podobe managed beans. Microcontainer spravuje a riadi beh týchto služieb. JBoss je licensovaný pod GNU Lesser General Public License (GNU PL).

Používanie aplikačného servera JBoss je veľmi jednoduché, jeho spustenie môžete vykonať ručne prostredníctvom konzoly a nájdeným inštaláčnym adresárom, ktorý obsahuje skript run.sh, ktorý spustí JBoss. Po spustení serveru je možné k nemu implicitne pristupovať na localhost na porte 8080.

## Kapitola 3

# OptaPlanner

OptaPlanner je open source framework a pokračovanie frameworku JBoss Drools, ktorý rieši a optimalizuje rôzne plánovacie problémy, ktoré sú reprezentované XML súborom, s rozličným stupňom náročnosti. Optaplanner využíva pri riešení problému, ktoré nemusí vždy nájsť, optimalizačné algoritmy a metaheuristické metódy s využitím skóre. Skóre je hodnota, ktorá reprezentuje bodové hodnotenie optimálnosti dosiahnutého riešenia. Výsledným riešením je to riešenie, ktoré má najvyššie skóre a má podobu XML súboru.

Tento framework neurčuje striktne akými algoritmami a metódami sa má daný problém vyriešiť, ale konfiguráciu ponecháva na strane užívateľa. OptaPlanner je určený pre jazyk Java, preto riešenie je vykonávané triedami v tomto jazyky. Tieto triedy sú špecifické pre daný problém, a musia byť schopné získať potrebné informácie z defičného súboru problému, ktorý reprezentuje zadanie problému, musia byť schopné vykonávať postupné kroky vedúce k riešeniu problému(napr. v prípade problému N Dám presúvať dámy, tak aby sa vždy nachádzali vo validných pozíciách) a prostriedky, ktoré ohodnotnia krok a prekalkulujú celkové skóre. A na záver vrátiť najlepšie riešenie.

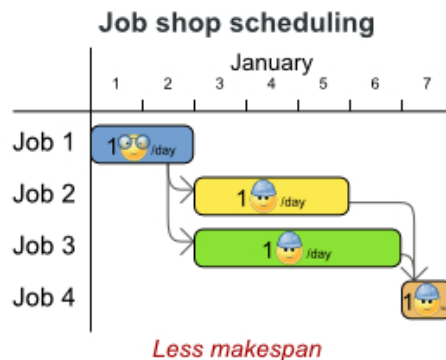
Postup riešenia problému, kalkulácií skóre sa opakuje pre rôzne scenáre(napr. v prípade N Dám pre rôzne alternatívne kombinácie pohybov, ktoré sú rozpracované súčasne). vráti sa riešenie s najlepším skóre v podobe súboru vo formáte XML(napr. v prípade riešenia problému N Dám poskytne najlepšie možné riešenie rozmiestnenia). OptaPlanner sa snaží vždy nájsť optimálne riešenie vzhľadom k optimalizačným algoritmom a metaheuristickým metódam a dostupnému času, ale niekedy nie je schopný poskytnúť na predchádzajúce podmienky optimálne riešenie(riešenie je ukončené predčasne z dôvodu vyčerpania dostupného času). Výhodou tohto frameworku je možnosť aplikovania na NP-úplne problémy(ktoré sú riešené v dostupnom čase).

### 3.1 Plánovací problém

Plánovacím problémom môžeme obecné rozumieť akýkoľvek problém, ktorý vyžaduje od nás zdroje a predikciu na priradenie zdrojov, nájdenie riešenia takého, aby výsledok bol v konečnom dôsledku najlepší, cenovo aj časovo najpriateľnejší.

V bežnom živote, rovnako ako ja v podnikových sférach sa stretávame s rôznymi plánovacími problémami. Môže ísť o problémy ako správne naplánovať cestu vozidiel(áut, lodí, ...), aby sme ju spravili za čo najkratší čas, rovnako môžeme požadovať aby cesta bola, čo finančne najpriateľnejšia. Rovnako môžeme plánovať rozvrh práce zamestnancov vo firme, aby zbytočne nespomalovali chod ostatných zamestnanci, ktorí sú na ich práci závislí a ne-

museli zbytočne čakať. Plánovať môžeme spúšťanie testovania aplikácií v rámci vývojarskej firmy, aby niektoré úlohy boli otestované skôr ako iné, no musí byť čo najefektívnejšie vyváženie použitých zdrojov (procesorového času) a zbytočne prostriedkami nemrhali. Pokiaľ je problém dostatočne komplexný potom je veľmi vhodné použiť Optaplanner.



Obr. 3.1: Problém rozvrhnutia práce, prevzaté z [<http://www.optaplanner.org/>].

Obrázok č. 3.1 zobrazuje typické použitie OptaPlanner-u. Môžeme vidieť v nasledujúcom obrázku vystupujú 4 osoby (označené obdĺžnikom modrej, žltej, zelenej a oranžovej farby), ktoré vykonávajú nejakú činnosť. Ich činnosť je špecifická a silne závisí od práce predchádzajúcich a teda nemôžu začať pracovať pokiaľ nie je spravená práca predchádzajúceho pracovníka. V prípade náročnosť zadania takého problému je pomerne jednoduché naplánovať správne poradie činností. Problém nastáva, ak by v danom obrázku bolo niekoľko násobne viac ôsob. V tomto prípade štandardným prístupom by mohlo dôjsť k neefektívnemu rozdeleniu práce a k zbytočnému mrhaniu času. Preto je vhodné použiť OptaPlanner, ktorý sa snaží ich činnosti maximálne optimalizovať a jednotlivé činnosti zvolí v následnosti tak, aby výsledná práca bola spravená za najkratší možný čas vzhľadom na činnosť, ktorá sa optimalizuje.

Na obrázku č. 3.1 je uvedená prirodzená definícia plánovacieho problému, ktorá môže v oblasti informačných technológií spôsobiť nejednoznačnosť v jej interpretácii, preto je používaný súbor vo formáte XML, v ktorom definujeme počiatočné zadanie plánovacieho problému. Formát XML súboru závisí od zadania problému. V prípade, že si zobere problém N Dám, tak zadanie súboru obsahuje presnú pozíciu dám na šachovnici. Keď si zobereme problém obchodného cestujúceho, tak definičný XML súbor obsahuje zoznam miest a jednotlivé vzdialenosti od seba. Obsah definičného súboru nemá jednoznačný formát, ale vždy závisí od plánovacieho problému.

## 3.2 Princíp

Princíp riešenia je založený na konfigurácii OptaPlanner-u prostredníctvom konfiguračného súboru, v ktorom sa nastavujú optimalizačné algoritmy a metaheuristické metódy, ktorý sa snažia v spolupráci s triedami na riešenie vyberať vždy najlepšie kroky pri riešení. Rovnako sa vytvorí a v konfiguračnom súbore zdefinujú javovské triedy na získanie potrebných dát z definičného súboru, prostriedky na kalkuláciu skóre.

Riešenie problémom sa začína tvorením XML definičného problému špecifického pre daný

problém. Následne sa vytvoria triedy pre získanie dát z XML súboru, triedy pre vykonávanie krokov plánovania(napr. v prípade N dám presúvanie dám na validné pozície) a prostriedky pre kalkuláciu skóre a nastavenie konfiguračného súboru pre daný problém, ktorý bude bližšie popísaný v nasledujúcej kapitole 3.3. Aby bolo jasné aké akcie(kroky plánovania) sú povolené pre daný problém sú definované v triedach pre riešenie obmedzenia. Tie definujú spôsob ohodnotenia skóre pre každý krok: [4]

- Negatívne „hard“ obmedzenie, ktoré nesmú byť porušené. Pri zistení tohto typu obmedzenia je krok ohodnotený záporným skóre(to indikuje nekorektnosť kroku vzhľadom k plánovaciemu problému)
- Negatívne „soft“ obmedzenie, ktoré by nemali byť porušené pokiaľ sa dá tomu vyhnúť. Pri zistení tohto typu obmedzenia je krok ohodnotený kladným skóre s nízkou hodnotou
- Pozitívne „soft“ obmedzenia, ktoré by mali splnené pokiaľ je to možné. Pri zistení tohto typu obmedzenia je krok ohodnotený kladným skóre s vysokou hodnotou

Pre každý rozpracovaný krok plánovacieho problému sa priebežne sčítava priebežné skóre s predchádzajúcim. Týmto spôsobom dostaneme viaceré riešenia s rozličným skóre. Kalkulácia skóre je vykonávaná špeciálnymi prostriedkami(triedami), pričom existujú 3 spôsoby, akým je skóre kalkulované:

- Jednoduchá kalkulácia skóre 1 metódou
- Inkrementálna kalkulácia skóre prostredníctvom viacerých metód
- Drools kalkulácia skóre - táto konfigurácia definuje pravidlá pre kalkulovanie skóre

Drools kalkulácia skóre využíva vlastnú DRL syntax a je daná súborom, ktorý obsahuje pravidlá [3]. Každé pravidlo je dané svojim názvom a podmienkou, v ktorej sa overuje priebežné riešenie problému(napr. v prípade N Dám priebežné rozloženie dám), ktorá v prípade splnenia upravuje skóre.

Spustenie riešenia je dané zavolaním hlavnej metódy solve z triedy Solver, ktorá spúšťa riešenie problému. Postup riešenia je nasledovný:

1. Overenie prostriedkov(definičného súboru, konfiguračného súboru(obsahuje spôsob kalkulácie, definičného triedy, použitie plánovacích algoritmy a metaheuristických metód) a prostriedkov na kalkuláciu skóre)
2. Načítanie obsahu XML súboru
3. Vykonanie kroku podľa použitia plánovacích algoritmov
4. Optimalizácia kroku v prípade použitia metaheuristických metód
5. Ohodnotenie kroku(v závislosti od použitia prostriedkov na kalkuláciu skóre 3.2)
6. Vykonanie alternatívneho kroku vzhľadom(napr. v prípade N Dám presunutie dámy na ľavú stranu šachovnice, miesto pravej)
7. Optimalizácia alternatívneho kroku v prípade použitia metaheuristických metód
8. Ohodnotenie kroku(v závislosti od použitia prostriedkov na kalkuláciu skóre 3.2)

9. Opakovanie krokov 3., 4., 5., 6. až dokým nie je dosiahnuté riešenie alebo plánovanie nie je predčasne ukončené
10. Nájdenie riešenia alebo predčasné ukončenie plánovanie vzhľadom na vysoké poskytnuté skóre (je možné použiť v prípade, že riešenie problému nebolo nájdené v dostupnom čase) a vrátenie najlepšie riešenia vo formáte XML súboru

### 3.3 Konfiguráciu OptaPlanneru

Konfigurácia OptaPlanner sa realizuje prostredníctvom XML súboru, ktorá má 3 povinné časti a 4. voliteľnú:

- Nastavania definičných tried plánovacie problému a nastavania tried zabezpečujúce plánovanie (Domain model configuration)
- Nastavenie definícií skóre (Score Configuration)
- Nastavenie použitia plánovacích algoritmov (Optimization algorithms configuration), ktoré voliteľne obsahuje nastavenie metaheuristickej metódy

Pre lepšiu prehľadnosť je uvedená ukážka konfiguračného súboru.

Listing 3.1: 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>

```

Nastavenie konfiguračného súboru(solver) pre riešenie problému vyváženie cloudu na obr.3.1 pozostáva z viacerých častí:

- Na riadku č.3 uvedená medzi značkami environmentMode hodnota „FAST\_ASSERT“, ktorá umožňuje OptaPlanner detekovať chyby v implementácii
- Na riadku č.5 je uvedená medzi značkami solutionClass hodnota „org.optaplanner.examples.cloudbalancing.domain.CloudBalance“, ktorá odkazuje na definičnú triedu modelu problému vyváženie cloudu
- Na riadku č.6 je uvedená medzi značkami planningEntityClass hodnota „org.optaplanner.examples.cloudbalancing.domain.CloudProcess“, ktorá odkazuje na triedu, ktorá realizuje riešenie(plánovanie) problému
- Na riadku č.9 je uvedená medzi značkami scoreDefinition hodnota „HARD\_SOFT“, ktorá hovorí, že pri kalkulácii skóre použijeme len hard obmedzenia 3.2

- na riadku č.10 je uvedená medzi značkami `simpleScoreCalculatorClass` hodnota „`org.optaplanner.examples.cloudbalancing.solver.score.CloudBalancingSimpleScoreCalculator`“, ktorá odkazuje na triedu, ktorá kalkuluje skóre pri riešení problému
- Na riadku č.11 je uvedená medzi značkami `scoreDrl` hodnota „`/org/optaplanner/examples/cloudbalancing/solver/cloudBalancingScoreRules.drl`“, ktorá odkazuje na Drools definíciu kalkulácie skóre 3.2
- na riadku č.15 je uvedená medzi značkami `maximumSecondsSpend` hodnota „120“, ktorá hovorí, že riešenie musí byť nájdené do 120 sekúnd v opačnom prípade dôjde k ukončeniu riešenia a vráteniu najlepšieho dosiaľ dosiahnutého riešenia
- Na riadku č.18 je uvedená medzi značkami `constructionHeuristicType` hodnota „`FIRST_FIT_DECREASING`“, ktorá označuje použitie plánovacieho algoritmu `FIRST_FIT_DECREASING` [19]
- Na riadku č. 20 je uvedená medzi značkami `pickEarlyType` hodnota „`FIRST_NON_DETERIORATING_SCORE`“, ktorá označuje použitie pri kalkulovaní skóre najprv nezhoršujúce sa skóre(použitie kladného skóre)
- Na riadku č. 25 je uvedená medzi značkami `entityTabuSize` hodnota „`entityTabuSize`“, ktorá značí použitie metaheuristickej metódy pri riešení `TABU SEARCH`, s veľkosťou tabuľky 7 [19]
- Na riadku č. 28 je uvedená medzi značkami `acceptedCoundLimit` hodnota „1000“, ktorá označuje počet náhodných krokov, ktoré sú vyhodnotené počas 1 kroku riešenia problému

### 3.4 Výsledky plánovacieho problému

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. Rovnako je možné rozpracovať viacero riešení problému prostredníctvom rôznych plánovacích algoritmov a metaheuristických metód, pričom je poskytnuté riešenie s najlepším skóre reprezentované XML súborom. OptaPlanner rovnako podporuje tvorbu štatistík a výstupu v podobe grafu z výsledku plánovania.



## Kapitola 4

# Aplikácia

V tejto kapitole postupne uvedieme požiadavky na aplikáciu, analýzu systému, návrh aplikácie, implementáciu, testovanie a nakoniec vyhodnotíme aplikáciu a navrhujeme jej možné rozšírenia.

### 4.1 Špecifikácia požiadavkov

V tejto kapitole postupne rozobereme požiadavky na systém monitorovania stavu úloh. Základnou úlohou systému je monitorovanie úloh. Na jednej strane bude systém schopný zobrazovať stav plánovacích úloh, na druhej stranej bude môcť systém plánovacie úlohy spúšťať/pozastaviť.

Úlohy bude možné triediť podľa určitého kritéria, rovnako systém bude schopný aj úlohy vyhľadávať. Jednotlivé úlohy je možné aj mazať, alebo zmeniť definíciu plánovacieho problému 3.1 a úlohu znovu spustiť. Novú úlohu bude možné do systému vložiť a následne spustiť.

Úlohy bude môcť systém publikovať, čím sa myslí akcia, ktorá vytvorí pre úlohu špeciálne URL, na ktoré po kliknutí zobrazí stránku s názvom úlohy a obsahom XML definíčného súboru. Úlohu bude možné aj odpublikovať a po pristúpení k odpublikovanej úlohe sa vráti prázdny obsah.

Systém bude rozdelený podľa užívateľ do 3 užívateľských rolí (Administrátor, Plánovač, Čitateľ):

- Administrátor - má prístup ku všetkým úlohám v systéme, úlohy môže editovať, vytvárať, mazať, publikovať, 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

Užívatelia sú organizované do väčších celkov (organizácií). Preto systém bude schopný spravovať užívateľov, rovnako aj spravovať organizácie, ktoré bude schopný prehľadne zobrazovať, triediť a vyhľadávať podľa určitého kritéria. Užívateľov a organizácie je možné vytvárať.

Každý užívateľ si bude môcť v systéme meniť svoj email a heslo. Vytvorený užívateľ sa do systému prihlasuje, pričom po prihlásení je sprístupnená len časť systému podľa užívateľskej role prihláseného užívateľa. Aplikácia bude obsahovať bezpečnostné mechanizmy,

ktoré zabezpečujú aplikáciu proti neautorizovanému prístupu užívateľov. Vstupy do systému budú:

- Definičný súbor plánovacieho problému
- Užívatelia systému, ktorý vykonávajú akcie v systéme
- Organizácie, do ktorých sú začleňovaný užívatelia

Výstupy zo systému budú:

- zoznam plánovacích úloh v prehľadnej tabuľke
- zoznam užívateľov a organizácií, ktoré sa rovnako zobrazujú v prehľadnej tabuľke

V predposlednom rade treba spomenúť, že výslednej rozhrania bude schopné byť prenositeľné na mobilné telefóny.

## 4.2 Analýza

Výslednú aplikáciu môžeme rozdeliť na 2 časti: 1. backend aplikácie, ktorý beží na Java EE serveri JBoss 2.11 2.frontend aplikácie grafické užívateľské rozhranie.

Zameriame sa najprv na grafické užívateľské rozhranie. Pri analýze grafického užívateľského rozhrania je potrebné vyriešiť problém jeho návrhu a možnosti jeho interakcie. Použitie technológie JSF 2.2.2 je pomere jednoznačná, keďže z Java EE poskytuje jednoduchú interakciu a uchovávanie stavov jednotlivých komponent. Jej výhodou je jednoduchá integrácia s aplikačným serverom JBoss.

Problémom, ktoré užívateľské rozhranie potrebuje vyriešiť je pravidelné obnovovanie obsahu tabuliek plánovacích úloh, organizácií a užívateľov, ktoré prostredníctvom technológie JSF je pomerne málo konfigurovateľné. Lepšie riešenie poskytuje použitie frameworku Rich Faces 2.4, ktorý priamo integruje Ajax do všetkých jeho komponent [10].

Posledným problémom, ktorý treba pri analýze grafického užívateľského rozhrania vyriešiť je prenositeľnosť na mobilné zariadenia. V tom nám pomôže framework Twitter Bootstrap 2.3. Prenositeľnosť je možná na mobilné rozhrania disponujúce ľubovoľne veľkou zobrazovacou jednotkou. Treba ale zdôrazniť, na ktorých webových prehliadačoch je možné aplikáciu bez problémov prehliadať:

- Na systéme Android: Chrome, Firefox
- Na systéme iOS: Chrome, Safari
- Na systéme Mac OS X: Chrome, Firefox, Opera, Safari
- Na systéme Windows: Chrome, Firefox, Internet Explorer(verzia 8 - 11), Opera, Safari
- Na systéme Linux: Chromium, Firefox

Tento framework sa vždy snaží podporovať najnovšie verzie všetkých vyššie uvedených prehliadačov. Podpora ostatných prehliadačov nie je odporúčaná z dôvodu neočakávaného chovania.

V druhej časti sa zameriame na problémy backend-u aplikácie. Celá aplikácia potrebuje udržiavať a spravovať dáta. Dátami sú mienené informácie o úlohách, užívateľoch a organizáciach. Z toho dôvodu bolo treba vyriešiť otázku voľby vhodnej databázovej technológie.

Existuje niekoľko možností, ktoré sa dajú ľahko integrovať s Jboss-om 2.11. Keďže nároky na vyťaženosť prístupu k dátam, rovnako aj množstvo uložených dát sú malého merítka bolo vhodné zvoliť k tomu adekvátnu databázovú technológiu a tou technológiou je MySQL 2.8.

Následne treba spomenúť problém komunikácie s užívateľským rozhraním. Grafické užívateľské rozhranie potrebuje komunikovať s databázou odkiaľ získava aktuálne informácie o úlohách, užívateľoch a organizáciách. Rovnako sa do databázy zapisujú priebežné informácie o plánovaní. Vzhľadom na podmienku nezávislosti použitia databázovej technológie bola použitá technológia JPA 2.6.

Pre publikovanie bola zvolená jednoduchšia varianta webovej služby a to RESTful webová služba 2.5.2, ktorá bude namapovaná na URI „task/parameter“ a parameter predstavuje ID úlohy, ktorá sa má zobraziť. Chovanie tejto služby je možné rozdeliť podľa prihláseného a neprihláseného užívateľa:

- Neprihlásený užívateľ - V prípade pokusu o prístup k verejnej úlohe bude zobrazený jej názov a XML definičný súbor. V prípade pokusu k neverejnej službe bude vrátený prázdny obsah stránky.
- Prihlásený užívateľ podľa užívateľskej role:
  - Administrátor - Má prístup k všetkým plánovacím úlohám
  - Plánovač - Má prístup k úlohám v rámci organizácie, do ktorej patrí
  - Čitateľ - Má prístup k úlohám v rámci organizácie, do ktorej patrí

Výsledné užívateľské rozhranie bolo potrebné zabezpečiť voči neautorizovanému prístupu. Existuje priamo zabezpečiť aplikáciu pomocou štandardného API Java EE, no bol zvolený framework Seam 2.9, ktorý možno jednoducho integrovať pod JBoss.

Komunikácia s plánovacou časťou je realizovaná prostredníctvom webovej služby 2.5 prostredníctvom HTTP protokolu. Z dôvodu použitia štandardných komunikačných protokolov a nižším nákladom na prevádzkovanie bola zvolená „Big“ webová služba 2.5.1. Užívateľské rozhranie predstavuje klienta, ktorý volá metódy na spustenie a pozastavenie výpočtu. PlannerService predstavuje koncový bod webovej služby a zachytáva správy od klienta a zabezpečuje spúšťanie/pozastavenie výpočtu(plánovania).

PlannerService je realizovaná v podobe session bean-y 2.7.1, ktorá reprezentuje webovú službu a obsahuje funkčnosť pre spustenie a zastavenie výpočtu. Pri spustení výpočtu sú informácie predávané message-driven bean 2.7.2, ktorá zabezpečuje spúšťanie plánovania prostredníctvom OptaPlanner-u 3.

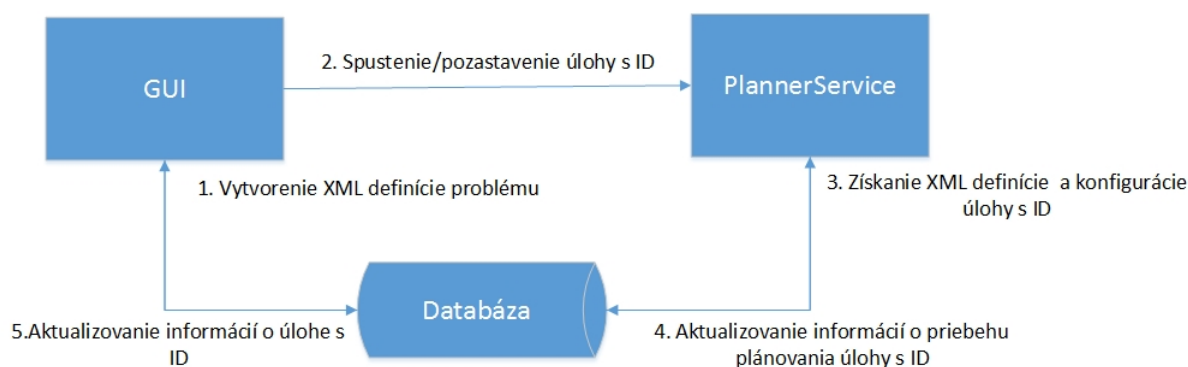
Rovnako boli použité štandardné prostriedky na otestovanie funkčnosti kúsky kódu pomocou JUnit testov a nástroju Arquillian 2.10.

Kvôli závislosti časti systému PlannerService na entitných triedach, bola aplikácia užívateľského rozhrania rozdelená na multimodulový projekt.

### 4.3 Návrh aplikácie

Výsledná aplikácia je rozdelená na 2 časti. Na časť reprezentujúci grafického užívateľského rozhranie s podporou prihlasovania, užívateľských rol, zabezpečenia proti neautorizovanému prístupu. Rovnako je schopné zobrazovať úlohy, užívateľov a organizácie podľa užívateľskej role. Rozhranie pravidelne aktualizuje informácia o úlohách, užívateľoch a organizáciách z databázy.

Pre spustenie výpočtu úlohy komunikuje pomocou webovej služby s „PlannerService“ (optimalizovaná pre riešenie problému N Dám, vyváženie cloudu, plánovania prác a problém obchodného cestujúceho), ktorá implementuje spracovanie informácií. Pri požiadavke o spustenie/pozastavenie výpočtu spracovania úlohy sa predá v HTTP požiadavky ID úlohy. Webová služba následne zaradí požiadavok o spustení do JMS fronty. Message-driven bean-a následne postupne odoberá požiadavky z fronty a vyhodnocuje. Pritom najprv nájde potrebný XML definičný súbor v databáze a spustí výpočet pomocou OptaPlanner. Priebežné informácie (čas do skončenia plánovania, pokrok vo výpočte) sú priebežne vkladane do databáze, čo umožňuje užívateľovi prostredníctvom rozhrania sledovať stav úlohy. Pozastavenie úlohy dôjde prostredníctvom zmeny stavu vo webovej službe, čo pozastaví plánovanie.



Obr. 4.1: Diagram komunikácie

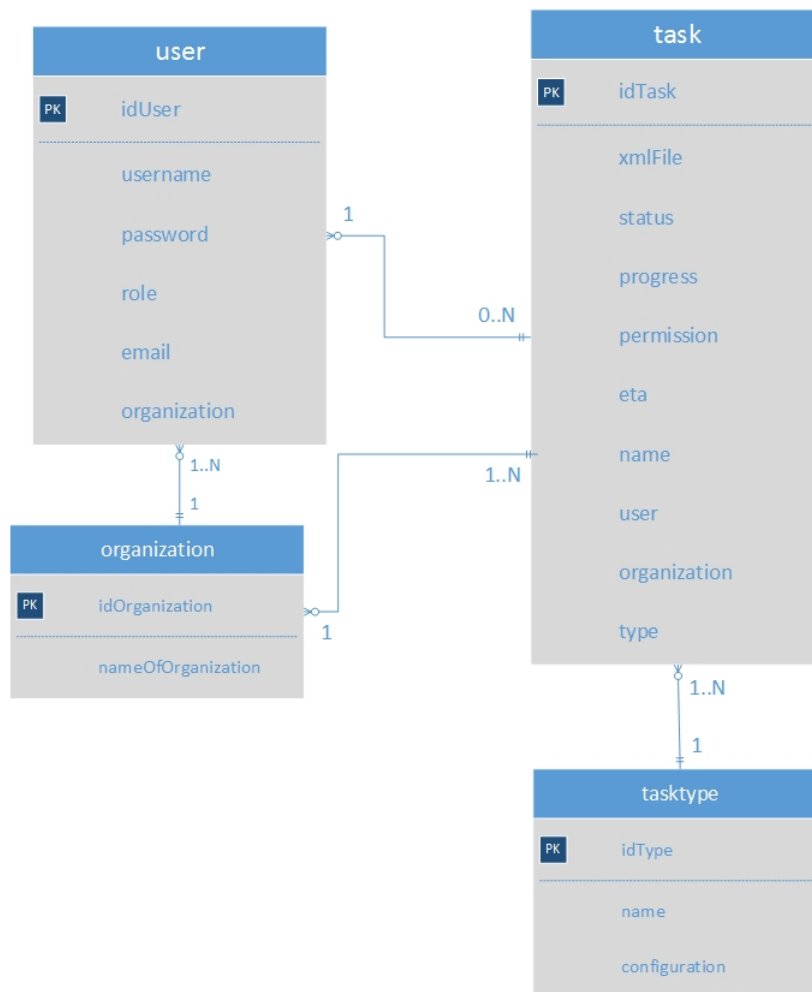
Na obrázku č.4.1 je popísaný spôsob komunikácie užívateľského rozhrania s PlannerService. 1.krokom je vytvorenie XML súboru plánovacieho problému prostredníctvom užívateľského rozhrania a následne uloženie definície do databázy. 2.krok je zaslanie žiadosti s ID úlohy o spustenie/zastavenie prostredníctvom webovej služby PlannerService(OptaPlanner), ktorý žiadosť spracuje. Ten v 3. kroku získa z databázy potrebný definičný XML súbor. Následne sa spustí plánovanie a priebežne sa ukladajú v kroku č. 4 informácie o pokroku úlohy, a čase ukončenia úlohy. Následne užívateľské rozhranie v kroku č. 5 pravidelne získava informácie o úlohe z databázy a zobrazuje ich v prehľadnej tabulke.

Celý návrh aplikácie bol otestovaný prostredníctvom skupiny odborných a laických užívateľov s cieľom zdôrazniť rýchlu učiacu sa krivku užívateľského rozhrania. Následne prebiehalo testovanie prostredníctvom užívateľov, ktorý testovali validáciu vstupov, prihlasovanie, správne vyhľadávanie jednotlivých entít(úloh, užívateľov, organizácií).

#### 4.3.1 Návrh modelu databázy

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 má určité položky. ER diagram sa skladá z 3 entít: user - entita, ktorá reprezentuje užívateľa, task - entita, ktorá reprezentuje úlohu a organization - entita, ktorá reprezentuje organizáciu a tasktype - entita, ktorá reprezentuje typ problému. 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 organizácii pre zlepšenie rýchlosti získania výsledkov a zjednodušenia ich nájdenia. Každá entita obsahuje



Obr. 4.2: ER diagram

primárny kľúč(jedná sa o silné entitné množiny), ktorý je odvodený od názvu a začína predponou „id\_“ a pokračuje názvom entity s CamelCase notáciou(každé slovo začína veľkým písmenom a slová sú spojené dokopy). Poďme sa pozrieť bližšie na jednotlivé entity. Entitná množina `tasktype` obsahuje 3 položky a to `idType`, ktorá reprezentuje primárny kľúč, `name`, ktorý reprezentuje názov plánovacieho problému(napr. N Dám, problém vyváženia cloudu, ...) a `configuration`, ktorá reprezentuje XML konfiguráciu pre daný typ problému.

Entitná množina `organization` obsahuje 2 položky jednou z nich je primárny kľúč a ďalšou názov organizácia podľa, ktorej sú zaraďovaní 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 odkaz na organizáciu, ku ktorej je užívateľ priradený. Nakoniec prejdime k entitnej množine `task`. Táto entitná množina obsahuje primárny kľúč, ďalej obsahuje XML súbor, ktorý reprezentuje danú úlohu(v našom prípade N dám), stav úloh(`stateOfTask`, ktorý reprezentuje rôzne stavy úlohy), ktorý si podrobnejšie rozobereme. Úloha sa môže nachádzať v jednom z nasledujúcich stavov:

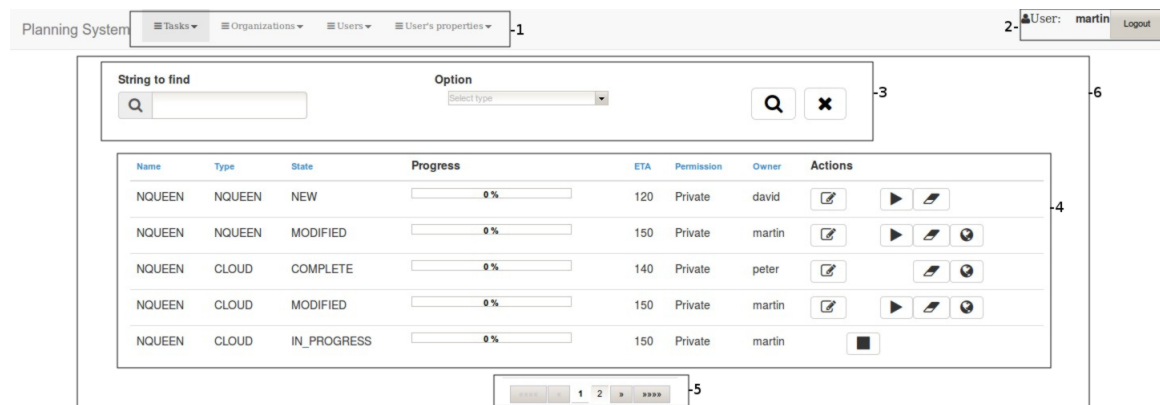
- NEW - úloha bola vytvorená

- 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.

### 4.3.2 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. Užívateľské rozhranie je popísané na nasledujúcom obrázku: Na obrázku č.4.3 môžeme vidieť návrh užívateľského rozhrania.



Obr. 4.3: Návrh užívateľského rozhrania

Rozhranie je rozdelené do 6 častí, ktoré môžeme rozoznať na obrázku číslami od 1 do 6, ktoré sú aj ohraňované. Celé rozhranie môžeme rozdeliť do nasledujúcich častí:

- Oblasť č.1 predstavuje navigačné menu, kde sú jednotlivé akcie rozdelené do rolovacích zoznamov. Pre kliknutie na príslušný zoznam dôjde k jeho odrolovaniu a zobrazeniu položiek. Po kliknutí na položku dôjde k zmene obsahu stránky.
- 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 tabuľky nižšie a naplní ju nájdenými výsledkami.
- Oblasť č.4 predstavuje tabuľky, 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 lexikografickému 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 pre vytvorenie úlohy, editovanie úlohy, spravovanie užívateľov, spravovanie organizácií, zmenu hesla a prihlasovanie je možné dohľadať v prílohe.

## Kapitola 5

# Implementácie

Nasledujúca kapitola pojedná o oboch častiach systému pre monitorovanie stavu plánovacích úloh. Najprv rozobereme aplikáciu pre užívateľské rozhranie 5.1 a následne Planner-Service 5.2, ktorá zabezpečuje riešenie plánovacích úloh prostredníctvom OptaPlanner-u. Pre technológiu MySQL bol zvolený MySQL server vo verzii 5.5.37. Obe časti boli založené na nástroji maven s použitím vývojového prostredia JBoss Developer Studio vo verzii 7.1.0 GA[5].

### 5.1 Aplikácie pre užívateľské rozhranie

Aplikácie pre užívateľské rozhranie, ktorá je schopná zobrazovať informácie o úlohách, užívateľoch a organizáciach a umožňovať ich správu. Základom tejto aplikácie je komunikácia s databázou. Komunikáciu zabezpečuje JBoss a to tak, že sa v súbore `persistence.xml` pre našu aplikáciu (*optaplanner.controller*) správne nastaví odkaz na `datasource` a definíciu entitných tried.

#### 5.1.1 Prihlasovanie

Prihlasovanie je realizované prostredníctvom frameworku Seam. Základom je vytvorenie komponent na XHTML stránke pre zadanie mena a hesla užívateľa. Tieto údaje sú spracované v backing bean-e (triede) s názvom *LoginBean*, ktorá je súčasťou balíku *org.jboss.optaplanner.controller.beans*. Táto trieda obsahuje aj validátory (metódy `validateUsername/validatePassword`), ktoré kontrolujú existenciu užívateľa a správnosť hesla v prípade, že existuje a podľa zistených informácií (existuje užívateľ/neexistuje, validné/nevalidné heslo) sa zobrazí komponenta *h:outputText*, ktorá obsahuje príslušný text.

V prípade, že validácia prebehne úspešne zavolá sa metóda *authenticate*, ktorá zabezpečí získanie užívateľskej role zadaného užívateľa, ktorú následne vloží do životného cyklu aplikácie pomocou metódy *setUser*, a to pomocou triedy *org.picketlink.idm.api.User* sa vloží ID užívateľa a užívateľská rola.

Navigácia užívateľa sa realizuje nastavením navigačných pravidiel v súbore *faces-config.xml*, do ktorého sa podľa užívateľskej role užívateľa nastaví hodnota premennej *isX(Admin/Planner/Reader)* na hodnotu `TRUE` a zabezpečí presmerovanie užívateľa na stránku podľa role:

- Rola `Administrator` bude presmerovaná na stránku `Administrator.xhtml`
- Rola `Planner` bude presmerovaná na stránku `Planner.xhtml`



- Rola Reader bude presmerovaná na stránku Reader.xhtml

Úspešné prihlásenie je dané nastavením metódy *setStatus* na hodnotu SUCCESS, v prípade, že validácia údajov neprebehne úspešne nastavíme prihlasovanie na neúspešné prostredníctvom metódy *setStatus* na hodnotu FAILURE. Po úspešnom prihlásení je možné identitu ľahko získať nainjektovaním(úvedením anotácie @Inject) triedy Identity, z ktorej je možné získať prihlasovacie meno užívateľa, ktorá sa zobrazuje na stránke.

Problematika odhlasovania úzko súvisí s prihlasovaním a je v podstate jednoduchá. Na XHTML stránke sa nachádza grafická komponenta *h:commandButton*, ktorá v atribúte action volá metódu logout pre príslušnú backing beanu. Tá spôsobí zavolanie metódy identity.logout, ktorá odobere identitu daného užívateľa(zamedzí mu opätovnú prístup k stránke podľa jeho role) a presmeruje ho na prihlasovacie stránku(Login.xhtml).

### 5.1.2 Zabezpečenie

Úzko s prihlasovaním súvisí aj problematika zabezpečenia aplikácie proti neautorizovanému prístupu. Teda povedzme užívateľ s rolou Planner by chcel prísť na stránku, ktorá je určená pre rolu Administrator. Aplikácia mu to nedovolí a v prípade o takýto pokus bude užívateľ presmerovaný naspäť na prihlasovaciu stránku. Princíp je taký nájden rozhranie, ktoré je anotovaná anotáciou @ViewConfig. Táto trieda obsahuje výpočtový typ, ktorý obsahuje anotácie @ViewPattern, ktorej obsah je stránka, na ktorú má byť povolený prístup. Pri každej takej anotácii je nachádza názov užívateľskej role uvedený prostredníctvom anotácie. Použitie názvu užívateľskej role je dané vytvorením špeciálneho rozhrania, ktoré je anotované anotáciou *SecurityBindingType*. Uvedenie anotovanej užívateľskej role vedľa stránky, ktorá má byť povolená spôsobí zavolanie triedy *Autorization*, ktorá je anotovaná anotáciou *Service*, ktorá overí vloženú identitu prostredníctvom metódy *authenticate* z triedy *LoginBean* a vráti odpoveď. Takýmto spôsobom sa povolí prístup pre užívateľskú rolu na danú stránku. Vedľa každej anotácie @ViewPattern sa nachádza aj anotácia *AccessDeniedView*, ktorá spôsobí presmerovaní na jej obsah, v prípade, že prihlásený užívateľ nemá danú užívateľskú rolu. Tento postup sa opakuje neustále v prípade pokusu o prístup k akejkoľvek stránke uvedenej rozhraní anotovaní anotáciou *ViewConfig*.

### 5.1.3 Komunikácie s PlannerService

Základom komunikácie s výpočtovou časťou systému PlannerService je vygenerovanie klienta z WSDL súboru webovej služby. Preto bolo potrebné vykonať nasledovné kroky:

- Nasadenie PlannerService na JBoss
- Zavolanie skriptu wsconsume.sh, ktorý je súčasťou aplikačného serveru s prepínačom -k a cestou k WSDL súboru
- Skopírovanie vygenerovaných tried do aplikácie pre užívateľské rozhranie do balíku *org.jboss.optaplanner.controller.service*

Po týchto krokoch sa v metóde *runTask* a *stopTask* volá metóda, ktorá vytvorí webovú službu a zavolá metódu runTask/pauseTask, ktoré sú súčasťou PlannerService s argumentom ID úlohy. Tieto metódy sú súčasťou backing bean pre užívateľskú rolu Planner a Administrator.

### 5.1.4 Logika aplikácie

Pre každú užívateľskú rolu bola vytvorená 1 XHTML stránka a backing beana a to nasledovne:

- Pre rolu Administrator je určená stránka *Administrator.xhtml* a backing beana(trieda) *AdministratorBean*
- Pre rolu Planner je určená stránka *Planner.xhtml* a backing beana(trieda) *PlannerBean*
- Pre rolu Reader je určená stránka *Reader.xhtml* a backing beana(trieda) *ReaderBean*

Všetky backing beany sú súčasťou balíka *org.jboss.optaplanner.controller.beans*. Aby beany mohli byť správne používané je potrebné ich uviesť v súbore *faces-config.xml*. To sa urobí uvedením medzi značky *managed-bean*, kde uvedieme názov beany, triedu vrátane cesty v hierarchii balíkov a typ beany(ktorý bol zvolený na session). Backing bean-y obsahujú metódy a vlastnosti, ktoré bola zobrazované/prevzaté z komponent na .xhtml stránkach. Všetky vlastnosti museli spĺňať princíp POJO. Pre potreby získavania dát z databáze bola použitá trieda *databaseOp*, ktorá je súčasťou balíku

*org.jboss.optaplanner.controller.database*, pričom vytvára inštanciu triedy *EntityManager*, ktorý využíva entitné triedy. Táto trieda obsahuje metódy na vytváranie úloh, užívateľov, organizácií, rovnako aj mazanie, editovanie jednotlivých položiek, rovnako aj získavanie. Tieto dáta sú následne predávané backing bean-ám podľa potreby.

### 5.1.5 Implementácia rozhrania

Pre implementáciu rozhrania bola použitá technológia XHTML stránok. Pre každú užívateľskú rolu bola vytvorená XHTML stránku identitická s názvom užívateľskej role. Pre prihlasovanie bola použitá stránka *Login.xhtml* stránka. Na *Login.xhtml* boli umiestnené komponenty na zadanie užívateľského mena a hesla vrátane skrytých validačných komponent. Na tejto stránke boli použité správne CSS frameworky na zabezpečenie prenositeľnosti na mobilné zariadenia a zároveň poskytlí užívateľskú prívetivosť.

Pri implementácii XHTML stránok 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. 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 z databáze, čo zabezpečuje ich aktuálnosť prostredníctvom komponenty *a4j:poll*, ktorá je vytvorená pre každú tabuľku a pravidelne volá metódu, ktorá získava údaje z databáze. 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 vyhľadávaní sa preruší obnovenie obsahu tabuliek a zobrazí sa informácie o vyhľadávanom reťazci a časovom razítke kedy bolo vyhľadávanie realizované. S vyhľadanými položkami je rovnako možné realizovať všetky akcie. Pre potreby opätovného obnovenia obsahu tabuľky je potrebné stlačiť tlačidlo pod tabuľkou s názvom

*showX(Users, Tasks, Organizations)*, ktoré sa nachádza na bielom páse pre rýchlejšie zorientovanie užívateľa. Toto tlačidlo spôsobí pre danú tabuľku(*users, tasks, organizations*) získanie aktuálnych dát z databázy zavolaním metódy z triedy *databaseOp* *getAllX(Tasks/Users/Organizations)*, ktoré vytvoria dotaz na získanie aktuálnych dát z databázy. Tieto dáta sú predané príslušnej tabuľke a zároveň sa obnoví obnovovanie obsahu tabuliek.

Pri každej položke v tabuľke je možné vykonávať isté akcie ako je vymazať danú entitu(*task, user, organization*), 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 zápäti rozoberem) je možné entity aj vytvárať. Vytváranie je veľmi jednoduché, keď užívateľ vyplní všetky polia, ktoré musí mať daná entita sa zavolá metóda z *backing bean*(napr. na editovanie *editTask*, na vytvorenie organizácie *createOrganization, ...*), ktoré spôsobia zavolanie metódy z triedy *databaseOp*, ktoré zabezpečia vytvorenie novej entity.

Každú tabuľku je možné aj radiť. Radenie prebieha kliknutím na názov stĺpca tabuľky(zvýraznený modrou farbou), pričom daný stĺpec implementuje funkciu radenia pre daný stĺpec. Pri kliknutí na názov stĺpca dôjde k zavolaniu metódy(napr. pre stĺpec ID sa zavolá metóda *sortById*), ktorá je daná atribútom *action* v grafickej komponente *h:commandLink*. Metóda radenia je implementovaná prostredníctvom triedy *Collections*, ktorá obsahuje metódu *sort*, ktoré triedia model(trieda, ktorá obsahuje rovnaké položky ako príslušná databázová tabuľka) danej entity, ktorá vytvorí komparátor, ktorý porovná 2 položky daného modelu a upraví ich poradie.

Vytváranie úloh(*taskov*) je zaradené do samostatnej záložky kvôli lepšej prehľadnosti. Užívateľ vyplní názov a prostredníctvom komponenty na nahrávanie súboru z knižnice Rich Faces nahrať obsah do databázy. Ďalej rozoberem záložku *change password*, ktorá umožňuje si pre daného užívateľa zmeniť heslo, vyplní pritom heslo a potvrdenie hesla a heslo sa zmení. Nakoniec rozoberem záložku *editTask*, táto záložka je pri bežnom prehliadaní neviditeľná je to spravené kvôli bezpečnosti. Táto záložka sa aktivuje editovaním úlohy v záložke *task* v tabuľke tlačidlom *Edit Task*, ktorá nás prepne do záložky *Edit Task*, 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“.

### 5.1.6 Publikovanie úloh

Ďalšou podstatnou časťou aplikácie pre užívateľské rozhranie je možnosť publikovať/odpublikovať úlohu(*task*). Túto akciu je možné realizovať prostredníctvom tlačidla v tabuľke *Publish Task/Unpublish Task*. Tieto tlačidlá nie sú vždy prístupné, podmienkou je, že úloha je nastavená ako privátna a nachádza v stave „MODIFIED“ alebo „COMPLETE“. Naopak odpublikovanie úlohy je možné kedykoľvek podmienkou je, aby úloha bola nastavená ako verejná(*public*). Publikovanie je realizované zavolaním metódy „*publishTask*“. V tejto metóde dôjde k zmene stavu úlohy na verejnú, pričom informácia sa uloží do databázy. Následne sa v stĺpci *permission* zobrazí text *Public* modrou farbou, ktorý po kliknutí zobrazí názov úlohy a XML súbor plánovacej úlohy. Pričom po kliknutí sa prejde na odkaz *url aplikácie/task/id úlohy*. Na časť URL *task/id* je namapovaná rest webová služba, ktorá je súčasťou balíku *org.jboss.optaplanner.controller.restservice*, kde sa nachádza trieda *RESTPublishTask*, ktorá reprezentuje práve túto službu starajúcu sa o publikovanie úloh. Táto úloha obsahuje v anotácii *@Path()* len znak „/“, čo znamená že sa namapuje na akékoľvek URL, ale namapovanie na reťazec *task* sa realizuje v súbore *web.xml*. Táto služba obsahuje 1 metódu „*getUserById*“, ktorá dostane ako parameter ID úlohy. Toto id

úlohy získa zo zadaného URL. Dôležitou anotáciou je anotácia `@Produces()`, ktorá obsahuje hodnotu „text/html“, ktorá hovorí, že vrátená odpoveď metódy bude HTML súbor a teda výsledok bude zobrazený v prehliadači. Táto metóda na svojom začiatku vytiahne informácie o úlohe(názov, XML súbor a povolenie). Na základe povolenia určí, či je úloha nastavená ako verejná, ak nie je vráti prázdnu stránku. V prípade, že je úloha verejná vráti stránku, ktorá obsahuje informáciu o názvu úlohy a XML súbor. Prístup k tomuto k tejto službe nie je podmienený prihlasovaním.

### 5.1.7 Validácia

Všetky grafické komponenty obsahujú validáciu na neprázdne, niektoré aj na nevalidné komponenty. Všetky komponenty, do ktorých sa zadáva nejaká informácia je realizovaná grafickou komponentou *h:inputText*, ktoré spracovávajú užívateľské vstupy. Každá komponenta obsahuje atribút *required* nastavenú na hodnotu *true*, ktorá spôsobí automatickú validáciu v prípade nezadanej hodnoty. Každá komponenta obsahuje aj atribút *requiredMessage*, ktorý ako hodnotu obsahuje reťazec, ktorý sa zobrazí v prípade, že nie je zadaná hodnota. Rovnako obsahuje aj atribút *ID* s nejakou jedinečnou hodnotou pre identifikáciu komponenty. Aby informácia o nezadaní bola zobrazená je potrebné vytvoriť komponentu *h:message*, ktorá obsahuje atribút *for*, ktorý obsahuje *id h:inputText* komponenty, pre ktorú má byť správa zobrazená. Niektoré komponenty(napr. validácia prihlásenia) sú validované na základe validátorov, ktoré obsahujú odkaz na metódy v triede *LoginBean*, ktoré v prípade potreby nastavujú zobrazenie komponenty *h:outputText* prostredníctvom nastavenia atribútu *rendered* na hodnotu *true*, pričom táto komponenta obsahuje text podľa danej situácie(napr. neznámy užívateľ, nevalidné heslo). V opačnom prípade je komponenta skrytá, teda hodnota atribútu je nastavená na hodnotu *false*.

## 5.2 PlannerService

*PlannerService* predstavuje časť systému pre monitorovania, ktorá zabezpečuje spracovanie požiadavok od aplikácie z užívateľského rozhrania. Základom je trieda *OptaPlannerWebService*, ktorá predstavuje „Big“ webovú službu, ktorá je súčasťou balíku *org.jboss.optaplanner.service.server*. Tá je špeciálne anotovaná prostredníctvom anotácie `@WebService`, ktorá označuje, že daná trieda je webovou službou. Tá obsahuje 2 metódy „*runTask*“ a „*pauseTask*“, ktoré sú anotované anotáciou `@WebMethod`, a sú teda prístupné a môžu byť vzdialene zavolané klientom(aplikáciou pre užívateľské rozhranie). Ako argument obsahujú metódy hodnotu typu *long*, ktorá predstavuje ID plánovacej úlohy, s ktorou sa má daná akcia vykonať.

Druhou dôležitou triedou je *OptaPlannerMessageBean*, ktorá je súčasťou balíku *org.jboss.optaplanner.service.server*. Táto trieda predstavuje Message-driven bean-u a je anotovaná anotáciou `@MessageDriven` a pre prijímanie požiadaviek využíva JMS frontu s názvom *OptaPlanner*.

Základným predpokladom je korektné nastavenie súboru *persitence.xml*, ktorý odkazuje na entitné triedy a odkaz na *datasource*.

Princíp fungovania tejto časti systému je nasledujúca:

- Klient(aplikácia užívateľského rozhrania) zadá požiadavku na spustenie/pozastavenie úlohy, ktorú realizuje zavolaním metódy webovej služby *PlannerService* s ID plánovacej úlohy

- Metóda `runTask` `PlannerService` vytvorí spojenie s triedou `OptaPlannerMessageBean`, a následne vloženie správy s ID úlohy do JMS fronty `OptaPlanner`
- Trieda `OptaPlannerMessageBean`, ktorá je Message-driven bean tá obsahuje odkaz rovnako na JMS frontu `OptaPlanner`. Tá obsahuje metódu `onMessage`, ktorá zabezpečuje spracovanie správ z JMS fronty `OptaPlanner` správu po správe
- Metóda `onMessage` získa zo správy ID úlohy, ktorú následne získa z databáze, zmení stav úlohy na `IN_PROGRESS` a vytvorí inštanciu triedy `ProblemSolver`, ktorej konštruktor dostane ako parameter XML súbor plánovacej úlohy z databáze. Táto trieda je súčasťou balíku `org.jboss.optaplanner.service.solver`. Po vykonaní tohto kroku sa spustí metóda `run`, ktorá nastaví konfiguráciu pre riešenie problému N Dám. Následne sa zavolá metóda `execute`, ktorá spustí výpočet(riešenie).
- V message-driven bean sa v cykle neustále získava skóre z triedy `ProblemSolver` na základe, ktorého sa počítajú hodnoty do času skončenia úlohy a pokroku, ktoré sa zapisujú do databáze.
- Po ukončení/vyriešení úlohy sa z triedy `ProblemSolver` získa najlepšie riešenie(XML predpis), ktorý sa uloží do databáze s informáciami o dokončení úlohy(zmení stav úlohy na `COMPLETE`)
- Tento postup sa opakuje pre všetky správy v JMS fronte `OptaPlanner`

Na koniec spomeniem princíp fungovania metódy na pozastavenie vykonávania plánovania:

- Klient(aplikácie užívateľského rozhrania) zavolá metódu `pauseTask` s argumentom ID úlohy, ktorá sa má pozastaviť
- Metóda `pauseTask` triedy `OptaPlannerWebService` spôsobí zmenu stav úlohu na `PAUSE`
- Message-driven bean v metóde `onMessage` neustále obnovuje hodnoty o priebehu vykonávani funkcie a podmienke cyklu je podmienka, že úloha je nastavená na stav `IN_PROGRESS`. Teda v prípade zmeny stavu sa cyklus zastaví a teda aj výpočet(riešenie)

## 5.3 Testovanie

Testovanie prebiehalo na servery JBoss AS 7.1.1 Final najprv prostredníctvom jednoduchých JUnit testov, ktoré malo overiť komplikovanú funkčnosť metód. Následne sa pre overenie funkčnosti databáze 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.

V ďalšej časti prebiehalo testovanie medzi konkrétnymi užívateľmi. Išlo o 4 informatiky skúsených užívateľov a 4 laikov. Užívatelia testovali celkovú funkčnosť aplikácie a

hľadali prípadné chyby, ktoré neodhalilo predošlé testovanie. Aplikácia bola vložená na cloudovú službu OpenShift, ktorá umožnila prístup k aplikácii prostredníctvom internetu. Následne bol skupine užívateľov predložený odkaz na nasadenú aplikáciu a prihlasovacie údaje k užívateľovi s rolou Administrator, Planner a Reader.

Užívatelia nasledne 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. Na záver zhrniem testy, ktoré boli užívateľmi realizované:

- Overenie funkčnosti prihlasovania s validnými/nevalidnými údajmi
- Overenie funkčnosti záložky task(úloh) - mazanie úloh, editovanie úloh, vyhľadávanie úloh vrátane validácie, publikovanie/odpublikovanie úloh, navigácia medzi stránkami úloh tabuľky, pri editovaní úlohy sa overovalo skrytie záložky edit task pri kliknutí na inú záložku, radenie úloh podľa všetkých stĺpcov
- Overovanie funkčnosti záložky user(užívateľ) - vytváranie nového užívateľa s validnými-/nevalidnými údajmi, vyhľadávanie užívateľov vrátane zadania nevalidných údajov, mazanie užívateľov, editovanie informácií o užívateľoch, zmena hesla užívateľovi
- Overenie funkčnosti záložky organization(organizácia) - vytváranie organizácie, vrátane vyhľadávania s validnými/nevalidnými údajmi, radenie organizácii, mazanie organizácií, editovanie názvu organizácie
- Overenie funkčnosti záložky changepassword(zmena hesla) - zmenu hesla s validnými-/nevalidnými údajmi pre aktuálne prihláseného užívateľa
- Testovanie užívateľskej prívetivosti rozhrania skúsenými a laickými užívateľmi, rovnako otestovanie užívateľského rozhrania na mobilnom telefóne

Rovnako boli užívateľom predložené XML súbory pre riešenie problému N Dám v rozložení pri 4,8,16 dām. Užívatelia nahrali tieto súbory do systému a sledovali priebeh riešenia plánovacieho problému prostredníctvom PlannerService.

Užívateľské rozhranie bolo otestované pre mobilné telefóny na zariadení HUAWEI Honour 2 s prehliadačom Mozilla Firefox, v ktorom sa zobrazovalo korektne.

## 5.4 Vyhodnotenie aplikácie

Po testovacej fáze nasledovala fáza vyhodnotenia aplikácie. Cieľovej skupine bol po opravení chýb aplikácie predložený dotazník, do ktorého výplňami rôzne informácie, kde dávali spätnú väzbu, chyby v návrhu, rovnako aj v intuitívnosti ovládania. Cieľovou skupinou aplikácie sú užívatelia bez akejkoľvek predchádzajúcej skúsenosti s touto aplikáciou s vekým rozsahom medzi 20 - 40 rokov. Preto bola aplikácia predložená najprv užívateľom skúseným, ktorým bol poskytnutý predchádzajúci styk s aplikáciou a laickým užívateľom, ktorý nemali žiadny predchádzajúci styk. Výsledkom zistenia, rovnako vyplývajúce z výsledkou dotazníka je že užívateľské rozhrania až na niektoré časti je veľmi intuitívne. Užívatelia sa ihneď vedeli zorientovať a vykonať danú akciu, vytvoriť užívateľa, organizáciu, úlohu. Problém na, ktorý

narazili bolo zorientovať pri vyhľadávaní úloh/užívateľov/organizácií a nájsť tlačidlo pod tabuľkou a obnoviť všetky údaje v tabuľke. Rovnako oceňovali možnosť zobrazovanie tlačidla *Save Changes* pri editovaní tabuľky vedľa položky, ktorá je práve editovaná v danom riadku. Pri vyhľadávaní ocenili používatelia zachovania zadaných informácií pre vyhľadávanie. V záložke user management (správa užívateľov) navrhovalo presunutie tabuľky s užívateľmi na začiatok, keďže sa nachádzala na neprehľadnom mieste. Užívatelia ocenili možnosť radenia tabuliek po kliknutí na daný stĺpec aj spôsob realizácie. Užívatelia by ocenili pri úlohách mať možnosť informácie o časovom razítke vytvorení úlohy. Prehliadania pomocou tabuliek im prišlo ako veľmi vhodné rovnako aj použitie stránkovania. Užívateľom chýbala možnosť vyhľadávať podľa viacerých kritérií.

Aplikácia by mohla byť upravená do užívateľsky prívetivejšieho rozhrania a mohli byť zahrnuté všetky názory užívateľov. Rovnako by PlannerService mohla byť rozšírená o spracovanie aj iných typov plánovacích úloh, minimálne tie, ktoré sú podporované štandardnou implementáciou frameworku OptaPlanner, napr. plánovanie práce, problém obchodného cestujúceho, . . . . Rovnako použité databázové technológie umožňujú rozšírenie databázovej schémy pomocou entitnej a jej následné vygenerovanie.

## 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 odbremenuje 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, rovnako sú predstavené možnosti rozšírenia rozhrania a urobenie tohto rozhrania oveľa užívateľsky prívetivejšie a efektívnejšie. 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 najvhodnejší java EE kontajner vzhľadom na použité technológie. 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 aplikácia pre užívateľské rozhranie rozdelená do viacerých balíkov.



# Literatúra

- [1] Ament, J. D.: *Arquillian Testing Guide*. Packt Publishing, 2013, ISBN 978-1782160700.
- [2] Andrea Steelman, J. M.: *Murach's Java Servlets and JSP*. Mike Murach & Associates, 2008, ISBN 978-1890774448.
- [3] Ary, J.: *Instant Drools Starter*. Packt Publishing, 2013, ISBN 978-1782165545.
- [4] Bali, M.: *Drools JBoss Rules 5.X Developer's Guide*. Manning Publications, 2007, ISBN 978-1933988344.
- [5] Company, S.: *Maven: The Definitive Guide*. O'Reilly Media, 2008, ISBN 978-0596517335.
- [6] David Geary, C. S. H.: *Core JavaServer Faces (3rd Edition)*. Prentice Hall, 2010, ISBN 978-0137012893.
- [7] Debu Panda, D. L., Reza Rahman: *EJB 3 in Action*. Packt Publishing, 2013, ISBN 978-1782161264.
- [8] Felke-Morris, T.: *Web Development and Design Foundations with XHTML, 5th Edition*. Addison-Wesley, 2010, ISBN 978-0132122702.
- [9] Gandy, D.: Font Awesome [online].  
<http://fortawesome.github.io/Font-Awesome/>, 2014-11-03 [cit. 2013-10-25].
- [10] III, A. T. H.: *Ajax: The Definitive Guide*. O'Reilly Media, 2008, ISBN 978-0596528386.
- [11] Javid Jamae, P. J.: *JBoss in Action: Configuring the JBoss Application Server*. Manning Publications, 2008, ISBN 978-1933988023.
- [12] Jendrock, E.; Cervera-Navarro, R.; Evans, I.; aj.: The Java EE 6 Tutorial [online].  
<http://docs.oracle.com/javaee/6/tutorial/doc/>, 2013-11-03 [cit. 2013-10-25].
- [13] Kaczanowski, T.: *Practical Unit Testing with JUnit and Mockito*. Tomasz Kaczanowski, 2005, ISBN 978-1584504184.
- [14] Kathy Sierra, B. B.: *Head First Java*. O'Reilly Media, 2005, ISBN 978-0596009205.
- [15] Mark Richards, D. A. C., Richard Monson-Haefel: *Java Message Service*. O'Reilly Media, 2009, ISBN 978-0596522049.
- [16] Merrick Schincariol, M. K.: *Pro JPA 2*. Apress, 2013, ISBN 978-1430249269.

- [17] Michael Juntao Yuan, T. H.: *JBoss Seam: Simplicity and Power Beyond Java EE*. Prentice Hall, 2007, iISBN 978-0131347960.
- [18] Piroumian, V.: *Test Driven Development: By Example*. Addison-Wesley Professional, 2002, iISBN 978-0321146533.
- [19] Skiena, S. S.: *The Algorithm Design Manual*. Springer, 2012, iISBN 978-1849967204.
- [20] Thilo, M. O. J. T. C. R. J.: Twitter Bootstrap [online].  
<http://getbootstrap.com/>, 2013-12-16 [cit. 2013-12-27].

## Dodatok 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 <http://jbossas.jboss.org/downloads>
- MySQL Connector/J minimálne vo verzii 5.0.8, ktorý je súčasťou CD
- MySQL databázový server najmenej vo verzii 5.5.37(na distribúcií ubuntu je možné nainštalovať príkazom „apt-get install mysql-server mysql-client“, alebo je možné postupovať podľa nasledujúceho návodu <http://dev.mysql.com/doc/refman/5.1/en/linux-installation.html>)
- Webový prehliadač Mozilla Firefox najmenej vo verzii 29.0 alebo Google Chrome najmenej vo verzii 34.0(v prostredí ubuntu je možné ho nainštalovať nasledujúcim príkazom „apt get install firefox/google-chrome“)
- Nástroj Maven(môžeme ho nainštalovať príkazom v prostredí ubuntu „sudo apt-get install maven“ alebo využiť odkaz s popisom <http://maven.apache.org/download.cgi>)
- Verzovací nástroj Git (môžeme ho nainštalovať príkazom v prostredí ubuntu „sudo apt-get install git“ alebo využiť odkaz s popisom <http://git-scm.com/book/en/Getting-Started-Installing-Git>)

V prvom rade je potrebné nainštalovať MySQL server nakonfigurovať databázu s názvom „optaplanner“ s užívateľským menom „root“ a heslom „root“. Následne je potrebné rozbaľiť stiahnutý JBoss aplikačný server na súborový systém. V ďalšom kroku nastaví náš aplikačný server pre správne použitie MySQL databáze. To urobíme zkopírovaním súboru standalone-full.xml do adresára JBOSS\_HOME<sup>1</sup>/standalone/configuration a prepíše aktuálny obsah súboru.

Následne treba ovládač mysql-connector-java-5.1.29-bin.jar umiestniť 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)

---

<sup>1</sup>JBOSS\_HOME predstavuje koreňový adresár serveru JBoss na disku

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

Nakoniec skopírujeme súbor `standalone.conf` do adresára `JBOSS_HOME/bin` a prepíšeme aktuálny obsah súboru. Nakoniec získame aplikáciu `PlannerService`. Je potrebné aplikáciu stiahnuť a vytvoriť, to urobíme nasledovne, ale predtým je potrebné nainštalovať závislosť z aplikácie `optaplanner.controller`:

1. Nastavíme sa do koreňového adresára `optaplanner.controller` a následne do zložky `Entities`
2. Zadáme príkaz „`mvn clean package`“ a následne „`mvn install`“
3. Následne zadáme príkaz „`git clone https://github.com/marvec/PlannerService.git`“
4. Nastavíme sa do zložky pomocou príkazu „`cd PlannerService`“
5. Zadáme príkaz „`mvn clean package`“

Aplikáciu môžeme spustiť nasledovne:

- Skopírujeme pripravený adresára `optaplanner.controller.war` do adresára `JBOSS_HOME/standalone/deployments`
- Skopírujeme súbor `optaplanner-service.war` z adresára, kde sme spustili príkaz „`mvn clean package`“ a nastavíme sa v ňom do podadresára „`target`“. Odtiaľ skopírujeme súbor `optaplanner-service.war`<sup>2</sup> do adresára `JBOSS_HOME/standalone/deployments`
- Prejdeme do zložky `JBOSS_HOME/standalone/bin` a spustíme skript `standalone.sh`

K aplikácií pristúpime zadaním adresy „`http://localhost:8080/optaplanner.controller.war/`“ do webového podporovaného webového prehliadaču a môžeme sa prihlásiť do systému nasledovne:

- Pre užívateľskú rolu Administrátor - prihlasovacie meno: martin, heslo:martin
- Pre užívateľskú rolu Plánovač - prihlasovacie meno:peter heslo:peter
- Pre užívateľskú rolu Čitateľ - prihlasovacie meno:david heslo:david

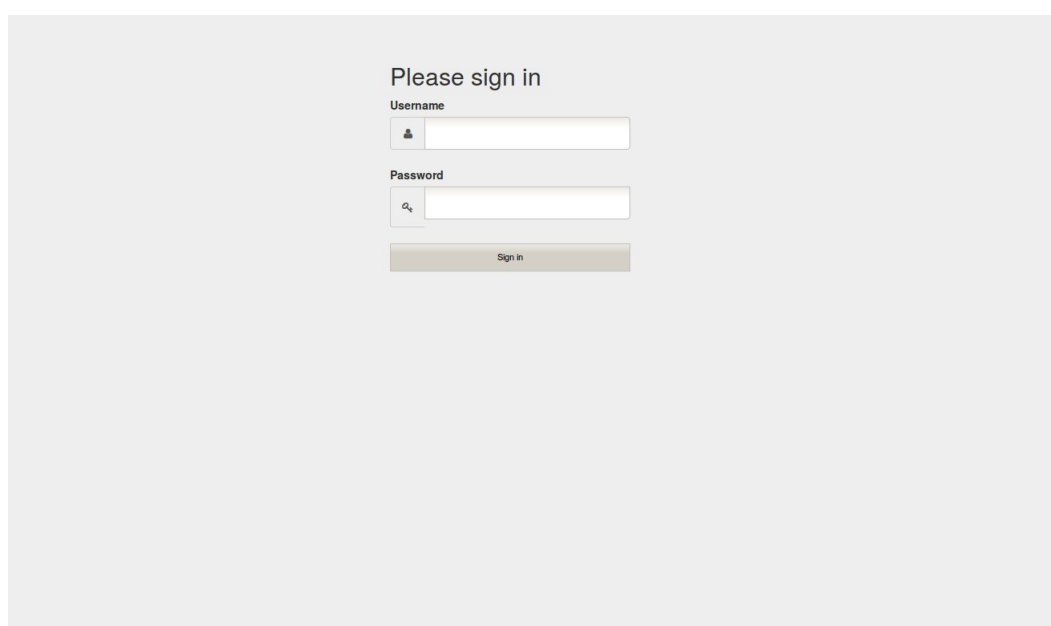
Aplikáciu `optaplanner.controller` je možné preložiť príkazom v jej zložke príkazom „`mvn clean package`“ a následne zabaliť do súboru `war` pomocou príkazu „`mvn war:war`“.

---

<sup>2</sup>optaplanner-service.war predstavuje PlannerService

## Dodatok B

# Užívateľské rozhranie



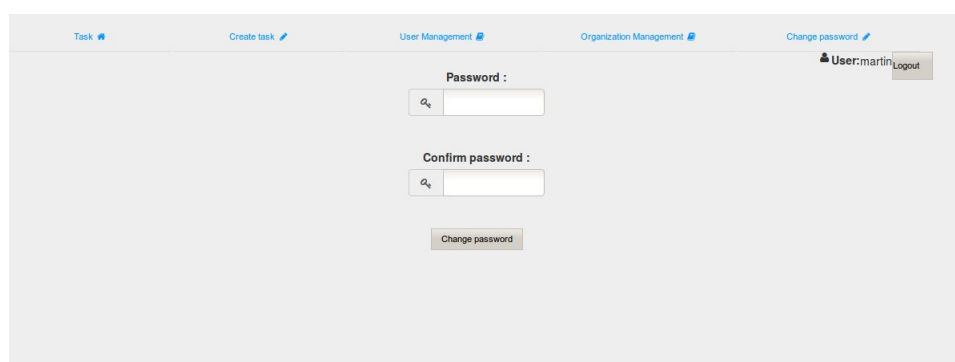
Please sign in

Username

Password

Sign in

Obr. B.1: Prihlasovacia obrazovka



Task Create task User Management Organization Management Change password

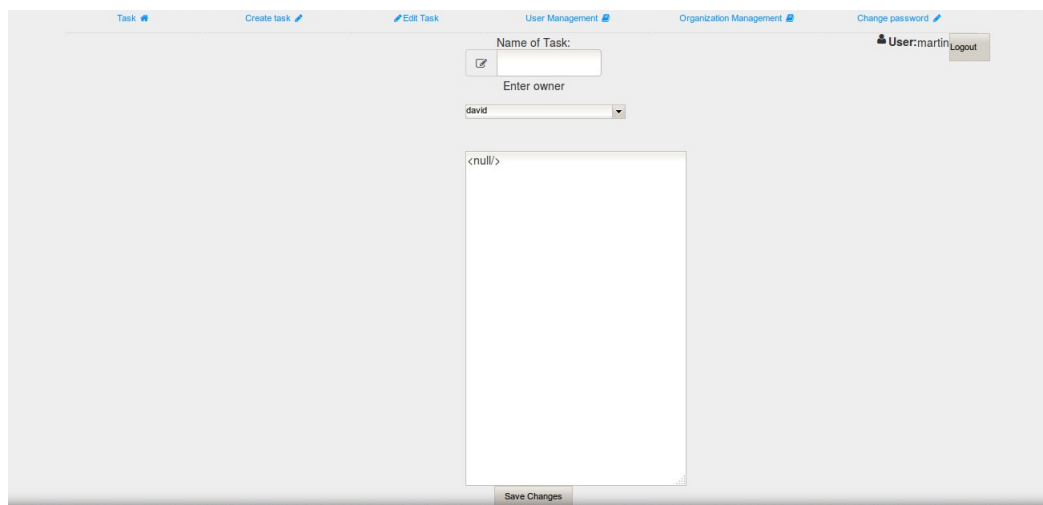
User: martin Logout

Password :

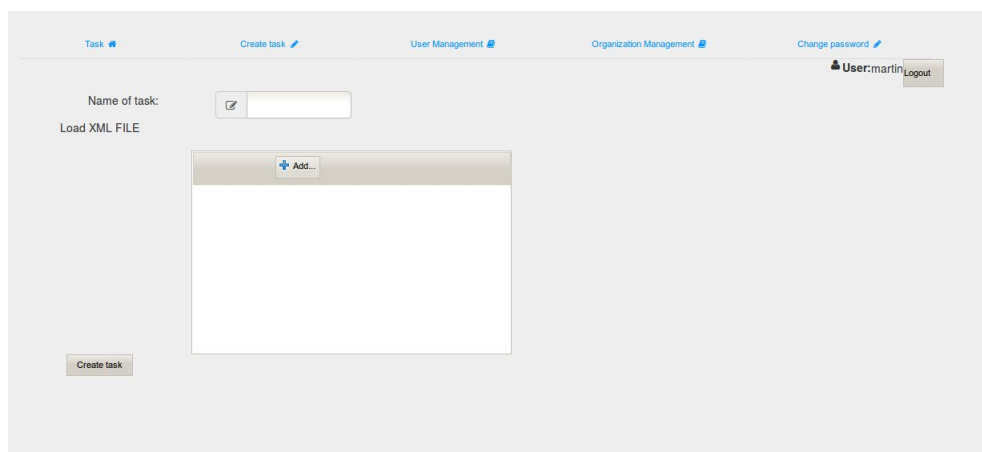
Confirm password :

Change password

Obr. B.2: Zmena hesla



Obr. B.3: Editovanie úlohy



Obr. B.4: Nahrávanie novej úlohy

The screenshot shows the 'User Management' section of a web application. At the top, there are navigation links: 'Task', 'Create task', 'User Management', 'Organization Management', and 'Change password'. The 'User Management' link is active. On the right, a user profile 'User: martin' is shown with a 'Logout' button.

The main area contains a form to 'Create User'. It has two columns of input fields. The left column has 'Enter String to find:' and 'Select option to find:' (with a dropdown menu). The right column has 'Username', 'Password', 'Re-type password', 'Email', 'Organization' (dropdown), and 'Role' (dropdown). A 'Find' button is between the columns. Below the form is a 'Create User' button.

Below the form is a table of existing users:

Username	Email	Role	Organization	Action
martin	martin@martin.cz	Administrator	dsadsad	<a href="#">Edit User</a> <a href="#">Delete User</a> <a href="#">Change Password</a>
david	david@david.cz	Reader	gdfgdfg	<a href="#">Edit User</a> <a href="#">Delete User</a> <a href="#">Change Password</a>
peter	peter@peter.cz	Planner	Oracle	<a href="#">Edit User</a> <a href="#">Delete User</a> <a href="#">Change Password</a>
martin3	martin@martin.cz	Administrator	dsadsad	<a href="#">Edit User</a> <a href="#">Delete User</a> <a href="#">Change Password</a>
david3	david@david.cz	Reader	gdfgdfg	<a href="#">Edit User</a> <a href="#">Delete User</a> <a href="#">Change Password</a>

At the bottom, there is a pagination control showing '1 2' and a 'Show all users' button.

Obr. B.5: Spravovanie užívateľov

The screenshot shows the 'Organization Management' section of a web application. At the top, there are navigation links: 'Task', 'Create task', 'User Management', 'Organization Management', and 'Change password'. The 'Organization Management' link is active. On the right, a user profile 'User: martin' is shown with a 'Logout' button.

The main area contains a form to 'Create organization'. It has two columns of input fields. The left column has 'Enter String to find:' and 'Select option' (with a dropdown menu). The right column has 'Organization' and 'Create organization' button. A 'Find' button is between the columns.

Below the form is a table of existing organizations:

ID Organization	Name of Organization	Action
1	dsadsad	<a href="#">Edit Organization</a> <a href="#">Delete Organization</a>
2	gdfgdfg	<a href="#">Edit Organization</a> <a href="#">Delete Organization</a>
3	Oracle	<a href="#">Edit Organization</a> <a href="#">Delete Organization</a>
4	Red Hat1	<a href="#">Edit Organization</a> <a href="#">Delete Organization</a>
5	IBM1	<a href="#">Edit Organization</a> <a href="#">Delete Organization</a>

At the bottom, there is a pagination control showing '1 2' and a 'Show all organization' button.

Obr. B.6: Spravovanie organizácií

## Dodatok C

# Dotazník

### C.1 Obsah dotazníka

**Grafické užívateľské rozhranie pre systém OptaPlanner**

Som študentom tretieho ročníka FIT VUT v Brne. Mojou bakalárskou prácou je tvorba užívateľského rozhrania pre Systém monitorovania plánovacích úloh. Ide o aplikáciu, ktorá dokáže spúšťať plánovanie pre problém N dām, nahrať XML súboru N dām a následne spustenie. Rovnako je možné úlohy editovať, mazať, vyhľadávať, alebo radiť. Užívateľ môžu rovnako spravovať iných užívateľov a organizácie.

V rámci aplikácie môžete 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

Chcel by som Vás požiadať o vyplnenie jednoduchého dotazníku, na základe ktorého môžem 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

Obr. C.1: Všeobecné informácie



### 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 zobraziť pri správe organizácií ?**

☐ Názov organizácie

☐ Jednoznačný identifikátor organizácie

☐ Čas vytvorenia organizácie

☐ Užívateľ, ktorý vytvoril organizáciu

☐ Other:

**Aké informácie by ste chceli zobrazovať pri úlohách?**

☐ Názov úlohy

☐ Jednoznačný identifikátor úlohy

☐ Čas do konca spracovania

☐ Percentuálne hodnotenie spracovania

☐ Kedy bola úloha spustená

☐ Stav úlohy

☐ Definičný súbor danej úlohy

☐ Other:

Obr. 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 numericky podľa ich jednoznačného číselného identifikátora

☐ Other:

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

	Nedôležité				Dôležité
Možnosť zoradovať úlohy podľa rôznych kritérií	<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 zobrazené úlohy podľa určitého kritéria	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Obr. C.3: Hodnotenie

**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 vyhľadavanie 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 prehliadaním 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"/>

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

**Aká funkcionálna vám chýbala?**

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

Obr. C.4: Záverečné hodnotenie

## Dodatok D

# CD so zdrojovými kódmi

Priložené CD obsahuje nasledujúce súbory:

- optaplanner.controller.war - adresár aplikácie pre užívateľské rozhranie obsahujúce preložené zdrojové súbory
- 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 bakáalarskej práce
- mysql-connector-java-5.1.29-bin.jar - ovládač pre prácu s MySQL databázou
- 4queens.xml, 8queens.xml, 16queens.xml - definičné súbory pre plánovací problém 4, 8 a 16 dām
- src - adresár, ktorý obsahuje zdrojové kódy k aplikáciám pre užívateľské rozhranie(optaplanner.controller)
- docs - adresár so zdrojovými textami k dokumentáciám