

# 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 vytvořit systém plánování, který uloží plánovací problém do systému, spustí jeho plánování a monitoruje průběh. Systém plánování byl rozdělen na části uživatelského rozhraní vytvořeného pomocí technologií JavaServer Faces, Rich Faces a Twitter Bootstrap, prostřednictvím kterého můžeme nahrávat zadání problému, spouštět a pozastavit běh úloh a druhou část reprezentovanou webovou službou v kombinaci s technologií Enterprise JavaBeans, která zpracovává požadavky na spouštění/pozastavení běhu plánování, vykována 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 Enterprise Edition 6 a aplikace byla nasazena na aplikační server JBoss. Systém byl odzkoušen na umělém plánovacím problému N Dam a praktickém problému ze společnosti Red Hat vyvážením cloudu a cestovatelském turnaji na platformě UNIX uživateli prostřednictvím cloudové služby OpenShift.

## Abstract

Task of this bachelor work is to create system for monitoring of statuses of planned jobs according to requirements defined by sponsor Red Hat. The main target is to understand what planning problem means, how can be defined, stored to system, how to initiate his planning and to monitor progress. Planning system has been split to the 2 parts. The first part is user interface build up based on technologies JavaServer Faces, Rich Faces a Twitter Bootstrap which allows recording, beginning and put on hold of running jobs. The second part represents web service with combination of technology Enterprise JavaBeans which processes requirements for beginning/ put on hold of planning run accomplishes with utilization of framework OptaPlanner, planning progress is being stored to MySQL database. Information about planning problems is obtained from this database and continuously is being displayed in user interface. For implementation both parts has been used platform Java EE 6 and has been deployed in Java EE container JBoss. System has been tested on artificial planning problem N Queen and practical problems from Red Hat componany cloud balancing and travelling tournament on platform UNIX via cloud service OpenShift by users have been trying to fix different planning problems.

## Klíčová slova

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

## Keywords

Java Enterprise Edition 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

24. 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>Implementácia</b>	<b>3</b>
1.1	Aplikácie pre užívateľské rozhranie . . . . .	3
1.1.1	Prihlasovanie . . . . .	4
1.1.2	Komunikácia s PlannerService . . . . .	4
1.1.3	Implementácia rozhrania . . . . .	5
<b>2</b>	<b>Úvod</b>	<b>6</b>
<b>3</b>	<b>Java Enterprise edition 6</b>	<b>7</b>
3.1	Špecifikácia platformy . . . . .	7
3.2	Trojvrstvový model . . . . .	7
3.2.1	JavaServer Faces . . . . .	9
3.2.2	Navigácia . . . . .	10
3.3	Webová služba . . . . .	11
3.3.1	Big webová služba . . . . .	12
3.3.2	RESTful webová služba . . . . .	12
3.4	Java Persistence API . . . . .	13
3.5	Enterprise JavaBeans . . . . .	14
3.5.1	Message-driven Bean . . . . .	14
3.5.2	Session Bean . . . . .	15
3.6	Možnosti spolupráce Javy EE s databázou . . . . .	15
3.7	Seam . . . . .	15
3.8	Ostatné technológie pre tvorbu užívateľského rozhrania . . . . .	16
3.9	Možnosti testovania v Jave EE . . . . .	17
3.10	JBoss Aplikačný server . . . . .	17
<b>4</b>	<b>OptaPlanner</b>	<b>18</b>
4.1	Plánovací problém . . . . .	18
4.2	Princíp . . . . .	19
4.3	Konfigurácia OptaPlanneru . . . . .	21
4.4	Výsledky plánovacieho problému . . . . .	23
4.4.1	Ukážku výsledku plánovacieho problému . . . . .	23
4.4.2	Štatistiky . . . . .	25
<b>5</b>	<b>Analýza a návrh aplikácie</b>	<b>27</b>
5.1	Špecifikácia požiadavkov . . . . .	27
5.2	Špecifikácia požiadavkov . . . . .	27
5.3	Analýza . . . . .	28

5.4	Návrh aplikácie . . . . .	30
5.4.1	Návrh modelu databáze . . . . .	31
5.4.2	Návrh užívateľského rozhrania . . . . .	32
5.4.3	Publikovanie úloh . . . . .	34
5.4.4	Validácia . . . . .	34
5.5	PlannerService . . . . .	35
5.6	Testovanie systému plánovania . . . . .	35
5.7	Vyhodnotenie aplikácie . . . . .	36
<b>6</b>	<b>Záver</b>	<b>38</b>
<b>A</b>	<b>Inštalácia</b>	<b>41</b>
<b>B</b>	<b>Užívateľské rozhranie</b>	<b>43</b>
<b>C</b>	<b>Dotazník</b>	<b>46</b>
<b>D</b>	<b>CD so zdrojovými kódmi</b>	<b>49</b>

# Kapitola 1

## Implementácia

Kapitola pojednáva o oboch častiach systému pre monitorovanie stavu plánovacích úloh. Najprv rozoberieme časť systému pre užívateľské rozhranie 1.1 a následne časť plánovania (PlannerService) 5.5, ktorá zabezpečuje riešenie plánovacích úloh. Pozrieme sa na problematiku spojenú s prihlasovaním a prípadnú validáciu údajov a navigáciu po úspešnom prihlásení. Rovnako bude rozobratý princíp komunikácie časti užívateľského rozhrania s plánovacou časťou systému prostredníctvom webovej služby na úrovni implementácie. Na záver uvedieme postup testovania systému monitorovania spolu s vyhodnotením a jej možným rozšírením.

### 1.1 Aplikácie pre užívateľské rozhranie

Aplikácie pre užívateľské rozhranie, ktorá je schopná zobrazovať informácie o úlohách, užívateľoch a organizáciách a umožňovať ich správu. Aplikácia je rozdelená do 2 modulov kvôli závislosti plánovacej časti systému na 1 z modulov:

- Rodičovský modul, ktorý je rozdelený do 5 balíkov.
- Modul Entities, ktorý je rozdelený do 1 balíka a obsahuje entitné triedy pre objektovo-relačné mapovanie databáze. Obsahuje triedy, ktorých názvy odpovedajú názvom databázových tabuliek.

Rodičovský modul sa delí na nasledujúce balíky:

- `org.jboss.optaplanner.controller.database` - balík obsahuje triedu, ktorá zabezpečuje operácie pre vytvorenie, mazanie, editáciu alebo vyhľadanie dát v databáze
- `org.jboss.optaplanner.controller.beans` - balík obsahuje managed beany, ktoré zabezpečujú uchovávanie dát a vykonanie akcií spojených s prihlasovaním a hlavnú managed bean-u (AdministratorBean), ktorá realizuje všetky operácie od zobrazenia užívateľov, organizácií, plánovacích úloh až po ich editáciu, mazanie a vytváranie
- `org.jboss.optaplanner.controller.service` - tento balík obsahuje triedy, ktoré zabezpečujú komunikáciu s plánovacou časťou systému
- `org.jboss.optaplanner.controller.restservice` - balík obsahuje triedu pre vytvorenie REST-ful webovej služby, ktorá slúži k zverejňovaniu informácií o plánovacích úlohách

- org.jboss.optaplanner.controller.model - balík obsahuje triedy, ktoré mapujú príslušné entitné triedy, ktoré dopĺňajú o ďalšie informácie

Komunikácia s databázou je realizovaná prostredníctvom aplikačného servera. Základom je správne nastavený súbor persistence.xml, v ktorom sú uvedené informácie o entitných triedach a odkaz na datasource aplikačného servera (špeciálna definícia v rámci aplikačného servera, ktorá obsahuje informácie o databáze a prihlasovacie údaje).

### 1.1.1 Prihlasovanie

Základom prihlasovania sú komponenty na stránke Login.xhtml pre zadanie mena a hesla užívateľa. Údaje sú následne spracované v managed bean-e (triede) s názvom LoginBean, ktorá je súčasťou balíku org.jboss.optaplanner.controller.beans. Táto trieda rovnako obsahuje aj validátory (metódy validateUsername/validatePassword), ktoré kontrolujú existenciu užívateľa a validitu hesla. V prípade, že užívateľ neexistuje alebo je zadané nevalidné heslo, je zobrazená komponenta h:outputText, ktorá zabezpečí zobrazenie príslušnej informácie.

V prípade, že validácia prebehne úspešne, zavolá sa metóda authenticate z modulu Seam Security. Táto metóda získa užívateľskú rolu zadaného užívateľa a ID užívateľa, ktorú následne vloží do životného cyklu aplikácie pomocou metódy setUser, ktorá je súčasťou balíku org.picketlink.idm.api.User.

Úspešné prihlásenie je dané nastavením metódy setStatus na hodnotu SUCCESS, v prípade, že validácia údajov neprebehne úspešne, sa nastaví metóda setStatus na hodnotu FAILURE. Po úspešnom prihlásení je možné identitu užívateľa získať nainjektovaním (uvedením anotácie @Inject) pred triedu Identity, z ktorej je možné získať prihlasovacie meno užívateľa, ktoré sa zobrazuje na stránke.

Po prihlásení užívateľa je realizovaná navigácia užívateľa na stránku Tasks.xhtml. Navigácia je implementovaná pomocou pravidla v súbore faces-config.xml. Rozlíšenie užívateľskej role pri prihlasovaní je dané zobrazovaním položiek v menu aplikácie prostredníctvom, ktorého je možné vykonávať akcie.

Problematika odhlasovania úzko súvisí s prihlasovaním. Menu aplikácie obsahuje komponentu h:commandButton, ktorá v atribúte action volá metódu logout. Tá spôsobí zavolanie metódy identity.logout, ktorá odobere identitu daného užívateľa (zamedzí mu opätovný prístup k stránke podľa jeho role) a presmeruje ho na prihlasovaciu stránku (Login.xhtml).

### 1.1.2 Komunikácia s PlannerService

Základom komunikácie s plánovacou 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 JBoss s prepínačom -k a cestou k WSDL súboru
- Skopírovanie vygenerovaných tried do aplikácie pre užívateľské rozhranie do balíku org.jboss.optaplanner.controller.service

Na stránke Tasks.xhtml je zobrazená tabuľka úloh, ktorá pre daný stav úlohy povoľuje rôzne akcie. Jedným z tých akcií je spustenie/pozastavenie behu plánovania úlohy. Pri stlačení tlačidla na spustenie úlohy/pozastavenie je zavolaná metóda runTask/pauseTask



z triedy `AdministratorBean`. Táto metóda vytvorí inštanciu webovej služby a zavolá jej metódu `runTask/pauseTask`, ktoré sú súčasťou koncového bodu webovej služby `Planner-Service`. Tieto metódy sú volané s argumentom `ID`, ktorý odpovedá `ID` úlohy, ktorá má byť pozastavená/spustená.

### 1.1.3 Implementácia rozhrania

Pre implementáciu rozhrania bola použitá technológia XHTML stránok. Pre každú užívateľskú rolu sú sprístupnené rovnaké stránky až na to, že do každej stránky je vložené menu a to stránkou `template.xhtml`, pričom položky menu sú zobrazované podľa užívateľskej role. Stránka `template.xhtml` obsahuje menu, ktoré je rozdelené do kategórií. V pravej hornej časti sa nachádza informácia o prihlásenom užívateľovi vrátane tlačidla na odhlásenie.

Pri kliknutí na danú kategóriu sa vyroluje zoznam, ktorý obsahuje rôzne položky v závislosti od kategórie. Položky `View Tasks`, `View Users` a `View Organization` obsahujú komponenty `h:dataTable` z knižnice JSF pre zobrazenie dát úloh, užívateľov a organizácií. Tieto dáta sú pravidelné obnovované z databáze, čo zabezpečuje ich aktuálnosť prostredníctvom komponenty `aj:poll`, ktorá je vytvorená pre každú tabuľku a pravidelne volá metódu, ktorá získava údaje z databáze. Rovnako každá stránka obsahuje pole pre vyhľadávanie, pričom je možné zvoliť podľa, ktorého stĺpca sa bude vyhľadávať. Výsledky sa zobrazia do rovnakej tabuľky (reprezentované komponent `h:dataTable`) pričom zobrazené položky budú odpovedať nájdeným výsledkom. Pri vyhľadávaní sa preruší obnovovanie obsahu tabuliek a zobrazí sa informácia o vyhľadávanom reťazci a časovom razítke kedy bolo vyhľadávanie realizované. S vyhľadanými položkami je rovnako možné realizovať všetky akcie ako s obnovovanými dátami, ktorých obsah je pravidelne obnovovaný.

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. Všetky akcie je možné vykonávať prostredníctvom komponenty `h:commandButton`, ktoré volajú príslušné metódy z `AdministratorBean`. Jednotlivé tlačidlá reflektujú individuálny stav danej entity (úlohy, užívateľa alebo organizácie). Rovnako je možné jednotlivé entity (organizácie, úlohy, užívateľov) vytvárať prostredníctvom stránok `Create Task/Organization/User`. Tie zavolajú metódu z triedy `AdministratorBean`, ktorá zabezpečí jej vytvorenie a uloženie do databáze.

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 zavolanie metódy (napr. pre stĺpec `ID` sa zavolá metóda `sortById`), ktorá je daná atribútom `action` v komponente `h:commandLink`. Metóda radenia je implementovaná prostredníctvom triedy `Collections`, ktorá obsahuje metódu `sort`, ktoré triedia model (trieda, ktorá obsahuje položky tabuľky) danej entity, ktorá vytvorí komparátor, ktorý porovná 2 položky daného modelu a upraví ich poradie.

## Kapitola 2

# Úvod

V poslednej dobe sa pre tvorbu rozsiahlych aplikácií, ktoré kladú dôraz na rýchlosť, bezpečnosť a transakčné spracovanie používa trojvrstvový model. Komplexným riešením pre vývoj rozsiahlych informačných systémov s využitím trojvrstvého modelu je platforma Java Enterprise Edition, 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 zadáný spoločnosťou Red Hat. Tento systém je schopný riešiť rozličné plánovacie problémy reprezentované definičným súborom vo formáte XML. Riešenie takto zadáných plánovacích problémov je realizované frameworkom OptaPlanner, ktorý na základe pravidiel a definičného súboru pre danú úlohu sa pokúsi nájsť najlepšie riešenie na základe plánovania problému, ktoré poskytne ako výsledok vo forme XML súboru, ktoré reprezentuje najlepšie dosiahnuté riešenie. Rovnako je proces analýzy a vývoja výsledného systému obsahom tejto správy.

Nasledujúca kapitola 3 sa venuje Java Enterprise Edition platforme, rovnako jej trojvrstvovému modelu spolu s použitými technológiami.

V tretej kapitole 4 sa pojednáva postupne od základov problematiky plánovania až po bližšie vysvetlenie pojmu plánovací problém. V tejto kapitole sa ešte oboznámime s princípom plánovania prostredníctvom frameworku OptaPlanner, rovnako aj jeho konfiguráciou.

V štvrtej kapitole 5 je prezentovaná špecifikácia požiadaviek, rovnako ako aj analýza plánovacieho systému spolu s návrhom užívateľského rozhrania a databázovej schémy. V predposlednej kapitole 1 je uvedená implementácia výsledného systému. Na záver kapitoly sú uvedené metódy a postupy testovania grafického užívateľského rozhrania. V záverečnej kapitole 6 je zhrnutý obsah celej práce, sú zhodnotené jej prínosy a možnosti ďalšieho rozšírenia.

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

## Kapitola 3

# Java Enterprise edition 6

Táto kapitola poskytuje prehľad o platforme Java Enterprise Edition 6 (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. Všetky spomenuté technológie sú použité pre tvorbu systému monitorovania, konkrétne pre tvorbu užívateľského rozhrania, pre prácu s databázou a pre komunikáciu medzi jednotlivými časťami systému.

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, ktorý je určený pre jazyk Java.

### 3.1 Špecifikácia platformy

Java EE je platforma, ktorá zastrešuje viaceré technológie a definuje prostriedky určené pre zjednodušenie vývoja podnikových aplikácií [9]. 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. Týmito 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 Javy EE je špecifikácia Java SE, ktorá je vyvíjaná prostredníctvom Java Community Process [17]. Ten predstavuje spoluprácu viacerých firiem, ktoré sa podieľajú na výsledných špecifikáciách platformy.

### 3.2 Trojvrstvový model

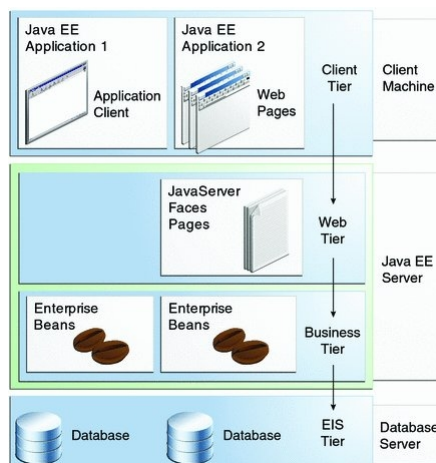
Java EE definuje rozdelenie aplikácie do vrstiev, ktoré medzi sebou komunikujú. Toto rozdelenie sprehľadňuje a uľahčuje vývojové cykly jednotlivých častí aplikácie [9]. Každá vrstva je reprezentovaná komponentami, ktoré odpovedajú zodpovednosti danej vrstvy a sú vytvorené technológiami z platformy Java EE:

- 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 komponent 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 alebo externé systémy, ktoré označujeme názvom Enterprise Information System (EIS). Úlohou tejto vrstvy je ukladanie a sprístupňovanie dát.

Na obrázku č. 3.1 môžeme vidieť trojvrstvový model Java EE aplikácie: Klient pristupuje



Obr. 3.1: Trojvrstvový model Javy EE rozdelený na klientskú vrstvu, strednú vrstvu a najnižšiu vrstvu. Prevzaté z [9].

k Java EE aplikácií prostredníctvom webového prehliadača, alebo klientskej aplikácie, preto ho delíme nasledovne:

- Tenký klient - pozostáva z webového 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 a môže sa priamo dotazovať podnikových komponentov 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 podnikovými komponentami pre zistenie dodatočných informácií (typicky informácií z databáze). Podniková vrstva je reprezentovaná podnikovými komponentmi (beanami), ktoré tvoria logiku aplikácie. Tieto komponenty môžu prijímať požiadavky priamo od 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.

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

### 3.2.1 JavaServer Faces

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

Komponenty si udržiujú 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`, ktorá je súčasťou frameworku. Tá zabezpečuje spracovanie jednotlivých požiadavok, ktoré prichádzajú z URL adresy stránky aplikácie. Spracovanie požiadavky je dané životným cyklom spracovania požiadavky [6].

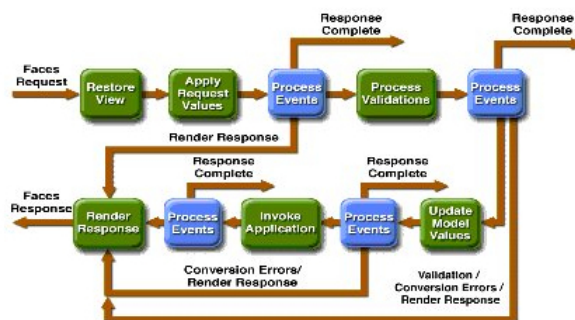
Model tvoria triedy, ktorých premenné a metódy sú zviazané s komponentami v prezentačnej časti. Tieto triedy majú meno prostredníctvom, ktorého sú adresované a rozsah ich životnosti v aplikácii.

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. 3.2 môžeme vidieť životný cyklus JSF aplikácie. Celý životný cyklus spracovania požiadavky sa zo 6 fáz: *Restore View*, *Apply Request values*, *Process validation*, *Update model values*, *Invoke application*, *Render response*.

Cyklus sa začína fázou *Restore View*, keď je kliknuté na tlačidlo alebo na odkaz sa vytvorí náhľad stránky, spoja sa všetky spracovania udalostí, validátory a komponenty sa uložia do inštancie `FacesContext-u`. V ďalšej fáze *Apply Request Values* je vytvorený strom komponentov zo stránky a sú získané nové hodnoty použitím metódy `decode`. Hodnoty sú potom uložené lokálne do komponentov. Pokiaľ nastane chyba, tak je propagovaná a generovaná do `FacesContext-u`. Na konci tejto fázy sa vykoná znova dekodovanie. Vo fáze *Process Validation* sa spracujú všetky registrované validátory ku komponentom. Pokiaľ nastala chyba, tak je táto informácia uložená do `FacesContext-u`. Počas ďalšej fázy *Update*



Obr. 3.2: Životný cyklus JSF aplikácie pri spracovaní požiadavky. Prevzaté z [9].

Model Values sa nastaví do komponent lokálne nové hodnoty. Počas predposlednej fáze Invoke Application sú spracované rozličné žiadosti ako potvrdzovanie formulára alebo odkaz na inú stránku. V poslednej fáze Render Response dôjde k žiadosti o vytvorenie stránky s novými hodnotami v kontajneri [6].

## Facelets

Facelets označuje deklaratívny jazyk pre tvorbu prezentačnej časti. Táto technológia nahradila staršiu technológiu JavaServer Pages [19]. Výhodou je oddelenie prezentačnej časti od aplikačnej logiky, pričom pre tvorbu stránok je použitá technológia XHTML [5].

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. Pri preložení aplikácie dochádza k vygenerovaniu príslušnej komponenty na stránke, pričom typicky ide o HTML grafické komponenty alebo o komponenty definované v rámci knižnice daného menného priestoru.

## Managed Bean

Managed Bean 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. Managed Bean-a 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) [6]. Tento jazyk zabezpečuje obojsmerné viazanie hodnôt medzi komponentami a Managed Bean. JSF zabezpečuje pravidelné aktualizácie hodnôt v Managed Bean-e v rámci fáze aktualizácie hodnôt.

### 3.2.2 Navigácia

Framework JSF definuje navigáciu medzi stránkami vytvorenými rôznymi značkovacími jazykmi po vykonaní nejakej akcie (napr. kliknutí na odkaz, tlačidlo) na základe sady navigačných pravidiel uložených v súbore faces-config.xml. Každé navigačné pravidlo definuje za akých podmienok má prebiehať navigácia z jednej stránky na stránku ďalšiu. Navigačné

pravidlá sú aplikované na základe aktuálne zobrazenej stránky. Po vybratí navigačné pravidla je prístup na ďalšiu stránku závislý od invokačnej metódy komponenty a logického výsledku referencovaného komponentom.

Listing 3.1: Ukážka konfigurácie navigácie v súbore faces-config.xml

```
1 <navigation-rule>
2   <from-view-id>/login.xhtml</from-view-id>
3   <navigation-case>
4     <from-action>#{LoginForm.login}</from-action>
5     <from-outcome>success</from-outcome>
6     <to-view-id>/storefront.xhtml</to-view-id>
7   </navigation-case>
8 </navigation-rule>
```

Na obrázku č. 3.1 je ukážka navigačného pravidla uloženého v súbore faces-config.xml pre JSF aplikáciu:

- Na riadku č. 1 je uvedená párová značka `<navigation-rule>`, ktorá zaobaluje 1 navigačné pravidlo. Takýchto navigačných pravidiel môže byť v súbore faces-config.xml viacero.
- Na riadku č. 2 je uvedená párová značka `<from-view-id>`, ktorá obsahuje reťazec `/login.xhtml`. Ten definuje, že navigačné pravidlo je aplikované pokiaľ je zobrazená stránka `login.xhtml`
- Na riadku č.3 je uvedená párová značka `<navigation-case>`. V jej obsahu sú uvedené podmienky za akých sa vykoná navigácia, rovnako aj stránka, na ktorú vykoná navigácia
- Na riadku č. 4 je uvedená párová značka `<from-action>`, ktorá obsahuje reťazec `LoginForm.login`. Tá odkazuje na metódu `login` v beane `LoginForm`. Tá definuje, že navigácia bude realizovaná pri zachytení(stlačení odkazu/tlačidla) udalosti `login`
- Na riadku č.5 je uvedená párová značka `<from-outcome>`, ktorá obsahuje hodnotu `success`. Táto značka indikuje vykonanie navigácie pri úspešnom (bezchybnom) invokovaní metódy na riadku č. 4
- Na riadku č. 6 je uvedená značka párová `<to-view-id>`, ktorá obsahuje reťazec `/storefront.xhtml`. Obsah tejto značky definuje, na ktorú stránku bude realizovaná navigácia v prípade splnenia všetkých podmienok uvedených na riadku č.2, č.3, č.4, č.5

### 3.3 Webová služba

Webová služba je softwarový systém navrhnutý na podporu interoperability medzi rôznymi zariadeniami prostredníctvom počítačovej siete [9]. Komunikácia medzi zariadeniami prebieha prostredníctvom HTTP protokolu vymieňaním Extensible Markup language(XML) správ. XML je značkový jazyk, ktorý definuje sadu pravidiel pre kódovanie dokumentu vo formáte zrozumiteľnom človeku prostredníctvom ľubovoľných značiek.

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 metód a pri nasadení na Java EE server môžu byť vzdialene (po sieti) zavolané [11].

### 3.3.1 Big webová služba

Big webová služba je druh webovej služby, ktorý pre svoju implementáciu používa API JAX-WS [9]. 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 a 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. 3.3 ukazuje spôsob komunikácie medzi klientom,



Obr. 3.3: Ukážka komunikácie Big Webovej služby medzi klientom a službou. Prevzaté z [9].

ktorý sa nachádza v ľavej časti obrázku a webovou službou, ktorá sa nachádza v pravej časti obrázku. Komunikácia prebieha prostredníctvom vymieňania SOAP správ.

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 volania, rovnako aj poskytovanú odpoveď. Správy volania a odpovedí webovej služby sú vymieňané prostredníctvom SOAP správ prostredníctvom HTTP protokolu.

JAX-WS API je pomerne komplikované, preto celá komplexnosť je vývojárovi zakrytá a 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. Čo sa týka vývoja webovej služby, tak sa jedná o jednoduchú javovskú triedu, ktorá používa anotáciu `javax.jws.WebService`, konkrétne anotáciu `@WebService`, ktorá označuje, že sa jedná o koncový bod webovej služby. Táto trieda následne definuje metódy, ktoré môžu byť vzdialene volané. Aby mohla byť metóda metódou webovej služby volaná vzdialene musí byť anotovaná prostredníctvom anotácie `javax.jws.WebMethod @WebMethod` [11].

### 3.3.2 RESTful webová služba

RESTful webová služba je druh webovej služby, ktorý pre svoju implementáciu používa API JAX-RS [9]. Tento druh webovej služby nevyžaduje striktné používanie XML formátu a doručovanie správ vo formáte SOAP ako Big webová služba 3.3.1. Tento typ webovej služby je sprístupňovaný 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 pristupovať k RESTful webovej službe. 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 (uchovávané sa vo vyrovnávacej pamäti) kvôli zvýšeniu výkonnosti. Nevýhodou je, že pri vytváraní klienta a služby musí byť použité rovnaké rozhranie z dôvodu explicitnej nepodporovateľnosti jednoznačného formátu správ pre komunikáciu [11]. Výhodou použitia tohto typu webovej služby je jednoduchosť implementácie producenta a spotrebiteľa.

### 3.4 Java Persistence API

Java Persistence API (JPA) je framework jazyku Java, ktorá poskytuje prístup a spravovanie relačných dát v databáze pomocou prístupu objektovo-relačného mapovania [16]. Princíp objektovo-relačného mapovania predstavuje namapovanie javovskej triedy, ktorú nazývame entita na databázovú tabuľku. Entita je perzistentný doménový objekt (javovská trieda), ktorej inštancia reprezentuje riadok v databázovej tabuľke. Základný artefaktom v programovaní je pre entity povinnosť obsahovať vlastnosti, ktoré priamo odpovedajú schéme vytvorenej databáze. 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 špecifickú vlastnosť (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`

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 anotáciu `javax.persistence.JoinColumn`, v ktorej parametroch uvedieme názvy vlastností druhej entity, ktoré sú súčasťou vzťahu [16].

Pre prácu s jednotlivými entitami sa používa `javax.persistence.EntityManager`. `EntityManager` je trieda, ktorá dokáže vytvárať, 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 refazcovo 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,

pretože 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 vzťahov, ktoré sú typovo bezpečné. Výhodou tohto API, pre použitie na dotazovanie je 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. automatické vytvorenie schémy databázy z entitných tried [16].

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.

## 3.5 Enterprise JavaBeans

Enterprise JavaBeans (EJB) je technológia, ktorá umožňuje vytvárať komponenty, ktoré sa nachádzajú v strednej podnikovej vrstve trojvrstvého aplikačného modelu 3.2 [13]. Tieto komponenty sú modulárne, keďže je možné 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 - Komponent pôsobí v roli poslucháča určitého typu správ, na ktorých príjem reaguje vykonaním určitých akcií [9]
- Session bean - Komponent vykoná úlohy pre klienta. Voliteľne môže implementovať webové služby [9]

### 3.5.1 Message-driven Bean

Message-driven bean (MB) je typ EB, ktorá umožňuje aplikáciám asynchrónne spracovanie správ. Táto beana prijíma správy z JMS fronty, ktoré následne analyzuje a vykonáva s nimi príslušné akcie [15]. JMS je technológia, ktorá umožňuje komunikovať komponentom 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 beane v tom, že sa k takému typu beane neprístupuje 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 [13]. Výhodou MB je ekvivalentnosť MB, to znamená že správy môžu byť priradené na

spracovanie jej ľubovoľnej inštancii. Výhodou je asynchrónne vyvolanie, ktoré nevyťažuje prostriedky servera.

### 3.5.2 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 [13].

SB delíme na 3 typy:

- Stateful Session Bean - Udržiava hodnoty premenných, pričom každá beana reprezentuje unikátny stav klienta/sedenia. Pokiaľ sa sedenie odstráni 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.

## 3.6 Možnosti spolupráce Javy EE s databázou

Java EE definuje nový spôsob prístupu k databáze a to objektovo relačné mapovanie, ktoré bolo vysvetlené v kapitole 3.4. Java EE podporuje pre ukladanie a správu dát množstvo databázových technológií. Tieto technológie môžu byť relačné (MySQL, PostgreSQL) alebo nerelačné (MongoDB). MySQL predstavuje jednu z podporovaných možností spolupráce Javy EE s databázovou technológiou. Táto technológia je open source a je vhodná pre malé a stredne veľké aplikácie, pričom je vyvíjaná spoločnosťou Sun Microsystems. MySQL patrí medzi klient-server technológie, kde užívatelia predstavujú klientov, ktorý prístupujú k dátam a server tieto dáta sprístupňuje a rovnako vykonáva nad nimi databázové operácie [4]. Klient môže typicky bežať na rovnakom počítači ako server, alebo môže vzdialene (po sieti) pristupovať k databázovému serveru.

Podporuje tvorbu databázových procedúr, databázových triggerov, rovnako ukladanie internacionálnych znakov. Ďalšími výhodami tejto technológie je jednoduchosť inštalácie a multiplatformosť, preto je možné ju nasadiť na systémy s operačným systémom Windows, Linux alebo Mac Os. Medzi nevýhody tejto technológie patrí neefektívna práca s databázovými transakciami a neefektívne ukladanie veľkého množstva dát [4].

## 3.7 Seam

Seam je aplikačný framework pre Javu EE, ktorý definuje uniformný komponentný model pre podnikovú logiku aplikácie [20]. Seam rieši integráciu EJB 3.5 a JSF 3.2.1 spolu. Medzi ďalšie výhodné vlastnosti tohto frameworku patrí integrácia Asynchronous JavaScript and XML (Ajax), rovnako aj vstavaná podpora javascriptu a efektívne spracovanie webových dotazov [7].

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

Základom bezpečnosti modulu Seam Security je autentifikácia, čo je proces 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, ...) [20].

Ďalšou vlastnosťou je Identity Management, ktoré predstavuje množinu API pre správu užívateľov, skupín a užívateľských rol, ktorá je súčasťou Seam Security API. Identity Management je poskytovaný Seam komponentom 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 aplikácie. Týmto spôsobom (prítomnosťou triedy Identity Bean) sa overuje užívateľ. V rámci autentifikácie sú definované v 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 [20].

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.

### 3.8 Ostatné technológie pre tvorbu užívateľského rozhrania

Výsledné užívateľské rozhranie bolo rozšírené o ďalšie prostriedky pre podporu AJAXU a interaktívnych grafických komponent s podporou prenositeľnosti na mobilné zariadenia [7]. Týmto technológiami je CSS framework Twitter Bootstrap a framework s podporou AJAX-u RichFaces.

Twitter Bootstrap je framework, ktorý obsahuje súbor nástrojov pre vytváranie webových stránok a webových aplikácií [12]. Ponúka podporu webových technológií HTML, CSS, JavaScript a mnohých grafických 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.

RichFaces je open-source framework s podporou Asynchronous Javascript and XML(AJAX) [7], ktorý predstavuje rozšírenie JSF frameworku 3.2.1. RichFaces 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.

### 3.9 Možnosti testovania v Java EE

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 [14]. JUnit je súčasťou balíku org.junit [10]. 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 a to docielime anotovaním metód @Before anotáciou alebo po spustení testu a to docielime anotáciou @After pred názvom metódy. 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, .... 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 [1]. 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 nástroju Arquillian sa deje použitím anotácie @RunWith. Táto anotácia zabezpečí spustenie testov v Java EE kontajneri. Následne tento nástroj spustí kontajner a nasadí obsah testovacieho archívu, pričom musí byť anotovaný anotáciou @Deployment. Archív obsahuje testy so špecifickými triedami a knižnicami, ktoré potrebuje. Testy sa následne vykonajú vo vnútri kontajneru, preto je možné otestovať podnikové a webové komponenty za behu.

### 3.10 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 aplikácií. 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 [8]. 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 [8].

Používanie aplikačného servera JBoss je možné vykonať ručne prostredníctvom konzoly a nájdením inštalačného adresára, ktorý obsahuje skript run.sh, ktorý spustí JBoss. Po spustení serveru je možné k nemu implicitne pristupovať na localhost-e na porte 8080.

## Kapitola 4

# OptaPlanner

OptaPlanner je open source framework, ktorý rieši a optimalizuje rôzne plánovacie problémy, ktoré sú reprezentované XML definičným súborom pre daný problém. 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 je reprezentované 1 výstupným súborom vo formáte XML.

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 proces riešenia je riadený triedami v tomto jazyku. Tieto triedy sú špecifické pre daný problém musia byť dodané spolu so zadáním problému, pričom a musia byť schopné získať potrebné informácie z definičného 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 ohodnotia krok a prekalkulujú celkové skóre a na záver vrátiť najoptimálnejšie riešenie [3].

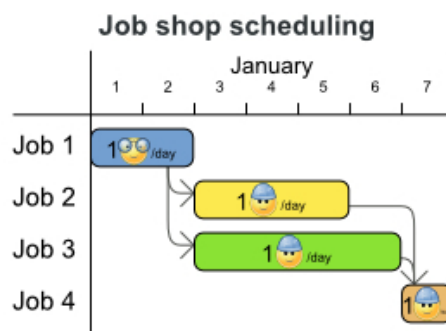
Postup riešenia problému a kalkulácie skóre sa opakuje pre rôzne scenáre (napr. v prípade N Dám pre rôzne alternatívne kombinácie pohybov, ktoré sú rozpracované súčasne). OptaPlanner sa snaží vždy nájsť optimálne riešenie vzhľadom k optimalizačným algoritmom a metaheuristickým metódam a dostupnému času, ale niekedy nie je schopný poskytnúť na predchádzajúce podmienky optimálne riešenie (riešenie je ukončené predčasne z dôvodu vyčerpania dostupného času). Za podmienok vyčerpania dostupného času je vrátené doposiaľ najlepšie dosiahnuté riešenie. Výhodou tohto frameworku je možnosť aplikovania na rozličné plánovacie problémy, ktorých presné (analytické) riešenie neexistuje alebo je veľmi ťažké ich nájsť v dostupnom čase. Príkladom môže byť skupina problémov označovaných ako NP-úplné problémy, ktoré sa vyznačujú tým, že ich riešenie nie je možné nájsť v dostupnom čase.

### 4.1 Plánovací problém

Plánovacím problémom môžeme obecné označiť 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 nespomaľovali chod ostatných zamestnancov, ktorí sú na ich práci závislí a nemuseli zbytočne čakať. Plánovať môžeme spúšťanie testovania aplikácií v rámci vývojárskej firmy, aby niektoré úlohy boli otestované skôr ako iné, no musí byť čo najefektívnejšie vyvážením použitých zdrojov (procesorového času) a zbytočne prostriedkami nemrhali. Pokiaľ je problém dostatočne komplexný, potom je veľmi vhodné použiť Optaplanner.



Obr. 4.1: Ukážka problému rozvrhnutia práce. Prevzaté z [3].

Obrázok č. 4.1 zobrazuje typické použitie OptaPlanner-u. Môžeme vidieť, že na 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 pracovníkov a teda nemôžu začať pracovať pokiaľ nie je dokončená práca predchádzajúceho pracovníka. V prípade náročnosti zadania takého to problému je pomerne jednoduché naplánovať správne poradie činností. Problém nastáva, ak by v danom obrázku bolo niekoľko násobne viac osôb. V tomto prípade by štandardným prístupom 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 zvolíť v následnosti tak, aby výsledná práca bola spravená za najkratší možný čas vzhľadom na činnosti, ktoré sa optimalizujú.

## 4.2 Princíp

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

Riešenie problému sa začína tvorbou 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 nastaví sa konfiguračný súbor pre daný problém, ktorý bude bližšie popísaný v nasledujúcej kapitole 4.3. Aby bolo jasné aké akcie (kroky plánovania) sú povolené pre daný problém sú definované v triedach pre riešenie obmedzenia. Tie ovplyvňujú aj

spôsob ohodnotenia skóre pre každý krok: [3]

- 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 obmedzenie, 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 kalkulácie skóre 1 metódou
- Inkrementálna kalkulácie skóre prostredníctvom viacerých metód
- Drools kalkulácia skóre - táto konfigurácia definuje vlastné pravidlá pre kalkulovanie skóre

Drools kalkulácia skóre využíva vlastnú DRL syntax a je daná súborom, ktorý obsahuje pravidlá [2]. 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, definičné triedy, použitie plánovacích algoritmov a metaheuristických metód) a prostriedkov na kalkuláciu skóre
2. Načítanie definičného XML súboru
3. Vykonanie kroku podľa nastavenia plánovacích algoritmov
4. Optimalizácia kroku v prípade použitia metaheuristických metód
5. Ohodnotenie kroku(v závislosti od použitia prostriedkov na kalkuláciu skóre 4.2)
6. Vykonanie alternatívneho kroku (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 4.2)
9. Opakovanie krokov 3., 4., 5., 6. až dokým nie je dosiahnuté riešenie alebo plánovanie nie je predčasne ukončené (napr. kvôli vyčerpania dostupného času)



10. Nájdenie riešenia alebo predčasné ukončenie plánovania vzhľadom na vysoké poskytnuté skóre (je možné použiť v prípade, že riešenie problému nebolo nájdené v dostupnom čase) a vrátenie najlepšieho riešenia vo formáte XML súboru

OptaPlanner definuje akým spôsobom je povolené vykonanie kroku pri plánovaní, pričom definuje aj hodnotu skóre (kladnú alebo zápornú), ktorú udelí pri vykonaní konkrétneho typu kroku. Spôsob kalkulácie skóre môžu jednoduchého charakteru, rovnako môže ísť o komplexnejšie pravidlá, ktoré sú uvedené v samostatnom súbore. Tieto spôsoby kalkulácie skóre je možné ľubovoľne vzájomne kombinovať a pritom zlepšovať presnosť dosiahnutého riešenia.

### 4.3 Konfigurácia OptaPlanneru

OptaPlanner je ovládaný konfiguračným súborom vo formáte XML, v ktorom užívateľ nastavuje aký plánovací problém bude riešený, rovnako aj spôsob akým sa bude riešiť a v poslednom rade nastaví podporované optimalizačné algoritmy a metaheuristické metódy, ktoré ovplyvňujú rýchlosť a spôsob dosiahnutia výsledku. Jednotlivé algoritmy a metódy je možné ľubovoľne kombinovať, rovnako je možné spustiť výpočet pre viacero scenárov optimalizačných algoritmov a metaheuristických metód [3]. Konfigurácia OptaPlanneru má 3 povinné časti a 4. voliteľnú:

- Nastavenie definičných tried plánovacieho problému a nastavenie tried zabezpečujúce plánovanie (Domain model configuration)
- Nastavenie definície skóre (Score Configuration)
- Nastavenie použitia plánovacích algoritmov (Optimization algorithms configuration), ktoré voliteľne obsahuje nastavenie metaheuristických metód

Pre lepšiu prehľadnosť je uvedená ukážka konfiguračného súboru.

Listing 4.1: Ukážka konfigurácie problému Cloud Balancing

```
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 vyváženia cloudu na obr. 4.1 pozostáva z viacerých častí:

- Na riadku č.3 uvedená medzi značkami enviromentMode hodnota FAST\_ASSERT, ktorá umožňuje OptaPlanneru 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 vyváženia 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 4.2
- 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 4.2
- 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 [18]
- 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 [18]
- Na riadku č. 28 je uvedená medzi značkami `acceptedCoundLimit` hodnota 1000, ktorá označuje počet náhodných krokov, ktoré sú vyhodnotené počas 1 kroku riešenia problému

V tejto sekcii bol ukázaný konkrétny príklad nastavenia OptaPlanner-u, pričom boli ukázané nastavenia definície problému, kalkuláciu skóre, spôsoby ovplyvňovania procesu plánovania (napr. predčasné ukončenie) a nastavenie optimalizačných algoritmov a metaheuristických metód.

## 4.4 Výsledky plánovacieho problému

Výsledkom plánovania prostredníctvom OptaPlanner je jediný výstupný súbor vo formáte XML. Formát výsledného súboru je problémovo špecifický a odpovedá formátu súboru vstupného, napr. v prípade, že definičným súborom problému bol problém N Dám, ktorý obsahuje počiatočné rozostavenie dām na šachovnici, tak výsledok obsahuje rozostavenie dām na šachovnici, aby sa vzájomne neohrozovali. Ďalej typicky každý definičný súbor obsahuje zoznam prostriedkov (napr. v prípade N Dām šachovnicu) spolu s rozličnými vlastnosťami a najlepším dosiahnutým skóre.

Pre výstupný súbor plánovacieho problému neexistuje XSD schéma, ktorá by definovala jednotlivé elementy a ich prípustné hodnoty a vlastnosti [3]. Výstupný súbor predstavuje na úrovni implementácie serializáciu objektov špecifického pre daný plánovací problém. Pokiaľ si zoberieme príklad N Dām, tak jednotlivé objekty predstavujú dámy na šachovnici, pričom objekty (dámy) majú vlastnosť ako riadok a stĺpec, kde sa nachádzajú. Postup serializácie sa realizuje pri zapisovaní výsledku (nájdenní najlepšieho riešenia) riešenia do výstupného súboru, kde sa objekty riešenia (napr. dámy na šachovnici) prevedú do XML formátu.

### 4.4.1 Ukážku výsledku plánovacieho problému

V nasledujúcej časti by som rád ukázal výstupný súbor pre problém 4 Dām. Počiatočným zadáním problému bolo rozostavenie Dām na 1.riadku šachovnice.

Listing 4.2: Ukážka výsledného riešenia problému N Dám

```

1 <NQueens id="1">
2   <id>0</id>
3   <n>4</n>
4   <columnList id="2">
5     <Column id="3">
6       <id>0</id>
7       <index>0</index>
8     </Column>
9     <Column id="4">
10      <id>1</id>
11      <index>1</index>
12    </Column>
13    <Column id="5">
14      <id>2</id>
15      <index>2</index>
16    </Column>
17    <Column id="6">
18      <id>3</id>
19      <index>3</index>
20    </Column>
21  </columnList>
22  <rowList id="7">
23    <Row id="8">
24      <id>0</id>
25      <index>0</index>
26    </Row>
27    <Row id="9">
28      <id>1</id>
29      <index>1</index>
30    </Row>
31    <Row id="10">
32      <id>2</id>
33      <index>2</index>
34    </Row>
35    <Row id="11">
36      <id>3</id>
37      <index>3</index>
38    </Row>
39  </rowList>
40  <queenList id="12">
41    <Queen id="13">
42      <id>0</id>
43      <column reference="3"/>
44      <row reference="9"/>
45    </Queen>
46    <Queen id="14">
47      <id>1</id>
48      <column reference="4"/>
49      <row reference="11"/>
50    </Queen>
51    <Queen id="15">
52      <id>2</id>
53      <column reference="5"/>
54      <row reference="8"/>
55    </Queen>
56    <Queen id="16">
57      <id>3</id>
58      <column reference="6"/>

```

```

59      <row reference="10" />
60    </Queen>
61  </queenList>
62  <score id="17">0</score>
63</NQueens>

```

Na obrázku č. 4.2 je ukázaný výstup OptaPlanneru pre problém 4 Dám. Každý element XML súboru obsahuje atribút ID, podľa ktorého je možného ho jednoznačne adresovať. Využitie tejto vlastnosti bude ukázané neskôr v texte. Celý súbor je zaobalený v párovej značke `<NQueens>`, ktorý značí že obsahom súboru je N Dám, pričom sa nachádza na riadku č.1. Dôležitou značkou je značka párová značka `<nl>` na riadku č. 3, ktorej hodnota 4 značí počet Dám, ktoré sa budú na šachovnici nachádzať. Celý súbor môžeme rozdeliť na 4 časti:

- Definíciu stĺpcov šachovnice ohraničené značkou `columnList` na riadku č. 4
- Definíciu riadkov šachovnice ohraničené značkou `rowList` na riadku č. 22
- Definíciu pozície dām na šachovnici ohraničené značkami `queenList` na riadku č. 40
- definíciu hodnoty skóre ohraničené značkou `score` na riadku č.63

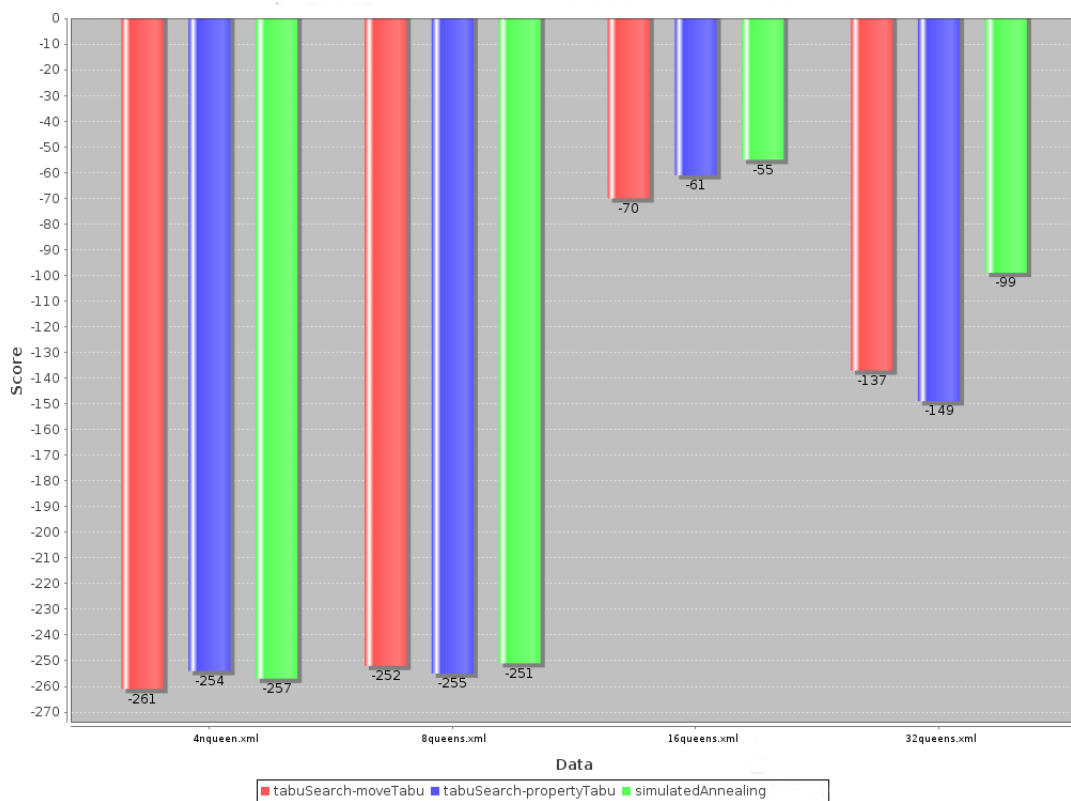
Definícia stĺpcov obsahuje zoznam stĺpcov šachovnice. Každý stĺpec je definovaný značkou `<Column>`, ktorá obsahuje značku `<index>`, ktorá definuje jeho relatívne poradie od začiatku šachovnice. Hodnoty týchto značiek nie sú jedinečné. Definícia riadkov obsahuje zoznam riadok šachovnice. Každý riadok je definovaný značkou `<Row>`, ktoré obsahuje značku `<index>`, ktorá definuje jeho relatívne poradie od začiatku šachovnice. Hodnoty týchto značiek nie sú jedinečné. Definícia pozície dām obsahuje zoznam dām a ich výsledné rozostavenie na šachovnici. Každá dáma je označená značkou `<Queen>` s atribútom `id` k jeho jednoznačnej identifikácii. Každá dáma obsahuje značku `<id>` a značky `<column>` a `<row>` a s atribútmi `reference`. Hodnoty týchto atribútov obsahujú číselnú informáciu, ktorá značí číslo stĺpca a riadka, kde sa dáma nachádza. Odkazujú na hodnoty `id` atribútov značiek pre stĺpec a a pre riadok, kde sa dáma nachádza. Na riadku č. 62 je uvedená párová značka, ktorej hodnota značí skóre najlepšieho riešenia. Táto ukážka možného výstupu plánovania prostredníctvom OptaPlanneru nie je obecná pre všetky plánovacie problémy.

#### 4.4.2 Štatistiky

OptaPlanner rovnako podporuje tvorbu štatistík a výstupu v podobe grafu z výsledku plánovania. Základom je vytvorenie špeciálnej konfigurácie, ktorá ako podmnožinu obsahuje konfiguráciu uvedenú v kapitole 4.3. V tejto konfigurácii sa nastavuje typ podporovaného grafického výstupu, napr. celkové najlepšie dosiahnuté skóre pre 1 alebo viacero konfigurácií, výstup pre najhoršiu konfiguráciu, celkový čas strávený pri riešení problému, využitie pamäte pri vykonávaní jednotlivých krokov.

Na nasledujúcom obrázku je zobrazený grafický výstup z OptaPlanneru pre riešenie problému 4 Dám, 8 Dám, 16 Dám a 32 Dám. Pre každý problém boli použité rozličné optimalizačné algoritmy s cieľom nájsť najoptimálnejšie riešenie pre problém N Dám.

Na obrázku č. 4.2 je zobrazený grafický výstup z OptaPlanneru pre hodnoty skóre pre rôzne rozostavenia dām a rozličné optimalizačné algoritmy. Graf má na ose Y nanosené číselné hodnoty skóre a na ose X definičný súbor, pre ktorý bol vyprodukovaný daný grafický výstup. Pre každý použitý definičný súbor (4 Dám, 8 Dám, 16 Dám, 32 Dám) boli použité 3 optimalizačné algoritmy: `tabuSearch-moveTabu` (reprezentovaný stĺpcom červenej



Obr. 4.2: Ukážka grafického výstupu z OptaPlanneru pre rôzne problémy 4 Dám, 8 Dám, 16 Dám a 32 Dám s použitím rôznych plánovacích algoritmov. Prevzaté z [3].

farby), tabuSearch-propertyTabu (reprezentovaný stĺpcom modrej farby) a simulatedAnnealing (reprezentovaný stĺpcom zelenej farby). Najlepšie skóre je také, ktorého hodnota je najväčšia (v tomto prípade, ktorá sa najviac blíži 0). Z daného grafického výstupu môžeme usúdiť, že pre problém N Dám s počiatočným rozložením Dám na 1.riadku šachovnice (najvyššom) je najlepší optimalizačný algoritmus SimulatedAnnealing. Takýto grafický výstup môžeme dosiahnuť pre ľubovoľný plánovací problém. Rovnako môžeme dostať aj iný grafický výstup v závislosti od vlastností, ktoré chceme sledovať (napr. celkový čas strávený pri plánovaní, využitie pamäte).

## Kapitola 5

# Analýza a návrh aplikácie

V tejto kapitole postupne uvedieme požiadavky na aplikáciu, analýzu systému, návrh aplikácie, implementáciu. Rovnako si prejdeme rozdelenie systému plánovaniu na 2 časti a to grafické užívateľské rozhranie a plánovaciu časť optimalizovanú pre problémy N Dám, vyváženia cloudu a cestovateľského turnaja s možnosťou rozšírenia pre akýkoľvek plánovací problém. V kapitole je ukázaný dátový model aplikácie spolu s návrhom užívateľského rozhrania.

### 5.1 Špecifikácia požiadavkov

V tejto kapitole postupne uvedieme požiadavky na aplikáciu, analýzu systému, návrh aplikácie, implementáciu, testovanie a nakoniec vyhodnotíme aplikáciu a navrhujeme jej možné rozšírenia. Rovnako si prejdeme rozdelenie systému plánovaniu na 2 časti a to grafické užívateľské rozhranie a plánovaciu optimalizovanú pre problémy N Dám, vyváženia cloudu a cestovateľského turnaja s možnosťou rozšírenia pre akýkoľvek plánovací problém. V kapitole je ukázaný dátový model aplikácie spolu s návrhom užívateľského rozhrania.

### 5.2 Špecifikácia požiadavkov

V tejto kapitole postupne rozoberieme požiadavky na systém monitorovania stavu úloh. Základnou úlohou systému je monitorovanie úloh. Na jednej strane bude systém schopný zobrazovať stav plánovacích úloh, na druhej strane 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 4.1 a úlohu znovu spustiť. Novú úlohu bude možné do systému vložiť a následne spustiť. Bude možné vytvárať nové definície plánovacích problémov.

Úlohy bude môcť systém publikovať, čím sa myslí akcia, ktorá vytvorí pre úlohu špeciálne URL, na ktoré sa po kliknutí zobrazí stránku s názvom úlohy a obsahom XML definičného súboru. Úlohu bude možné aj odpublikovať a po prístupí 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 ich vytvárať, editovať, mazať úlohy, publikovať a odpublikovať
- Čitateľ - úlohy môže len zobrať v rámci svojej organizácie, publikovať, odpublikovať

Užívatelia sú organizovaní 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 užívateľským menom a heslom, 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 začleňovaní 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 výsledné grafické užívateľské rozhranie bude prenositeľné na mobilné telefóny.

## 5.3 Analýza

Výslednú aplikáciu môžeme rozdeliť na 2 časti: 1. backend aplikácie, ktorý beží na Java EE serveri JBoss 3.10 a 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 s užívateľov. Použitie technológie JSF 3.2.1 je pomerne jednoznačné, keďže z Java EE technológií 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 RichFaces 3.8, ktorý priamo integruje AJAX do všetkých jeho komponent [7].

Posledným problémom, ktorý treba pri analýze grafického užívateľského rozhrania vyriešiť je prenositeľnosť na mobilné zariadenia. V tom nám pomôže framework Twitter Bootstrap 3.8. 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

Podpora ostatných prehliadačov nie je odporúčaná z dôvodu neočakávaného chovania.

V druhej časti sa zameriame na problémy backend-u aplikácie. Celá aplikácie potrebuje udržiavať a spravovať dáta. Dátami sú mienené informácie o úlohách, užívateľoch a organizáciách. Z toho dôvodu bolo treba vyriešiť otázku voľby vhodnej databázovej technológie. Existuje niekoľko možností, ktoré sa dajú ľahko integrovať s JBoss-om 3.10. 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 3.6.

Následne treba spomenúť problém komunikácie databáze s užívateľským rozhraní. Grafické užívateľské rozhranie potrebuje komunikovať s databázou odkiaľ získava aktuálne informácie o úlohách, užívateľoch a organizáciách. Rovnako sa do databáze zapisujú priebežné informácie o plánovaní. Vzhľadom na podmienku nezávislosti použitia databázovej technológie bola použitá technológia JPA 3.4.

Pre publikovanie bola zvolená jednoduchšia varianta webovej služby a to RESTful webová služba 3.3.2, ktorá bude namapovaná na URI task/parameter a parameter predstavuje ID úlohy, ktorý sa má zobrazíť. Chovanie tejto služby je možné rozdeliť podľa prihláseného a neprihláseného užívateľa:

- Neprihlásený užívateľ - V prípade pokusu o prístupu 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 3.7, ktorý možno jednoducho integrovať pod JBossom.

Komunikácia s plánovacou časťou je realizovaná prostredníctvom webovej služby 3.3 prostredníctvom HTTP protokolu. Z dôvodu použitia štandardných komunikačných protokolov a nižším nákladom na prevádzkovanie bola zvolená Big webová služba 3.3.1. Užívateľské rozhranie predstavuje klienta, ktorý volá metódy na spustenie a pozastavenie plánovania. 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 3.5.2, ktorá reprezentuje webovú službu a obsahuje funkčnosť pre spustenie a zastavenie výpočtu. Pri spustení výpočtu sú informácie predávané message-driven bean-e 3.5.1, ktorá zabezpečuje spúšťanie plánovania prostredníctvom OptaPlanner-u 4.

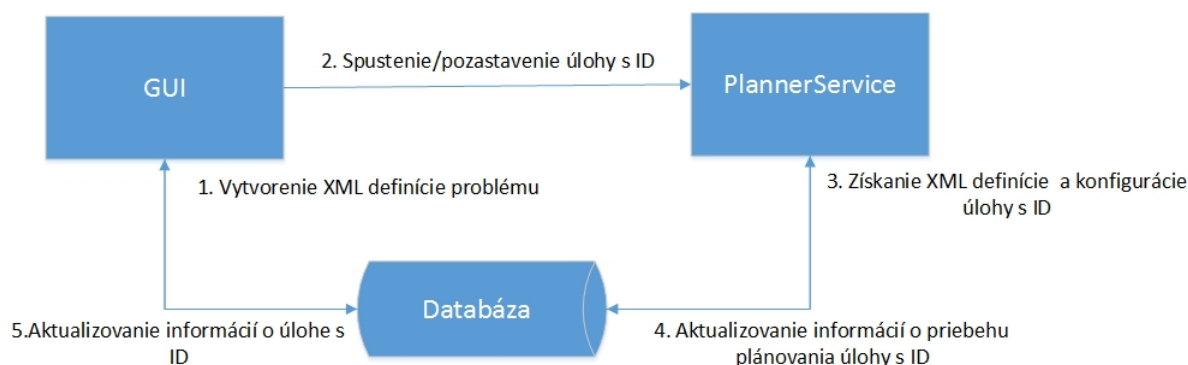
Rovnako boli použité štandardné prostriedky na otestovanie funkčnosti kódu pomocou JUnit testov a nástroju Arquillian 3.9.

Kvôli závislosti časti systému PlannerService na entitných triedach, bola aplikácia užívateľského rozhrania rozdelená do viacerých modulov.

## 5.4 Návrh aplikácie

Výsledná aplikácia je rozdelená na 2 časti. Na časť reprezentujúci grafické 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 databáze.

Pre spustenie výpočtu úlohy komunikuje pomocou webovej služby s PlannerService (optimalizované pre riešenie problému N Dám, vyváženie cloudu 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. K pozastaveniu úlohy dôjde prostredníctvom zmeny stavu vo webovej službe, čo pozastaví plánovanie.



Obr. 5.1: Diagram komunikácie časti užívateľského rozhrania s plánovacou časťou systému

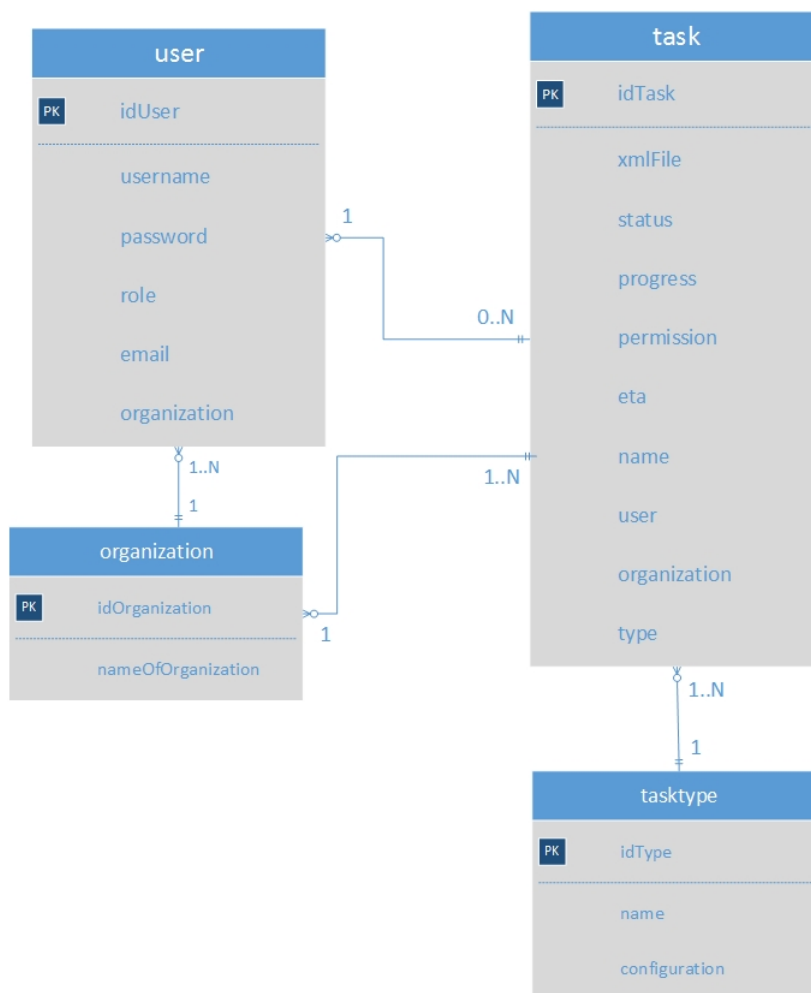
Na obrázku č.5.1 je popísané spôsob komunikácie užívateľského rozhrania s PlannerService. Základom je vytvorenie a vloženie XML definičného súboru plánovacieho problému prostredníctvom užívateľského rozhrania a následne uloženie definície do databáze. Následne sa užívateľovi zobrazí nová pridaná úloha v tabuľke úloh, ktorú mu bude umožňovať spustiť/pozastaviť plánovanie prostredníctvom tlačidla vedľa úlohy. Po stlačení dôjde k zaslaniu žiadosti o spustenie plánovania úlohy s ID úlohy prostredníctvom HTTP protokolu webovej služby PlannerService(OptaPlanner), ktorá žiadosť spracuje. Táto webová služba získa z databáze potrebný XML definičný súbor podľa ID úlohy, ktorú obdržala. Následne zmení stav úlohy a spustí plánovanie. Priebežne pritom ukladá informácie o pokroku úlohy, a čase ukončenia úlohy. Časť systému reprezentovaná užívateľským rozhraním pravidelne získava informácie o úlohách z databáze a zobrazuje ich v prehľadnej tabuľke.

Celý návrh aplikácie bol otestovaný prostredníctvom skupiny odborných a laických užívateľov s cieľom zdôrazniť rýchlu učiacu sa krivku užívateľského rozhrania. Následne prebie-

halo testovanie prostredníctvom užívateľov, ktorý testovali validáciu vstupov, prihlasovanie, správne vyhľadávanie jednotlivých entít(úloh, užívateľov, organizácií).

#### 5.4.1 Návrh modelu databáze

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



Obr. 5.2: ER diagram databáze pre systém monitorovania stavu plánovacích úloh. Zložený z tabuliek úloh, organizácií, užívateľov a typov úloh.

Tento obrázok zobrazuje jednotlivé entity, ktoré sú potrebné v databáze, každá z nich ma určité položky. ER diagram sa skladá zo 4 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 vytvorených 0 až N úloh. Taktiež pre zjednodušenie je každá úloha priradená priamo organizácií pre zlepšenie rýchlosti získania výsledku a zjednodušenia ich nájdenia. A nakoniec je každá úloha určitého typu. Každá entita obsahuje primárny kľúč(jedná sa o silné entitné množiny), ktorý je odvodený od názvu a začína predponou id\_ a pokračuje názvom entity

s 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 zaraďovaný jednotliví užívatelia a úlohy. Ďalej prejdime k entitnej množine user. Táto entita má rovnako primárny kľúč. Ďalej obsahuje položku pre užívateľské meno(username), heslo(password), email, užívateľskú rolu(role) a cudzí kľúč organization, ktorý obsahuje na 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, stav úlohy(stateOfTask, ktorý reprezentuje rôzne stavy úlohy), ktoré si podrobnejšie rozobereme. Úloha sa môže nachádzať v jednom z nasledujúcich stavov:

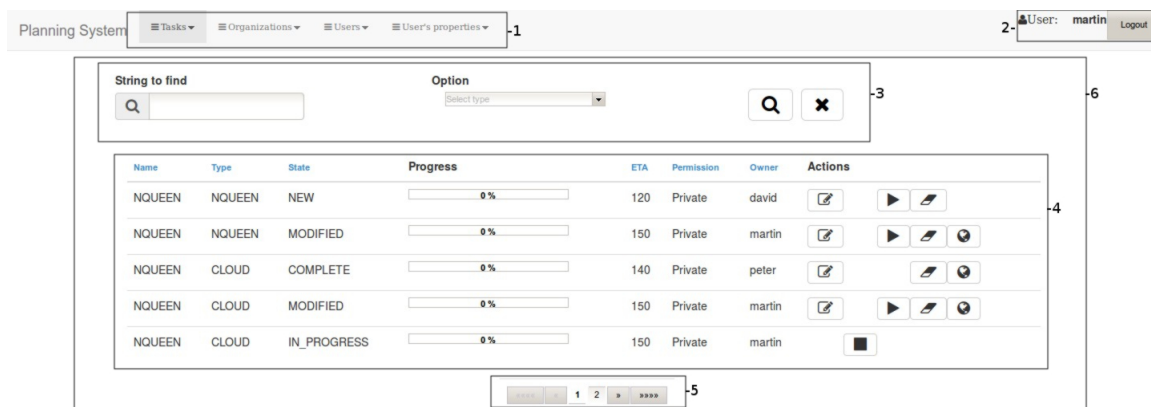
- NEW - úloha bola vytvorená
- MODIFIED - XML súbor bol modifikovaný
- WAITING - úloha čaká na spracovanie
- IN\_PROGRESS - práve prebieha výpočet
- PAUSED - úloha je pozastavená
- COMPLETE - úloha je dokončená

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(permission), 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.

#### 5.4.2 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. Jednotlivé možnosti práce so systémom sú zakomponované do rolovacích zoznamov, ktoré sa skladajú z položiek, v ktorých je sprístupnená príslušná funkčnosť. Výsledné rozhranie je prenositeľné aj na mobilné zariadenie. Užívateľské rozhranie je popísané na nasledujúcom obrázku: Na obrázku č.5.3 môžeme vidieť návrh užívateľského rozhrania. Rozhranie je rozdelené do 6 častí, ktoré môžeme rozoznať na obrázku číslami od 1 do 6, ktoré sú aj ohraničené. Celé rozhranie môžeme rozdeliť do nasledujúcich častí:

- Oblasť č.1 predstavuje navigačné menu, kde sú jednotlivé akcie rozdelené do 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



Obr. 5.3: Ukážka návrhu užívateľského rozhrania pre zobrazenie plánovacích úloh

- Oblasť č.3 predstavuje jednu z funkčných možností. Jedná sa o vyhľadávanie, ktoré je zložené zo vstupného prvku, do ktorého zadáme vyhľadávaný reťazec a druhá časť predstavuje menu, z ktorého zvolíme stĺpec podľa, ktorého bude prebiehať vyhľadávanie. Následne je možnosť realizovať tlačidlom Find, ktoré prekreslí obsah tabuľky nižšie a naplní ju nájdenými výsledkami.
- Oblasť č.4 predstavuje tabuľku, ktorá je dynamicky obnovovaná a reaguje na asynchrónne ukladanie dát z webovej služby. 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 to odkaz, preto je možné naň ho kliknúť. Po kliknutí sa zobrazí stránka s informáciami o názve úlohy a xml súbore. 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.
- Oblasť č.6 predstavuje funkčnú oblasť. Táto oblasť je špecifická pre každú stránku, 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.

Zvyšné návrhy rozhrania pre vytvorenie úlohy, editovanie úlohy, spravovanie užívateľov, spravovanie organizácií, zmenu hesla a prihlasovanie je možné dohľadať v prílohe. Pre každú akciu ako je napr. vytvorenie úlohy, vytvorenie nového typu úlohy, zobrazenie užívateľov, plánovacích úloh, ...je vytvorená samostatná XHTML stránka, ktorá je obsluhovaná z

hlavnej managed bean, ktorá realizuje všetky akcie a uchováva všetky potrebné stavy pre ďalšiu prácu.

Pre výsledné rozhranie je použitý framework Twitter Bootstrap 3.8, z ktoré sú použité dostupné komponenty pre vytváranie menu, vyskakovacie okná a ikony.

### 5.4.3 Publikovanie úloh

Ďalšou podstatnou časťou aplikácie pre užívateľské rozhranie je možnosť publikovať/odpublikovať úlohu (task). Túto akciu je možné realizovať prostredníctvom tlačidla v tabuľke úloh Publish Task/Unpublish Task na stránke Tasks.xhtml. 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 z triedy AdministratorBean. V tejto metóde dôjde k zavolaniu metódy changePermission z balíka org.jboss.optaplanner.controller.data. Zavolaním tejto metódy dôjde k zmenu stavu úlohy v databáze na verejnú. Informácia sa spropaguje do tabuľky úloh a následne sa v stĺpci permission zobrazí text Public modrou farbou, ktorý po kliknutí zobrazí názov úlohy a XML súbor plánovacej úlohy. Po kliknutí na odkaz sa zavolá RESTFul webová služba s parametrom ID úlohy. Na časť URL task/id je namapovaná RESTFul 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 služba obsahuje 1 metódu getUserById, ktorá dostane ako parameter ID úlohy. Toto ID úlohy je získavané 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 informáciu o úlohe (názov, XML súbor, povolenie a užívateľa spolu s organizáciou, do ktorej je zaradený). 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 prihlásením.

### 5.4.4 Validácia

Všetky grafické komponenty obsahujú validáciu na neprázdne, niektoré aj na nevalidné údaje (napr. validná emailová adresa). Všetky komponenty, do ktorých sa zadáva nejaká informácia sú realizované 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, čo sú metódy, ktoré sú zavolať ešte pred vykonaním akcie tlačidla. Tento typ validácie je použitý pre stránku Login.xhtml. Takáto metóda nastaví zobrazenie komponenty h:outputText prostredníctvom nastavenia atribútu rendered na hodnotu true, pričom táto komponenta obsahuje text podľa danej si-

tuácie(napr. neznámy užívateľ, nevalidné heslo). V opačnom prípade je komponenta skrytá, teda hodnota atribútu je nastavená na hodnotu false.

## 5.5 PlannerService

PlannerService predstavuje časť systému, ktorá zabezpečuje spracovanie požiadavok na plánovanie od aplikácie z užívateľského rozhrania. Aplikácia využíva ako závislosť modul Entites z časti systému užívateľského rozhrania. Aplikácia je rozdelená do nasledovných balíkov:

- org.jboss.optaplanner.service - balík obsahuje triedu pre vytvorenie koncového bodu webovej služby spolu s príslušnými metódami na spustenie/pozastavenie plánovania a triedu, v ktorej prebieha plánovanie
- org.jboss.optaplanner.solver - balík obsahuje triedu, ktorá implementuje metódy na získanie dát, spustenie/pozastavenie činnosti plánovacieho frameworku
- org.jboss.optaplanner.util - balík obsahuje triedu, ktorej úlohou je deserializácia informácií z XML definičného súboru plánovacieho problému

Komunikácia s databázou je realizovaná prostredníctvom aplikačného servera. Základom je správne nastavení súbor persistence.xml, v ktorom sú uvedené informácie o entitných triedach a odkaz na datasource aplikačného servera.

Základom je trieda OptaPlannerWebService, ktorá predstavuje Big webovú službu, ktorá je súčasťou balíku org.jboss.optaplanner.service.server. Táto trieda obsahuje metódy startTask (long id), ktorá má parameter ID úlohy, ktorú má spustiť a pauseTask (long id), ktorá má parameter ID úlohy, ktorej plánovanie má pozastaviť.

Metóda runTask vytvorí spojenie s frontou Optaplanner, do ktorej sa vkladajú ID úloh, ktoré majú byť naplánované, pomocou príkazu „sender = session.createProducer (queue)“. Týmto sa vytvorí spojenie a príkazom sender.send (message) sa vloží správa do JMS fronty.

V ďalšej časti vstupuje trieda OptaPlannerMessageBean, ktorá reprezentuje Message-bean-u pre príjem správ a ich následné spracovanie. Základom triedy je metóda onMessage (Message message), ktorá asynchrónne spracováva správy vo fronte. Základom je získanie úlohy z databáze, to je realizované príkazom Task task = em.find(Task.class, new Long(msg)). Metóda získa informácie o úlohe a príkazom execute spustí plánovanie.

Základom plánovania je cyklus, ktorý testuje korektnosť stavu na RUNNING (bežiaci) a prebiehajúce plánovanie. V cykle sa z triedy ProblemSolver získava informácia o skóre a kalkulujú a ukladajú sa nové informácie o pokroku a čase do skončenia do databáze.

Po skončení je z triedy ProblemSolver získané riešenie s najlepším skóre metódou getBestSolution (), ktoré sa uloží do databáze. Spolu s ním sa uloží informácie o ukončení plánovania (zmenu stavu, nastavenie pokroku na 100%).

Metóda pauseTask (long id) pracuje v princípe veľmi jednoducho. Jej základom je príkaz setStatus (TaskStatus.PAUSED), ktorý zmení stav úlohy na pozastavený. V plánovacom cykle dôjde k porušeniu podmienky a ukončeniu jej priebehu. Pozastavenú úlohu je možné znova spustiť opakovaným zavolaním metódy startTask (long id).

## 5.6 Testovanie systému plánovania

Testovanie prebiehalo na serveri JBoss AS 7.1.1 Final najprv prostredníctvom JUnit testov, ktoré mali overiť komplikovanú funkčnosť metód. Následne sa pre overenie funkčnosti

databáze použil framework Arquillian, ktorý umožňuje nasadenie tried priamo do kontajneru, čo zjednodušuje testovanie. Prostredníctvom tohoto frameworku sa testovala celková funkčnosť aplikácie. Postupným budovaním aplikácie sa pristupovalo k testovaniu navrhnutých častí. Junit testy boli postupne skonštruované pre jednoduchšie metódy, ako je overenie funkčnosti vyhľadávania entít, mazanie entít a pridanie entít do zoznamu úloh.

V ďalšej časti prebiehalo testovanie medzi konkrétnymi užívateľmi. Išlo o 4 informaticky 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 užívateľom s rolami Administrátor, Plánovač a Čitateľ.

Užívatelia následne testovali vytváranie užívateľov, organizácií, úloh. Následne mohli sledovať stav spracovania plánovacích úloh. Aplikáciu otestovali pod 2 prehliadačmi a to Google Chrome vo verzii 34.0 a Mozilla Firefox verzie 28.0. Bol použitý operačný systém linux 3.13.0-24-generic s operačným systémom Kubuntu 14.04. Aplikácia sa správala pod oboma rovnako a korektne. Po odhalení chýb boli chyby ohlásené a odstránené a aplikácia bola následne opäť nasadená. Tento postup sa opakoval až dokým neboli odhalené všetky chyby. 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, 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 plánovanie cloudu, N Dám, vyváženia cloudu a cestovateľského turnaja. Užívatelia nahrali tieto súbory do systému a sledovali priebeh riešenia plánovacieho problému.

Užívateľské rozhranie bolo otestované pre mobilné telefóny na zariadení HUAWEI Honour 2 s prehliadačom Google Chrome 35.0, v ktorom sa zobrazovalo korektne.

## 5.7 Vyhodnotenie aplikácie

Po testovacej fáze nasledovala fáza vyhodnotenia aplikácie. Cieľovej skupine bol po opravení chýb aplikácie predložený dotazník, do ktorého vyplňali rôzne informácie, kde dávali spätnú



väzby, chyby v návrhu, rovnako aj chyby v intuitívnosti ovládania. Cieľovou skupinou aplikácie sú užívatelia bez akejkolvek predchádzajúcej skúsenosti s touto aplikáciou s vekovým rozsahom medzi 20 - 40 rokov. Preto bola aplikácia predložená najprv užívateľom skúseným, ktorým bol poskytnutý predchádzajúci styk s aplikáciou a laickým užívateľom, ktorí nemali žiadny predchádzajúci styk. Výsledkom zistenia, rovnako vyplývajúce z výsledku 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. Počiatočným problémom bolo zorientovanie sa v tlačidlách, ktoré neobsahovali popis. Ten sa mi zobrazil až premiestnením kurzora nad dané tlačidlo. Po zistení tohto faktu, rovnako aj po zistení faktu použitia konštantných typov ikon pre vkladanie nových údajov, mazanie a vyhľadávanie bolo pre užívateľov veľmi rýchle vykonať danú akciu. Rovnako ocenili rozdelenie tlačidiel do mriežky, kde rovnaké akcie ležali pod sebou, čo im ešte urýchlilo celý proces práce s nimi.

Rovnako oceňovali možnosť zobrazovanie tlačidla Save Changes pri editovaní tabuľky vedľa položky, ktorá je práve editovaná v danom riadku spolu s tlačidlom Drop Changes na zahodenie vykonaných zmien. Pri vyhľadávaní ocenili užívatelia zachovanie zadaných informácií pre vyhľadávanie. Užívatelia by ocenili pri úlohách mať možnosť informácie o časovom razítke vytvorenia úlohy. Prehliadania pomocou tabuliek im prišlo ako veľmi vhodné rovnako aj použitie stránkovania. Užívateľom chýbala možnosť vyhľadávať podľa viacerých kritérií súčasne.

Aplikácia by mohla byť upravená do užívateľsky prívetivejšieho rozhrania a mohli byť zahrnuté všetky názory užívateľov. Rovnako by PlannerService mohla byť rozšírená o spracovanie ďalších typov plánovacích problémov, rovnako aj o vlastné plánovacie problémy, pričom rozhranie obsahuje možnosť na vytváranie vkladanie ľubovoľného typu plánovacích problémov. Rovnako použitá technológia prístupu k databáze umožňuje jednoduché rozšírenie existujúcej databáze schémy a jej následne vytvorenie. Princíp vkladanie plánovacích problémov prostredníctvom XML súboru by mohol byť nahradený sprievodcom, kde užívateľ vyplní položky a sprievodca vygeneruje príslušný definičný súbor.

## Kapitola 6

### Záver

Plánovanie a s ním spojené problémy nás stretávajú v bežnom živote čoraz častejšie. Ešte väčšie problémy tohto typu majú organizácie, ktoré musia dennodenne riešiť ako naplánovať efektívnu prácu svojich zamestnancov, ako správne komunikovať so zákazníkom a mnoho iných problémov. Riešenie klasickým prístupom a to využitím ľudskými zdrojmi je časovo neefektívne, rovnako treba brať do úvahy ľudský faktor. Preto vzniklo riešenie, ktoré odbremeňuje organizácie od riešenia komplikovaných plánovacích úloh. Taký software je šírený pod licenciou open-source a nazýva sa Optaplanner. Tento systém je následne možné využívať pre akúkoľvek oblasť plánovania, aká len nás napadne. Jediné obmedzenie tohto systému sú použité plánovacie algoritmy kombinované s rôznymi heuristikami.

Užívateľ je schopný definovať problém, 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 prívetivejším a efektívnejším. 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 iných plánovacích úloh. V poslednom rade kvôli lepšej pochopiteľnosti aplikácie by mohla byť aplikácia kvalitne zdokumentovaná.

# Literatúra

- [1] Ament, J. D.: *Arquillian Testing Guide*. Packt Publishing, 2013, iISBN 978-1782160700.
- [2] Ary, J.: *Instant Drools Starter*. Packt Publishing, 2013, iISBN 978-1782165545.
- [3] Bali, M.: *Drools JBoss Rules 5.X Developer's Guide*. Manning Publications, 2007, iISBN 978-1933988344.
- [4] DuBois, P.: *MySQL (5th Edition) (Developer's Library)*. Addison-Wesley Professional, 2013, iISBN 978-0321833877.
- [5] Felke-Morris, T.: *Web Development and Design Foundations with XHTML, 5th Edition*. Addison-Wesley, 2010, iISBN 978-0132122702.
- [6] Geary, D.: *Core JavaServer Faces (3rd Edition)*. Prentice Hall, 2010, iISBN 978-0137012893.
- [7] Holdener, A. T.: *Ajax: The Definitive Guide*. O'Reilly Media, 2008, iISBN 978-0596528386.
- [8] Jamae, J.: *JBoss in Action: Configuring the JBoss Application Server*. Manning Publications, 2008, iISBN 978-1933988023.
- [9] Jendrock, E.: The Java EE 6 Tutorial [online]. <http://docs.oracle.com/javaee/6/tutorial/doc/>, 2013-11-03 [cit. 2013-10-25].
- [10] Kaczanowski, T.: *Practical Unit Testing with JUnit and Mockito*. Tomasz Kaczanowski, 2005, iISBN 978-1584504184.
- [11] Kali, M.: *Java Web Services: Up and Running*. O'Reilly Media, 2013, iISBN 978-1449365110.
- [12] Otto, M.: Twitter Bootstrap [online]. <http://getbootstrap.com/>, 2013-12-16 [cit. 2013-12-27].
- [13] Panda, D.: *EJB 3 in Action*. Packt Publishing, 2013, iISBN 978-1782161264.
- [14] Piroumian, V.: *Test Driven Development: By Example*. Addison-Wesley Professional, 2002, iISBN 978-0321146533.
- [15] Richards, M.: *Java Message Service*. O'Reilly Media, 2009, iISBN 978-0596522049.
- [16] Schincariol, M.: *Pro JPA 2*. Apress, 2013, iISBN 978-1430249269.

- [17] Sierra, K.: *Head First Java*. O'Reilly Media, 2005, iISBN 978-0596009205.
- [18] Skiena, S.: *The Algorithm Design Manual*. Springer, 2012, iISBN 978-1849967204.
- [19] Steelman, A.: *Murach's Java Servlets and JSP*. Mike Murach & Associates, 2008, iISBN 978-1890774448.
- [20] Yuan, M. J.: *JBoss Seam: Simplicity and Power Beyond Java EE*. Prentice Hall, 2007, iISBN 978-0131347960.

## Dodatok A

# Inštalácia

V tejto kapitole by som Vám rád objasnil postup inštalácie aplikácie. V prvom rade uvediem potrebné prostriedky pre beh aplikácie:

- JBoss aplikačný server najmenej vo verzii 7.1.1.Final, ktorý je súčasťou CD
- 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 ho 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“)

V prvom rade je potrebné nainštalovať MySQL server nakonfigurovať databázu s názvom „optaplanner“ s užívateľským menom „root“ a heslom „root“. Následne je potrebné skopírovať JBoss aplikačný server na súborový systém z CD.

Následne treba vytvoriť potrebnú databázu a naplniť ju dátami, preto spustíme termi-nál:

- Zadáme príkaz `mysql -u root -e create database optaplanner -p` a zadáme heslo root
- Zadáme príkaz `mysql -u root -p optaplanner < cesta/k/suboru/create.sql`(bude od nás vyžadované heslo root, ktoré zadáme)
- Zadáme príkaz `mysql -u root -p optaplanner < cesta/k/suboru/insert.sql`(bude od nás vyžadované heslo root, ktoré zadáme)

Aplikáciu môžeme spustiť nasledovne:

- Skopírujeme adresár JBoss-7.1.1.Final na disk
- Skopírujeme pripravený adresára `optaplanner.controller.war` a `optaplanner-service.war` do adresára  
`JBOSS_HOME1/standalone/deployments`

---

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

- Prejdeme do zložky `JBOSS_HOME/standalone/bin` a spustíme skript `standalone.sh` a to zadáním do terminálu príkazu `./standalone.sh`

K aplikácií pristúpime zadáním adresy `http://localhost:8080/optaplanner.controller.war/` do podporovaného webového prehliadača a môžeme sa prihlásiť do systému nasledovne:

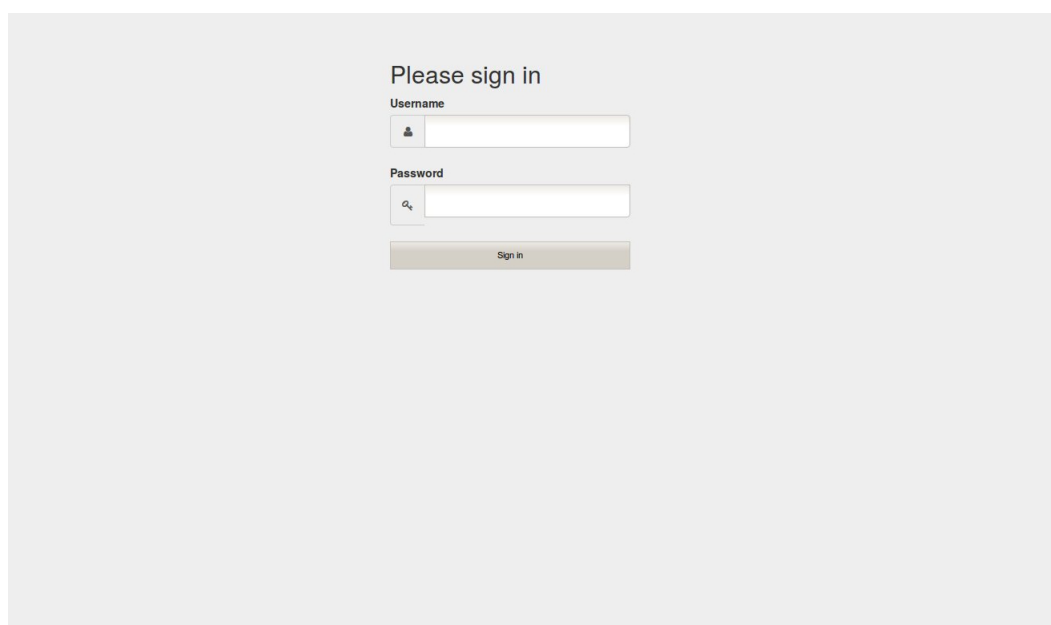
- Pre užívateľskú rolu Administrátor - prihlasovacie meno: martin, heslo: martin
- Pre užívateľskú rolu Plánovač - prihlasovacie meno: peter heslo: peter
- Pre užívateľskú rolu Čitateľ - prihlasovacie meno: david heslo: david

Aplikácia je prístupná aj prostredníctvom cloudovej služby OpenShift na adrese `http://optaplanner-xmagam00.rhcloud.com/optaplanner.controller/`. Prihlasovacie údaje pre jednotlivé užívateľské role sú zhodné s údajmi uvedenými vyššie.

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

## Dodatok B

# Užívateľské rozhranie



Please sign in

Username

Password

Sign in

Obr. B.1: Prihlasovacia obrazovka

Planning System Tasks Organizations Users User's properties User: martin Logout

String to find Option

Q [ ] Select type [ ] [ ]

Name	Type	State	Progress	ETA	Permission	Owner	Actions
NQUEEN	NQUEEN	COMPLETE	100 %	0	Private	david	[ ] [ ] [ ]
NQUEEN	NQUEEN	COMPLETE	0 %	140	Private	peter	[ ] [ ] [ ]
NQUEEN	NQUEEN	COMPLETE	100 %	0	Private	martin	[ ] [ ] [ ]
CLOUD	CLOUDBALANCING	COMPLETE	100 %	0	Private	martin	[ ] [ ] [ ]
CLOUD	CLOUDBALANCING	COMPLETE	100 %	0	Private	martin	[ ] [ ] [ ]

1 2 3 4 5 6 7 8 9 10

Obr. B.2: Zobrazenie úloh

Planning System Edit Task Organizations Users User's properties User: martin Logout

Edit Task

Name Q NQUEEN

Type NQUEEN

Owner david

XML

```
<NQueens id="1">
  <id>0</id>
  <ip>4</ip>
  <columnList id="2">
    <column id="3">
      <ip>0</ip>
      <index>0</index>
    </column>
    <column id="4">
      <ip>1</ip>
      <index>1</index>
    </column>
    <column id="5">
      <ip>2</ip>
      <index>2</index>
    </column>
    <column id="6">
      <ip>3</ip>
      <index>3</index>
    </column>
  </columnList>
</NQueens>
```

Save Changes

Obr. B.3: Editovanie úlohy

Planning System Tasks Organizations Users User's properties User: martin Logout

Create Task

Name NQUEEN

Type NQUEEN

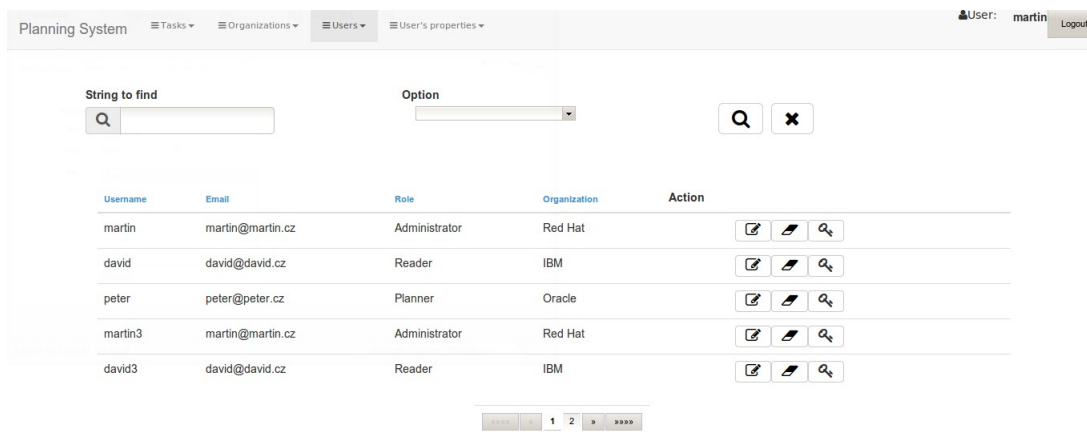
File

+ Add ...

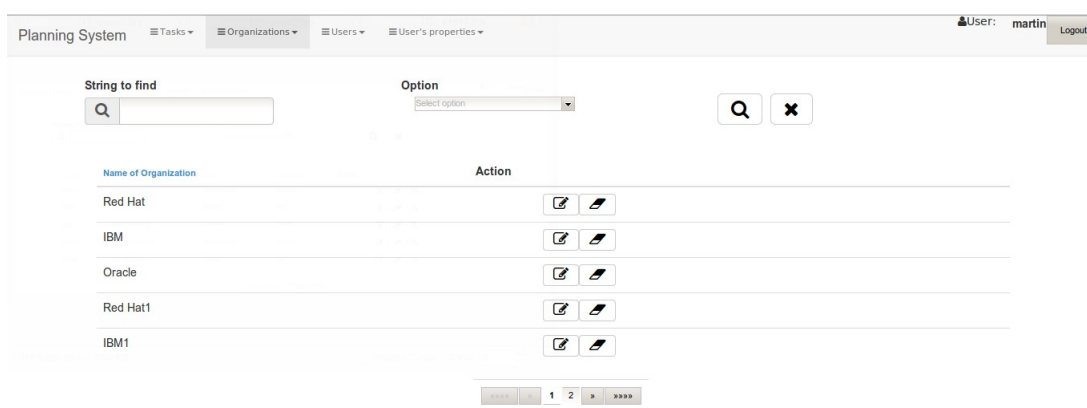
Create task

Obr. B.4: Nahrávanie novej úlohy





Obr. B.5: Spravovanie užívateľov



Obr. B.6: Spravovanie organizácií

## Dodatok C

## Dotazník

**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 aplikácie, 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 zobrazovať 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 zobrazované 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 prehľadania 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
- create.sql - SQL súbor s definíciami tabuliek
- insert.sql - SQL súbor s naplnením dát tabuliek
- bachelor\_thesis.pdf - elektronická verzia textovej časti bakalárskej práce
- 4queens.xml, cloudbalance.xml, travellingtournament.xml - definičné súbory pre plánovacie problémy
- src - adresár, ktorý obsahuje zdrojové kódy k aplikáciám pre užívateľské rozhranie(optaplanner.controller)
- docs - adresár so zdrojovými textami k dokumentácií