

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

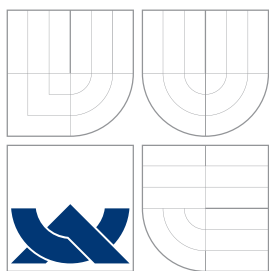
BACHELOR'S THESIS

AUTOR PRÁCE

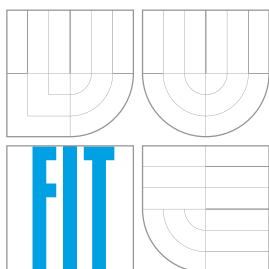
AUTHOR

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. ZDĚNEK LETKO, Ph.D.

BRNO 2014

Abstrakt

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

Abstract

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

Klíčová slova

Java EE 6, Java, Java Beans, Java Server Faces, Monitorování, Twitter, Bootstrap, Optaplanner, Webová služba, Enterprise Java Bean, JBoss, Rich Faces, Model, Komponenta, Maven, Arquillian, Plánování, MySQL, Uživatel, Uživatelská role, Omezení, Plánovací problém, Úloha Red Hat .

Keywords

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

Citace

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

System monitorování stavu plánovacích úloh

Prohlášení

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

.....
Martin Maga
18. května 2014

Poděkování

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

© Martin Maga, 2014.

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

Obsah

| | | |
|----------|----------------------------------|-----------|
| 1 | Úvod | 3 |
| 2 | Java Enterprise edition 6 | 5 |
| 2.1 | Motivácia | 5 |
| 2.2 | Špecifikácia platformy | 5 |
| 2.3 | Aplikačný model | 6 |
| 2.4 | JavaServer Pages | 7 |
| 2.4.1 | JavaServer Faces | 8 |
| 2.4.2 | JSF aplikácia | 9 |
| 2.5 | Webová služba | 11 |
| 2.5.1 | Big webová služba | 11 |
| 2.5.2 | RESTful webová služba | 12 |
| 2.6 | Princíp webových komponent | 12 |
| 2.7 | Java Persistence API | 13 |
| 2.8 | Enterprise JavaBeans | 15 |
| 2.8.1 | Session Bean | 15 |
| 2.8.2 | Message-driven Bean | 16 |
| 2.9 | Convention over Configuration | 16 |
| 2.10 | Twitter Bootstrap | 18 |
| 2.11 | Rich Faces | 18 |
| 2.12 | MySQL | 19 |
| 2.13 | Seam | 19 |
| 2.14 | Testovanie | 20 |
| 2.15 | WildFly Aplikačný server | 21 |
| 3 | OptaPlanner | 22 |
| 3.1 | Plánovací problém | 22 |
| 3.2 | Výsledky plánovacieho problému | 24 |
| 3.3 | Princíp | 24 |
| 3.4 | Konfiguráciu OptaPlanneru | 26 |

| | | |
|----------|---|-----------|
| 4 | Aplikácia | 29 |
| 4.1 | Špecifikácia požiadavkov | 29 |
| 4.2 | Analýza | 30 |
| 4.3 | Návrh aplikácie | 32 |
| 4.3.1 | Návrh modelu databáze | 33 |
| 4.3.2 | Návrh užívateľského rozhrania | 34 |
| 5 | Implementácie | 37 |
| 5.1 | Aplikácie pre užívateľské rozhranie | 37 |
| 5.1.1 | Prihlasovanie | 37 |
| 5.1.2 | Zabezpečenie | 38 |
| 5.1.3 | Komunikácie s PlannerService | 39 |
| 5.1.4 | Logika aplikácie | 39 |
| 5.1.5 | Implementácia rozhrania | 40 |
| 5.1.6 | Publikovanie úloh | 41 |
| 5.1.7 | Validácia | 42 |
| 5.2 | PlannerService | 42 |
| 5.3 | Testovanie | 43 |
| 5.4 | Vyhodnotenie aplikácie | 44 |
| 6 | Záver | 46 |
| A | Inštalácia | 49 |
| B | Užívateľské rozhranie | 51 |
| C | Dotazník | 55 |
| C.1 | Obsah dotazníka | 55 |
| D | CD so zdrojovými kódmi | 58 |

Kapitola 1

Úvod

V úvode by som Vás rád oboznámil s témou mojej bakalárskej práce Systém monitorovania stavu plánovacích úloh, ktorá bola zverejnená spoločnosťou Red Hat. Tento systém je schopný riešiť rozličné plánovacie problémy, ktoré definovaný vo formáte XML. Plánovacím problémom chápeme rovnako problémy z bežného života (napríklad optimálna cesta pre vozidlá logistickej spoločnosti), tak aj problémy z hľadiska informačných technológií (napríklad plánovanie testov na serveri). Podmienkou možnosti plánovania je existencia pravidiel a definičných entít pre konkrétny problém. Riešenie je realizovaný frameworkom OptaPlanner, ktorý na základe pravidiel a definičných entít pre danú úlohu sa pokúsi nájsť najlepšie riešenie, ktorý poskytne ako výsledok. Práca obsahuje popis monitorovacieho systému, ktorý sa skladá z grafického rozhrania, ktoré zobrazuje stav plánovacích problémov a priebežný stav výpočtu a výpočtovej časti, ktorá rieši plánovacie problémy frameworkom OptaPlanner-om, ktoré medzi sebou komunikujú. Časť realizujúca riešenie je optimalizovaná pre riešenie problému N Dám. Obe časti systému sú založené na Java EE technológiách v kombinácii s rôznymi štýlovacími frameworkami, ktoré zabezpečili prenositeľnosti užívateľského rozhrania na mobilný telefón. Užívateľské rozhranie je sprístupňované na základe role užívateľa a obsah mechanizmy, ktoré zabezpečujú systém pred zneužitím.

Rozhranie obecnne umožňuje zobrazovať informácie o úlohách, vyhľadávať úlohy podľa určitého kritéria, editovať definície úloh rovnako aj spúšťať/pozastatovať plánovanie úloh. Okrem umožňuje spravovať užívateľov a organizácie, ktoré sú sprístupnené len konkrétnej užívateľskej role. Výsledkom práce je intuitívne rozhranie s rýchlou učiacou sa krivkou, ktoré je otestované širokou škálou užívateľov, ktorý okrem toho vyjadrili svoje osobné pocity z navrhnutého rozhrania a poskytli spätnú väzbu na možné vylepšenia rozhrania. Rozhranie bolo okrem otestované na platforme UNIX prostredníctvom nástroja Arquillian a JUnit so zreteľom na citlivé časti systému.

Druhá kapitola sa venuje Java EE platforme a jej technológiám potrebných k vytvoreniu systému². Nájdete tu stručné vysvetlenie spojené poprekladané s obrázkami pre lepšiu názornosť. Na záver kapitoly bude uvedená implementácia Java EE technológií open-source kontajnerom JBoss.

V tretej kapitole bude vysvetlený systém OptaPlanner počínajúc elementárnymi čas-

ťami potrebnými k dotvoreniu celkového obrazu o probléme plánovania³. Bude bližšie vysvetlený pojem „plánovací problém“, rovnako bude rozobratý 1 konkrétny typ problému s obrázkom pre lepšie pochopenie. V tejto kapitole sa oboznámime s princípom plánovania prostredníctvom tohto frameworku, rovnako ja konfiguráciu toho frameworku.

V štvrtej kapitole sa prezentujeme špecifikáciu systému monitorovania, rovnako aj analyzuje použité technológie⁴. Následne prejdeme ku konkrétnemu návrhu a rozdeleniu systému na jednotlivé časti. V ďalšej časti uvedieme spôsob implementovania a celú kapitolu ukončíme zmienkou a testovaní aplikácie a vyhodnotení. V záverečnej kapitole zhrneme obsah celej práce, zhodnotíme jej prínos a možnosť ďalšieho rozšírenia⁶.

V sekcii príloh nájdeme postup na inštaláciu a spustenie aplikácie rovnako ako aj kompletný prehľad navrhnutého rozhrania a predloženého dotazníka^{??}.

Kapitola 2

Java Enterprise edition 6

2.1 Motivácia

Nasledujúca kapitola poskytuje prehľad o platforme Java EE 6 vrátane technológií, ktoré sú používané pri implementácii systému monitorovania. Kapitola sa zameriava na pochopenie obecného princípu vytvárania aplikácií založených na Java Enterprise Edition(Java EE) platforme. Ďalej prechádza k vysvetleniu konkrétnych technológií Java EE pre tvorbu užívateľského rozhrania, rovnako aj komunikácie medzi časťami systému. Následne sú vysvetlené obecné princípy používaný jazyk Java, na ktorom je postavená platforma Java EE spolu s nástrojmi na implementáciu a testovanie výsledného systému.

V závere kapitoly je rozobratý Java EE kontajner JBoss, ktorý je použitý pre nasadením beh a spravovanie výsledného systému^{2.15}. Dôvodom použitia jazyka a na nej založenej platforme je použitie Java EE kontajnera, rovnako aj možnosť použitia pokročilých nástrojov na testovanie a nástroja na spravovanie závislosti, ktorý je primárne určený pre jazyk Java.

2.2 Špecifikácia platformy

Základom Java EE je štandard Java Standard Edition(Java SE). Java EE predstavuje platformu a poskytuje knižnice určené zjednodušenie vývoja komplexných webových a podnikových aplikácií^[12]. Tieto aplikácie sú viacvrstvé z dôvodu lepšej prenositeľnosti, nasaditeľnosti a modifikovateľnosti. Na Java EE môžeme nahliadať ako na kolekciu špecifikáciu od Sun/Oracle. Java označuje okrem programovacieho jazyka, tak isto aj platformu. Java platforma sa skladá z virtuálneho stroja a príslušného Application Programming Interface(API). Virtuálny stroj je behové prostredie našej aplikácie zloženého z tried, ktoré bolo preložené do byte kódu. Tento virtuálny stroj býva navrhnutý pre konkrétny operačný systém. API je sada vytvorených tried, ktoré môžeme využiť pri implementácii aplikácie a sprístupňuje funkčnosť virtuálneho stroja^[11].

Základom používania Java EE aplikácií je prítomnosť štandardného API využívané enterprise aplikáciami. Java EE teda poskytuje, ktoré usnadňujú vývoj v rôznych oblastiach, či je to oblasť webových služieb(napríklad Java API for XML Web Services),

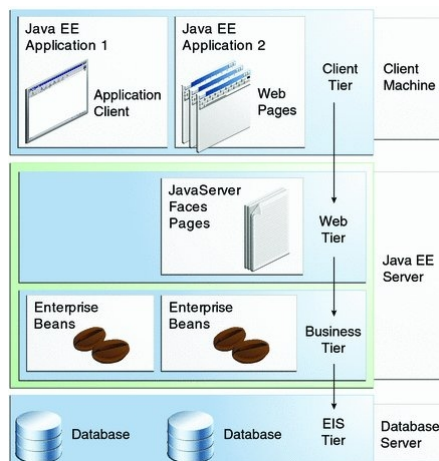
správa transakcií(napríklad Java Transaction API) alebo rôzne iné oblasti. Aplikácia, ktorá pokrýva všetky API, ktoré spĺňajú špecifikáciu Sun/Oracle pre Java EE sa nazýva aplikačný server. Aplikačný server rovnako poskytuje aj klasické služby na jeho spravovanie. Referenčnou implementáciou je server GlassFish. Základom aplikácie sú komponenty, ktoré predstavujú základné jednotky aplikácie, ktoré sa nasadzujú na server. Komponent existuje niekoľko druhou, ale len niektoré sa nasadzujú na aplikačný server. Každý aplikačný server obsahuje kontajnery, ktoré sa starajú o poskytnutie funkcionality konkrétnej komponente. Java EE platforma špecifikuje aplikačný model Java EE aplikácie, ktorá je rozdelená do vrstiev podľa funkčnosti.

2.3 Aplikačný model

Java EE definuje aplikácie, ktoré sú viacvrstvové(multitier). Pojmom viacvrstvosť je myslené rozdelenie aplikácie podľa funkčnosti na menšie celky(ktoré nazýva stupne), ktoré majú nastarosť určitú úlohu. Každá vrstva je predstavovaná inými technológiami. Vo výsledku jednotlivé vrstvy medzi sebou komunikujú a toto rozdelenie uľahčuje a zprehľadňuje presnosť vývojové cykly aplikácie. Každá vrstva je reprezentovaná komponenta, ktorá predstavuje funkčnú časť programu zostavenú z tried a súborov, ktorá je vložená do Java EE aplikácie a interaguje s inými komponentami[8]. Jednotlivé komponenty sa následne rôzne inštalujú na rôzne vrstvy v závislosti od ich príslušnosti(každý stupeň môže byť fyzicky na inom aplikačnom serveri). Jednotlivé stupne sa skladajú z rôznych komponent, pričom stupne sú rozdelené nasledovne:

- Klientský stupeň sa skladá z klientských komponent, ktoré bežia na klientskom počítači
- Stredná vrstva sa skladá z webových a podnikových komponent, ktoré bežia na Java EE serveri, ktorý predstavuje prostredie pre nasadenie, spravovanie a beh podnikových a webových komponent Java EE aplikácie
- Najnižšia vrstva predstavuje externé systémy využívané Java EE aplikáciou. Typicky sa jedná o databázový server a externé systémy označujeme názvom „Enterprise Information System(EIS)“

Typicky beží medzi klientskom a databázou častou viac-vláknový Java EE server. Na základe tohto rozdelenia môžeme uviesť, že platforma Java EE sa používa vývoj vo webovej a podnikovej vrstve, ktoré bežia na Java EE serveri. Na obrázku č. 2.1 môžeme vidieť viacvrstvové rozdelenie. Klient pristupuje k Java EE aplikácii na Java EE serveri z klientskej stanice, prostredníctvom tenkého klienta(webový prehliadač), ktorý sa nazýva „tenký klient“(pretože sa nedotazuje priamo na databázový server), alebo klientská aplikácia, ktorý sa nazýva „hrubý klient“. Tenký klient pozostáva z: Webové prehliadača, ktorý zobrazuje stránky a dynamické webové stránkz pozostávajúce z rôzneho značkovacieho jazyka, ktoré sú generované webovými komponentami. Tenký klient sa dotazuje prostredníctvom Hypertext Transfer protokolu(HTTP), čo je internetový protokol pre výmenu hyperxtoových dokumentov, na webové komponenty na Java EE serveri. Hrubý



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

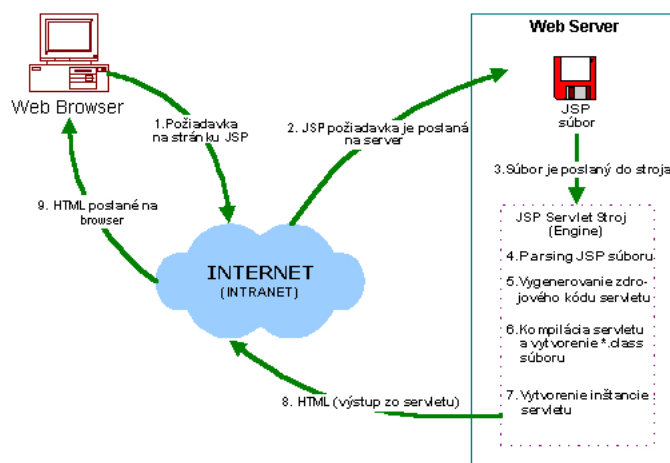
klient, ktorý môže byť reprezentovaný rozličnými Java SE technológiami pre tvorbu užívateľských rozhraní, sa môže priamo dotazovať podnikových komponent a preskočiť tak komunikáciu s webovými komponentami.

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 EnterpriseJava Beans, ktoré vytvárajú logiku aplikácie. Webová vrstva reprezentovaná je reprezentovaná webovými komponentami, ktoré spracovávajú požiadavky od užívateľa a generujú odpoveď, ktorú posielajú naspäť užívateľovi. Môžu pritom kontaktovať aj podnikové komponenty pre zistenie dodatočných informácií. Podniková vrstva je reprezentovaná podnikovými komponentami, ktoré tvoria základ aplikácie. Tieto komponenty môžu prijímať požiadavky od klienta alebo webovej vrstvy a následne generujú odpovede, pričom môžu komunikovať s najnižšou vrstvou (napríklad komunikovať s databázovým serverom). Táto 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. Vrstva býva označovaná skratkou EIS.

2.4 JavaServer Pages

JavaServer Pages(JSP) technológia je jazyk, ktorý umožňuje priamo vkladanie Java kódu do HyperText Markup Language(HTML) kódu. HTML je značkovací jazyk pre vytváranie webových stránok, ktorý obsahuje HTML značky. Pre vloženia java kódu v HTML stránke sa používajú nasledujúce značky: `<% %>` medzi, ktoré sa vloží príslušný java kód. Takéto časti v HTML stránke sa nazývajú „skriptlety“. Tieto skriptlety sú dynamické, to znamená, že sú vykonávané za behu aplikácie. Behom aplikácie je myslené nasadenie jsp stránky(stránka obsahujúca skriptlety) na Java EE server, ktorý zabezpečuje jeho vykonávanie prostredníctvom volania jsp kontajneru.



Obrázek 2.2: JSP architektúra prevzate z [<http://interval.cz/clanky/jaserver-pages-pro-vsechny/>]

Na nasledujúcom obrázku č.2.2 je zobrazený princíp technológie JSP. Základnou časťou je existencia JSP stránky a jej nasadenie na Java EE serveri. V 1.kroku existuje užívateľ, ktorý je reprezentovaný webovým prehliadačom, ktorý zažiada o JSP stránku. Java EE server prijme požiadavku od klienta a zistí, že sa jedná o požiadavku o JSP stránku. Ten zavolá JSP servlet kontajner na spracovanie žiadosti, ktorý obsahuje JavaServer Pages prekladač, ktorý obsluhuje spracovanie, kontrolu a generovanie. Ten JSP servlet stroj spracováva JSP stránku a vyhodnocuje skripty a nahradzje ich výsky HTML kódom, ktorý produkuje na výstup. Výstupom zo JSP servletu, ktorý vznikol ako požiadavka o JSP stránku je html stránka, ktorá je predaná užívateľovi, ktorý si ju zobrazí. Výhodou tejto technológie je, že pri žiadosti o JSP stránku je, že pri zmene sa nemení celý obsah stránky ale len jej časť, ktorá bola zmenená. Takže takéto JSP stránky sú dynamické a umožňujú zmenu obsahu za behu.

2.4.1 JavaServer Faces

JavaServer Faces(JSF) je framework pre tvorbu užívateľských rozhraní webových aplikácií, ktoré bežia na Java EE serveri[4]. JSF framework vytvára aplikácie na základe Model-View-Controller(MVC). MVC predstavuje softwarovú architektúru, ktorá rozdeľuje aplikáciu na dátový model, užívateľské rozhranie a riadiacu logiku do nezávislých častí. Princíp je nasledujúci:

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

Pri využití tohto frameworku programátor vkladá predpripravené komponenty(tlačidlá, vyskakovacie okná, rolovacie zoznamy, ...) a mapuje ich na príslušné triedy. JSF sa skladá z 2 častí:

- JSF API - obsahuje komponenty užívateľského rozhrania, umožňuje ich správu, validáciu vstupov, zpracovanie udalostí, navigáciu a iné
- Knižnica tagov(tag library), ktorá môže byť alternatívne nahradená JSP knižnicou tagov - prostredníctvom týchto špeciálnych tagov vkladáme komponenty užívateľského rozhrania na stránku a upravuje ich chovanie pomocou atribútov alebo mapovaním na triedy. Každá komponenta je definovaná triedou, ktorá určuje jej funkcionality. Tagy jednotlivých knižníc sú rozlišované na základe menných priestorov.

JSF umožňuje mať výstup v podobe HTML jazyka, alebo iného jazyka v závislosti od definičných tried komponent. Základnou implementáciu prevádza JSF komponenty do HTML kódu.

2.4.2 JSF aplikácia

JSF aplikácia je klasická webová aplikácia, ktorá obsahuje aj svoje špecifiká. Základná štruktúra JSF aplikácie je nasledujúca, pričom nie všetky časti sú povinné:

- Súbory značkovanie jazyka HTML alebo Extensible Hypertext Markup Language(XHTML)[6], ktoré obsahujú komponenty užívateľského rozhrania z knižnice tagov, ktoré môžu byť namapované na tzv. „managed bean-y“
- Managed Beans - java triedy, ktoré sú spravované JSF frameworkom. Najdôležitejšie sú „backing bean“, ktoré zabezpečujú funkcionality na HTML/XHTML stránke, udržiavajú stav komponent, zpracovávajú udalosti, validáciu Ich konfigurácia sa realizuje v súbore *faces-config.xml*
- Konfiguračný súbor *faces-config.xml*, v ktorom sa definujú backing beany spolu s ich typom, navigácia, validátory(java triedy, ktorá spracovávajú zadané hodnoty a generujú výstup), ...
- Popisovač nasadenia *web.xml*, ktorý umožňuje nastavenie uvítacích stránok, filtre, servlety a ...

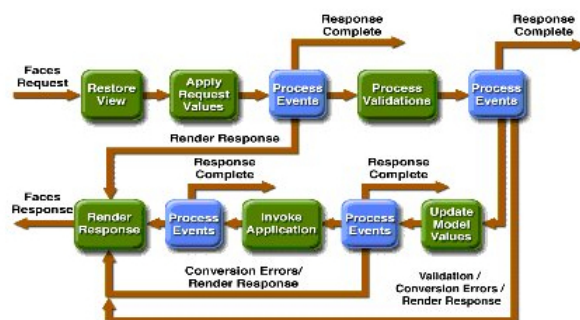
Neoddeliteľnou súčasťou tohto frameworku je Expression Language(EL), je jazyk, ktorý umožňuje dynamicky pristupovať k metódam javovských tried,(backing bean) rovnako dokáže získať a nastaviť hodnotu danej komponenty.Pri preklade sa vygenerujú závislosti na backing beans. Backing beans dokáže za behu spracovávať údaje zadané na webovú stránku, rovnako dokáže obstaráť validáciu vstupov, následne metódy a vlastnosti, ktoré sú volané alebo sú im predávané údaje z vygenerovanej stránky(HTML alebo XHTML) do backing bean-y alebo opačne.

Obecne Managed Beans môžu byť nasledujúceho typu, pričom typy managed beans sa uvádzajú v súbore *faces-config.xml*:

- @RequestScoped Managed beana prežíva pokiaľ existuje HTTP požiadavka. Vytvára sa pri vytvorení požiadavky a zaniká pri zrušení HTTP požiadavky
- @NoneScoped Managed Beana existuje tak dlho ako existuje vyhodnotenie Facelets na stránke, po vyhodnotení zaniká
- @ViewScoped Managed beana prežíva pokiaľ existuje interakcia s danou JSF stránkou. Vytvára sa pri žiadosti o danú stránku a zaniká pokiaľ užívateľ prejde na inú JSF stránku
- @SessionScoped Managed bean prežíva tak dlho pokiaľ existuje HTTP sedenie. Vytvára sa pri 1.požiadavke o danú stránku a zaniká pri invalidácii daného HTTP sedenia
- @ApplicationScoped Managed Bean prežíva dokiaľ existuje aplikácia. Je vytvorená pri prvej interakcii s aplikáciou a zaniká pri ukončení aplikácie
- @CustomScoped Managed Bean existuje dokiaľ existuje záznam o bean-e v custom Map, ktorá je vytvorená pre existenciu danej beany

V poslednom rade treba uviesť životný cyklus JSF aplikácie.

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



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

cyklus JSF aplikácie. Počas fázy Restore View, keď je kliknuté na tlačidlo alebo na link sa vytvorí náhľad stránky, spoja sa všetky spracovania udalostí, validátory a komponenty a uložia sa do inštancie FacesContext. V ďalšej fáze Apply Request Values nové hodnoty sú získané použitím metódy decode. Hodnoty sú potom uložené lokálne do komponenty. Pokiaľ nastane chyba, tak je propagovaná a generovaná do FacesContext-u. Na konci tejto fázy sa vykoná znova dekódovanie pokiaľ stály nejaké nové hodnoty vo fronte na spracovanie. Vo fáze Process Validations spracuje všetky registrované validátory ku komponentám. Pokiaľ nastala chyba tak je táto informácia uložená do FacesContext-u.

Počas ďalšej fázy Update Model Values nastavi do komponent lokálne nové hodnoty. Počas predposlednej áze Invoke Application je spracované rozličné žiadosti ako potvrdenie formulára alebo link na iný stránku. V poslednej fáze Render Response dôjde k renderu stránku s novými hodnotami v kotajnery.

2.5 Webová služba

Web Service je sotwarový systém navrhnutý na podporu inteoperability medzi rôznymi zariadeniami prostredníctvom počítačovej siete. Komunikácia prebehia prostredníctvom HTTP protokolu vymeniením Extensible Markup language(XML) správ. XML je značkovací jazyk, ktorý definuje sadu pravidiel pre kódovanie dokumentu vo formáte porozumiteľnom človeku prostredníctvom ľubovolných tagov. Webové služby poskytujú interoperabilitu medzi rôznymi platformami naprieč počítačovou sieťou. Webová služba umožňuje komunikáciu medzi rôznymi aplikáciami, ktoré bežia na rôznych platformách. Tento aspekt je umožnený tým, že aplikácie komunikujú prostredníctvom HTTP protokolu. Komunikácia prostredníctvom webovej služby sa delí na 2 účastníkov. Prvý účastník produkovateľ(producer), ktorý vytvára požiadavok a spotrebiteľ(consumer), ktorý prijíma požiadavok. Komunikácia prebieha medzi týmito dvoma účastníkmi výmenov správ. Webová služba môže byť technicky implementovaný rôznymi možnosťami a prostredníctvom Big Web Service alebo Restful Webservice, pričom v princípe ako o java triedy, ktoré obsahujú špeciálne definície triedy a metód a pri nasadení na Java EE server môžu byť vzdialenie(po sieti) zavolané ich metódy.

2.5.1 Big webová služba

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



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

komunikácie medzi klientom, ktorý sa nachádza v ľavej časti obrázku a webovou službou, ktorá sa nachádza vpravej časti obrázku. Komunikácia prebieha prostredníctvom vymie-

naní SOAP správ. Rovnako ako na klientovi tak aj web service obsahuje potrebné API, ktoré spracováva SOAP správy a predáva ich ďalej.

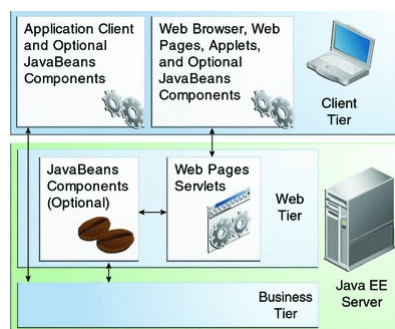
Tento typ webovej služby obsahuje definíciu vo formáte Web Service Description Language(WSDL). WSDL je definícia vo formáte XML, ktorá popisuje aké akcie webová služba poskytuje a spôsob ich invokácie, rovnako aj odpoveď. Správy volania a odpovedí web service sú vymieňané prostredníctvom SOAP správ prostredníctvom HTTP protokolu. JAX-WS API je pomerne komplikované, preto celá komplexnosť je vývojárovi zakrytá a je jediné, čo definuje vývojár sú metódy, ktoré je možné vzdialene volať. Rovnako vývojár nespracováva SOAP správy, ale celá táto problematika je riešená prostredníctvom prostredníctvom API. Veľká výhoda je platformová nezávislosť, ktorá je dosiahnutá prostredníctvom Javy. Tak isto toto API umožňuje prístup k ne-Javovským web service, čo prináša veľkú flexibilitu. Čo sa týka vývoja web service, tak sa jedná o jednoduchú Java triedu, ktorá používa anotáciu `javax.jws.WebService`, konkrétne anotáciu `@WebService`, ktorá označuje, že sa jedná o web service endpoint. Táto trieda následne definuje metódy, ktoré môžu byť vzdialené volané. Aby mohla byť metóda metódou web service musí byť anotovaná prostredníctvom anotácie `javax.jws.WebMethod` `@WebMethod`. API ponuká aj ďalšie možnosti ako ovplyvňovať životný cyklus web service.

2.5.2 RESTful webová služba

?? RESTful webová služba je druh webovej služby, ktorý pre svoju implementáciu používa API JAX-RS[12]. Tento typ webovej služby umožňuje vytvárať webové služby, ktoré sú určené pre základné a ad hoc integračné riešenia. Tento druh webovej služby rovnako nevyžaduje striktné používanie XML formátu a doručovanie správ vo formáte SOAP. K tomuto typu webovej služby je prístupované 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 obsah všetky potrebné informácie, pričom je možné označiť ako cachovateľný(uchávajúci sa vo vyrovnávanej pamäti) kvôli zvýšeniu výkonosti. Rovnako pri vytváraní klienta a služby musí byť použité rovnaké rozhranie z dôvodu explicitnej nepodporovateľnosti SOAP správ pre komunikáciu.

2.6 Princíp webových komponent

Java EE webové komponenty sú softwarové komponenty, ktoré spracovávajú prichádzajúci HTTP požiadok a poskytujú naň odpoveď. Všetky Java EE webové komponenty sú postavané na servletoch. Servlety sú javovské triedy, ktoré dynamicky spracovávajú požiadavky a tvoria odpovede. Súčasťou servletov alebo webových stránok sú technológie JavaServer Faces technológiu(JSF)2.4.1 and JavaServer pages(JSP)2.4. Technológie JavaServer Faces a JavaServer Pages podporujú spracovanie užívateľských vstupov a ich predanie a spracovanie podnikovou vrstvou.



Obrázek 2.5: Webové komponenty prevzate z [<http://docs.oracle.com/javaee/6/tutorial/doc/bnaay.html>]

Na nasledujúcom obrázku č.2.5 je ukázaný princíp fungovania webových komponent. V hornej ľavej časti obrázku sa nachádza klientská vrstva, ktorá obsahuje buď len webový prehliadač po prípade Applety, ktoré čiastočne obsahujú logiku aplikácie. Applet je aplikácia, ktorú spúšťa užívateľ prostredníctvom webového prehliadača a je vykonávaná virtuálnym strojom. V hornej pravej časti môže byť klient reprezentovaný aplikačným klientom, ktorý obsahuje úplnú prezentačnú logiku aplikácie a teda v tom prípade, odpadá potreba spracovania vstupov po prípade nejakej generovania HTML stránky. Takýto klient komunikuje už len priamo s Java EE serverom, konkrétne podnikovou vrstvou, ktorá implementuje zvyšnú logiku aplikácie a je reprezentovaný technológiou Enterprise Java Beans. V prípade, že máme k dispozícii tenkého klienta, klient komunikuje prostredníctvom webového prehliadača s HTML alebo XHTML stránky, ktoré sú vytvorené technológiou JavaServer Faces2.4 alebo JavaServer Pages2.4, ktoré spracovávajú požiadavky od klienta(vstupy) a následne komunikuje s podnikovým stupňom, ktorý obsahuje logiku reprezentovanú Enterprise Java Beans technológiou, ktorý následne môže komunikovať s databázovým serverom. Odpoveď je následne „predaná“ stránkam vytvorené prostredníctvom JavaServer Faces alebo JavaServer Pages technológiou a následne zobrazená užívateľovi v podobe výstupu webovej stránky.

2.7 Java Persistence API

Java Persistence API(JPA) je framework jazyku Java, ktorá poskytuje prístup a spravovanie dát v databáze pomocou prístupu *objektovo relačné mapovanie* [14]. JPA je nezávislé nad použitou databázou technológiou, je možné vytvárať dotazy nad MySQL, SQL databázou, ... Tento prístup umožňuje mapovanie dát medzi z databázových tabuliek na objekty javy(entity). Entita je základnou jednotkou frameworku JPA, s ktorým pracujeme pri manipulácii s dátami. Entita je je odľahčený perzistentný doménový objekt, ktorý typicky reprezentuje tabuľku v relačnej databáze a každá jej inštancia je riadkom v tabuľke. Základný artefaktom v programovaní je pre entity entitná trieda, ktorá obsahuje vlastnosti, ktoré priamo odpovedajú schéme vytvorenej databáze. Každá entitná trieda musí spĺňať určité kritéria:

- Entitná trieda priamo musí byť anotovaná `javax.persistence.Entity` anotáciou. Anotácia je reťazec obsahujúci znak `@` nasledovaný reťazcom, pričom môže v zátvorkách obsahovať ďalšie parametre, ktorý pridáva ďalšie informácie o označenej (anotovanej položke). Anotovať môže rovnako metódy, triedy ale aj vlastnosti tried.
- Entitná trieda musí mať parametrický konštruktor, aby bolo možné vytvárať nové entity
- Každá vlastnosť 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. Jednotlivé vlastnosti môžu byť dodatočne anotované kvôli kontrole na hodnotu konkrétneho typu alebo vlastnosti (nenulovosť, špeciálny formát, ...).
- Každá entitná trieda musí mať unikátny identifikátor. Týmto identifikátorom chápeme primárny kľúč, čo je vlastnosť, ktorá dokáže v databáze jednoznačne identifikovať záznam. Primárny kľúč býva anotovaný prostredníctvom anotácie `javax.persistence.Id`

Rovnako treba spomenúť, že každá entitná trieda môžu byť vo vzťahu s inými entitami. V prípade, že vlastnosť entity je súčasťou vzťahu s inou entitou použijeme niektorú z nasledujúcich anotácií podľa násobnosti vzťahu: *@One-to-one*, *@One-to-many*, *@Many-to-one*, *@Many-to-Many*. Rovnako uvedenie vlastnosť/vlastnosti druhej entity, ktoré sa podieľajú na vzťahu. To urobíme tak, že našu vlastnosť ešte anotujeme anotáciou `javax.persistence.JoinColumn`, v ktorej parametroch uvedieme názvy vlastností druhej entity, ktoré sú súčasťou vzťahu. JPA ponúka aj iné, pokročilé možnosti mapovania, pre naše potreby nám budú stačiť nasledujúce informácie.

Aby sme mohli s entitami pracovať potrebujeme si vytvoriť inštanciu triedy `javax.persistence.EntityManager`. `EntityManager` je trieda, ktorá dokáže vytvárať a odstraňovať entity, umožňuje ich vyhľadávať, rovnako aj vytvárať dotazy nad databázou. Dotazy, ktoré môžeme vytvoriť pomocou JPA sa podobajú klasickému jazyku Structured Query Language (SQL), ktorý dokáže vytvárať dotazy nad databázou, avšak dotazovací jazyk JPA má niekoľko rozdielov. Tento jazyk sa nazýva Java Persistence Query Language (JPQL), čo je ako bolo spomenuté jazyk podobný SQL, pričom tento jazyk je reťazcovo založený a je nezávislý na zvolenej databázovej technológii a objektové vlastnosti, čo znamená, že pri tvorbe dotazov používame názvy vlastností entitných tried a názvy entitných tried. Problém JPQL je typová nebezpečnosť, čo vyžaduje pretypovanie výsledkov dotazu z entity manager-a. To môže spôsobiť chyby, ktoré nemusia byť odchytené počas kompilácie. JPA definuje ešte Criteria API, ktoré je využívané k vytváraniu dotazov nad entitami a vzťahy, ktoré sú typovo bezpečné. Výhodou tohto API, pre použitie na dotazovanie, je rovnako možnosť vytvárať dynamické dotazy, ktoré majú lepšiu výkonnosť ako JPQL. Aby `EntityManager` bol schopný pracovať s určitými entitnými triedami je nutné vytvoriť *perzistentnú jednotku (persistence unit)*, čo je XML predpis, do ktorého uvedieme entitné triedy, odkaz na databázu po prípade ďalšie vlastnosti a ten vložíme do súboru `persistence.xml`. Tento súbor predstavuje konfiguráciu,

ktorá obsahuje okrem názvu entitných tried, tak aj rôzne iné vlastnosti, napr. je možné automaticke vytvoriť pri načítaní so súboru schému databáze. V tomto súbore sa rovnako nachádza dôležitá položka a to je datasource, ktorý definuje odkaz na databázu, s ktorou pracujeme. Na záver kapitoly zhrniem princíp práce s JPA:

- Vytvorenie entitných tried spolu s vlastnosťami, správne naanotovanie tried, pričom návrh entitných tried odpovedá návrhu schémy databáze, ktorý požadujeme
- Registrácia entitných tried v súbore persistence.xml, v ktorom nastaví aj odkaz na nami používanú databázu
- Vytvorenie inštancie triedy EntityManager, pričom môžeme explicitne uviesť názov perzistentnej jednotky, s ktorou pracujeme(perzistentných jednotiek môže byť viac)
- Pracujeme s databázou spôsobom, vytváraním, mazaním, editovaním hodnôt entitných tried, ktoré zapisujeme do databáze EntityManager-om, alebo vytvárame dotazy, ktoré realizujeme EntityManager-om a výsledky podľa potreby spracovávame.

2.8 Enterprise JavaBeans

EnterpriseJavaBeans(EJB) je technológia, ktorá umožňuje vytvárať komponenty, ktoré bežia v strednej, konkrétne podnikovej vrstve aplikačného modelu Java EE[5]. Takéto komponenty sú modulárne, keďže je možné ich vytvoriť a spravovať viac inštancií a môžeme do nich umiestniť logiku našej aplikácie. 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 pozerať 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. Základnou podmienkou nasadenia na Java EE server je prítomnosť EJB kontajneru, do ktorého sa inštalujú vytvorené triedy. Triedy vytvorené týmto API nazýva *Enterprise Bean-y(EB)*.

EB sa delia na 2 kategórie:

- Message-driven bean - Pôsobí v rolu poslucháča určitý typ správ, na ktorých príjem reaguje vykonaním určitých akcií
- Session bean - Vykoná úlohy pre klienta. Môže implementovať webové služby[12]

2.8.1 Session Bean

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

SB môže byť 3 typov:

- Stateful Session Bean - beany udržiava hodnoty premených, každá beana reprezentuje unikátny stav klienta/bean sedenia. Pokiaľ sa sedenie odstráni stav zmizne.

- Stateless Session Bean - Neudržiava stav komunikácie s klientom. Počas invokácie metódy takejto beanu môže inštancia obsahovať premenné, ktoré môžu obsahovať špecifický stav vzhľadom na klienta, alebo len počas invokácie metódy. Stav po ukočení mizne, rovnako tento typ SB je možné použiť k implementácii webovej služby.
- Singleton Session Bean - Tento typ beanu je inštanciovaný len raz a pretrváva počas celého životného cyklu aplikácie. Využíva sa pri zdieľaní a súčasnom prístupe viacerých užívateľov.

Tento typ beanu môžeme použiť pokiaľ potrebuje udržať stav medzi klientskými volaniami metód, rovnako pokiaľ potrebuje odľahčiť aplikáciu a zvýšiť výkonnosť použijeme tento typ beanu konkrétne stateless session bean.

2.8.2 Message-driven Bean

Message-driven bean (MDB) je typ EB, ktorá umožňuje Java EE aplikáciám asynchrónne spracovanie správ. Tento bean prijíma Java Messaging Services (JMS) správy z JMS fronty, ktoré následne analyzuje a vykonáva akcie. JMS je technológia, ktorá umožňuje komunikovať komponentám prostredníctvom správ. JMS fronty sú obyčajné fronty, do ktorých sa na jednom konci pri zavolaní MB vloží špecifická JMS správa a na druhej strane je MB postupne tieto správy odoberané a teda spracované len raz. JMS správa prostredníctvom, v ktorých sa prenášajú rôzne informácie (špecifické hodnoty, ...). JMS správa môže byť typicky viacerých typov. Správy zaradené v JMS fronte môžu byť poslané rôznymi Java EE komponentami, alebo aj iným systémom, ktorý nepoužíva Java technológiu. Tieto beanu nespracovávajú len JMS správy ale aj iné typy správ. Zásadný rozdiel je oproti session bean v zásade v tom, že sa k takému typu beanu 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. Výhodou MB je ekvivalencia MB, to znamená že správy môže EJB kontajner ľubovoľne inštancii. Klienti prístupujú k message-driven bean, napr. zasielaní správ do cieľa pre message-driven beanu je *MessageListener*. Message-driven bean má ďalšie zaujímavé vlastnosti a to, že môžu byť vyvolané asynchrónne, ktoré nevyťažujú tak prostriedky servera, žijú relatívne krátko a sú bezstavové.

2.9 Convention over Configuration

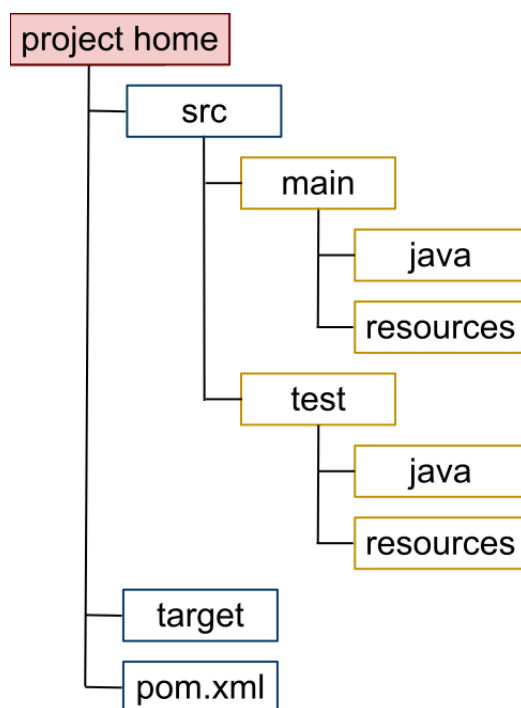
Convention over Configuration je softwarové paradigma, ktoré zjednodušuje prácu vývojárovi použitím štandardného modelu práce pre všetky projekty. Toto paradigma odľahčuje vývojára od nekonzistentnosti pri vývoji spôsobom nastavovaním rozličných konfiguračných súborov a snaží sa tento prístup zminimalizovať.

Jedným z prostriedkov, ktorý implementuje toto paradigma je Maven. Maven je stavebný automatizačný nástroj, ktorý je primárne používaný pre projekt v jazyku

Java. Maven definuje spôsob akým bude software zostavený, rovnako aj definuje závislosti(Dependency Management). Celý obsah je definovaný v XML súbore, v ktorom sa rovnako definujú závislosti na externé moduly, poradie zostavovania komponent a požadované rozšírenia.

Maven obsahuje predefinované úlohy ako kompilácia, testovanie, balíkovanie a nasadzovanie, ktoré sú štandardným vývojovým krokom každého projektu[3]. Výhodou tohto nástroja je dynamické sťahovanie javovských knižníc a rozšírení z jedného alebo viacerých repozitárov(miesto, kde sa nachádzajú java knižnice) ako napr. Maven 2 Central Repository a ukladá ich v lokálnom repozitári na disku. Následne v prípade potreby danej knižnice v projekte sa použije lokálna kópia knižnice pokiaľ je k dispozícii, v opačnom prípade dôjde k jej stiahnutiu.

Každý projekt vytváraný pomocou nástroja maven sa konfiguruje XML súboru, ktorý využíva Project Object Model(POM) a nazýva sa pom.xml, pričom sa nachádza v koreňovom adresári projektu. POM je XML súbor s konfiguráciami projektu, závislosti a inými informáciami o projekte. Každý Maven projekt spĺňa štandardnú adresárovú štruktúru.



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

obrázok č. 2.6 ukazuje základnú adresárovú štruktúru maven projektu. Každý maven projekty sa skladá z project home, ktorý obsahuje súbor pom.xml a všetky ostatné podadresáre. Ďalej sa skladá z priečinkov src, kde sa nachádzajú zdrojové kódy a target, kde sa ukladajú preložené triedy[3]. Adresár src sa ďalej skladá z adresáru main, ktorý ešte obsahuje adresára java, ktorý obsahuje java zdrojový kód pre daný projekt a resources,

ktorý obsahuje prostriedky pre daný projekt ako sú rôzne súbore, ktoré obsahujú nastavenie prostriedkov pre daný projekt. Podadresár src sa skladá z adresára test, ktorý rovnako ako src obsahuje podadresár java, ktorom je umiestnený java zdrojový kód pre testovanie projektu. Podadresár test obsahuje ďalší adresár resources, ktorý obsahuje prostriedky potrebné pre testovanie. Celá táto štruktúra predstavuje základne adresáru štruktúru pre maven projekt a tak to robí viac prenositeľný. Jednotlivé závislosti pre projekt jednoducho definuje v súbore pom.xml. Preložením projektu sa preložia všetky triedy a uložia do adresára target. Celý projekt môže byť pre väčšiu modularitu rozdelený na moduly, pričom každý modul rovnako spĺňa základnú Maven adresárovú štruktúru. Každý maven projekt je možné dať ako cieľ jednú z fázy životného cyklu. Životný cyklus môže byť z jednej fáz: 1. validate - validácia korektnosti projektu a kontrola dostupnosti potrebných informácií pre projekt ,2.kompilácia - kompilácia zdrojového kódu projektu, 3.test - testovanie zkompilovaného zdrojového kódu, táto fáza nie je vyžadovaná 4.package - zabalenie zkompilovaného projektu do balíku, napríklad jar, 4. integration-test - spracovanie a nasadenie balíku pokiaľ je to potrebné do prostredia, kde môžu bežať integračné testy, 5.verify-run - beh a overenie balíku, že spĺňa všetky kritéria pre spustenie, 6. install - inštalácia balíku do lokálneho repozitára, v prípade, že potrebuje použiť balík ako závislosť 8.deploy - nasadenie projektu do kontajneru a spustenie. Jednotlivé fázy môže spustiť príkazom „mvn názov životného cyklu“, napríklad mvn package.

2.10 Twitter Bootstrap

Twitter Bootstrap je dostupný súbor nástrojov pre vytváranie moderného webu a webových aplikácií[18]. Ponúka podporu najrôznejších webových technológií HTML, CSS, JavaScript a mnoho prvkov, ktoré je možné ľahko implementovať do svojej stránky. 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 stiahnuté kaskádové štýly a javascriptový súbor.

Výhodou týchto nástrojov je jednoduché je jeho jednoduché používanie a možnosť použitia aj na mobilných telefónoch. Podrobné vysvetlenie jednotlivých komponent nájdete na nasledujúcej adrese <http://getbootstrap.com/>, rovnako aj s príkladmi použitia.

Bootstrap obsahuje rozšírenie Font Awesome, čo je CSS framework, ktorý obsahuje rôzne grafické ikony, ktoré je možné integrovať do HTML kódu[7].

2.11 Rich Faces

Rich faces je open-source framework s podporou Asynchronous Javascript and XML(AJAX)[9], ktorý predstavuje rozšírenie JSF frameworku 2.4.1. Rich Faces obsahuje API, ktoré obsahuje grafické komponenty s podporou Ajax-u. Rich Faces je možné ľahko integrovať pomocou nástroja maven[?]. 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. Rovnako sa MySQL technológia snaží pri vykonávaní transakcií dotazy optimalizovať.

2.12 MySQL

MySQL je databázová technológia, ktorá je vhodná pre malé a stredne veľké aplikácie, rovnako poskytuje dobrý výkon pri vykonávaní transakcií. Umožňuje vytvárať procedúry, databázové triggere a jej inštalácia je pomerne jednoduchá a nezaberá veľa diskového priestoru, rovnako je MySQL multiplatformová, keďže je možné ju nasadiť na systémy s operačným systémom Windows, Linux, Mac Os. Medzi nevýhody tejto technológie patrí neefektívna práca s databázovými transakciami, neefektívne ukladanie veľkého množstva dát. MySQL je open source a je vyvíjaná spoločnosťou Sun Microsystems.

2.13 Seam

Na zabezpečenie Java EE aplikácie bol vybraný open-source framework Seam[15]. Seam je aplikačný framework pre enterprise Javu, ktorý definuje uniformný komponentný model pre podnikovú logiku aplikácie. Seam rieši integráciu EJB2.8 a JSF2.4.1 spolu. Medzi ďalšie výhodné vlastnosti tohto frameworku patrí integrácia Asynchronous JavaScript and XML(Ajax)[9], rovnako aj vstavaná podpora javascriptu a efektívne spracovanie webových dotazov.

My sa zameriame na modul Seam security, ktorý obsahuje množstvo mechanizmov na zabezpečenie našej enterprise aplikácie. Základom každej bezpečnosti je autentifikácia, čo je process vytvorenia alebo potvrdenia identity užívateľa. Užívateľ potvrdzuje svoju identitu prostredníctvom užívateľského mena a hesla. Seam security poskytuje API prostredníctvom, ktorého je možné sa autentizovať z rozličných zdrojov(databáze, ...). Ďalšou vlastnosťou je Identity Management, ktoré je množina API na správu užívateľov, skupín a užívateľských rol. Identity Managent je poskytovaný v Seam komponentou PicketLink IDM, ktorá spravuje uloženie užívateľov v rozličných bezpečnostných úložiskách. Seam security je k dispozícii v prostredníctvom nástroja Maven.

Základom autentifikácie je Identity Bean, čo je java trieda, ktorá reprezentuje identitu užívateľa a pri úspešnej autentifikácii je identita je vložená do životného cyklu aktuálneho sedenia. V rámci autentifikácie sú definované metódy *Login(prihlásenie)* a *Logout(odhlásenie)*. Základom každej triedy, ktorá realizuje autentifikáciu je metóda, v ktorej prebieha autentifikácia užívateľa. Počas autentifikácia sa overí pravosť užívateľa a v metóde authenticate sa prostredníctvom metódy *setStatus* nastaví úspech(SUCCESS) alebo neúspech(FAILURE) pri overení zadaných údajov. Po autentifikácii dôjde k vloženiu identity do životného cyklu užívateľa, ktorú je možné získať z triedy Identity prostredníctvom anotácie @Inject triedy Identity.

Seam security modul poskytuje spôsob akým zabezpečiť svoje triedy a metódy prostredníctvom anotácií tohto API.

Ďalší modul, ktorý nás zaujíma je Seam Faces, ktorý obsahuje API na zabezpečenie

prístupu k HTML a XHTML stránkam. Túto funkčnosť nazývame *Faces View Configuration*, ktorá nám umožňuje spojenie so Seam Security modulom na obmedzenie/povolenie prístupu pre danú užívateľskú rolu, prepisovanie URL, k HTML/XHTML stránkam HTML/XHTML stránky sú anotované prostredníctvom špeciálnych vlastností a umiestnené vo java triede, ktorá obsahuje výčte. Trieda je anotovaná tagom `@View-Config`. Vo výčte sú obsiahnuté anotácie na obmedzenie prístupu k HTML/XHTML stránkam. Základom obmedzenia prístupu je tvorba rozhraní, ktoré obsahujú autorizačné metódy, ktoré overujú identity daného užívateľa. Názvy týchto rozhraní použijeme ako anotácie vo výčte, kde každej stránke pridružíme názov rozhrania (overujúce identitu užívateľa), ktorému priradíme anotáciu `@ViewPattern()`, ktorá obsahuje názov XHTML/HTML stránky, ktorej je prístup povolený. Vytvorením viacerých stránok a rozhraní a následnou anotáciou môžeme sprístupniť časti systému rôznym užívateľom. V tomto výčte je rovnako definovaný postup pri neautorizovanom prístupe k stránke. Uvedením tagu `@AccessDeniedView`, do ktorého parametru vložíme názov stránky, na ktorú bude neautorizovaný užívateľ presmerovaný, k anotácii `@ViewPattern`, hovoríme ako sa má zachovať v prípade, že o danú stránku zažiada užívateľ bez potrebných oprávnení. Seam framework patrí pod divíziu JBoss, takže je pomerne jednoduché ho integrovať pod aplikačný server JBoss-u.

2.14 Testovanie

V poslednom rade uvedieme technológie, ktoré budeme používať pre testovanie výslednej aplikácie. Základom testovania je nástroj JUnit a nástroj Arquillian. V prvom rade sa budeme venovať nástroju JUnit. JUnit je unit testovací nástroj pre programovací jazyk Java. JUnit sa používa pre typ testovania, ktorý sa nazýva „test-driven development“ [16] a je jedným z kolekcie unit testovacích nástrojov. JUnit býva súčasťou balíku `org.junit` [13]. Testovacie metódy sú anotované prostredníctvom `@Test` anotácie. JUnit rovnako umožňuje vykonať kód pred spustením testu, to docielime anotovaním metód `@Before` anotáciou alebo po spustení testu, to docielime anotáciou `@After`. V testovacej metóde potom vykonáme nejaký kód a očakávaný výstup porovnáme s nami očakávaným výsledkom prostredníctvom metódy `Assert`. JUnit testy sú písané pre otestovanie konkrétnej funkčnosti kódu. Cieľom testovania prostredníctvom JUnit sú malé kúsky kódu, ako metódy alebo triedy.

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. Arquillian integruje aj ďalšie testovacie nástroje, napr. JUnit 4, TestNG 5, Treba zdôrazniť, že nariadením od JUnit testov umožňuje testovanie v java EE kontajnery (GlassFish, JBoss) [1]. Tento framework má zásadnú výhodu v prenositeľnosti testov na rôzne podporované Java EE kontajnery. Nástroj pri spustení automaticky zabalí do archívu všetky potrebné prostriedky pre platformu.

Použitie Arquillian sa deje použitím anotácie `@RunWith Arquillian` v našej javovskej testovej triede, ktoré zabezpečí spustenie testov. Následne tento nástroj spustí kontajner

a nasadí testovací archív, ktorý je daný anotáciou @Deployment. Archív obsahuje testy so špecickými triedami a knižnicami. Testy sa následne vykonajú vo vnútri kotajneru. Čo znamená, že môže použiť otestovať podnikové a webové komponenty za behu. Písanie testov s nástrojovou Arquillian začína tvorbou javovskej triedy, ktorá vyzerá ako štandardná testovacia trieda vytvorená nástrojom JUnit, pričom obsahuje vyššie spomenuté špecifické anotácie, ktoré umožňujú pri spustení testu vytvorenie archívu, nasadenie na Java EE kontajner a následne spustenie testov. Aby tento nástroj mohol byť použitý je nutné mať k dispozícii všetky potrebné prostriedky(knižnice), ktoré môžeme získať prostredníctvom nástroja maven a následne je potrebné nakonfigurovať v XML súbore arquillian.xml použitie Java EE kontajnera. Arquillian.xml je xml súbor, ktorý použítie Java EE kontajneru a ďalšie špecifické vlastnosti.

2.15 WildFly Aplikačný server

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

WildFly je open-source aplikačný server verzie 8, ktorý vznikol premenovaním aplikačného serveru JBoss, čo je vlastne skratka pre JavaBeans Open Source Application Server. Pre naše potreby budeme používať WildFly v verzii 7, preto bude používaný názov JBoss. JBoss je aplikačný server, ktorý je založený na platforme Java a Java Enterprise Edition.[10]. Tento typ AS je open-source, preto je možné jeho stiahnutie spolu so zdrojovými kódmi. Používanie aplikačného servera JBoss je veľmi jednoduché jeho spustenie môžete vykonať ručne prostredníctvom konzole a nájdeným inštalačným adresárom JBoss-u a následne adresárom bin, ktorý obsahuje skript run.sh, ktorý spustí AS. Druhou možnosťou je spustenie prostredníctvom IDE. Po spustení serveru je možné k nemu implicitne pristupovať na localhost na porte 8080 počítača. Základným stavebným kameňom JBoss AS je JBoss Microcontainer. JBoss Microcontainer je refaktORIZÁCIA JBoss JMX Microkernel aby podporoval POJO nasadzovanie a samostatné použitie mimo aplikačného servera. Microcontainer plní funkciu jadra, do ktorého sa registrujú všetky služby. Služby, ktoré majú byť prístupné sa registrujú v podobe managed beans. Microcontainer spravuje a riadi beh týchto služieb. Prostredníctvom rozhrania Java Management Extension je možné tento server spravovať. JBoss implicitne podporuje databázový server Hypersonic SQL, ktorý má ale obmedzené možnosti a preto je určený len na testovanie. JBoss je licenovaný pod GNU Lesser General Public License(GNU PL).

Kapitola 3

OptaPlanner

OptaPlanner je open source framework a pokračovanie frameworku JBoss Drools, ktorý vykonáva a optimalizuje rôzne plánovacie problémy, ktoré sú reprezentované XML súborom, s rozličným stupňom náročnosti. Optaplanner využíva pri riešení problému, ktoré nemusí vždy nájsť, optimalizačné heuristiky 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. Tento framework neurčuje striktné akými algoritmami a metódami sa má daný problém vyriešiť, ale konfiguráciu ponecháva na strane užívateľa. OptaPlanner je určený pre jazyk Java, preto riešenie je vykonávané triedami v tomto jazyky. Tieto triedy sú špecifické pre daný problém, a musia byť schopné získať potrebné informácie z defičného súboru problému, ktorý reprezentuje zadanie problému, musia byť schopné vykonávať postupné kroky vedúce k riešeniu problému (napr. v prípade problému N Dám presúvať dámy, tak aby sa vždy nachádzali vo validných pozíciách) a prostriedky, ktoré ohodnotnia krok a pričítajú ho ku celkovému skóre.

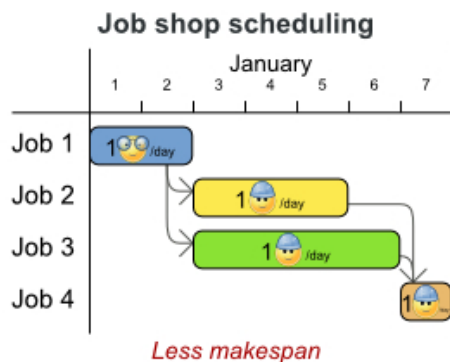
Samozrejme postup riešenia problému, kalkulácií skóre sa opakuje pre rôzne scenáre (napr. v prípade N Dám pre rôzne kombinácie pohybov) a vráti sa riešenie s najlepším skóre v podobe súboru vo formáte XML (napr. v prípade riešenia problému N Dám poskytne najlepšie možné riešenie). OptaPlanner sa snaží vždy nájsť optimálne riešenie vzhľadom k optimalizačným algoritmom a metaheuristickým metódom a dostupnému času, ale niekedy nie je schopný poskytnúť na predchádzajúce podmienky optimálne riešenie. Výhodou tohto frameworku je možnosť aplikovania na NP-úplne problémy. OptaPlanner je dostupný prostredníctvom nástroja Maven.

3.1 Plánovací problém

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

V bežnom živote, rovnako ako ja v podnikových sférach sa stretávame s rôznymi plánovacími problémami. Môže ísť o problémy ako správne naplánovať cestu vozidiel (aút,

lodí,...), aby sme ju spravili za čo najkratší čas, rovnako môžeme požadovať aby cesta bola, čo finančne najefektívnejšia. Rovnako môžeme plánovanie rozvrhu práce zamestnancov vo firme, aby zbytočne nespomalovali chod a ostatných zamestnancov, ktorí sú na ich práci závislí nemuseli zbytočne čakať. Plánovať môžeme spúšťanie testovania aplikácií v rámci vývojarskej firmy, aby niektoré úlohy boli otestované skôr ako iné no musí byť čo najefektívnejšie vyvážené a zbytočne nemrhali časovým kvantom. Pokiaľ je problém dostatočne komplexný potom je veľmi vhodné použiť Optaplanner.



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

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

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

3.2 Výsledky plánovacieho problému

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

OptaPlanner podporuje niekoľko optimalizačných algoritmov ako efektívne nájsť tieto veľké množstvá riešení. V závislosti na prípade použitia, niektoré optimalizačné algoritmy dosahujú lepšie výsledky ako ostatné, ale to je nemožné povedať dopredu. Pri plánovaní, je ľahké prepnúť algoritmus optimalizácie, zmenou konfigurácie Solver-u.

3.3 Princíp

Princíp riešenia problému je založené na konfigurácii OptaPlanner tvorbou javovských tried na získanie potrebných dát z definičného súboru, prostriedkov na kalkuláciu skóre a aplikáciu odkiaľ spúšťami výpočet. Riešenie problémom sa začína tvorbou XML definičného problému špecifického pre daný problém. Následne sa vytvoria triedy pre získanie dát z XML súboru, triedy pre vykonávanie krokov(napr. v prípade N dám presúvanie dám na validné pozície) a prostriedky pre kalkuláciu skóre a nastavenie konfiguračného súboru pre daný problém, ktorý bude bližšie popísaný v nasledujúcej kapitole 3.4. Aby bolo jasné aké akcie sú povolené pre daný problém sú definované v triedach pre riešenie obmedzenia: [2]

- Negatívne "hard"obmedzenie, ktoré nesmú byť porušené
- Negatívne "soft"obmedzenie, ktoré by nemali byť porušené pokiaľ sa dá tomu vyhnúť.
- Pozitívne "soft"obmedzenia, ktoré by mali splnené pokiaľ je to možné(môžu viesť k lepšiemu skóre)

Prostriedky pre kalkulácie skóre môžu byť 3 typov:

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

Drools kalkulácia skóre využíva vlastnú DRL syntax a je daná súborom, ktorý obsahuje pravidlá. 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. Treba zdôrazniť, že v konfiguračnom súbore užívateľ nastavuje optimalizačné algoritmy a metaheuristické metódy, ktorý sa snažia v spolupráci s triedami na riešenie vyberať vždy najlepšie kroky pri riešení.

Spustenie riešenia je dané zavolaním hlavnej metódy odkiaľ sa spúšťa riešenie problému. a spustí vykonávanie(plánovanie). Postup je nasledovný:

1. Overenie prostriedkov(definičného súboru, konfiguračného súboru(spôsob kalkulácie, definičného triedy, použitie plánovacích algoritmy a metaheuristických metód) a prostriedkov na kalkuláciu skóre)
2. Načítanie sa obsah XML súboru
3. Vykonanie kroku podľa použitia plánovacích algoritmov
4. Optimalizácia kroku v prípade použitia metaheuristických metód
5. Ohodnotenie kroku(v závislosti od použitia prostriedkov na kalkuláciu skóre3.3)
6. Vykonanie alternatívneho kroku vzhľadom(napr. v prípade N Dám presunutie dámy na ľavú stranu šachovnice, miesto pravej)
7. Optimalizácia alternatívneho kroku v prípade použitia metaheuristických metód
8. Ohodnotenie kroku(v závislosti od použitia prostriedkov na kalkuláciu skóre3.3)
9. Opakovanie krokov 3., 4., 5., 6. až dokým nie je dosiahnuté riešenie alebo plánovanie nie je predčasne ukončené
10. Nájdenie riešenia alebo predčasné ukončenie plánovanie vzhľadom na vysoké poskytnuté skóre(je možné použiť v prípade, že riešenie problému vzhľadom na dostupný čas a použitie plánovacích algoritmov nebolo nájdené)

3.4 Konfiguráciu OptaPlanneru

Konfigurácia OptaPlanner sa realizuje prostredníctvom XML súboru, ktorá má 3 povinné časti a 4. voliteľnú. Pre lepšiu prehľadnosť je uvedená ukážka konfiguračného súboru.

Listing 3.1: Vyváženie cloudu

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

Nastavenie konfiguračného súboru(solver) pre riešenie problému vyváženie cloudu pozostáva z 3 častí:

- Nastavania definičných tried plánovacie problému, nastavania tried zabezpečujúce plánovanie, nastavenie definícií skóre a nastavenie použitia plánovacích algoritmov, po prípade nastavenia metaheurestických metód
- Súbor je rozdelený na 3 časti:
 - Domain model configuration(začínajúc riadkom č.4), v ktorom sú uvedené triedy definujúce problém a riešenie
 - Score Configuration(začínajúc riadkom č.7) definujúce spôsob kalkulácie skóre vrátane triedy
 - Optimalization algorithms configuration(začínajúc riadkom č.13) sú uvedené optimalizačné algoritmy, vrátane spôsobu ukončenia plánovia
- Na riadku č.3 uvedená medzi značkami *enviromentMode* hodnota „FAST_ASSERT“, ktorá umožňuje OptaPlanner detekovať chyby v implementácii
- Na riadku č.5 je uvedená medzi značkami *solutionClass* hodnota „org.optaplanner.examples.cloudbalancing.domain.CloudBalance“, ktorá odkazuje na definičnú triedu modelu problému vyváženie cloudu
- Na riadku č.6 je uvedená medzi značkami *planningEntityClass* hodnota „org.optaplanner.examples.cloudbalancing.domain.CloudProcess“, ktorá odkazuje na triedu, ktorá realizuje riešenie(plánovanie) problému
- Na riadku č.9 je uvedená medzi značkami *scoreDefinition* hodnota „HARD_SOFT“, ktorá hovorí, že pri kalkulácii skóre použijeme len hard obmedzenia [3.3](#)
- na riadku č.10 je uvedená medzi značkami *simpleScoreCalculatorClass* hodnota „org.optaplanner.examples.cloudbalancing.solver.score.CloudBalancingSimpleScoreCalculator“, ktorá odkazuje na trieda, ktorá kalkuluje skóre pri riešení problému
- Na riadku č.11 je uvedená medzi značkami *scoreDrl* hodnota „/org/optaplanner/examples/cloudbalancing/solver/cloudBalancingScoreRules.drl“, ktorá odkazuje na Drools definíciu kalkulácie skóre [3.3](#)
- 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 vrátenia 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 [\[17\]](#)

- Na riadku č. 20 je uvedená medzi značkami *pickEarlyType* hodnota „FIRST_NON_DETERIORATING_SCORE“, ktorá označuje použitie pri kalkulovaní skóre najprv nezhoršujúce sa skóre (ohodnotenie, ktoré zvyšuje hodnotu celkového skóre)
- Na riadku č. 25 je uvedená medzi značkami *entityTabuSize* hodnota „entityTabuSize“, ktorá značí použitie metaheuristickej metódy pri riešení TABU SEARCH[17], s veľkosťou tabuľky 7
- Na riadku č. 28 je uvedená medzi značkami *acceptedCoundLimit* hodnota „1000“, ktorá označuje počet náhodných krokov, ktoré sú vyhodnotené počas 1 kroku riešenia problému

Kapitola 4

Aplikácia

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

4.1 Špecifikácia požiadavkov

V tejto kapitole postupne rozobereme požiadavky na systém monitorovania úloh. Základnou úloh systému je monitorovanie úloh. Na jednej strane bude systém schopný zobrazovať stav plánovacích úloh, na druhej stranej bude môcť systém plánovacie úloh spúšťať/pozastaviť. Úlohy bude možné triediť podľa určitého kritéria, rovnako systém bude schopný aj úlohy vyhľadávať. Jednotlivé úlohy je možné aj mazať, alebo zmeniť definíciu plánovacieho problému^{3.1} a úlohu znovu spustiť. Novú úlohu bude možné do systému vložiť a následne spustiť. Úlohy bude môcť systém publikovať, čo sa myslí akcia, ktorá vytvorí pre úlohu špeciálne URL, na ktoré po kliknutí zobrazí stránku z názvom úlohy a obsahom XML definičného súboru. Úlohu bude možné aj odpublikovať. Systém bude rozdelený podľa užívateľ do 3 užívateľských rolí(Administrátor, Plánovač, Čitateľ). Užívatelia sú organizované do väčších celkov(organizácie). 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. Užívateľ si bude môcť ľubovoľne meniť heslo v systéme. Vytvorený užívatelia sa do systému prihlasuje, pričom po prihlásení je sprístupnená len časť systému podľa užívateľskej role prihláseného užívateľa. Aplikácia bude obsahovať bezpečnostné mechanizmy, ktoré zabezpečujú aplikáciu proti neautorizovanému prístupu užívateľov. Vstupmi do systému budú:

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

Výstupom systému je zoznam plánovacích úloh v prehľadnej tabuľke, rovnako aj zoznam užívateľov a organizácií, ktoré sa rovnako zobrazujú v prehľadnej tabuľke. V predposlednom rade treba spomenúť, že výslednej rozhrania bude schopné byť prenositeľné na mobilné telefóny.

V poslednom rade treba uviesť rozsah úloh, ktoré bude môcť každá užívateľská vykonávať:

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

Poslednou podmienkou bolo zvoliť vhodný prístup k databáze, ktorý by bol univerzálny a teda nezávislý na použitej databázovej technológii.

4.2 Analýza

Výslednú aplikáciu môžeme rozdeliť na 2 časti: 1. backend aplikácie, ktorý beží na Java EE serveri JBoss 2. frontend aplikácie grafické užívateľské rozhranie. Zameriame sa najprv na grafické užívateľské rozhranie. Pri analýze grafického užívateľského rozhrania je potrebné vyriešiť problém jeho návrhu a možnosti jeho interakcie. Použitie technológie JSP2.4 by síce pripadalo do úvahy, problém je že táto technológia neposkytuje žiadne grafické komponenty a jeho interakcia s inými komponentami je pomerne komplikovaná. Z tohto dôvodu bola použitá technológia JSF2.4.1, ktorá spĺňa túto podmienku. 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 technológia je pomerne málo konfigurovateľné. Lepšie riešenie poskytuje použitie frameworku Rich Faces2.11, ktorý priamo integruje Ajax[9], do všetkých jeho komponent. Posledným problémom, ktorý treba pri analýze grafického užívateľského rozhrania vyriešiť je prenositeľnosť na mobilné zariadenia. V tom nám pomôže framework Twitter Bootstrap. 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
- iOS: Chrome, Safari
- Mac OS X: Chrome, Firefox, Opera, Safari
- Windows: Chrome, Firefox, Internet Explorer(verzia 8 - 11), Opera, Safari
- Linux: Chromium, Firefox

Tento framework sa vždy snaží podporovať najnovšie verzie všetkých vyššie uvedených prehliadačov. Podpora ostatných prehliadačov nie je odporúčaná z dôvodu neočakávaného chovania. Ako rozšírenie bol použitý CSS framework Font Awesome??, ktorý obohacuje rozhranie o grafické ikony.

V druhej časti sa zameriame na problémy backend-u aplikácie. Celá aplikácia potrebuje udržiavať a spravovať dáta. Dáta je 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-om2.15. Keďže nároky na vyťaženosť prístupu k dátam, rovnako aj množstvo uložených dát sú malého merítka bolo vhodné vzoliť k tomu adekvátnu databázovú technológiu a tou technológiu je MySQL2.12. Následne treba spomenúť problém komunikácie s grafickým užívateľským rozhraním. Grafické užívateľské rozhranie potrebuje komunikovať s databázou odkiaľ získava aktuálne informácie o úlohách, užívateľoch a organizáciách. Rovnako sa do databázy zapisujú priebežné informácie o plánovaní. Vzhľadom na podmienku nezávislosti použitia databázovej technológie bola použitá technológia JPA2.7. Rovnaký prístup k databáze využíva aj OptaPlanner. Užívateľské rozhranie je schopný spúšťať plánovanie systému OptaPlanner, ktoré je optimalizované pre riešenie problému N Dám. K tomuto rozhraniu je prístupované prostredníctvom webovej služby2.5 prostredníctvom HTTP protokolu. Z dôvodu použitia štandardných komunikačných protokolov a nižším nákladom na prevádzkovanie bola zvolená „Big“ webová služba2.5.1. Užívateľské rozhranie predstavuje klienta, ktorý volá metódy na spustenie a pozastavenie výpočtu. OptaPlanner obsahuje koncový bod, ktorý zachytáva správy od klienta a zabezpečuje spúšťanie/pozastavenie výpočtu(plánovania). Výsledné užívateľské rozhranie bolo potrebné zabezpečiť voči neautorizovanému prístupu. Existuje priamo zabezpečiť aplikáciu pomocou štandardného API Java EE, no bol zvolený framework Seam2.13, ktorý možno jednoducho integrovať pod JBoss.

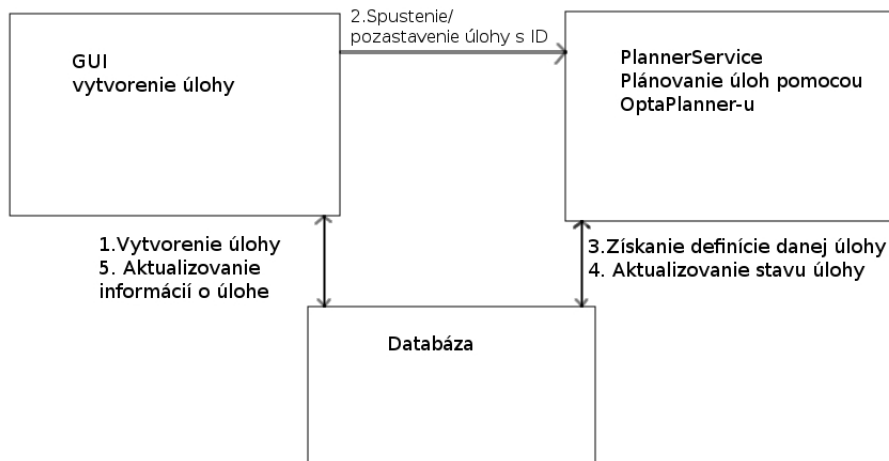
Koncový bod webovej služby je reprezentovaný v podobe session bean-y2.8.1, ktorá obsahuje funkčnosť pre spustenie a zastavenie výpočtu. Pri spustení výpočtu sú informácie predávané message-driven bean2.8.2, ktorá zabezpečuje spúšťanie plánovania prostredníctvom OptaPlanner-u3.

Pre publikovanie má byť vytvorená RESTful webová služba??, ktorá bude namapovaná na URI „task/parameter“ a parameter predstavuje ID úlohy, ktorá sa má zobraziť. Chovania má byť v prípade verejnej úlohy zobrazenie úlohy a to informácií o jej mene a definičnom súbore a v prípade privátnej úlohy vrátenie prázdnej HTML stránky.

Kvôli závislosti časti systému PlannerService na entitných triedach, bola aplikácia užívateľského rozhrania na multimodulový projekt. Modulom bol pritom balík s entitnými triedami, ktorý bol nainštalovaný ako závislosť do lokálneho repozitára prostredníctvom nástroja maven. PlannerService následne len má uvedený vo svojom konfiguračnom súbore(pom.xml)2.9 závislosť na modul entitných tried.

4.3 Návrh aplikácie

Výsledná aplikácia je rozdelená na 2 časti. Na časť reprezentujúci grafického užívateľského rozhranie s podporou prihlasovanie a užívateľských rol, zabezpečenia proti neautorizovanému prístupu. Rovnako je schopné zobrazovať úlohy, užívateľov a organizácie podľa užívateľskej role. Rozhranie pravidelne aktualizuje informácia o úlohach, užívateľoch a organizáciach z databáze. Pre spustenie výpočtu úlohy komunikuje pomocou webovej služby s „PlannerService“(optimalizovaná pre riešenie problému N Dám), ktorá implementuje spracovanie informácií. Pri požiadavke o spustenie/pozastavenie výpočtu spracovania úlohy sa predá v HTTP požiadavky ID úlohy. Webová služba následne zaradí požiadavok o spustení do jms fronty. Message-driven bean-a následne postupne odoberá požiadavky z fronty a vyhodnocuje. Pritom najprv nájde potrebný XML definičný súbor v databáze a spustí výpočet pomocou OptaPlanner. Priebežné informácie(čas do skončenia plánovania, pokrok vo výpočte) sú priebežne vkladane do databáze, čo umožňuje užívateľovi prostredníctvom rozhrania sledovať stav úlohy. Pozastavenie úlohy dôjde prostredníctvom zmeny stavu vo webovej službe, čo pozastaví plánovanie.



Obrázek 4.1: Diagram komunikácie

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

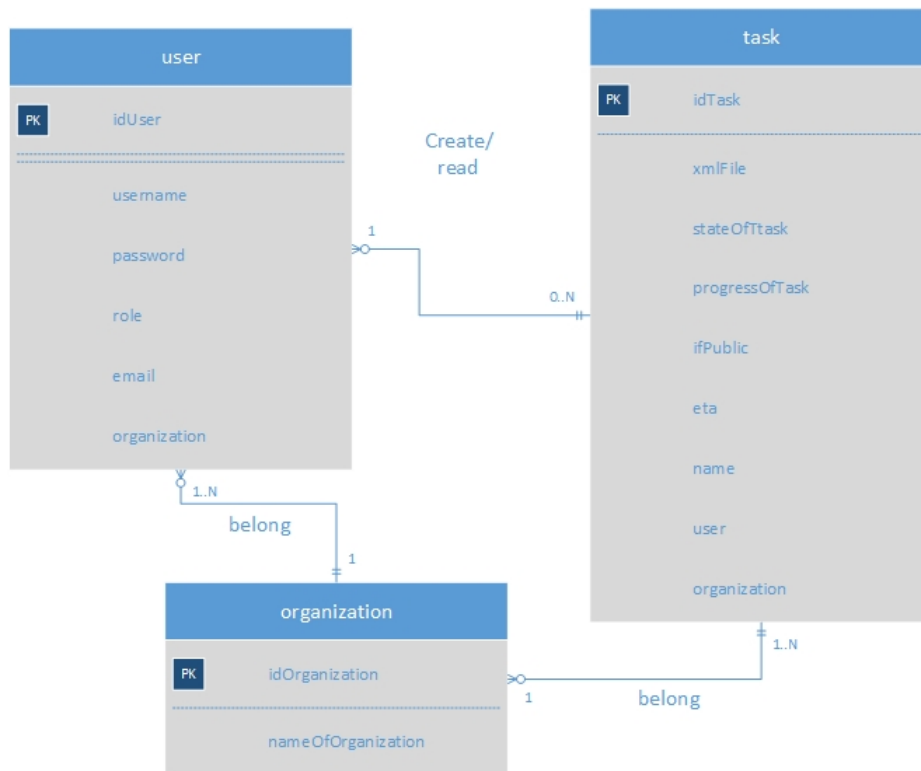
Celý návrh aplikácie bol otestovaný prostredníctvom skupiny odborných a laických užívateľov s cieľom zdôrazniť rýchlu učiacu sa krivku užívateľského rozhrania. Rovnako boli použité štandardné prostriedky na otestovanie funkčnosti kúsky kódu pomocou JUnit testov a nástroju Arquillian. Následne prebiehalo testovanie prostredníctvom užívateľov, ktorý testovali validáciu vstupov, prihlasovanie, správne vyhľadávanie jednotlivých entít(úloh, užívateľov, organizácií).

4.3.1 Návrh modelu databáze

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

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

- NEW - úloha bola vytvorená
- MODIFIED - xml súbor bol modifikovaný
- WAITING - úloha čaká na spracovanie



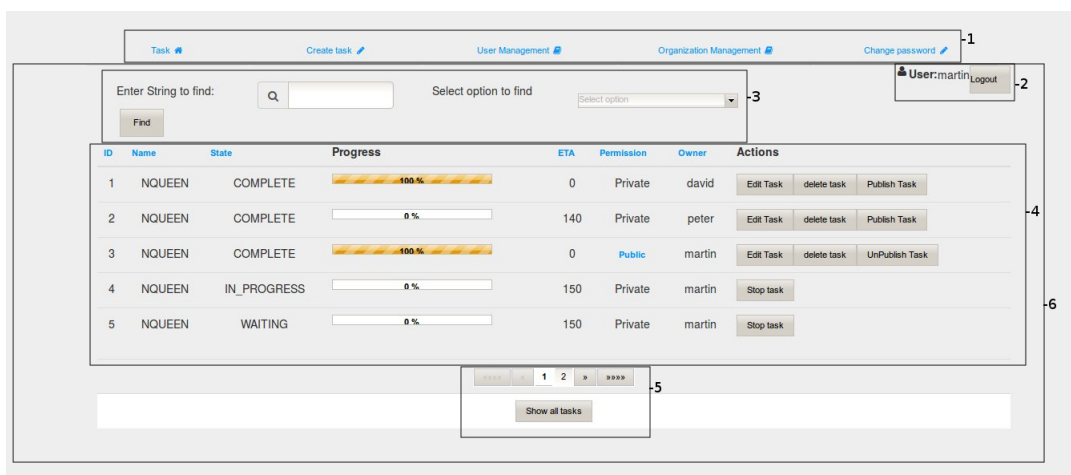
Obrázek 4.2: ER diagram

- 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 súkromnú alebo verejnú(ifPublic), názov úlohy(name) a cudzie kľúče user, ktorý odkazuje na užívateľa, ktorým bola úloha vytvorená a organization, ktorá odkazuje na organizáciu užívateľa, ktorým bola vytvorená. V ďalšej kapitole sa pozrieme na use case diagram.

4.3.2 Návrh užívateľského rozhrania

Výsledné rozhranie kladie dôraz na jednoduchosť a prehľadnosť zobrazených úloh. Z tohto dôvodu boli implementované mechanizmy vyhľadávania úloh, organizácií a užívateľov. Rovnako možnosti lexikografického triedenia. Po prihlásení do systému jednotlivé možnosti sú následne zakomponované do záložiek, v ktorých je prístupné príslušná funkčnosť. Výsledné rozhranie je prenositeľné aj na mobilné zariadenie. Užívateľské rozhranie je



Obrázek 4.3: Návrh užívateľského rozhrania

popísané na nasledujúcom obrázku: Na obrázku č.4.3 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 záložiek podľa ich názvu. Pre kliknutie na príslušnú záložku sa zmení aj obsah na stránke.
- Oblasť č.2 obsahuje informáciu o prihlásenom užívateľovi , rovnako obsahuje aj tlačidlo „Logout“, prostredníctvom ktorého sa môže užívateľ z aplikácie odhlásiť
- Oblasť č.6 predstavuje funkčnú oblasť. Táto oblasť je špecifická pre každú záložku, ktorá reprezentuje jej obsah. V tej oblasti sú umiestnené typicky obsahy databázových tabuliek, nástroje na vyhľadávanie, rôzne akcie, ktoré je možné vykonávať s dátami, rovnako aj možnosti na vytváranie entít
- Oblasť č.3 predstavuje jednu z funkčných možností. Jedná sa o vyhľadávanie, ktoré je zložené zo vstupného prvku, do ktorého zadáme vyhľadávaný reťazec a druhá časť predstavuje menu, z ktorého zvolíme stĺpec na vyhľadávanie. Následne je možnosť realizovať tlačidlom Find, ktoré prekreslí obsah tabuľky nižšie a naplní ju nájdenými výsledkami.
- Oblasť č.4 predstavuje tabuľky, ktorá je dynamicky obnovená a reaguje na asynchronné ukladanie dát z web service, ktoré sa dynamicky obnovujú každé 4 sekundy. Tabuľka je rozdelená do stĺpcov. Názvy stĺpcov, ktoré sú označené modrou farbou sú zároveň odkazy, na ktoré je možné kliknúť. Po kliknutí na daný odkaz dôjde k lexikografickému zoradeniu obsahu tabuľky podľa daného stĺpca striedavo vzostupne alebo zostupne. Rád by som upozornil na stĺpec progress, ktorý pre každú úlohu zobrazuje stav spracovania úlohy. Rovnako musím zdôrazniť stĺpec Permission,

ktorý zobrazuje, či je úloha verejná alebo privátna. Pokiaľ je úloha verejná(Public), tak je tento odkaz zobrazený modrou farbou, čo znamená, že je odkaz preto je možné naň ho kliknúť. Po kliknutí sa zobrazí stránka s informáciami o názve úlohy a obsahuje výsledného xml súboru. Tento odkaz je možné následne ľubovoľne preposlať a pristupovať k nemu. V poslednom rade treba zdôrazniť stĺpec „Actions“, ktorý je najdôležitejší pre každú úlohu povoľuje sadu akcií. Jednotlivé akcie sú reprezentované tlačidlami, pritom odrážajú aktuálny stav spracovania úlohy spolu s ďalšími informáciami o úlohe.

- Oblasť č.5 predstavuje komponentu na stránkovanie, aby pri rozsiahlom obsahu sa nezväčšoval neúmerne veľkosť stránky.

Zvyšné návrhy rozhrania pre vytvorenie úlohy, editovanie úlohy, spravovanie užívateľov, spravovanie organizácií, zmenu hesla a prihlasovanie je možné dohľadať v prílohe.

Kapitola 5

Implementácie

Nasledujúca kapitola pojedná o oboch častiach systému pre monitorovanie stavu plánovacích úloh. Najprv rozobereme aplikáciu pre užívateľské rozhranie 5.1 a následne PlannerService 5.2, ktorá zabezpečuje riešenie plánovacích úloh prostredníctvom OptaPlanner-u. Pre technológiu MySQL bol zvolený MySQL server vo verzii 5.5.37. Obe časti boli založené na nástroji maven s použitím vývojového prostredia JBoss Developer Studio vo verzii 7.1.0 GA. Na koniec treba spomenúť, že správnu komunikáciu systému s databázou je potrebné nastaviť v konfigurácii JBoss-u datasource[10], v ktorom sa nastaví odkaz na databázu spolu s prihlasovacími údajmi. Pre používanie JMS fronty v časti systému PlannerService 5.2 je potrebné nakonfigurovať v konfigurácii JBoss-u frontu s názvom *OptaPlanner*. V poslednom rade pre použitie zvolenej databázovej technológie je potrebné stiahnuť potrebnú knižnicu a vložiť adresára *deployments*.

5.1 Aplikácie pre užívateľské rozhranie

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

5.1.1 Prihlasovanie

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

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

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

- Rola Administrator bude presmerovaná na stránku *Administrator.xhtml*
- Rola Planner bude presmerovaná na stránku *Planner.xhtml*
- Rola Reader bude presmerovaná na stránku *Reader.xhtml*

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

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

5.1.2 Zabezpečenie

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

5.1.3 Komunikácie s PlannerService

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

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

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

5.1.4 Logika aplikácie

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

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

Všetky backing beany sú súčasťou balíka *org.jboss.optaplanner.controller.beans*. Aby beany mohli byť správne používané je potrebné ich uviesť v súbore *faces-config.xml*. To sa urobí uvedením medzi značky *managed-bean*, kde uvedieme názov beany, triedu vrátane cesty v hierarchii balíkov a typ beany(ktorý bol zvolený na session). Backing bean-y obsahujú metódy a vlastnosti, ktoré bola zobrazované/prevzaté z komponent na .xhtml stránkach. Všetky vlastnosti museli spĺňať princíp POJO. Pre potreby získavania dát z databázy bola použitá trieda *databaseOp*, ktorá je súčasťou balíku *org.jboss.optaplanner.controller.database*, pričom vytvára inštanciu triedy *EntityManager*, ktorý využíva entitné triedy. Táto trieda obsahuje metódy na vytváranie úloh, užívateľov, organizácií, rovnako aj mazanie, editovanie jednotlivých položiek, rovnako aj získavanie. Tieto dáta sú následne predávané backing bean-ám podľa potreby.

5.1.5 Implementácia rozhrania

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

Pri implementáciu `xhtml` pre užívateľské role sa zameriam na užívateľskú rolu Administrátor, keďže rola Plánovač a Čitateľ prevzali všetku implementáciu a komponenty práve od Administrátor, ale len v obmedzenom množstve, teda komponenty vrátane akcií, ktoré mohli vykonávať. XHTML stránka sa skladá v hornej časti z menu, ktoré je implementované ako záložky prostredníctvom Twitter Bootstrap-u. V pravej hornej časti sa nachádza informácia o prihlásenom užívateľovi vrátane tlačidla na odhlásenie.

Pri kliknutí na záložky sa zobrazí obsah, ktorý odpovedá názvu záložku. Záložky „user management, task, organization management“ obsahujú komponenty `h:dataTable` z knižnice JSF pre zobrazenie dát. Tieto dáta sú pravidelné obnovované z databáze, čo zabezpečuje ich aktuálnosť prostredníctvom komponenty `a4j:poll`, ktorá je vytvorená pre každú tabuľku a pravidelne volá metódu, ktorá získava údaje z databáze. Každá z tých záložiek obsahuje pole pre vyhľadávanie pričom je možné zvoliť podľa, ktorého stĺpca sa bude vyhľadávať. Výsledky sa zobrazia do tabuľky(`h:dataTable`) pričom zobrazené položky budú odpovedať nájdeným výsledkom. Pri vyhľadávaní sa preruší obnovenie obsahu tabuliek a zobrazí sa informácie o vyhľadávanom reťazci a časovom razítku kedy bolo vyhľadávanie realizované. S vyhľadanými položkami je rovnako možné realizovať všetky akcie. Pre potreby opätovného obnovenia obsahu tabuľky je potrebné stlačiť tlačidlo pod tabuľkou s názvom *showX(Users, Tasks, Organizations)*, ktoré sa nachádza na bielom páse pre rýchlejšie zorientovanie užívateľa. Toto tlačidlo spôsobí pre danú tabuľku(`users,taks,organizations`) získanie aktuálnych dát z databáze zavolaním metódy z triedy *databaseOp* `getAllX(Tasks/Users/Organizations)`, ktoré vytvorí dotaz na získanie aktuálnych dát z databáze. Tieto dáta sú predané príslušnej tabuľke a zároveň sa obnoví obnovenie obsahu tabuliek.

Pri každej položke v tabuľke je možné vykonávať isté akcie ako je vymazať danú entitu(`task,user,organization`), po prípade ju editovať, alebo vykonávať množstvo iných akcií. Akcie pritom reflektujú individuálny stav danej entity. Pri každej z tých záložiek okrem `task`(ktorú v zápätí rozoberem) je možné entity aj vytvárať. Vytváranie je veľmi jednoduché, keď užívateľ vyplní všetky polia, ktoré musí mať daná entita sa zavola metódu z `backing bean`(napr. na editovanie `editTask`, na vytvorenie organizácie `createOrganization, ...`), ktoré spôsobí zavolanie metódy z triedy *databaseOp*, ktoré zabezpečia vytvorenie novej entity.

Každú tabuľku je možné aj radiť. Radenie prebieha kliknutím na názov stĺpca tabuľky(zvýraznený modrou farbou), pričom daný stĺpec implementuje funkciu radenia pre daný stĺpec. Pri kliknutí na názov stĺpca dôjde k zavolanie metódy(napr. pre stĺpec ID sa zavola metóda `sortById`), ktorá je daná atribútom `action` v grafickej komponente *h:commandLink*. Metóda radenia je implementovaná prostredníctvom triedy *Collections*,

ktorá obsahuje metódu `sort`, ktoré triedia model(trieda, ktorá obsahuje rovnaké položky ako príslušná databázová tabuľka) danej entity, ktorá vytvorí komparátor, ktorý porovná 2 položky daného modelu a upraví ich poradie.

Vytváranie úloh(taskov) je zaradené do samostatnej záložky kvôli lepšej prehľadnosti. Užívateľ vyplní názov a prostredníctvom komponenty na nahrávanie súboru z knižnice Rich Faces nahrá obsah do databázy. Ďalej rozoberem záložku `change password`, ktorá umožňuje si pre daného užívateľa zmeniť heslo, vyplní pritom heslo a potvrdenie heslo a heslo sa zmení. Nakoniec rozoberem záložku `edittask`, táto záložka je pri bežnom prehliadaní nevidelná je to spravené kvôli bezpečnosti. Táto záložka sa aktivuje editovaním úlohy v záložke `task` v tabuľke tlačidlom `Edit Task`, ktorá nás prepne do záložky `Edit Task`, v ktorej sa už aktivuje obsah a užívateľ vyplní názov úlohy, vlastníka úlohy a nakoniec edituje xml súbor úlohy. Potvrdením sa vytvorí úloha so stavom „MODIFIED“.

5.1.6 Publikovanie úloh

Ďalšou podstatnou časťou aplikácie pre užívateľské rozhranie je možnosť publikovať/odpublikovať úlohu(task). Túto akciu je možné realizovať prostredníctvom tlačidla v tabuľke `Publish Task/Unpublish Task`. Tieto tlačidlá nie sú vždy prístupné, podmienkou je, že úloha je nastavená ako prívátna a nachádza v stave „MODIFIED“ alebo „COMPLETE“. Naopak odpublikovanie úlohy je možné kedykoľvek podmienkou je, aby úloha bola nastavená ako verejná(public). Publikovanie je realizovaná zavolaním metódy „`publishTask`“. V tejto metóde dôjde k zmene stavu úlohy na verejnú, pričom informácia sa uloží do databázy. Následne sa v stĺpci `permission` zobrazí text *Public* modrou farbou, ktorý po kliknutí zobrazí názov úlohy a XML súbor plánovacej úlohy. Pričom po kliknutí sa prejde na odkaz *url aplikácie/task/id úlohy*. Na časť URL `task/id` je namapovaná rest webová služba, ktorá je súčasťou balíku *org.jboss.optaplanner.controller.restservice*, kde sa nachádza trieda *RESTPublishTask*, ktorá reprezentuje práve túto službu starajúcu sa o publikovanie úloh. Táto úloha obsahuje v anotácii `@Path()` len znak „/“, čo znamená že sa namapuje na akékoľvek URL, ale namapovanie na reťazec `task` sa realizuje v súbore `web.xml`. Táto služba obsahuje 1 metóda „`getUserById`“, ktorá dostane ako parameter ID úlohy. Toto id úlohy získa zo zadaného URL. Dôležitou anotáciou je anotácia `@Produces()`, ktorá obsahuje hodnotu „`text/html`“, ktorá hovorí, že vrátená odpoveď metódy bude HTML súbor a teda výsledok bude zobrazený v prehliadači. Táto metóda na svojom začiatku vytiahne informácie o úlohe(názov, XML súbor a povolenie). Na základe povolenia určí, či je úloha nastavená ako verejná, ak nie je vráti prázdnu stránku. V prípade, že vráti stránku, ktorá obsahuje informáciu o názvu úlohy a XML súbor. Prístup k tomuto k tejto službe nie je podmienený prihlasovaním.

Pritom dôjde k vytvoreniu stránky, ktorej názov odpovedá ID úlohy a do súboru(stránky), do ktorého sa zapíše názov úlohy a obsah XML súboru. Táto stránka je následne prístupná pod URL `http://optaplanner.controller/task/ID.html`. Pričom k tejto stránke pristupovať kedykoľvek až dokým nie je úloha odpublikovaná. V tom prípade dôjde k zmazaniu súboru.

5.1.7 Validácia

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

5.2 PlannerService

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

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

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

Princíp fungovania tejto časti systému je nasledujúca:

- Klient (aplikácia užívateľského rozhrania) zadá požiadavku na spustenie/pozastavenie úlohy, ktorú realizuje zavolaním metódy webovej služby PlannerService s ID plánovacej úlohy
- Metóda *runTask* PlannerService vytvorí spojenie s triedou *OptaPlannerMessageBean*, a následne vložením správy s ID úlohy do JMS fronty *OptaPlanner*

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

Na koniec spomeniem princíp fungovania metódy na pozastavenie vykonávania plánovania:

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

5.3 Testovanie

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

V ďalšej časti prebiehalo testovanie medzi konkrétnymi užívateľmi. Išlo o 4 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 užívateľovi s rolou Administrator, Planner a Reader.

Užívatelia nasledne testovali vytváranie užívateľov, organizácií, úloh. Následne mohli sledovať stav spracovania plánovacích úloh. Aplikáciu otestovali pod 2 prehliadačmi a to Google Chrome vo verzii 34.0 a Mozilla Firefox verzie 28.0. Bol použitý operačný systém linux 3.13.0-24-generic s operačným systémom Kubuntu 14.04. Aplikácia sa správala pod obomi rovnako a korektne. Po odhalení chýb boli chyby ohlásené a odstránené a aplikácia bola následne opäť nasadená. Tento postup sa opakoval až dokým neboli odhalené všetky chyby. Na záver zhrniem testy, ktoré boli užívateľmi realizované:

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

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

5.4 Vyhodnotenie aplikácie

Po testovacej fáze nasledovala fáza vyhodnotenia aplikácie. Cieľovej skupine bol po opravení chýb aplikácie predložený dotazník, do ktorého výplňami rôzne informácie, kde dávali spätnú väzbu, chyby v návrhu, rovnako aj v intuitívnosti ovládania. Cieľovou skupinou aplikácie sú užívatelia bez akejkoľvek predchádzajúcej skúsenosti s touto aplikáciou s

vekým rozsahom medzi 20 - 40 rokov. Preto bola aplikácia predložená najprv užívateľom skúseným, ktorým bol poskytnutý predchádzajúci styk s aplikáciou a lacikým užívateľom, ktorý nemali žiadny predchádzajúci styk. Výsledkom zistenia, rovnako vyplývajúce z výsledkou dotazníka je že užívateľské rozhrania až na niektoré časti je veľmi intuitívne. Užívatelia sa ihneď zorientovať vykonať danú akciu, vytvoriť užívateľa, organizáciu, úlohu. Problém na, ktorý narazili bolo zorientovať pri vyhľadávaní úloh/užívateľov/organizácií a nájsť tlačidlo pod tabuľkou a obnoviť všetky údaje v tabuľke. Rovnako oceňovali možnosť zobrazovanie tlačidla *Save Changes* pri editovaní tabuľky vedľa položky, ktorá je práve editovaná v danom riadku. Pri vyhľadávaní ocenili užívatelia zachovania zadaných informácií pre vyhľadávanie. V záložke user management(správa užívateľov) navrhovalo presunutie tabuľky s užívateľmi na začiatok, keďže sa nachádzala na neprehľadnom mieste. Užívatelia ocenili možnosť radenia tabuliek po kliknutí na daný stĺpec aj spôsob realizácie. Užívatelia by ocenili pri úlohách mať možnosť informácie o časovom razítke vytvorenia úlohy. Prehliadania pomocou tabuliek im prišlo ako veľmi vhodné rovnako aj použitie stránkovania. Užívateľom chýbala možnosť vyhľadávať podľa viacerých kritérií.

Aplikácia by mohla byť upravená do užívateľsky prívetivejšieho rozhrania. Rovnako by PlannerService mohla byť rozšírená o spracovanie aj iných typov plánovacích úloh, minimálne tie, ktoré sú podporované štandardnou implementáciou frameworku Opta-Planner, napr. plánovanie práce, problém obchodného cestujúceho, . . .

Kapitola 6

Záver

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

Literatura

- [1] Ament, J. D.: *Arquillian Testing Guide*. Packt Publishing, 2013, ISBN 978-1782160700.
- [2] Bali, M.: *Drools JBoss Rules 5.X Developer's Guide*. Manning Publications, 2007, ISBN 978-1933988344.
- [3] Company, S.: *Maven: The Definitive Guide*. O'Reilly Media, 2008, ISBN 978-0596517335.
- [4] David Geary, C. S. H.: *Core JavaServer Faces (3rd Edition)*. Prentice Hall, 2010, ISBN 978-0137012893.
- [5] Debu Panda, D. L., Reza Rahman: *EJB 3 in Action*. Packt Publishing, 2013, ISBN 978-1782161264.
- [6] Felke-Morris, T.: *Web Development and Design Foundations with XHTML, 5th Edition*. Addison-Wesley, 2010, ISBN 978-0132122702.
- [7] Gandy, D.: Font Awesome [online].
<http://fortawesome.github.io/Font-Awesome/>, 2014-11-03 [cit. 2013-10-25].
- [8] Gupta, A.: *Java EE 6 Pocket Guide*. O'REILLY, 2012, ISBN 978-1-449-33668-4.
- [9] III, A. T. H.: *Ajax: The Definitive Guide*. O'Reilly Media, 2008, ISBN 978-0596528386.
- [10] Javid Jamae, P. J.: *JBoss in Action: Configuring the JBoss Application Server*. Manning Publications, 2008, ISBN 978-1933988023.
- [11] Jendrock, E.: *The Java EE 6 Tutorial: Basic Concepts (4th Edition) (Java Series)*. Addison-Wesley Professional, 2010, ISBN 978-0137081851.
- [12] Jendrock, E.; Cervera-Navarro, R.; Evans, I.; aj.: The Java EE 6 Tutorial [online].
<http://docs.oracle.com/javaee/6/tutorial/doc/>, 2013-11-03 [cit. 2013-10-25].
- [13] Kaczanowski, T.: *Practical Unit Testing with JUnit and Mockito*. Tomasz Kaczanowski, 2005, ISBN 978-1584504184.

- [14] Merrick Schincariol, M. K.: *Pro JPA 2*. Apress, 2013, iISBN 978-1430249269.
- [15] Michael Juntao Yuan, T. H.: *JBoss Seam: Simplicity and Power Beyond Java EE*. Prentice Hall, 2007, iISBN 978-0131347960.
- [16] Piroumian, V.: *Test Driven Development: By Example*. Addison-Wesley Professional, 2002, iISBN 978-0321146533.
- [17] Skiena, S. S.: *The Algorithm Design Manual*. Springer, 2012, iISBN 978-1849967204.
- [18] Thilo, M. O. J. T. C. R. J.: Twitter Bootstrap [online].
<http://getbootstrap.com/>, 2013-12-16 [cit. 2013-12-27].

Príloha A

Inštalácia

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

- JBoss aplikačný server najmenej vo verzii 7.1.1.Final. Je možné ho zdarma stiahnuť z <http://jbossas.jboss.org/downloads>
- MySQL Connector/J minimálne vo verzii 5.0.8, ktorý je súčasťou CD
- MySQL databázový server najmenej vo verzii 5.5.37(na distribúcií ubuntu je možné nainštalovať príkazom „apt-get install mysql-server mysql-client“, alebo je možné postupovať podľa nasledujúceho návodu <http://dev.mysql.com/doc/refman/5.1/en/linux-installation.html>)
- Webový prehliadač Mozilla Firefox najmenej vo verzii 29.0 alebo Google Chrome najmenej vo verzii 34.0(v prostredí ubuntu je možné ho nainštalovať nasledujúcim príkazom „apt get install firefox/google-chrome“)

V prvom rade je potrebné nainštalovať MySQL server nakonfigurovať databázu s názvom „optaplanner“ s užívateľským menom „root“ a heslom „root“. Následne je potrebné rozbalíť stiahnutý JBoss aplikačný server na súborový systém. V ďalšom kroku nastaví náš aplikačný server pre správne použitie MySQL databáze. To urobíme zkopírovaním súboru standalone-full.xml do adresára JBOSS_HOME¹/standalone/configuration a prepíše aktuálny obsah súboru.

Následne treba ovládač mysql-connector-java-5.1.29-bin.jar do adresára JBOSS_HOME/standalone/deployments. Následne treba vytvoriť potrebnú databázu to a naplniť ju dátami :

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

¹ JBOSS_HOME predstavuje koreňový adresár serveru JBoss na disku

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

Nakoniec skopíruje súbor `standalone.conf` do adresára `JBOSS_HOME/bin` a prepíše aktuálny obsah súboru.

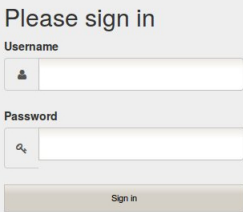
Aplikáciu môžeme spustiť nasledovne:

- Skopírujeme adresára `optaplanner.controller.war` do adresára `JBOSS_HOME/standalone/deployments`
- Skopírujeme súbor `PlannerService.war` do adresára `JBOSS_HOME/standalone/deployments`
- Prejdeme do zložky `JBOSS_HOME/standalone/bin` a spustíme skript `standalone.sh`

K aplikácií pristúpime zadaním adresy „`http://localhost:8080/optaplanner.controller.war/`“ do webového podporovaného webového prehliadaču.

Příloha B

Užívateľské rozhranie



Please sign in

Username

Password

Sign in

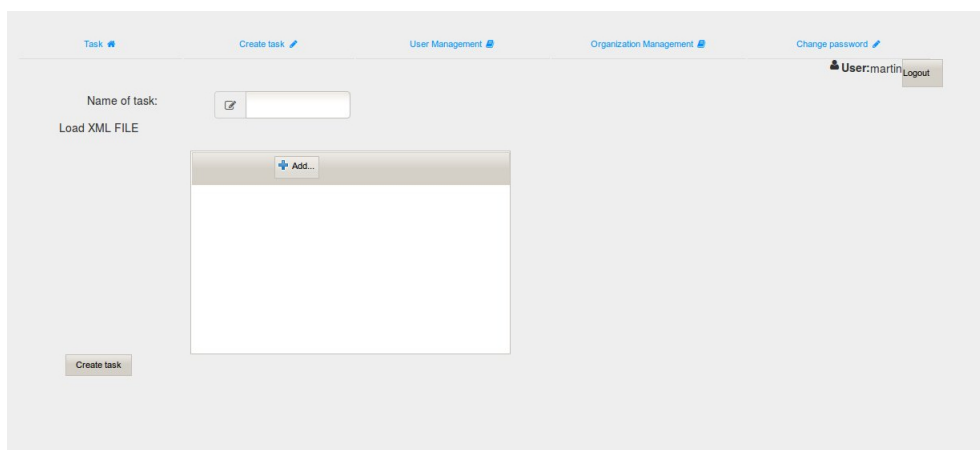
Obrázek B.1: Prihlasovacia obrazovka

The screenshot shows a web application interface with a top navigation bar containing links: Task, Create task, User Management, Organization Management, and Change password. The 'Change password' link is active. In the top right corner, the user is logged in as 'User:martin' with a 'Logout' button. The main content area contains a 'Password :' label above a text input field with a search icon. Below it is a 'Confirm password :' label above another text input field with a search icon. At the bottom of the form is a 'Change password' button.

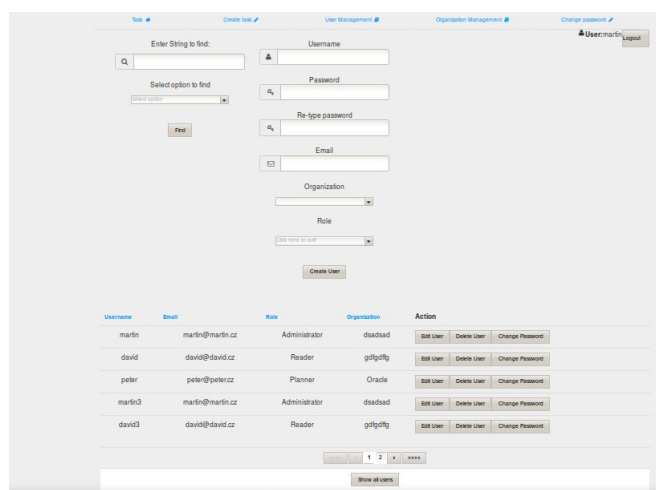
Obrázek B.2: Zmena hesla

The screenshot shows the 'Edit Task' form in the same web application. The top navigation bar is identical, but the 'Edit Task' link is now active. The form fields include: 'Name of Task:' with a text input and a search icon; 'Enter owner' with a dropdown menu showing 'david'; and a large text area containing '<null>'. A 'Save Changes' button is located at the bottom of the form. The user 'User:martin' remains logged in.

Obrázek B.3: Editovanie úlohy



Obrázek B.4: Nahrávanie novej úlohy



Obrázek B.5: Spravovanie užívateľov

Task

Create task

User Management

Organization Management

Change password

User: martinLogout

Enter String to find:

Organization

Select option

Create organization

Find

| ID Organization | Name of Organization | Action |
|-----------------|----------------------|---|
| 1 | dsadsad | <div>Edit OrganizationDelete Organization</div> |
| 2 | gdfgdfg | <div>Edit OrganizationDelete Organization</div> |
| 3 | Oracle | <div>Edit OrganizationDelete Organization</div> |
| 4 | Red Hat1 | <div>Edit OrganizationDelete Organization</div> |
| 5 | IBM1 | <div>Edit OrganizationDelete Organization</div> |

12>>>>

Show all organization

Obrázek B.6: Spravovanie organizácií

Příloha C

Dotazník

C.1 Obsah dotazníka

Grafické uživatelské rozhraní pro systém OptaPlanner

Som študentom tretieho ročníka FIT VUT v Brne. Mojou bakalárskou práce je tvorba uživatelského rozhrania pre Systém monitorovania plánovacích úloh. Ide o aplikácie, ktorá dokáže spúšťať plánovanie pre problém N dám, nahrať XML súboru N dám 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

Obrázek C.1: Všeobecné informácie

Vaše preferencie

Aké informácie by ste chceli zobrazovať pri správe užívateľov ?

☐ Užívateľské meno

☐ Posledný čas prístupu do systému

☐ Užívateľské heslo

☐ Other:

Aké informácie by ste chceli zobraziť pri správe organizácií ?

☐ Názov organizácie

☐ Jednoznačný identifikátor organizácie

☐ Čas vytvorenia organizácie

☐ Užívateľ, ktorý vytvoril organizáciu

☐ Other:

Aké informácie by ste chceli zobrazovať pri úlohách?

☐ Názov úlohy

☐ Jednoznačný identifikátor úlohy

☐ Čas do konca spracovania

☐ Percentuálne hodnotenie spracovania

☐ Kedy bola úloha spustená

☐ Stav úlohy

☐ Definičný súbor danej úlohy

☐ Other:

Obrázek C.2: Preferencie

Aký systém triedenia úloh by Vám najviac vyhovoval ?

☐ Všetky úlohy na jednom mieste a možnosť lexikografického zoradenia

☐ Zobrazenie časti 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"/> |

Obrázek 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 vyhľadávanie úloh, užívateľov, organizácií ?

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

Aký dojem máte s prehliadania úloh, užívateľov, organizácií ?

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

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

Aká funkcionality vám chýbala?

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

Obrázek C.4: Závěrečné hodnocení

Příloha D

CD so zdrojovými kódy

Priložené CD obsahuje nasledujúce súbory:

- optaplanner.controller.war - adresár JSF aplikácie obsahujúci potrebné zdrojové kódy a .xhtml stránky vrátane konfiguračných súborov
- PlannerService.war - adresár s EJB a web service, ktorý obsahuje zdrojové kódy vrátane konfiguračných súborov
- install.txt - súbor s popisom inštalácie
- create.sql - sql súbor s definíciami tabuliek
- insert.sql sql súbor s naplnením dát tabuliek
- bachelor_thesis.pdf - elektronická verzia textovej Časti bakalarskej práce
- mysql-connector-java-5.1.29-bin.jar - ovládač pre prácu s MySQL databázou
- 4queens.xml, 8queens.xml, 16queens.xml - definičné súbory pre plánovací problém 4, 8 a 16 dām