VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

WEBOVÉ ROZHRANÍ PRO SPRÁVU A MONITOROVÁNÍ ÚLOH NA SUPERPOČÍTAČI

THESIS TITLE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **DENIS BUKOVINSKÝ**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MARTA JAROŠ**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Bukovinský Denis**
Program: Informační technologie
Název: **Webové rozhraní pro správu a monitorování úloh na superpočítači**
Web Interface for Task Management and Monitoring on a Supercomputer
Kategorie: Informační systémy

Zadání:

1. Seznamte se s programovacím jazykem Python a jeho knihovnami Peewee, Flask, Unittest.
2. Seznamte se se štábní kulturou a metodikou návrhu software ve výzkumné skupině SC@FIT (GitLab).
3. Prostudujte softwarový balík k-Dispatch, zaměřte se na integraci požadované funkčnosti.
4. Navrhněte webové rozhraní pro administrátory systému, zdokumentujte nutné zásahy do struktury databáze.
5. Navržené řešení naimplementujte a vygenerujte dokumentaci.
6. Otestujte navržené webové rozhraní pomocí jednotkových testů i typických uživatelských scénářů.
7. Zhodnoťte navržené řešení a jeho přínos pro administrátory superpočítačových systémů.

Literatura:

- Dle pokynů vedoucí.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 až 4 zadání.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Jaroš Marta, Ing.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 26. října 2018

Abstrakt

Jedným z cieľov práce bolo dokončiť webové rozhranie pre aplikácie pristupujúce k databáze systému k-Dispatch, ktorý monitoruje bežiace a naplánované úlohy na superpočítači. Ďalším cieľom bolo vytvoriť webové grafické rozhranie pre administrátora systému, ktoré používa implementované webové rozhranie. Pomocou tohto grafického rozhrania môže administrátor spravovať a dohliadať nad systémom.

Po preštudovaní špecifikácie webového rozhrania som implementoval požadovanú funkčnosť, ktorú som integroval do systému k-Dispatch. Administrátorské rozhranie bude obsahovať funkčnosť na základe diagramov prípadov použitia, ktoré som vytvoril zo špecifikácie webového rozhrania. Toto rozhranie používa webové rozhranie systému k-Dispatch, vďaka ktorému má prístup do databázy.

Webové rozhranie bolo implementované v jazyku Python s použitím microframeworkov. Administrátorskú časť aplikácie tvoria dynamické web stránky vytvorené pomocou HTML, kaskádových štýlov a JavaScriptu. Server tieto dynamické stránky generuje pomocou pred vytvorených šablón.

Abstract

One of the goals of the project was to complete the web interface for applications that access the k-Dispatch database, which monitors both running and scheduled tasks on a supercomputer. Another goal was to create a web-based graphical interface for the system administrator using already implemented web interface. Using this graphical interface, the administrator can manage and supervise the system.

After studying the web interface specification, I implemented the required functionality that I integrated into the k-Dispatch system. The admin interface will include functionality based on the use-case diagrams that I created from the Web interface specification. This interface uses the k-Dispatch web interface to access the database.

The web interface was implemented in Python using microframeworks. The admin part of the application consists of dynamic web pages created using HTML, cascading styles and JavaScript. The server generates these dynamic pages using pre-created templates.

Kľúčové slová

Python, Flask, JavaScript, web server, web stránka, webové rozhranie, REST API, PostgreSQL, Objektovo relačné mapovanie

Keywords

Python, Flask, JavaScript, web server, web page, web interface, REST API, PostgreSQL, Object-relational mapping

Citácia

BUKOVINSKÝ, Denis. *Webové rozhraní pro správu a monitorování úloh na superpočítači.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Marta Jaroš

Webové rozhraní pro správu a monitorování úloh na superpočítači

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pani Ing. Marta Jaroš. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Denis Bukovinský
16. mája 2019

Poďakovanie

Chcel by som poďakovať pani Ing. Marte Jaroš za jej vedenie, poskytnutie materiálov a trpezlivosť so mnou.

Obsah

Kapitola 1

Úvod

Táto práca sa zameriava na dokončenie webového aplikačného rozhrania systému k-Dispatch a administrátorské webové grafické rozhranie skladajúce sa z niekoľkých dynamických web stránok.
k-Dispatch vznikol pre potreby jednoduchého realizovania vysoko výkonných výpočtov. Dnes je potrebné tieto úlohy realizovať distribuovane a paralelne na niekoľkých procesoroch súčasne. Realizácia tohto prístupu je ale veľmi zložitá pre užívateľov, ktorí nemajú dostatočné IT vzdelanie z tejto oblasti. Preto vznikol k-Dispatch, ktorý je riešením týchto problémov týkajúcich sa aplikácie vysoko výkonných výpočtov mimo odboru IT.
k-Dispatch využíva webové aplikačné rozhranie pre komunikáciu s užívateľskými aplikáciami, napríklad aplikácia pre administrátora celého systému. Webové rozhranie je implementované vo webovom serveri, ktorý je súčasťou k-Dispatch a používa protokol HTTP pre komunikáciu. Webové rozhranie je tvorené moderným RESTful prístupom, ktorý je dynamický, flexibilný a pri zmenách na strane klienta zostáva logika serveru zachovaná a naopak. Dáta sú ukladané v PostgreSQL relačnej databáze, obsahujú napríklad informácie o užívateľoch, výpočtových prostriedkoch a ďalšie dôležité informácie. Dáta, ktoré si k-Dispatch vymieňa s klientmi sú vo formáte JSON, ktorý je často používaný spolu s prístupom REST. JSON je oproti formátu XML jednoduchší, rýchlejší na čítanie a písanie.
Klientské aplikácie môžu byť napríklad desktop aplikácie ale aj webové aplikácie používajúce akýkoľvek webový prehliadač. Práve táto vlastnosť webových aplikácií - používanie webového prehliadača ako klienta, im dáva výhodu oproti desktop aplikáciám pretože sú dostupné z rôznych zariadení a operačných systémov, pričom nie je potrebná ich inštalácia na každom novom počítači.
Webové rozhranie je implementované v jazyku Python s použitím frameworkov, ktoré uľahčujú autentizáciu užívateľov, zjednodušenie prístupu k dátam v databáze a ďalšie časté úlohy pri vývoji. S už používanými frameworkmi dokončím webové rozhranie a rozšírim ho o URL pre webové grafické rozhranie administrátora. Administrátor bude mať k dispozícii grafické rozhranie, pomocou ktorého bude môcť spravovať systém jednoducho zo svojho webového prehliadača. Po vytvorení potrebného rozhrania pre túto webovú aplikáciu vytvorím šablóny dynamických web stránok, ktoré používa Python framework pracujúci podobne ako PHP, s pomocou HTML, kaskádových štýlov, JavaScriptu a špeciálnych blokov, ktoré používa framework. Tieto dynamické web stránky budú generované zo šablón a tie budú plnené dátami z databázy, preto musia byť dynamické a nemôžu byť statické. Takto vytvorená web stránka sa pošle ako odpoveď prehliadaču.

Kapitola 2

k-Dispatch

2.1 Úvod do problematiky

Problematika vysoko výkonných výpočtov, z anglického spojenia *high-performance computing* ďalej len HPC, je oblasť, ktorá sa zaoberá zvýšením výkonu pri náročných výpočtoch použitím paralelizácie na viacerých procesoroch a akcelerátoru, napríklad grafické karty. Výpočty, ktoré sa zadávajú týmto zariadenia obsahujú viaceré úlohy s možnými vzájomnými závislosťami, ktoré by mal užívateľ zohľadniť pri spúšťaní výpočtu. To si ale vyžaduje dostatočnú znalosť v oblasti superpočítačov, vysokovýkonných výpočtov, plánovania a paralelizmu, aby sa investované hodiny do prípravy aj vrátili v podobe správnych výsledkov. Systém k-Dispatch rieši tieto problémy a preto uživatelia nemusia mať toľko znalostí z tejto oblasti. Služby k-Disptach sú klientskými aplikáciami poskytované pomocou webového rozhrania pomocou ktorého budú spravovať alebo používať takýto systém[8].

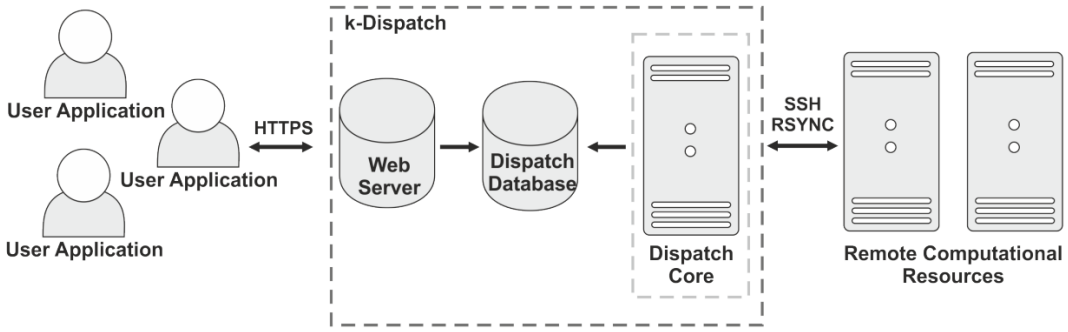
2.2 k-Dispatch ako riešenie

k-Dispatch je modulárny systém pre jednoduché používanie HPC. Hlavnou myšlienkou k-Dispatch je priblížiť používanie superpočítača k obyčajnému užívateľovi, bez toho aby musel študovať ako fungujú, ako správne plánovať beh výpočtu a ďalšie zložité veci z tejto oblasti. Vytvorením vhodného užívateľského rozhrania, vyzualizácií a simulácií by sa dosiahlo zvýšenie efektivity výpočtov, zníženie potrebných zdrojov, zníženie CPU času, zníženie pamäťových nárokov a iné. k-Dispatch obsahuje web server s rozhraním, ktoré sprístupňuje databázu, v ktorej sú uložené údaje napríklad o prebiehajúcich výpočtoch, užívateľoch, skupinách a podobne. Pomocou tohto webového rozhrania môžu komunikovať klientské aplikácie so systémom k-Dispatch. k-Dispatch poskytuje webové grafické rozhranie pre administrátorov systému pre jednoduchšiu správu[11].

Schéma

k-Dispatch sa skladá z viacerých častí, modulov, ktoré spolu vzájomne tvoria celok, jedná sa o web server, dispatch databázu a dispatch jadro. Jednotlivé časti systému môžu používať rôzne protokoly pre komunikáciu. Komunikácia systému s klientskými aplikáciami prebieha napríklad pomocou HTTPS z modulu web serveru. Plánovač komunikuje s HPC pomocou SSH alebo RSYNC protokolu z modulu dispatch jadra. Databáza môže byť súčasťou web serveru, alebo môže byť umiestnená mimo web serveru. Web server ponúka REST API

pre klientskú komunikáciu. Administrátor má k dispozícii webové rozhranie pre správu užívateľov, skupín užívateľov, HPC zdrojov a iných[11]. Schéma aplikácie je možné vidieť na obrázku 2.1.



Obr. 2.1: Schéma k-Dispatch aplikácie[11].

Databáza

Databáza obsahuje informácie o užívateľoch ku príkladu meno, emailovú adresu, ich rolu v rámci systému a iné. Role sú Dispatch Server Module (DSM) Admin, Hospital Admin, Planner, Reviewer. DSM Admin má úplnú kontrolu nad celým systémom, Hospital admin má nižšie oprávnenia ale stále dokáže meniť napríklad skupiny používateľov. Planner je užívateľ, ktorý používa výpočtové prostriedky pre svoju prácu a navrhuje a zadáva úlohy na spracovanie. Reviewer má len pozorovateľský charakter a nemôže v systéme nič meniť[11]. Licencie slúžia skupinám používateľom aby mohli uskutočňovať výpočty. Skupiny používateľov s licenciou nakupujú alokácie výpočtových zdrojov, aby mohli zadávať úlohy. Pre spustenie výpočtov je treba ukladať do databázy záznamy o predchádzajúcich spusteniach úloh, liečebných plánoch (TP), závislostiach úloh, konfiguráciách a ďalšie informácie potrebé pre spúšťanie a monitorivanie úloh[11].

Web server

Web server je súčasťou k-Dispatch aplikácie ako je možné vidieť na obrázku 2.1. Je naprogramovaný v skriptovacom jazyku Python, viac v časti 5.4. Využíva micro framework Flask[1], napríklad pre spracovanie prichádzajúcich požiadaviek, kedy po prečítaní cesty rozhodne, ktorá funkcia sa vykoná. Stará sa tiež aj o základnú autentizáciu užívateľov. Web server slúži primárne na sprístupnenie databázy aplikácie k-Dispatch pre klientské aplikácie, napríklad protokolom HTTPS, môže tiež slúžiť ako hostiteľ jednoduchej administratorskej konzoly. Aplikačné rozhranie je vytvárané podľa konceptu REST, Representational State Transfer Application interface, viac v časti 5.3. Sprístupňuje funkcionalitu ako modifikácia používateľa, pridanie nového výpočtového prostriedku, pridanie novej licencie. Rozdeľuje zdroje v databázy do niekoľkých skupín, každá má unikátnu časť cesty v URL, podľa ktorej sa dá rozpoznať ku ktorému zdroju sa viaže dotaz.

[1] http://flask.pocoo.org/

Webové rozhranie administrátora

Webové rozhranie bolo spomenuté ako sekundárne využitie web serveru v predchádzajúcej časti 2.2. k-Dispatch má 2 rôzne administrátorské oprávnenia, podľa zamerania. Rozhranie ale bude používať len Dispatch Server Module (DSM) administrátor.
DSM Admin - Administrátor s IT vzdelaním, ktorý spravuje systém. Má najvyššie oprávnenia Admin má možnosť pomocou tejto jednoduchej web stránky pristúpiť k serveru a spravovať ho, bez nutnosti akejkoľvek klientskej aplikácie. Web je vhodný spôsob vďaka rozšíreniu prehliadačov na všetky platformy a tým pádom je aplikácia spravovateľná odkiaľkoľvek.
Administrátor môže manipulovať s užívateľmi, modifikovať ich, meniť heslá, vytvárať nových a ďalšie, viď tabuľka 2.1. Všetky akcie sú vidieť na use-case diagramoch v prílohách práce.
Data, ktoré server poskytuje a operácie nad ním:

Názov	URL	Popis operácií
Používatelia	/users	zobraziť užívateľov, pridať/deaktivovať/aktivovať/upraviť používateľa, zobraziť plány liečby užívateľa, use-case diagram príloha A.1
Skupiny užívateľov	/groups	zobraziť skupiny, vytvoriť/deaktivovať/aktivovať/upraviť skupinu, presúvať členov skupín, zobraziť plány liečby podľa skupiny, use-case diagram príloha A.2
Alokácie zdrojov HPC	/allocations	zobraziť alokácie, vytvoriť/deaktivovať/aktivovať/upraviť alokáciu, use-case diagram príloha A.3
HPC	/hpc	zobraziť HPC, vytvoriť/deaktivovať/aktivovať/upraviť HPC, use-case diagram príloha A.4
Licencie	/licenses	zobraziť licencie, upraviť/vytvoriť, zobraziť licencie pre konkrétneho užívateľa, use-case diagram príloha A.5
Plány liečby	/tps	zobraziť plány, prerušiť plán, zobraziť simulovaný plán, use-case diagram príloha B.1

Tabuľka 2.1: Rozdelenie dát do skupín a ich URL.

Plány liečby sú konkrétna aplikácia systému zameraná na použitie v zdravotníctve, ale systém je možno jednoducho upraviť na použitie v iných aplikáciách vysoko výkonných výpočtov v iných odboroch[11].

Kapitola 3

Stav a ciele práce

3.1 Stav k-Dispatch

V dobe prevzatia projektu k-Dispatch pre potreby BP bola implementovaná väčšina rozhrania web serveru zahŕňajúc správu užívateľov, skupín užívateľov, výpočtových prostriedkov, licencií, liečebných plánov. Avšak, chýbala správa alokácií výpočtových prostriedkov a cesty pre administrátorské webové grafické rozhranie. Databáza už bola navrhnutá a bol vytvorený inštalačný skript, ktorý naplní databázu ukážkovými dátami. Web server ešte nemal web stránku pre administrátora, cez ktorú bude môcť kontrolovať užívateľov, skupiny užívateľov, plány liečby, aktívne a neaktívne clustery a ďalšie časti systému.

3.2 Ciele práce

Cieľom semestrálneho projektu bolo zoznámenie sa so skriptovacím jazykom Python, knižnicami Flask, Peewee a Unittest. Osvojiť si štábnu kultúru výskumnej skupiny SCFIT a pracovať podľa jej odporúčaní a pravidiel. Zamyslieť sa nad tým, ako budem integrovať potrebnú funkčnosť do k-Dispatch. Dokumentáciu k zdrojovým súborom budem generovať pomocou knižnice/nástroja Sphinx, používa ho aj Flask a Peewee, frameworky používané v tomto projekte.

Návrh

Doplnil som návrh aplikácie o tieto body:

- Webové rozhranie bolo navrhnuté v dodanej špecifikácii pri preberaní projektu. Ja som pridal cestu pre úvodnú webstránku administrátora a ďalšie potrebné podstránky.
- Navrhol som prípady užitia web stránky administrátora podľa ktorých sa budú postupne vyvíjať jej jednotlivé časti a funkcionalita.
- Súčasťou návrhu administrátorskej web stránky je aj jej dizajn a layout. Realizovaná bude pomocou dynamických stránok, ktoré budú po požiadavku na ne zobrazovať aktuálne dáta z databázy, detailnejšie o dynamických stránkach v 2.2 a o databáze v 2.2. Využívam responzívne prvky z frameworku Bootstrap[1] ako stavebné prvky pre

[1] https://getbootstrap.com/

layout ktorá je následne plnená obsahom stránky. Kostru stránky som vytvoril ako šablónu pre framework Jinja2[2], ktorý je súčasťou Flask[3].

- Navrhnúť testovacie prípady, ktoré overia funkčnosť implementácie. Najprv jednotkové testy na serverovej časti a potom automatizovane testy web stránky administrátora.

Implementácia

Implementácia navrhnutých bodov bude prebiehať takto:

- Implementácia jednotlivých diagramov užitia bude zväčša ako jedna webová stránka z ktorej sa budú odkazovať na podstránky, napríklad pri zozname používateľov bude odkaz na podstránku s detailom užívateľa. Odkaz bude smerovať do REST API serveru, ktoré odošle detail užívateľa a ten bude zobrazení užívateľovi.
- Každá takáto stránka alebo podstránka bude vychádzať z vytvorenej šablóny, ktorá sa na web servery naplní dátami a odošle ako odpoveď prehliadaču.
- Na klientskej strane bude využitý JavaScript napríklad pre kontrolu formulárov.
- Stránky budú dizajnované pomocou kaskádových štýlov, ktoré budú vychádzať z frameworku Bootstrap.

Testovanie

Priebeh testovania počas vývoja a na jeho konci:

- Testovanie administrátorskej web stránky bude prebiehať najprv podľa diagramov prípadov použitia, potom na základe vypracovaných testovacích scenárov.
- Testovanie bude prebiehať aj na úrovni jednotkových testov funkcií na strane serveru.
- Automatické testovanie front-end aplikácie bude prebiehať najprv ručne počas vývoja, následne sa vytvoria automatické testovacie scenáre, ktoré budú automaticky testovať funkčnosť.
- Pre testovanie REST API je vhodný nástroj JMeter[4], front-end bude testovaní nástrojom Selenium[5]

3.3 Súčasný stav

Časť RESTful rozhrania web serveru, ktorá zabezpečuje alokácie výpočtových prostriedkov som doimplementoval, ostáva len dokončiť jednotkové testy pre túto časť. Pridal som zatiaľ jednu cestu pre testovanie návrhu šablón administrátorského webového grafického rozhrania. Vytvoril som základnú šablónu webového grafického rozhrania, ktorú som otestoval pomocou pridanej cesty v rozhraní web serveru. Otestoval som použitie frameworku Bootstrap[6] pre potreby administrátorského rozhrania.

[2] http://jinja.pocoo.org/
[3] http://flask.pocoo.org/
[4] http://jmeter.apache.org/
[5] https://www.seleniumhq.org/
[6] https://getbootstrap.com/

Kapitola 4

Metodiky vývoja SW

Pri práci používam verzovací nástroj GitLab[1] poskytnutý skupinou SC@FIT, kde si vytváram úlohy a k ním vetvy pre prehľadnosť. Vývoj som začal študovaním už vytvorených častí aplikácie a to predovšetkým databázy a rozhrania web serveru. Pred začatím písania akéhokoľvek kódu som si prečítal SC@FIT Handbook, kde sú popísané pravidlá a o odporúčania, ktorými by sa mali práce v rámci skupiny SC@FIT držať. Funkcionalitu vhodne štruktúrujem a integrujem do celku k-Dispatch. Testy vytváram až po implementácii funkčnosti, prvotné testovanie ešte počas vývoju vykonávam pomocou nástroju Postman[2].

4.1 Vývoj SW v SC@FIT

Vývoj v skupine SC@FIT sa riadi príručkou SC-FIT-Handbook, ktorej autormi sú Jiří Jaroš, Bradley E. Treeby, Jakub Budisky a Filip Veverka. Slúži na definovanie štýlu písania udržovateľného a jednoducho rozšíriteľného kódu preovšetkým v C/C++.
Príručka obsahuje všeobecné pravidlá písania kódu ako je oddelovanie binárnych operátorov od operandov, veľkosť odsadenia bloku kódu, konvencie pre pomenovávanie premenných, tried, konštánt a ďalších. Špecifikuje používanie Doxygen komentárov, z ktorých sa následne generuje dokumentácia. Definuje ako zapisovať základné konštrukcie programovacích jazykov ako je podmienka, cyklus s pevným počtom opakovaní, s podmienkou na konci, ako zapisovať podmienky a výrazy v podmienkach. Špecifikácia obsahuje tiež triedy, metódy, funkcie a ich konvencie pre pomenovávanie, definovanie a deklaráciu, umiestnenie kódu. Hlavičkové súbory, ak ich jazyk obsahuje, by mali obsahovať napríklad mechanizmy proti opätovnému zahrnutiu definícií, anglicky *include guard*. Súbory v projekte by mali byť usporiadané do logickej štruktúry vyplývajúcej z daného programovacieho jazyka. Každý súbor by mal začínať presne danou hlavičkou spoločnou pre všetky súbory a obsahovať základné informácie o súbore, verzii, autoroch a iné[7].

Issue Tracking

Pri vývoji v skupine SC@FIT sa využíva GitLab Issue Tracker pre sledovanie issues, ako postupuje ich riešenie, kto akú zmenu vykonal. V Issue Tracker sa dajú triediť commits podľa míľnikov, issues, náveští.
GitLab workflow integruje issue napríklad tým, že ich môžeme kdekoľvek referencovať po-

[1] https://about.gitlab.com/
[2] https://www.getpostman.com/

mocou znaku „#“, toto je často použité k commit správe. Tieto správy môžu obsahovať text formátovaný jazykom Markdown, doporučuje sa vzhľadom na sprehľadnenie textu, pridanie linkov či obrázkov do textu pre lepšie pochopenie. Ku každému novému issue by sa mala založiť nová samostatná vetva, v ktorej budú vykonané zmeny a bude presne vidieť aké zmeny bolo treba spraviť pre splnenie, issue by mala byť pravidelne aktualizovaná vzhľadom na stav implementácie. Issue má svoje identifikačné číslo a každý commit musí obsahovať číslo issue, ku ktorej sa viaže. Po uzavretí práce na issue sa vetva spája späť do originálnej vetvy, väčšinou `master`, kde sa po vyriešení konfliktov prejavia zmeny.
Každá issue by mala obsahovať kľúčové slovo, podľa ktorého sa dá rozpoznať či ide o implementáciu novej funkčnosti alebo len opravu chyby. Popis issue je štruktúrovaný text, ktorý odpovedá na otázky, ako, prečo a za akým účelom vznikol tento text. Zodpovedný človek je osoba, ktorá je zodpovedá za implementáciu alebo opravu, ktorá je popísaná v issue. Dátum dokončenia určuje čas, do ktorého je treba stihnúť uzavrieť issue so všetkými náležitosťami. Míľnik slúži na organizovanie issues a commits do prehľadnej štruktúry, prehľad nových funkčostí v nadchádzajúcej vydávanej verzii.
Podrobný popis je súčasťou každej vytvorenej issue a slúži zodpovednej osobe pre zorientovanie sa, prípadne pre pozvaných kolegov. Rozsah issue nie je obmedzený a mal by obsahovať všetky potrebné informácie, napríklad ako navodiť spomínanú chybu[7].

Merge Requests

Požiadavky na spojenie vetiev by mali obsahovať dostatok informácií aby mohla pridelená osoba rozhodnúť a spojení alebo odmietnuť spojenie a vrátiť požiadavku späť. Nikdy by sa nemala do `master` vetvy pripájať iná vetva ako `development`, hlavne kvôli prehľadnosti. Do `development` vetvy by mali byť zlučované všetky ostatné vetvy týkajúce sa jednotlivých issue. Zodpovedná osoba za merge by mala byť rozdielna od osoby, ktorá issue vytvorila alebo riešila[7].

Test Driven Development

Hlavná myšlienka - najprv testy, potom implementácia spĺňajúca testy. Testami riadený vývoj z anglického *Test Driven Development* sa zameriava na krátke testovanie krátkych scenárov, môžu to byť aj len funkcie testované pomocou jednotkových testov. Typický postup pre tento prístup je, že programátor si vytvorí automatické testy v prostredí, v ktorom pracuje. Po prvom spustení všetky testy musia skončiť neúspechom až potom môže začať programátor písať kód, ktorý tieto testy splní[2].
Test Driven Development prichádza do programovacích/skriptovacích jazykov väčšinou ako framework napríklad Java - JUint, Python - Unittest[2].

Kapitola 5

Webová aplikácia a jej technológie

5.1 Webová aplikácia

Webovou aplikáciou sa rozumie aplikácia využívajúca architektúru klient-server, ktorá komunikuje medzi web serverom a klientom využívajúc väčšinou sieť Internet. Najčastejším klientom používaným na komunikáciu s web serverom je webový prehliadač. Tento klient je dostupný na väčšine operačných systémov už pri inštalácii a to je výhoda využitia web serveru pri komunikácii so službami na internete. Webové prehliadače podporujú rôzne technológie bez nutnosti ďalšej inštalácie, napríklad JavaScript. Toto je jeden z dôvodov rozšírenia web aplikácii.
Komunikácia cez počítačovú sieť internet v drvivej väčšine prebieha pomocou protokolu HTTP/HTTPS, ktoré sú aplikačné protokoly pracujúce s portom TCP/80. HTTPS ponúka zabezpečenie komunikácie pomocou TLS/SSL šifrovania obsahu správy[1].
Väčšina ľudí, lajkov pozná web server typicky poskytujúci web stránky s textom, obrázkami a ďalšími viditeľnými prvkami. Web stránky sa delia na 2 hlavné kategórie a to podľa toho kedy sú vytvárané.
Web stránky môžu byť vytvorené pri návrhu web serveru, nazývajú sa statické, nemenia za žiadnych okolností svoj obsah a sú uložené na servery v podobe súboru, ktorý sa odosiela ako odpoveď na požiadavku. Opakom sú dynamické stránky, ktoré sa až pri požiadavku na ne vytvoria. Príkladom môže byť naplnenie web stránky dátami z databázy, ktoré sa môžu lubovolne v čase meniť, preto sa nedá použiť statická stránka. Dynamická stránka je uložená ako návod ako ju vytvoriť keď príde dotaz, toto sa využíva ak chceme výslednú stranku rôzne parametrizovať[6].
Avšak web server môže slúžiť aj ako spôsob prístupu k vzdialeným službám, ktoré následne posielajú výsledky klientovi. Výsledky môžu byť v podobe surových dát, ktoré až klient interpretuje užívateľovi, alebo pošle už spracované dáta a klient ich len zobrazí užívateľovi. Formáty pre výmenu dát môžu byť XML, JSON, a podobné, to predovšetkým keď sa jedná o surové dáta. Naopak môže poslať aj obrázok, ktorý zobrazí webový prehliadač. Protokol HTTP, ktorý je používaný na komunikáciu s web servermi dokáže vďaka rozšíreniu MIME posielať a prijímať rovnako ako mail akékoľvek súbory klientom[13].

5.2 Front-end

Front-end sa rozumie tá časť aplikácie, ktorú vidí používateľ a s ktorou interaguje pri práci s aplikáciou. Front-end vizualizuje dáta pre užívateľa, ktoré sú získané z back-end časti

aplikácie. Front-end webovej aplikácie sa nachádza väčšinou v internetovom prehliadači, pretože je to najrozšírenejší klient.[4] Internetový prehliadač ponúka podporu mnohým technológiám ako je AJAX, React, AngularJS, CSS, rôzne frameworky ako je napríklad Bootstrap a samozrejme hlavný nástroj a to je HTML. O týchto technológiách sa bude písať v časti 5.4.

5.3 Back-end

Back-end je pre užívateľa často „neviditeľný“, pretože je umiestnený niekde na serveri, za nejakou IP a portom. Back-end je oddelený od Front-end, napríklad internetovou sieťou, cez ktorú back-end a front-end komunikujú a vymieňajú si dáta. Back-end využíva širokú škálu skriptovacích jazykov, kompilovaných jazykov, frameworkov, menej či viac komplexných. Back-end hojne využíva databázy ako mieto kde si ukladá dáta. Najčastejšie sa používajú relačné databázové systémy. Súčasťou back-end je implementácia rozhrania cez ktoré komunikuje back-end s front-end.

REST API

Skratka pre Representational state transfer application programming interface, použiteľný nie len v kontexte web serveru. REST je zbierka princípov ako sa definujú zdroje, identifikácia zdrojov, metódy prístupu ku zdrojom, operácie, ktoré sa môžu vykonávať so zdrojmi. REST sa najčastejšie používa v implementácii webových serverov, kde každý jeden resource aplikácie má svoju jednoznačnú URI, komunikujú protokolom HTTP. REST paradigma definuje operácie, ktoré sú zhrnuteľné v akronyme CRUD(create, read/retrieve, update,delete). Tieto operácie sa prirodzene dajú premietnúť do HTTP metód GET, POST, PUT a DELETE[9].
Popis jednotlivých písmen CRUD:

- Create - HTTP metóda POST, v tele metódy POST je popis novo vytváraného zdroju.
- Retrieve/read - HTTP metóda GET, vykonaním volania GET voči URI zdroju sa v tele odpovede vráti popis zdroju.
- Update - HTTP metóda PUT, telo tejto metódy obsahuje len položky, ktoré chceme v zdroji zmeniť, alebo pridať.
- Delete - HTTP metóda DELETE, väčšinou neobsahuje telo, v URL je špecifikovaný zdroj a jeho ID alebo iný identifikátor.

5.4 Technológie webových aplikácií

Python

Moderný dynamický interpretovaný skriptovací jazyk používaný v mnohých aplikáciách. Podporovaný na viacero platformách vďaka tomu, že používa interpret na vykonanie zdrojových skriptov. Podporuje viacero programovacích paradigiem, objektové, imperatívne, procedurálne a funkcionálne. Najnovšia verzia jazyka je 3.x. Má slabú typovú kontrolu, používa dynamické typovanie a typová kontrola prebieha za behu programu. Podporuje dedičnosť objektov a polymorfizmus. Podporuje použitie známych grafických rozhraní ako

napríklad PyQt, PyGTK a podobné, preto dokáže vytvárať plnohodnotné aplikácie s užívateľským rozhraním[15].
Python ponúka množstvo knižníc a užitočných frameworkrov. Frameworky slúžia na jednoduchší prístup k databázovým prostriedkom, tvorbe užívateľských rozhraní, tvorbe jednotkových testov implementácie elementárnej funkčnosti, návrhu serverov, autentizácii užívateľov a mnoho ďalších[15].
Príklad zdrojového kódu v skriptovacom jazyku Python

```
# komentar
import os
variable = 1
for letter in "example":
    print(letter)
print("Hello world")
print(variable)
```

Import knižníc prebieha príkazom `import`. Ako oddeľovače blokov kódu sa používa tabulátor alebo postupnosť medzier. Riadky nie sú ukončené žiadnym špeciálnym znakom, napríklad `;` v programovacom jazyku C/C++, okrem znaku konca riadku. Typ premennej `variable` je určený až po vykonaní príkazu priradenia[15].

Flask

Flask[1] je microframework pre jazyk Python, ktorý sa využíva pri programovaní web serverov. Zakladá si na frameworkoch Jinja2[2], odkiaľ si berie šablóny a frameworku Werkzeug[3]. Microframework znamená, že neobsahuje veľa funkcionality ale je jednoduché ju tam ľahko integrovať. Na rozdiel od Django2[4] necháva užívateľa vybrať si vlastnú knižnicu pre objektovo relačné mapovanie. Veľmi vhodný na implementáciu REST rozhrania web serveru. Každá URL sa dá pomocou anotácie namapovať na funkciu, ktorá sa po HTTP požiadavke zavolá. Typ HTTP požiadavky sa dá špecifikovať v anotácii a tak sa dajú oddeliť implementácie pre rôzne typy požiadaviek[3].
Príklad minimálnej aplikácie vo frameworku Flask:

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def hello_world():
    return 'Hello, World!'
```

Anotácia `@app.route('/')` označuje, že funkcia `hello_world` sa vykoná pri akejkoľvek požiadavke na URI „/“.
Alternatívy Flask frameworku:

- Django[5] - vysokoúrovňový web framework pre Python. Umožňuje rýchly vývoj vďaka veľa predpripraveným knižniciam na rozdiel od Flask. Rieši väčšinu problémov vznikajúcich pri vývoji webov, napríklad autentizáciu užívteľov. Jednoducho škálovateľný[5].

[1] http://flask.pocoo.org/
[2] http://jinja.pocoo.org/
[3] http://werkzeug.pocoo.org/
[4] https://www.djangoproject.com/
[5] https://www.djangoproject.com/

- Pyramid[6] - dobre zdokumentovaný microframework, jednoduchý začiatok, ponúka na výber množstvo voliteľných balíkov priamo na stránke frameworku, uchováva konfiguráciu mimo systému a tým sa vyhýba vedľajším efektom.

Šablóny stránky

Framework Jinja2, ktorý v sebe Flask integruje, ponúka šablóny pre vytváranie dynamických stránok podobne ako sa to robí v jazyku PHP.
Dynamické stránky sú také stránky, ktorých obsah sa vytvára až pri požiadavke užívateľa. Vhodné napríklad pri zobrazovaní dát z databázy.
Šablóny obsahujú okrem CSS,HTML, JS a iných web technológií aj špeciálne označené bloky. Tieto špeciálne označené bloky majú svoju syntax a sémantiku, ktorá je podobná, prirodzene, jazyku Python. Do blokov sa vkladá obsah premenných. Vnútri bloku je možné použiť napríklad for cyklus pre iteráciu skrz iterátor objektu alebo pole.
Ak už sme takúto šablónu vytvorili, dáta do šablóny dostaneme tak, že funkcii `render_template`, ktorá vykresluje šablónu, dáme ako voliteľné parametre premenné ktoré obsahujú požadované dáta. Voliteľný parameter pomenujeme rovnako ako v šablóne a Jinja2 zariadi aby sa dáta objavili v šablóne[3]. Príklad šablóny:

```
<html lang=cs>
  <head>
    {% block header %}  {% endblock %}
    <title>Admin console - {% block pageTitle %}  {% endblock %}</title>
  </head>
  <body>
    <h1><b>Admin console - {% block contentTtitle %}  {% endblock %}</b></h1>
    {% block content %}  {% endblock %}
  </body>
</html>
{% block script %}  {% endblock %}
```

Notácia `{% block <meno> %} {% endblock %}` označuje miesto, kde bude umiestnený obsah a dáva bloku unikátne meno.
Blok naplníme takto:

```
{% extends '<nazov_sablony>' %}

{% block pageTitle %}
  Index
{% endblock %}
```

Kľúčové slovo `extends` určuje ktorú šablónu budeme plniť. Následne medzi `block pageTitle` a `endblock` vložíme obsah bloku, ktrý bude vložení do rozširovanej šablóny.

Databáza

Databáza je jeden z najpoužívanejších spôsobov na ukladanie dát v serveroch. Databáz existuje viac druhov (relačné, objektové, ...), najčastejšie sa používajú relačné databázy.

[6] https://trypyramid.com/

Relačnú databázu môžeme chápať ako relačný model dát alebo ako konkrétnu implementáciu systému riadenia bázy dát v softwarovom produkte. Základným stavebným prvkom relačnej databázy je tabuľka, ktorá má jeden alebo viac stĺpcov. Tabuľka obsahuje okrem stĺpcov riadky, ktoré reprezentujú jeden záznam v tabuľke. Podstatnou vlastnosťou databáz je vytváranie vzťahov medzi tabuľkami, reláciami, pomocou cudzích kľúčov, ktoré odkazujú na záznamy v iných tabuľkách a na základe rovnosti týchto cudzích kľúčov sa dajú spájať záznamy. Napríklad záznam o zamestnancovi môže obsahovať cudzí kľúč do tabuľky obsahujúcej pracovné pozície. Tento vzťah môžeme využiť pri hľadaní pracovníkov na konkrétnej pozícii[10].
Dôležitým objektom, ktorý sa vyskytuje v databázach je *pohľad*(anglicky *view*). Databázové pohľady sa chovajú podobne ako tabuľka/y, môžeme sa na dáta, ktoré obsahujú dopytovať klauzulou `SELECT`. Pohľady obsahujú predom vytvorený predpis, ktorý špecifikuje aké dáta pohľad obsahuje. Výhodou je, že nezaberajú skoro žiadne miesto, pretože dáta, ktoré pohľad poskytuje sú fyzicky uložené v tabuľkách, ktoré figurujú v definícii pohľadu. Nevýhodou je, ak pohľad používa viacero agregačných funkcií, či spojení, prípadne je databáza rozsiahla, môže byť získanie dát časovo náročnejšie. Jedným z riešení je vytvorenie indexov nad určitým stĺpcom, alebo vytvorenie *materializovaného pohľadu*. Návrh rozloženia tabuliek môže vychádzať napríklad z Entity Relationship diagramu, diagramu tried a iných. Databáza môže ukladať okrem textových dát aj obrázky alebo veľké binárne objekty[10].

JavaScript

JavaScript je interpretovaný programovací jazyk, ktorý je vykonávaný interpretom, napríklad v prehliadači[13]. V technológiách webu sa používa ako na strane klienta tak na strane serveru, napríklad na interpretáciu odpovede serveru, validáciu odosieľaného formulára predtým ako je odoslaný. Javascript sa stal dôležitou súčasťou architektúry webov a preto bol zaradený do webových štandardov spoločnosti World Wide Web Consortium. JavaScript má mnoho frameworkov a knižníc pre uľahčenie vývoja[14] .
Zdrojový kód má príponu .js, ak je zahrnutý v HTML súbore, je uzavretý párovou značkou `<script></script>` s atribútom `type` obsahujúcim typ skriptu. Značka `script` môže mať naviac atribúty `src` určujúci zdroj externého skriptu[14].
Príklad zdrojového kódu v JavaScript v HTML súbore spustiteľnom v prehliadači

```
<html>
    <head>
    <title>Hello world!</title>
    </head>
    <body>
        <script type="text/javascript">
            let a = 1;
            let b = "a";
            let c = a + b;
            alert("Hello world!\n" + c);
    </script>
    </body>
</html>
```

Jednoduchý kód v JS vypíše v modálnom okne reťazec „Hello world!“ a za ním riadkom oddelený reťazec „1a“, ktorý vznikol operáciou sčítanie z premenných `a` a `b`, kde sa uplatnila

implicitná typová konverzia na reťazec. Dátové typy premenných boli známe až v dobe interpretácie, preto nebolo možné prísť v predstihu na túto podivnú, avšak povolenú, typovú konverziu. Svojou syntaxou zdrojového kódu sa JS podobné na jazyky C/C++.

HTML

HTML je zkratka pre Hypertext Markup Language. HTML je značkovací jazyk popisujúci ako sa majú data zobraziť na výstup, napríklad v internetovom prehliadači. HTML súbory sú najčastejšie prenášané cez internet protokolom HTTP ale je možné ich prenášať akýmkoľvek iným protokolom, ktorý dokáže preniesť súbory, napríklad FTP. Najnovšia verzia je HTML5. Obsah, ktorý má byť príslušne naformátovaný sa musí nachádzať v príslušnej značke(tagu) z ktorej je pri zobrazovaní prečítaní a umiestnení na presne definované miesto. Tagy môžu mať atribúty, podobne ako v XML, ktoré dodatočne definujú vlastnosti tagu alebo obsahu, môžu definovať farbu, zarovnanie, zmenu riadkového na blokový element a opačne. Značky môžu byť párové, `<html></html>` ale aj nepárové, napríklad odriadkovanie `<br/>`[14].
Druhy HTML značiek:

- Štrukturálne - Formátujú dokument, rozdeľujú na bloky, napríklad `<h1></h1>`
- Sémantické - Značka popisuje aký má význam jej obsah, napríklad `<title></title>` udáva, že jeho obsah má význam nadpisu
- Štylistické - Dodávajú napríklad písmu font, farbu, hrúbku. Značka `<b></b>` naformátuje text vnútri na hrubo zvýraznený. Tieto značky majú najväčší a priamy vplyv na to, ako web stránka vyzerá v prehliadači.

HTML dokument má presne danú štruktúru súboru, ktorá musí byť dodržaná aby sa obsah zobrazil správne.
Minimálny HTML5 dokument, ktorý obsahuje len nadpis prvej úrovne `<h1></h1>`

```
<!DOCTYPE html>
<html>
 <head>
 </head>
 <body>
  <h1>Hello World!</h1>
  <!-- komentár v HTML -->
 </body>
</html>
```

CSS

Kaskádové štýly alebo CSS je rozšírenie HTML,XML a XHTML. Konceptom CSS je oddeliť obsah a štruktúru dokumentu od jeho vizuálnej stránky. Oddelenie prezentačnej časti od obsahovej umožnilo znovu použiteľnosť týchto štýlov na iných stránkach alebo podstránkach. CSS je možné použiť viackrát na jednej stránke, napríklad oddeliť režimy zobrazenia, obrazovka PC, tlačiareň, prezentácia a iné. Znovupoužiteľnosť CSS pravidiel umožnila vznik frameworkov, ktoré už obsahujú vytvorené pravidlá, napríklad Blueprint[7], Boots-

[7] https://blueprintcss.io/

trap[8], Foundation[9] a iné.
Import CSS definícií je možný viacerými spôsobmi, môže to byť vpísanie pravidla priamo do elementu, čím sa toto pravidlo použije len pre tento element. Môžeme pravidlo vložiť do párového HTML tagu `<style></style>`, elemtom
`<link rel='stylesheet' href='style.css' type='text/css'>` v hlavičke HTML dokumentu. Odkaz na CSS môže smerovať aj do siete internet, na server kde je CSS súbor priamo vložený, takto sa napríklad distribuuje framework Bootstrap. Pripojenie je tiež možné HTTP hlavičkou `Link: <style.css> rel:stylesheet`[12]. Príklad CSS pravidla v HTML dokumente:

```
<style type="text/css">
h1 {
  color: red;
}
</style>
```

Vyššie uvedené pravidlo nastaví farbu všetkých nadpisov na červenú.

[8] https://getbootstrap.com/
[9] https://foundation.zurb.com/

Kapitola 6

Implementácia

Implementácia pozostáva so zmien v už existujúcej databáze, ktorá je popísaná v časti 2.2, úpravy databázových pohľadov a niektorých tabuliek popíšem ďalej. Vytvorení front-end Flask šablón pre jednotlivé stránky webového rozhrania, ktoré sa skladajú z HTML, CSS, JavaScriptu a funkčných prvkov pre Flask template modul(premenné, cykly ...)[1]. Pridanie back-end ciest pre prihlásenie sa do admin konzoly

6.1 Databáza

Vo svojej práci som používal už vytvorenú relačnú databázu. Databáza využíva open-source implementáciu PostgreSQL[2], ktorá podporuje štandard SQL. Databáza sa inštaluje pomocou inštalačného skriptu `instalDB.py`, tento skript zároveň vytvorí ukážkové dáta.

Zmeny oproti pôvodnej databáze

V databáze som vykonal zmeny v tabuľkách `usergroupstate`, `hpcstate` a `allocationstate`. Tieto zmeny je možné vidieť na obrázku REFOBRAZOK, kde sú zvýraznené červenou farbou.
Pridal som nový stĺpec s názvom `color`, ktorý je typu `text`. Tento stĺpec reprezentuje akú farbu, farba je zapísaná vo formáte RGB[3], bude mať daný stav v grafoch vo front-ende. Záznamy v týchto tabuľkách reprezentujú stav určitej entity v systéme (užívateľov, skupiny užívateľov ...). Pri prvom návrhu implementácie zobrazenia dát v grafoch som priamo do definície grafov vo front-ende vkladal názvy a farby stavov, ktoré som zobrazoval. Avšak tento prístup je veľmi neflexibilný, čo i len zmena názvu stavu v tabuľke kde sú uložené, by spôsobila nekonzistenciu medzi názvom stavu v databáze a názvom v grafe zobrazenom na web stránke. Tento strnulý prístup som nahradil získavaním farieb a názvov stavov priamo z databázy. Tým pádom ak príde ku zmene názvu stavu v databáze, zmena sa prejaví pri ďalšom načítaní web stránky s grafom.
Upravil som databázový pohľad `treat_plan_view`, ktorý poskytuje informácie o treatment plane spolu s identifikátorom alokácie, používateľoch, ktorý ho spravujú, HPC na ktorom je vyhodnocovaný. V tomto pohľade mi chýbala informácia v ktorý deň v týždni bol naposledy aktualizovaný. Preto som do neho pridal stĺpec s názvom *tp_dow*, ktorý reprezentuje, ktorý deň v týždni bol naposledy aktualizovaný treatment plan. Pre naplnenie tohto stĺpca som

[1] http://flask.pocoo.org/docs/1.0/templating/
[2] https://www.postgresql.org/
[3] https://en.wikipedia.org/wiki/RGB_color_model

použil PostgreSQL[4] funkciu `date_part`[5], ktorá má dva parametre, prvý určuje, aký údaj chceme extrahovať z dátumu, ja som použil „dow“ ,ktorý vyberie z časovej známky deň v týždni. Druhý parameter zmienenej funkcie je časová známka, ktorá sa má spracovať. Tento nový stĺpec som použil aby som vedel zaradiť výsledky treatment planov do grafov podľa dní v týždni a podla aktuálneho stavu.

Nové databázové objekty

Vytvoril som si nový databázový pohľad `hpc_jobs_view`, ktorý spája tabuľky `allocation`, `treatmentplan` a `job` za účelom získanie všetkých jobov(job = najmenšia nedeliteľná časť treatment planu, ktorá sa počíta na superpočítači) práve spracovávaných na konkrétnom HPC.Pohľad obsahuje stĺpec s dňom v týždni, v ktorom bola posledná zmena stavu jobu, hodinu dňa, v ktorej bola posledná zmena stavu, identifikátor statusu jobu, HPC a treatment planu, identifikátor HPC, na ktorom je job počítaný, stav HPC spojeného s jobom, časovú známku zmeny treatment planu a jobu . Tento databázový pohľad využívam, napríklad keď potrebujem získať počty jobov v jednotlivý deň na konkrétnom HPC, získané dáta sú následne vykreslené do grafu.

6.2 Back-end

Pre svoju prácu som potreboval do implementovať v back-ende rozhranie pre alokácie, rozhranie pre prístup do admin konzoly, hlavne prihlasovaciu a úvodnú stránku. Ďalej som upravil časť existujúcich ciest a pridal som možnosť aby vracali HTML stránku, ktorá je súčasťou konzoly.

REST rozhranie `allocation`

Toto rozhranie poskytuje informácie o alokovaných výpočtových zdrojoch. Väčšina rozhrania podlieha oprávneniam DSM Admina.
Slúži na vytvorenie novej alokácie alebo modifikáciu už existujúcej. Modifikáciou sa rozumie zmena určitých údajov o alokácii, napríklad zmena dátumu expirácie ale aj zmena stavu alokácie. Rozhranie obsahuje špeciálnu cestu pre deaktiváciu alokácie. Pri deaktivácii alokácie sa pošle HTTP požiadavka typu `DELETE`.
Pre kontrolu prístupových práv používam objekt triedy `UserView` definovanej v súbore `database.py`, je to podtrieda triedy `BaseModel`. Trieda `BaseModel` je podtrieda triedy `Model` z modulu PeeWee[6], k zdedením metódam a atribútom pridáva triedu `Meta` a v nej atribút `database`, ktorý obsahuje databázové pripojenie. Tento objekt reprezentuje DB pohľad `user_view`. Prínos tohto pohľadu oproti tabuľke `user` je, že spája

Úprava stávajúcich REST ciest

Aby som sa vyhol pridávaniu ďalších ciest do špecifikácie systému, rozhodol som sa použiť už definované cesty. Pre rozoznanie volania kedy chcem od koncového bodu serializované dáta(napríklad JSON[7],...) a kedy HTML stránku pre administrátorskú konzolu, rozhodol

[4] https://www.postgresql.org/
[5] http://www.postgresqltutorial.com/postgresqldate_part/
[6] http://docs.peeweeorm.com/
[7] https://www.json.org/

som sa pre pridanie url query parametra s názvom `page`. Tento parameter má len jednu hodnotu a to je text `true`. Volaný koncový bod rozlíši volanie a odošle HTML stránku vygenerovanú zo šablóny. Ak má parameter inú hodnotu ako som spomenul tak ignoruje tento parameter a vracia serializované dáta. Dáta závisia od zvoleného koncového bodu. Napríklad ak chceme stránku so zoznamom alokácií, tak pošleme HTTPS dotaz typu GET na koncový bod `/allocations?page=true`. Odpoveďou na takto zaslaný dotaz bude HTML stránka obsahujúca tabuľku so všetkými alokáciami. Podobne to funguje aj s používateľmi, skupinami používateľov a HPC. Ak chceme detail používateľa, skupiny, alokácie alebo HPC odošleme HTTPS GET požiadavku na príslušný koncový bod, ktorý vracia len jeden zdroj a url query parametrom `page=true`.

Šablóny modulu Jinja 2

Systém používa framework Flask, ktorý obsahuje modul Jinja 2 pre podporu vytvárania šablón HTML stránok, tým pádom umožňuje dynamicky generovať webové stránky podobne ako napríklad skriptovací jazyk PHP[8].
Pre svoje riešenie som využil niekoľko šablón, ktoré reprezentujú jednotlivé pod stránky konzoly a jednu šablónu som si vytvoril ako predlohu, ktorá nemá žiadny obsah a len definuje obsah menu a pätičku stránky.
Zoznam použitých šablón:

- admin_template.html
- admin.html
- allocation.html
- allocations.html
- group.html
- groups.html
- hpc.html
- hpcs.html
- login.html
- newuser.html
- tp.html
- tps.html
- user.html
- users.html

Hlavná šablóna `admin_template.html` obsahuje už spomenuté menu spoločné pre všetky pod stránky, pätičku a bloky pre vkladanie obsahu zo šablón, ktoré využívajú túto predlohu. Tým pádom sa vyhneme duplicite HTML kódu a zaistíme konzistenciu menu naprieč

[8] https://php.net/

pod stránkami.
Schému blokov a ich umiestnenie v rámci HTML schémy stránky je možné vidieť v časti 5.4. Táto šablóna obsahuje blok s názvom `header`, ktorý má umožniť vkladať do HTML elementu `head` dodatočné informácie ktorejkoľvek šablóne, ktorá využíva túto hlavnú šablónu.
Blok s názvom `title` má jediný význam a to, aby si každá šablóna vložila svoj vlastný nadpis do predlohy web stránky z hlavnej šablóny.
Ďalší blok je `content`, ako už napovedá názov, najpodstatnejší blok šablóny, do ktorého sa bude vkladať obsah jednotlivých pod stránok.
Posledný blok je `script`, do ktorého sú vkladané webové skripty, napríklad JavaScript, ktorý vytvára grafy alebo poskytuje vyhľadávanie a radenie obsahu tabuliek.

Získavanie dát pre grafy a tabuľky

V upravených REST API cestách, ako boli popísané v časti 6.2, som implementoval dotazovanie sa na dáta v databáze. Toto dotazovanie sa vykoná až pri rozpoznaní správne nastaveného query parametra `page`, aby sa nespomalila činnosť koncových bodov oproti stavu na začiatku práce.
Pre získavanie dát používam ORM(mapovanie objektov na relačnú schému databázy a vice versa) modul PeeWee[9], ktorý mapuje tabuľky relačnej databázy na objekty v jazyku Python. Dva základné objekty, ktoré z tohoto modulu sú Table a View.

6.3 Front-end

Front-end pozostáva z niekoľkých web stránok. Web stránky pozostávajú z

Prihlasovacia a úvodná stránka

Prihlasovanie do systému som riešil pomocou modulu Flask-Login, ktorý je súčasťou frameworku Flask.
Vstupný bod do systému tvorí prihlasovacia stránka kde je nadpis a tlačidlo na prihlásenie do systému. Tlačidlo vyvolá modálne okno kde je užívateľ požiadaný o zadanie prihlasovacích údajov. Prihlasovacie údaje musia byť správne a užívateľ musí mať aj dostatočnú úroveň oprávnení.
Po prihlásení je užívateľ poslaný na úvodnú stránku ktorá má názov „Admin console - Index“. Na tejto stránke sa nachádzajú grafy informujúce administrátora o stave systému. Na stránke sú 4 grafy a menu pre pohyb na ďalšie pod stránky.

Layout stránky

Layout HTML stránky ktorá je grafickým výstupom systému som riešil pomocou CSS frameworku Bootstrap 3[10]. Framework poskytuje tzv. *grid systém*. Skladá sa z viacerých CSS tried, ktoré plnia rôzne úlohy. Celý tento systém je obalený triedou `container`, táto trieda obaľuje celý systém riadkov a stĺpcov. Riadky sú definované v triede `row`, stĺpce sú definované v triede `col-*`.
Užitočným prvkom z tohoto frameworku sú záložky, do ktorých je možné organizovať obsah.

[9] http://docs.peeweeorm.com
[10] https://getbootstrap.com/docs/3.3/

Použil som ich napríklad na pod stránke detailu užívateľa, kde na aktívnej záložke sú detaily používateľa v tabuľke a na ďalšej záložke sú zobrazené grafy.
Menu je tiež postavené na prvkoch CSS frameworku Bootstrap 3. Využíva štýly tlačidiel, ktoré poskytuje. Jedna z položiek menu je typu tzv. *dropdown*, kedy sa po rozkliknutí vyroluje detailnejšie menu, ktoré má dva stĺpce a sú v ňom rozdelené tlačidlá do skupín.

Grafy, získavanie dát z back-endu

Hlavným vyjadrovaním prostriedkom web stránok sú grafy. Grafy vyjadrujú napríklad koľko úloh sa práve počíta na danom super počítači. Používam niekoľko rôznych typov grafov, podľa toho čo vyjadrujú. Napríklad pre štatistiku v ktorý deň v týždni(pondelok, utorok, ...) sa uskutočnilo koľko jobov a v akom boli stave som použil stĺpcoví graf. Jeden stĺpec vyjadruje jeden deň v týždni a je pomerne rozdelený podľa počtu jednotlivých jobov. Ďalej na znázornenie ako sa v čase míňali hodiny priradené do alokácie som použil líniový graf, kde je na ose X dátum kedy nastal odpočet pridelených hodín. Na ose Y je potom počet hodín.
Pre vykresľovanie používam JavaScript knižnicu Chart.js[11]. Táto knižnica používa HTML element canvas do ktorého je vykreslený interaktívny graf. Dáta pre tieto grafy čerpám z back-endu. Pri každej požiadavke na HTML stránku je vyhodnotených niekoľko databázových dotazov, ktoré získajú z databázy potrebné údaje, napríklad počty jobov v určitom stave. Po získaní všetkých potrebných údajov ich usporiadajú väčšinou do poľa, ktoré je predané šablónovaciemu modulu Jinja2. Údaje sú predávané ako keyword argumenty funkcii, ktorá má na starosti vykreslenie šablóny. Tá následne spojí predané premenné a ich mená s menami premenných v šablóne. Tieto premenné sú v špeciálnych oddeľovačoch `{{ <názov premennej> }}`. Takto zapísaná premenná je následne vypísaná na výstup.

Graf treatment planu

Každý treatment planu sa skladá z určitého počtu jobov. Tieto joby na sebe závisia a sú vykonávané v predom určenom poradí. Na podstránke s detailami o treatment plane je vložený graf, ktorý vyjadruje tieto závislosti. Graf sa skladá z uzlov, ktoré reprezentujú jednotlivé joby a hrán, ktoré znázorňujú závislosť. Jednotlivé uzly grafu majú rôzne farby podľa toho, v akom stave sa nachádzajú. Ďalej obsahujú text v tvare `ID: <jobID>` vyjadrujúci ID jobu.
Pre vizualizáciu týchto závislostí som si vybral open-source JavaScript knižnicu vis.js[12]. Do elementu `canvas` je vykreslený celý graf, ktorý je interaktívny napríklad zobrazuje popisky uzlov.

Tabuľky a vyhľadávanie

Tabuľky používam na pod stránkach, kde napríklad zobrazujem zoznam všetkých používateľov, ktorý sa nachádzajú v systéme. Aby táto tabuľka nebola príliš široká, tak vyberám len niektoré údaje o používateľovi. Podobné tabuľky som vytvoril aj pre stránky so zoznamom super počítačov, alokácií, užívateľských skupín, jobov a treatment planov.
Pre lepšie používanie tabuliek som použil jQuery plug-in DataTables[13]. Plug-in poskytuje pokročilú interakciu s HTML tabuľkami jednoduchým nastavením. Obsahuje veľa rôznych

[11] https://www.chartjs.org/
[12] http://visjs.org/
[13] https://datatables.net/

nastavení. V základom nastavení ponúka radenie možnosť zoradiť tabuľku podľa obsahu v jednotlivých stĺpoch. Základné nastavenie tiež poskytuje jednoduché vyhľadávanie, ktoré hľadá zhodu vo všetkých položkách riadku. Ak nepríde ku zhode, riadok tabuľky sa schová. Pri pokročilejších nastaveniach je možné priradiť vyhľadávanie pre konkrétny stĺpec. Toto vyhľadávanie môže byť buď to rolovacie menu s všetkými unikátnymi hodnotami v stĺpci, alebo užívateľ zadáva reťazec ktorý sa porovnáva s každou položkou stĺpca. Pri každom uvoľnení klávesy a zdaní znaku sa prejdú všetky zobrazené riadky a skryjú sa tie, ktoré sa už nezhodujú so zadaným reťazcom.

6.4 Testovanie

Testovanie prebiehalo na úrovni zdrojových textov pomocou unit testov funkcií. Následne som testoval web rozhranie počas jeho programovania na rôzne prípady užitia.

Jednotkové testy

Jednotkové testy som použil vo funkciách, ktoré spadajú pod rozhranie alokácií. Pri vytváraní jednotkových testov som použil python framework `unittest`. Tento frameworku je založený na objektoch. Hlavnou triedou je TestCase, ktorá reprezentuje testovací prípad. Pri vytváraní testovacieho prípadu sa vytvorí nová trieda, ktorej predchodcom je práve trieda TestCase. Jednotlivé testy sa píšu od metód tejto triedy.
Pre testovanie rozhrania alokácií som vytvoril triedu `AllocationsTestCase`. Táto trieda obsahuje metódu `setUp` a `tearDown`, ktoré sa starajú o pripravenie všetkého potrebného pre spustenie testovania a následné upratanie systému po teste.
Ostatné metódy v tejto triede zodpovedajú každá za jeden koncový bod rozhrania alokácií.

Testovanie programátorom

Počas vývoja back-endu som testoval jeho chovanie pomocou aplikácie Postman[14]. Cez túto aplikáciu som posielal HTTPS requesty na práve vyvíjaní koncový bod. V aplikácii sa dajú vytvárať requesty a ukladať už vykonané requesty. Tiež som si vytvoril sadu uložených requestov pre jednoduché testovanie mnou implementovaných koncových bodov. Front-end časť aplikácie som z časových dôvodov stihol otestovať len počas programovania jednotlivých častí/pod stránok. Po dokončení určitej časti som vždy skontroloval či je stále všetko v poriadku a že som zmenou alebo pridaním nepokazil inú časť webu.

[14] https://www.getpostman.com/

Kapitola 7

Záver

Cieľom mojej práce bolo navrhnúť webové rozhranie pre správcov aplikácie k-Dispatch, ktoré im umožní jednoduchú kontrolu a správu systému. Pred začatím implementácie webového rozhrania som sa zoznámil s už existujúcim serverom a použitými technológiami. Preštudoval som možnosti akými sa môže vývoj uberať. Následne som si vytvoril diagram prípadov užití pre jednotlivé skupiny koncových bodov. K tomuto som využil špecifikáciu ktorá je súčasťou softwarového balíka. Po vytvorení diagramov som sa pustil do vytvárania šablón pre jednotlivé pod stránky. Rozvrhol som si čo je treba aby bolo dostupné na jednotlivých web stránkach. Po preskúmaní možností vývoja webových aplikácií som dospel k záveru, že najvhodnejším front-end riešením bude kombinácia HTML, CSS a JavaScriptu. Je to asi najviac podporovaná kombinácia technológií naprieč rôznymi platformami.

Literatúra

[1] Andersson, M.: *Introduction to web server traffic modeling and control research* . 2005, [Online; navštívené 23.01.2019].
URL http://portal.research.lu.se/ws/files/6091809/625294.pdf

[2] Ashbacher, C.: *Test Driven Development: By Example*. Packt Publishing Ltd., prvé vydanie, 2003, ISBN 9780321146533.

[3] Daniel Gaspar, J. S.: *Mastering Flask Web Development*. Packt Publishing Ltd., druhé vydanie, 2018, ISBN 978-1-78899-540-5.

[4] Dostalová, Z.: *Frontend vs. Backend*. Czechitas z.s., 2014, [Online; navštívené 16.01.2019].
URL https://www.czechitas.cz/cs/blog/zaciname-s-it/frontend-vs-backend

[5] Foundation, D. S.: *Django overview*. 2019, [Online; navštívené 17.01.2019].
URL https://www.djangoproject.com/start/overview/

[6] Gralla, P.: *Creating web pages : all in one*. Indianapolis: Sams, 2005, ISBN 0-672-32690-6.

[7] Jaros, J.; Treeby, B. E.; Budisky, J.; aj.: *SC@FIT Handbook*. VUT FIT Brno, 2018.

[8] Jaros, M.; Jaros, J.: *Framework for Planning, Executing and Monitoring Cooperating Computations*. Ostrava, 2017, [Online; navštívené 13.01.2019].
URL http://www.fit.vutbr.cz/research/pubs/pres.php?file=%2Fpub%2F11782%2FIT4I-poster.pdf&id=11782

[9] Leonard Richardson, S. R., Mike Amundsen: *RESTful Web APIs: Services for a Changing World*. O'Reilly Media, Inc., 2013, ISBN 978-1449358068.

[10] Malinowski, E.; Zimányi, E.: *Introduction to Databases and Data Warehouses* . Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, ISBN 9783540744047.

[11] Marta Jaroš and Jiři Jaroš: Scientific Workflows Management. In *PAD 2018*, Západočeská univerzita v Plzni, 2018, ISBN 978-80-7302-151-1.

[12] McFarland, D. S.: *CSS*. Sebastopol: O'Reilly, Štvrté vydanie, 2015, ISBN 978-1-491-91805-0.

[13] Mcpeak, J.: *Beginning JavaScript*. John Wiley & Sons, Incorporated, 2015, ISBN 9781118903339.

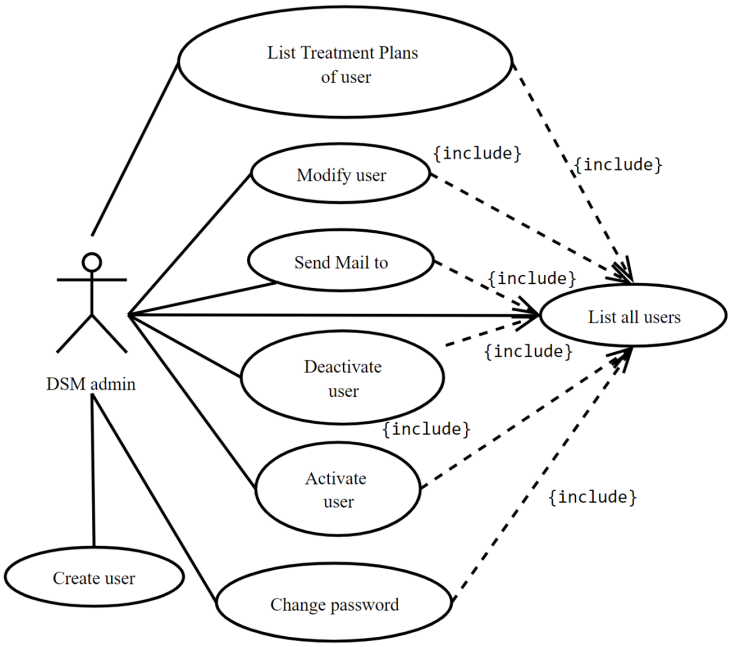
[14] Munzert, S.; Rubba, C.; Meißner, P.; aj.: *Automated Data Collection with R* . Chichester, UK: John Wiley & Sons, Ltd, 2014, ISBN 9781118834817.

[15] Summerfilend Mark, L. K.: *Python 3: výukový kurz*. Computer Press, 2010, ISBN 978-80-251-2737-7.

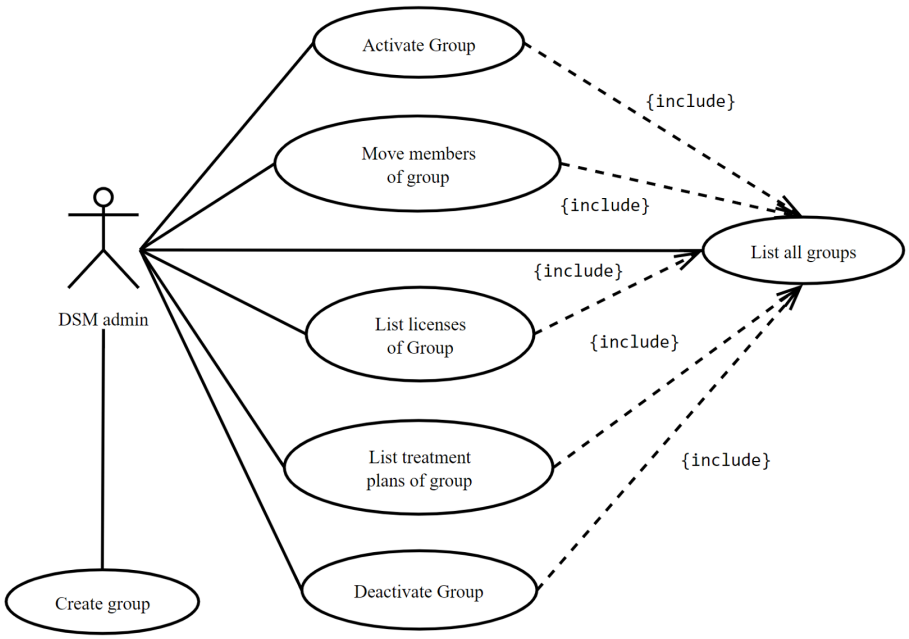
0

Príloha A

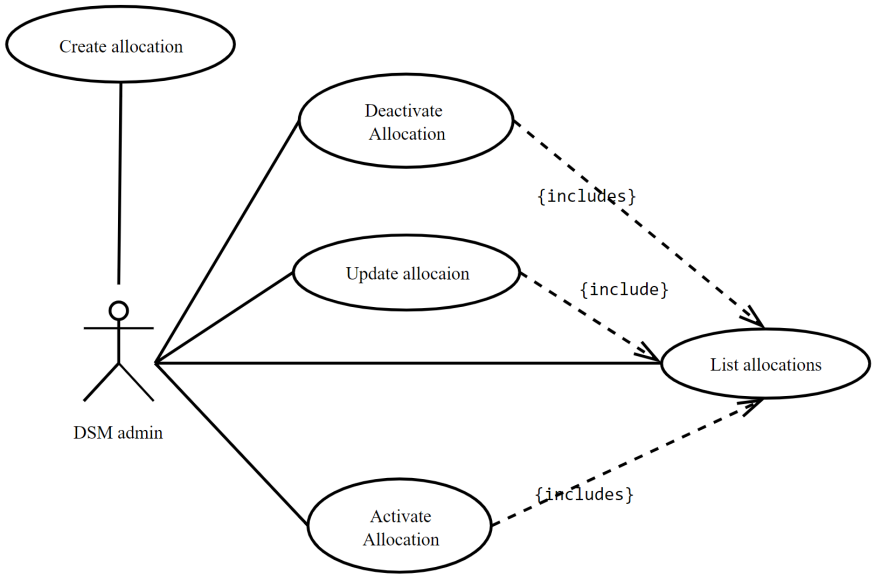
Use-case diagramy webového rozhrania



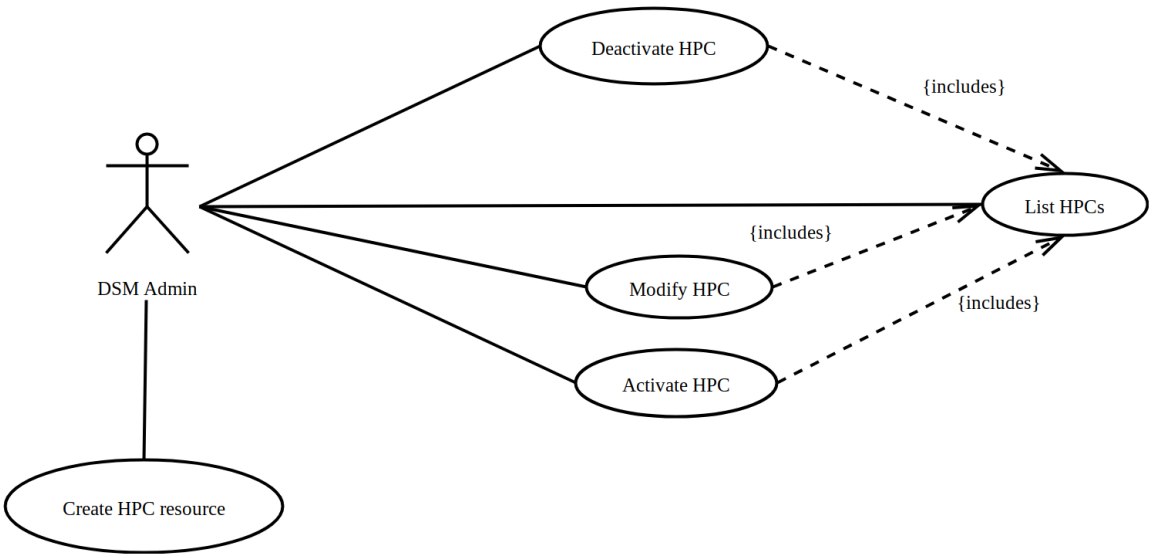
Obr. A.1: Use-case diagram správi užívateľov. Hlavným prípadom užitia je zobrazenie zoznamu používateľov, odkiaľ si administrátor vyberie, čo bude s užívateľom robiť. Vytvorenie užívateľa nezávisí na zozname užívateľov.



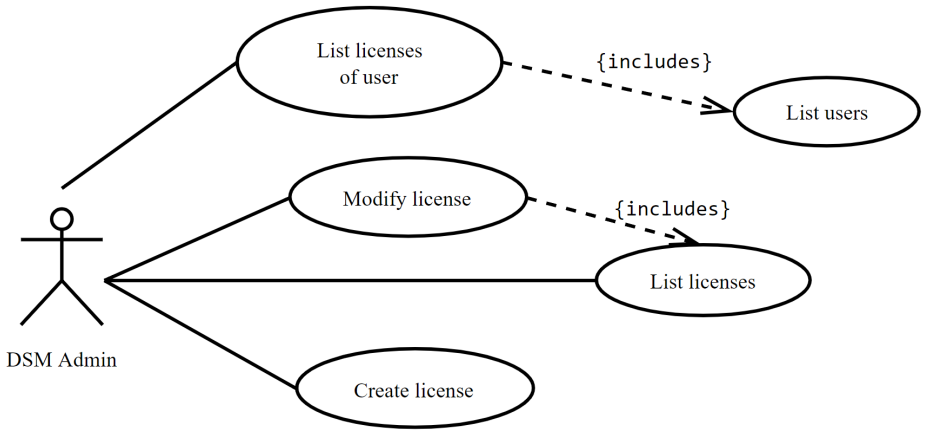
Obr. A.2: Use-case diagram správi skupín užívateľov. Rôzne akcie súvisia so zoznamom skupín užívateľov, z ktorých si administrátor vyberie skupinu a vykoná zmeny alebo zobrazí informácie. Vytvorenie skupiny užívateľov je nezávislý prípad užitia.



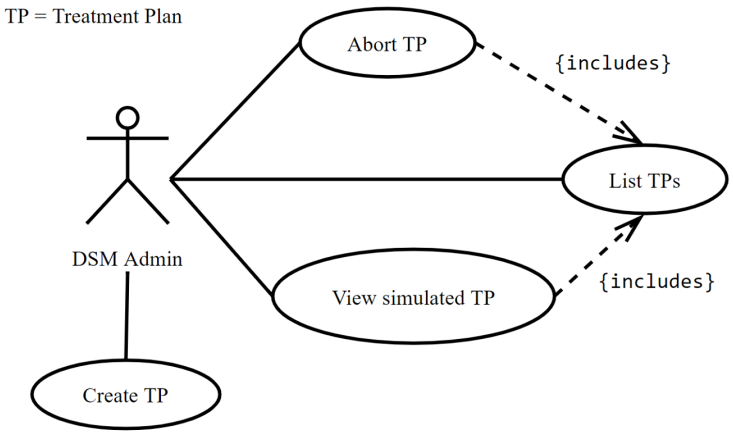
Obr. A.3: Use-case diagram správi alokácií. Zo zoznamu alokácii si vyberie administrátor alokáciu a môže ju zmeniť, deaktivovať alebo aktivovať. Vytvorenie alokácie je spoločný prípad užitia pre administrátora, Hospital admina ale aj plánovača.



Obr. A.4: Use-case diagram správi výpočtových prostriedkov (HPC). Podobné prípady užitia ako v prípade alokácií, modifikácia, aktivácia a deaktivácia výpočtového prostriedku. Vytvorenie výpočtového prostriedku je ale dostupné len pre Dispatch Server Module administrátora.



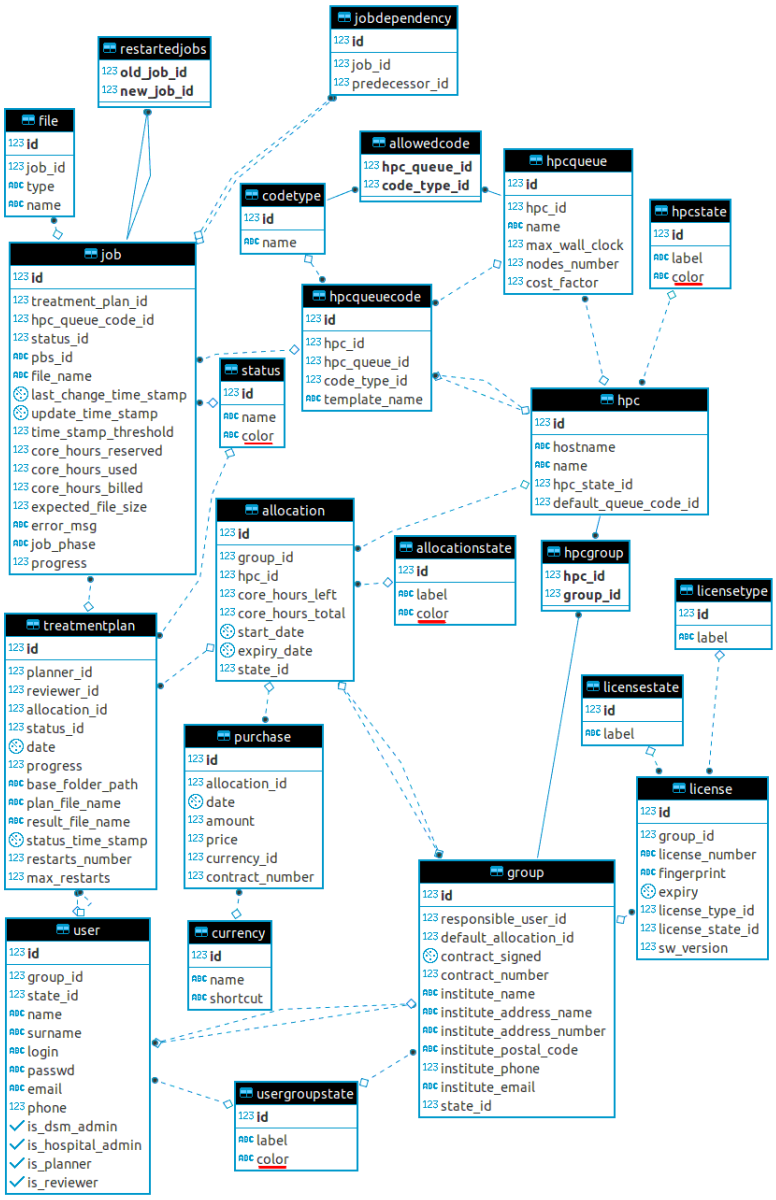
Obr. A.5: Use-case diagram správi licencií. Administrátor môže vytvoriť licenciu, tú následne modifikovať, licenciu si vyberie zo zoznamu licencií. Taktiež si môže zobraziť licenciu konkrétneho používateľa.



Obr. A.6: Use-case diagram správi liečebných plánov. Vytvorení liečebný plán (TP) môž prerušiť alebo si zobraziť jeho simulovaný priebeh.

Príloha B

Zmeny v databáze



Obr. B.1: Schéma aktuálneho stavu databázy po mnou vykonaných zmenách, ktoré sú podčiarknuté červenou farbou

Príloha C

REST API allocations

Metóda	Cesta	Popis	Vstup Výstup
GET	/allocations	Vráti zoznam všetkých alokácií	Vstup: - Výstup: Pole alokácií
POST	/allocations	Vytvorí novú alokáciu výpočtových prostriedkov	Vstup: alokácia(bez allocID) Výstup: alokácia
GET	/allocations/{allocID}	Vráti alokáciu s id=allocID. Požiadavka zlyhá ak používateľove {groupID} nie je spojené s žiadnou licenciou s {alocID}.	Vstup: - Výstup: alokácia
PUT	/allocations/{allocID}	Aktualizuje alokáciu s id=allocID	Vstup: alokácia Výstup: alokácia
DELETE	/allocations/{allocID}	Zmaže alokáciu s id=allocID	Vstup: alokácia Výstup: alokácia
GET	/allocations/{allocID}/hpc	Vráti HPC spojené s alokáciou s id=allocID. Požiadavka zlyhá ak používateľove {groupID} nie je spojené so žiadnou licenciou {allocID}	Vstup: - Výstup: hpc