VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

WEBOVÉ ROZHRANÍ PRO SPRÁVU A MONITOROVÁNÍ ÚLOH NA SUPERPOČÍTAČÍCH

WEB INTERFACE FOR TASK MANAGEMENT AND MONITORING ON SUPERCOMPUTERS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **PETR DANČÁK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MARTA JAROŠ**
SUPERVISOR

BRNO 2021

Ústav počítačových systémů (UPSY) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Dančák Petr**
Program: Informační technologie
Název: **Webové rozhraní pro správu a monitorování úloh na superpočítačích**
Web Interface for Task Management and Monitoring on Supercomputers
Kategorie: Web

Zadání:

1. Seznamte se s programovacím jazykem Python a jeho knihovnami Peewee, Flask a Unittest.
2. Seznamte se s metodikou návrhu software ve výzkumné skupině SC@FIT.
3. Prostudujte softwarový balík k-Dispatch a jeho komunikační rozhraní.
4. Navrhněte webovou aplikaci pro administrátory výpočetních služeb na superpočítačích využívající systém k-Dispatch pro možnost správy a monitorování úloh v online režimu.
5. Navržené řešení implementujte, zdokumentujte a ověřte jeho funkci pomoci jednotkových testů i typických uživatelských scénářů.
6. Zhodnoťte dosažené výsledky, jejich přínos pro administrátory superpočítačových systémů a diskutujte možnosti pokračování projektu.

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 https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Jaroš Marta, Ing.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 30. října 2020

Abstrakt

Práce je zaměřena na tvorbu webového rozhraní pro systém k-Dispatch, pomocí kterého je možné monitorovat, vytvářet a editovat již existující úlohy, uživatele, stroje, alokace a skupiny. Návrh aplikace byl vytvořen na základě principu rozložení obsahu na stránce a principu seskupení objektů na základě podobnosti. U postupu návrhu jsou porovnány dostupné nástroje. Implementace je napsána převážně v jazyce Python3 za využití frameworku Flask, který usnadňuje zpracování požadavků a autentizaci uživatelů. Dále jsou v implementaci využité jazyky HTML, JavaScript, Jinja2 a jiné. Nakonec je v práci zmíněno samotné testování, pro které byly použité jednotkové testy společně s technologií Selenium a uživatelským testováním.

Abstract

Bachelor's thesis focuses on web development process to create interface for k-Dispatch system, with which we can monitor, create and edit existing tasks, users, machines, allocations and groups. Web design was created based on web content layout principle and object grouping based on similarities principle. Comparison of usable tools is in design procedure section. Implementation uses mainly written in Python3 language with usage of Flask framework, which is helpful with processing requests and user authentication. Furthermore in implementation are used languages as HTML, JavaScript, Jinja2 and other. For test phase were used unit tests together with Selenium interface and user testing.

Klíčová slova

Flask, Webové rozhraní, Python, Peewee, PostgreSQL, nástěnka, JavaScript, návrh webu, HTML, CSS, Jinja2, jednotkové testy, Selenium

Keywords

Flask, Web interface, Python, Peewee, postgreSQL, dashboard, JavaScript, web design, HTML, CSS, Jinja2, unit tests, Selenium

Citace

DANČÁK, Petr. *Webové rozhraní pro správu a monitorování úloh na superpočítačích.* Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Marta Jaroš

Webové rozhraní pro správu a monitorování úloh na superpočítačích

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením paní Ing. Marty Jaroš. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Petr Dančák
10. května 2021

Poděkování

Rád bych poděkoval vedoucí mé práce Ing. Martě Jaroš, která se mi po celou dobu bakalářské práce věnovala.

Obsah

Kapitola 1

Úvod

Jednou z nejvíce obecně používaných technologií ve světě internetu jsou webové stránky. Jejich existence lidem usnadnila život, a to z více důvodů. Mezi tyto důvody patří snazší přístup k informacím nebo možnost nakoupit z pohodlí domova bez ohledu na to, jestli je člověk odborník nebo méně zkušený uživatel. Proto by webová stránka měla být co nejvíce intuitivní, ale zároveň obsahovat veškerou funkcionalitu, kterou uživatel vyžaduje. V určitém okamžiku může nastat problém, jelikož čím více funkcionality webová stránka má, tím se stává méně intuitivní. Proto by měl programátor najít kompromis mezi zmíněnými vlastnostmi.

Tato práce vychází z již započaté implementace, která byla zadána výzkumnou skupinou superpočítačových technologií **SC@FIT**[1].

Cílem mé bakalářské práce bylo navrhnout a implementovat webové rozhraní pro správu a monitorování úloh na superpočítači. Webová aplikace cílí na různé typy uživatelů. Uživatelům umožňuje zadat výpočetní úlohy, jejichž výpočet je realizován na superpočítačích, a monitorovat stav jejich výpočtu. Aplikace pro účely monitorování vytváří nástěnku s grafy, statistikami a upozorněními. Obsah nástěnky je dynamicky přizpůsoben roli uživatele a vybranému časovému intervalu k monitorování. V aplikaci je možné zobrazený obsah řadit a filtrovat. Uživatelé jsou v aplikaci shromážděni do skupin.

Práce je koncipována do kapitol, kde první kapitolu tvoří úvod. Ve druhé kapitole je popsána teoretická část obsahující popis systému, pro který je práce vytvořena a teorii k návrhu, podle které se má programátor řídit. Ve třetí kapitole jsou popsány technologie, které byly použity nebo souvisí s prostředím práce. Například jazyk **Python** [40], vývojové prostředí **PyCharm** [30], značkovací jazyk **Jinja2** [33], *framework* **Flask** [16] a jeho moduly, značkovací jazyk **HTML** [9], technologie **Bootstrap 3** [39], **JavaScript** [36] a technologie pro tvorbu grafického návrhu vzhledu webu. Ve čtvrté kapitole se nachází postup práce při návrhu aplikace, specifikace cíle práce a návrh řešení. V páté kapitole je popsána obsluha a implementace jednotlivých stránek, funkčních prvků, kde se práce detailně věnuje cíli implementované části a jejímu řešení. V šesté kapitole je popsán proces testování jednotkovými a uživatelskými testy. Poslední kapitolou je závěr, ve kterém je vyhodnocení práce a popis možného pokračování práce.

[1] https://www.fit.vut.cz/research/group/sc@fit/.cs

Kapitola 2

Teoretická část

2.1 Vývoj webové aplikace

Vývojem se rozumí proces, který je spojený s tvorbou webové aplikace. Tento proces je převážně zaměřený na programování komunikace klienta se serverem a k jejich interakci za použití webového prohlížeče. Ve většině případů vývoj webové aplikace zahrnuje definici problému, návrh řešení problému, ve formě grafického návrhu, konzultaci s potenciálními zákazníky, studium vhodných technologií, implementaci a testování řešení. [31]

Webová aplikace

Webová aplikace je počítačový program s interaktivním webovým prostředím používající technologie **HTML**, **CSS**, **JavaScript** a často i nějaký typ databáze. Databáze musí podporovat operace **CRUD**, což jsou základní operace nad perzistentními daty. **CRUD** je zkratka pro anglická slova s významem:

- *Create* - Vytvoř nový záznam v databázi.
- *Read* - Přečti záznam z databáze.
- *Update* - Uprav záznam v databázi.
- *Delete* - Smaž záznam z databáze.

K webové aplikaci se přistupuje pomocí webových prohlížečů. Jelikož často dochází k záměně pojmů web a webová aplikace, tak jsou tyto pojmy definovány následovně. Web je často statický a slouží pro příjem informací, zatímco webová aplikace podporuje interakci a slouží k obsluze **CRUD** operací. [31]

Proces vývoje

Proces vývoje webové aplikace se skládá z více podprocesů. Prvním podprocesem je definice cíle. Aby bylo vůbec možné začít navrhovat aplikaci, tak musí být nejdříve jasné, k čemu aplikace slouží. Druhým podprocesem je plánování funkčnosti webové aplikace. Jakmile je jasné, k čemu aplikace slouží, tak je důležité specifikovat co aplikace bude umožňovat. Poté, co se dokončí podproces plánování funkčnosti, se může začít tvořit vizuální návrh na základě zaměření aplikace. Výsledek vizuálního návrhu se pak musí ověřit. Nejlépe se výsledek ověří prezentací cílovému uživateli. Ten pak sdělí svůj názor na základě kterého

se návrh upraví. Jakmile je návrh ověřený, tak se začne volit vhodná technologie, která se použije k realizaci. Volba technologie se neřídí dle popularity, nýbrž tím, co je vhodné použít, jelikož některá technologie může být zbytečně složitá nebo primitivní. Po zvolení technologie se postupně začne s implementací. Nejdříve se u databáze zvolí, co a jak se bude ukládat, a podle toho se vytvoří databázové schéma. Poté se začne zároveň pracovat na vnější a vnitřní stránce aplikace. Vnější stránka aplikace by se měla co nejvíce podobat konečné podobě návrhu. K dosažení cílového vzhledu se použije technologie **HTML**, **CSS** a **JavaScript**. Zároveň s vnější stránkou se pracuje na vnitřní stránce, která je často implementačně složitější. Hlavní funkcí stránky je propojení klienta s vizuální stránkou obohacenou o data z databáze po provedení autorizace uživatele. Předposledním podprocesem je otestování webové aplikace. Testování webové aplikace se vykonává během implementace a po jejím dokončení. Testování může probíhat automaticky i manuálně, kde záleží na typu testu. Testuje se z hlediska ověření funkčnosti, využitelnosti, kompatibility a bezpečnosti aplikace. Posledním podprocesem je zajištění kvalitních serverů, na kterých aplikace bude umístěna. Společně se zajištěním serverů je nutné si zaregistrovat doménu sloužící jako přístupový bod aplikace. Pokud se jedná o službu, která se umístí na *cloudové* servery, tak bude nutné zajistit správnou instalaci implementace použitím kontejnerů. [31]

2.2 Teorie k návrhu

Při návrhu webové stránky je nutné vzít v potaz to, jak uživatel graficky zpracovává zobrazovaný obsah stránek. To popisuje princip rozložení obsahu na stránce. Z toho se uživatel dozví, jak umístit prvky, aby uživatel měl nejdůležitější části stránky v nesouladu, nebo v souladu s vizuálním vstřebáváním obsahu. Pokud programátor chce, aby byl obsah přehledný, tak by se měl těchto vzorů držet. V případě potřeby je možné uspořádat prvky v nesouladu se vzory. Nesoulad se vzorem se využívá u článků a podobných typů webů, protože nesoulad nutí uživatele přečíst celý obsah. Uživatelé vstřebávají obsah v určitých vzorech. Jedním z těchto vzorů je tvar písmene **F** [34]. Uživatel nejdříve čte obsah v horizontální směru a následně ve vertikálním. Vzorů, podle kterých uživatel získává informace, existuje mnoho, ale nejčastěji se uvádí tvar písmene **F**. Tohle bylo zjištěno pozorováním lidského oka určitého vzorku lidí. Dále se doporučuje nastudovat princip seskupení objektů na základě podobnosti. Základem principu je, že uživatelé vnímají lépe grafy se stejnou velikostí, tlačítka s podobnou funkčností, umístěním a podobně. Dále se programátor musí vyvarovat zbytečnému přehlcování obsahu a rozdělit ho na logicky jednodušší prvky. Příkladem je návrh stránky s uživatelským profilem, kde je vhodné oddělit změnu informací od změny hesla. [22] [2]

V praxi je nutné umět vyhodnotit implementační náročnost s použitím daných technologií, finanční náročnost z hlediska licencí, platů zaměstnanců a časovou náročnost z hlediska stanovených termínů. [22]

Pro design je nezbytně důležité, aby byl jednoduchý, přehledný a intuitivní. Čehož lze dosáhnout seskupením částí, které spolu souvisí do větších celků vhodně rozeznatelných od ostatních. Design nesmí být zbytečně barevně pestrý, ale měl by využívat různých odstínů barev a barevných palet, kde barvy řídících prvků drží jejich funkčnost. [22]

Na začátku, kdy ještě není jasné, jak by měl vzhled vypadat, je vhodné načrtnout hrubé rozložení prvků. Tím dojde k ujasnění představy o vzhledu aplikace a začne proces tvorby *wireframu* (makety). Makety slouží k přesnějšímu návrhu vzhledu, ke kterému by se měl vzhled cílové aplikace co nejvíce přiblížit. Na rozdíl od tvorby skic se pro tvorbu maket využívají editory. [2]

Tvorba nástěnky

Pro přehledné zobrazení stavu webové aplikace se využívají nástěnky, které se uvádí pod anglickým názvem *dashboard*. Nástěnky jsou základní část při sdělení informací uživateli. Nástěnky je možné modifikovat, aby vyhovovali konkrétním potřebám uživatele. Dobře navržená nástěnka dokáže uživateli zkrátit analýzu informací o velké množství času a sdělit mu informace, které by jinak nebyly viditelné. Nástěnky mohou obsahovat informace o systému v textové i grafické podobě. Konkrétními prvky nástěnky jsou grafy a tabulky.

Při tvorbě nástěnek je vhodné dbát na správné aspekty návrhu. Jedním z aspektů je fakt, že člověk vnímá spíše celky než větší počet dílčích prvků. Proto se používá pravidlo jednoduchosti. To doporučuje prvky shlukovat do větších logických celků. Dalším aspektem, na který by měl programátor myslet, je pravidlo blízkosti. Pravidlo říká, že funkční celky se rozdělují na základě pozice a vzdálenosti. Toho lze dosáhnout tak, že navzájem související prvky budou v bezprostřední blízkosti, zatímco jednotlivé celky se nacházejí v dostatečné vzdálenosti. Za důležitý aspekt se považují také barvy prvků, které by měly dávat v určitém kontextu smysl. Nepoužívat zbytečně moc barev a barevně odlišovat důležité části, aby se neztratila výraznost. Barevné odlišení objektů by mělo mít nějaký význam. U grafů a tabulek se nedoporučuje přidávat zbytečná ohraničení a rámy. To samé platí i pro dekorace a 3D prostorové grafy.

Existují různé typy grafů, které se používají k rozdílným sdělením. Pokud chce programátor porovnávat hodnoty, tak se vyplatí použít sloupcové grafy, protože rozdíly v rámci koláčového grafu nemusí být zřetelné. Pro vývoj hodnot v čase se vyplatí použít spojnicový graf. Při zobrazení distribuce hodnot se doporučuje graf hustoty a histogram. U korelace hodnot je vhodné použít bodový a bublinový graf. Jestliže se jedná o data spojená s územím, tak je možné použít kartogram.[22]

2.3 Teorie k testování webové aplikace

Testování webové aplikace je proces kontroly výskytu chyb, které byly zavedeny během implementace. Testování se provádí předtím, než se aplikace zpřístupní pro cílové uživatele jako konečný produkt. Skládá se z ověření funkcionality, použitelnosti, bezpečnosti a kompatibility.

1. **Testování funkcionality** ověřuje uživatelské rozhraní, aplikační rozhraní a databázi použitím automatických nebo manuálních testů. Provádí se za účelem ověření možných funkcí. Testuje dostupnost všech odkazů, funkčnost odesílaných formulářů, grafické funkční prvky a proveditelnost jednotlivých úkonů, které aplikace umožňuje. [18]

2. **Testování použitelnosti** se zaměřuje na orientaci uživatelů v systému aplikace. Především se testuje intuitivnost přístupu na různé stránky, viditelnost všech odkazů, dostatečná výraznost tlačítek a dobrá zpracovatelnost obsahu [18]

3. **Testování rozhraní** ověřuje bezproblémový provoz aplikačního, webového a databázového rozhraní. U aplikačního rozhraní dochází k ověření správně předaných chybových kódů a těla odpovědi. U webového rozhraní přijímání požadavků a u databáze získávání dat s předem očekávanými výsledky. Jde především o testování komunikace jednotlivých vrstev aplikace. [18]

4. **Testování kompatibility** určuje, zda webová aplikace je správně zobrazena na požadovaných, často nejpoužívanějších webových prohlížečích. Do této kategorie testování patří ověření správného zobrazení aplikace i na mobilních zařízeních. Tohle testování je velmi důležité, jelikož jednotlivé prohlížeče mohou mít rozdílně řešené zobrazení a funkčnost použitých prvků. Rozdílné chování se může projevit i napříč jinými verzemi webového prohlížeče. [18]

5. **Testování bezpečnosti** je především nutné řešit, pokud se jedná o aplikaci obsahující soukromá data. Testování probíhá ověřením neautorizovaných přístupů, kde je uživateli bez nutných oprávnění zamítnut přístup. Dále se ověřuje zabezpečení přenosu dat a proces autorizace uživatele. [18]

6. Poslední fáze testování je **uživatelské testování**. To probíhá vytvořením testovacích scénářů. Testovací scénář je seznam kroků, které musí tester provést za účelem ověření správného chování. Scénáře se předají vybrané skupině lidí o určité velikosti k vykonání se záměrem získat zpětnou vazbu. [18]

2.4 Verzování aplikace

Verzování je proces sloužící k zaznamenávání změn. Jedná se především o změny v souborech, které je možné i po čase vrátit do předchozí podoby. Zároveň verzování slouží k ověření změn prováděné různými uživateli. Proces verzování navíc umožňuje udržovat aktuální verzi aplikace napříč různými počítači. [6]

Existují různé typy verzování, ale hlavní jsou pouze tři. Prvním typem je lokální verzování, kdy jsou soubory zálohovány v místní databázi. Zde se pak ukládají všechny provedené změny. Jedná se o nejstarší typ verzování a jeho nejrozšířenějším představitelem je nástroj **RCS**[1]. Dalším typem je centralizované verzování, kterého je představitelem **SVN**[2]. Považuje se za následovníka lokálního verzování, kdy centralizované verzování vzniklo za účelem společného vývoje aplikací. Pro tyto účely se využívá server obsahující všechny zálohované soubory. To má řadu výhod. Například je možné spravovat úpravu souborů a spolupracovníci tuší, kdo na čem pracuje. Naopak velkým problémem je výpadek serveru. Během výpadku není možné přistupovat a ukládat soubory. Poslední typ se nazývá distribuované verzování. Tento typ verzování se od ostatních liší tím, že na klientské straně není pouze aktuální verze aplikace, ale zároveň je tam i celá databáze změn, která byla v předešlém typu pouze na serveru. Tím se vyřeší problém minulého typu, kdy mohlo dojít k výpadku serveru společně se ztrátou dat. Teď by server pouze získal zálohu od některého z klientů a tím i vrátil databázi změn do původního stavu. Tohoto využívá například technologie **GIT** [6].

GIT v SC@FIT

Pro verzování a zálohování aplikace byl vytvořen *repozitář* s využitím služby **GIT** na serveru skupiny **SC@FIT**[3]. Zde se struktura, **issues**, spojování větví, názvy větví a mnoho dalšího řídí pravidly právě této skupiny. Pravidla jsou popsána v textovém souboru **SC-FIT-handbook** [29] a je kladen velký důraz na jejich dodržování.

[1] https://www.gnu.org/software/rcs/
[2] https://subversion.apache.org
[3] https://www.fit.vut.cz/research/group/sc@fit/.cs

V **SC-FIT-handbook** se vyskytuje kapitola o pracovním postupu s technologií **GIT** v rámci této skupiny. Ten popisuje základní scénáře, jak začít nový projekt, implementovat nové rozšíření, nahlašovat chyby, vytvořit novou větev v **GIT** a jak napsat požadavek na sloučení větví.

- Pokud chce programátor založit nový projekt, tak musí vytvořit nový **GIT** repozitář. Poté musí nahrát list běžných značení a přidat první *issue* s určitým názvem, pravidelně číslovaným od jedničky. Po přidání *issue* musí programátor vytvořit strukturu předem určených souborů, které uloží do větvě a spojí s hlavními větvemi *master* a *development*. Nakonec programátor *issue* uzavře.

- Proces vývoje softwaru probíhá tak, že se nejdříve přidá nová *issue*, u které se využije některá ze šablon. Po přidání *issue* se vytvoří nová větev odvozená z větve *development* s názvem stejným, jako byl udán v *issue*. Postupně, jak programátor pracuje, tak ukládá postup do nově vytvořené větve k daným změnám společně s komentáři. Vývojář pak na konci implementace zdokumentuje změny tak, že je zapíše do obsahu *issue*.

- Spojení větví probíhá tak, že programátor stáhne změny z *development* větve a vyřeší konflikty, které mohly vzniknout spojením větví. Programátor pak uzavře *issue* pro aktuální větev a vytvoří nový požadavek na spojení aktuální větve s vyřešenými konflikty do *development* větve a požadavek přiřadí nějakému z kontrolorů. Kontrolor pak provede kontrolu a buď spojení povolí, nebo zamítne.

Pokud se mluví o *issue* v kontextu technologie **GIT**, tak se jedná o specifický nástroj sloužící programátorům k popisu zaváděných změn v softwaru. Každá *issue* má svého řešitele, automaticky přiřazený identifikátor, značení, ze kterého se dá předběžně poznat o jaké změny se jedná a další informace. Na *issue* je možné odkazovat v textu použitím mřížky následované identifikátorem *issue*. To je vhodné použít například v textu u požadavku na spojení větví. Po dokončení změn se musí uzavřít *issue*, čímž se oznámí ostatním, že je proces zaváděných změn dokončen. Uzavřená *issue* se nesmí smazat, aby nedošlo ke ztrátě možnosti sledovat postup vývoje. [29]

2.5 k-Dispatch

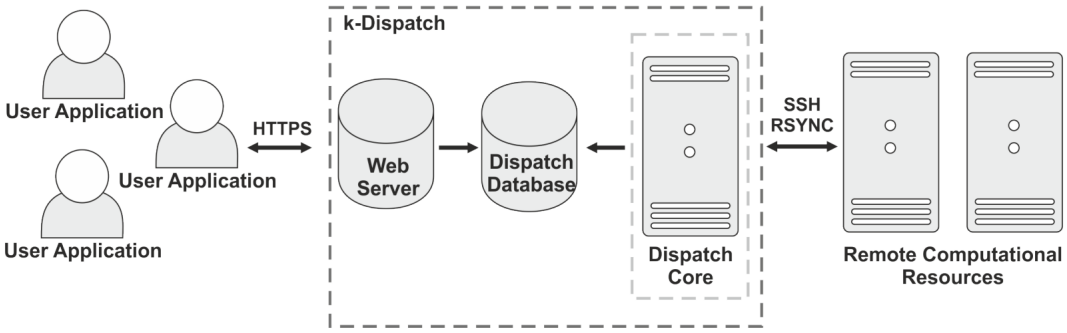
O k-Dispatch

k-Dispatch [28] je služba umožňující uživateli zadávat úlohy, monitorovat jejich průběh a spravovat je na superpočítačích. Služba usnadňuje uživateli práci abstrakcí operací, aby byl uživatel co nejvíce odstíněn od implementačních detailů. Uživatelé pracují v rámci skupiny s předem domluvenými výpočetními prostředky, které může skupina využít. Domluva je v systému zadána jako alokace. Alokace je zakoupený výpočetní čas na určitém výpočetním prostředku.

K výpočtu úloh jsou použity *cloudové* služby a superpočítačová centra. Systém dále zajišťuje proces kontrol a oprav chyb, aby byl uživatel projektu izolován od vnitřních problémů. V systému dochází k optimalizaci práci s poskytnutými zdroji pro jednotlivé úlohy, aby se minimalizoval výpočetní čas a tím i konečná cena. Po spuštění se periodicky monitoruje průběh jednotlivých úloh a aktualizují se údaje v databázi, aby bylo možné uživateli sdělit aktuální informace o zadané úloze. [28]

Architektura

k-Dispatch se skládá ze tří hlavních modulů, které lze vidět na obrázku 2.1. Jedná se o webový server, databázi a jádro. Uživatel komunikuje s webovým serverem pomocí **REST api** webové aplikace. **REST api** je aplikační rozhraní, které poskytuje možnost komunikace se serverem ve formě zasílání požadavků. Ta pak komunikuje s databází na základě uživatelských akcí a autentizačních údajů od uživatele. Databáze obsahuje všechna potřebná data o uživatelích, léčebných plánech (zadaných úlohách), úkolech, skupinách, alokacích a mnoha dalších informacích. Data se postupně aktualizují z výpočetních prostředků jádrem, které se stará o jejich řízení, kontrolu a monitorování. [28]



Obrázek 2.1: Struktura služby k-Dispatch. Převzato z [27].

Databáze

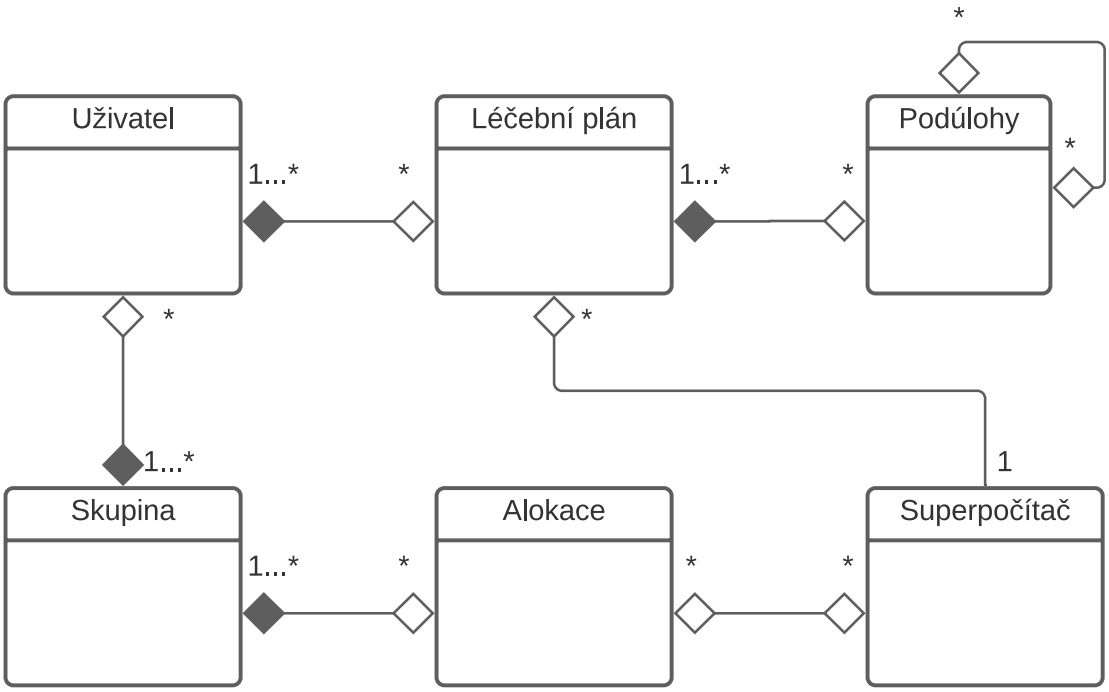
Služba **k-Dispatch** používá pro ukládání dat relační databázi **PostgreSQL** [17]. Při práci s daty v databázi se počítá s přístupem pouze u řídícího jádra a webové rozhraní služby **k-Dispatch**.

V textu se nachází pojmy jako výpočetní prostředek, superpočítač, *cluster* a *cloud*. Tato pojmenování jsou synonyma a slouží jen pro obohacení obsahu. Pod textem na obrázku 2.2 lze vidět velice zjednodušené schéma databáze bez atributů a mnoha dalších tabulek, které jsou méně relevantní k mé práci. Schéma popisuje léčebné plány, které mají své podúlohy a byly spuštěny na nějakém výpočetním prostředku. U podúloch jsou popsány jednotlivé závislosti na jiných podúlohách, jejich příslušnost k uživatelům v systému, stav a další informace. Dále databázové schéma obsahuje informace o skupinách, jejich členech, alokacích a dalších objektech. Skupiny jsou v databázi celky, jenž mají představovat určité shluky uživatelů, jako je například personál nějaké nemocnice, vědecká skupina a jiné. U uživatelů musí být v databázi specifikováno jejich jméno, příjmení, heslo uložené v zašifrované podobě, přihlašovací jméno a jejich role. Role se dělí na **Reviewer, Planner, Hospital Admin** a **DSM Admin**. Význam rolí je následující:

- **DSM Admin** má oprávnění vytvářet nové skupiny, uživatele, výpočetní prostředky, alokace, spravovat prostředky a zobrazit si mnohem rozsáhlejší množství dat než jiní uživatelé. Možností, co může uživatel s touto rolí dělat, je více, ale vypsány jsou pouze ty hlavní.
- **Hospital Admin** je správce určité skupiny. Může vytvářet nové členy skupiny, měnit jejich role, zobrazit si léčebné plány skupiny a další.

- Hlavním účelem role **Planner** je vytvářet nové léčebné plány.
- **Reviewer** má možnost kontroly určitých léčebných plánů.

Pro řízení přístupu k výpočetním prostředkům a jejich domluveným omezením slouží alokace. Alokace obsahuje datum pro začátek a konec platnosti alokace, aktuální stav, příslušnost skupin k superpočítačům a jiné. K popisu úloh je zapotřebí v databázi uchovávat jména souborů, ze kterých byly úlohy vytvořeny, stav úlohy, zadavatele, počet neúspěšných pokusů a další informace. [28]



Obrázek 2.2: Zjednodušená část databázového schématu.

Kapitola 3

Technologie

Kapitola pojednává o technologiích, které byly z důvodu tvorby bakalářské práce prostudovány a porovnány s konkurenty. Popis technologií bude obsahovat informace o jejím správci, obecný popis, výhodách, nevýhodách a možnosti uplatnění v projektu.

3.1 Programovací jazyk

Výběr vhodného programovacího jazyku

Práce vychází z již započaté implementace. Implementace je původně naprogramovaná v jazyce **Python3** [40]. Programovací jazyk byl vybrán, protože je výkonnostně dostačující, vhodný pro prototypování, dobře rozšiřitelný a poskytuje velké množství dostupných modulů.

O jazyku Python

Python3 je nástupce programovacího jazyku **Python2**. Jedná se o jazyk dynamicky typovaný, který je známý především svou striktností a přehledností kódu. Provádění programu zařizuje interpret. Bohužel se kvůli tomu hůře provádí statické testování. Výhodou je větší volnost při práci s parametry funkcí a proměnnými. Jazyk je mimo jiné objektově orientovaný. Umožňuje použití tříd a jejich metod.

Třídy jsou v programovacích jazycích novodobé struktury, které na sebe mohou navazovat. Navazování se u tříd říká dědičnost. Třídy mohou obsahovat metody, což jsou funkce třídy, které nějakým způsobem pracují s vlastními daty. Používáním metod vzniká určitá abstrakce, díky které má programátor větší možnosti. [5] [8] [37] [4]

Ve zbytku práce bude **Python3** pro zjednodušení nazýván **Python**.

3.2 Vývojové prostředí

Výběr vhodného vývojového prostředí

Po výběru programovacího jazyku je dalším krokem volba vhodného prostředí, které práci na projektu usnadní a zefektivní. Jedním z často používaných vývojových prostředí pro jazyk **Python** je **PyCharm** [30]. **PyCharm** je mezi vývojáři velmi oblíbený díky svému modernímu vzhledu a funkčnosti, kterou nabízí a skvělému ladícímu nástroji. Dalším velmi

oblíbeným **Python** vývojovým prostředím je prostředí **Eclipse**[1]. **Eclipse** byl původně vytvořen pro vývoj odlišného programovacího jazyka, ale díky rozšířením je v něm možné vyvíjet i **Python** aplikace. **Eclipse** je oblíbený kvůli nápovědě při tvorbě kódu, ladícímu nástroji, podpoře *frameworku* **Django** [14] a interaktivní konzoli. Posledním zmíněným vývojovým prostředím je prostředí **Visual Studio Code**[2] vyvíjené společností **Microsoft**[3]. Prostředí obsahuje spoustu rozšíření. Jedním z rozšíření je rozšíření o vývoj jazyka **Python**. Velkou výhodou tohoto prostředí je jeho přehlednost, podpora velkého množství rozšíření, nápověda při tvorbě kódu a podpora různých operačních systémů. [13]

Pro účely tvorby webového rozhraní se nejvíce vyplatí vývojové prostředí **PyCharm**, které se více hodí na tvorbu webových aplikací a nabízí studentům profesionální verzi zdarma.

O vývojovém prostředí PyCharm

PyCharm je vývojové prostředí, které slouží výhradně pro vývoj **Python** aplikací. Obsahuje velký počet podpůrných nástrojů, které pomáhají vývojářům při vývoji webových aplikací. Jedním z těchto nástrojů je propracovaný ladící nástroj. Další užitečnou funkcí vývojového prostředí je jeho napovídání při psaní kódu. Pro každý projekt v **PyCharmu** je možné zvolit existující prostředí, případně vytvořit prostředí nové. Prostředím se rozumí verze **Python** interpretu a jeho moduly. Prostředí je možné vytvořit manuálním výběrem z nabídky nebo na základě některého správce balíčků **Anaconda** [1] a **Pip** [10].

PyCharm je produktem společnosti **JetBrains** [30]. Společnost nabízí tři podobné edice vývojového prostředí, které se liší v nabídce funkcionality a ceně. Samotná základní verze je zdarma pro všechny a je možné ji stáhnout z oficiálních stránek. Profesionální verze je placená, ale pro studenty je možné získat licenci zdarma.

3.3 Správa modulů

Pro spuštění aplikace je nutné mít na systému nainstalované požadované moduly. Správnou instalaci modulů zajišťují správci balíčků, kteří požadované moduly stáhnou, nainstalují a nastaví. Mezi nejpoužívanější správce modulů pro jazyk **Python** jsou správci **Anaconda** [1] a **Pip** [10].

Anaconda se používá k ukládání a sdílení prostředí mezi uživateli a jejich stroji. Stará se o aktualizaci balíčků na novější verzi. Poskytuje privátní i veřejné sdílení prostředí. **Pip** je správce balíčků pro jazyk **Python**. Je možné jej použít za využití příkazové řádky.

3.4 Framework a moduly pro tvorbu webové aplikace

Výběr vhodného frameworku pro vývoj webové aplikace

Pro tvorbu webové aplikace se používají *frameworky*, které programátorovi usnadní práci. *Frameworky* programátorovi pomohou s komunikací, řízením systému, vytvářením odpovědí, zpracováním požadavků a přesměrováním uživatele podle standardů.

V jazyce **Python** jsou dva hlavní *frameworky* pro tvorbu webových aplikací, které byly určeny na základě popularity a velikosti komunity. Těmito *frameworky* jsou **Django**[14]

[1] https://www.eclipse.org
[2] https://code.visualstudio.com
[3] https://www.microsoft.com/cs-cz

a **Flask** [16]. Každý z těchto *frameworků* má své výhody v jiné oblasti. Samotný **Flask** je oproti **Djangu** lehčí na naučení, protože není tak komplexní jako samotné **Django** a je více svobodné v použitých modulech. Doporučuje se začátečníkům a uživatelům, kteří vytváří méně rozsáhlou webovou aplikaci. Konkrétně je v **Djangu** integrované objektově relační mapování, které ve **Flasku** není, ale může být přidáno moduly. Dále **Django** obsahuje integrované funkce pro práci s přihlašováním, správou účtů a podporou *session*. Ve **Flasku** nic takového integrováno není, ale je možné použít rozšíření pro podobnou funkčnost jakou má **Django**. To samé platí i pro administrátorské prostředí, které je opět v **Djangu** na rozdíl od **Flasku** integrováno. Obecně platí, že **Django** má většinu možností integrovaných, zatímco **Flask** je nutné doplnit o rozšíření ve formě modulů, které nemusí spolu plně spolupracovat, jako v případě **Djanga**. [20]

Flask

Flask [16] je *framework* pro tvorbu webových aplikací v jazyce **Python**. Usnadňuje práci se zpracováním klientských požadavků a nabízí programátorovi možnost vytvářet odpovědi. Před zpracováním požadavku kontroluje, jestli je **HTTP** metoda požadavku podporována. Pokud metoda podporována není, tak dochází k automatickému zaslání **HTTP** stavového kódu 405. **HTTP** kód 405 znamená, že metoda není na serveru povolena.

Pro zpracování požadavků se vytváří funkce, které obsahují chování aplikace. U funkcí lze definovat podporované metody **HTTP** požadavků a jiné vlastnosti. Nad každou funkci je možné vložit dekorátor. Dekorátor je odkaz na funkci, která může kontrolovat autentizaci uživatele před vykonáním funkce.

Možnosti a moduly Flask

Flask poskytuje programátorům velkou škálu modulů, nástrojů, tříd a mnoho dalších prvků, díky kterým je vývoj webu mnohem jednodušší. Modulů *frameworku* **Flask** je mnoho, proto budou popsány pouze vybrané.

flask-login

Jedná se o modul rozšiřující funkčnost **Flask** o správu a rozpoznávání uživatelů komunikujících s webovou aplikací. Podporuje ukládání dočasných informací o uživateli, jako je například identifikační číslo uživatele. [12]

flask-httpauth

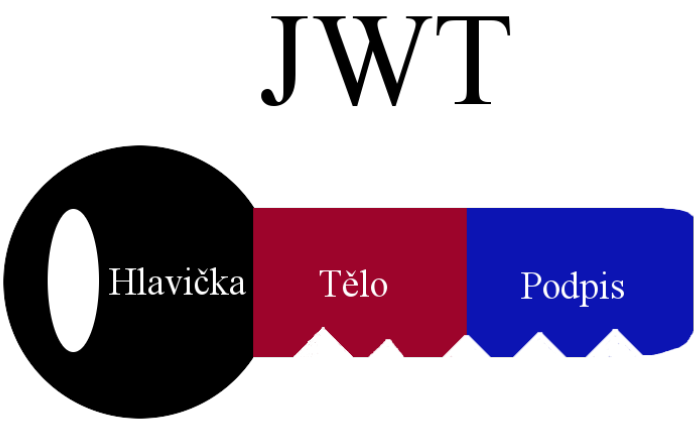
Modul **flask-httpauth** [11] slouží pro restrikci přístupu k **REST api**. Kontroluje práva jednotlivých uživatelů pro přístup ke zdrojům a jejich použití při každém požadavku.

Rozdíl mezi flask-httpauth a flask-login

Hlavním rozdílem těchto dvou modulů je odlišný způsob identifikace, kdy **flask-login** využívá pro autentizaci klientskou a serverovou *session*, ve které uschovává informace pro autentizaci. Na rozdíl od toho **flask-httpauth** autentizuje uživatele pomocí informací obsažených v **HTTP** hlavičce, ve které je zadáno jméno a heslo. Odeslání jména a hesla je nutné při každém dotazu poskytnout v hlavičce **HTTP**.

JSON Web Tokens

Jedná se o modul, který slouží pro autentizaci na základě *tokenu*. Uživatel sdělí serveru své údaje, pomocí kterých se chce přihlásit, a pokud dojde k přihlášení, tak server vytvoří *token* na základě struktury zobrazené na obrázku 3.1. *Token* je zašifrován tajným symetrickým klíčem obsahující zakódované informace, mezi které může například patřit datum expirace *tokenu* a uživatelské jméno. Tímto řešením se zajistí, že uživatel nemá nikde uloženy své údaje, které by mohly být odcizeny. Namísto toho ukládá pouze *token*, který má expirační dobu, takže by případné zneužití nebylo trvalé. [3]



Obrázek 3.1: Struktura JWT *tokenu*.

3.5 Značkovací jazyk

Jedním z velice důležitých prvků, který *framework* **Flask** podporuje, je jazyk **Jinja2**. **Jinja2** slouží pro dynamické vytváření webového kódu na základě proměnných, podmínek a funkcí. Pro vytvoření webového kódu se využívá **Flask** funkce **render_template**. Funkce vyžaduje cestu šablony s **Jinja2** kódem, případně proměnné využité v šabloně. Jazyk **Jinja2** na základě podmínek a cyklů dokáže vytvořit uživatelsky specifický, přehledný a udržitelný kód.

Pro odlišení **Jinja2** kódu od webového kódu se používá značení. Existují dva typy značení. Jedno značení slouží pro zobrazení hodnot ve výsledném webovém kódu. Značí se dvěma složenými závorkami z obou stran, mezi které se vloží proměnná nebo výraz pro vypsání. Proměnné je možné získat z parametru funkce **render_template**. Druhé značení slouží pro postupné procházení listů, vykonávání cyklů a provádění určitého chování na základě vyhodnocení podmínky. Značení se skládá z podmínky, kódu a ukončovacího značení. Podmínka se vyskytuje mezi závorkami s procentem. Postupné procházení se provádí pomocí klíčového slova **for**, po kterém se napíše jméno prvku z listu a samotný list. Tím je možné postupně procházet prvky listu a pro každý prvek dynamicky vytvořit webový kód. Kód je umístěn mezi podmínku a ukončovací značení, případně další podmínku. Sa-

motné značení je tvořeno složenou závorkou na začátku a na konci, ke které se přidá značka procenta. Značení vypadá následovně:

```
{% Jinja2 kód %}
   webový kód
{% Jinja2 ukončení kódu%},
  {{ Jinja2 proměnná }}.
```

3.6 Práce s databází

Server, pro který je webové rozhraní určeno využívá databáze **PostgreSQL** [17]. Ke komunikaci s databází se využívá modul **Peewee** [32].

O Peewee

Peewee je **Python** modul, který slouží ke komunikaci s databází. Ke komunikaci využívá objektově relačního mapování. Objektově relační mapování je způsob přístupu k datům databáze jako k objektům v používaném programovacím jazyce. **Peewee** vyžaduje, aby ke každému modelu databáze (tabulky databáze) existovala **Python** třída. Modely databáze obsahují stejné řádky jako jednotlivé tabulky databáze, proto je možné brát model jako tabulku databáze a rovnou nad modelem provádět operace **SELECT, UPDATE, DELETE** a další. [21]

Peewee rozhraní

Pro práci s databází je vhodné mít zkušenosti s jazykem **SQL**, jelikož modul **Peewee** obsahuje rozhraní abstrahující jazyk **SQL** v jazyce **Python**. K realizaci operací nad databází slouží metody nad programátorem vytvořenými třídami. Využitím metod dojde k zpřehlednění a zjednodušení kódu.

Pro získávání dat se v jazyce **SQL** využívá operace **SELECT**. Tato operace je abstrahována metodou třídy, ze které dědí třídy databázových modelů. Metoda se nazývá **select** a vrací list řádků z tabulky. Pokud by bylo vhodné získat jen jeden řádek z tabulky, především z důvodu optimalizace, tak je možné použít metodu **get**. Metoda je podobná metodě **select**, ale vrací jen jeden řádek tabulky a je rychlejší. Jestliže se žádný řádek v tabulce nenachází, tak metody vyvolají výjimku **DoesNotExist**. Tím donutí programátora s touto skutečností počítat a ošetřit popsaný stav.

Vkládání dat do tabulky je v jazyce **SQL** realizováno operací **INSERT**. **Peewee** operaci abstrahuje metodou **create** pocházející z nadtřídy. Pro zachování perzistentní databáze se musí zapsané řádky uložit. K tomu slouží metoda nadtřídy **save**. Metodou **create** se vytváří pouze jednotlivé řádky tabulky. Pro vložení většího množství řádků, se vyplatí použít metodu **insert_many**.

Pro úpravu dat se v jazyce **SQL** používá operace **UPDATE**. **Peewee** provádí úpravu změnou hodnot objektu a následným uložením objektu. Přesný postup je získání objektu z databáze metodou **get** nebo **select**, následnou změnou objektu a uložením objektu metodou **save**. Pro odebrání dat z databáze slouží metoda **delete**, která se zavolá nad získaným objektem.

Na operaci **SELECT** je možné navázat klíčovým slovem pro specifikaci výběru. K výběru slouží klíčové slovo **WHERE**, za kterým se specifikují detaily výběru. V modulu

Peewee je klíčové slovo nahrazeno metodou **where**, které se podmínky předají jako argumenty metody. Metoda **where** navazuje na výstup metody **select**.

Pro možnosti řazení je v **SQL** možné použít klíčové slovo **ORDER BY**. Za klíčovým slovem se specifikují jména sloupců, podle kterých chceme záznamy řadit. Pořadí sloupců určuje prioritu řazení. U řazení lze zvolit řazení vzestupně, nebo sestupně. **Peewee** pro možnosti řazení poskytuje metodu **order_by**. V argumentech metody se specifikují sloupce výstupu a nastaví se řazení vzestupně, nebo sestupně funkcemi **desc** a **asc**. Pokud není specifikován typ řazení, tak dochází k sestupnému řazení. Metodu lze použít pouze na výstup metody **select**.

Pro získání výstupu spojení dvou tabulek a více tabulek, je nutné použít klíčové slovo **JOIN**. Klíčové slovo očekává jméno tabulky a podmínku spojení. Typů spojení tabulek se dají interpretovat Vennovými diagramy. Jedná se o typ sjednocení, průnik, inverzi a další. V **Peewee** je spojování tabulek zařízeno metodou **join**. Metoda očekává jméno tabulky pro spojení, argument typu spojení a výraz pro spojení tabulek. [32]

3.7 Tvorba webových stránek

HTML

HTML [9] je zkratka pocházející z anglického jazyka znamenající *Hyper Text Markup Language*. Volně přeloženo na hypertextový značkovací jazyk. **HTML** je standardní jazyk pro tvorbu webových aplikací. Slouží pro popis struktury webu použitím **HTML** značek. Značkám lze přiřadit vlastnosti jako je například třída, identifikátor a další.

Struktura **HTML** programu se vždy skládá z hlavičky a těla. Hlavička může obsahovat odkaz na nějaký kaskádový styl, který popisuje estetickou stránku webového zobrazení, titulek stránky, kódovaní znaků a jiné. Tělo programu se skládá z **HTML** prvků tvořící obsah stránky, jako jsou například nadpisy, bloky obsahující text, obrázky, odkazy a další.

HTML je velice využívaná, stále vyvíjená technologie. Poslední verze **HTML** je verze **HTML5** rozšiřující podporu pro video, audio a dále.

CSS

CSS [35], z anglického jazyka *Cascading Style Sheets*, je soubor popisů designu **HTML** značek. Značí se na základě tříd, identifikátoru a značek. **CSS** je možné specifikovat přímo u jednotlivých **HTML** značek nebo odděleně od **HTML** kódu v jiném souboru, kterých může být více. Popisovat lze polohu na webu, chování vůči jiným značkám, barvu textu, pozadí, reakce na uživatelovy akce. Akce může být najetí myší na značku, kliknutí a podobně.

CSS popis je možné nastavit buď staticky nebo dynamicky za použití jazyku **JavaScript** [36]. V jazyce **JavaScript** lze vybírat **HTML** značky a měnit jejich **CSS** popis.

Bootstrap

S vývojem logiky webových aplikací se vyvíjí i jejich zobrazení, které je neodmyslitelnou součástí všech webových aplikací. Proto se začaly používat metody pro urychlení a usnadnění procesu vývoje zobrazení. Jednou z technik je použití předem připravených pravidel. Pravidla obsahují **CSS** pro **HTML** kód, který si vývojář upraví pro své účely. [39]

Bootstrap je soubor předem připravených **CSS** pravidel a **JavaScript** kódů řídící design **HTML** značek. **Bootstrap** obsahuje pravidla pro zjednodušení rozložení značek

na stránce, responzivitu vzhledu na základě rozlišení, barvy pro zvýraznění účelu značek a další. K aplikaci pravidel se u **HTML** značek používá atribut *class*. V atributu se specifikují třídy, které se mají ke změně vzhledu značky použít.

Hlavní výhodou použití **Bootstrap** je rychlost návrhu vzhledu webové aplikace. K urychlení dojde použitím předem definovaných stylů, u kterých není nutné ověřovat funkčnost. Další výhodou je rozsáhlá komunita technologie **Bootstrap**. Výhodou velké komunity je pomoc při řešení problémů.

Javascript

Webové aplikace často potřebují dynamicky reagovat na akce uživatele, aniž by muselo dojít k novému načtení stránky. Právě k tomu slouží skriptovací jazyk **JavaScript** [36], který dokáže dynamicky měnit obsah webové aplikace a funguje na straně uživatele, takže zbytečně nezatěžuje server.

Javascript je interpretovaný objektově orientovaný skriptovací jazyk, který se hlavně využívá pro práci s webovými aplikacemi. Běh **JavaScript** programů je řízen na straně klienta, většinou v prohlížeči. Syntaxe jazyka je podobná jazyku **C++**.

Hlavní výhodou použití jazyku **JavaScript** je výsledná rychlost aplikace. **JavaScript** je totiž spuštěn na straně klienta, a tak se zbytečně nezatěžuje server. Další hlavní výhodou je jeho jednoduchost, která do vývoje webových aplikací neodmyslitelně patří. Webové aplikace často obsahují spoustu kódů, a tak je vhodné mít co nejmenší počet řádků kódů, čímž se obecně kód zpřehlední. Další velkou výhodou je popularita jazyku v komunitě programátorů. Díky tomu je možné na internetu najít spoustu návodů a rad. Poslední výhodou, která bude zmíněna je schopnost obohatit webové prostředí o interaktivní prvky. Interaktivní prvky se jinak velice špatně implementují bez použití jazyku **JavaScript**.

React

Jednou z často používaných knihoven pro vývoj **JavaScript** kódu je knihovna **React** [23]. Knihovna je udržovaná společností **Facebook**[4]. **React** je založený na tvoření jazyku z komponent. Komponenty jsou struktury popisující prvky aplikace. Strukturami se minimalizuje velikost kódu pro tvorbu uživatelského prostředí a zároveň se kód zpřehlední. Programátor může vytvářet a používat své vlastní komponenty nebo může použít **HTML** značky.

Vue

Další knihovnou pro skriptovací jazyk **JavaScript** je knihovna **Vue** [41]. Knihovna využívá komponenty, které se skládají z **HTML** a **JavaScript** kódu. Komponenty jsou pak dále používány společně s **HTML** značkami.

3.8 Tvorba návrhu webového prostředí

Pro tvorbu návrhů designu existuje spousta desktopových a webových aplikací s různými styly a funkčními prvky. Do porovnání nebudou zahrnuty placené nástroje.

[4] https://www.facebook.com

Porovnání aplikací

Výběr aplikací probíhal na základě doporučení a webového vyhledávání aplikací pro tvorbu *wireframů*. *Wireframe* je vizuální reprezentace aplikace nebo obecně produktu. *Wireframe* webové aplikace slouží pro zobrazení rozložení obsahu. Z výsledku hledání a doporučení byly vyzkoušeny některé aplikace, mezi které patří **MockFlow**[5], **MockingBird**[6], **framebox**[7], **Gimp**[8] a **PowerPoint**[9]. [38]

Všechny vyzkoušené aplikace byly zdarma k použití a lišily se v podstatných částech. V maximálním rozsahu návrhu na počet stránek. Ten byl u verzí zdarma často omezen. Dále počet možných prvků k použití představující tlačítka, články, loga a podobně. Poslední zásadní věc, ve které se aplikace lišily jsou formáty výstupní aplikace. Některé aplikace podporovaly velkou škálu formátů, některé například jen umožňovaly export do **pdf** nebo nepodporovaly exportování návrhu vůbec. **Gimp** je desktopová aplikace určená spíše pro účely kreslení. Na rozdíl od předešlých aplikací neobsahuje knihovnu předem připravených prvků. Výhodou je naprostá svoboda návrhu bez omezení týkající se chybějících prvků a omezení rozsahu. Výsledek je možné exportovat do souboru s příponou **pdf**, **png** a mnoho dalších. Tato aplikace se hodí spíše pro tvorbu skic. Nejvíce se od ostatních aplikací lišila aplikace **PowerPoint**. V aplikaci je na výběr z mnoha různých obrázků, které se dají kombinovat a nahradit jimi prvky webové aplikace. Počet stránek není ničím omezen a je možné exportovat návrh do souboru s příponou **pdf** a **png**.

Tvorba grafů

Z důvodu zobrazení dat na nástěnce je nutné zvolit vhodnou technologii pro tvorbu grafů pomocí jazyku **JavaScript**. Knihoven pro tvorbu grafů existuje hodně, ale práce se zaměří pouze na dvě z nich. Těmito knihovnami jsou **ChartJS** [7] a **Plotly** [25] [24].

Plotly poskytuje velké množství typů zobrazení. Grafy mohou být jednoduché a nebo velmi komplexní. Jedním z komplexnějších typů zobrazení je zobrazení na mapě, na které se dají zobrazit například body, přímky a dokonce i plochy. Knihovna nabízí mimo jiné i velkou škálu funkcí například pro různé typy přiblížení a oddálení grafu.

Pro tvorbu grafů byla vybrána technologie **ChartJS**. Tato knihovna je oproti předchozí o dost jednodušší a hodí se pro rychlé tvoření grafů obvyklých forem. Vývoj knihovny je řízen komunitou pod *open-source* licencí. K vizualizaci využívá podpory **HTML5** ve formě pláten. Je to jedna z nejvíce používaných knihoven pro tvorbu grafů ve webových aplikacích.

Pro vytvoření grafu je nutné nejdříve v **HTML** souboru vytvořit značku plátna, do kterého se pomocí jazyku **JavaScript** vkládají data. Poté je možné vytvořit graf ve formě objektu knihovny **ChartJS** s definovanými parametry.

3.9 Technologie pro testování

Testování je v dnešní době stěžejní část vývoje softwaru. Jedná se o proces prováděný za účelem odhalení chyb vzniklých psaním kódu. Proces se může provádět manuálně i automaticky. Pro oba tyto způsoby existují nástroje usnadňující hledání chyb. Mezi nástroje pro

[5] https://www.mockflow.com
[6] https://gomockingbird.com/home
[7] http://framebox.org
[8] https://www.gimp.org
[9] https://www.microsoft.com/cs-cz/microsoft-365/powerpoint

automatické testování webového uživatelského rozhraní patří například nástroj **Selenium** [19]. V případě manuálního testování je možné použít linuxové nástroje.

Selenium

Selenium [19] je *software* pro automatické testování uživatelského webového rozhraní. Uživatelské rozhraní **Selenium** testuje použitím *webdriveru*, což je nástroj nahrazující prohlížeč, nad kterým má **Selenium** implementované funkce. Funkce pak umožňují pracovat s elementy stránky. Elementy stránky je možné získat na základě specifikace, která může být zadána mnoha způsoby. Vyhledávání prvků může být zadáno specifikováním **CSS** tříd prvku a identifikátorem prvku. Pokud se na stránce vyskytuje více prvků se stejnou identifikací, tak **Selenium** vrátí pouze první nalezený prvek a ostatní ignoruje pokud se nejedná o funkci, která vrací všechny nalezené prvky. Vyhledávání na základě **CSS** tříd může být nevhodné pokud se často mění vzhled stránky. Důvodem je, že jakákoliv změna může ovlivnit chování testu. Pokud je získaný element například vstupní pole formuláře, pak je možné jej vyplnit. Dále **Selenium** umožňuje vykonat nějakou akci. Akce může být kliknutí na tlačítko, zaslání klávesy, posun myši a další. Pro ověření úspěšně dokončené úlohy je možné zkontrolovat výskyt elementů na stránce, případně ověřit aktuální *url* stránky a jiné.

Pro lehčí tvorbu testů bez znalosti rozhraní je možné použít rozšíření do prohlížeče. Rozšíření umožňuje zaznamenávat uživatelovy akce a následně je exportovat do vybraného programovacího jazyku. Jedná se o rozšíření webového prohlížeče s názvem **Selenium IDE**. **Selenium IDE** umožňuje vytvářet celé kolekce testovacích plánů a jejich scénářů.

unittest

Unittest [15] je *framework* pro tvorbu jednotkových testů. Slouží pro automatické testování. Poskytuje uživateli mnoho nástrojů, které se dají použít pro tvorbu a spouštění testů. Umožňuje programátorovi vytvářet kolekce testovacích případů. Testovací případ je určitá posloupnost podmínek, která se vykoná za účelem ověření bezchybné práce *softwaru*. Každý jednotkový test má několik fází. Některé z fází bývají u jednotkových testů stejné.

1. Fázi přípravy, ve které dojde k přípravě prostředí před zahájením testování nějakého z testovacích případů.
2. Fázi s testováním testovacího případu.
3. Fázi úklidu, kdy se vrátí změny, které testovací případ provedl, aby nebyly ovlivněny výsledky ostatních testovacích případů.

Kapitola 4

Návrh

Kapitola obsahuje detailní zadání bakalářské práce. Návrh vzhledu aplikace, k tomu použité technologie a proces testování.

4.1 Zadání práce

- Nastudovat si použité technologie započaté práce, teorii k návrhu a prozkoumat další technologie, o které by práce mohla být rozšířena. Použitými technologiemi jsou programovací jazyk **Python**, knihovny **Peewee**, **Flask**, **unittest** a správce balíčků **Anaconda**.
- Seznámit se s pracovním postupem výzkumné skupiny **SC@FIT** na základě příručky **SC@FIT Handbook**. K tomu mi byl poskytnut přístup k serveru pro zprovoznění aplikace.
- Prostudovat softwarový balík **k-Dispatch** a jeho komunikační rozhraní. Návody a popisky aktuálního řešení jsem mohl nalézt ve službě **GIT**, kde mi byl vytvořen repozitář.
- Navrhnout řešení webové aplikaci. Návrh bude obsahovat diagramy případů užití vycházejících z aplikačního rozhraní, návrh vzhledu jednotlivých stránek na základě teorie návrhu, **mockup** a návrh testování za využití jednotkových testů. Diagram případu bude obohacen o možnost data exportovat, filtrovat a řadit. Dále bude navrhnuta nástěnka a její sdělení uživatelům.
- Navržené řešení implementovat zvolenými technologiemi a kód řádně okomentovat. Implementovat jsem měl stránky pro jednotlivé zdroje z databáze. Stránky měli umožňovat jejich výčet, úpravu, deaktivaci a tvorbu. Pro zobrazení stránek na klientské straně využít technologie **HTML**, **CSS** a **JavaScript**. K zpracování klientských požadavků využít *framework* **Flask**. Pro stránku s nástěnkou získat data z databáze knihovnou **Peewee** a obsah zobrazit nějakou z **JavaScript** knihoven. Zajistit funkční systém přihlašování uživatelů, obnovy hesla a ošetření chybových stavů stránky. Kód bude okomentován tak, aby bylo možné z něj vytvořit **doxygen**[1] dokumentaci.
- Implementace se otestuje jednotkovými testy a uživatelským testováním. Pro účely testování mi byly poskytnuty jednotkové testy ověřující funkčnost aplikačního roz-

[1] https://www.doxygen.nl/index.html

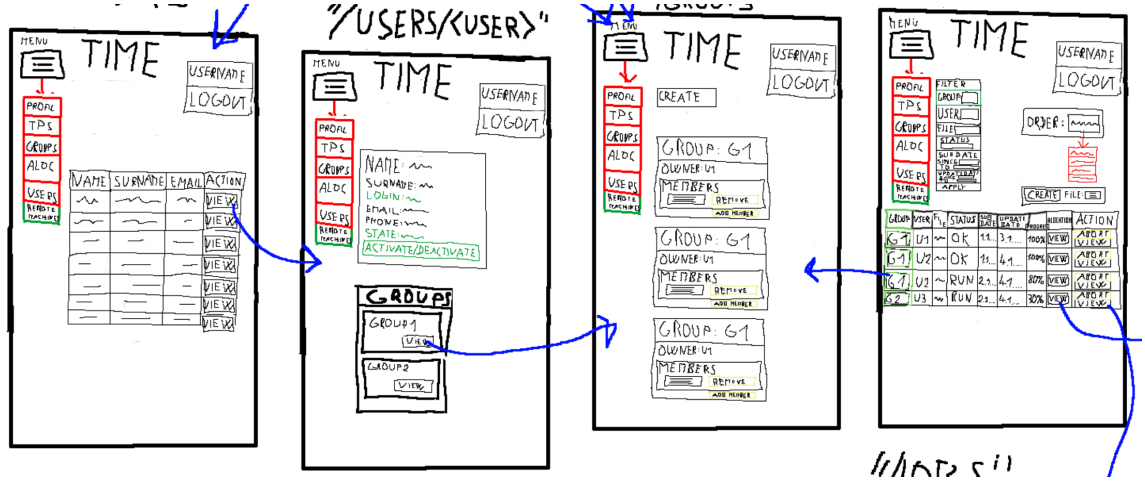
hraní implementované pomocí knihovny **unittest**. Testovací sada bude doplněna o testy kontrolující správnou funkci nově vytvořeného aplikačního rozhraní. Následně vytvořit testovací sadu ověřující grafické uživatelské rozhraní použitím technologie **Selenium**. Pro výsledné řešení vytvořit uživatelské scénáře, databázi s testovými daty a aplikaci na serveru spustit. Následně scénáře předat skupině lidí a řešení na základě zpětné vazby upravit.

4.2 Návrh vzhledu aplikace

Při procesu tvorby návrhu vzhledu byla nejdřív vytvořena skica. Ze skici se vytvořil detailnější návrh vzhledu a z detailnějšího návrhu vzhledu byl vytvořen **mockup**. Nakonec se z **mockupu** vytvořilo funkční řešení.

Skica

Pro účely skici projektu byla zvolena technologie **Gimp**. **Gimp** byl použit pro velmi hrubý nákres rozložení a struktury aplikace, aby bylo jasné, jaké stránky je vhodné vytvořit a co by na nich mohlo být. Návrh vycházel z popisu aplikačního rozhraní pro komunikaci modulů **k-Dispatch**. Byl vytvořen na čistě bílém pozadí za použití myši. Pod textem na obrázku 4.1 lze vidět část hrubého návrhu. Jedná se o část s navrženým vzhledem uživatelského profilu, stránky s uživateli, skupinami a léčebnými plány. Modré šipky naznačují přechody mezi stránkami.



Obrázek 4.1: Část skici pro stránku s uživateli, profilem, skupinami a léčebnými plány.

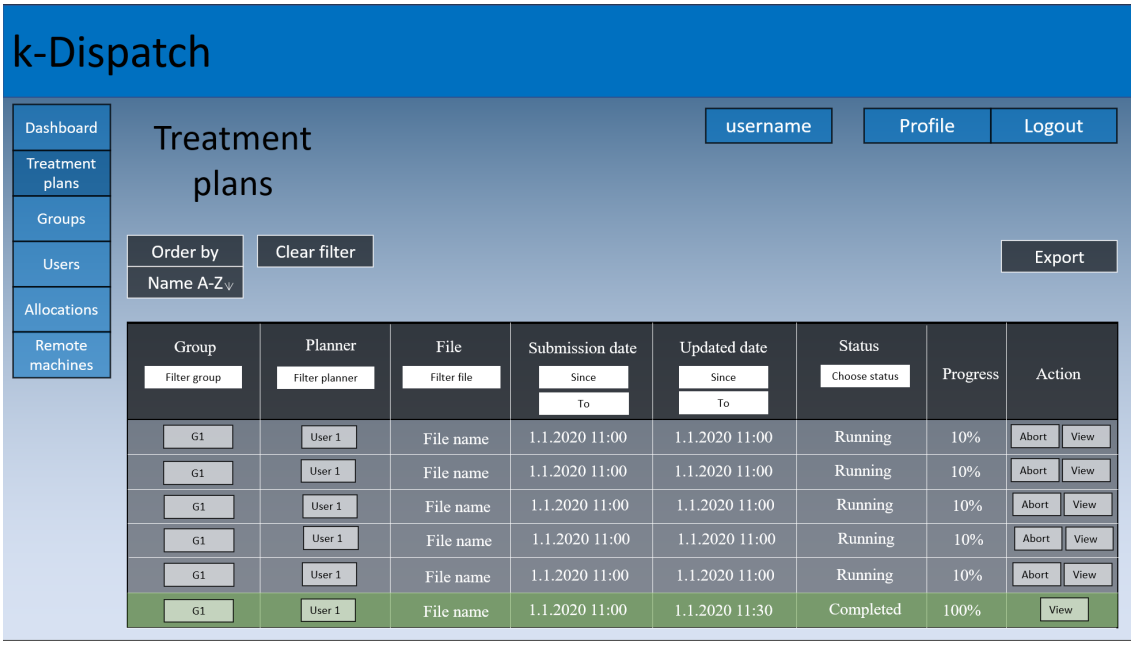
Detailní návrh

Z hrubého návrhu byl vytvořen přesnější návrh, na který bylo použito prostředí aplikace **PowerPoint**. **PowerPoint** byl použit, jelikož je projekt rozsáhlejšího charakteru, mám zkušenosti s touto aplikací a aplikace nabízí velké množství prvků, které lze k návrhu použít.

Jako základ jednotlivých návrhů stránek posloužily snímky duplikované z prvního snímku obsahujícího společné rozložení menu a pozadí. Stránky se tvořily na základě **k-Dispatch** rozhraní, ze kterého bylo přibližně jasné, co je zapotřebí na stránce zobrazit, hrubého ná-

vrhu a představě co by stránka mohla nabídnout. Pro zobrazení jednotlivých prvků byly použity grafické obrazce a textová pole.

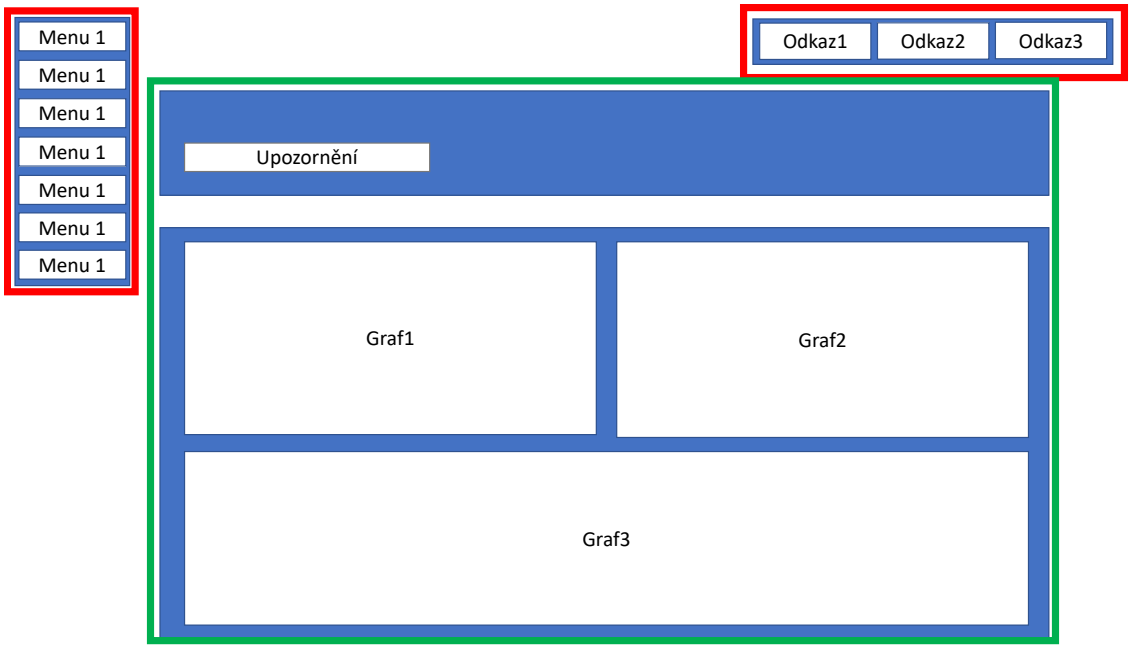
Na obrázku 4.2 lze vidět jednu z navržených stránek. Jedná se o návrh stránky s léčebnými plány. Léčebné plány jsou zobrazeny ve formě tabulky, protože je to relativně běžný způsob, jak zobrazit velké množství dat. Tabulka má ve většině sloupců v hlavičce filtr, který je spjatý s filtrovanými daty. Mimo jiné je možné záznamy řadit a to na základě určitého klíče. Pro export zobrazených léčebných plánů je na stránce stejnojmenné tlačítko. Léčebné plány jsou v tabulce barevně odlišeny na základě aktuálního stavu a mohou obsahovat odlišná tlačítka pro jejich správu. Zbytek návrhu lze vidět v příloze A.



Obrázek 4.2: Detailní návrh stránky s léčebnými plány.

Návrh nástěnky

Rozložení návrhu nástěnky se řídí principem rozložení obsahu na stránce. Je tedy v souladu s vizuálním vstřebáváním obsahu. Na obrázku 4.3 lze vidět rozložení stránky s nástěnkou. To, co je v červeném rámečku, je základ každé stránky až na stránku se změnou hesla a přihlášením. Základem stránky je menu s neměnnými odkazy na stránky umístěné v levém rohu. Dále je základem menu s uživatelsky specifickými odkazy na jeho osobní profil, vlastní skupinu a odkaz k odhlášení v pravém horním rohu. Červená část je rozložena podle vzoru písmene **F**. Specifický obsah každé stránky je v zeleném rámečku. V případě nástěnky je specifickým obsahem část s upozorněními a část s grafy. Při pozorném zkoumání rozložení je možné vidět rozložení obsahu nástěnky do tvaru písmene **E**. Díky tomuto typu rozložení uživatel lépe a rychleji vstřebá obsah. Dále byl použit princip seskupení. Logicky jsou upozornění umístěny do odlišného bloku, jak část s grafy. Část s grafy je též umístěna do jednoho bloku, ve kterém se sdružuje s ostatními grafy na základě uživatelské role.



Obrázek 4.3: Rozvržení stránky

Návrh obsahu nástěnky vychází z účelu webové aplikace, analýzy databáze, zaměření uživatelských rolí a konzultací s vedoucí práce. Při tvorbě návrhu bylo myšleno na to, že uživatelské role mají odlišné zaměření. Uživatel s rolí **DSM admin** potřebuje znát informace o výpočetních prostředcích, skupinách a alokacích. Z hlediska **výpočetních prostředků**:

- Poměr využití výpočetních prostředků. Poměr využití výpočetních prostředků administrátorovi napoví, na jaký výpočetní prostředek přesměrovat úlohy při přetížení nějakého z výpočetních prostředků. K jakým výpočetním prostředkům přiřadit nově domluvené alokace.
- Aktuální stav výpočetního prostředku. Kolik úloh je na výpočetním prostředku aktuálně spuštěno. Informuje administrátora o aktuálním využití výpočetních prostředků.
- Statistiky za časovou dobu. Kdy prostředek nebyl v činném stavu, kolik vypočítal úloh za určitou časovou dobu, využití výpočetního prostředku v hodinách a další.
- Chybovost výpočetních prostředků. Chybovost výpočetních prostředků upozorní administrátora na možný problém s výpočetním prostředkem. Chybovost je určena jako počet úloh určitého stavu v poměru k celkovému počtu úloh.
- Počet alokovaných hodin na jednotlivých prostředcích. Může pomoci při rozhodování k jakému výpočetnímu prostředku novou alokaci přiřadit.
- Souhrnné využití výpočetních prostředků. Souhrnným využitím je myšleno zobrazení úloh ze všech prostředků v rámci určitého časového intervalu. Souhrnné využití výpočetních prostředků lze lehce zobrazit v spojnicovém grafu. Z těchto informací lze predikovat budoucí vývoj zatížení systému.

Informování **DSM admin** ohledně **skupin**:

- Aktivita skupin. Napoví administrátorovi při hledání zdroje nadměrného využití služby.
- Rozdělení skupin. Upozorní administrátora na možné budoucí vytížení výpočetního prostředku při práci více skupin naráz. Vyplatí se znát z důvodu přiřazování alokací.
- Počet členů. Pro skupiny s větším množstvím členů obecně platí, že více zatíží výpočetní prostředky a webovou aplikaci.

Informování **DSM admin** vzhledem k **alokacím**:

- Datum expirace alokace. Datum expirace administrátora upozorní na ukončení spolupráce s nějakou skupinou a úkon domluvy další alokace.
- Plynulost čerpání alokace. Dokáže administrátora připravit na budoucí chování skupiny.
- Využití expirované alokace. Pomůže administrátorovi při domluvě dohody nové alokace. Sdělí administrátorovi doporučenou velikost nové alokace.

Z pohledu uživatele s rolí **Hospital admin** je dobré znát informace o členech skupiny a alokacích. Z hlediska **členů skupiny** je dobré vědět:

- Aktivita členů skupiny. Sdělí uživateli jak pracují členové skupiny počtem zadaných úloh systému v určitém časovém intervalu. Pro tento účel se hodí sloupcový graf, ve kterém lze hodnoty lehce vyčíst a porovnat. Úlohy lze v grafu rozdělit podle jejich aktuálního stavu, ale zhorší se tím jejich čitelnost.

Informování **Hospital admin** o **alokacích**:

- Datum expirace alokace. Upozorní na uplatnění zbývajících výpočetních hodin.
- Využití alokací. Kolik výpočetních hodin bylo uplatněno, kolik zbývá, kolik bylo využito mnou.

Uživatel s rolí **Planner** především nahrává léčebné plány k výpočtu na výpočetní prostředky a přitom čerpá ze skupinové alokace. Z hlediska uživatele **Planner** je dobrá znát o **léčebných plánech**:

- Aktuální stav posledních léčebných plánů. Uživatel nemusí zbytečně hledat v tabulce, ale při přihlášení se mu zobrazí aktuální stav úloh s odkazem na podúlohy.

Z hlediska **alokace**:

- Datum expirace alokace. Sdělí uživateli, do kdy může využívat výpočetních prostředků.
- Počet zbývajících hodin alokace. Slouží k informování uživatele při využívání alokací o nedostatku nebo dostatku výpočetních hodin.

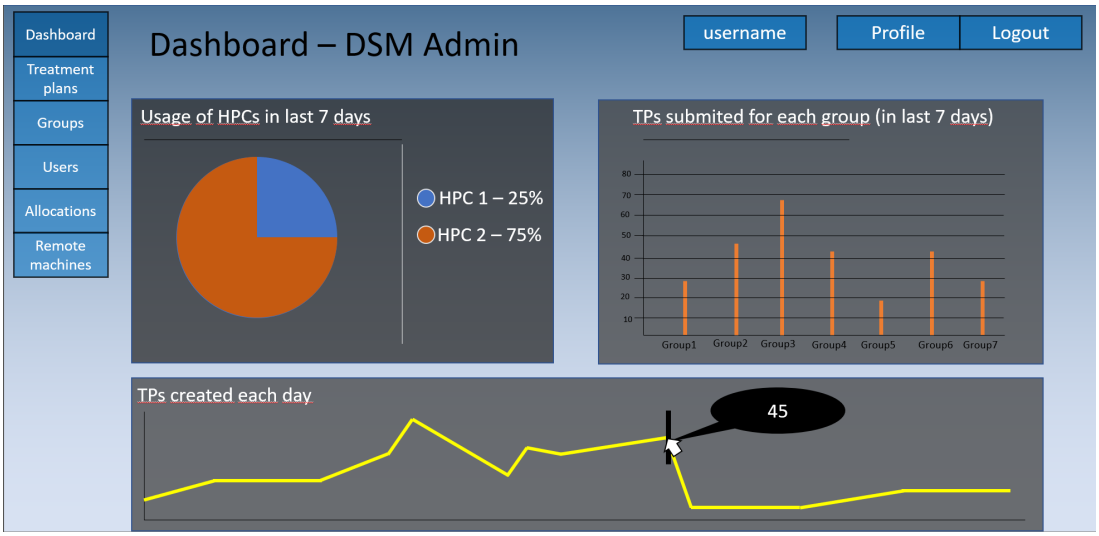
Uživateli **Reviewer** je dobré sdělit o **alokacích**:

- Datum expirace alokace. Může upozornit vedoucího skupiny na případné domluvení nové alokace a členy skupiny na vhodnou dobu pro plánování léčebných plánů.

Uživateli **Reviewer** je dobré sdělit o **výpočetních prostředcích**:

- Aktuální stav výpočetního prostředku. Pokud je nějaký z výpočetních prostředků skupinové alokace nedostupný, dojde k opoždění výpočtu. Sdělení může být ve formě upozornění.

Na základě výše uvedených bodů byly navrhnuty grafy. Návrh nástěnky se skládá ze 4 snímků, které jsou od sebe navzájem odlišné. Liší se zobrazenými grafy na základě role uživatele. Na obrázku 4.4 jsou zobrazeny některé grafy uživatele s rolí **DSM admin**. Grafy popisují počet vytvořených léčebných plánů za uživatelem určenou dobu, počet vytvořených léčebných plánů dle skupin, poměr využití výpočetních prostředků, využití jednotlivých výpočetních prostředků a statistický výpis uživatelů, skupin, alokací a výpočetních jednotek. Maximální podporovaný časový interval grafů je 365 dní. Grafy jsou umístěny záměrně na šedém pozadí, aby vynikl jejich obsah, který je oproti pozadí barevně pestrý.



Obrázek 4.4: Část návrhu nástěnky pro roli **DSM admin**.

Návrh nástěnky pro roli **Hospital admin** obsahuje upozornění na končící alokaci, graf využití alokace skupinou a graf s výčtem vytvořeným léčebných plánů jednotlivými členy skupiny. Návrh nástěnky uživatele **Planner** se skládá z grafu poměru aktuálních stavů léčebných plánů, výpisu posledních léčebných plánů společně s jejich stavem a graf s alokacemi. Poslední návrh se skládá pouze z grafu s alokacemi s jejich využitím. Většinu grafů je možné vidět na obrázku B.4 v příloze.

Graf pro popis počtu vytvořených léčebných plánů byl vytvořen s účelem zjistit, jak moc je nebo byl systém v určité době vytížen. Jedná se o prostorový graf. Obsahuje průměrné hodnoty vytvořených plánů, dokončených plánů, průměrnou hodnotu plánů s chybou, počty celkových plánů, chybových plánů a dokončených plánů pro jednotlivé dny spojené přímkou. Z grafu je možné vyčíst dny, případně období, ve kterém docházelo k chybám systému.

Graf počtu vytvořených léčebných plánů dle skupin je sloupcového typu. Slouží k určení skupin s největší zátěží výpočetních prostředků. Jednotky jsou léčebné plány a zobrazeny jsou všechny skupiny, které vytvořily v administrátorem určené době alespoň jeden léčebný plán. Sloupce skupin mají stejnou barvu, protože se především skupiny porovnávají mezi sebou. V potaz se neberou rozdíly mezi jednotlivými léčebnými plány.

Graf poměrů využití jednotlivých výpočetních prostředků je koláčového tvaru. Části koláče mají barvu na základě barevné palety. Každá část grafu představuje výpočetní prostředek. Jednotka grafu je hodina. Slouží především pro vzájemné porovnání výpočetních prostředků. Využitím může být nastavení zátěže nějakého vytíženého prostředku na méně vytížený prostředek. Případně může sloužit pro inspiraci při uzavírání dohody o alokaci.

Graf výpočetních prostředků je koláčového typu. Obsahuje počty podúloh rozdělených na základě jejich aktuálního stavu. Obsah je nemožné určit, proto je pro zobrazení použita paleta barev. Graf se generuje pro každý výpočetní prostředek, i když na něm nebyly spuštěny žádné podúlohy. Obsahuje totiž část se statistikami. Ve statistikách je zapsáno, kolik skupin má alokaci na daném prostředku, celkový počet alokovaných hodin na prostředku, počet zbývajících hodin alokací na prostředku a procento neúspěchu podúlohy.

Statistiky jsou častým prvkem nástěnky. Obsahují především základní informace o aktivních uživatelích, alokacích, skupinách, výpočetních prostředcích. Na tyto statistiky je možné kliknout a získat tak jejich výpis. Dále statistiky obsahují informace o nových skupinách, alokacích a léčebných plánech. Zobrazeny jsou pouze tyto prvky z důvodu databázového omezení. Statistiky se získávají filtrováním dat z databáze.

Upozornění na končící alokaci upozorňuje na základě data expirace. Informuje uživatele o počtu dnů do konce alokace. Upozornění se objeví při počtu 30 a méně dní do konce platnosti alokace.

K zjištění, jak pracují členové skupiny, je vytvořen graf s počtem vytvořených plánů jednotlivých členů skupiny. Jedná se o sloupcový graf. Sloupce mají stejnou barvu, jelikož slouží především k porovnání.

Graf poměrů aktuálních stavů léčebných plánů je koláčového tvaru. Pro odlišení stavů léčebných plánů je v grafu použita barevná paleta. Graf slouží uživateli především k informaci o aktuálním počtu léčebných plánů a jejich stavu.

Výpis posledních léčebných plánů obsahuje tabulku. Tabulka se skládá z řádku, ve kterých je uvedeno identifikační číslo léčebného plánu a aktuální stav léčebného plánu společně s odkazem na stránku s podúlohami.

Graf alokace je koláčového typu. Pro každou aktivní alokaci skupiny se generuje nový graf. Obsahuje využité výpočetních hodiny alokace, zbývající hodiny alokace a uživatelem využité alokace. Barvy částí jsou předem určeny na základě jejich příslušnosti. Využité hodiny jsou červenou barvou, využitelné hodiny zelenou barvou a uživatelovi hodiny barvou modrou. Slouží k informování o využití alokace

Mockup

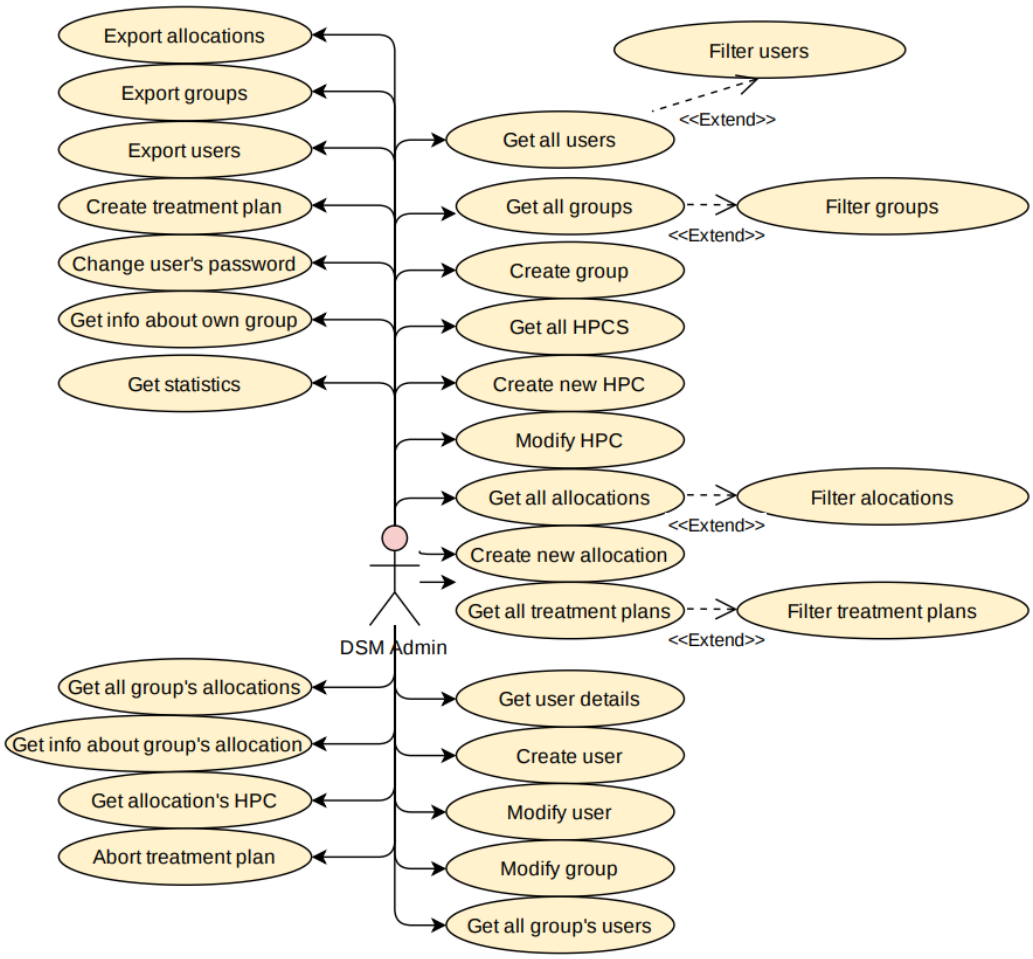
Mockup nebo také maketa, vychází z **Flask** šablony[2] webové aplikace upravované na základě předešlého návrhu. Šablona je předem připravené zobrazení, které slouží programátorovi jako základ k jeho vlastní tvorbě. Často obsahuje rozvržení základních objektů stránky, jako je například menu stránky, pozadí a styl. K úpravě stylu makety byla využita technologie **HTML** a **CSS**.

4.3 Diagram případů užití

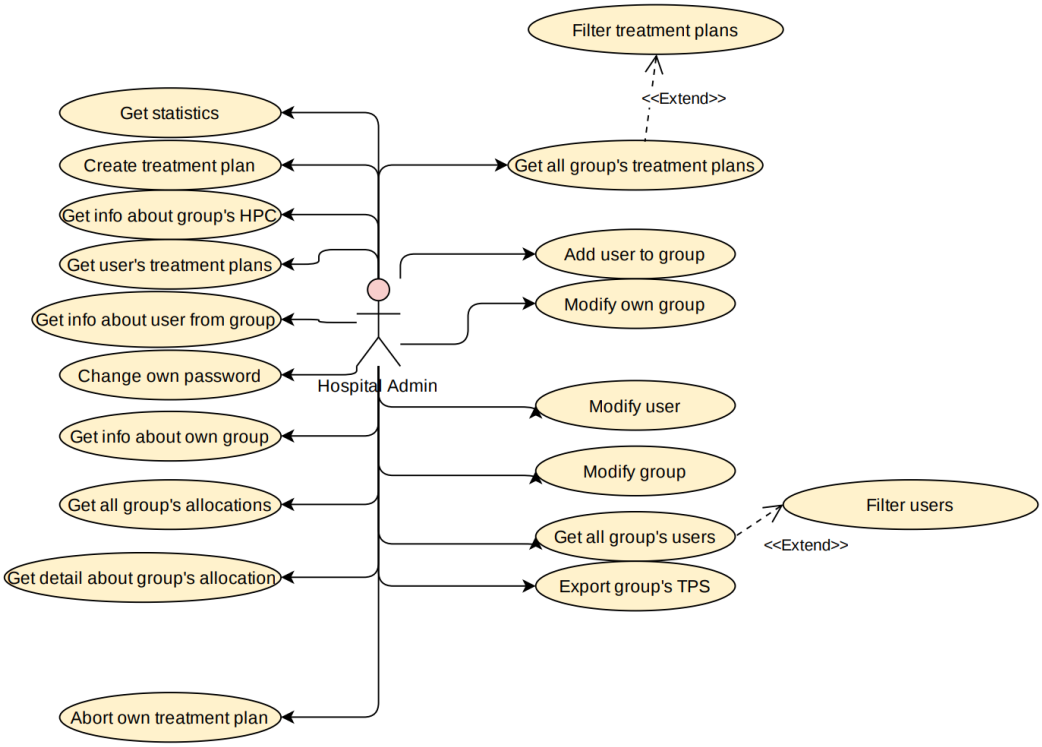
Diagram případů užití byl vytvořen na základě popisu komunikace přes aplikační rozhraní jednotlivých modulů služby **k-Dispatch**. Nejdříve byly zakresleny všechny známé systémové role. Role **Hospital admin** viz obrázek 4.6, **DSM admin** viz obrázek 4.5, **Planner**

[2] https://appseed.us/admin-dashboards/flask-dashboard-light-blue

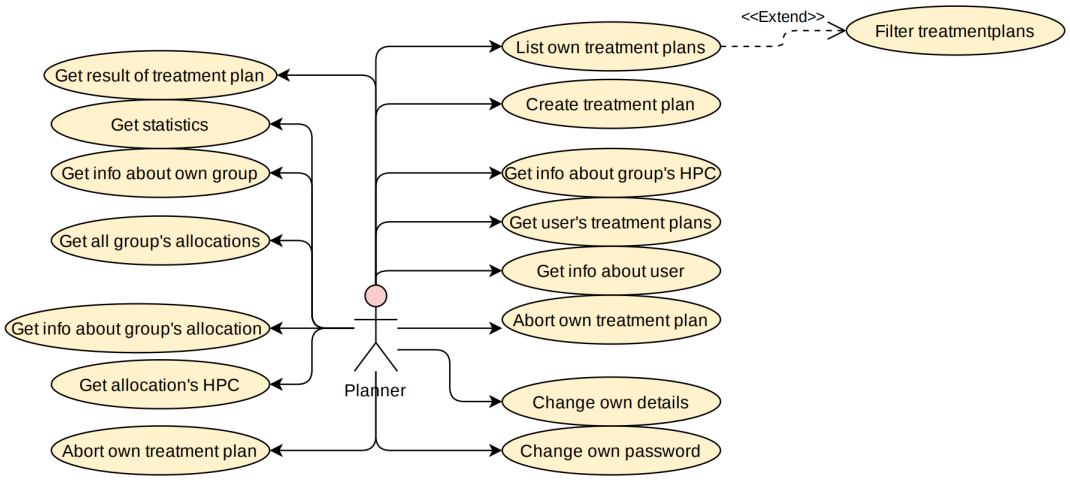
viz obrázek 4.7 a role **Reviewer** viz obrázek 4.8. Těm se přiřazovaly případy užití, které byly odvozeny z popisu rozhraní pro komunikaci mezi jednotlivými částmi systému. Nakonec se diagramy obohatily o nové případy užití, které by měla stránka podporovat. Mezi případy užití, které byly doplněny patří možnost záznamy exportovat, filtrovat a zobrazit si statistiky (nástěnka).



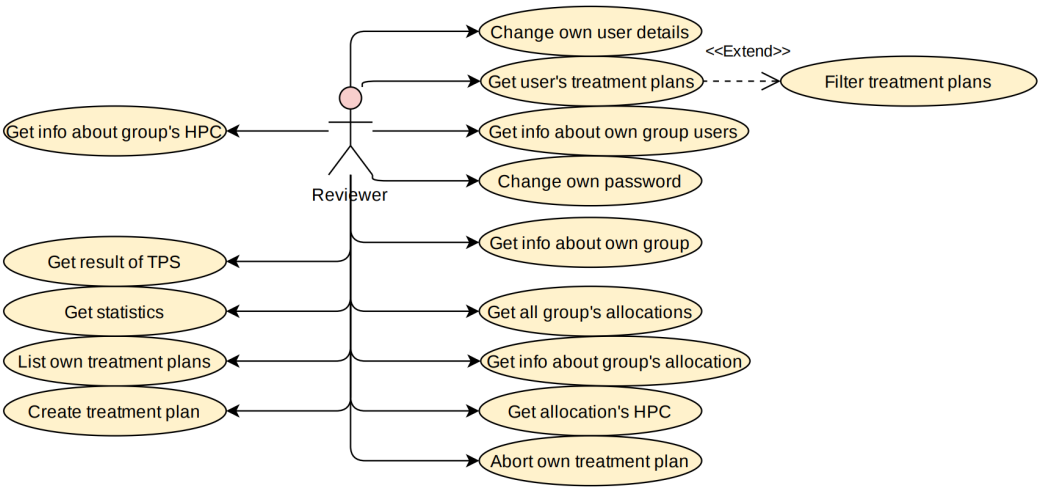
Obrázek 4.5: Diagram případů užití pro roli **DSM admin**.



Obrázek 4.6: Diagram případů užití pro roli **Hospital admin**.



Obrázek 4.7: Diagram případů užití pro roli **Planner**.



Obrázek 4.8: Diagram případů užití pro roli **Reviewer**.

4.4 Výběr technologií

Programovací jazyk a *framework* aplikace je dán ze započaté práce, která je v jazyce **Python** a využívá *framework* **Flask**. To samé platí u databáze, která má již vytvořené schéma s implementovanými modely s využívající knihovnu **Peewee**.

Pro vzhled stránek se použije knihovna **Jinja2**, která ze šablon a dat vytvoří finální strukturu stránky v **HTML** kódu. Stránky budou popsány v jazyce **HTML** za použití **CSS** a technologie **JavaScript**. Aby měl projekt nějaký základ **CSS** pravidel, tak se použije technologie **Bootstrap**. Pro zobrazení grafů na stránce nástěnky se využije knihovna **ChartJS**.

4.5 Návrh testování

Testovat se bude automatickými testy, manuálně a uživateli s předem daným scénářem. Automaticky se otestuje aplikační rozhraní a grafické uživatelské rozhraní. K testování aplikačního rozhraní se využijí již implementované testy doplněné o testy pro alokace. Tyto testy se budou spouštět při práci na implementaci a po dokončení práce. Pro implementaci testů se využije knihovna **unittest**. K otestování grafického uživatelského prostředí se vytvoří nové testy, které využijí knihovny **unittest** a **Selenium**. Testy se budou spouštět zároveň s testy pro aplikační rozhraní a pokryjí základní uživatelské operace s léčebnými plány, uživateli, výpočetními prostředky, alokacemi a skupinami. Na konci práce se vytvoří testovací scénáře, které se předají určité skupině uživatelů. Uživatelé po dokončení testování sdělí své poznatky. Na základě těchto poznatků se pak provedou změny.

Kapitola 5

Implementace

Kapitola obsahuje popis konkrétních technologií, použitých postupů a implementačních detailů. Kapitola je členěná do podkapitol, jejichž názvy odpovídají implementaci nějaké funkčnosti. Finální výsledek aplikace je možné vidět v příloze B.

5.1 Autorizace

Původní řešení

Původní návrh, ze kterého tato práce vychází, pro autorizaci používal moduly **http-auth** a **flask-login**. Moduly se používaly dva, jelikož je nutné autorizovat uživatele pro **REST api** a na přihlašovací stránce. Problém nastal při autorizaci přes přihlašovací stránku, přičemž se údaje uložily do *session*. Server chybně požadoval autentizaci i přes **HTTP** hlavičku, která nebyla nastavena, a tak vrátil návratový kód 401, což způsobilo, že prohlížeč po uživateli požadoval přihlašovací údaje znovu. Ty si následně prohlížeč uložil a poslal v **HTTP** hlavičce při každém dotazu, takže nešlo uživatele odhlásit.

Řešení

Nejdříve bylo nutné prostudovat možná řešení. Jedním z řešení bylo použít pouze modul **http-auth**, u něhož byl problém s odhlašováním. Prohlížeč si totiž odeslané autorizační údaje uschoval a posílal při každém požadavku. Problém tohoto řešení spočíval v tom, že nebylo možné údaje z webového prohlížeče smazat. Z toho důvodu se uživatel nemohl odhlásit. Dalším řešením bylo použití *tokenu* v *session*. Tohle řešení vyřeší předešlý problém s odhlašováním, jelikož k odhlášení stačí pouze smazat *token* ze *session*. Také je možné nastavit si dobu pro přihlášeného uživatele. Proto bylo zvoleno řešení s použitím *tokenu* uloženého v *session*.

Při zavedení řešení se vyskytl problém s použitím **REST api**, jelikož implementovaný systém pracoval pouze se *session*. Proto je při každém požadavku, ve kterém je nutno provést autorizaci, provedena kontrola na obsah **HTTP** hlavičky. Hlavička se kontroluje na existenci řádku s atributem **Authorization** obsahující uživatelské údaje. Pokud atribut nastaven je, tak se nejdříve zkontroluje, jestli uživatel se zadaným uživatelským jménem existuje. Pokud neexistuje, v odpovědi se nastaví stavový kód s hodnotou 401. V opačném případě se porovná uživatelem zadané heslo, které se zpracuje šifrovací funkcí a výsledek se porovná se zašifrovaným uživatelským heslem v databázi. Na základě výsledku porovnání

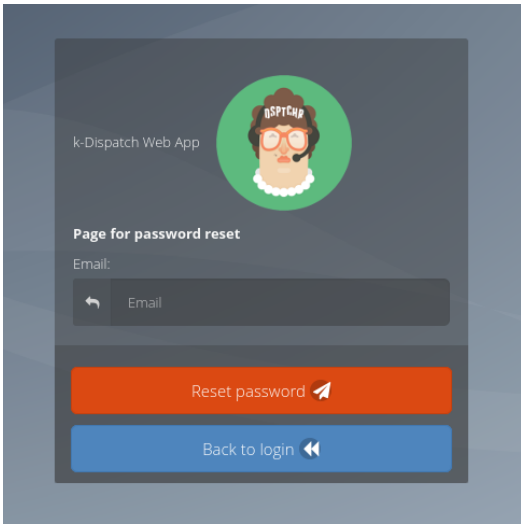
dojde buď k navrácení do kódu funkce označenou dekorátorem pro autorizaci, nebo se opět zašle odpověď se stavovým kódem 401.

Pro tvorbu *tokenů* je použit modul **jwt**, který zařizuje i jejich šifrování. Pro šifrování se používá protokol **HMAC-SHA256**. *Token* obsahuje dobu platnosti, uživatelské jméno, aby bylo jasné, kdo je přihlášen.

5.2 Zapomenuté heslo

Pro změnu zapomenutého hesla je vytvořena stránka viz obrázek 5.1. Odkaz pro změnu hesla se vyskytuje na stránce pro přihlášení. Po přesměrování na stránku se změnou hesla se uživateli zobrazí pole pro zadání *e-mail*, ke kterému se váže uživatelský účet. Jakmile uživatel zadá -mail a požadavek na změnu hesla odešle, tak je okamžitě přesměrován zpět na stránku pro přihlášení. Aby se zamezilo zjišťování výskytu uživatelského *e-mail* v databázi, tak se zpětná vazba po odeslání požadavku s *e-mail*, který je v databázi neliší.

Jakmile server požadavek přijme, pokusí se získat uživatele z databáze se zadaným *e-mail*. Pokud žádný nenalezne, vrátí odpověď s přesměrováním na přihlašovací stránku. Jestliže uživatele nalezne, vytvoří server na základě prefixu odkazu, uživatelského jména, aktuálního času a tajného klíče nový odkaz s klíčem platným dvacet čtyři hodin. Nakonec se vytvořený odkaz předá funkci navazující spojení s mailovým klientem, u kterého má server vytvořený *e-mail* a pomocí rozhraní knihovny **smtplib**[1] odešle odkaz na uživatelský *e-mail*. Poté, co uživatel dostane *e-mail* a zobrazí si získaný odkaz, zpracuje server požadavek na zobrazení stránky, kde z *url* získá klíč, který na základě tajného klíče rozšifruje, zkontroluje a pokud je klíč platný, tak zobrazí uživateli stránku se vstupem pro změnu hesla. Po zadání nového hesla a odeslání formuláře metodou **POST** na stejnou adresu server zpracuje požadavek. Nejdříve rozšifruje klíč a při nálezu hesla ve formuláři změní heslo a přesměruje uživatele na přihlašovací stránku.



Obrázek 5.1: Část stránky pro obnovu zapomenutého hesla.

[1] https://docs.python.org/3/library/smtplib.html

5.3 Zpracování požadavků

Pro aplikaci změn v systému se vytváří nové požadavky. K odeslání požadavku se používají **HTML** metody **GET** a **POST**, které ve většině případů odesílají formulář s obsahem na určitou *url* serveru. Server požadavek zkontroluje na obsah všech potřebných polí a informací. Server navíc zkontroluje uživatelská práva k výkonu požadavku. Pokud uživatel práva má, tak se požadavek zpracuje a vrátí se odpověď. Pokud uživatel práva nemá, tak server odpoví **HTML** kódem 403. Pokud dojde k chybě při zpracování požadavku, tak server odpoví **HTML** kódem 500.

5.4 Aplikační rozhraní pro alokace

Většina základního aplikačního rozhraní, která byla nutná vytvořit, již byla vytvořena před bakalářskou prací. Jediné, co zbývalo dokončit, bylo základní aplikační rozhraní pro práci s alokacemi. Rozhraní pro získání všech alokací z databáze, zisk pouze informací o určité alokaci, tvorba nové alokace a úprava nějaké z existujících alokací, v rámci které je zablokování a odblokování alokace.

Zpracování požadavků pro zisk všech alokací využívá metodu **GET**. Zadávající uživatel musí mít roli **DSM admin**, jinak server odmítne požadavek zpracovat. Uživatelovi se správnou rolí se vrátí obsah získaný modelem **GroupAllocationView** databázového pohledu ve formě **Python** slovníku, který se pošle v těle odpovědi. Pro zisk informací pouze o jedné alokaci je aplikační rozhraní implementovaná nad metodou **GET**. Výběr alokace spočívá na identifikátoru specifikovaném v *url*. Při kontrole přístupu se ověřuje uživatelská role **DSM admin** nebo příslušnost alokace k vlastní skupině. Data alokace se získají **Peewee** modelem **GroupAllocationView** a transformují do **Python** slovníku, který se pošle v těle odpovědi. K zpracování požadavku k vytvoření alokace se v programu vyskytuje funkce **addNewAllocation**. Funkce odpovídá pouze na požadavky **HTTP** metody **POST**. K vytvoření alokace musí mít uživatel roli **DSM admin**. Po ověření uživatelské role se přijatý formulář převede do **Python** slovníku, u kterého se zkontrolují názvy klíčů. Poté se pomocí **Peewee** abstrakce nad tabulkou s alokacemi zavolá metoda **create**. Metodě se předají data ze získaného slovníku. Vytvořená alokace se uloží a vrátí v obsahu těla odpovědi. Pro zpracování požadavku k úpravě informací o alokaci je v aplikaci implementována funkce **putAllocationId**. Požadavek na úpravu musí mít v *url* identifikační číslo alokace a musí být poslán **HTTP** metodou **PUT**. Pro úpravu alokace musí mít uživatel roli **DSM admin**. Po kontrole funkce transformuje a zkontroluje příchozí formulář, získá alokaci s požadovaným identifikátorem a snaží se změnit alokační údaje. Poté změny uloží a vrátí v těle odpovědi údaje o alokaci.

5.5 Nástěnka

Stránka slouží pro přehledné zobrazení a upozornění uživatelů o zdrojích, končících alokacích, výpadcích výpočetních prostředků a dalších informacích. Na stránku s nástěnkou má přístup každý uživatel, jelikož se jedná o výchozí bod po přihlášení uživatelů. Obsah je rozdělen na dvě hlavní části. Na blok s upozorněními a blok s grafy.

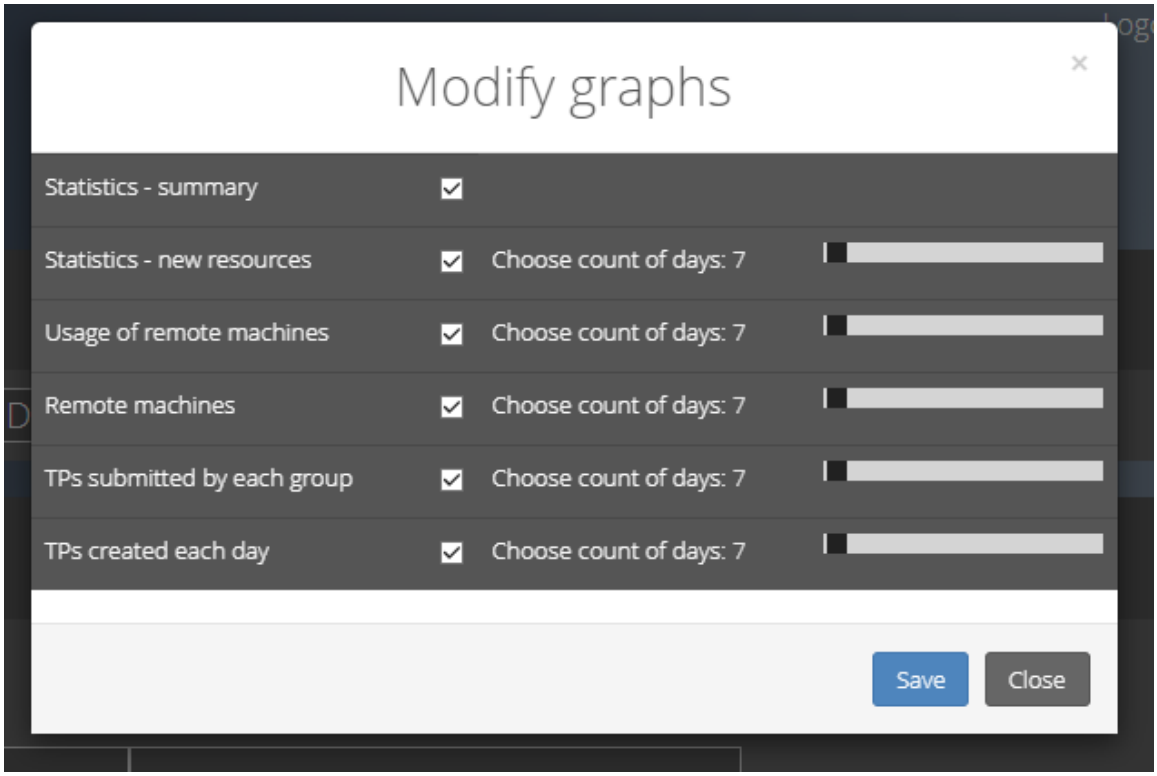
Část s upozorněními je zobrazena nad částí s grafy. Upozornění je umístěno nad grafy z důvodu menší velikosti bloku a větší důležitosti. V části s upozorněními se aktuálně mohou zobrazit dva typy zpráv. Prvním typem zpráv je upozornění na výpadek jakéhokoliv

výpočetního prostředku. Tato zpráva se zobrazí pouze uživatelům s rolí **DSM admin**. Dalším upozorněním, které se může zobrazit, je oznámení o končící alokaci. Toto upozornění se zobrazí uživatelům s rolí **Hospital admin, Planner** a **Reviewer** za předpokladu, že nějaká ze skupinových alokací má datum expirace kratší než třicet dnů. Zbytek upozornění zatím nebyl navrhnut k implementaci, ale je možné počet typů upozornění lehce rozšířit přidáním textu s upozorněním do **Python** pole se jménem **alerts**. Pokud pole neobsahuje žádné upozornění, objeví se namísto výpisu text **No alerts**.

Část s grafy je oproti části s upozorněními více složitější. Nejdříve je nutné získat velké množství dat, ty zpracovat a předat v příslušné struktuře. Aktuální implementace obsahuje pro zisk a zpracování dat funkce umístěné v souboru **graphs_for_index.py** z důvodu větší přehlednosti kódu. Většina funkcí, které slouží pro získání dat očekává mezi argumenty aktuálně používané filtry. Data se z databáze získávají metodami a pohledy implementovanými za použití modulu **Peewee**. Existuje více typů grafů zobrazující se na základě určité uživatelské role. Pokud má uživatel rolí více, sčítá se počet grafů s výjimkou stejného typu grafu zobrazující se pouze jednou. Jakmile jsou data pro specifické grafy získány, předají se funkci **render_template** očekávající mimo dat i šablonu, ze které vytvoří výsledné zobrazení stránky. O zobrazení stránky se stará jazyk **Jinja2** s kódem zapsaným v šabloně. Jazyk **Jinja2** postupně prochází podmínky zobrazení grafů a na základě nich vytvoří základ stránky. Všechny grafy jsou zobrazeny v jazyce **HTML** používající pro zobrazení prvek **canvas**. V těle prvku **canvas** se většinou nachází data specifická pro daný graf. O zobrazení grafu do prvku **canvas** se postará **JavaScript** knihovna **chartJS**. Knihovna nejdříve získá data z těla prvku **canvas** na základě specifické identifikace jednotlivých **canvas** prvků. Poté se data a jejich popisky uloží do **JavaScript** proměnných, které se předají do dat a popisků grafu.

Pro rozložení grafů v bloku s grafy jsou použity třídy technologie **Bootstrap**. Specificky se jedná o třídy **col**, **row** a **container**, které v bloku zaberou specifikovanou velikost, jenž se dynamicky mění s velikostí stránky. Každý řádek **row** se skládá z dvanácti částí **col**. Řádky jsou umístěny v bloku **container**, který se snaží zabrat co nejvíce místa na stránce a být umístěn v jejím horizontálním středu. Blok s upozorněními a blok s grafy jsou realizovány vlastním řádkem **row** a zabírají všech dvanáct částí řádku **col**. Jednotlivá upozornění se skládají za sebe s velikostí čtyř částí. Pokud dojde k přetečení maximální povolené velikosti dvanácti částí, tak se upozornění automaticky přesune na nový řádek. Grafy mají šířky násobku tří, což umožňuje pravidelnou skladbu grafů vedle sebe.

Uživatel má možnost přizpůsobit si své prostředí. Může si zvolit, jaké grafy si chce zobrazit a nastavit časovou osu grafů. Nastavení je možné po kliknutí na tlačítko **edit** v levém horním rohu. Po kliknutí na tlačítko se ztmaví pozadí stránky a objeví se nové okno s výběrem a nastavením grafů. To lze pozorovat na obrázku 5.2. K vytvoření nového okna je použita **Bootstrap** třída *modal*. Pro nastavení nových změn je potřeba stisknout tlačítko **save**, které změny odešle metodou **GET** v těle formuláře na stejnou *url*. Odpovědný server požadavek zpracuje a získá pouze data pro specifikované grafy. Nakonec, aby byly změny trvalé, se výběr grafů uloží do **cookies**.



Obrázek 5.2: Okno s výběrem a nastavením grafů specifického uživatele na stránce nástěnky.

Stránka obsahuje grafy typu:

- Koláčový graf zobrazující celkové zatížení výpočetních prostředků v hodinách.
- Koláčový graf zobrazující použití jednotlivých výpočetních prostředků k výpočtu úloh v hodinách.
- Plošný graf zobrazující počty vytvořených léčebných plánů s aktuálními stavy.
- Sloupcový graf zobrazující počet vytvořených léčebných plánů každé skupiny.
- Koláčový graf zobrazující využití alokací.

Část s grafy neobsahuje pouze grafy, ale i přehledy statistik. Mezi které patří:

- Souhrn statistik, který zobrazuje počet všech i aktivních uživatelů, skupin, výpočetních prostředků a alokací.
- Statistiky o nově vytvořených skupin, alokací a léčebných plánů.
- Zobrazení léčebných plánů, které byly naposledy vytvořeny.

5.6 Stránka s léčebnými plány

Stránka byla vytvořena na základě mírně upraveného návrhu. Ke změnám došlo kvůli technologickým důvodům. Základem stránky je pro většinu stránek stejná šablona, která obsahuje menu, odkaz na vlastní profil, tlačítko pro odhlášení a další funkční i nefunkční

prvky jako například pozadí stránky. Samotná stránka se pak skládá ze tří částí. První část tvoří nadpis, druhá zahrnuje tlačítko pro smazání filtru společně s tlačítkem obsahující možnost exportu záznamů a políčko pro výběr řazení záznamů. Poslední částí stránky je pak samotná tabulka s informacemi o léčebných plánech. Tabulka obsahuje informace dle role uživatele. Uživatelům s rolí **DSM Admin** se zobrazí tabulka se jménem skupiny, zadavatelem plánu, jménem souboru, ze kterého byl plán vytvořen, datum vytvoření plánu, aktuální stav, datum poslední aktualizace informací a část s možnými akcemi nad jednotlivými plány. Uživatelé s rolí **Hospital Admin** mohou vidět všechny léčebné plány vlastní skupiny. Uživateli s rolí **Planner** je umožněno zobrazit si pouze vlastní léčebné plány. Uživatel s rolí **Reviewer** si může zobrazit plány, u kterých je v databázi uveden jako kontrola. Poslední třem jmenovaným rolím se nezobrazí informace o skupině, protože si nemohou zobrazit léčebné plány ostatních skupin.

Řádky tabulky obsahující léčebné plány jsou na různých barevných podkladech vzhledem ke stavu léčebného plánu. Zde se barva pozadí vybírá na základě typického rozdělení. Léčebné plány končící úspěchem jsou na zeleném pozadí, plány končící chybou na červeném pozadí a zbytek na neutrálním bílém podkladu. Dále se řádky tabulky liší akcemi v posledním sloupečku. U řádků obsahující léčebné plány, jenž skončily bez chyby, je možné stáhnout výsledky. U aktivních řádků je na rozdíl od možnosti stáhnout výsledky tlačítko pro ukončení vykonávání léčebného plánu. Jinak je u všech řádků možné přejít na stránku s podúlohami zvoleného léčebného plánu.

V záhlaví tabulky se vyskytují jména sloupců společně s jejich filtry. Filtry se zasílají ve formě **HTML** formuláře požadavkem typu **GET** na *url* **/tps**. Filtr se aplikuje automaticky po zadání uživatelského vstupu, čímž se vytvoří událost změny stavu. Pro obsluhu události **HTML** značky existuje atribut **onchange**. V atributu se specifikuje **JavaScript** funkce pro automatické odeslání požadavku. Filtr se zašle aplikaci, která filtry zpracuje, podle nich vyfiltruje řádky získané z databáze a data z formuláře uloží do filtru zpět. Samotný filtr je v kódu reprezentován třídou dědící z nadtřídy modulu **flask-wtf**. Modul umožňuje vytvářet specifické části formuláře. Jejich ověření datových typů, nastavení vzhledu, pojmenování a další vlastnosti. Řádky není možné jen filtrovat, ale i řadit na základě zvolené možnosti. Možností je pět. Řádky je možné řadit abecedně podle sloupce se jménem vstupního souboru, podle data vytvoření léčebného plánu a nakonec podle stavu léčebného plánu.

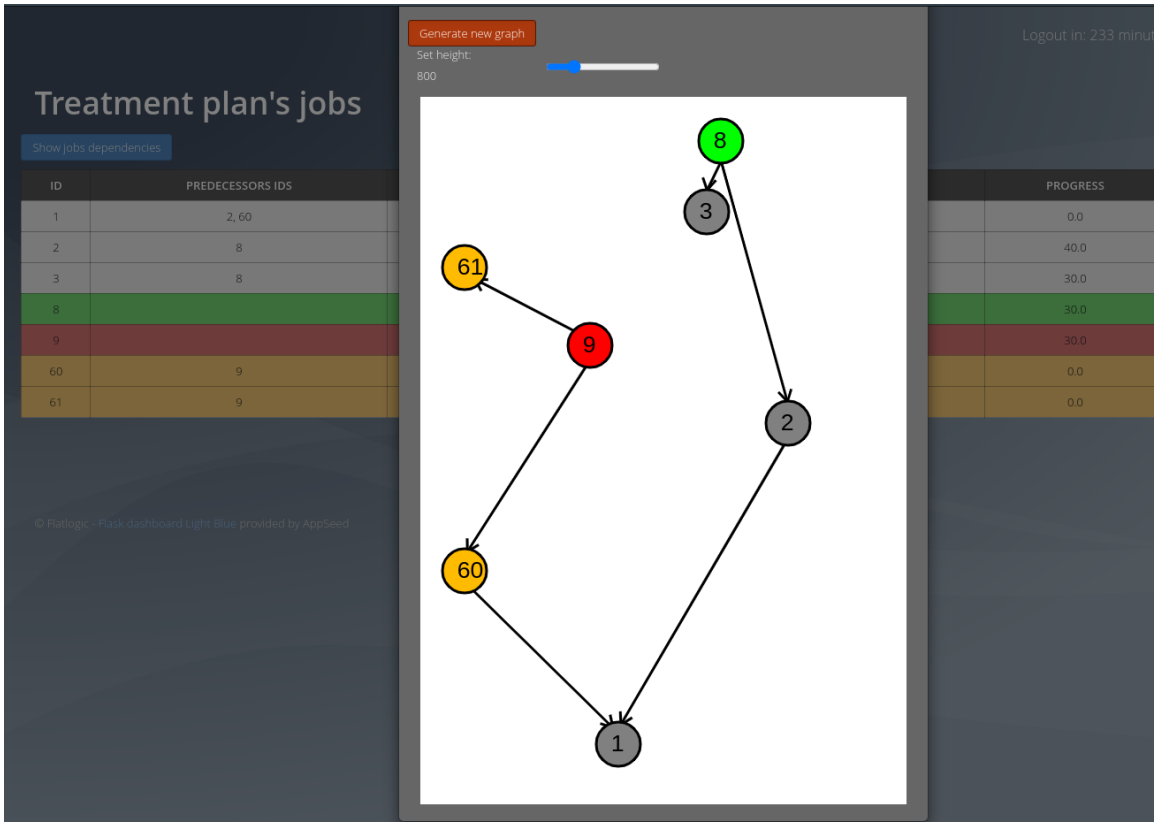
Zobrazení obsahuje tlačítko pro export dat aktuálně obsažených v tabulce. Export funguje tak, že se pošle metodou **GET** formulář na *url* specifickou pro export a v ní se specifikují argumenty filtru, které jsou aktuálně použity na zobrazená data. Filtry se stejně jako při zobrazení tabulky použijí pro filtraci dat a následně zapíší do souboru v **csv** formátu. Soubor se předá **Flask** funkci **send_file**, která soubor zpracuje a odešle v odpovědi serveru.

Při tvorbě stránky se vyskytly nečekané problémy. Jedním z problémů bylo vytvoření filtru pro filtraci stavu léčebného plánu. V základním řešení nebylo možné vybrat z více možností, protože to **HTML** a **CSS** nepodporují. Proto byla použita třída definovaná knihovnou **Bootstrap**. Třída umožňuje vybrat více možností. Bohužel možnosti nešly zaslat v těle formuláře. Proto byl vytvořen **JavaScript** kód a **HTML** skryté pole. Kód získá data a vloží je do **HTML** skrytého pole.

5.7 Stránka s podúlohami léčebných plánů

Pro každý léčebný plán existuje stránka s podúlohami. Při zpracovávání požadavku na zobrazení obsahu stránky se kontrolují stejné podmínky jako při zobrazení určitého léčebného

plánu. Stránka obsahuje tabulku s barevně odlišenými podúlohami na základě aktuálního stavu. Každá úloha má svůj identifikátor a může mít i předcházející úlohy. Proto se na stránce vyskytuje i tlačítko, které otevře nové okno s **HTML** plátnem zobrazující generovaný graf závislostí všech úloh vytvořený v jazyce **JavaScript**. To lze pozorovat na obrázku 5.3. Úlohy jsou v grafu barevně odlišeny. Pokud se v léčebném plánu vyskytuje velké množství úloh a graf je nepřehledný, je možné zvětšit pomoci **HTML** posunovače výšku plátna společně s možností znovu vygenerovat graf stisknutím tlačítka.



Obrázek 5.3: Plátno s grafem závislostí podúloh léčebného plánu.

5.8 Stránka se skupinami

Stránka se logicky dělí na dvě části. První částí zahrnuje funkční prvky, tedy tlačítka pro tvorbu nové skupiny, export dat a rozbalovacího menu pro filtraci dat. Druhá část obsahuje základní informace o skupinách a jejich členech. Stránka se skupinami je zpřístupněna všem uživatelům. Samotné zobrazení se liší na základě uživatelovy role. Pokud má uživatel roli **DSM admin**, má možnost vidět všechny skupiny, filtr, tlačítko pro export a tlačítko pro tvorbu skupiny. Jestliže má uživatel jakoukoliv jinou roli, zobrazí se pouze jeho vlastní skupina.

Data pro zobrazení se získají metodou **select**, která se zavolá nad databázových pohledem **GroupView**. Na získaná data se postupně aplikují filtry a podmínky metodou **where**. Po dokončení filtrace se nad získanými daty zavolá metoda **join**. Metodou dojde ke spojení přes identifikační číslo vlastníka skupiny, aby bylo možné zobrazit u skupiny namísto identifikátoru jeho jméno a příjmení. Dále se v kódu vytvoří **Python** slovník, kde

jsou položky indexovány skupinovým identifikačním číslem. Do vytvořeného slovníku se vloží všichni získaní členové skupiny. Získaná data se pak společně s vytvořenými formuláři předají funkci **render_template**, aby jazyk **Jinja2** mohl dynamicky vytvořit zobrazení skupin. Zobrazení tvoří tak, že postupně cyklí přes data skupin a na základě informací o skupině zobrazuje tlačítka, barvu pozadí a další prvky. U každé skupiny je další cyklus, který postupně prochází slovník se členy skupiny a vypisuje je do řádků tabulky s jejich rolí a odkazem na profil.

Rozbalovací menu pro filtraci dat je implementováno **Bootstrap** třídou **panel** a třídami souvisejícími s touto třídou. Menu pro filtraci obsahuje možnost filtrovat jméno skupiny a aktuální stav skupiny. Pro stav skupiny se využívá **HTML** prvek s výběrem jednoho stavu. Filtr se aplikuje zmáčknutím příslušného tlačítka odesílající formulář s daty filtru metodou **GET** na *url* aktuální stránky. Server pak požadavek přijme, přijatá data skupiny vyfiltruje a předá šabloně, kde jazyk **Jinja2** získaná data zformuje. Při exportu dat se aktuálně aplikovaný filtr přepošle metodou **GET** na specifickou *url* pro export dat, kde se znovu získají zobrazená data, která se pak přeformují do **csv** formátu a pošlou zpět klientovi ve formě souboru.

Pro vytvoření nové skupiny se na stránce vyskytne tlačítko. Po zmačknutí tlačítka se ztmaví stránka a zobrazí okno pro tvorbu nové skupiny. Okno obsahuje vstupy pro zadání potřebných informací o skupině například jméno skupiny. Po zadání informací se musí zmáčknout tlačítko pro vytvoření skupiny, které formulář odešle na určenou *url* metodou **POST**. Server požadavek zpracuje a po úspěšném vytvoření skupiny aktualizuje stránku. Jelikož je z databázových důvodů nutné nejdříve vytvořit skupinu a pak teprve přidat vlastníka skupiny, tak je možné, že některé skupiny nemají vlastníka. Proto se u skupin provádí kontrola na existenci vlastníka a u těch, které nemají vlastníka se vytvoří speciální tlačítko bez odkazu s textem nikdo. U každé skupiny, která má vlastníka je možné přejít na jeho profil pouhým kliknutím. Při vytváření stránky jazyk **Jinja2** vytvoří odkaz na základě uživatelova identifikátoru a **Flask** funkce **url_for**. Funkce vytvoří odkaz na základě jména funkce v programu a argumentu, jako je například uživatelovo identifikační číslo.

Při zobrazení skupin uživatel s rolí **DSM admin** má možnost deaktivace a aktivace skupiny. K deaktivaci skupiny je u každé skupiny zobrazeno tlačítko, které má oranžovou barvu pro zvýraznění funkčnosti. Deaktivovanou skupiny lze poznat z odlišné barvy bloku skupiny, která je světle červená. Pokud je skupina deaktivovaná, tak se namísto tlačítka pro deaktivaci objeví tlačítko pro aktivaci, které má tentokrát zelenou barvu pro zvýraznění odlišné funkčnosti. Deaktivace skupiny probíhá tak, že po stisknutí tlačítka se zobrazí dialogové okénko s upozorněním na deaktivaci skupiny. Upozornění vyžaduje potvrzení akce. Po potvrzení se pošle formulář pomocí **Ajaxu**[2]. **Ajax** je **JavaScript** technologie pro zasílání asynchronních požadavků. **Ajax** čeká na přijetí odpovědi a po jejím přijetí nebo určité době automaticky aktualizuje stránku.

U každé skupiny je vypsán seznam se členy skupiny, kde je zobrazeno celé jejich jméno a role, které uživatel má společně s odkazem na detail uživatele. Odkaz na detail uživatele je proveden stejně jako odkaz na vlastníka skupiny. Dále každý blok skupiny obsahuje odkaz na detailní informace o skupině.

[2] https://www.w3schools.com/js/js_ajax_intro.asp

5.9 Stránka s detaily skupiny

Na stránku s detailem skupiny se dá přejít za použití tlačítka v horní nabídce, které přesměruje uživatele na detail vlastní skupiny a nebo ze stránky se skupinami po kliknutí na tlačítko detail a podobně. Stránka s detailem skupiny obsahuje tři části. Jedna část slouží pro zobrazení všech alokací skupiny, další pro zobrazení a správu všech členů skupiny a poslední část slouží pro správu a zobrazení detailních informací o skupině.

Podle toho, jakou má uživatel roli a vztah ke skupině, tedy jestli je například vlastník, se liší jeho práva. Pokud má uživatel roli **DSM admin** nebo je vlastník skupiny, má možnost editovat informace o skupině, měnit role členů skupiny a vytvářet nové členy skupiny. Jestliže uživatel tuto roli nebo vztah nemá, nezobrazí se tlačítka pro změny a pole informací budou chráněna proti přepisu. Mimo jiné se při změně informací o skupině kontrolují role a vztah ke skupině. Tím se uživateli bez oprávnění znemožní upravovat informace, přidávat členy a měnit role členů.

Ke změně informací a zobrazení informací o skupině se na stránce využívá formulář, který je předem v **Python** kódu vytvořený a vyplněný aktuálními informacemi. Pro vytvoření formulářů se v kódu využívá modul **flask-WTF** [26]. Z něho se používá třída **FlaskForm** a různé typy vstupních polí. Z této třídy se následně odvodí třída **GroupDetails**. Z ní se vytváří nový objekt, u kterého se po získání dat z databáze nastaví hodnoty polí. Formulář je následně předán jazyku **Jinja2** pro přizpůsobení zobrazení formuláře na stránce. K odeslání formuláře slouží tlačítko uložit. Tlačítko odešle formulář na specifickou *url* metodou **POST**. Server data z formuláře transformuje do specifického slovníku a z něj pak upraví informace o skupině. Pokud dojde k úpravě informací bez chyb, tak dojde k přesměrování na stránku s detailem skupiny společně s **GET** argumentem v *url* obsahující zprávu, která se zobrazí v části s informacemi a informuje uživatele, že došlo k úspěšné úpravě informací.

Pro přidání nového člena skupiny musí být uživatel **DSM admin** nebo vlastník skupiny. Poté se mu zobrazí tlačítko umožňující přidat nového člena. Po kliknutí na tlačítko přidat nového člena se uživateli ztmaví stránka a objeví nové okno s formulářem pro tvorbu nového uživatele. Formulář obsahuje povinná a nepovinná pole. Povinná pole mají před jménem řádku žlutou hvězdičku. Pro vytvoření nového člena skupiny musí uživatel kliknout na tlačítko. Tlačítko spustí **JavaScript** funkci s **Ajax**. Funkce formulář odešle metodou **POST** na *url* aplikačního rozhraní pro tvorbu nového uživatele. Server přijatý požadavek s formulářem zpracuje, ověří odesílatele, přijaté hodnoty a pokud nedojde k chybě, vytvoří nového člena. Po přijetí odpovědi serveru funkce načte stránku.

Stránka umožňuje **DSM adminovi** a vlastníkovi skupiny změnit role členů skupiny. Pro změnu rolí člena skupiny musí uživatel kliknout na tlačítko u člena skupiny. Po kliknutí na tlačítko se stránka ztmaví a zobrazí se nové okno s nabídkou výběru rolí. Po potvrzení se zavolá jiná funkce, která formulář pošle metodou **POST** na *url* serveru pro změnu informací o uživateli. Funkce implicitně počká určenou dobu v řádu sekund a stránku znovu načte.

5.10 Stránka s uživateli

Stránka poskytuje možnost vytvořit nové uživatele, deaktivovat uživatele a zobrazit uživatele v tabulce. Zobrazené uživatele je možné filtrovat a exportovat do formátu **csv**. Na stránku má přístup pouze uživatel s rolí **DSM admin**, jelikož stránka obsahuje citlivá data.

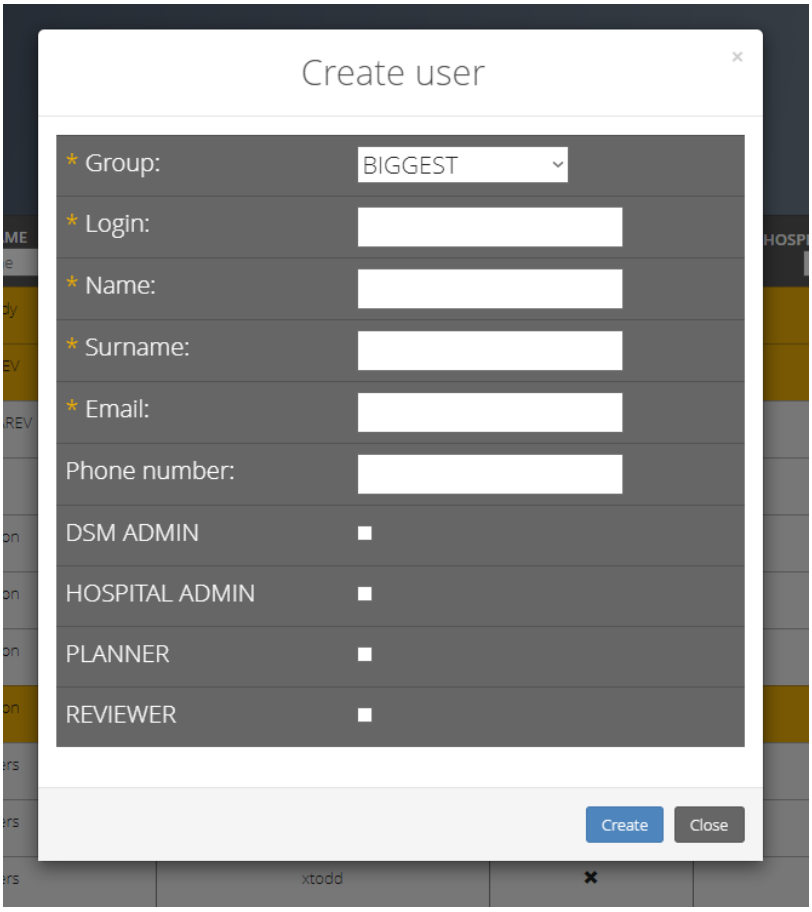
Tabulka s uživateli obsahuje filtry pro jednotlivé sloupce tabulky. V tabulce je možné filtrovat jméno skupiny, jméno uživatele, příjmení, *login*, role uživatele a aktuální stav uži-

vatele. Filtr se aplikuje automaticky dokončení zadávání filtru. Filtr se pošle ve formě formuláře metodou **GET** na aktuální *url*. Server po přijetí požadavku nejdříve zkontroluje práva tazatele, získá všechny uživatele z databáze a postupně aplikuje filtry nad získanými uživateli. Potom, co jsou filtry aplikovány, se tabulka s uživateli spojí s tabulkou skupin na základě skupinového identifikátoru. Díky tomu je možné u jednotlivých uživatelů zobrazit jméno skupiny.

Aktuální zobrazená data lze exportovat do formátu **csv** a to následovně. Uživatel klikne na tlačítko export. Po kliknutí na tlačítko export dojde k odeslání formuláře metodou **GET** s aktuálními filtry na *url* pro aplikační rozhraní serveru. Server požadavek zpracuje, získá všechny uživatele a aplikuje filtry. Výsledná data pak zapíše do souboru a ten následně odešle zpět uživateli.

Uživatele v tabulce je možné aktivovat a deaktivovat příslušným tlačítkem ve sloupci s akcemi nad uživatelem. Pokud není uživatel aktivní, řádek má světle červené pozadí a obsahuje tlačítko pro aktivaci. Bílé pozadí se objeví pro aktivní uživatele a obsahuje tlačítko pro deaktivaci uživatele. To jestli se zobrazí uživatel jako aktivní nebo neaktivní řídí jazyk **Jinja2**. Ten postupně prochází jednotlivé řádky ze získaných dat a podle informace o stavu aktuálního uživatele se rozhoduje, které ze zobrazení zvolí. Proces deaktivace uživatele probíhá tak, že při stisknutí tlačítka se vyvolá **JavaScript** kód. Kód odešle požadavek na stránku pro deaktivaci uživatele se specifickým identifikátorem uživatele doplněným během tvorby obsahu jazykem **Jinja2**. Server po přijetí požadavku zkontroluje práva uživatele s povinností mít roli **DSM admin**. Poté získá uživatele z databáze, nastaví stav na neaktivní a změny se pokusí uložit. Pokud se změny uloží, uživatel se přesměruje na stránku se všemi uživateli. Přesměrování **JavaScript** kód ignoruje a namísto toho stránku znovu načte. Při aktivaci uživatele se formulář odešle na jinou *url* s nastaveným novým stavem metodou **POST**. *Url* je určena pro změnu uživatelských údajů. Server opět zkontroluje práva uživatele a změní informaci o stavu uživatele. Po úspěšné změně je uživatel přesměrován na stránku s detaily o aktivovaném uživateli. **JavaScript** kód přesměrování opět ignoruje.

Tvorba uživatele je založena na **Bootstrap** třídě *modal*. Ta ztmaví stránku a zobrazí nové okno, ve kterém jsou pole s popisy k vyplnění stejně jako na obrázku 5.4. Nové okno pro tvorbu uživatele se objeví po zmáčknutí tlačítka vytvořit. Formulář se tím společně s vyplněnými poli odešle. Poté, co se formulář pošle metodou **POST** na aktuální *url*, server dostane požadavek na vytvoření nového uživatele. Formulář z proměnné **request.form** přeformátuje a pokusí se nového uživatele vytvořit. Pokud dojde k vytvoření nového uživatele, uživatel je přesměrován na stránku s detaily právě vytvořeného uživatele. Jelikož se formulář odesílá **JavaScript** kódem, tak dojde pouze k znovu načtení stránky.



Obrázek 5.4: Okno pro tvorbu uživatele využívající třídu *modal*.

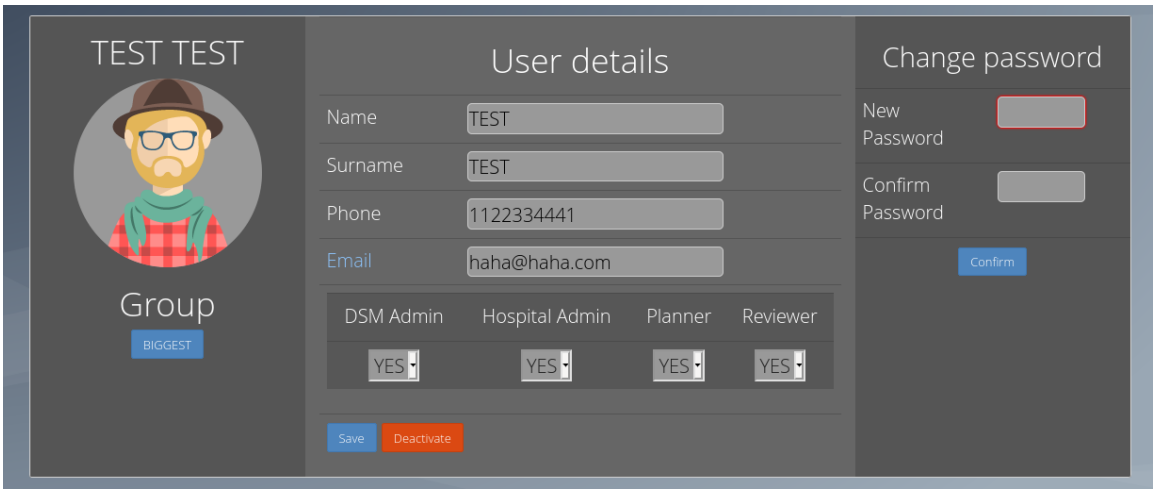
5.11 Stránka s detaily uživatele

Stránka s detaily uživatele slouží pro zobrazení informací o uživateli, ke správě uživatele a změně osobního hesla. Přístup na stránku s informacemi o uživateli má uživatel pouze tehdy, jestliže má roli **DSM admin** nebo stránka obsahuje informace o uživatelovi ze stejné skupiny, případně pokud se jedná vlastní profil zobrazený na obrázku 5.5.

Když na stránku přistupuje uživatel s rolí **DSM admin**, má možnost, na rozdíl od ostatních členů, měnit údaje o uživateli společně se zobrazenými rolemi. Ovšem nemá práva měnit uživatelské heslo. Pokud je stránka o přistupujícím uživateli, zobrazí se nejen možnost upravit osobní údaje bez rolí, ale i možnost změnit osobní heslo. Ostatní uživatelé mají pouze možnost zobrazit informace o uživateli.

Pro změnu informací o uživateli se uživateli zobrazí upravitelná pole vyplněná aktuálními informacemi. Pro uložení změn slouží tlačítko uložit zvýrazněné **Bootstrap** třídou **btn-primary**. Po zmáčknutí tlačítka se upravená pole pošlou ve formuláři metodou **POST** na *url* určenou pro změnu uživatelských údajů. Server po přijetí požadavku zkontroluje práva a vztah uživatele k požadavku. Pokud uživatel není dostatečně oprávněný, server přesměruje uživatele na stránku s chybou typu **Forbidden 403**. Jinak server data z formuláře zpracuje a změny uloží do databáze. Po úspěšné změně je uživatel přesměrován zpět na stránku s detaily společně se zprávou uloženou v argumentech *url*, která upozorní uživatele o změněných údajích.

Pokud uživatel chce změnit své heslo, musí zadat nové heslo, potvrdit ho a odeslat tlačítkem změnit. Tlačítko odešle formulář používaný pouze pro změnu hesla. Formulář se odešle metodou **POST** na *url* určenou pro změnu hesla. Server po přijetí požadavku zkontroluje přijaté heslo, jestli je nové heslo s novým potvrzeným stejné. Po úspěšné kontrole se změní uživatelské heslo a uživatel je přesměrován zpět na stránku s jeho detaily.



Obrázek 5.5: Vlastní profil uživatele se všemi rolemi.

5.12 Stránka s výpočetními prostředky

Stránka s výpočetními prostředky umožňuje přidávat nové výpočetní prostředky do databáze a zobrazovat výpočetní prostředky z databáze s možností je řadit a filtrovat. Vizuálně se stránka oproti předchozím stránkám neliší. Přístup na stránku mají pouze uživatelé s rolí **DSM admin**.

Pro přidání nového výpočetního prostředku musí uživatel kliknout na vyznačené tlačítko. To ztmaví pozadí a otevře nové okno s políčky pro vyplnění. Po odeslání dat server požadavek zpracuje a po bezchybném vytvoření nového serveru uživatele přesměruje na stránku se výpočetními prostředky.

Filtr a řazení funguje na stejném principu jako u ostatních stránek. Tedy odešle se formulář metodou **GET** na stejnou *url*. Ten server zpracuje a vrátí novou stránku s aktualizovanými daty.

5.13 Stránka s detaily výpočetního prostředku

Stránka s detaily výpočetního prostředku se skládá ze tří částí. Z části s informacemi, z části se všemi alokacemi, které se vztahují na daný prostředek a z části s grafem zobrazující využití výpočetního prostředku v posledních šesti týdnech. Stránka je přístupná pouze **DSM admin** a uživatelům vlastnící na daném výpočetním prostředku alokaci.

V části s detaily výpočetního prostředku je možné měnit informace. K tomu slouží předvyplněná pole s aktuálními daty v databázi. Ty je možné změnit kliknutím na tlačítko **uložit**. Tím se odešle formulář, server ho zpracuje a stránku aktualizuje. V další části jsou vypsány jednotlivé alokace získané z databázového pohledu **HpcAllocationView**. U každé jedné alokace je zobrazeno tlačítko odkazující uživatele na detail alokace. Poslední

část stránky obsahuje sloupcový graf zobrazující data o využití výpočetních prostředků v hodinách za posledních šest týdnů rozdělených do sloupců po týdnech.

5.14 Stránka s alokacemi

Stránka s alokacemi je vzhledově podobná předchozím stránkám, jelikož používá stejné třídy technologie **Bootstrap**. Přístup na stránku má každý uživatel. Zobrazení alokací se liší vzhledem k roli uživatele a jeho skupině. Pokud má uživatel roli **DSM admin**, zobrazí se mu všechny existující alokace. Jinak se uživatelům zobrazují pouze členské alokace skupiny.

Stránka umožňuje filtrovat záznamy zobrazené v tabulce. Filtrovat data je možné v každém sloupci. Pro aplikaci filtru je nutné zmáčknout tlačítko aplikovat filtr, které je zvýrazněné **Bootstrap** třídou **btn-primary**. Tlačítko způsobí odeslání formuláře metodou **GET** na stejnou *url*. Server po přijetí požadavku filtr aplikuje postupně, tedy prochází jednotlivé nastavení filtru a na základě přijatých dat řádky z databáze metodou **where** vyfiltruje. Data společně se šablonou předá funkci **render_template**, která vytvoří stránku na základě programu napsaného kódu v jazyce **Jinja2**. Výsledek se pak odešle uživateli ve formě webové stránky.

5.15 Stránka s detaily alokace

Stránka je přístupná na základě vlastnictví alokace. Pokud je uživatel členem skupiny patřící k alokaci, umožní se mu přístup. Jinak uživatel, který chce k alokaci přistoupit, musí mít roli **DSM admin**. V aktuální podobě slouží stránka pouze pro zobrazení informací bez možnosti měnit údaje alokace.

5.16 Nasazení na server

Z mnoha hledisek bylo nutné řešení zprovoznit na školním serveru **sc-gpu1**, kde se musela vytvořit nová databáze pro komunikaci s aplikací. Dále se musela zprovoznit samotná aplikace a služba zařizující komunikaci s počítači v jiné než privátní síti. Pro nasazení se použila technologie **Docker**[3] používající kontejnery, tedy předem připravené prostředí pro specifický účel. Pro nasazení se použily tři kontejnery vytvořené ještě před bakalářskou prací. Vzhledem k adresářové struktuře profilu na serveru byly již zmíněné kontejnery upraveny.

Po zprovoznění kontejnerů bylo možné nahrát testovací data do databáze. Data byla získaná příkazem **pg_dump**. Součástí balíčku je **PostgreSQL**, na kterém je databáze vytvořena. Data pak byla pomocí **scp** převedena na server a pomocí příkazu **pg_restore** v kontejneru nahrána do databáze.

[3] https://www.docker.com

Kapitola 6

Testování

Testování probíhalo během psaní kódu. Po každé dokončené fázi se spustily jednotkové testy pro ověření správně fungujícího aplikačního rozhraní. K testování došlo i na konci implementace. Testovalo se aplikační rozhraní, grafické uživatelské rozhraní a provádělo se testování na uživatelích.

6.1 Automatické testy

Pro automatické testování byly využity jednotkové testy. Testy jsou vytvořeny pro testování **REST api** a testování grafického uživatelského rozhraní webu. Většina jednotkových testů pro testování **REST api** byla implementována před bakalářskou prací. Testy pro kontrolu **REST api** byly obohaceny o nové jednotkové testy pro testování alokací. Testy pro testování uživatelského rozhraní používají kombinaci technologií **Selenium** a **unittest**.

Testování grafického uživatelského rozhraní

Automatické testy uživatelského rozhraní využívají kombinaci technologií **Selenium** a **Python** modulu **unittest**. **unittest** se stará o řízení testů, zatímco **Selenium** se využívá pro samotné testování grafického uživatelského rozhraní. Proces testování probíhá tak, že se spustí kolekce testovacích scénářů modulu **unittest**. Modul nejdříve nastaví prostředí pro jednotlivé případy tím, že nahraje testovací data do databáze, nastaví maximální dobu čekání na provedení akce, nastaví výchozí *url* prohlížeče, typ prohlížeče, který se použije. Prohlížeč se otevře buď s grafickým výstupem pro uživatele, a nebo se spustí v takzvaném *headless* módu. *Headless* mód je úprava, která slouží pro testování bez grafického výstupu. V módu *headless* je pak možné vytvářet pomocí funkce snímky prohlížeče, aby byl způsob jak by mohl tester manuálně ověřit výstup například stažením souboru na lokální počítač. Pak se postupně začne vykonávat testovací případ. V průběhu a na konci testovacího případu se můžou vyskytovat kontroly existence prvků na stránce, popřípadě aktuální *url* stránky. Po dokončení testovacího scénáře se vrátí databáze zpět do původního stavu a uzavře se prohlížeč.

Aktuální sada testů se zaměřuje na uživatele, výpočetní prostředky, skupiny a léčebné plány. U uživatelů se testuje přihlášení, vytváření nového uživatele, úprava údajů o uživateli, deaktivace uživatele, filtrování uživatelů na základě jména, skupiny, příjmení, *loginu* a role **DSM admin**. Testovací sada pro testování skupin obsahuje testovací případ pro deaktivaci skupiny, tvorbu nové skupiny a úpravu nějaké ze skupin. Léčebné plány se testují na zrušení a filtraci. Poslední sada testuje výpočetní prostředky na existenci stránky

s detaily, úpravu informací výpočetního prostředku, tvorbu nového prostředku a různé typy filtrování prostředků.

Testy využívající technologii **Selenium** byly mnohokrát přepisovány, protože se měnil vzhled stránek. Konkrétně jednotlivé třídy a identifikátory **HTML** prvků, na kterých jsou testy závislé pro jejich vyhledávání. Poslední větší úprava testů proběhla po ukončení uživatelského testování, jelikož byly provedeny změny na základě zpětné vazby.

Testování aplikačního rozhraní

Další implementované testy jsou pro ověření správného chování aplikačního rozhraní spravující alokace. Testy testují požadavek na vrácení všech alokací z databáze, vrácení informací pouze o specifické alokaci, úpravu informací alokace a tvorbu nové alokace.

Testování požadavku na vrácení všech alokací testuje bezchybný a chybný přístup k aplikačnímu rozhraní. Pro bezchybný průběh se využívá uživatel s rolí **DSM admin** posílající požadavek funkcí **getMethod** na specifickou *url*. Následně dojde k porovnání získaného **HTTP** kódu s očekávaným kódem. Dále se porovnává obsah těla zprávy, který musí být stejný jako očekávaný. Při vytváření očekávaného obsahu docházelo k chybě. Očekávané časy se neshodovaly se získanými, proto se teď nové časy získávají přímo z databáze. Po úspěšném dokončení bezchybného průběhu se testuje přístup k aplikačnímu rozhraní uživatelem bez vlastnictví role **DSM admin**. Očekávaný výstup **HTTP** kód 403 pro zprávu **Forbidden** je pak porovnán se získaným kódem.

Pro ověření aplikačního rozhraní navracející uživateli pouze specifickou alokaci se využívají dva odlišní uživatelé lišící se v uživatelských rolích. Nejdříve se testuje za pomocí uživatele s rolí **DSM admin** odesílající požadavek na specifickou *url* metodou **GET** funkcí **getMethod**. Získaná data a **HTTP** kód porovná s očekávanými hodnotami. Po úspěšném otestování přístupu prvního uživatel se testuje přístup druhého uživatele. Uživatel sice potřebnou roli **DSM admin** nemá, ale je členem skupiny patřící alokace. Průběh testu je pak stejný jako v prvním případě a to společně s očekávaným výstupem.

Testování tvorby nové alokace pomocí aplikačního rozhraní se testuje uživatelem s rolí **DSM admin** a uživatelem bez role **DSM admin**. Uživatel odešle požadavek metodou **POST** na specifickou *url* funkcí **postMethod**. Funkci se předají data alokace. Pokud má uživatel roli **DSM admin**, tak se kontroluje **HTTP** kód a obsah těla zprávy. V opačném případě se kontroluje pouze odepření přístupu uživateli, kterému se pouze sdělí **HTTP** kód **403**.

Aplikační rozhraní pro úpravu alokace se testuje jednotkovými testy kontrolující správné chování serveru na různé typy požadavků. Prvním požadavkem na server je požadavek zaslán metodou **PUT** na *url* pro aplikační rozhraní změny údajů alokace. Zde se specifikuje nové jméno alokace a pro autorizaci se použijí údaje uživatele s rolí **DSM admin**. Vytvoří se očekávaný obsah pro porovnání s příchozím obsahem a dojde k porovnání **HTTP** kódů. Očekává se kód **200** značící bezchybné zpracování serveru. Dále se porovná obsah odpovědi s očekávaným obsahem. V druhém požadavku se použije jiné nové jméno alokace a jiný uživatel pro zaslání požadavku. Průběh testování je stejný, ovšem na konci se neporovnává obsah zprávy, ale pouze **HTTP** kód **403**.

6.2 Uživatelské testování

První testování

Testování probíhalo hlavně ke kontrole intuitivnosti navrženého vzhledu a poskytované funkčnosti při práci s webovou aplikací. K testování byla vytvořena testovací databáze obsahující mimo jiné i uživatelské účty vytvořené pro práci s testy takové, aby nebyla zanedbána žádná z uživatelských rolí. Pro každou uživatelskou roli byl vytvořen formulář sloužící jako testovací plán skládající z otázek, uživatelských přístupových údajů a úkolů, které má uživatel vykonat. Formulářů bylo celkem pět. Průměrná doba vyplnění trvala kolem sedmi minut (odpozorováno). Testování se zúčastnilo pět lidí.

Na začátku každého formuláře byly obecné otázky na věk uživatele, jeho zkušenosti s programováním, oblíbený webový prohlížeč a dobu strávenou na počítači nebo mobilu v hodinách denně. Z odpovědí na formuláře bylo možné vyčíst, že uživatelé používají především webové prohlížeče typu **Google Chrome**[1] a **Mozilla Firefox**[2], tráví na počítači nebo mobilu kolem šesti hodin denně, mají buď malé nebo žádné znalosti v programování a jednalo se především o lidi s věkem mezi 18 až 26 lety (zbytek věk neudal). Z těchto údajů bylo vyvozeno, že se zmiňovanými prohlížeči nebyl problém. Jednalo se o mladší generaci uživatelů, která má poměrně dobré zkušenosti s webovými aplikacemi, obecně se více vyzná v internetovém prostředí a pravděpodobně nemají žádné omezující zrakové vady projevující se především u starších generací.

Vyhodnocení jednotlivých formulářů

V prvním formuláři, sloužícím pro otestování uživatelského prostředí s rolí **Reviewer**, byla testována orientace na stránce s léčebnými plány, jelikož to by mělo být hlavní náplní uživatele s touto rolí. Z odpovědí na formulář bylo vyčteno, že většině uživatelů se podařilo testování dokončit, ale s velkými výhradami. Nejvíce zmiňovanou výhradou bylo tlačítko pro aplikaci filtrů. Testerům vadilo, že se filtry neaplikují automaticky po zadání, ale že je nutné použít tlačítko pro filtr, které bylo nevýrazné. Proto byl předělán systém odesílání filtru. K odeslání není nově nutné použít tlačítko filtr, ale **JavaScript** aplikuje filtr okamžitě po dokončení editace filtru. Dále testeři měli problém se zrušením použitých filtrů, proto bylo přidáno namísto tlačítka s použitím filtru tlačítko pro smazání aktuálně používaného filtru.

Další formulář sloužil pro otestování prostředí uživatele s rolí **Planner**. V tomto případě byl testovací scénář navržen tak, aby více detailněji otestoval stránku s léčebnými plány. Většina uživatelů dokončila testované úkoly úspěšně se stejnými výtkami a jednou zcela odlišnou výtkou. Jednalo se o výtku u tlačítka pro export dat. Podle uživatelů by se mělo přejmenovat, a tak bylo tlačítko přejmenováno.

Ve třetím formuláři došlo k testování role **Hospital admin**. Testovaly se především úkony spojené se správou skupiny a přehledností nástěnky. S těmito úkony uživatelé neměli na základě zpětné vazby problémy a výtky byly především k možnosti zadat telefonní číslo, které se neskládalo pouze s čísel. Proto byla do aplikace přidána kontrola na testování obsahu telefonního čísla.

Předposlední dotazník byl zaměřen na testování uživatele s rolí **DSM admin**. Tento dotazník byl nejobsáhlejší a zároveň i nejsložitější na dokončení. Došlo k testování orientace na stránce se skupinami, uživateli a léčebnými plány. Paradoxem tohoto dotazníku bylo

[1] https://www.google.com/intl/cs_CZ/chrome/
[2] https://www.mozilla.org/cs/firefox/

to, že uživatelé většinu úloh provedli bez sebemenších problémů. Jediný problém, který měla většina uživatelů byl úkol najít úlohy léčebného plánu a do dotazníku zadat jejich počet. Tento problém byl vyhodnocen spíše jako selhání nedostatečného popisu systému v dotazníku, jelikož uživatelé neměli nejmenší ponětí, k čemu systém slouží a co poskytuje.

Poslední dotazník byl pro všechny role stejný, jelikož se jednalo o testování prostředí vlastního profilu a vlastní skupiny. Uživatelé si v testovacích scénářích vyzkoušeli upravovat informace o svém účtu na profilové stránce, společně s hledáním informací na stránce vlastního profilu a vlastní skupiny. S těmito scénáři neměli uživatelé na základě výsledků a zpětné vazby žádné problémy.

Druhé testování

Pro účely druhého testování byla upravena databáze z prvního běhu, která byla obohacena o nového uživatele se všemi uživatelskými rolemi. K testování byl vytvořen pouze jeden dotazník. Dotazník obsahoval otázky na použitý webový prohlížeč, bodové zhodnocení vzhledu stránek od 0 do 1 a zpětnou vazbu ve formě volně psaného textu.

Vyhodnocení druhého testování

Druhého testování se zúčastnilo sedmnáct uživatelů. Uživatelé nejdříve hodnotili vzhled stránky *Dashboard*, které dali průměrné hodnocení 0.7. Uživatelé převážně uváděli problém s titulkem stránky, který zasahuje do obsahu, příliš výrazné barvy grafů a ohraničení částí grafů. Dále uživatelé uváděli nestrukturovanost grafů do logických celků, což jim přišlo chaotické. Na rozdíl od toho se uživatelům líbila jednoduchost grafů, možnost grafy modifikovat, rychlost načítání stránky a překvapivě i přehlednost. Na základě toho byly změněny barvy grafů, smazané rámečky a titulek zkrácen pouze na jméno služby.

Další stránku, kterou měli uživatelé hodnotit byla stránka s léčebnými plány. Uživatelé dali stránce s léčebnými plány hodnocení 0.75. Reakce uživatelů na stránku byla převážně pozitivní s výjimkou pár uživatelů, kterým se nelíbilo chybějící ukládání filtrů při přechodu na jinou stránku, oddělený prvek pro řazení a chybějící upozornění při kliknutí na tlačítko *abort*. Naopak se skoro všem uživatelům líbilo filtrování a řazení dat. Z důvodu pozitivní zpětné vazby se design aplikace nijak neměnil. Pouze bylo přidáno dialogové okno, které slouží pro potvrzení přerušení výpočtu plánu.

Stránka se skupinami dostala od uživatelů hodnocení 0.76. Zavinila to především tabulka se členy skupiny, u které se uživatelům nelíbila část s posuvníkem, velikost tabulky, barva deaktivované skupiny a samotná existence tabulky. Uživatelé ocenili rozložení skupin, styl filtru, přehlednost stránky a nápadité zobrazení skupin. Proto byla změněna barva deaktivované skupiny, upraveno zobrazení části s posuvníkem a přidána kontrola deaktivace skupiny.

Nejlepší hodnocení dostala stránka s uživateli, které uživatelé dali 0.82. Uživatelům nejvíce vadilo přesměrování na stránku s detaily uživatele po jejich aktivaci a barva vyskakovacího okna pro tvorbu uživatele. Jinak uživatelé hodnotili stránku velice pozitivně. Na základě této zpětné vazby byla změněna barva okna pro tvorbu uživatele a upraveno přesměrování při aktivaci.

Uživatelé hodnotili stránku s výpočetními prostředky hodnocením 0.75. Zdůvodnili to rozložením a velikostí tabulky, řazením mimo tabulku, nemožností odstranění výpočetního prostředku a stejnou barvou pro dva různé stavy výpočetního prostředku. Uživatelé pozitivně hodnotili filtrování, řazení, přehlednost a správnou funkčnost. Změna se provedla pouze u velikosti a polohy tabulky.

Poslední stránka, kterou měli uživatelé hodnotit, je stránka s alokacemi. Stránka získala hodnocení 0.7. Uživatelům chyběla možnost záznamy řadit, vymazat aplikovaný filtr, chybějící tlačítko pro návrat z detailu a chybějící možnost přidat novou alokaci. Naopak bylo pozitivně hodnoceno filtrování záznamů, vzhled, přehlednost a jednoduchost. Bylo přidáno tlačítko pro vymazání filtru.

Kapitola 7

Závěr

Hlavním cílem této práce bylo vytvořit uživatelsky přívětivé funkční webové rozhraní sloužící ke správě a monitorování úloh na superpočítači. Následně ho otestovat jednotkovými testy. Výsledkem mé práce je webová aplikace podporující správu a monitorování systému **k-Dispatch**. Vzhled a funkčnost aplikace se přizpůsobuje vzhledem k uživatelským rolím. Aplikace umožňuje vytvářet pomocí grafického uživatelského rozhraní nové uživatele, skupiny, léčebné plány, výpočetní prostředky. Dále aplikace umožňuje zobrazit, exportovat, vyfiltrovat a seřadit uživatele, alokace, skupiny, léčebné plány společně s jejich úlohami a výpočetní prostředky v grafickém zobrazení, ze kterého by měly být snadno získatelné požadované informace. Aplikace neumožňuje pouze vytvářet nové zdroje, ale i upravovat již existující včetně deaktivace a aktivace. Hlavní částí výsledné aplikace je nástěnka upravitelná ve formě změny časové osy jednotlivých grafů a výběrů zobrazených grafů. Grafy obsahují pro aktuálního uživatele relevantní data jako je například využití jednotlivých prostředků za zvolené období společně se statistikami a přehledem. Dále byla aplikace rozšířená o aplikační rozhraní pro správu alokací. Mimo aplikaci bylo obohaceno i databázové schéma o pohled **run_job_view**, které slouží pro získání dat využitých v grafu na stránce s detaily jednotlivých prostředků.

Pro aplikaci byly vytvořeny automatické testy sloužící k ověření správné funkčnosti aplikačního rozhraní pro správu alokací a automatické testy pro ověření funkčnosti grafického uživatelského rozhraní. K testování grafického uživatelského rozhraní byla využita technologie **Selenium**. Nakonec se aplikace společně s databází zprovoznila na školním serveru **sc-gpu1**. K zprovoznění se využily kontejnery technologie **docker**. Aplikace na serveru byla využita k uživatelskému testování. Na základě vyhodnocení tohoto testování došlo ke změně převážně vzhledu, chování a funkčnosti.

Motivací k výběru práce byl především jazyk **Python**, který mě v minulosti velice zaujal. Proto jsem si ho chtěl podrobněji nastudovat a získat pro praxi potřebné zkušenosti, které by mi pomohly při hledání práce.

Na práci je možné navázat doplněním funkčnosti stránek alokací, protože v aktuálním stavu webové aplikace je možné pouze alokace filtrovat a zobrazit si u každé detaily. Dále rozšířit nástěnku o přehledy, grafy a další typy upozornění. Případně se více zaměřit na mobilní zařízení, protože aktuální verze aplikace je primárně vytvořena pro počítače.

Literatura

[1] *Anaconda Software Distribution.* Anaconda Inc., 2020. Dostupné z: https://docs.anaconda.com/.

[2] Aanderud, T. *Working on Your Dashboard Layout?* [online]. [cit. 2021-4-27]. Dostupné z: https://towardsdatascience.com/working-on-your-dashboard-layout-9b7c38d7b61e.

[3] Auth0. *Introduction to JSON Web Tokens* [online]. [cit. 2020-12-30]. Dostupné z: https://jwt.io/introduction.

[4] Bradford, L. *Python 2 or Python 3?* [online]. [cit. 2020-12-29]. Dostupné z: https://learntocodewith.me/programming/python/python-2-vs-python-3/.

[5] Carbonnelle, P. *PYPL PopularitY of Programming Language.* [online]. [cit. 2020-12-29]. Dostupné z: http://pypl.github.io/PYPL.html.

[6] Chacon, S. *Getting Started - About Version Control* [online]. [cit. 2021-4-21]. Dostupné z: https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control.

[7] community, C. *Chart.js* [online]. [cit. 2021-1-18]. Dostupné z: https://www.chartjs.org/docs/latest/.

[8] contributors, W. *History of Python.* [online]. [cit. 2020-12-29]. Dostupné z: https://en.wikipedia.org/wiki/History_of_Python.

[9] contributors, W. *HTML* [online]. [cit. 2020-12-30]. Dostupné z: https://en.wikipedia.org/wiki/HTML.

[10] developers, T. pip. *Pip 20.3.3* [online]. [cit. 2021-1-14]. Dostupné z: https://pypi.org/project/pip/.

[11] Docs, I. . c. Read the. *Flask-HTTPAuth* [online]. [cit. 2020-12-30]. Dostupné z: https://flask-httpauth.readthedocs.io/en/latest/.

[12] Docs, I. . c. Read the. *Flask-Login* [online]. [cit. 2020-12-30]. Dostupné z: https://flask-login.readthedocs.io/en/latest/.

[13] Fincher, J. *Python IDEs and Code Editors* [online]. [cit. 2021-1-13]. Dostupné z: https://realpython.com/python-ides-code-editors-guide/.

[14] Foundation, D. S. *Django* [online]. 2013 [cit. 2021-5-02]. Dostupné z: https://djangoproject.com.

[15] Foundation, P. S. *Unit testing framework* [online]. [cit. 2021-4-10]. Dostupné z: https://docs.python.org/3/library/unittest.html.

[16] Grinberg, M. *Flask web development: developing web applications with python*. "O'Reilly Media, Inc.", 2018.

[17] Group, T. P. G. D. *PostgreSQL: The World's Most Advanced Open Source Relational Database* [online]. [cit. 2021-4-28]. Dostupné z: https://www.postgresql.org.

[18] Guru99. *Web Application Testing: 8 Step Guide to Website Testing* [online]. [cit. 2021-4-13]. Dostupné z: https://www.guru99.com/web-application-testing.html.

[19] Guru99. *What is Selenium? Introduction to Selenium Automation Testing* [online]. [cit. 2021-4-9]. Dostupné z: https://www.guru99.com/introduction-to-selenium.html.

[20] Herman, M. *Django vs. Flask in 2020: Which Framework to Chooseu* [online]. [cit. 2020-12-30]. Dostupné z: https://testdriven.io/blog/django-vs-flask/.

[21] Hoyos, M. *What is an ORM and Why You Should Use it* [online]. [cit. 2020-12-30]. Dostupné z: https://blog.bitsrc.io/what-is-an-orm-and-why-you-should-use-it-b2b6f75f5e2a.

[22] Hynek, J. *Vizualizace a vizuální vnímán* [Private online document]. Božetěchova 1/2, 612 00 Brno-Královo Pole: [b.n.], 2020.

[23] Inc., F. *Overview of the React documentation and related resources* [online]. [cit. 2020-12-30]. Dostupné z: https://reactjs.org/docs/getting-started.html.

[24] Inc., P. T. *Plotly JavaScript Open Source Graphing Library* [online]. Plotly Technologies Inc., 2015 [cit. 2021-1-18]. Dostupné z: https://plotly.com/javascript/.

[25] Inc., P. T. *Plotly.js* [online]. Montreal, QC: Plotly Technologies Inc., 2015 [cit. 2021-1-18]. Dostupné z: https://github.com/plotly/plotly.js/.

[26] Jacob, D. *FlaskWTF* [online]. [cit. 2021-3-29]. Dostupné z: https://flask-wtf.readthedocs.io/en/stable/.

[27] Jaroš, M. a Jaroš, J. *Framework for Planning, Executing and Monitoring Cooperating Computations* [online]. [cit. 2021-4-28]. Dostupné z: https://www.fit.vut.cz/research/publication/11782/.

[28] Jaroš, M., Treeby, E. B., Jaroš, J. a Georgiou, P. K-Dispatch: A Workflow Management System for the Automated Execution of Biomedical Ultrasound Simulations on Remote Computing Resources. In: *Proceedings of the Platform for Advanced Scientific Computing Conference, PASC 2020*. Association for Computing Machinery, 2020, s. 1–10. DOI: 10.1145/3394277.3401854. ISBN 978-1-4503-7993-9. Dostupné z: https://www.fit.vut.cz/research/publication/12125.

[29] Jaroš, J., Treeby, B. E., Budisky, J. a Vaverka, F. *SC@FIT Handbook*. University of Technology, 2019.

[30] JetBrains. *PyCharm* [online]. [cit. 2020-12-30]. Dostupné z: https://www.jetbrains.com/pycharm/.

[31] Johnston, J. *A beginners guide to web application development* [online]. [cit. 2021-4-13]. Dostupné z: https://www.budibase.com/blog/web-application-development/.

[32] leifer charles. *Peewee docs* [online]. [cit. 2020-12-29]. Dostupné z: http://docs.peewee-orm.com/en/latest/.

[33] Pallets. *Templates* [online]. [cit. 2020-12-30]. Dostupné z: https://flask.palletsprojects.com/en/1.1.x/templating/.

[34] Pernice, K. *F-Shaped Pattern of Reading on the Web* [online]. [cit. 2020-12-30]. Dostupné z: https://www.nngroup.com/articles/f-shaped-pattern-reading-web-content/.

[35] Republic, T. *CSS Tutorial* [online]. [cit. 2020-12-30]. Dostupné z: https://www.tutorialrepublic.com/css-tutorial/.

[36] Republic, T. *Javascript Tutorial* [online]. [cit. 2020-12-30]. Dostupné z: https://www.tutorialrepublic.com/javascript-tutorial/.

[37] Rossum, G. van a Drake, F. L. *History of the software* [online]. [cit. 2020-12-29]. Dostupné z: https://docs.python.org/2.0/ref/node92.html.

[38] Spencer, J. *12 Free Mockup and Wireframing Tools for Web Designers* [online]. [cit. 2020-12-30]. Dostupné z: https://makeawebsitehub.com/mockup-and-wireframe-tools/.

[39] team, B. *Introduction to Bootstrap* [online]. [cit. 2020-12-30]. Dostupné z: https://getbootstrap.com/docs/5.0/getting-started/introduction/.

[40] Van Rossum, G. a Drake, F. L. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN 1441412697.

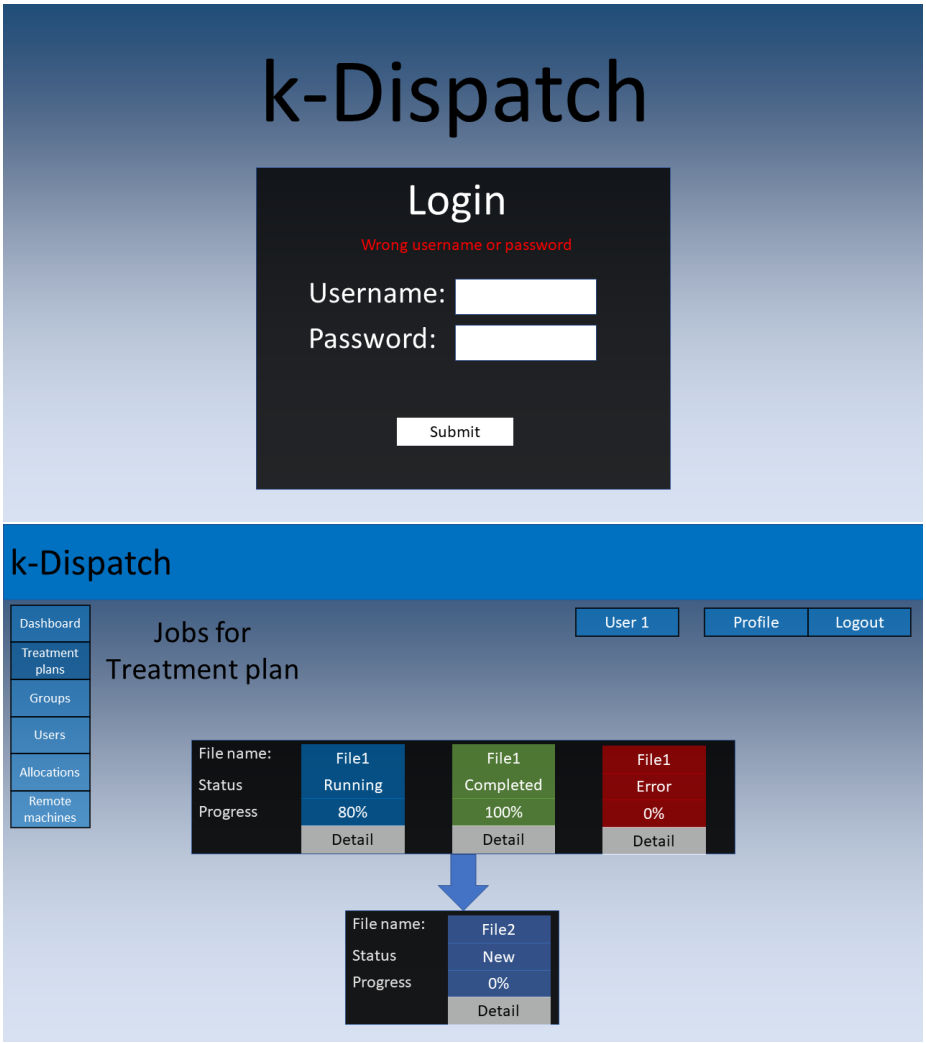
[41] You, E. *What is Vue.js?* [online]. [cit. 2020-12-30]. Dostupné z: https://vuejs.org/v2/guide/.

Příloha A

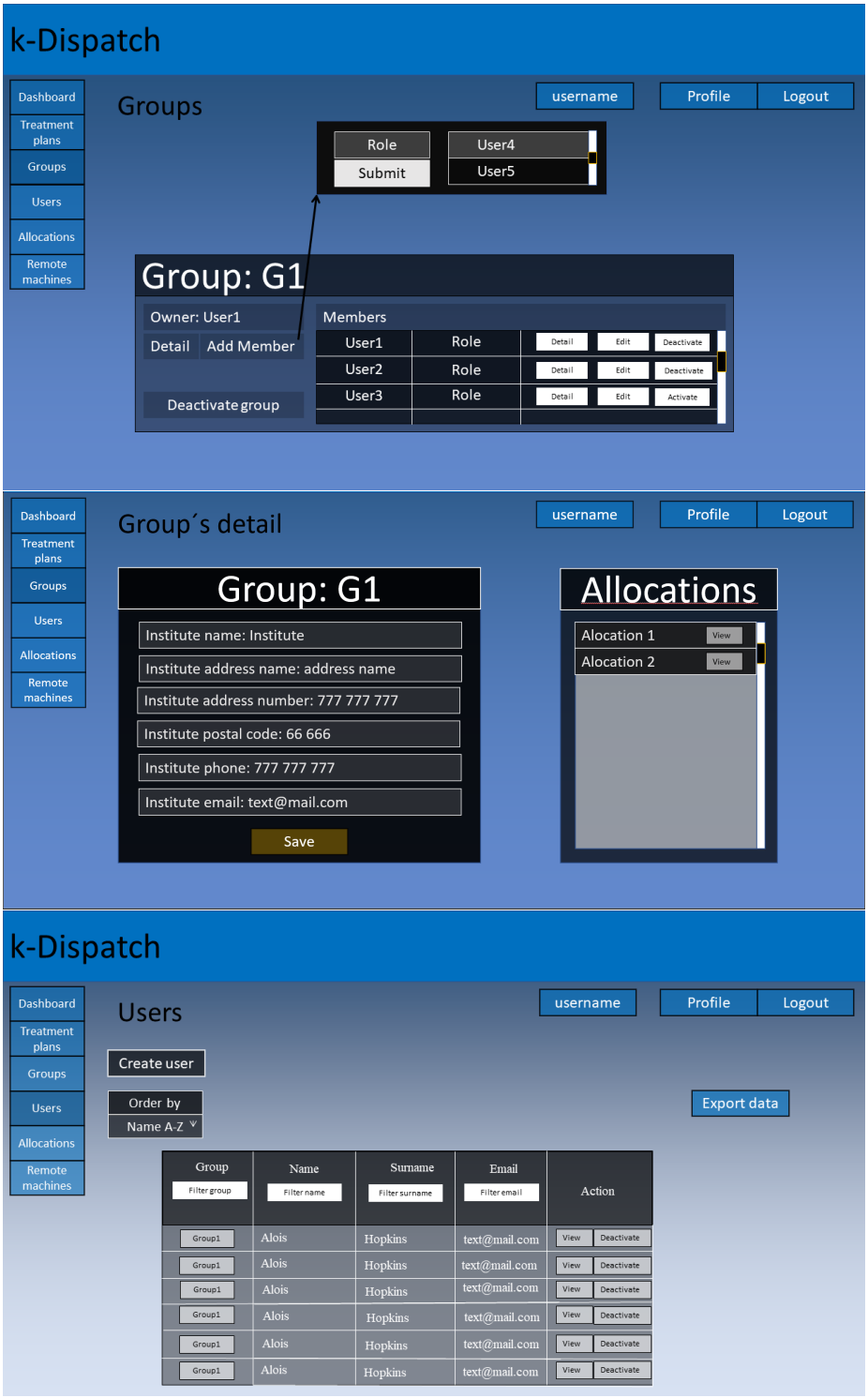
Detailní návrh

Template parts description

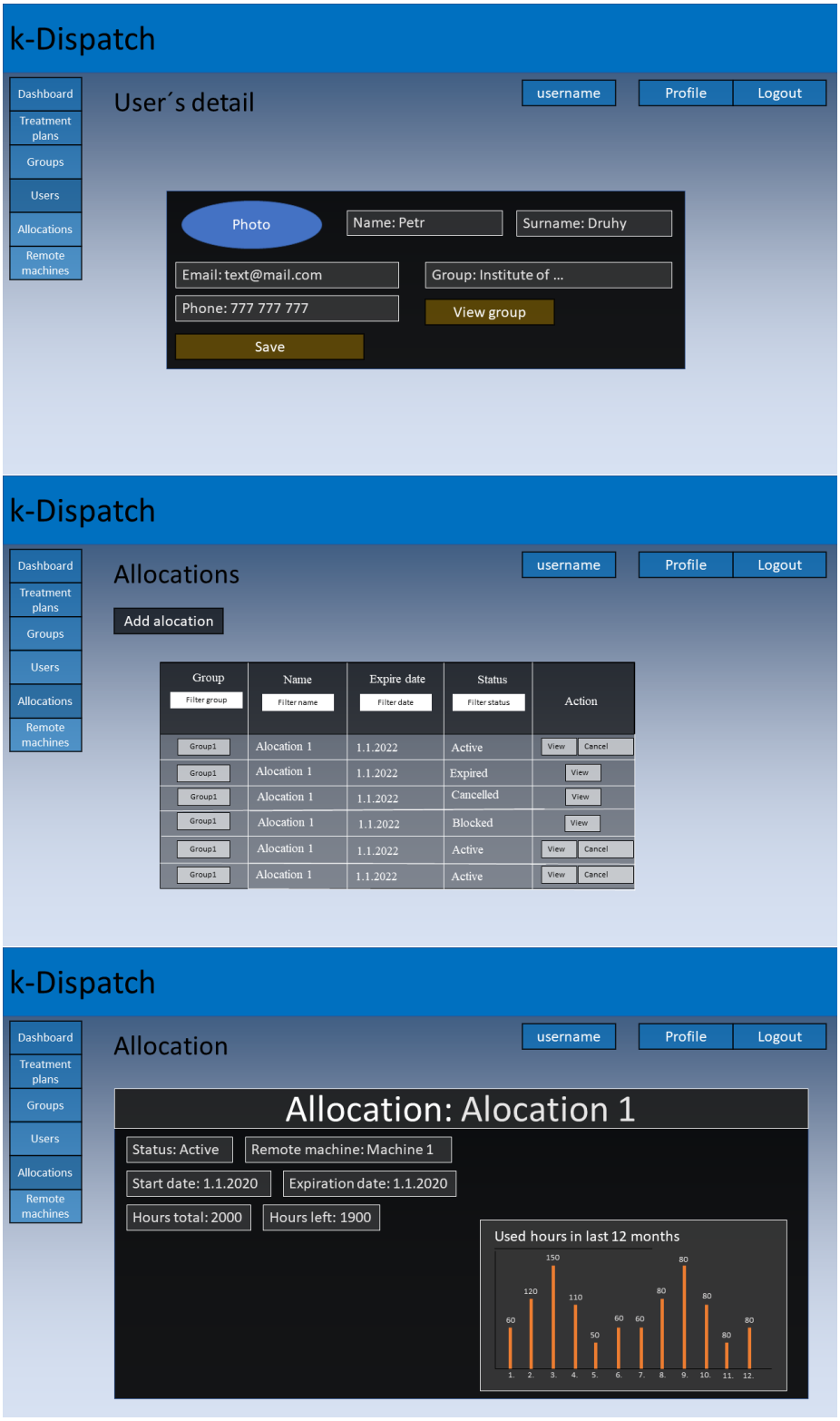
V této příloze je zobrazený detailní návrh vzhledu popsaný v návrhu.



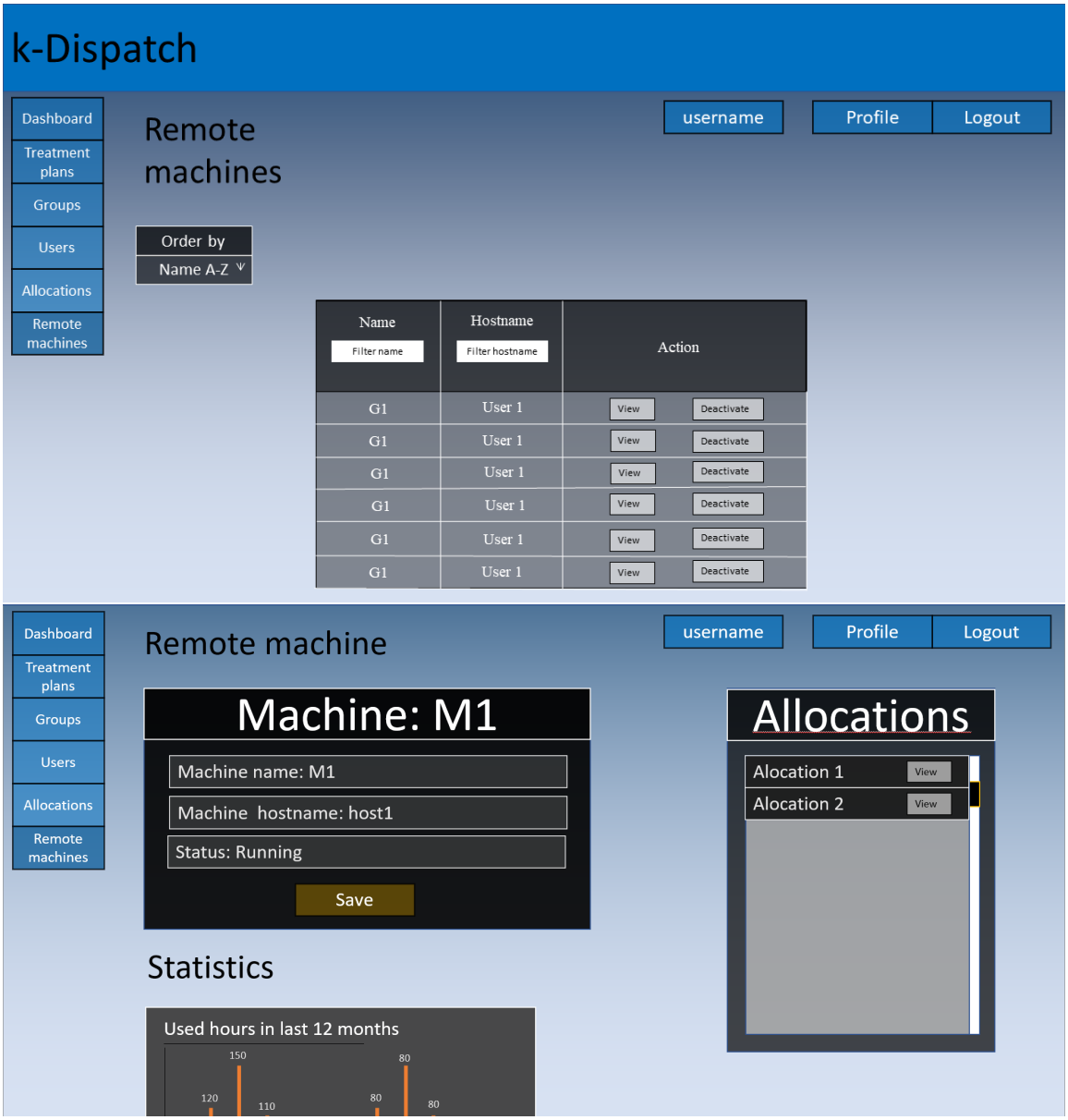
Obrázek A.1: Detailní návrh vzhledů stránek pro přihlášení a zobrazení podúloh.



Obrázek A.2: Detailní návrh vzhledů stránek se skupinami, detailem skupiny a uživateli.



Obrázek A.3: Detailní návrh vzhledů stránek uživatelského detailu, alokací a detailu alokace.

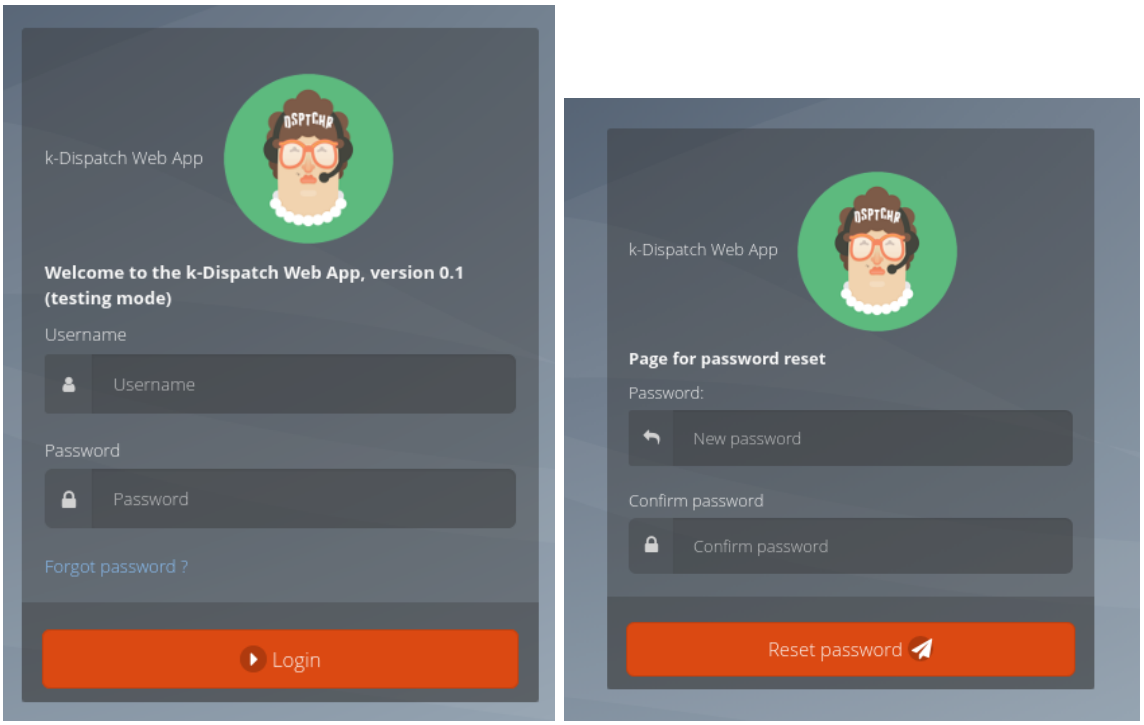


Obrázek A.4: Detailní návrh vzhledů stránek s výpočetními prostředky a detailem výpočetního prostředku.

Příloha B

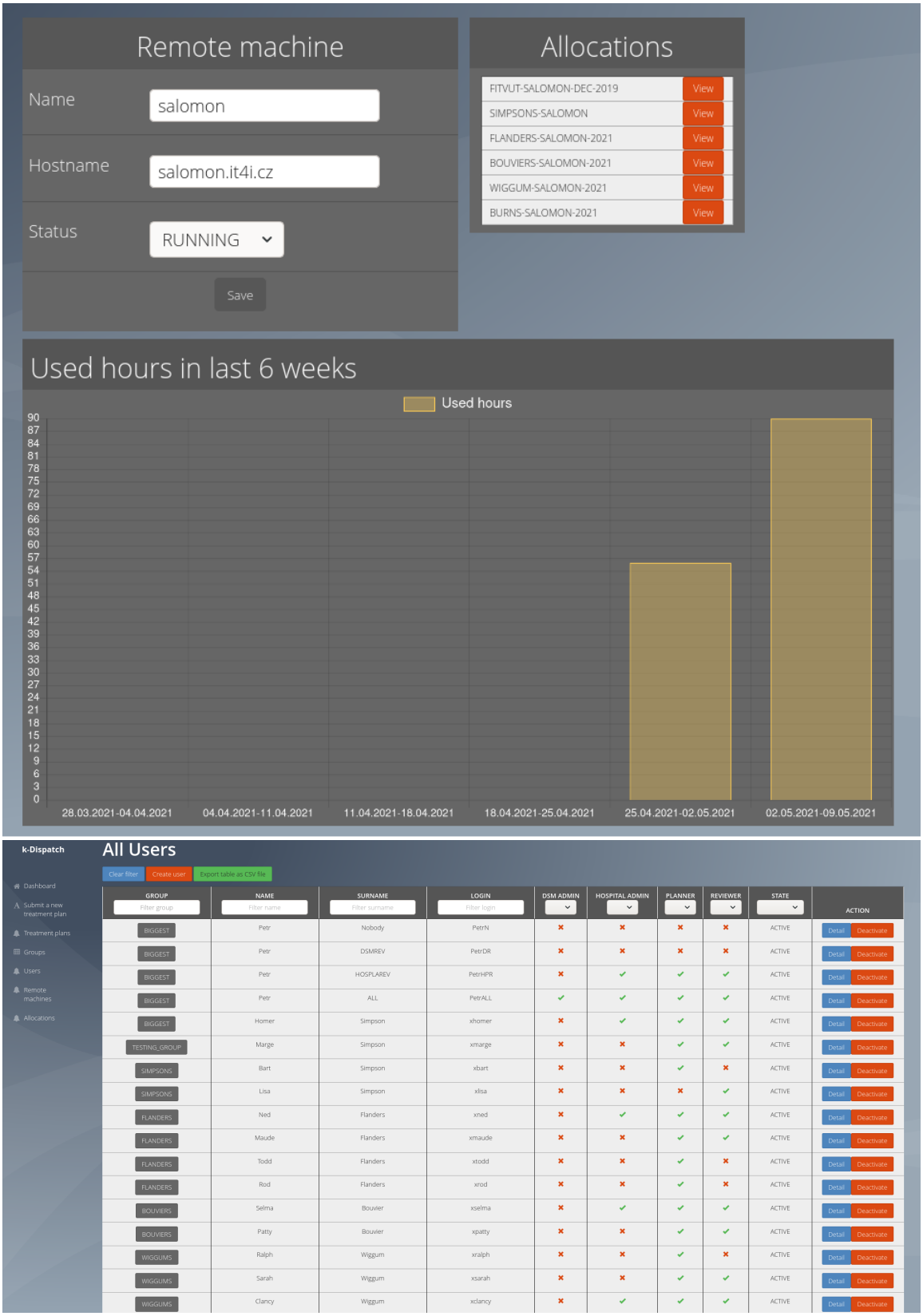
Finální podoba aplikace

V této příloze se vyskytují části finální webové aplikace.

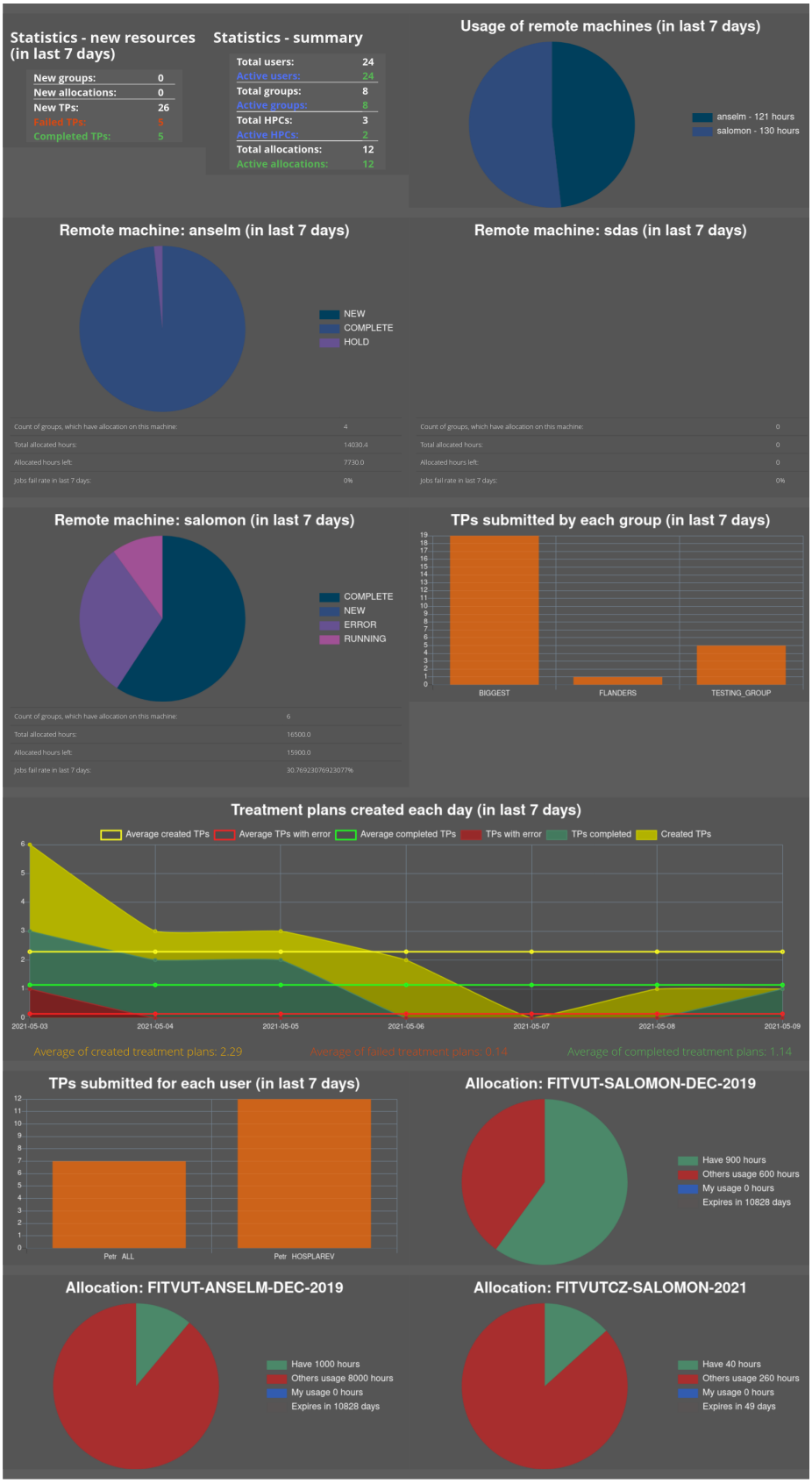


Obrázek B.1: Část stránky pro přihlášení uživatele a část stránky pro změnu zapomenutého hesla.

Obrázek B.2: Stránka se skupinami, stránka s detailem skupiny a výpočetními stroji.



Obrázek B.3: Stránka s detailem výpočetního stroje a stránka s uživateli.



Obrázek B.4: Ukázka exportu grafů uživatele se všemi rolemi na stránce nástěnky.

Obrázek B.5: Stránka s léčebnými plány a stránka s podúlohami léčebného plánu.

Obrázek B.6: Stránka s alokacemi a stránka s detailem alokace.

Příloha C

Obsah DVD

V přiloženém DVD se vyskytuje:

- xdanca01.pdf - Text práce.
- Source/ - Adresář obsahující všechny zdrojové soubory.
- Readme.txt - Soubor Readme obsahující manuál na spuštění aplikace.
- Tex/ - Zdrojové soubory, ze kterých byla vytvořena tato práce.