Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Webové rozhranie pre systém SALSA

Bakalárska práca

2020 **Dominik Matis**

Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky

Webové rozhranie pre systém SALSA

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	9.2.1. Informatika
Školiace pracovisko:	Katedra počítačov a informatiky (KPI)
Školiteľ:	Ing. Matej Madeja
Konzultant:	RNDr. Martin Vaľa, PhD.

2020 Košice **Dominik Matis**

Názov práce:	Webové rozhranie pre systém SALSA
Pracovisko:	Katedra počítačov a informatiky, Technická univerzita v Košiciach
Autor:	Dominik Matis
Školiteľ:	Ing. Matej Madeja
Konzultant:	RNDr. Martin Vaľa, PhD.
Dátum:	29. 05. 2020
Kľúčové slová:	dávkovacie systémy, monitorovacie systémy, webová aplikácia, React

Abstrakt: Táto bakalárska práca obsahuje proces vývoja webového rozhrania, ktoré má slúžiť používateľom na uľahčenie práce pri zadávaní, spúšťaní a ovládaní úloh spúšťaných na uzloch v rámci klastra. V úvode tejto práce je analýza viacerých monitorovacích a dávkovacích systémov, pričom sú tieto systémy aj porovnané. V ďalšej časti práce je návrh samotnej webovej aplikácie, pričom v tejto časti sú spomenuté viaceré moderné softvérové rámce a knižnice používané pri vývoji webových aplikácií. Okrem výberu technológie sú v tejto časti spomenuté aj technológie použité pre prenos informácií. Nasledujúcou časťou je návrh používateľského rozhrania. Táto časť obsahuje návrh postupnosti obrazoviek, prototyp aplikácie a tiež overenie tohto prototypu. Po tejto časti nasleduje samotná implementácia, ktorá popisuje, ako bola webová aplikácia vytvorená. V závere tejto práce je toto riešenie vyhodnotené a tiež tu sú opísané bezpečnostné hrozby tohto riešenia.

Thesis title:	Web interface for system SALSA
Department:	Department of Computers and Informatics, Technical University of Košice
Author:	Dominik Matis
Supervisor:	Ing. Matej Madeja
Tutor:	RNDr. Martin Vaľa, PhD.
Date:	29. 05. 2020
Keywords:	job scheduling systems, monitoring systems, web application, React

Abstract: This bachelor thesis contains the process of developing a web interface that is intended to make it easier for users to enter, run, and control tasks that run on nodes within a cluster. The introduction of this work is an analysis of several monitoring and job scheduling systems, while these systems are also compared. The next part of the work is the design of the web application itself, while several modern software frameworks and libraries used in the development of web applications are mentioned. In addition to the choice of technology, the technologies used for information transmission are also mentioned in this section. The next part is the design of the user interface. This section contains the design of the screen sequence, the prototype of the application, and also the verification of this prototype. This section is followed by the implementation itself, which describes how the web application was created. At the end of this work, this solution is evaluated and also the security threats of this solution are described here.

59012

TECHNICKÁ UNIVERZITA V KOŠICIACH

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Katedra počítačov a informatiky

ZADANIE BAKALÁRSKEJ PRÁCE

Študijný odbor: **Informatika**
Študijný program: **Informatika**

Názov práce:

Webové rozhranie pre systém SALSA

Web Interface for SALSA System

Študent: **Dominik Matis**
Školiteľ: **Ing. Matej Madeja**
Školiace pracovisko: **Katedra počítačov a informatiky**
Konzultant práce: **RNDr. Martin Vaľa, PhD.**
Pracovisko konzultanta:

Pokyny na vypracovanie bakalárskej práce:

1. Analyzovať existujúce možnosti monitorovania a dávkovania úloh na klastre a s tým spojené technológie.
2. Navrhnúť riešenie monitorovacieho a dávkovacieho systému, ktoré bude slúžiť na zadávanie, spúšťanie, ovládanie úloh a na monitorovanie stavu klastra.
3. Implementovať navrhnuté riešenie.
4. Overiť navrhnuté riešenie a vyhodnotiť ho.
5. Vypracovať dokumentáciu podľa pokynov vedúceho práce.

Jazyk, v ktorom sa práca vypracuje: slovenský
Termín pre odovzdanie práce: 29.05.2020
Dátum zadania bakalárskej práce: 31.10.2019

TECHNICKÁ UNIVERZITA V KOŠICIACH FAKULTA ELEKTROTECHNIKY A INFORMATIKY

12.

.....
prof. Ing. Liberios Vokorokos, PhD.
dekan fakulty

Čestné vyhlásenie

Vyhlasujem, že som záverečnú prácu vypracoval(a) samostatne s použitím uvedenej odbornej literatúry.

Košice, 29.05.2020 .....

Vlastnoručný podpis

Poďakovanie

Na tomto mieste by som rád poďakoval svojmu vedúcemu práce a konzultantovi práce za ich čas a odborné vedenie počas riešenia mojej záverečnej práce.

Rovnako by som sa rád poďakoval svojim rodičom a priateľom za ich podporu a povzbudzovanie počas celého môjho štúdia.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Experimenty vo fyzike vysokých energií vo svete generujú stále viac a viac dát, ktorých komplexita sa zvyšuje exponenciálne. Tieto dáta je potrebné analyzovať a neskôr aj zobraziť používateľovi napríklad vo forme grafu či histogramu. Našťastie, spolu so zvyšujúcim sa množstvom experimentálnych dát sa vyvíja aj hardvér, ktorý je schopný poskytovať väčšie množstvo výpočtových zdrojov na spracovanie takýchto dát. Obyčajný stolný počítač, by na takéto výkonovo náročné výpočty vôbec nestačil. Preto sa vytvárajú počítačové klastre, teda veľké množstvo počítačov spojených do jednej väčšej výpočtovej jednotky. Klaster obsahuje uzly, pričom každý takýto uzol je jedným menším počítačom. Na synchronizáciu a spravovanie všetkých uzlov v klastri je potrebný špeciálny softvér nazývaný aj dávkovací systém. Taktiež je potrebné aj veľké úložisko dát, s ohromným množstvom diskov uložených na serveroch. Viacero takýchto serverov tvorí tzv. sieťové úložisko dát. CERN využíva sieťové úložiská ako sú napríklad EOS [1], DPM [2], dCache [3] a i. Zosieťovanie dávkovacích systémov a úložiskových elementov je zoskupené do infraštruktúry GRID [4].

Dávkovací systém je typ softvéru, ktorý zabezpečuje orchestráciu, prideľovanie a monitorovanie všetkých úloh, ktoré sú a budú spustené na klastri. Medzi najznámejšie dávkovacie systémy patria napríklad SLURM [5], HTCondor [6] alebo Torque [7].

Aj keď sú v dnešnej dobe klastre veľmi výkonné, môže nastať situácia, pri ktorej sa vyčerpajú všetky výpočtové zdroje, ktoré sú určené pre úlohy. Práve kvôli tomu je potrebné aspekty týkajúce sa klastrov, ako napr. stav počítačovej siete, schopnosť čítania a zápisu na disk, využitie grafickej procesorovej jednotky, sledovať. Na tento účel slúžia monitorovacie aplikácie. Medzi najznámejšie aplikácie radíme napríklad Nagios [8], Netdata [9] alebo Monit [10].

Pretože všetky dávkovacie systémy a aj monitorovacie aplikácie majú určité

obmedzenia, rozhodli sme sa preto o vytvorenie webového rozhrania, ktoré má slúžiť ako zjednotenie dávkovacieho systému a monitorovacej aplikácie. Toto rozhranie umoží jeho používateľom pridávanie, rozdeľovanie a spúšťanie úloh a zároveň monitorovanie stavu klastra a jednotlivých uzlov v klastri z jedného prostredia. Taktiež toto rozhranie skráti dobu manipulácie s klastrom, teda od vytvorenia skriptu na spustenie úlohy až po samotné ukončenie úlohy. Našou úlohou je teda zjednodušiť a spríjemniť prácu s klastrom a jeho uzlami.

1 Formulácia úlohy

Cieľom tejto bakalárskej práce je vytvoriť webové rozhranie, ktoré má slúžiť používateľom na uľahčenie práce pri zadávaní, spúšťaní a ovládaní úloh, ktoré sú spúšťané na uzloch v rámci klastra. Taktiež má slúžiť ako monitorovací nástroj, s ktorým bude jednoduchšie zistiť stav jednotlivých komponentov klastra. Celkovo má tento projekt za cieľ spríjemniť prácu s klastrom a urýchliť spúšanie úloh.

Webové rozhranie má obsahovať dve sekcie. Prvá sekcia bude slúžiť na manipuláciu s úlohami pre klaster, teda bude náhradou za dávkovací systém. Druhá sekcia umožní sledovanie stavu jednotlivých elementov klastra, ktoré sú potrebné pre jeho plynulý chod. Ako príklad je možné uviesť zaťaženie centrálnych procesorových jednotiek, zaťaženie grafických procesorových jednotiek, rýchlosť zápisu a čítania z disku.

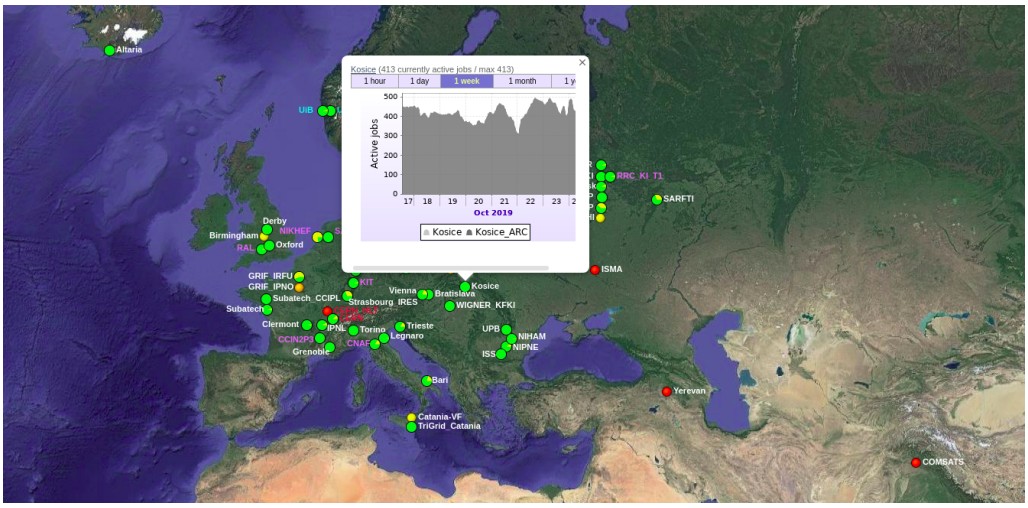
2 Analýza

Keďže táto práca má za úlohu zjednotiť dávkovací systém a monitorovaciu aplikáciu, je potrebné bližšie opísať tieto systémy, kde sú použité a taktiež je potrebné zamerať sa aj na ich výhody, nevýhody a ich vzájomné porovnanie.

2.1 GRID

Ako bolo v úvode spomenuté, GRID je infraštruktúra, ktorá má za úlohu zosieťovať spolu dávkovacie a úložné systémy. Jednou z možných implementácií je AliEn, ktoré používa experiment ALICE [11] v CERNe. Ako uvádzajú autori v publikácií [11], AliEn je skupinou nástrojov a služieb, ktoré implementujú GRID infraštruktúru. Pozostáva z viacerých komponentov a bol navrhnutý tak, aby ho bolo možné použiť popri iných implementáciách infraštruktúry GRID. Na obrázku 2.1 je možné vidieť miesta v Európe, kde sa nachádzajú klastre, ktoré v sebe obsahujú niektorú z implementácií GRID-u a taktiež je označený klaster nachádzajúci sa v Košiciach a je možné vidieť jeho vyťaženosť na grafe.

Ako jedným z najviac používaných úložných systémov v AliEn-e je systém EOS. EOS je, ako je v zdroji [1] uvedené, distribuovaný úložný systém zložený len z diskov podporujúci viaceré komunikačné protokoly. EOS je vyvíjaný v CERN-e od roku 2010 a je používaný na ukladanie dát z experimentov, ako sú LHC, ALICE, LHCB, ATLAS. Pre zaistenie bezpečnosti sú dáta duplikované a rozdistribuované medzi dve hlavné počítačové strediská nachádzajúce sa vo Švajčiarsku a Maďarsku. V dnešnej dobe je v EOS zahrnutých viac ako 44 000 pevných diskov o celkovej veľkosti viac ako 140 PB. Na obrázku 2.2 je možné vidieť využitie úložného priestoru a taktiež aj použitý systém a celkovú veľkosť v Košiciach a v CERN-e a na obrázku 2.3 informácie o všetkých klastroch, ktoré sú použité na experiment ALICE.



Obr. 2.1: Klastre v Európe

Disk storage elements

	ALICE::Kosice, ALICE::CERN										
	AliEn SE		Catalogue statistics						Storage-pr		
SE Name	AliEn name	Tier	Size	Used	Free	Usage	No. of files	Type	Size	Used	Free
1. CERN - CTA	ALICE::CERN::CTA	0	953.7 GB	9.975 MB	953.7 GB	0.001%	1	CTA	953.7 GB	37 MB	953.6 GB
2. CERN - EOS	ALICE::CERN::EOS	0	23.17 PB	19.09 PB	4.082 PB	82.38%	533,378,491	FILE	23.17 PB	19.96 PB	3.212 PB
3. CERN - EOSALICEDAQ	ALICE::CERN::EOSALICEDAQ	0	6.743 PB	5.356 PB	1.387 PB	79.43%	3,230,672	FILE	-	-	-
4. CERN - OCDB	ALICE::CERN::OCDB	0	318.3 TB	8.208 TB	310.1 TB	2.579%	3,916,984	FILE	318.3 TB	6.07 TB	312.3 TB
5. Kosice - EOS	ALICE::Kosice::EOS	2	571 TB	274.2 TB	296.8 TB	48.02%	10,889,768	FILE	571 TB	274 TB	297.1 TB
6. Kosice - SE	ALICE::Kosice::SE	2	420.2 TB	223.9 TB	196.3 TB	53.29%	5,468,992	FILE	-	-	-
Total			**31.19 PB**	**24.94 PB**	**6.254 PB**		**556,884,908**		**24.04 PB**	**20.23 PB**	**3.808 PB**

Obr. 2.2: Využitie úložiska v Košiciach a CERN-e

55. SPbSU - EOS	ALICE::SPbSU::EOS	2	72.76 TB	27.18 TB	45.58 TB	37.36%	791,639	FILE	72.76 TB	29.21 TB	43.55 TB	40.15%	Xrootd v4.8.4	4.4.8					24.10.2019 12:48	24	0	0	Test..
56. SPbSU - SE	ALICE::SPbSU::SE	2	38.14 TB	9.278 TB	28.86 TB	24.33%	155,277	FILE	38.14 TB	10.47 TB	27.67 TB	27.46%	Xrootd v4.8.1					Test..	24.10.2019 13:34	24	0	0	Test..
57. Strasbourg_IRES - SE2	ALICE::Strasbourg_IRES::SE2	2	291.1 TB	152.6 TB	138.5 TB	52.42%	2,157,223	FILE	291.1 TB	165.8 TB	125.2 TB	56.97%	Xrootd v4.8.5					Test..	24.10.2019 12:53	24	0	0	
58. Subatech - EOS	ALICE::Subatech::EOS	2	1.336 PB	1.076 PB	266.6 TB	80.52%	15,563,700	FILE	1.336 PB	1.079 PB	262.5 TB	80.81%	Xrootd v4.9.1	4.5.8					24.10.2019 12:47	24	0	0	
59. SUT - SE	ALICE::SUT::SE	2	103.3 TB	70.1 TB	33.21 TB	67.86%	477,030	FILE	103.3 TB	83.27 TB	20.04 TB	80.61%	Xrootd v4.9.1					Test..	24.10.2019 13:42	24	0	0	
60. Torino - SE	ALICE::Torino::SE	2	1.405 PB	1.063 PB	350.2 TB	75.66%	29,943,553	FILE	1.405 PB	1.11 PB	301.6 TB	79.03%	Xrootd v4.3.0					Test..	24.10.2019 13:38	24	0	0.296%	Test..
61. Trieste - SE	ALICE::Trieste::SE	2	61.84 TB	37.11 TB	24.73 TB	60.02%	598,049	FILE	61.84 TB	37.48 TB	24.36 TB	60.6%	Xrootd v3.3.4					Test..	24.10.2019 13:42	24	0	0	Test..
62. Troitsk - SE	ALICE::Troitsk::SE	2	112.7 TB	64.6 TB	48.1 TB	57.32%	1,337,842	FILE	112.7 TB	75.45 TB	37.23 TB	66.96%	Xrootd v4.10.1					Test..	24.10.2019 13:40	24	0	0	
63. UNAM_T1 - EOS	ALICE::UNAM_T1::EOS	2	573.1 TB	293.3 TB	279.8 TB	51.18%	8,610,931	FILE	573.1 TB	309.6 TB	263.4 TB	54.03%	Xrootd 3.3.6	0.3.35				Test..	24.10.2019 12:48	24	0	0.298%	Test..
64. UPB - EOS	ALICE::UPB::EOS	2	65 TB	37.66 TB	27.34 TB	57.93%	1,269,790	FILE	65.47 TB	37.66 TB	27.81 TB	57.53%	Xrootd v4.7.1						24.10.2019 12:54	24	0	0	
65. ZA_CHPC - EOS	ALICE::ZA_CHPC::EOS	2	383 TB	166.6 TB	216.4 TB	43.51%	7,673,730	FILE	348.8 TB	198.4 TB	150.4 TB	56.88%	Xrootd v4.9.1	4.5.8					24.10.2019 12:51	24	0	0	
Total			93.49 PB	70.25 PB	23.25 PB		1,668,821,971		81.71 PB	62.65 PB	19.06 PB				57	59	59	20					34

Obr. 2.3: Využitie niektorých úložísk z klastrov pre experiment ALICE

2.2 Dávkovacie systémy

V zdroji [12] je dávkovací systém definovaný ako systém, ktorý spúšťa mnoho dávkovacích úloh, ktoré môžu byť vykonané aj bez interakcie s človekom. Dávkovací systém sa volá dávkovací, pretože spúšťa úlohy v dávkach, teda viacero úloh za sebou. Ďalšie sekcie sú zamerané na najznámejšie dávkovacie systémy.

2.2.1 SLURM

Ako píšu Soner a Özturan [13], SLURM je softvér s otvoreným zdrojovým kódom slúžiaci na riadenie zdrojov, ktorý je distribuovaný pod licenciou GPL. Tento softvér bol navrhnutý s ohľadom na jednoduchosť, presnosť a škálovateľnosť. Ako ďalej uvádza Jette a Grondona [5], systém SLURM sa skladá z troch hlavných funckcií.

Prvou funkciou je alokácia a dealokácia prístupu ku zdrojom pre používateľov na čas potrebný na vykonanie ich úloh. Zdrojmi sú v tomto prípade uzly v klastri. Druhou kľúčovou funkciou systému SLURM je rámec, dávkovací systém, ktorý umožňuje spúšťanie, vykonávanie a monitorovanie úloh, prevažne paralelných, na alokovaných uzloch. Poslednou funkciou je schopnosť systému rozhodovať o požiadavkách na zdroje a riadiť rad požiadaviek čakajúcich na vybavenie. Niekedy môže nastať, že rôzne požiadavky požadujú zdroje, ktoré sú momentálne alokované pre iného používateľa, vtedy sa daná požiadavka posunie do radu podľa priority.

Používateľom je poskytnuté rozhranie príkazového riadku, pomocou ktorého môžu komunikovať so systémom SLURM. Architektúra systému sa skladá z démona, ktorý beží na každom výpočtovom uzle klastra a jedného riadiaceho démona spusteného len na manažmentovom uzle, prípadne aj jeho dvojičke.

2.2.2 HTCondor

Ako uvádza zdroj [14], HTCondor je vysoko priepustný výpočtový rámec s otvoreným zdrojom slúžiaci na spúšanie paralelných úloh, ktoré sú neinteraktívne, teda nepotrebujú interakciu s používateľom. Funkcionalitou je podobný systému SLURM tým, že v ňom existujú dva druhy uzlov, hlavné a pracovné. Hlavné uzly prijímajú úlohy od používateľov a plánovač, ktorý sa taktiež nachádza v hlavnom uzle, naplánuje spustenie používateľských úloh na pracovných uzloch a taktiež posúva všetky požiadavky na zdroje zo skupiny pracovných uzlov. Okrem samotného plánovača hlavný uzol obsahuje aj vyjednávača, ktorý riadi a prioritizuje prístup ku dostupným zdrojom pre dávkovací systém HTCondor. Zberateľ je ďalším démonom nachádzajúcim sa na hlavnom uzle. Jeho úlohou je pozbierať informácie o všetkých zdrojoch pracovných uzlov.

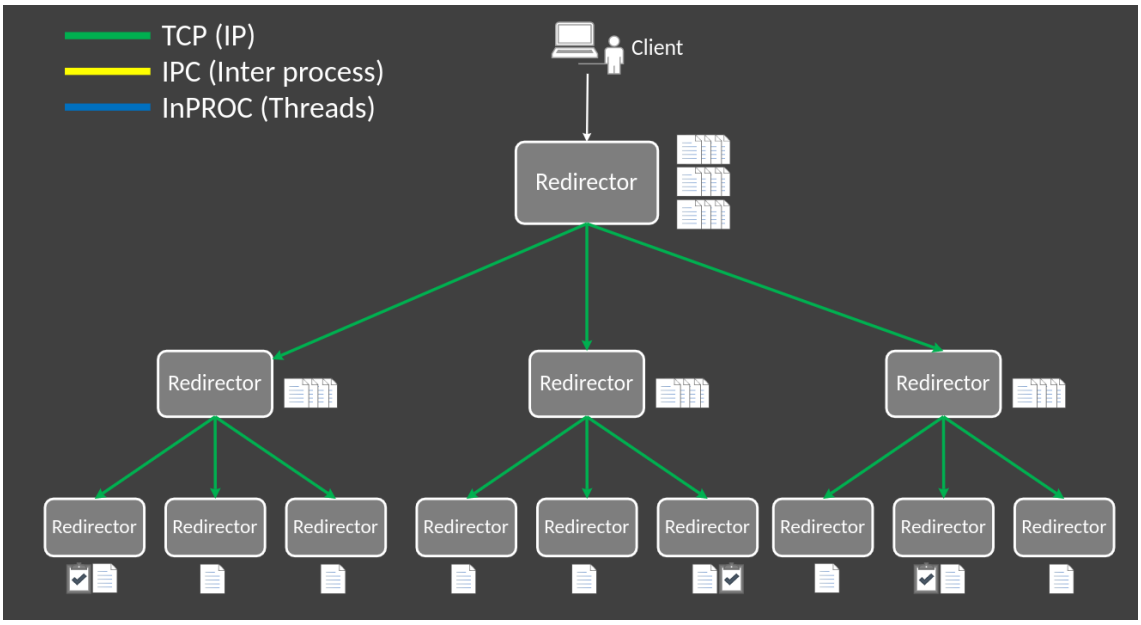
2.2.3 Torque

Ako uvádza príručka [7], Torque je správca zdrojov, ktorý môže byť použitý aj v dávkovacích systémoch. Klaster obsahujúci Torque obsahuje podobne ako SLURM a HTCondor dva druhy uzlov, jedným je hlavný uzol a ostatné sú určené ako výpočtové uzly. Používateľské príkazy na prácu s úlohami je možné aplikovať na hlavný uzol. Hlavný uzol taktiež podobne ako v ostatných dvoch systémoch obsahuje aj plánovač, ktorý komunikuje s hlavným démonom ohľadom práce so zdrojmi a ich uvoľnením pre jednotlivé úlohy. Torque je v dnešnej dobe zastaralý a nahradili ho iné systémy.

2.2.4 SALSA

SALSA je dávkovací systém, ktorý sa zameriava hlavne na to, aby bola analýza dát v N-rozmernom priestore jednoduchšia. Tento systém je stále vo vývoji. Obsahuje distribučný systém siete, ktorá môže byť vytvorená na viacerých úrovniach, ako napr. na úrovni celých klastrov, uzlov, procesov alebo aj na úrovni vlákien. Pomocou systému SALSA je možné vytvoriť akúkoľvek stromovú štruktúru. Používateľ môže jednoducho ovládať prácu s úlohami cez príkazový riadok a cez príkazový riadok taktiež dostáva informácie o stave všetkých úloh, ktoré sú spustené, dokončené alebo čakajú vo fronte na vykonanie. SALSA sa zameriava taktiež na kroky, ktoré sú potrebné pri analýze fyziky vysokých energií. Táto analýza sa skladá z niekoľkých krokov, a to rekonštrukcie experimentálnych alebo generovaných dát, analýzy rekonštruovaných dát, vygenerovanie výstupu vo forme histogramov a posledným krokom je práca s histogramami na vytvorenie výsledných obrázkov. SALSA dokáže vykonať prvé dva spomenuté kroky. Výhodou systému SALSA je, že histogramy z druhého kroku rozdistribuuje medzi viaceré výpočtové jednotky a vďaka tomu je možné rozanalyzovať veľké N-rozmerné histogramy, ktoré by sa pri klasickej analýze ani nezmestili do pamäte RAM. Tento systém pozostáva z viacerých typov uzlov, jedným z nich je hlavný uzol a druhým je pracovný uzol, ktorý vykonáva danú úlohu.

Ako je možné vidieť na obrázku 2.4, dávkovací systém SALSA má stromovú štruktúru, pričom obsahuje niekoľko úrovní. Na najvyššej úrovni sa nachádza sprostredkovateľ, ktorý označuje skupinu klastrov, ktoré slúžia na vykonávanie úloh. Pod skupinou klastrov je myslené, že na tejto úrovni je sprostredkovateľ ve-



Obr. 2.4: Stromová štruktúra uzlov

domý o každom klastri, na ktorom je spustený dávkovací systém SALSA. Táto úroveň sa vyznačuje tým, že zvyčajne ohraničuje jednu lokáciu, v ktorej sa nachádzajú výkonné klastre. Taktiež je na obrázku zjavné, že na tejto úrovni je veľmi veľa úloh na vykonanie. Úlohou sprostredkovateľa na tejto úrovni je zabezpečiť vykonanie týchto úloh pomocou prerozdelenia medzi jednotlivé klastre, pričom sa dbá na maximálne využitie výpočtových možností klastrov. Na ďalšej úrovni je možné vidieť jednotlivé klastre, ktoré sú dostupné v rámci jednej lokácie (napr. v rámci Spojeného Inštitútu pre Jadrový Výskum). Na tejto úrovni sa taktiež nachádzajú sprostredkovatelia, ktorých úlohou je taktiež rozdistribuovať zadané úlohy čo najefektívnejšie. Klaster je možné definovať ako zhluk mnohých uzlov, serverov, ktoré sú spoločne prepojené. Ich úlohou je poskytnúť väčší výpočtový výkon. Klastre vedia o všetkých svojich uzloch, čo sú vlastne jednotlivé výkonné počítače, ktoré slúžia na veľmi náročné výpočty. V prípade, že sprostredkovatelia nie sú spoločne s pracovnými uzlami v rámci jedného klastra, tak je možné povedať, že sprostredkovatelia nepotrebujú byť veľmi výkonní, ich jedinou úlohou je len rozdeľovať úlohy medzi pracovné uzly a následne zbierať výsledky výpočtov. Pri tejto stormovej štruktúre je treba spomenúť spôsob, akým sa úlohy rozposielajú a výsledky prijímajú na sprostredkovateľov. Dávkovací systém SALSA poskytuje niekoľko možností spojenia, pričom základným spôsobom prepojenia je pomo-

cou spoľahlivého protokolu TCP za pomoci IP smerovania. Tento spôsob je účinný najmä v prípadoch, ak sa sprostredkovatelia a jednotlivé pracovné uzly nenachádzajú v rámci jedného miesta, resp. klastra. Na obrázku 2.4 je takéto prepojenie naznačené zelenou farbou medzi každou úrovňou.

Druhým možným spôsobom prenosu dát je za pomocou medziprocesovej komunikácie (IPC, angl. *Inter-Process Communication*). Medziprocesová komunikácia je v zdroji [15] opísaná ako typ komunikácie, ktorý prebieha medzi viacerými vláknami alebo procesmi. Tento typ komunikácie pomáha pri probléme, keď rodičovský proces vytvorí dcérsky proces. V klasickom prípade nie je možné posielať informácie medzi rodičom a potomkom. V prípade medziprocesovej komunikácie je hlavným nástrojom využívaným pri posielaní dát medzi procesmi tzv. rúra (angl. *pipe*). Pri pojme rúra je tiež potrebné spomenúť výraz duplex. Duplex je typ komunikácie, ktorý určuje, akým smerom môžu dáta smerovať, resp. či je premávka obojsmerná alebo jednosmerná. V prípade obojsmernej premávky sa tento typ komunikácie nazýva plný duplex (angl. *full duplex*), to znamená, že dáta môžu prúdiť v oboch smeroch súčasne. Opakom je polovičný duplex (angl. *half duplex*), ktorý povoľuje posielanie dát len v jednom smere súčasne.

Ďalším spôsobom komunikovania medzi sprostredkovateľmi a pracovnými uzlami je komunikácia v rámci procesu (InPROC, angl. *InProcess Communication*). Tento druh umožňuje na rozdiel od IPC komunikovať len medzi vláknami jedného procesu. Tento druh komunikácie je vhodný, ak je potrebné komunikovať v rámci jedného procesu, pričom tento proces nevytvára žiadne dcérske procesy.

2.3 Monitorovacie systémy

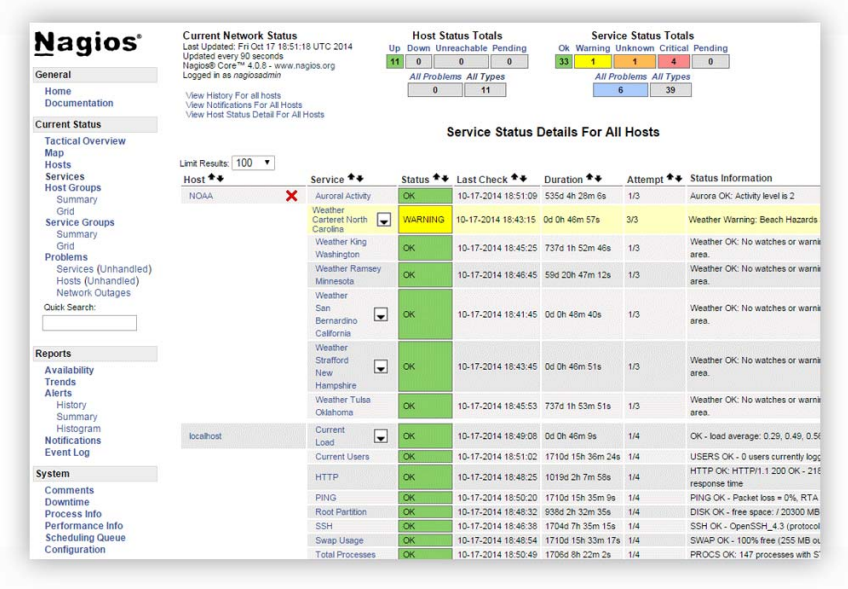
Aby sme mohli overovať stav klastrov, sledovať ich záťaž a poprípade kontrolovať ich vyťaženosť, potrebujeme mať nejakým spôsobom kontrolu nad daným klastrom z pohľadu softvéru aj hardvéru. Pre tento účel slúžia monitorovacie systémy. Ich úlohou je sledovať stav klastrov a informovať používateľa alebo administrátora o ich stave.

2.3.1 Nagios

V zdroji [8] je Nagios predstavený ako monitorovací systém s otvoreným zdrojovým kódom, ktorý má pomocť pri identifikácií a riešení problémov ešte skôr, ako

by to mohlo poškodiť kritické časti infraštruktúry. Tento systém bol podľa zdroja [16] spustený v roku 1999 ako *NetSaint*, ktorého meno sa v roku 2002 premenovalo na *Nagios*. Ďalej, ako je v zdroji [17] uvedené, Nagios poskytuje širokú škálu vlastností, medzi ktoré by sme mohli zaradiť napr. centrálny rozhľad nad monitorovanou infraštruktúrou, automatické reštartovanie služieb a aplikácií, v ktorých nastanú chyby, API, pomocou ktorého je možné pridávať rôzne aplikácie tretích strán a v neposlednom rade aj veľkú komunitu používateľov po celom svete.

Používateľské rozhranie monitorovacieho systému Nagios je jednoduché na používanie, pričom poskytuje všetky potrebné informácie v tabuľkovom tvare. Snímku obrazovky systému Nagios je možné vidieť na obrázku 2.5, na ktorom je možné tiež vidieť tabuľku, v ktorej sú zobrazené jednotlivé služby na jednotlivých uzloch v klastri a tiež ich stav, dátum poslednej kontroly a tiež krátke informácie o stave.

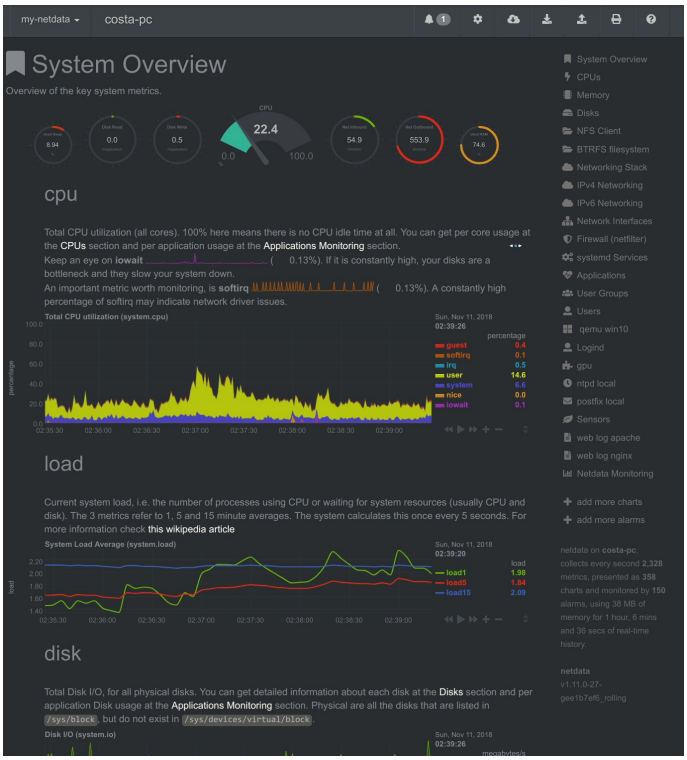


Obr. 2.5: Snímka obrazovky pri používaní monitorovacieho systému Nagios

2.3.2 Netdata

Ako je uvedené v zdroji [9], Netdata je monitorovací systém s otvoreným zdrojovým kódom určený na monitorovanie systémov a aplikácií. Je distribuovaný, monitoruje výkon a zdravie systému aplikácií v reálnom čase. Je vhodný na použitie ako na reálnych, tak aj na virtuálnych serveroch, kontajneroch a taktiež na IoT

zariadeniach. Snímku obrazovky z používania systému Netdata je možné vidieť na obrázku 2.6.

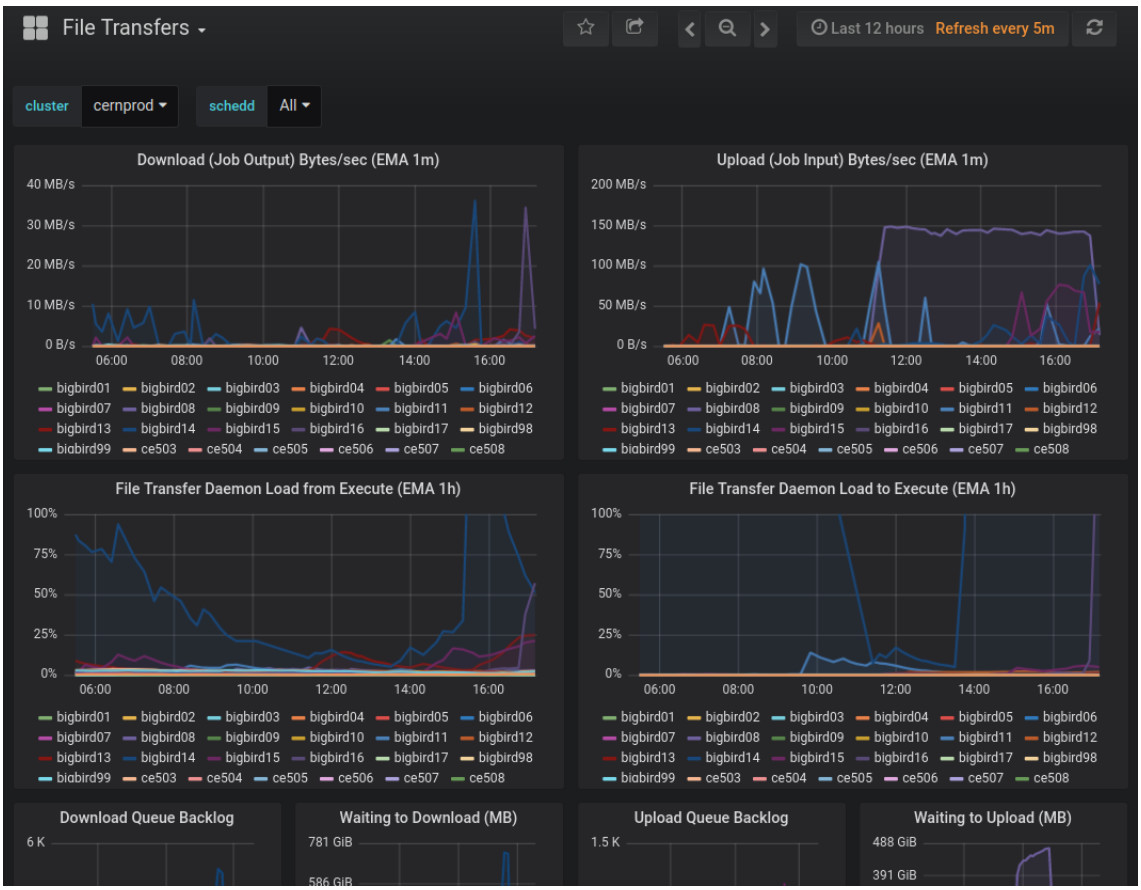


Obr. 2.6: Snímka obrazovky systému Netdata

2.3.3 Monit

Ako je uvedené v publikácií [10], CERN používal monitorovací systém LEMON, ktorý zbieral a ukladal informácie zo všetkých počítačov dátových centier, IT služieb a rôznych iných zariadení, ako napríklad sieťových prepínačov a snímačov prostredia ako teplota, vlhkosť a pod. Okrem systému LEMON bol používaný aj tzv. WLCG monitoring, ktorý mal za úlohu monitorovať celú infraštruktúru GRID. Tento obsahoval niekoľko špecializovaných aplikácií vyvíjaných v CERN-e. Do roku 2013 boli oba systémy založené na relačných databázach, neskôr boli navrhnuté na použitie s inými technológiami určenými na dátovú analýzu akými sú napríklad Apache Hadoop alebo Apache Spark. Neskôr sa tieto systémy spojili a vytvorili Monit, ktorý ma slúžiť na monitoring pre dátové centrá a aj pre WLCG. Architektúra monitorovacieho systému Monit spočíva v niekoľkých úlohách na dátach, medzi ktorými sú dátové zdroje, teda získavanie dát z rôznych zdrojov.

Ďalšou úlohou je prenos dát, teda prenos od zdroja až po úložisko, spracovanie dát, pričom táto úloha zabezpečí prípravu dát na zobrazenie, neskôr je tieto spracované dáta potrebné uložiť a následne je nutné tieto dáta určitým spôsobom vizualizovať. Na obrázku 2.7 je možné vidieť vizualizáciu dát pomocou grafov.

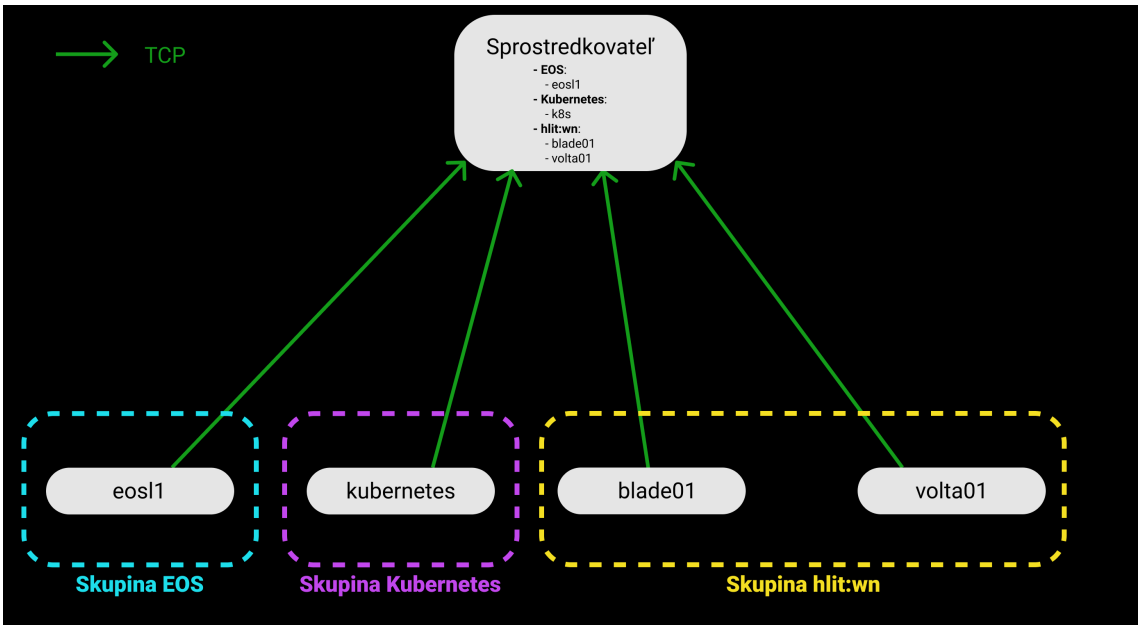


Obr. 2.7: Ukážka monitorovacieho systému Monit
[18]

2.3.4 Obmon

Tento monitorovací systém bol napísaný v jazyku C++, pričom na zisťovanie informácií o systéme používa knižnicu **libgtop**. Obmon je voľne dostupný a jeho zdroj je otvorený, teda ktokoľvek môže upravovať jeho zdrojový kód pre vlastné potreby. Tento systém posiela aktualizácie ohľadom stavu systému každú sekundu pomocou knižnice ZMQ, pričom nemá vlastnú databázu, ktorú je v prípade nutnosti možné doimplementovať. Obmon môže byť nasadený na každý uzol z klastrov a dáta budú posielané na brokera (z angl. *sprostredkovateľa*), odkiaľ môžu byť

čítané. V tomto prípade sa API pripája taktiež pomocou knižnice ZMQ na tohto sprostredkovateľa a ďalej tieto dáta poskytuje pre webovú aplikáciu. Dáta sú v tomto API uložené, aby bolo možné dodať všetky získané dáta od poslednej požiadavky, ktorú vyvolala napríklad webová aplikácia. API, ktoré je používané v tomto prípade je **zmq2ws**, pričom rovnako ako **Obmon** je to voľne dostupný projekt s otvoreným kódom. Výhodou tohto projektu je možnosť napojenia API na viacerých sprostredkovateľov a dáta od nich sú potom kategorizované podľa toho, od ktorého sprostredkovateľa prišli, resp. o aký klaster sa jedná.



Obr. 2.8: Stromová štruktúra monitorovacieho systému Obmon

Ako je možné vidieť na obrázku 2.8, systém má podobnú stromovú štruktúru ako systém SALSA. Činnosť každého uzlu je monitorovaná pomocou senzoru, ktorý tieto dáta odosiela na sprostredkovateľa spolu s informáciou o tom, do ktorej skupiny klastrov daný uzol patrí. Táto komunikácia prebieha cez spoľahlivého protokolu transportnej vrstvy TCP. Na obrázku sú jednotlivé skupiny klastrov zobrazené pomocou prerušovanej čiary, pričom na rozlíšenie je použitá iná farba a názov skupiny, do ktorej uzol, resp. klaster patrí je možné vidieť pod touto čiarou. Okrem dát a informácie o skupine je taktiež posielané meno hosťa (angl. *hostname*), ktorý jednoznačne určuje daný uzol v počítačovej sieti. Informácie sú posielané v intervale 1 sekundy, pričom pre niektoré informácie je potrebné robiť 2 merania. Medzi tieto informácie patrí využitie siete, disku a CPU. V prípade týchto

dát sa meranie robí každú sekundu a aktuálna hodnota je vypočítaná pomocou dvoch hodnôt a to aktuálnej a hodnoty z predošlého merania. Pomocou tejto informácie je možné na sprostredkovateľovi vytvoriť pole skupín klastrov a tiež každá z týchto skupín môže obsahovať pole jednotlivých uzlov, resp. klastrov. Obrázok 2.8 taktiež znázorňuje reálnu štruktúru klastrov, ktoré sú používané v Spojenom Inštitúte pre Jadrový Výskum v meste Dubna v Rusku a tiež v Slovenskej Akadémii Vied v Košiciach.

2.3.5 Porovnanie monitorovacích systémov

Keďže každý monitorovací systém má výhody aj nevýhody, tabuľka 2.1 zobrazuje rozdiely medzi niektorými faktormi, ktoré sú dôležité pri monitorovaní.

Vlastnosť	Nagios	Netdata	Monit	Obmon
Backend	Shell	C++	Shell	C++
Webová technológia	PHP	JavaScript	JavaScript	JavaScript
Obnovovací interval	60s	1s	automaticky	1s

Tabuľka 2.1: Porovnanie monitorovacích systémov
[9, 18, 19]

2.3.6 Vyhodnotenie monitorovacích systémov

V tomto projekte sme sa rozhodli použiť poslednú možnosť a to Obmon, pretože bol vytvorený práve pre prácu so systémom SALSA, pričom dáta z tohto senzoru je možné získať od nami vytvoreného API **zmq2ws**, ktoré bolo taktiež vytvorené pre prácu so systémom SALSA.

2.4 Webové technológie

V tejto časti bude rozbor technológií, ktoré je možné použiť na implementovanie tejto webovej aplikácie. Na komunikáciu s API použijeme WebSocket. Táto aplikácia je spravovaná cez systém git a taktiež za pomoci platformy GitLab, na ktorej je možné túto aplikáciu postaviť a dokonca aj nasadiť.

2.4.1 Angular

V zdroji [20] je Angular predstavený ako softvérový rámec, ktorý je vhodný na použitie pri vývoji efektívnych jednostránkových aplikácií. Jednostránková aplikácia (angl. Single-Page Application, skr. SPA) je typ aplikácie, pri ktorej webová aplikácia pracuje s webovým prehliadačom tak, že mení dynamicky mení svoj obsah vzhľadom na nové dáta zo servera. Takýto prístup nevyžaduje od webového prehliadača, aby vykreslil celú stránku nanovo. Angular používa šablóny na vytváranie komponentov, pričom sa riadi vzorom MVC, teda Model-View-Controller. *View*, teda vzhľad je vytvorený pomocou jazyka HTML. Reprezentuje to dáta, ktoré sú zobrazené používateľovi. Úprava vzhľadu komponentu je možná za pomoci štýlovacieho jazyka CSS. *Model* reprezentuje formu dát, ako bude zobrazená používateľovi. V softvérovom rámci Angular je **model** služba, ktorá nevie o vzhľade a ani kontroléri aplikácie. Model je napísaný v jazyku TypeScript, ktorý Angular používa. Hlavný rozdiel medzi jazykom TypeScript a JavaScript je ten, že jazyk TypeScript podporuje typy, čiže je menej náchylný na chyby napríklad pri vložení dát zlého typu. Posledným dielom je kontrolér (angl. *controller*), ktorý zabezpečuje funkcionalitu komponentu. Zvyčajne obsahuje triedu komponentu a taktiež dekorátor **@Component**, ktorý slúži ako informácia o tom, kde sa nachádza šablóna komponentu, štýlovanie komponentu a hlavne názov komponentu, pod akým sa bude môcť použiť v HTML kóde ako tag. Okrem samotných šablón používa Angular aj direktívy, ktoré zabezpečujú logiku programu v šablóne tak, že menia štruktúru vzhľadu.

2.4.2 Vue.js

Vue.js, ako je v zdroji [21] uvedené, je knižnica, ktorá bola vytvorená za účelom rýchleho prototypovania, no v dnešnej dobe je jej použitie možné v rôznych reaktívnych aplikáciách. Podobne ako Angular, aj Vue.js sa drží vzoru MVC. Táto knižnica umožňuje jednoducho spojiť model do vzhľadu aplikácie, pričom tento model je vo forme jednoduchého objektu v jazyku JavaScript. Pre vytvorenie Vue.js aplikácie je za potreby, aby HTML element, v ktorom chceme, aby sa daná aplikácia spustila, mala identifikátor. Vue.js aplikácia sa začína v čase, keď pridáme do zdrojového kódu volanie konštruktora triedy **Vue**. Tento konštruktor berie ako argument objekt, ktorý obsahuje atribút **el**, ktorý označuje element, v ktorom má byť

aplikácia spustená. Okrem tohto atribútu môže obsahovať ešte ďalšie atribúty ako napríklad **data**, ktorý zabezpečí dáta, ku ktorým bude možné neskôr pristúpiť v HTML kóde. Vue.js podobne ako Angular používa direktívy, ktoré pomahájú pri vypisovaní polí alebo iných podobných štruktúr.

2.4.3 React

React je, ako uvádza zdroj [22], knižnica naprogramovaná v programovacom jazyku JavaScript, ktorý je v dnešnej dobe používaný primárne práve na vývoj webových aplikácií. React je vyvíjaný spoločnosťou Facebook. Ako je ďalej v tomto zdroji uvedené, medzi hlavné atribúty knižnice React môžeme zaradiť to, že je deklaratívny, teda vytvorený kód je možné ľahšie odladiť a je aj ľahšie predpovedať, ako sa bude kód správať. Celý princíp tejto knižnice je založený na komponentoch, čiže celé používateľské rozhranie je vytvorené za pomoci malých častí, ktoré sú ľahšie ovládateľné a komplexné stavy celého rozhrania je takto možné lepšie upravovať. Každý komponent ktoré si spravuje svoj vlastný interný stav, pričom všetky tvoria komplexné riešenie celku. Ďalším atribútom, ktorý React ponúka, je možnosť vytvárania mobilných aplikácií za pomoci React Native. Pri vytváraní komponentov v React-e je možné použiť JSX. Ako je napísané v zdroji [23], JSX je syntaxové rozšírenie pre JavaScript, ktorý sa podobá na kód napísaný v značkovacom jazyku HTML.

2.4.4 Výber technológie na vývoj

Po dlhom rozhodovaní sme sa rozhodli použiť na vývoj webovej aplikácie knižnicu React. Pre knižnicu React sme sa rozhodli preto, lebo jeho použitie je jednoduché, nie sú potrebné žiadne šablóny, ako je tomu u rámca Angular a taktiež táto knižnica podporuje jazyk JSX, čo si myslíme, že je krajšie ako použitie direktív, ktoré sú dostupné v rámci Angular a knižnici Vue.js.

2.4.5 WebSocket

V zdroji [24] je WebSocket definovaný ako protokol, ktorý umožňuje obojsmernú komunikáciu medzi klientom a vzdialeným hostiteľom. Výhodou tejto technológie je, že pre obojsmernú komunikáciu nepotrebuje otvárať viacero HTTP spojení.

Pri vytváraní spojenia medzi klientom a hostiteľom sa vytvára otváracie podávanie ruky, ktoré je kompatibilné so serverom, ktorý spracováva HTTP požiadavky. Taktiež ako je napísané v zdroji [25], WebSocket server môže spracovávať požiadavky na rovnakom porte ako HTTP server (teda 80 v prípade HTTP alebo 443 v prípade HTTPS). Pre túto webovú aplikáciu sme sa rozhodli použiť práve tento spôsob komunikácie, pretože umožňuje to, aby webová aplikácia nebola nútená sa aktualizovať po každom získaní nových dát. Je možné povedať, že webová aplikácia odoberá údaje od API servera bez toho, aby jej poslala aplikácia požiadavku.

2.4.6 GitLab

GitLab, ako je uvedené v zdroji [26], je platforma, ktorá poskytuje nástroje na vývoj a prevádzku aplikácií (angl. *DevOps*). Táto platforma má otvorený zdroj, čo znamená, že ktokoľvek môže vidieť zdrojový kód tejto aplikácie a dokonca môže vytvárať vlastné zmeny, ktoré sa neskôr môžu dostať do reálnej verzie aplikácie. V našom projekte sme sa rozhodli použiť nástroj GitLab CI/CD. Continuous Integration (CI, z angl. kontinuálna integrácia) je spôsob, ako vývojári môžu automaticky testovať a postaviť svoj kód do spustiteľných súborov. Týmto spôsobom je možné odhaliť chyby v kóde oveľa skôr ako bude výsledná aplikácia dodaná používateľovi. Na druhej strane, Continuous Delivery / Deployment (CD, z angl. kontinuálne dodávanie / nasadenie) je metóda, pomocou ktorej je možné uvoľniť do produkcie. V prípade kontinuálneho dodávania je toto uvoľnenie možné manuálne a v prípade kontinuálneho nasadenia je tento proces automatický.

2.4.7 Bootstrap

Bootstrap je podľa zdroja [27] softvérový rámec, ktorý slúži na vytváranie dizajnovo príťažlivých používateľských rozhraní. Tento rámec obsahuje všetko potrebné, aby vývojár mohol vytvoriť používateľské rozhranie, vrátane rôznych tém, pričom niektoré sú bezplatné a niektoré sú platené. Rámec Bootstrap obsahuje taktiež sadu ikon, ktoré sú vo vektorovom formáte, a teda nestrácajú na kvalite pri zmene veľkosti a sú vhodné na použitie na rôznych rozlíšeniach používateľských obrazoviek.

Medzi výhody tohto rámca je možné zarátať rýchlosť vývoja, pričom tento rámec už má pripravené štýlovania, ktoré je možné aplikovať na jednotlivé elementy

aplikácie pomocou špeciálnych tried. Samozrejmosťou v dnešnej dobe je, že Bootstrap je vytvorený najmä pre mobily, teda medzi jeho vlastnosti je možné zaradiť responzívnosť.

Na druhú stranu, medzi hlavné nevýhody Bootstrapu je nutné spomenúť to, že stránky, či aplikácie, ktoré boli naštýlované za pomoci tohto rámca zvyčajne vyzerajú podobne, keď nie rovnako. Ďalšou nevýhodou je to, že rámec potrebuje pre svoj správny chod skripty napísané v jazyku JavaScript, pričom spolu tieto kódy zaberajú desiatky kilobajtov, a teda spomaľujú načítavanie stránky.

2.4.8 Material-UI

Druhým možným riešením grafického používateľského rozhrania je rámec Material-UI. Tento rámec je, ako uvádza zdroj [28] vytvorený priamo pre knižnicu React a ich hlavným cieľom je pomôcť urýchliť vývoj používateľských rozhraní v knižnici React a tiež zlepšiť ich prístupnosť pre používateľov. Tento rámec poskytuje vývojárom možnosť naštýlovať si vlastnú tému pomocou zmeny palety farieb, typografie a mnoho iných atribútov.

Výhodou Material-UI je to, že nepotrebuje pre svoj chod dodatočné JavaScript kódy, ktoré predlžujú načítavanie stránky či aplikácie. Ďalšou výhodou je široká paleta predpripravených komponentov, s ktorými je možné vyskladať dizajn aplikácie.

Nevýhodou tohto rámca môže byť fakt, že tento rámec založila firma Google, a teda rovnaký dizajn je používaný aj na systéme Android. Vývojári, ktorí tvoria platformovo nezávislé aplikácie môžu mať ťažšiu situáciu s týmto rámcom, pretože podpora pre systém iOS nie je úplná a tiež používateľom systému iOS príde nezvyčajné použitie dizajnu používaného pre Android v systéme iOS.

V tejto webovej aplikácii sme sa rozhodli použiť práve tento rámec, pretože nechceme mať dizajn podobný mnohým iným webovým stránkam, či aplikáciám a tiež kvôli tomu, že Material-UI je vytvorený priamo pre knižnicu React a má pripravené všetky potrebné komponenty.

3 Konceptuálny návrh

V tejto časti práce bude spomenutý návrh tejto webovej aplikácie. Pomocou návrhu je možné určiť, aké požiadavky sú kladené na túto webovú aplikáciu. Cieľom tejto aplikácie je spríjemnniť a zefektívniť prácu s klastrom pre používateľov, pričom sa snaží o dosiahnutie väčšej interaktivity voči bežnej práci s klastrom pomocou príkazového riadku.

Prvá časť tejto kapitoly bude hovoriť o prípadoch použitia tejto webovej aplikácie. Táto časť presne špecifikuje, na aké prípady je možné túto aplikáciu použiť.

V druhej časti je spomenutá analýza objektov a operácií s týmito objektami. To je veľmi dôležité ja zjednotenie toho, čo je potrebné, aby táto aplikácia obsahovala a tiež, ako by danú aplikáciu mali pochopiť používatelia tejto webovej aplikácie.

3.1 Prípady použitia

Prípady použitia označujú situácie, v ktorých môže byť daná aplikácia či produkt, použité. Pri určovaní prípadov použitia je potrebné zadefinovať účastníka, podmienky, ktoré musia byť splnené pre vstup do daného prípadu použitia a tiež je potrebné zdôrazniť výstupné podmienky, ktoré sú uskutočnené v prípade úspešného použitia. Priebeh použitia je znázornený pomocou scenáru.

3.1.1 Zistiť stav klastra

Primárnym účastníkom tohto prípadu použitia je vedecký pracovník. Podmienkou, ktorú je nutné dodržať je to, že daný pracovník má otvorenú webovú aplikáciu. Úspechom je možné určiť stav, kedy sa pracovník dozvie informácie o aktuálnom stave klastra.

Postup pre úspešné použitie je taký, že pracovník si vyberie z bočného menu položku Obmon. Následne si vyberie jednu skupinu klastrov z viacerých ponúk-

nutých. Potom pracovník vyberie samotný klaster, ktorého stav chce vidieť. Webová aplikácia zobrazí tabuľku s údajmi o stave daného klastra.

Doplňujúcou funkcionalitou pri zobrazení tabuľky je možnosť zmeniť formát zobrazovania tabuľky.

3.1.2 Zobraziť svoj profil

Vedecký pracovník je účastníkom tohto prípadu použitia. Pri vstupe je podmienkou to, aby pracovník mal otvorenú aplikáciu v prehliadači a tiež, aby tento pracovník mal používateľský účet, pomocou ktorého sa prihlási. Za úspešné použitie v tomto prípade je možné považovať to, že daný pracovník môže vidieť svoj vlastný používateľský profil.

Scenár použitia začína tým, že pracovník klikne v aplikácii na tlačidlo Login, ktoré sa nachádza v hornej lište aplikácie na pravej strane. Po kliknutí na toto tlačidlo, aplikácia presunie pracovníka na podstránku s prihlasovacím formulárom, do ktorého zadá svoje prihlasovacie údaje. Po zadaní údajov do príslušných polí je možné kliknúť na tlačidlo Login. Po úspešnom prihlásení je pracovník presunutý na podstránku so svojim profilom.

V prípade, že používateľ je už prihlásený a nachádza sa na akejkoľvek podstránke tejto webovej aplikácie, môže pristúpiť ku svojmu profilu za pomoci ikony používateľa v hornej lište na pravej strane.

3.1.3 Zistiť, kto je pripojený a ktoré služby sú dostupné

Účastníkom prípadu použitia je taktiež vedecký pracovník. Vstupnou podmienkou je to, aby pracovník mal otvorenú aplikáciu v prehliadači. Za úspešné použitie v tomto prípade je možné považovať to, že daný pracovník môže vidieť to, kto je aktuálne pripojený, skratku krajiny, odkiaľ sa napája, ich operačný systém a tiež typ internetového prehliadača. Tiež bude môcť vidieť aktuálne dostupné služby a verzie API a webovej aplikácie.

Scenár použitia je taký, že pracovník navštívi hlavnú stránku webovej aplikácie alebo klikne na ikonu používateľov, ktorá sa nachádza v hornej lište pri ikone menu a čísle, ktoré udáva počet práve pripojených používateľov. Po presunutí na danú podstránku je možné vidieť mapu a v prípade, že sú práve pripojení aj iní používatelia tak je v mape zobrazená aj ich približná pozícia.

3.1.4 Spustiť úlohu na klastri

Účastníkom je rovnako ako v iných prípadoch použitia vedecký pracovník. Podmienkou pri vstupe je to, aby pracovník mal otvorenú aplikáciu v prehliadači, mal používateľský účet a bol prihlásený v aplikácii. Za úspešné použitie v tomto prípade je možné považovať to, že úloha, ktorú chcel pracovník spustiť sa spustila, resp. čaká vo fronte na vykonanie.

Postup pre úspešné použitie je taký, že pracovník si vyberie z bočného menu položku Salsa. Následne si vyberie jednu skupinu klastrov z viacerých ponúknutých. Potom pracovník vyberie samotný klaster, na ktorom chce spustiť úlohu. Webová aplikácia zobrazí panel s 3 časťami. Pracovník klikne na časť Submit, v ktorej sa nachádzajú vstupné polia pre zadanie príkazu a počtu vykonaní daného príkazu. Po zadaní požadovaných údajov môže pracovník kliknúť na tlačidlo Execute job, ktoré danú úlohu pošle na API server, ktorý ju potom pošle do systému SALSA a spustí ju, resp. pridá do frontu úloh na vykonanie. Pracovník je potom presunutý v aplikácii do sekcie Job queue, ktorá obsahuje tabuľku s úlohami.

Doplňujúcou funkcionalitou je možnosť pracovníka skopírovať si vygenerovaný príkaz pomocou spodného okna s príkazom. Pre skopírovanie do schránky je potrebné kliknúť na toto okno.

3.1.5 Spravovať úlohu na klastri

Účastníkom je vedecký pracovník. Vstupnou podmienkou je to, aby vedecký pracovník mal otvorenú aplikáciu v prehliadači, mal používateľský účet, bol prihlásený v aplikácii a mal spustenú úlohu. Za úspešné použitie v tomto prípade je možné považovať to, že stav vykonávania úlohy, ktorú pracovník spustil, je možné sledovať v reálnom čase a tiež je možné zastaviť jej vykonávanie.

Postup pre úspešné použitie je taký, že pracovník si vyberie z bočného menu položku Salsa. Následne si vyberie jednu skupinu klastrov z viacerých ponúknutých. Potom pracovník vyberie samotný klaster, na ktorom má spustenú úlohu. Webová aplikácia zobrazí panel s 3 časťami. Pracovník klikne na časť Job queue, v ktorej sa nachádza tabuľka s frontom úloh na vykonávanie a tiež vykonávanými úlohami. Tabuľka má možnosť na zobrazenie iba tých úloh, ktoré daný používateľ sám spustil. V sekundovom intervale je možné vidieť priebeh vykonávania jednotlivej úlohy, resp. vykonávaných úloh.

V prípade, že sa vedecký pracovník rozhodne ukončiť vykonávanie úlohy, ktorú sám spustil, je možné vykonávanie zastaviť pomocou tlačidla Cancel. Ak sa úloha, ktorú chce používateľ zrušiť, už dokončila, tak je možné ju odstrániť z frontu úloh pomocou tlačidla Remove.

Doplňujúcou funkcionalitou pri zobrazení tabuľky je možnosť zmeniť formát zobrazovania tabuľky.

3.2 Analýza objektov a operácií

Ak chcú vývojári, aby používatelia pochopili princíp toho, ako bude výsledná aplikácia pracovať, je potrebné, aby bola vykonaná analýza objektov a operácií medzi týmito objekatami. To pomáha lepšiemu pochopeniu, tak zo strany používateľov, ako aj vývojárom a to pri zjednotení postupu tvorby.

Aplikácia

Prvým objektom je samotná webová aplikácia. Používateľ v nej môže monitorovať stav klastra v reálnom čase, môže spravovať úlohy vykonávané na klastroch, môže zistiť, ktorí používatelia sú práve pripojení v aplikácii, môže sa prihlásiť.

Obmon

Ďalší objekt je Obmon. Používateľ v môže v objekte Obmon môže monitorovať stav klastra v reálnom čase, pričom dáta sú poskytované v sekundovom intervale bez nutnosti aktualizovať celú stránku. Taktiež má používateľ možnosť výberu klastra, ktorého stav chce monitorovať.

Salsa

Nasledujúcim objektom je Salsa. V objekte môže používateľ monitorovať práve vykonávané úlohy na klastri, ale aj úlohy nachádzajúce sa vo fronte úloh na spustenie. Okrem monitorovania úloh má používateľ právo zastaviť vykonávanie úloh, ktoré on sám spustil. Ďalšou operáciou je spustenie vlastnej úlohy na klastri priamo z webovej aplikácie.

4 Návrh používateľského rozhrania

Používateľské rozhranie bolo vytvárané iteratívne, formou prototypovania v 3 iteráciách. V každej iterácii bolo vykonané overenie formou používateľského testovania. Ako formálny spôsob hodnotenia rozhrania sme využili štandardnú metodiku SUS [1]. Prototyp bol vytvorený v prototypovacom nástroji Figma.

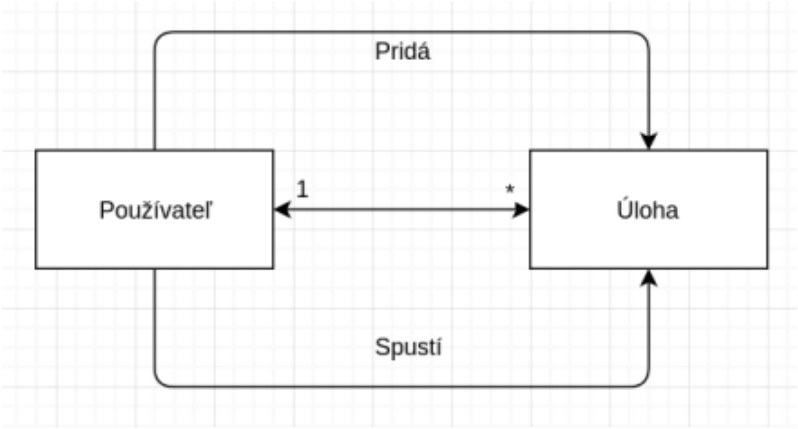
Konceptuálny model určuje spôsob, akým bude systém či produkt pracovať. Tento model obsahuje niekoľko dôležitých aspektov, medzi ktorými sú napríklad všetky akcie, ktoré môže koncový používateľ s daným produktom vykonávať, objekty, ktoré označujú určitú časť celého systému vzťahy medzi objektami a taktiež aj ich atribúty.

Úlohou tejto práce je umožniť koncovému používateľovi zrýchliť proces zadávania úloh, ktoré sa majú vykonať vo výkonnom klastri, na čo najmenší čas, a tým aj zvýšiť jeho produktivitu. Konceptuálny model pre tento projekt teda neobsahuje veľa objektov, ktoré sa zapájajú do procesu. Keďže používateľ pridáva, resp. spúšťa úlohu, tak sú potrebné dva objekty, a to objekt **používateľ** a objekt **úloha** (pozri obr. 4.1).

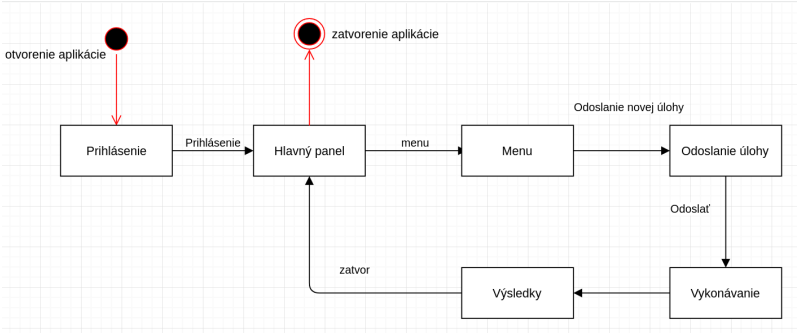
4.1 Návrh postupností obrazoviek

Pri vytváraní návrhu systému pomocou prototypovania je nutné vytvoriť obrazovky, ktoré sa budú používateľom zobrazovať postupne pri prechádzaní daným systémom. Webové rozhranie, ktoré je vytvárané v tejto práci, potrebuje niekoľko obrazoviek, medzi ktorými je napríklad obrazovka na prihlasovanie, obrazovka hlavného panelu, obrazovka na pridávanie úloh, obrazovka na prezeranie výsledkov z daných úloh a i. Na obrázku 4.2 je možné vidieť SSQ diagram obrazoviek, ktorý zobrazuje obrazovky rozhrania a taktiež aj prechody.

[1] System Usability Scale



Obr. 4.1: Diagram konceptuálneho modelu



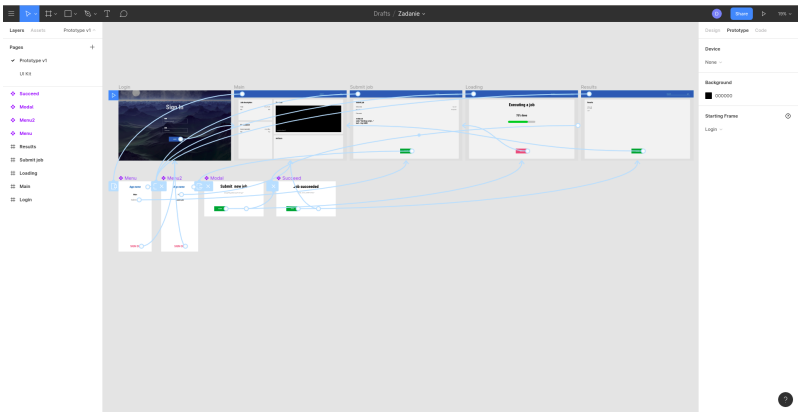
Obr. 4.2: SSQ diagram

4.2 Prvý prototyp

Za účelom vytvorenia prototypu bol použitý nástroj Figma. V rámci prototypu je potrebné zahrnúť to, akým spôsobom sa bude správať rozhranie pri určitých činnostiach vykonaných používateľom. Prototyp slúži na lepšiu ukážku toho, ako bude daný systém či produkt fungovať a mal by sa čo najviac približovať realite. Pomocou prototypu je možné zistiť nedostatky ešte predtým, ako budú reálne naprogramované, čiže ušetrí sa viac času pri vývoji.

Prototyp pre tento projekt začína prihlasovacou obrazovkou, pričom po prihlásení sa používateľ dostane do hlavného panelu, z ktorého môže buď zadať novú úlohu na spustenie alebo zistiť aktuálny stav klastra. V prípade, že používateľ chce pridať novú úlohu, bude požiadaný o zadanie súboru (skriptu), v ktorom je zadaná úloha na spustenie a taktiež bude po nahratí možné pozrieť obsah daného skriptu. Po odoslaní skriptu na spustenie sa začne vykonávať a po skončení

bude možné získať výsledky, ktoré bude možné priamo v tejto aplikácii prečítať. Snímku obrazovky z vyvíjania prototypu je možné vidieť na obr. 4.3. Bežiaci prototyp je možné vidieť na tejto adrese. Video z prototypu je možné vidieť na portáli YouTube.



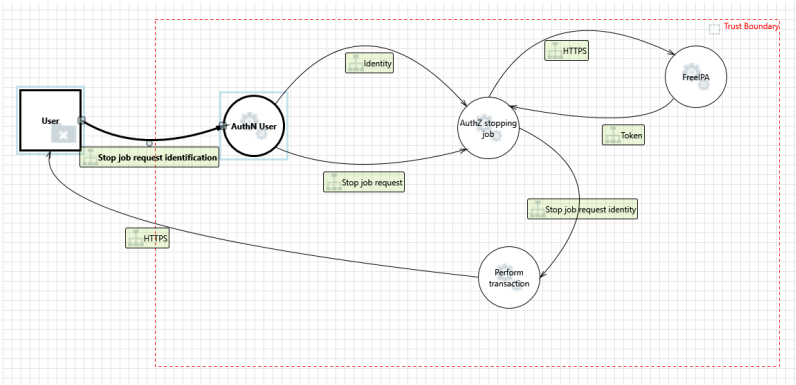
Obr. 4.3: Vývoj prototypu

4.3 Overenie prvého prototypu

Po vytvorení protypu systému je potrebné overiť jednoduchosť a používateľnosť používateľského rozhrania pomocou osôb, ktoré neboli pri vývoji systému, a teda nevedia, ako daný systém funguje. Tento projekt bol overený za pomoci troch osôb vo veku od 19 do 23 rokov, pričom žiadna z týchto osôb sa nevenuje oblasti, pre ktorú je tento systém vyvíjaný. Úlohou týchto osôb bolo spustiť úlohu a získať výsledky po vykonaní danej úlohy. Výsledky overenia prototypu sú, že tlačidlo menu by malo byť väčšie, pre spustenie úlohy je nutné otvorenie bočného menu aj keď by nemuselo byť a taktiež použitie veľkého množstva vyskakovacích okien. Z formulára SUS je zjavné, že tieto osoby by používali tento systém často, nie je zložitý a jeho ovládanie je jednoduché. Výsledná hodnota SUS návrhu je 87,5 čo je známka A, čo taktiež značí, že používatelia sú spokojní s používateľským rozhraním webovej aplikácie, a teda, že je dobre navrhnuté.

4.4 Analýza hrozieb

V rámci predmetu Bezpečnosť v počítačových systémoch bolo úlohou posledného zadania vyhotovenie bežpečnostnej analýzy bakalárskej práce, a teda tejto webovej aplikácie. Táto bezpečnostná analýza bola vytvorená pomocou programu **Microsoft Threat Modeling Tool 2016**, ktorý je možné bezplatne stiahnuť. Prvým krokom pri tvorbe analýzy bolo nakreslenie diagramu toku dát (angl. *data flow diagram*). Diagram pre túto webovú aplikáciu je zobrazený na obrázku 4.4.



Obr. 4.4: Diagram toku dát

Po nakreslení diagramu nastala analýza, pričom pre túto webovú aplikáciu sa ukázalo 19 hrozieb medzi ktorými boli hrozby ako získanie vyšších práv ako má daný používateľ, útočník sa môže vydávať za používateľa a získavať tak dôležité údaje, nedostupnosť služby, spúšťanie príkazov na diaľku bez autentifikácie a pod. V týchto 19 hrozieb bolo 18 zmiernených, respektíve úplne eliminovaných použitím tokenov JWT a zabezpečeného protokolu HTTPS.

5 Implementácia

Táto kapitola sa venuje implementácii webového rozhrania pomocou JavaScript knižnice React. Pri tvorbe rozhrania sa bude vychádzať z návrhu z predošlej kapitoly.

5.1 Tvorba webového rozhrania

Na začiatku je potrebné vytvoriť projekt a na to je možné použiť **create-react-app**, za pomoci čoho vieme vytvoriť základ React projektu bez akéhokoľvek nastavovania. Projekt môžeme vytvoriť pomocou príkazu:

```
npx create-react-app project
```

Project je v tomto prípade meno celého projektu. Okrem samotnej React knižnice budeme potrebovať aj niekoľko ďalších balíčkov, ktoré budú opísané v ďalších častiach.

5.1.1 Štruktúra projektu

Pre zachovanie konzistentnej hierarchie projektu sme sa rozhodli rozdeliť celú aplikáciu podľa metodológie **Atomic Design**, a teda komponenty rozdelíme do štyroch z pôvodných piatich úrovní (keďže šablóny nemajú v React projekte zmysel) a pre každú úroveň vytvoríme vlastný priečinok.

- `atoms/` - obsahuje najmenšie komponenty, ktoré sa už viac nedajú rozdeliť
- `molecules/` - sú to komponenty, ktoré sa skladajú z menších komponentov nachadzájucich sa v priečinku `atoms/`
- `organisms/` - sú to komponenty, ktoré sú zložené z atómov a molekúl

- `pages/` - sú to komponenty, ktoré sú celými stránkami, a teda obsahujú všetky potrebné komponenty

Okrem rozdelenia komponentov do týchto priečinkov sme taktiež vytvorili priečinky pre knižnicu **redux**. Bola vytvorená takáto štruktúra priečinkov:

- `actions/` - sú v ňom súbory obsahujúce funkcie, ktoré vytvárajú akcie, ktorými je aktualizovaný celkový stav
- `actionTypes/` - obsahuje súbory s konštantami typov akcií
- `reducers/` - súbory s funkciami, ktoré sa starajú o zmenu stavu
- `selectors/` - súbory obsahujúce funkcie, ktoré slúžia na vrátenie požadovanej časti z celého stavu
- `store/` - súbor, ktorý obsahuje samotný stav celej aplikácie

Mimo týchto priečinkov sa nachádza ešte niekoľko ďalších súborov, z ktorých najdôležitejšie sú **App.js** a **index.js**. Prvý menovaný súbor je hlavný komponent celého projektu, ktorý obsahuje všetky ostatné komponenty. Druhý súbor **index.js** slúži na vykreslenie celého projektu, presnejšie hlavného komponentu, čo je v tomto prípade App do niektorého elementu DOM. V tomto súbore je vhodné zaobaliť hlavný komponent inými komponentmi, ktoré dodajú celej aplikácii novú funkcionalitu. V tomto projekte bude použitých niekoľko takýchto komponentov a tieto budú vysvetlené ďalej v tejto práci.

5.1.2 Vytvorenie navigácie v aplikácii

Keďže chceme, aby sa v aplikácii dalo navigovať medzi viacerými stránkami, tak potrebujeme vytvoriť navigáciu, na čo je možné použiť balíček **react-router**. Po nainštalovaní tohto balíka je potrebné celý projekt, teda komponent App zaobaliť do špeciálneho komponentu **HashRouter** alebo **BrowserRouter**. Ako uvádza zdroj [29], rozdiel medzi týmito dvoma komponentmi je v tom, ako vyzerá výsledná URL pri prechádzaní medzi jednotlivými komponentmi. **BrowserRouter** využíva klasický zápis adries (napr. *http://www.priklad.sk/podstranka*), pričom pri tomto type je potrebné nakonfigurovať server, ktorý poskytuje danú stránku, aby

všetky podstránky ukazovali na daný projekt. Na druhej strane, **HashRouter** používa tvar URL adresy taký, že primárna časť adresy s doménou vyzerá rovnako, no zvyšok URL (teda podstránky) sa nachádzajú za znakom mriežky. Takýto tvar adresy umožňuje ovládanie navigácie v aplikácii bez toho, aby bolo potrebné špeciálne konfigurovať server. Práve kvôli tejto výhode bude v tomto projekte použitý **HashRouter**.

Pre lepšiu zrozumiteľnosť sme importovali tento komponent tak, že sme ho zároveň premenovali na **Router**. Po tom, ako sme komponent *App* zaobalili, musíme zadefinovať cesty a taktiež komponenty, ktoré sa zobrazia. Na tento účel slúži komponent **Route**, pričom tieto komponenty musia byť zaobalené ešte v komponente **Switch**, ktorý zobrazí prvý komponent, ktorý súhlasí z aktuálnou cestou. V tomto projekte je niekoľko komponentov, ktoré sa nachádzajú na týchto adresách:

- `/` - hlavná stránka
- `/profile` - stránka s profilom prihláseného používateľa
- `/auth` - stránka s prihlasovacím formulárom
- `/api/info` - stránka so zobrazením informácií, rovnaká ako hlavná stránka
- `/obmon/:service?/:sub?` - stránka so zobrazením údajov zo senzora Obmon, ktorý monitoruje stav klastrov, pričom tieto dáta sú zobrazované vo forme tabuľky
- `/salsa/:service?/:sub?` - stránka so zobrazením údajov zo systému SALSA, cez ktorý je možné spravovať úlohy

Premenné **service** a **sub**, pred ktorými sa nachádza v ceste dvojbodka, umožňujú vytvárať dynamickú URL, pričom tieto premenné budú obsahovať názvy z aktuálnej cesty. Otáznik za týmito premennými značí, že tieto premenné sú nepovinné, a teda cesta nemusí mať presne takýto tvar.

V prípade, že je potrebné sa vrátiť na predchádzajúcu stránku v aplikácií, je potrebné použiť históriu prehliadania. Tieto informácie je možné získať pomocou fukcie taktiež z balíčka *react-router* s názvom **useHistory**. Táto nám vráti objekt, s ktorým je možné manipulovať a v prípade, že je potrebné sa vrátiť na poslednú prehliadanú podstránku, tak je možné zavolať funkciu **push** z tohto objektu a s argumentom adresy v relatívnom tvare (t.j. / je domovská podstránka).

5.1.3 Vytvorenie globálneho stavu aplikácie

S knižnicou je možné vytvoriť 2 typy komponentov, a to funkcionálny komponent a komponent vytvorený pomocou triedy. Tento druhý vymenovaný komponent mal donedávna výhodu, že mal možnosť si udržať vlastný stav vo svojej vlastnej triede, čo pri funkcionálnom komponente nebolo možné. Od verzie knižnice React 16.8, ako je uvedené v zdroji [30], je možné používať tzv. háky (angl. *hooks*), ktoré umožňujú mať vlastný stav aj pre funkcionálne komponenty. Táto nová možnosť bude použitá aj v tomto projekte.

Keďže vo webovej aplikácii môže byť veľa komponentov a tie potrebujú medzi sebou často posielať dáta, React podporuje posielanie dát cez atribúty komponentov (angl. *props*). Čím viac dát treba posielať medzi komponentmi, tým neprehľadnejším sa kód stáva. Na vyriešenie tohto problému vznikol balík **Redux**, ktorý umožňuje mať globálny stav aplikácie (angl. *store*) a každý komponent môže pracovať s dátami z tohto globálneho stavu.

Tento globálny stav je vytvorený tak, že komponenty nemôžu do tohto stavu priamo zapisovať nové dáta, pretože z tohto stavu je možné iba čítať, a teda komponenty musia odoslať akciu, čo je objekt, ktorý by mal obsahovať minimálne typ a poprípade aj dáta. Takáto štruktúra objektu je často a takmer vždy vhodná na použitie. Tieto akcie prijímajú funkcie nazývané reduktory (angl. *reducers*), ktoré na základe typu akcie vrátia nový stav. Pre minimalizovanie chýb pri písaní typov akcii boli vytvorené konštanty. V nasledujúcej časti budú popísané všetky súčasti tohto globálneho stavu.

Akcie

Všetky funkcie, ktoré vytvárajú akcie sa nachádzajú v priečinku *actions*, pričom je tam niekoľko súborov. V tejto časti budú všetky tieto súbory opísané. Prvým súborom je **auth.js**, ktorý obsahuje akcie spojené s prihlasovaním. Akcia `loadingLogin` slúži na oznamenie komponentom, že práve prebieha autentifikácia, `authError` oznámi, že počas autentifikácie nastala chyba a `removeToken` odhlási autentifikovaného používateľa.

Ďalším súborom v tomto priečinku je súbor **salsa.js**. Tento súbor obsahuje funkcie, ktoré vytvárajú akcie, ktoré sú potrebné pre špecifikáciu dát zo systému SALSA, ktoré chceme sledovať a získavať. Funkcia `onSalsaNameChange` slúži na

nastavenie skupiny klastrov, od ktorej chceme počúvať dáta. Spolu s názvom skupiny klastrov je taktiež potrebné vedieť názov klastra, na ktorom chceme počúvať, to je možné pomocou akcie, ktorá je vytvorená funkciou `onSalsaSubChange`. Poslednou dôležitou funkciou, ktorá je potrebná na zrušenie počúvania nových dát je funkcia `onSalsaReset`.

Tretím súborom medzi akciami je **websocket.js**. Jeho jedinou funkciou je `writeToSocket`, pomocou ktorej je možné poslať dáta cez WebSocket do API. Táto funkcia, resp. akcia je veľmi dôležitá, keďže ona zabezpečuje informovanie API o tom, aké dáta je potrebné posielať klientovi cez WebSocket.

Posledným súborom je **zmq2wsServiceInfo.js**, ktorý obsahuje akcie podobné ako v súbore **salsa.js**. Keďže tento projekt bude prijímať dáta aj iné ako od systému SALSA, tak je potrebné zabezpečiť dynamickosť pri výbere z ponuky klastrov, od ktorých chce táto webová aplikácia prijímať aktualizácie. Funkcie `getNameService` a `getSubService`, ktorých účelom je nastaviť názov skupiny klastrov a tiež názov daného klastru, od ktorého chce webová aplikácia prijímať dáta. Akcia, ktorá je vygenerovaná funckiou `getSrc` slúži na zastenie toho, že prijímame dáta od správneho zdroja, keďže pri prechádzaní medzi viacerými klastrami je možné, že API pošle webovej aplikácií ešte dáta od klastra, od ktorého sa už predtým odpojil a teraz požaduje dáta od iného klastra. V prípade, že sa webová aplikácia odhlásila od počúvania dát od klastra, je potrebné, aby sa dáta zobrazené v tabuľke vymazali. Na tento účel je pripravená funkcia `getData`, ktorá nastaví dáta, ktoré sú zobrazené, čo v prípade, že aplikácia sa odhlásila od počúvania, je prázdne pole, a teda žiadne dáta nebudú v tabuľke zobrazené. Ďalším prípadom, kedy by tabuľka mohla ukazovať zlé dáta, je v prípade presunutia sa na časť webovej aplikácie, ktorá nesúvisí s počúvaním na dáta od API. V tom prípade je potrebné, aby sme mali istotu, že pri odchode sa odhlásime, tak bola vytvorená akcia pomocou funkcie `getIsSubscribed`, ktorá nastaví, či je webová aplikácia práve prihlásená na počúvanie dát od nejakého klastra. V prípade odchodu z časti stránky, kde sú dáta zobrazované, sa zavolá aj táto akcia a nastaví sa na hodnotu **false**, teda, že webová aplikácia nepočúva na žiadne dáta.

Reduktory

Po vyvolaní akcií, je potrebné ich nejakým spôsobom zaznamenať a zmeniť globálny stav. Túto funkcionalitu zabezpečujú funkcie nazývané reduktory. Keďže v

tomto projekte je veľa rozdielnych druhov akcií, tak je vhodné celkový reduktor rozdeliť na menšie, ktoré spracovávajú určité akcie, pričom tieto menšie reduktory sa neskôr spoja do jedného celkového reduktora pomocou funkcie *combineReducers*. V tomto projekte sú reduktory rozdelené do súborov podobne ako akcie s tým rozdielom, že tieto sú dostupné v priečinku *reducers*.

Prvým súborom je **auth.js**, ktorý riadi to, ako sa zmení globálny stav podľa akcií, ktoré sú používané pri prihlasovaní. Ako každý rekuktor, tak aj tento obsahuje základný stav, ktorý má aplikácia.

Nasledujúcim súborom je súbor **salsa.js**, ktorý obsahuje funkciu na spracovanie akcií pre prácu so systémom SALSA, pričom tieto zahŕňajú nastavenie dát v prípade novej zmeny, resetovanie dát, nastavenie názvu skupiny klastrov, od ktorej chceme získavať aktualizácie, nastavenie názvu daného klastra a taktiež v prípade zmeny skupiny klastrov, na ktorú aplikácia počúva je potrebné zmeniť informácie o aktuálnej skupine klastrov. Okrem týchto akcií sú tu zahrnuté aj akcie z balíku **@obl/redux-websocket-middleware**, ktorý bol vytvorený práve pre tento účel a akcia z tohto balíku spravuje akciu prijatia dát z WebSocketu.

Tretím súborom je **websocket.js**, ktorý má na starosti riadenie zmeny stavu a poskytovanie dát pre tabuľky v tomto projekte na zobrazenie informácií z klastrov. Prichádzajúce informácie sa ešte kontrolujú na typ, ktorý je zadaný v prichádzajúcich dátach na zistenie, aké informácie práve prišli. Pri tomto reduktore používame konštanty typov akcií, ktoré sú dostupné v balíčku **@obl/redux-websocket-middleware**, ktorý bol vytvorený práve pre tento účel.

Posledným súborom je **zmq2wsServiceInfo.js**, ktorý slúži podobne ako súbor **salsa.js** na nastavovanie informácií o skupine klastrov, samotných klastroch, prijímaných dátach a taktiež, či aplikácia práve počúva nejakú aktualizácie od nejakého klastra alebo nie.

Selektory

Keďže zvyčajne komponenty nepotrebujú mať dostupný celý globálny stav, ktorý môže byť veľmi veľký, tak sú použité funckie nazývané selektory, ktoré majú za úlohu vrátiť len konkrétnu časť globálneho stavu. Signatúra týchto funkcií je vždy rovnaká a to, že každý selektor berie ako argument celý globálny stav a vracia časť z celého objektu tohto stavu. Tak ako v predchádzajúcich častiach, tak aj tieto funckie sú rozdelené do súborov podľa toho, na čo slúžia. Nachádzajú sa v priečinku

selectors.

Rovnako ako reduktory a akcie, aj selektor má súbor **auth.js**, ktorý obsahuje funkcie, ktoré slúžia na získanie chybovej správy pri prihlasovaní, prihlasovacieho tokenu, používateľských dát a taktiež na zistenie, či aplikácia sa práve snaží prihlásiť.

Selektor **salsa.js** má za úlohu získať názov skupiny klastrov, názov klastra a taktiež URL adresu sprostredkovateľa (angl. *redirector*).

Súbor **tableDataWebsocket.js** úzko spolupracuje s reduktorom v súbore **tableWebsocket.js**, takže selektory v tomto súbore majú za úlohu získať z globálneho stavu dáta pre tabuľkový výpis v aplikácii, taktiež názov klastra, od ktorej chceme získavať informácie pre tabuľku.

Ďalším súborom je **websocket.js**, ktorý dodáva informácie o aplikácií a API, informuje o pripojení cez WebSocket a tiež o identifikátoru WebSocket spojenia.

Posledným súborom je **zmq2wsServiceInfo.js**, pričom tento vie poskytovať informácie o klastri, ktorá je práve počúvaná. Teda pomocou tohto selektora je možné získať názov skupiny klastrov, názov klastra, zdroj tohto klastra (napríklad či tieto dáta pochádzajú zo systému SALSA alebo z Obmon senzorov), dáta od tohto klastra a taktiež, či aplikácia počúva na zmeny daného klastra.

Aplikačné programové vybavenie

Redux požaduje, aby akcie boli objekty, na ktoré nesmú byť aplikované žiadne asynchrónne volania na API, ani žiadne iné funkcie. Keďže v tomto projekte je potrebné prihlasovanie a na tento účel je potrebné robiť API volania, tak je potrebné istým spôsobom povoliť, aby akcie mohli byť aj funkcie. Presne k tomuto účelu slúži balíček **redux-thunk**, ktorý nám umožní mať takúto funckionalitu. Okrem tohto aplikačného programového vybavenia je v tomto projekte použitá aj funkcia **createWebsocketMiddleware** z balíčka **@obl/redux-websocket-middleware**.

Globálny stav

Najdôležitejšou časťou celého reduxu je samotný globálny stav (angl. *store*). Vytvorenie tohto stavu je uložené v priečinku **store** v súbore **configureStore.js**.

V tomto súbore sú spojené všetky reduktory do jedného celku pomocou funckie *combineReducers*. Token, ktorý označuje používateľa za prihláseného je po celý čas uložený v *localStorage*. Tento token je zašifrovaný pomocou JWT a obsahuje

aj čas expirácie, teda po uplynutí času sa stáva token neplatným a to používateľa odhlási.

Pomocou funkcie *applyMiddleware* je možné použiť nami požadované midlvéry v globálnom stave. Návratová hodnota tejto funkcie je posunutá spolu s celkovým reduktorom a základným globálnym stavom do funkcie **createStore**, ktorá už vráti celkový globálny stav.

Ak chceme, aby všetky komponenty mali možnosť sa dostať ku dátam z tohto stavu, tak je potrebné celú aplikáciu (komponent *App*) zaobaliť podobne ako pri *react-router* pomocou obaľovacieho komponentu, ktorým je v tomto prípade komponent **Provider**. Tento komponent berie ako atribút *store*, ktorého hodnota je výstup z funckie *createStore*.

Na to, aby komponent mohol získať dáta z globálneho stavu, je potrebné, aby komponent použil funkciu *useSelector*, ktorá berie ako argument nami vytvorený selektor. To slúži na získanie časti z celkového stavu. Ak je potrebné, aby komponent mohol odosielať do reduktorov akcie, potrebuje špeciálnu funkciu na odoslanie (angl. *dispatch*). Funkcia, ktorá je potrebná, sa volá *useDispatch*, pričom táto, aj predošlá funkcia sa nachádza v balíčku **react-redux**. Funkcia *useDispatch* nám vráti inú funkciu, ktorá odosiela samotné akcie. Táto prijíma ako argument objekt akcie alebo funkciu, ktorá sa má zavolať.

5.1.4 Vytvorenie grafického rozhrania

Pre vytvorenie pekného grafického rozhrania sme sa rozhodli použiť UI softvérový rámec pre React s názvom **Material-UI**. Tento rámec poskytuje vzhľad v štýle Material Design, čo je známe najmä pre používateľov smartfónov s operačným systémom Android.

Tento balíček poskytuje komponenty, ktoré sú postavené na klasických HTML elementoch, pričom im pridávajú špeciálne štýlovanie, ktoré je neskôr možné samozrejme meniť.

Medzi ďalšie výhody tohto rámca môžeme zaradiť napríklad aj to, že bol vytvorený najmä pre mobily, teda je responzívny.

5.2 Prepojenie webovej aplikácie s API

Jednou z najdôležitejších funkcionalít, ktoré táto webová aplikácia potrebuje, je prepojenie s API. Bez tohto aplikačného rozhrania by nebolo možné získavať žiadne dáta a stránka by bola zbytočná.

Pri implementácii tejto webovej aplikácie sme sa museli rozhodovať o tom, ako prepojiť túto aplikáciu s API. Na výber boli dve možnosti, prvou možnosťou bol protokol HTTP a druhým bol protokol WebSocket.

Protokol HTTP je bežne používaný pri webových aplikáciách, no jeho problémom je, že tento protokol poskytuje len polovičný duplex, čo v komunikácii znamená, že v jednom čase je možné posielať dáta len jedným smerom. Reálne to znamená, že nie je možné, aby klient mohol poslať požiadavku v čase, keď server mu posiela odpoveď na predchádzajúcu požiadavku, teda klient musí čakať, kým mu nepríde odpoveď na predchádzajúce požiadavky od servera. Ďalšou nevýhodou je vzor, akým tento protokol funguje. Každú komunikáciu klienta so serverom musí začať klient, čo je nevýhodou napríklad pri získavaní nových dát automaticky bez potreby posielať požiadavku.

Na druhej strane je protokol WebSocket. Tento protokol je možné použiť vo väčšine moderných prehliadačov, pričom rieši mnohé nedostatky protokolu HTTP. Je dôležité spomenúť, že tento protokol podporuje plný duplex, a teda klient aj server môžu posielať dáta v tom istom čase. To znamená to, že klient nemusí čakať, kým mu príde odpoveď na jeho požiadavku od servera. Ďalšou výhodou je, že klient nie je jediný, kto môže začať spojenie. To je veľkou výhodou najmä pre aplikácie, ktoré potrebujú dáta v reálnom čase.

Kvôli tomu, že protokol WebSocket podporuje plný duplex a tiež, že server môže začať komunikáciu sme sa rozhodli použiť práve tento protokol. Aktualizované informácie sú posielané z API servera každú sekundu, pričom samotná webová aplikácia posiela požiadavky na odoberanie informácií a tiež zrušenie odberu.

5.3 Informácie zo senzorov Obmon

Získavanie informácií o klastri je možné zo sensorov systému Obmon. Tieto informácie sú vo webovej aplikácii zobrazené vo forme tabuľkového výpisu, pričom

každý riadok obsahuje informáciu o každom uzle, resp. klastri. Na informovanie používateľa webovej aplikácie bolo taktiež vytvorené farebné rozlíšenie niektorých buniek tabuľky, ktoré označujú stav, kedy sa meraná hodnota blíži ku svojmu maximu.

Pre samotným zobrazením tabuľky s dátami, je nutné, aby si používateľ vybral to, od ktorého klastra a z ktorej skupiny klastrov chce aktualizácie dostávať. Zoznam všetkých skupín klastrov je získaný pomocou vytvorenia požiadavky z webovej aplikácie, pričom táto je vytvorená a odoslaná na API server hneď, ako používateľ navštívi podstránku Obmon. Po prijatí poľa so skupinami klastrov je možný výber skupiny klastrov, od ktorej chce používateľ získavať nové dáta.

Po výbere klastra, od ktorého chce používateľ sledovať informácie, sa zobrazí tabuľka. Tabuľka obsahuje viacero riadkov, pričom každý riadok označuje jeden uzol v danom klastri. Podľa toho, či klaster obsahuje uzly s grafickými výpočtovými jednotkami alebo nie sú zobrazené stĺpce s informáciami o grafických kartách. Prvým stĺpcom v tabuľke je identifikátor uzlu, pričom tento označuje iba poradie zobrazených uzlov. Druhý stĺpec obsahuje mená uzlov, pričom tento údaj je známy aj ako meno hosťa (angl. *hostname*). Ďalšou zobrazenou informáciou je počet jadier daného uzlu. Nasleduje stĺpec s informáciou o celkovom zaťažení uzlu. Nasledujú stĺpce s údajmi o výpočtových jednotkách, pričom tu sa nachádzajú informácie, ktoré označujú využitie uzlu systémom, používateľom, využitie používateľom, pri ktorých bola zmenená priorita procesu a tiež to, ako veľmi boli výpočtové jednotky nečinné z dôvodu operáciu vstupu alebo výstupu. Posledným stĺpcom z tejto časti je stĺpec s informáciou o tom, akú dlhú dobu bol uzol nečinný.

Ďalšou časťou tabuľky sú informácie o využití dočasnej pamäte RAM. Táto sekcia pozostáva z troch stĺpcov, pričom prvý označuje množstvo používanej pamäte, druhý označuje, aká veľká časť disku je kešovaná, slúžiace pre rýchlejší prístup k dátam a posledný stĺpec obsahuje informáciu o celkovej veľkosti pamäte.

Štvrtou časťou tabuľky je informácia o práci s diskami. Tu môže používateľ nájsť informáciu o tom, aká je aktuálna rýchlosť zápisu a čítania z diskov. Okrem tejto celkovej informácie je možné zistiť danú informáciu v rámci jednotlivých diskov tak, že používateľ prejde kurzom cez danú bunku.

Ďalšia (v niektorých klastroch aj posledná) časť označuje prácu so sieťou. Prvý stĺpec označuje rýchlosť sťahovania a druhý označuje rýchlosť posielania. Rovnako

ako v predchádzajúcej časti, aj v týchto bunkách je možné získať informácie o jednotlivých sieťových rozhraniach tak, že používateľ prejde kurzorom ponad dané bunky.

V prípade uzlov, v ktorých sa nachádzajú grafické jednotky, sa v tabuľke nachádza ešte jedna časť. Táto časť informuje používateľa práve o grafických kartách. Prvým stĺcom je informácia o počte grafických jednotiek v rámci uzlu, nasleduje informácia o využití grafických kariet, využití pamäte grafických kariet a tiež celková veľkosť pamäte. V prípade týchto troch informácií sa môže používateľ informovať aj o stave jednotlivých grafických kariet pomocou toho, že prejdú nad danou bunkou kurzorom.

Okrem samotných uzlov sú v tabuľke ďalšie dva riadky, ktoré označujú celkové informácie a priemerné. V prípade celkových dát sú zobrazené všetky dáta okrem informácií o výpočtových a grafických jednotkách. Informácie s priemernými údajmi obsahuje všetky položky okrem informácií o grafických jednotkách.

Používateľ si môže všimnúť, že niektoré bunky sú zafarbnené, pričom ich priehľadnosť je rôzna. Pre rýchlejšiu identifikáciu dát je možné pozrieť len na priehľadnosť, resp. nepriehľadnosť danej bunky. Bunka, ktorá je viac priehľadná, teda jej farebnosť je slabšia, označuje, že daný údaj je bližší ku minimu. Naopak, v prípade, že bunka je nepriehľadná, označuje, že daný údaj sa blíži ku svojmu maximu.

5.4 Informácie zo systému SALSA

Keďže je potrebné, aby táto webová aplikácia bola schopná zadávať, spúšťať a ovládať úlohy, tak je nutné, aby vedela táto aplikácia komunikovať s dávkovacím systémom SALSA. Úlohy, ktoré sú práve vykonávané je možné vidieť na stránke v rozbaľovacom zozname. Okrem informácií o úlohách je v tomto zozname aj informácia o aktuálnej verzii systému SALSA, počet uzlov a tiež URL adresa manažérskeho uzlu klastra. Rovnako ako pri tabuľkovom výpise zo senzorov Obmon, aj v tabuľke úloh sú niektoré bunky farebne označené na indikáciu priebehu spúšťania jednotlivých úloh.

Pred samotným zobrazením rozbaľovacieho zoznamu, je nutné, aby si používateľ vybral to, od ktorého klastra a z ktorej skupiny klastrov chce informácie o úlohách dostávať. Zoznam všetkých skupín klastrov a tiež jednotlivých klastrov je získaný za pomoci API servera cez WebSocket spojenie.

Po výbere klastra, od ktorého chce používateľ sledovať spustené úlohy, sa zobrazí rozbaľovací zoznam. Tento zoznam obsahuje viacero údajov, pričom najprv je možné vidieť informáciu o verzii, celkovom počte jadier. Ďalšou položkou v rozbaľovacom zozname je fronta úloh (angl. *job queue*). Tu je zobrazená tabuľka, pričom každý riadok obsahuje dávku úloh, ktorá pozostáva z mnohých menších úloh. Tabuľka pozostáva z viacerých stĺpcov.

Prvý stĺpec obsahuje identifikátor dávky úloh, ktorý ich jednoznačne identifikuje. Druhým stĺpcom je identifikátor používateľa, čo je možné použiť pri zisťovaní, ktoré úlohy boli spustené daným používateľom. Ďalší stĺpec obsahuje identifikátor skupiny, z ktorej používateľ spustil danú dávku úloh. Nasleduje stĺpec s počtom úloh, čo označuje to, koľko úloh obsahuje každá dávka úloh. Ďalšie 4 stĺpce obsahujú informácie o počete úloh, ktoré čakajú na spustenie, ktoré sú práve spustené, koľko úloh je dokončených a koľko úloh neskončilo správne. Tieto stĺpce sú tiež zafarbené pre rýchlejšiu identifikáciu progresu vykonávania dávky úloh. Stĺpec s čakajúcimi úlohami (angl. *pending*) je zafarbený modrou farbou, čím priehľadnejšia je bunka, tým menej úloh ostáva na vykonanie. Stĺpec s bežiacimi úlohami (angl. *running*) je zafarbený oranžovou farbou, pričom platí, že čím viac úloh práve beží, tým nepriehľadnejšia je bunka. Ďalšími stĺpcami sú stĺpec s počtom dokončených úloh a stĺpec s počtom úloh, ktoré skončili neúspechom. Používateľ môže vidieť stĺpec s dokončenými úlohami zafarbený zelenou farbou, pričom jeho priehľadnosť určuje ich počet v pomere ku všetkým úlohám v rámci dávky úloh.

Ďalšie stĺpce obsahujú informácie o tom, koľko percent zo všetkých úloh je už dokončených (úspešne aj neúspešne), kedy bola dávka úloh spustená, ako dlho sa už úlohy vykonávajú, aký je odhadovaný čas do konca vykonávania a taktiež tu je stĺpec s časom ukončenia, pričom v prípade nedokončenej dávky úloh je tu ukázaný odhadovaný čas dokončenia. Čas začatia a čas ukončenia sú v relatívnom formáte, no je možné, aby používateľ videl presný dátum a čas za pomoci kurzora a to tak, že prejde kurzorom nad danú bunku.

Posledným stĺpcom je stĺpec s akciou. V týchto bunkách sa nachádzajú tlačidlá, pomocou ktorých je možné zrušiť, resp. vymazať danú dávku úloh z fronty. V prípade, že skupina úloh ešte nie je ukončená, tak toto tlačidlo bude obsahovať text *Cancel* (z angl. *zrušiť*). Druhou možnosťou je, že dávka úloh už je dokončená, v tom prípade sa zobrazí text *Remove* (z angl. *odstrániť*).

Okrem samotných riadkov tabuľky s jednotlivými dávkami úloh sa tu nachádzajú ešte riadky o priemerných informáciách o úlohách a tiež celkové informácie, teda celkový počet úloh, celkový počet výkonávaných a dokončených úloh. V tomto riadku sa tiež nachádza tlačidlo akcie, pomocou ktorého je možné zrušiť a odstrániť všetky úlohy.

Tlačidlá akcií pri úlohách sú zobrazené iba tým používateľom, ktorí spustili danú dávku úloh. To zaručuje, že nikto iný nemôže manipulovať s úlohami ako ten, kto ich spustil. Tlačidlo akcie v riadku s celkovými informáciami je zobrazené každému prihlásenému používateľovi, pričom to sa v budúcnosti zmení, pričom je plánované použitie rolí, ktoré by určovali, ktoré skupiny používateľov majú právo zrušiť všetky dávky úloh.

Poslednou položkou rozbaľovacieho zoznamu je časť s generovaním príkazu. Používateľovi je umožnené si napísať vlastný príkaz na vykonanie a tiež je možné zadefinovať počet spustení tohto príkazu. Po zadaní potrebných údajov o úlohe je možné skopírovať vygenerovaný príkaz z dolnej časti obrazovky a tento už je možné spustiť na klastri. Táto časť obsahuje aj tlačidlo *Execute job* (z angl. *vykonaj úlohu*), pričom po stlačení tohto tlačidla sa pošle HTTP požiadavka na API server, ktorý následne spustí danú úlohu na klastri.

6 Dosiahnuté výsledky

V tejto práci bolo potrebné implementovať aplikáciu na správu úloh vykonávaných na klastroch a taktiež monitorovanie týchto klastrov. Táto časť práce je venovaná overeneniu toho, či tento projekt spĺňa požiadavky, ktoré boli spomenuté na začiatku tejto práce, porovnanie manipulácie s klastrom pomocou tejto webovej aplikácie a klasickým príkazovým riadkom a tiež overenie jednoduchosti manipulácie s touto webovou aplikáciou.

6.1 Naplnenosť požiadaviek

Keďže táto webová aplikácia mala niekoľko požiadaviek, ktoré bolo potrebné splniť, v tejto časti bude zodpovedané to, ktoré požiadavky boli splnené. Táto sekcia je rozdelená na požiadavky, ktoré sú nutné pre splnenie zadania a požiadavky, ktoré je možné v budúcnosti implementovať.

6.1.1 Nutné požiadavky

Prvou požiadavkou bolo, že používateľ webového rozhrania má možnosť monitorovať klastre. Táto požiadavka je splnená, pričom používateľ dostáva aktualizované informácie v sekundovom intervale. Webová aplikácia aktualizuje dáta v tabuľke bez toho, aby bolo nutné aktualizovať celú stránku aplikácie.

Druhou požiadavkou bolo, aby používateľ bol schopný manipulovať s dávkami úloh, teda aby mohol spúšťať, zrušiť a monitorovať jednotlivé dávky úloh. Táto požiadavka je taktiež splnená. Podobne ako pri požiadavke s monitorovaním klastra sú používateľovi poskytované informácie o jednotlivých dávkach úloh vo fronte so sekundovou aktualizáciou. Pri spustených úlohách je možné vidieť pod identifikátorom dávky úloh aj ukazateľ postupu a pri úspešne a neúspešne dokončených úlohách z jednotlivých dávok je možné po prejdení kurzorom nad bunkou

vidieť aj jednotlivé identifikátory úloh v rámci dávky. V prípade spúšťania úloh je vytvorená funkcia a koncový bod na API serveri, ktorý zabezpečuje spustenie úlohy na klastri.

6.1.2 Budúce požiadavky

V tejto časti sú vymenované ďalšie vlastnosti, ktoré je možné v budúcnosti v tejto webovej aplikácii doimplementovať.

Vizualizácia fyzikálnych objektov

Zobrazenie grafov a histogramov vo webovom prostredí. To znamená, že po importovaní JavaScript implementácie ROOT (JSRoot - `https://root.cern.ch/js/`). Prostredie na vytvorenie rôznych možností zobrazenia týchto grafov a histogramov. Taktiež možnosť zobrazenia detektorov a dráh častíc.

Interaktívne spúšťanie úloh a zobrazenie výsledkov

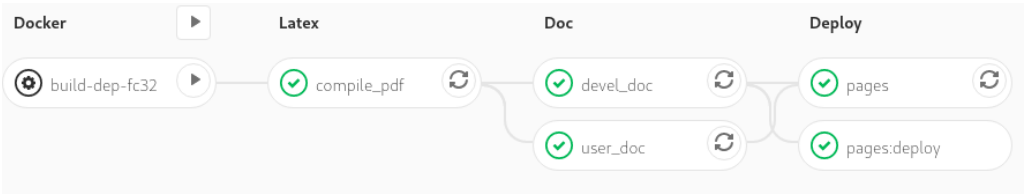
To znamená, že používateľ si vytvorí jedno až dvojrozmerný priestor a označením určitých bodov sa mu dynamicky spustia úlohy a vytvoria sa projekcie pre daný bod. Výsledok je potrebné dynamicky zobrazovať. Možné implementácie sú také, že za pomoci pohybu kurzoru dynamicky spustí úlohy a dané výsledky uloží do pamäte.

Ukladanie výsledkov do pamäte pre rýchle zobrazovanie

Výsledky z predošlých úloh budú uložené v pamäti a budú dostupné pre rýchle zobrazenie. Je potrebné vytvorenie algoritmu pre ukladanie a premapovanie do vnútorného úložiska prehliadača.

Vytvorenie prostredia na zreťazovanie úloh

Princíp tohto zreťazovania je podobné ako napríklad v Gitlab CI/CD Pipelines, ktorého ukážku je možné vidieť na obrázku 6.1. Úlohy budú vykonávané za sebou, pričom budú na seba nadväzovať.



Obr. 6.1: Zreťazovanie úloh v rámci Gitlab CI/CD

Monitorovanie procesov

Monitorovať CPU informácie a informácie o pamäti všetkých podprocesov úlohy. Proste implementacia top prikazu. V prípade monitorovania klastra by mohla byť možná implementácia tak, že každý riadok tabuľky bude možné rozkliknúť a zobrazia sa podrobné informácie o každom spustenom procese. Výstup by mohol byť podobný výstupu príkazu `top` v systéme založených na jadre Linux. V prípade monitorovania úloh na klastri by možná implementacia mohla byť taká, že každá úloha, teda riadok v tabuľke, by mohol byť klikateľný a po rozkliknutí by sa zobrazili informácie o všetkých podúlohách, teda podprocesoch danej úlohy.

Nasadenie na Kubernetes

Vytvorenie klastra na orchestračnom systéme Kubernetes pre jednoduchú škálovateľnosť. Práca so systémom Kubernetes je jednoduchá a preto by bolo vhodné použiť práve túto technológiu na nasadenie.

6.2 Overenie webovej aplikácie

Cieľom tejto webovej aplikácie bola vytvoriť rozhranie, pomocou ktorého je možné efektívne pracovať s klastrom, pričom medzi efektívnu prácu môžeme zaradiť aj zvýšenie interaktívnosti voči klasickému prístupu práce s klastrom a to za pomoci príkazového riadku. V tejto časti budú spomenuté jednotlivé funkcie webovej aplikácie a aj porovnanie voči iným systémom.

6.2.1 Monitorovanie klastra

Dôležitou úlohou pri tejto webovej aplikácii je schopnosť monitorovať klaster z webového prehliadača, čo je viac používateľsky príťažlivé, ako použitie príkazo-

vého riadku, pričom takýto výpis je vo forme klasického textu.

V tejto aplikácii má používateľ možnosť vidieť stav klastra pomocou tabuľkového výpisu, pričom dôležité informácie sú rozlíšené tak, že intenzita, resp. sýtosť farby určuje úroveň danej vlastnosti klastra. V prípade, že farba je priehľadná, to označuje, že daná vlastnosť je na svojom minime. Na druhej strane, v prípade, že farba je priveľmi sýta a je nepriehľadná, to označuje, že daná vlastnosť klastra sa blíži ku svojmu maximu. Takého farebné odlišovanie má pomôcť zrýchliť identifikáciu stavu využitia klastra, napríklad v prípade problémov s klastrom alebo jeho nedostupnosťou. Ďalšou výhodou tejto webovej aplikácie pri monitorovaní stav klastra je tá, že používateľ vidí aktuálny stav klastra, pričom tieto informácie sú aktualizované v intervale jednej sekundy a veľkým plusom je to, že samotnú webovú aplikáciu nie je potrebné aktualizovať vďaka použitiu knižnice React a tiež vďaka protokolu WebSocket, ktorý podporuje obojsmernú komunikáciu a pre posielanie správ potrebuje len jednu HTTP požiadavku. Klastre sú taktiež rozdelené do skupín, zvyčajne podľa ich geografickej pozície a teda v prípade veľkého množstva klastrov môže používateľ zjednodušiť výber daného klastra pomocou týchto skupín.

Tabuľka s informáciami obsahuje viacero stĺpcov, pričom tieto stĺpce sú ďalej rozdelené to kategórií ohľadom centrálnej procesorovej jednotky, dočasnej pamäte RAM, diskov, siete a v niektorých typoch klastrov aj grafických procesorových jednotiek. Niektoré bunky dokonca poskytujú na podrobnejší prehľad o stave jednotlivých vlastností klastra pomocou správy v prípade, že používateľ príde nad bunku s kurzorom. Tieto informácie sú zobrazené pre vlastnosti ako čítanie a zápis na disky, rýchlosť siete a využitie grafických procesorových jednotiek.

Pre porovnanie je možné uviesť klasický prístup pomocou zadávania príkazov v príkazovom riadku. V tomto prípade je potrebné sa pripojiť na hlavný server klastra, napríklad za pomoci zabezpečeného sieťového protokolu SSH. Po vstupe na server je nutné spustiť príkaz, ktorým je možné vidieť stav klastra, no nevýhodou je to, že výstup je vo forme obyčajného textu a často sa môže stať, že veľký objem textu sa stane neprehľadným.

6.2.2 Ovládanie úloh na klastri

Ďalšou dôležitou úlohou tejto webovej aplikácie je možnosť spúšťať, zastavovať a monitorovať úlohy vykonávané na klastri z webového prehliadača.

Používateľ má v tejto aplikácii možnosť vidieť informácie o fronte úloh, ktoré sú pripravené na vykonanie, práve vykonávané úlohy, možnosť spustiť úlohu a tiež možnosť zastaviť úlohu v prípade, že danú úlohu spustil práve daný používateľ. Podobne ako v prípade monitorovania stavu klastra, aj tu sú použité farby na rýchle rozlíšenie stavu vykonávaných úloh. Modrou farbou sú znázornené úlohy, ktoré ešte čakajú na vykonanie. S klesajúcim počtom týchto úloh sa zvyšuje aj priehľadnosť danej bunky. Práve vykonávané úlohy sú znázornené oranžovou farbou, pričom súčet všetkých vykonávaných úloh by mal byť rovný počtu jadier, resp. slotov, čo znamená, že klaster vykonáva úlohy efektívne. Počet úspešne dokončených úloh je znázornený zelenou farbou, pričom ak používateľ prejde kurzorom nad danú bunku, zobrazia sa identifikátory jednotlivých podúloh, ktoré skončili úspechom. Červenou farbou sú označené úlohy, ktoré skončili neúspechom a rovnako ako pri úspešne dokončených úlohách má používateľ možnosť vidieť identifikátory tých podúloh, ktoré sa neskončili úspešne.

Ďalšou veľmi dôležitou súčasťou tabuľky úloh sú stĺpce s dátumami a časmi spustení a ukončení úloh, v prípade práve vykonávaných úloh je možné vidieť odhadovaný čas do ukončenia úlohy. Ak chce používateľ vidieť dátum a čas v klasickom formáte, môže prísť kurzorom nad danú bunku a zobrazí sa mu okno s dátumom a časom.

Pre porovnanie je možné uviesť klasický prístup spúšťania úloh pomocou zadávania príkazov v príkazovom riadku. V tomto prípade je potrebné sa pripojiť na hlavný server klastra, napríklad za pomoci zabezpečeného sieťového protokolu SSH. Po vstupe na server je nutné spustiť príkaz, ktorý spustí používateľovu úlohu v systéme SALSA. To je možné pomocou rovnakého príkazu, aký je možné vidieť vo webovej aplikácií v okne pod vstupým poľom pre zadanie príkazu.

6.3 Testovanie webovej aplikácie

Pre potreby overenia jednoduchosti používania aplikácie bolo požiadaných viacero študentov Univerzity Pavla Jozefa Šafárika v Košiciach a tiež študentov z Dubny, ktorí budú túto webovú aplikáciu reálne používať. Ich zadaním bolo zistiť stav klastra a taktiež spustiť úlohu na klastri, sledovať proces vykonávania a následne zrušiť vykonávanie úlohy. Študenti mali dostupnú webovú aplikáciu aj samotný klaster na verejnej URL adrese `https://salsa-hlit.jinr.ru`, pričom

mali aj používateľské účty, s ktorými sa vedeli prihlásiť do tejto webovej aplikácie.

Toto testovanie malo za cieľ zistiť jednoduchosť ovládania webovej aplikácie. Viacero študentov potvrdilo, že ovládanie bolo jednoduché a zistenie stavu klastra bolo rýchle. Taktiež počas diskusie spomenuli niekoľko vlastností, ktoré by mohli byť pre túto webovú aplikáciu prínosom, a teda by bolo dobré ich v budúcnosti implementovať.

6.4 Výhody webovej aplikácie

Táto sekcia práce je určená na zhodnotenie výhod vytvorenej webovej aplikácii oproti iným aplikáciám, systémom a prístupom, ktoré umonžňujú takúto prácu s klastrom.

Za jednu z hlavých výhod tejto práce je jej schopnosť umonžniť používateľovi pracovať s klastrom interaktívne. Používateľ vie spustiť a vymazať úlohy, sledovať priebeh vykonávania jednotlivých úloh v intervale jednej sekundy a tiež má možnosť vidieť odhadovaný čas dokončenia vykonávania úlohy. Táto aplikácia tiež poskytuje väčšie pohodlie pri ovládaní úloh, pričom nie je potrebné, aby samotný používateľ bol nútený sa pripojiť na vzdialený server za pomoci sieťového protokolu SSH. Táto aplikácia je responzívna a dáva používateľovi možnosť, aby mohol spustiť príkaz napríklad aj z mobilných zariadení, čo pri iných spôsoboch práce s klastrom nebolo možné.

Z technologického hľadiska je výhodou použitie knižnice React a protokolu WebSocket. Dôvodom je, že tieto zabezpečujú to, že získavanie nových informácií z API servera prebieha bez nutnosti otvárania nových HTTP spojení. Ďalej klient nemusí posielať požiadavku na server na to, aby získal nové dáta. Samotnú webovú aplikáciu nie je potrebné aktualizovať pre zobrazenie nových informácií a teda sa šetrí aj sieťová premávka, napríklad pri použití mobilnej siete na zisťovanie stavu klastra.

Z pohľadu API servera je výhodou to, že ako monitorovací, resp. dávkovací systém nie je nutné použiť len systém Obmon, resp. SALSA, ale aj rôzne iné. Dôležitým konceptom tohto servera je fakt, že pre komunikáciu so systémami používa sieťovú knižnicu ZMQ. Výhodou tejto knižnice je fakt, že nepoužíva HTTP protokol z dôvodu, že tento protokol má veľmi vysoké výpočtové náklady a ZMQ posiela veľmi veľké množstvo správ, čo by teda nebolo veľmi efektívne. Práve kvôli

efektívnosti prenosu správ pracuje knižnica ZMQ na nízkej úrovni.

Okrem samotnej práce je v prílohe priložená používateľská príručka, ktorá slúži pre používateľov, ktorí si chcú spustiť webovú aplikáciu spolu s API serverom a systémom SALSA a taktiež je priložená aj systémová príručka slúžiaca pre vývojárov, ktorá obsahuje celkovú štruktúru webovej aplikácie, čo je vhodné pre prípad pokračovania v implementácií. Samotná webová aplikácia môže byť nasadená na akýkoľvek webový server, napríklad Apache alebo Nginx, stačí postaviť aplikáciu pomocou jedného príkazu, nastaviť URL adresy API servera a postavenú aplikáciu len skopírovať na daný server a aplikácia bude fungovať.

7 Záver

Táto práca mala za cieľ implementovať webovú aplikáciu, pomocou ktorej je možné monitorovať stav klastrov a taktiež zadávať, spúšťať a kontrolovať úlohy vykonávané na klastroch.

V prvej časti je čitateľ oboznámený dôkladnou analýzou o infraštruktúre GRID, rôznych monitorovacích systémoch a taktiež o rôznych systémoch na dávkovanie úloh na klastri.

Druhá časť bola venovaná návrhu a neskôr implementácií tejto webovej aplikácie na monitorovanie klastra a správu úloh na klastri. V tejto časti je spomenutý proces, pri ktorom bolo potrebné sa rozhodnúť o tom, ktorou technológiou bude výsledná webová aplikácia implementovaná. Tu boli spomenuté viaceré známe technológie na tvorbu webových aplikácií. Okrem výberu vhodnej technológie na tvorbu aplikácie bolo taktiež vhodné spomenúť, akým spôsobom budú dáta posielané medzi samotnou aplikáciou a API serverom a tiež na akej platforme bude zdrojový kód spravovaný. Po tomto kroku bolo potrebné navrhnúť prototyp používateľského rozhrania, ktoré sa časom samozrejme menilo a zlepšovalo.

Okrem procesu návrhu táto časť obsahuje aj samotnú implementáciu webovej aplikácie, ktorá bola najzaujímavejšou časťou celej práce. Používateľ bude musieť byť v tejto aplikácií prihlásený v prípade, že bude chcieť manipulovať s úlohami na klastri. Pre sledovanie stavu klastra nie je potrebné prihlásenie. Prihlasovací server je možné meniť pomocou konfigurácie API servera.

Vyhodnotenie bolo vykonávané študentami a taktiež aj pracovníkmi, ktorí budú túto webovú aplikáciu reálne používať, pričom je možné povedať, že cieľ tejto webovej aplikácie bol dosiahnutý. Táto aplikácia ponúka možnosť monitorovania klastra, pričom tieto dáta sú aktualizované v sekundovom intervale, teda tieto dáta sú aktuálne. Okrem monitorovania má táto aplikácia slúžiť na správu úloh vykonávaných na klastri, pričom táto podmienka je rovnako splnená, keďže používateľ

má umožnené spúšťať úlohy, sledovať vykonávané úlohy v reálnom čase, pričom úlohy, ktoré boli spustené tým istým používateľom je možné zastaviť, respektíve zrušiť. Samozrejme v práci sa nachádza niekoľko nedostatkov, ktoré je možné ďalej opraviť alebo doplniť.

Táto webová aplikácia je reálne nasadená, pričom v tejto dobe je reálne používaná. Keďže bola vytvorená za pomoci komponentového programovania, implementovanie nových častí aplikácie nie je náročné. Zdrojové kódy ku tejto webovej aplikácií sú dostupné na priloženom CD.

Literatúra

[1] AJ Peters, EA Sindrilaru a G Adde. "EOS as the present and future solution for data storage at CERN". In: *Journal of Physics: Conference Series*. Zv. 664. 4. IOP Publishing. 2015, s. 042042.

[2] Rayne Wiselman. *Data Protection Manager Documentation*. 2018. URL: https://docs.microsoft.com/en-us/system-center/dpm/?view=sc-dpm-2019.

[3] *dCache*. URL: https://www.dcache.org/.

[4] *Welcome to the Worldwide LHC Computing Grid*. URL: https://wlcg.web.cern.ch/.

[5] Andy B Yoo, Morris A Jette a Mark Grondona. "Slurm: Simple linux utility for resource management". In: *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer. 2003, s. 44–60.

[6] *Computing with HTCondor*. 2020. URL: https://research.cs.wisc.edu/htcondor/.

[7] *TORQUE Resource Manager*. 2013. URL: http://docs.adaptivecomputing.com/torque/4-2-6/torqueAdminGuide-4.2.6.pdf.

[8] *What is Nagios?* URL: https://www.nagios.org/about/.

[9] James Mills. *netdata*. 2020. URL: https://github.com/netdata/netdata.

[10] Alberto Aimar et al. "MONIT: Monitoring the CERN Data Centres and the WLCG Infrastructure". In: *EPJ Web of Conferences*. Zv. 214. EDP Sciences. 2019, s. 08031.

[11] Stefano Bagnasco et al. "AliEn: ALICE environment on the GRID". In: *Journal of Physics: Conference Series*. Zv. 119. 6. IOP Publishing. 2008, s. 062012.

[12] IBM Corporation. *What is batch processing?* 2014. URL: https://www.ibm.com/support/knowledgecenter/zosbasics/com.ibm.zos.zconcepts/zconc_whatisbatch.htm.

[13] Seren Soner a Can Özturan. "Integer programming based heterogeneous cpu–gpu cluster schedulers for slurm resource manager". In: *Journal of computer and system sciences* 81.1 (2015), s. 38–56.

[14] A. Hameurlain et al. *Transactions on Large-Scale Data- and Knowledge-Centered Systems XXX: Special Issue on Cloud Computing*. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2016. ISBN: 9783662540541. URL: https://books.google.sk/books?id=kaOwDQAAQBAJ.

[15] *Basic Guide of Interprocess Communication and Pipes*. 2014. URL: http://technotif.com/basic-guide-interprocess-communication-pipes/.

[16] *History of Nagios. Nagios History. Nagios.com*. URL: https://www.nagios.org/about/history/.

[17] *Nagios Features - Nagios*. URL: https://www.nagios.org/about/features/.

[18] *MONIT*. URL: http://monit.web.cern.ch/monit/.

[19] Eric Stanley. *nagios/main*. 2013. URL: https://github.com/ageric/nagios/blob/master/html/main.php.

[20] *Introduction to the Angular Docs*. URL: https://angular.io/docs.

[21] Olga Filipova. *Learning Vue. js 2*. Packt Publishing Ltd, 2016.

[22] *React*. URL: https://reactjs.org/.

[23] *Introducing JSX*. URL: https://reactjs.org/docs/introducing-jsx.html.

[24] Ian Fette a Alexey Melnikov. "The WebSocket Protocol". In: *RFC* 6455 (2011), s. 1–71.

[25] Y Furukawa. "Web-based control application using WebSocket". In: *ICALEPCS2011* (2011), s. 673–675.

[26] *What is GitLab?* URL: https://about.gitlab.com/what-is-gitlab/.

[27] Mark Otto. *Bootstrap*. URL: https://getbootstrap.com/.

[28] *About Us - Material-UI*. URL: https://material-ui.com/company/about/.

[29] *React Router: Primary Components*. URL: https://reacttraining.com/react-router/web/guides/primary-components.

[30] *Introducing Hooks*. URL: https://reactjs.org/docs/hooks-intro.html.

Zoznam skratiek

API Application Program Interface.

CD Continuous Delivery / Deployment.

CI Continuous Integration.

CSS Cascading Style Sheets.

DOM Document Object Model.

GPL GNU General Public License.

HTML HyperText Markup Language.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

InPROC InProcess Communication.

IPC Inter-Process Communication.

JSX JavaScript XML.

JWT JSON Web Token.

MVC Model-View-Controller.

RAM Random-access Memory.

SPA Single-Page Application.

SSH Secure Shell.

SSQ Screen Sequence Diagram.

URL Uniform Resource Locator.

ZMQ ZeroMQ.

Zoznam príloh

Príloha A CD médium – záverečná práca v elektronickej podobe,

Príloha B Používateľská príručka

Príloha C Systémová príručka

Príloha B: Používateľská príručka

Dominik Matis

28.05.2020

Táto aplikácia má za cieľ poskytnúť používateľovi možnosť kontroly klastra tak, že používateľ je schopný zistiť aktuálny stav všetkých uzlov v klastri a taktiež mu je umožnené spúšťať a zastavovať úlohy, ktoré sú vykonávané v tomto klastri. Táto práca bola testovaná v prehliadačoch *Google Chrome* a *Mozilla Firefox*, pričom tieto prehliadače sú aj odporúčané. Systém SALSA, API server **zmq2ws** boli spúšťané a testované na operačných systémoch *CentOS 7*, *Fedora 31* a *Fedora 32*. Okrem reálnej inštalácie zo zdrojového kódu je možné použiť aj Docker obraz alebo inštaláciu z RPM balíkov. Táto príručka bude ukazovať postup používania systému na operačnom systéme *CentOS 7* v Docker kontajneri a taktiež aj s nainštalovaním balíkov cez RPM.

1 Inštalácia

Predtým ako bude možné používať túto webovú aplikáciu, je nutné, aby bol tento projekt stiahnutý a taktiež aby boli všetky potrebné závislosti pre chod tejto aplikácie nainštalované. To je možné dosiahnuť buď použitím už predpripraveného Docker obrazu, ktorý je potrebné už len spustiť alebo nainštalovať balíky priamo do systému.

1.1 Spustenie cez Docker

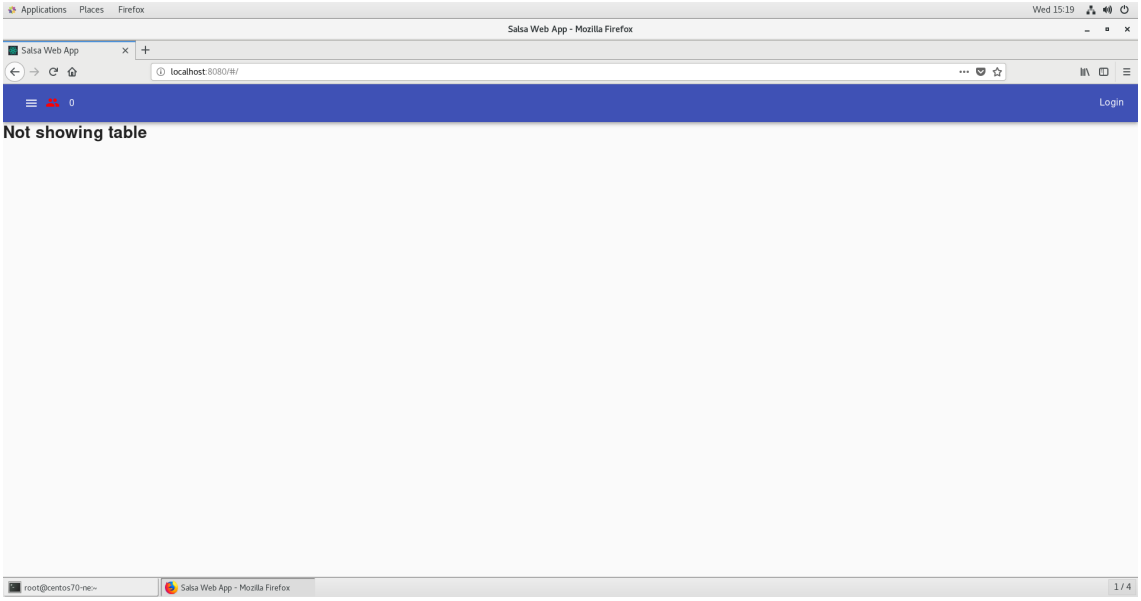
Pre spustenie webovej aplikácie je nutné stiahnuť Docker obraz a spustiť ho. Predvolene je v tomto obraze spustený NGINX server, ktorý počúva na porte 80, pričom v prípade, že chceme počúvať na inom porte, je potrebné pridať do príkazu parameter `-p 8080:80`, kde `8080` je port, na ktorom chceme počúvať a `80` je port na ktorom počúva NGINX server vo vnútri kontajnera. Argument `-d` spôsobí, že daný kontajner bude bežať v pozadí.

```
docker run -d -p 8080:80 registry.gitlab.com/domosino/thesis-webapp
```

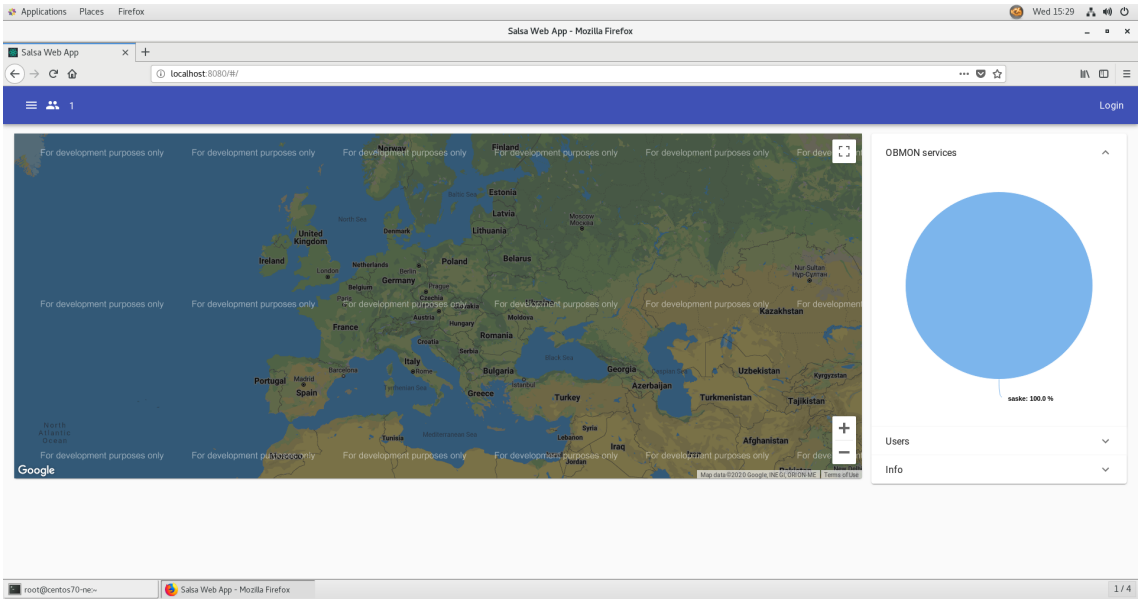
Po spustení kontajnera je možné navštíviť vo webovom prehliadači adresu http://localhost:8080, kde sa zobrazí táto webová aplikácia tak ako je na Obr. 1.

Ako je možné vidieť na obrázku, webová aplikácia je funkčná, no keďže nemá prístup na API server, tak ikona používateľov pri tlačidle menu je červená, čo značí, že aplikácia nie je pripojená cez WebSocket na server. Ten je možné spustiť ako ďalší Docker kontajner, pričom jediným argumentom, ktorý potrebuje je `--net host`, čo zabezpečí, že tento server sa bude tváriť ako keby bol spustený priamo na systéme z pohľadu siete. Po zadaní nasledujúceho príkazu sa stiahne obraz a spustí kontajner s API serverom.

```
docker run --net host registry.gitlab.com/ndmspc/zmq2ws
```

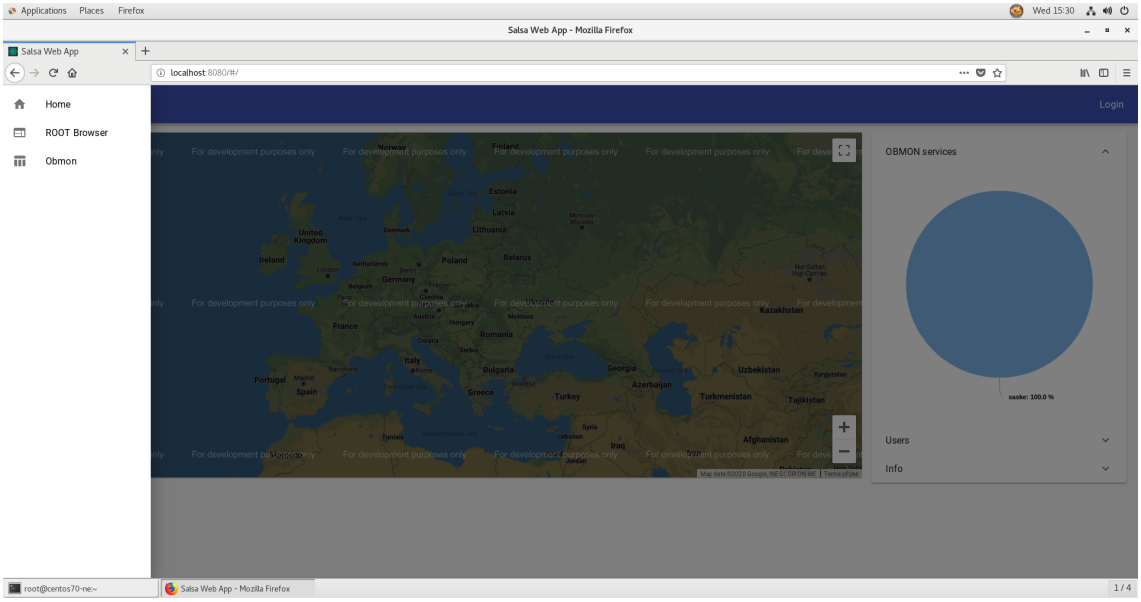


Obr. 1: Webová aplikácia bez pripojenia na API



Obr. 2: Webová aplikácia s pripojením na API

Na Obr. 2 je možné vidieť, že farba ikony používateľov sa zmenila z červenej na bielu a taktiež počet používateľov sa zmenil. To značí, že bolo nadviazané spojenie s API serverom. V prípade, že používateľ otvorí menu, tak bude mať možnosť pozrieť si dáta zo senzorov Obmon ako je zobrazené na Obr. 3.



Obr. 3: Menu webovej aplikácie

Keďže cieľom je monitorovať klaster a zadávať úlohy, je potrebné, aby bol systém SALSA spustený. To je monžné spustením ďalšieho kontajnera, pričom rovnako ako pri spustení API servera je potrebné spustiť aj tento obraz s argumentom `--net host` a v prípade, že chceme, aby tento kontajner bol spustený na pozadí, tak použijeme znovu parameter `-d`. Po spustení nasledujúceho príkazu je možné sledovať, že vo webovej aplikácií sa v menu objavila ďalšia možnosť tak ako je na Obr. 4.

```
docker run -d --net host registry.openbrain.sk/salsa/salsa
```

V tomto kroku už môžeme vo webovej aplikácií prejsť na sekciu SALSA pričom prídete na podstránku, ktorá vyzerá ako na Obr. 5.

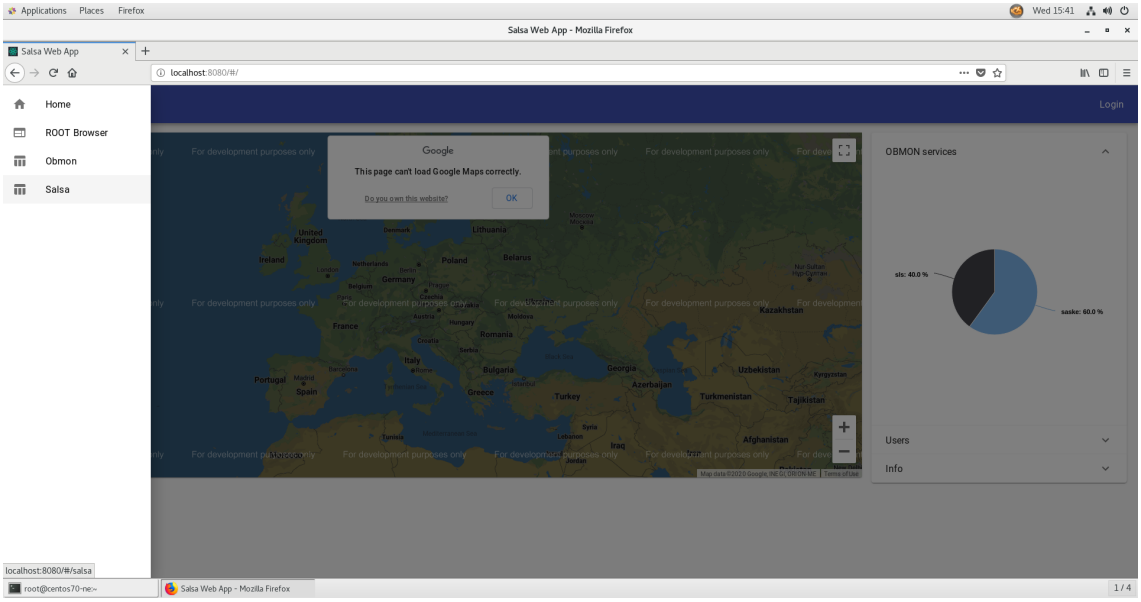
Keďže sme ešte na klastri nespustili žiadnu úlohu, tak vidíme, že vo výpise nám aplikácia hlási, že žiadne úlohy nie sú dostupné vo fronte na zobrazenie. Pre spustenie úlohy je možné spustiť nasledujúci príkaz.

```
docker run -it --rm --net host --entrypoint salsa-feeder \
registry.openbrain.sk/salsa/salsa -s tcp://localhost:41000 -t "sleep 1:100"
```

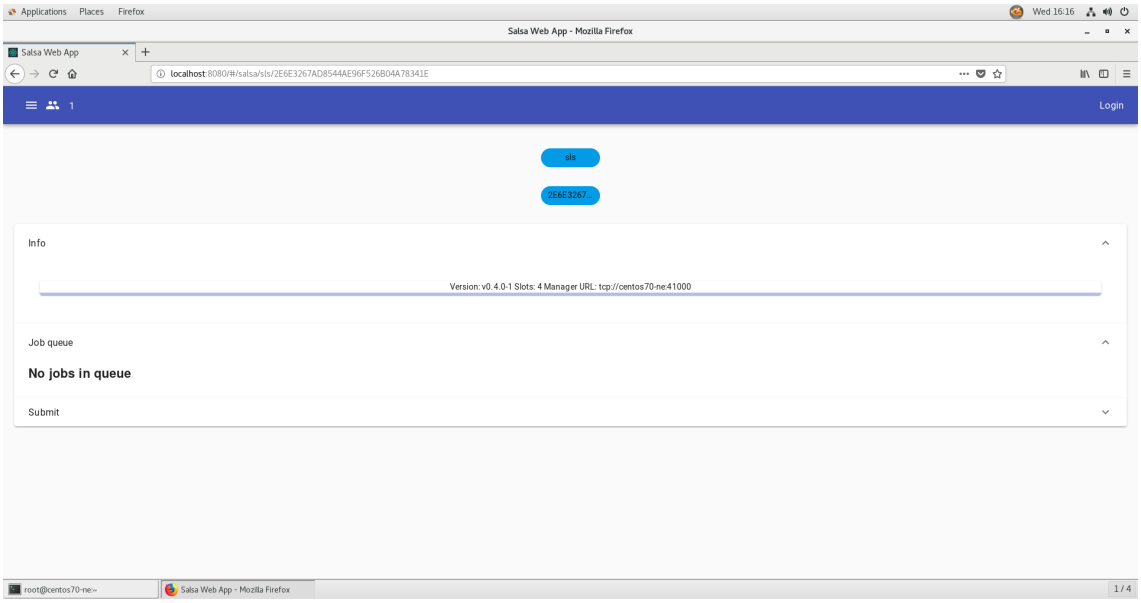
Po spustení je možné hneď vidieť vo webovej aplikácií priebeh vykonávania úlohy. Tabuľku so všetkými úlohami

1.2 Inštalácia cez RPM

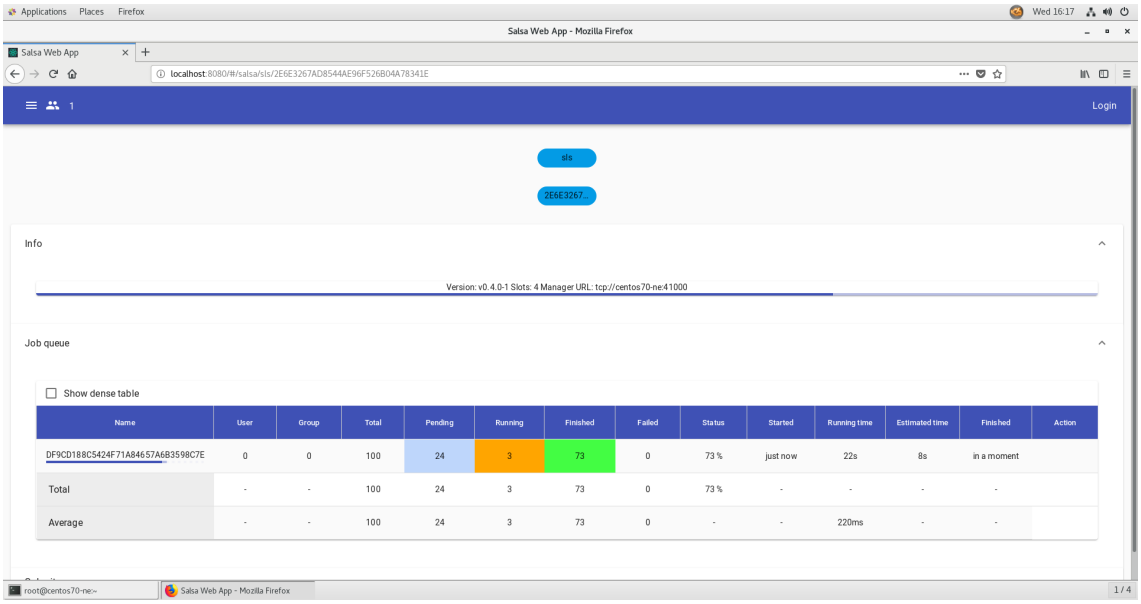
Pred samotnou inštaláciou projektu je potrebné dodať repozitáre epel, obl/stable a taktiež pridať najnovšiu verziu Node.js pomocou nasledujúcich príkazov:



Obr. 4: Menu webovej aplikácie pri spustenom systéme SALSA



Obr. 5: Monitorovanie systému SALSA



Obr. 6: Monitorovanie úlohy systému SALSA

1.2.1 Povolenie repozitára

```
yum install -y epel-release yum-plugin-copr
yum copr enable obl/stable -y
curl -sL https://rpm.nodesource.com/setup_10.x | sudo bash -
```

1.2.2 Inštalácia webovej aplikácie (slswebapp)

Tento projekt je uložený v RPM balíku **slswebapp** a ten si nainštalujeme pomocou príkazu

```
yum install slswebapp -y
```

Keďže primárne je priečinok s HTML súbormi pre NGINX server `/usr/share/nginx/html` a nami vytvorený projekt sa nachádza v priečinku `/usr/share/slswebapp/build`, je nutné prepísať konfiguračný súbor `/etc/nginx/nginx.conf` riadok nasledujúcim príkazom

```
cp /etc/nginx/nginx.conf.http.slswebapp /etc/nginx/nginx.conf
```

Po tomto prepísaní je možné túto webovú aplikáciu spustiť pomocou `nginx` služby cez nasledujúci príkaz:

```
systemctl enable --now nginx
```

1.2.3 Inštalácia API (zmq2ws)

Ďalším krokom je inštalácia API, ktoré sa nachádza v balíku `zmq2ws`, ten je možné nainštalovať a spustiť pomocou nasledujúcich príkazov:

```
yum install zmq2ws -y
systemctl enable --now zmq2ws
```

1.2.4 Inštalácia systému SALSA

Posledným krokom je inštalácia systému SALSA

```
yum install salsa -y
```

V systéme SALSA je za potreby spustiť viacero služieb. Medzi nimi sú redirector (*rdr*), worker (*wk*) a broker. Tieto služby je potrebné spustiť:

```
systemctl enable --now salsa@rdr salsa@wk salsa-broker
```

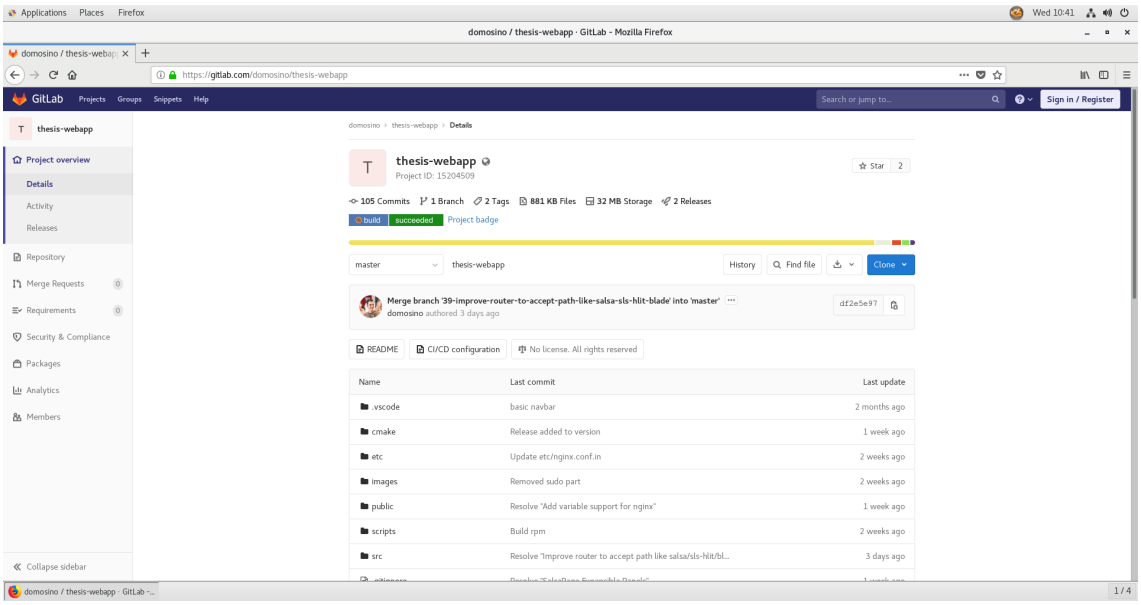
1.2.5 Spustenie testovacej úlohy

Po týchto krokoch je celý systém SALSA spolu s API a webovou aplikáciou pripravený a je možné na ňom spustiť úlohu. Tieto úlohy je možné spustiť cez príkaz `salsa-feeder`:

```
salsa-feeder -s tcp://localhost:41000 -t "sleep 1:1000"
```

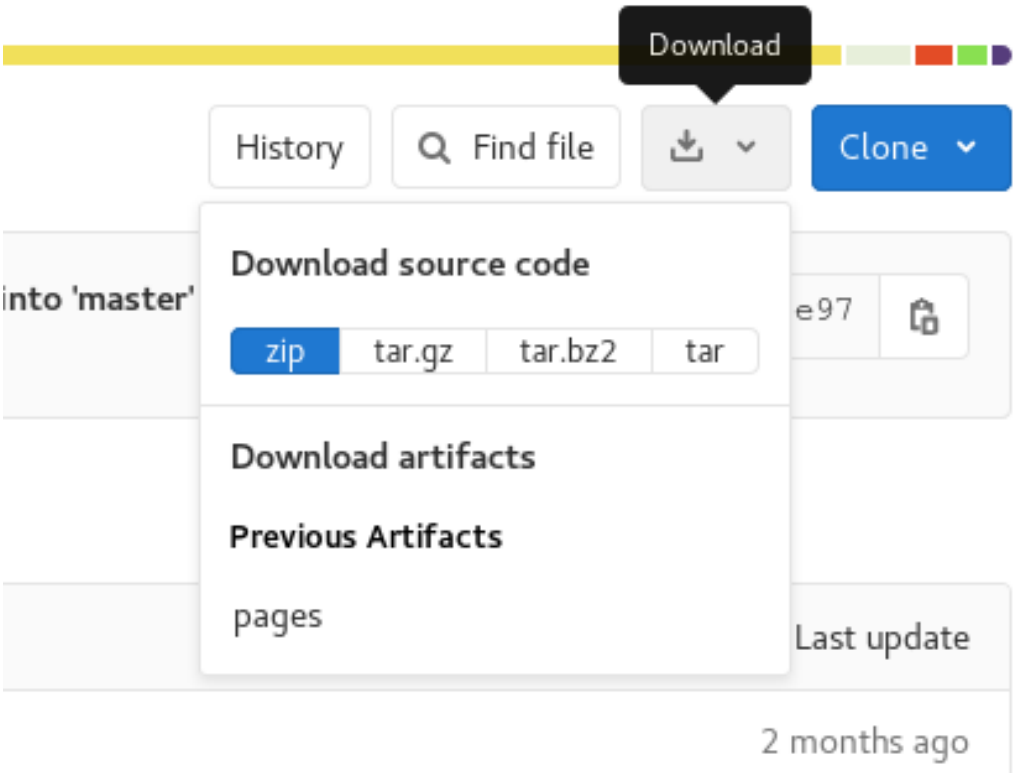
1.3 Inštalácia zo zdrojového kódu

Prvým krokom je stiahnutie projektu webovej aplikácie. Tento projekt sa nachádza na webovej platforme GitLab na adrese https://gitlab.com/domosino/thesis-webapp. Vzhľad platformy GitLab je možné vidieť na Obr. 1.



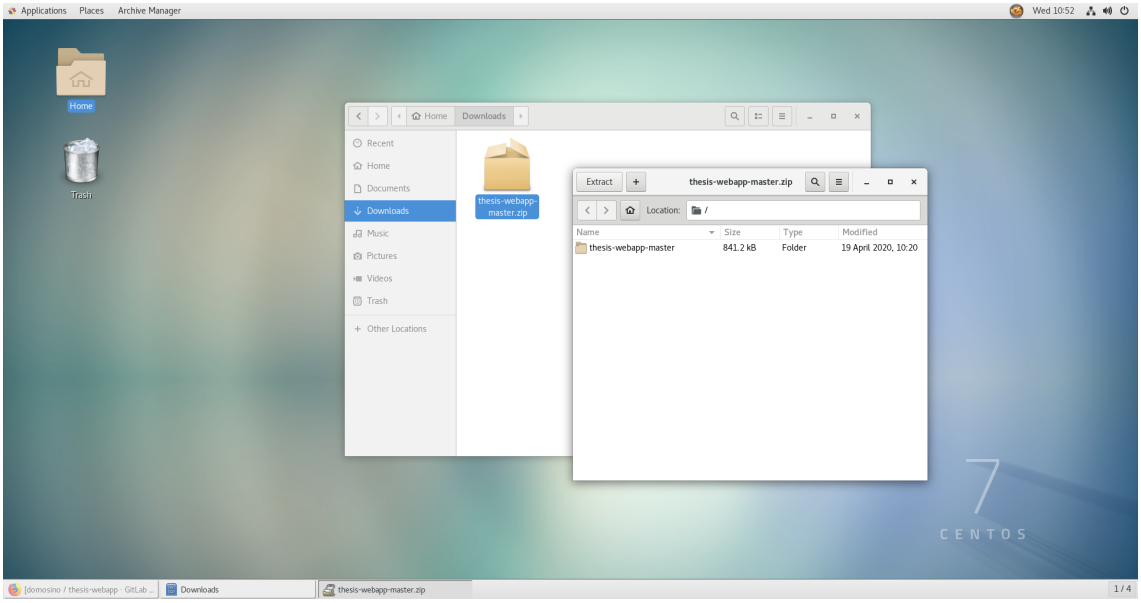
Obr. 7: Zdrojový kód aplikácie na platforme GitLab

Po otvorení tejto stránky je potrebné kliknúť na ikonu stiahnutia, ktorá sa nachádza na lište približne v strede obrazovky, hneď naľavo pri tlačidle **Clone**. Po kliknutí na túto ikonu sa rozbalí menu, kde je možnosť stiahnutia zdrojového kódu vo viacerých komprimovaných formátoch. Pre stiahnutie zdrojového kódu vo formáte *zip* je potrebné kliknúť na tlačidlo s názvom **zip** (predvolene označené modrou farbou), viď Obr. 2.



Obr. 8: Stiahnutie zdrojového kódu

Po stiahnutí je potrebné tento archív otvoriť. V OS CentOS je možné použiť grafické rozhranie alebo je možné použiť príkaz `unzip` v terminále. V prípade grafického rozhrania je potrebné otvoriť stiahnutý archív dvojklikom na tento súbor, pričom sa otvorí nové okno, ktoré nám ukazuje obsah tohto archívu. Pre rozbalenie obsahu tohto archívu je potrebné stlačiť tlačidlo **Extract** (Obr. 3).



Obr. 9: Rozbalenie archívu

Týmto sme získali zdrojový kód projektu, ktorý je možné spustiť. Keďže samotná aplikácia je napísaná poomocou knižnice React, ktorá je založená na programovacom jazyku JavaScript, je nutné, aby operačný systém mal v sebe nainštalovaný **Node.js** a **NPM**.

```
yum install -y nodejs npm
```

Po inštalácii softvérového rámca **Node.js** je možné stiahnuť, resp. nainštalovať závislosti pre túto webovú aplikáciu. Po nainštalovaní závislostí je možné webovú aplikáciu spustiť a táto bude dostupná na adrese http://localhost:3000.

```
npm install
npm start
```

1.3.1 Inštalácia API (zmq2ws)

Okrem samotnej webovej aplikácie je nutné mať spustený aj API server, ktorý poskytuje aktuálne informácie pre používateľa používajúceho webovú aplikáciu. Zdrojový kód tohto API servera sa taktiež nachádza na serveri GitLab, pričom repozitár je možné stiahnuť na nasledujúcom odkaze: https://gitlab.com/ndmspc/zmq2ws/-/archive/master/zmq2ws-master.zip. Po stiahnutí a otvorení archívu je potrebné nainštalovať závislosti pomocou príkazu:

```
npm install
```

A po získaní všetkých závislostí je možné tento API server spustiť pomocou príkazu

```
npm start
```

Tento API server je potom dostupný na porte `8442`. Po tomto kroku by mala farba ikony používateľov vo webovej aplikácii zmeniť farbu z červenej na bielu, pretože webová aplikácia nadviazala spojenie cez WebSocket s API serverom.

1.3.2 Inštalácia systému SALSA

Posledným krokom je pre správny chod webovej aplikácie je inštalácia dávkovacieho systému SALSA. Pre inštaláciu je potrebné stiahnuť repozitár z platformy GitLab pomocou nasledujúceho odkazu https://gitlab.openbrain.sk/salsa/salsa/-/archive/master/salsa-master.zip. Po stiahnutí a otvorení archívu je možné začať so samotnou inštaláciou. No predtým, ako spustíme inštaláciu, systém SALSA potrebuje niekoľko nástrojov nainštalovaných na cieľovom systéme, ktoré je možné nainštalovať pomocou nasledujúceho `yum` príkazu

```
yum install redhat-lsb cmake gcc-c++ zyre-devel spdlog-devel protobuf-devel \
jsoncpp-devel yaml-cpp-devel doxygen ncurses-devel ndm-devel help2man
```

Po pripravení systému na inštaláciu je možné spustiť inštaláciu pomocou inštalačného skriptu, ktorý sa nachádza v priečinku projektu `scripts` s názvom `make.sh` nasledujúcim spôsobom:

```
cd salsa
scripts/make.sh install
```

V tomto momente je systém SALSA nainštalovaný v priečinku projektu `build/src` a je možné ho spustiť nasledujúcimi príkazmi z priečinku projektu:

```
/build/src/salsa -c etc/config.json -n rdr,wk --debug
```

Tento príkaz spustí sprostredkovateľa a pracovný uzol v ladiacom móde.

1.3.3 Spustenie testovacej úlohy

V tomto kroku by sa mala objaviť položka SALSA v bočnom menu webovej aplikácie, pričom tabuľka úloh bude prázdna. Spustenie úlohy je možné cez príkaz `salsa-feeder` nasledujúco:

```
/build/src/salsa-feeder -s tcp://localhost:41000 -t "sleep 1:1000"
```

2 Prihlásenie

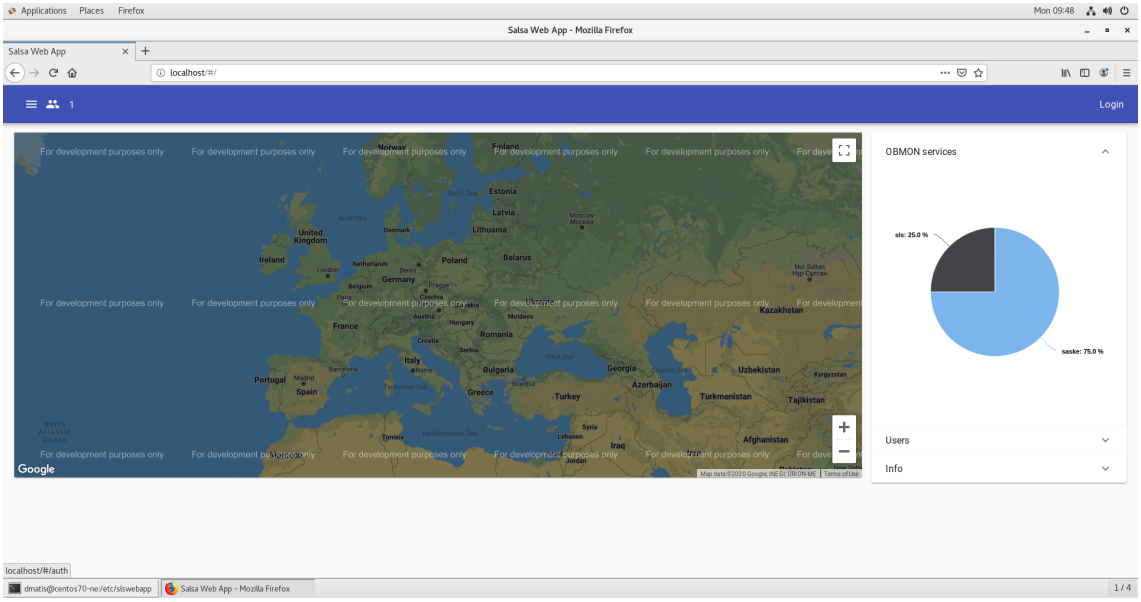
Po úspešnom spustení webovej aplikácie a jej správnom chode je možné sa v aplikácii prihlásiť. To je možné kliknutím na tlačidlo **Login**, ktoré sa nachádza na hornej lište úplne napravo. Toto tlačidlo je možné vidieť aj na Obr. 10.

Po kliknutí na toto tlačidlo sa používateľ presunie na stránku s prihlasovacím formulárom. Tu je potrebné zadať používateľské meno a heslo pre prístup do aplikácie. Prihlasovací formulár je na Obr. 11.

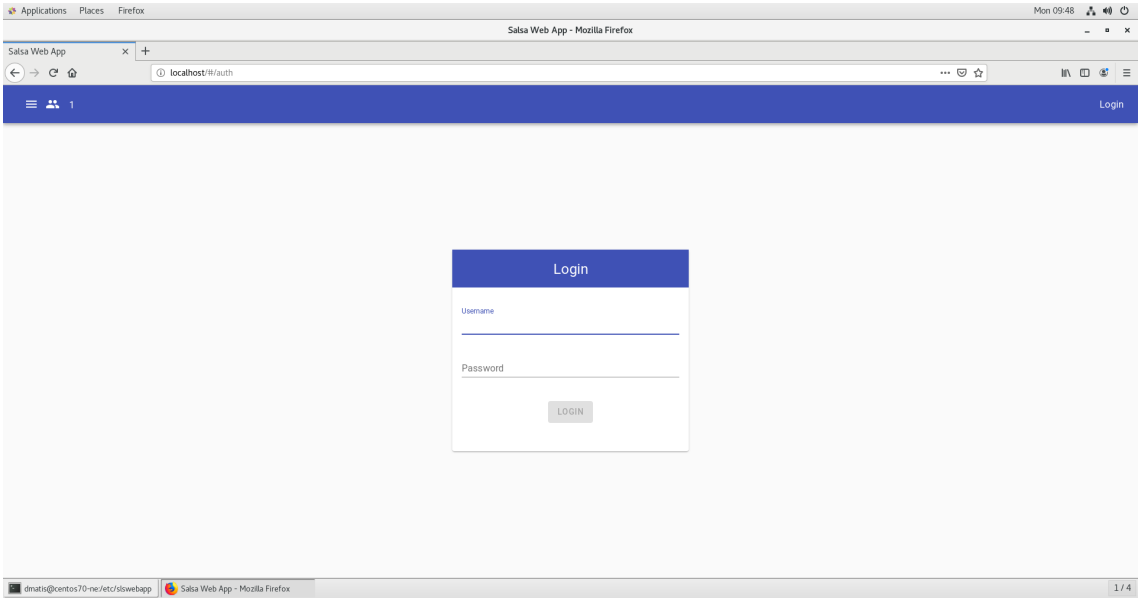
Po uvedení prihlasovacieho mena a hesla je možné kliknúť na modré tlačidlo **LOGIN**, ktoré sa nachádza v dolnej časti formulára. Toto tlačidlo je zobrazené na Obr. 12.

Po kliknutí na tlačidlo sa miesto tlačidla zobrazí animácia načítavania, čo znamená, že webová aplikácia sa snaží autentifikovať používateľa. Načítavanie a teda overovanie používateľa je možné vidieť na Obr. 13.

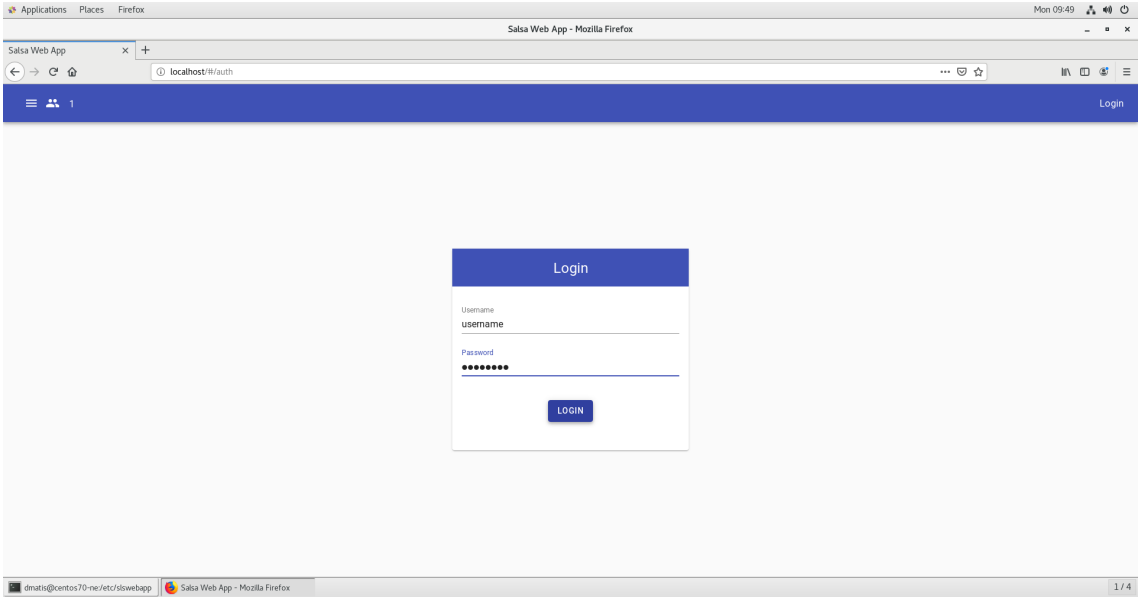
Po úspešnom overení používateľa nastáva jeho prihlásenie a používateľ je presunutý na stránku so svojim profilom, kde môže vidieť informácie o svojom účte tak, ako je to na Obr. 14.



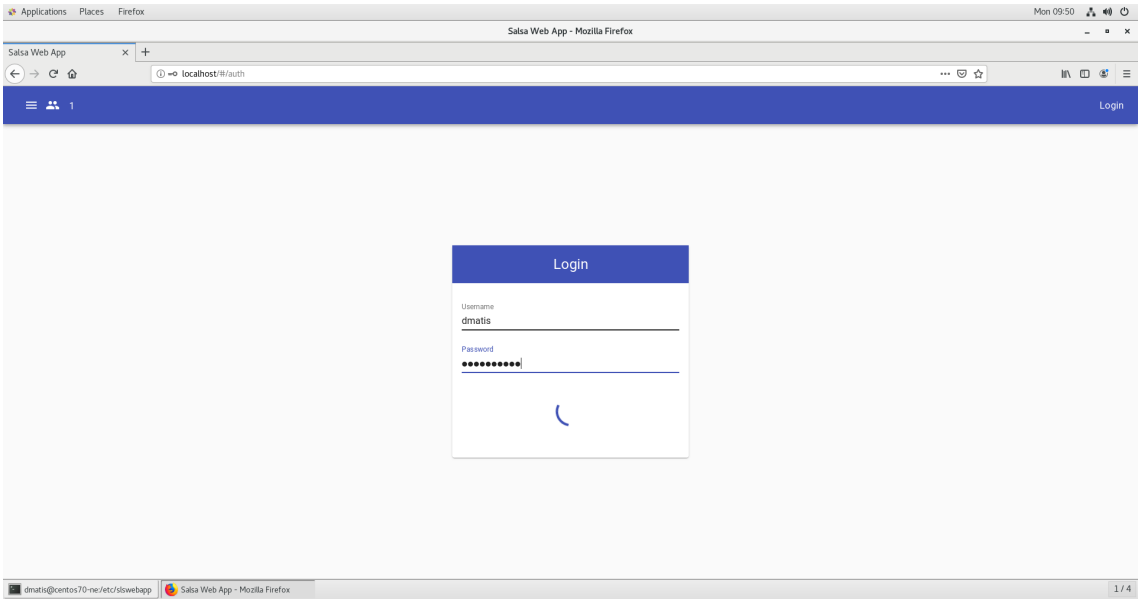
Obr. 10: Hlavná stránka



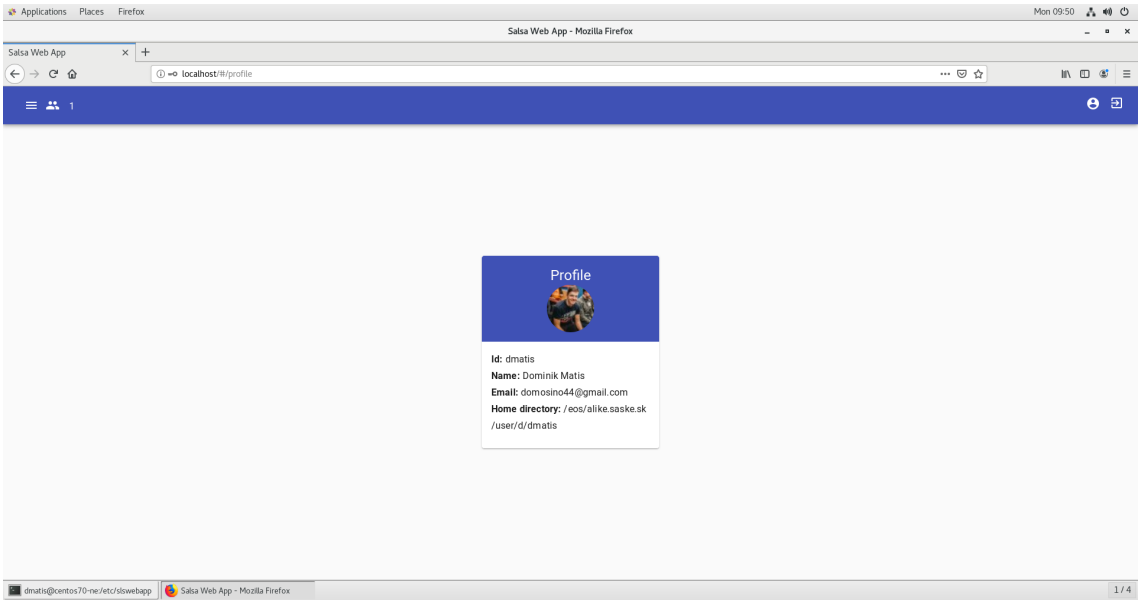
Obr. 11: Prihlasovacia stránka



Obr. 12: Vyplnený prihlasovací formulár



Obr. 13: Autentifikácia používateľa



Obr. 14: Profilová stránka používateľa

3 Monitorovanie klastra

Pre monitorovanie klastra je potrebné v menu vybrať položku **Obmon** tak, ako je to ukázané na Obr. 15.

Po výbere tejto položky z menu bude používateľ presunutý na stránku, na ktorej si bude môcť vybrať klaster, od ktorého chce dostávať informácie, presnejšie si vyberie službu a podslužbu. Ponuku je možné vidieť na Obr. 16.

Akonáhle je vybratá služba a podslužba na počúvanie, zobrazí sa tabuľka zobrazujúca rôzne informácie o danom klastri, resp. službe. Tabuľka obsahuje aj farebné rozlíšenie niektorých buniek pre lepšiu orientáciu. Snímok obrazokvky je zobrazený na Obr. 17.

Keďže tabuľka môže obsahovať veľmi veľa údajov a môže sa z nej stať dosť rozsiahla tabuľka, je možné upraviť tabuľku, aby mala kompaktnejší tvar. Tento kompaktný tvar je možné vidieť na predchádzajúcom obrázku. Predvolene je tabuľka v klasickom tvare, pričom zmena tvaru tabuľky je možná pomocou zaškrtnutia položky **Show dense table**, ktorá sa nachádza nad tabuľkou vľavo. Klasický vzhľad tabuĺky je vyobrazený na Obr. 18.

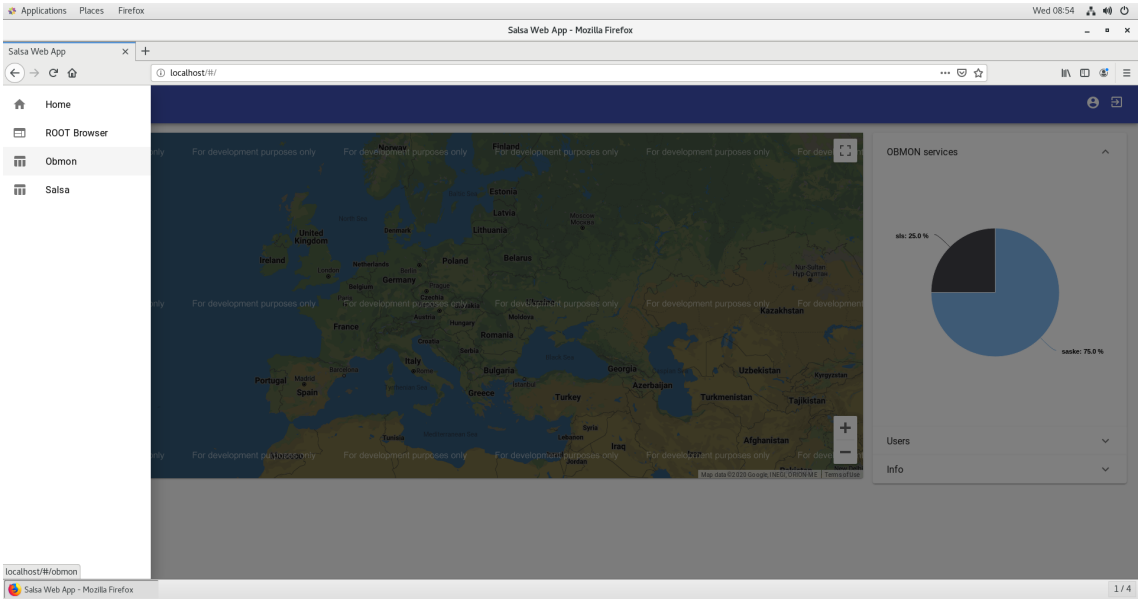
Niektoré bunky obsahujú aj ďalšie informácie, ku ktorým je možné sa dostať tým, že nad danú bunku príde používateľ kurzorom (Obr. 19)

4 Sputenie úlohy na klastri

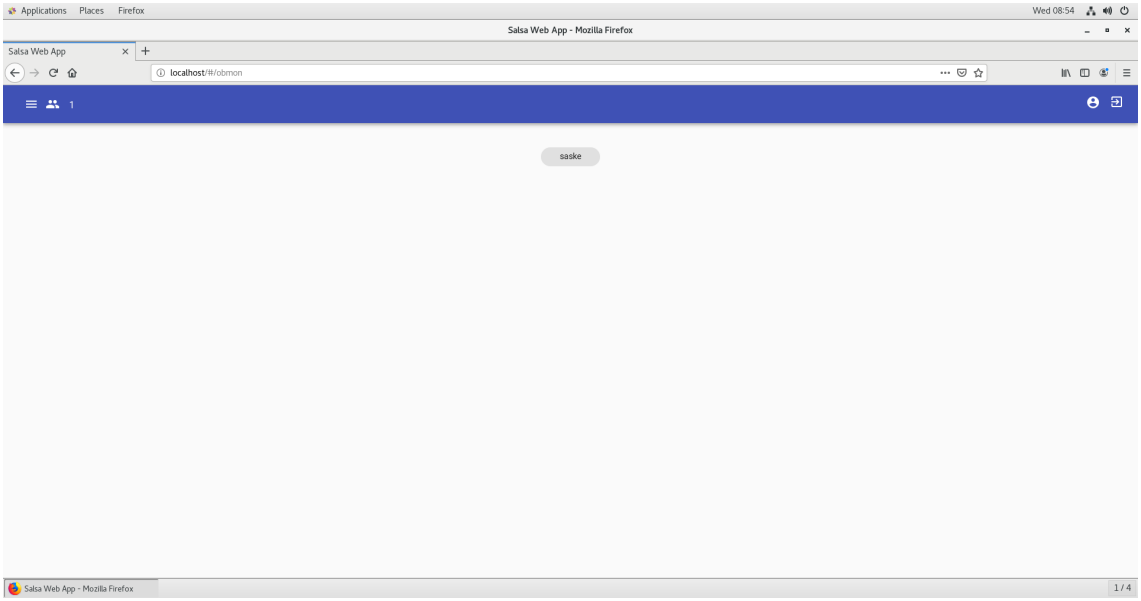
Pre spustenie úlohy je potrebné otvoriť stránku **Salsa** v hlavnom menu (Obr. 20).

Po výbere tejto položky z menu bude používateľ presunutý na stránku, na ktorej si bude môcť vybrať klater, od ktorého chce dostávať informácie, resp. chce spustiť úlohu. Presnejšie si používateľ vyberie službu a podslužbu (Obr. 21).

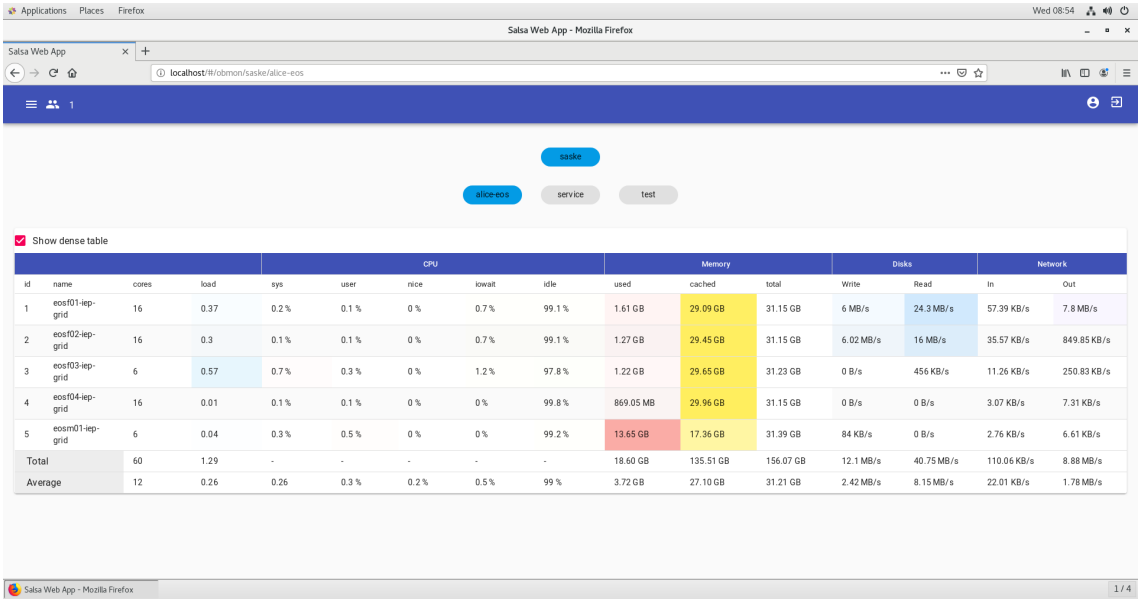
Akonáhle je vybratá služba a podslužba na počúvanie, zobrazí sa zoznam, kde je predvolene zobrazená informácia o verzii, počte jadier a tiež URL adrese sprostredkovateľa. Okrem tejto



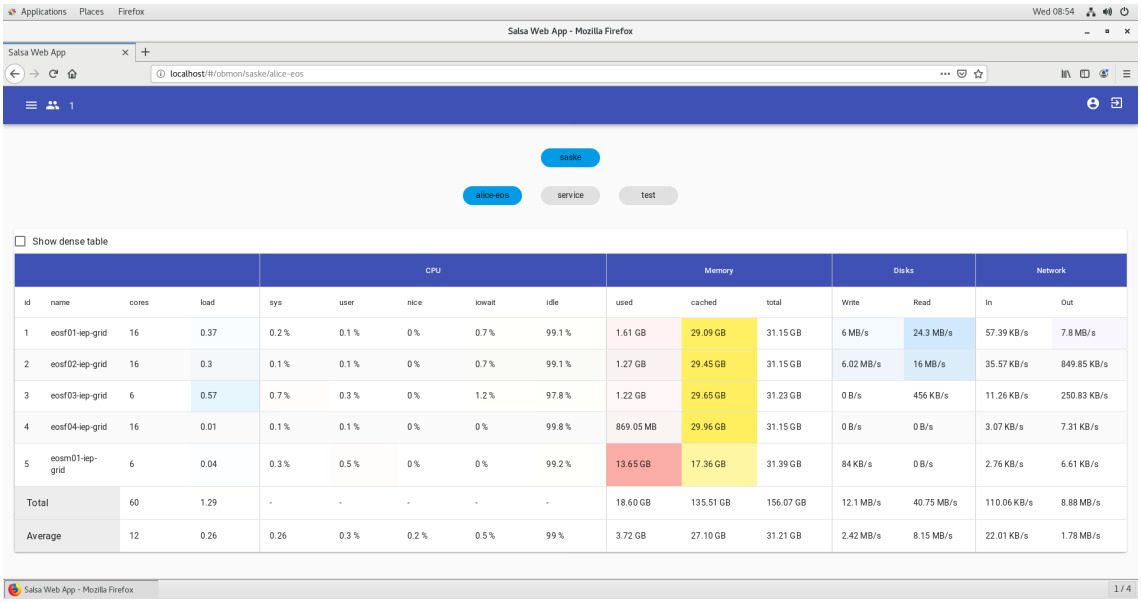
Obr. 15: Hlavné menu



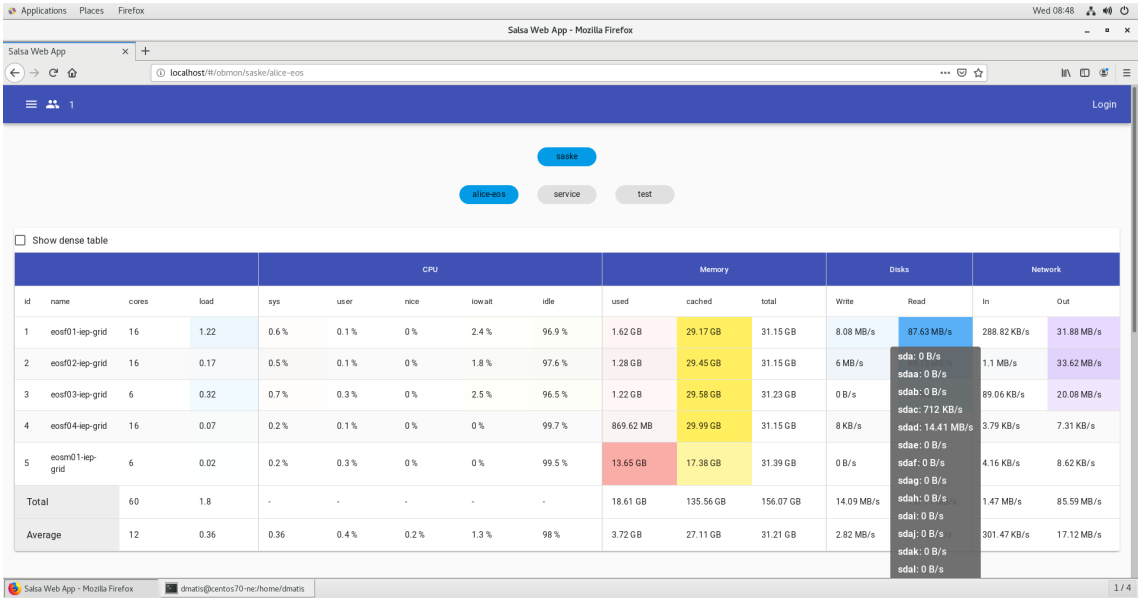
Obr. 16: Ponuka služieb na počúvanie



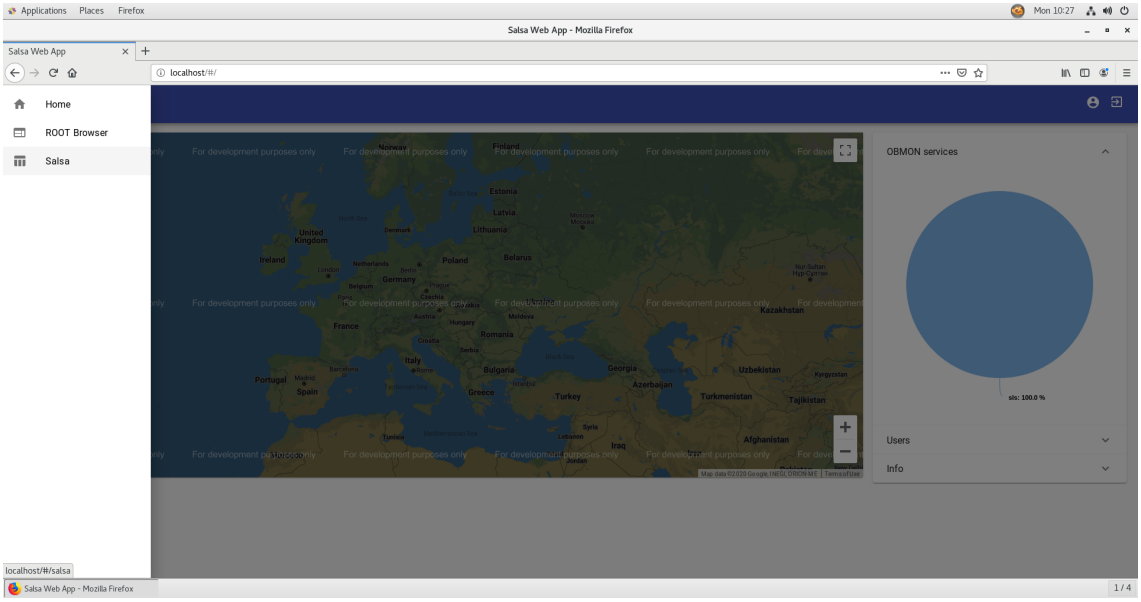
Obr. 17: Tabuľka zobrazujúca informácie



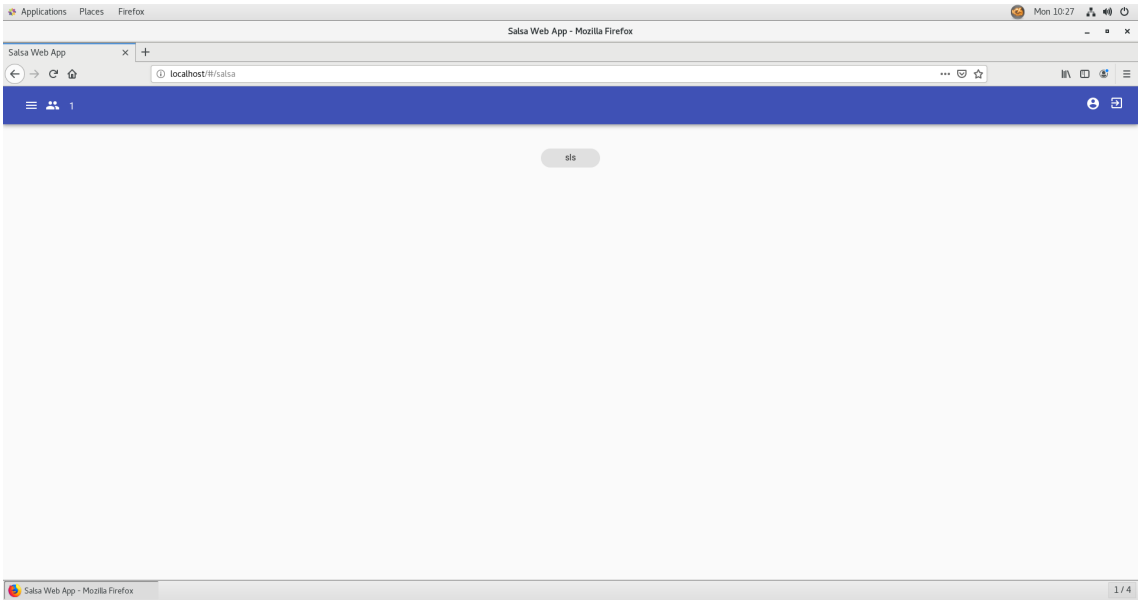
Obr. 18: Tabuĺka v klasickom tvare



Obr. 19: Informácie nad bunkami



Obr. 20: Hlavné menu



Obr. 21: Ponuka služieb

informácie je možný preklik aj na tabuľku s úloha mi **Job queue** alebo časťou pre zadávanie úlohy **Submit** (Obr. 22).

Po kliknutí na **Submit** sa dostaneme do časti pre zadávanie úlohy, kde je možné napísať príkaz, ktorý chceme spustiť a tiež to, koľko krát chceme, aby bol tento príkaz spustený. Pod týmto formulárom pre zadávanie je automaticky vygenerovaný príkaz, ktorý sa dynamicky mení v závislosti od zadaných údajov (Obr. 23).

Po kliknutí na vygenerovaný príkaz je tento príkaz skopírovaný do pamäti, pričom tento je možné spustiť v termináli. (Obr. 24).

Po sputení príkazu v termináli je možné sledovať v termináli ukazovateľ dokončenia úlohy, pričom tieto informácie je možné vidiť aj vo webovej aplikácií vo forme tabuľky (Obr. 25).

Okrem spustenia úlohy cez terminál je možné spustiť úlohu pomocou tlačidla **Execute job**, pričom po stlačení sa pošle HTTP požiadavka na API server a ten následne spustí úlohu na klastri. Taktiež po kliknutí na toto tlačidlo sa používateľ dostane do časti tejto stránky s tabuľkou úloh, ktoré sú alebo budú vykonávané.

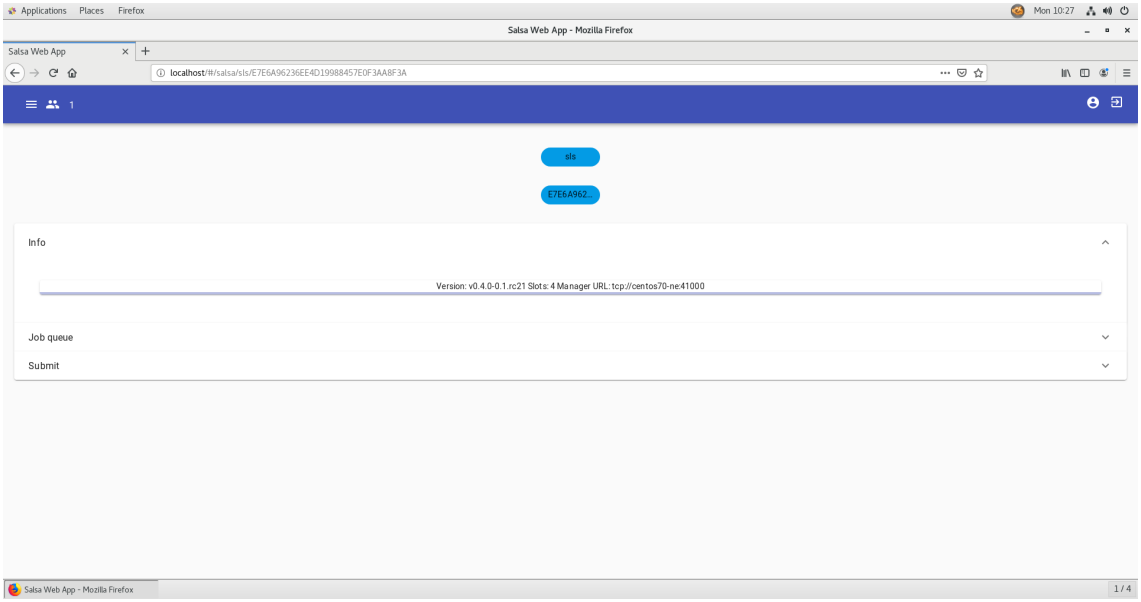
5 Monitorvanie úlohy na klastri

Keďže táto webová aplikácia ponúka okrem samotného generovania príkazov úloh aj monitorovanie stavu týchto úloh, tak pre monitorovanie úloh je potrebné kliknúť na stránke v hlavnom menu na položku **Salsa** (Obr. 26).

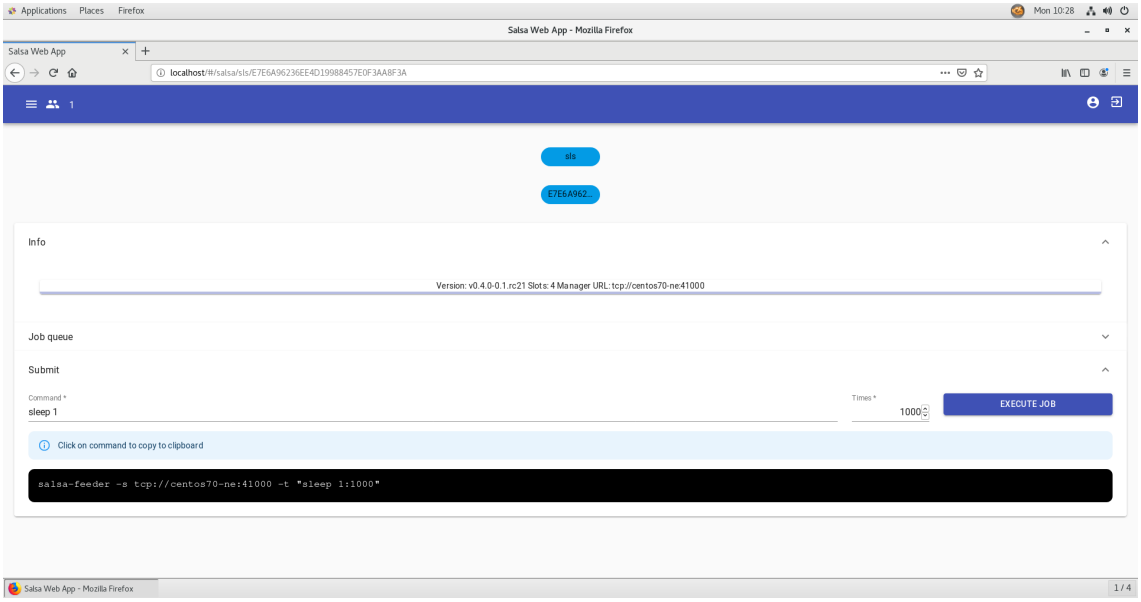
Po vybratí si služby a podslužby, je potrebné kliknúť na položku **Job queue** na stránke (Obr. 27).

V prípade, že je používateľ prihlásený, môže vidiť zaškrtávacie okienka pre výber iba tých úloh, ktoré on sám spustil a taktiež kompaktnejší tvar tabuľy. Ak je zaškrtnuté políčko o sledovaní iba vlastných úloh a nespustil žiadnu úlohu, tabuľka nebude zobrazená a miesto tabuľky je zobrazená informácia o tom, že žiadna úloha nie je vo fronte (Obr. 28).

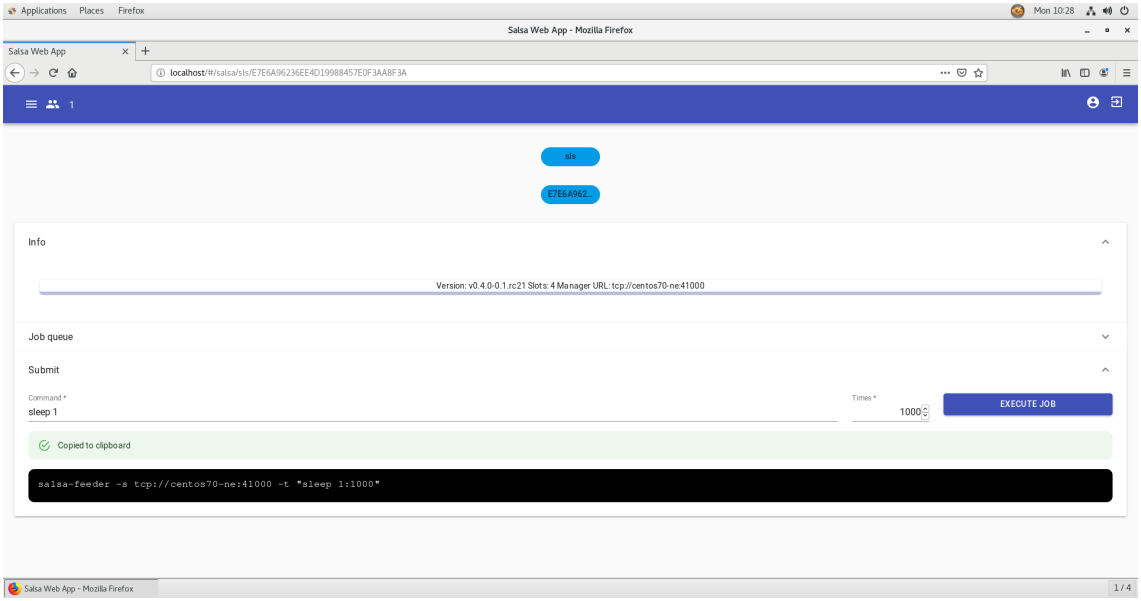
Akonáhle daný používateľ spustí úlohu, bude zobrazená tabuľka s informáciami o tom, kto danú úlohu spustil, koľko podúloh obsahuje úloha, koľko ešte čaká na vykonanie, koľko sa vykonáva, koľko



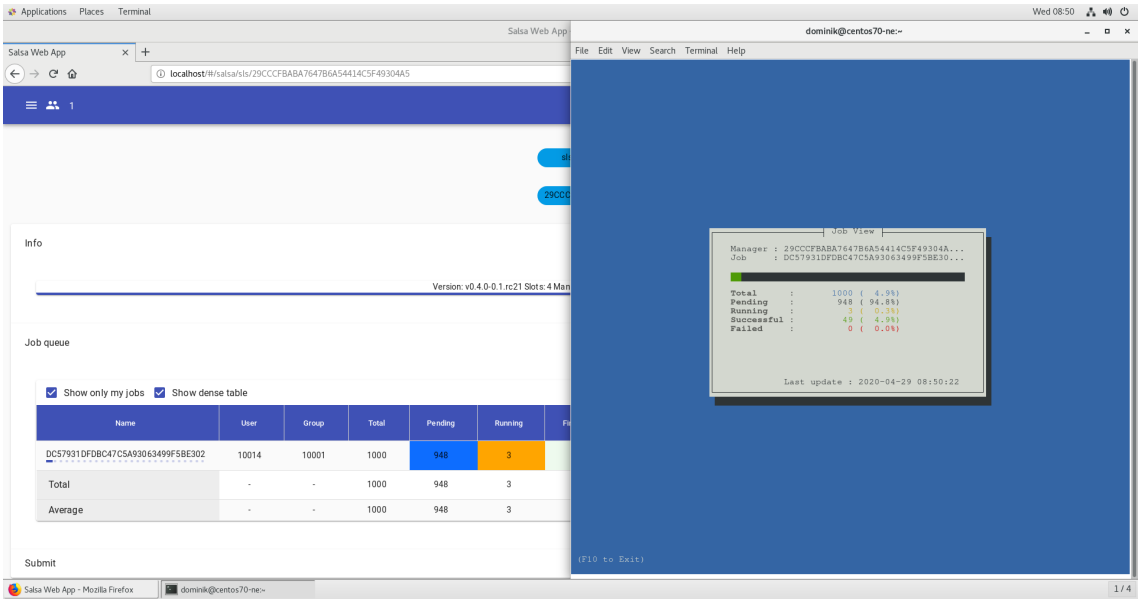
Obr. 22: Ponuka služieb



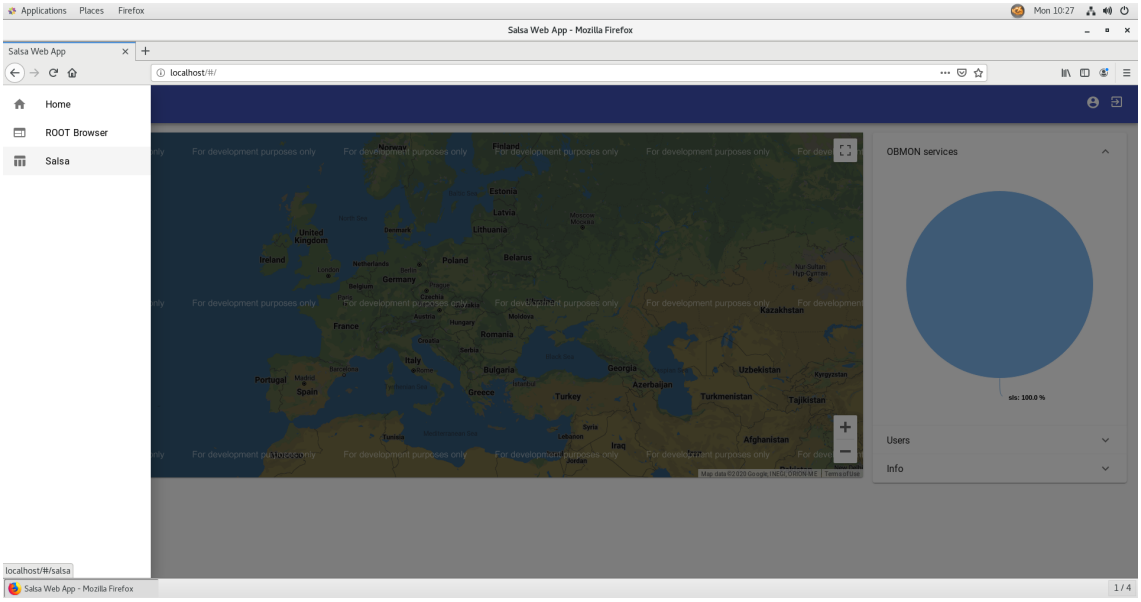
Obr. 23: Zadávanie úlohy



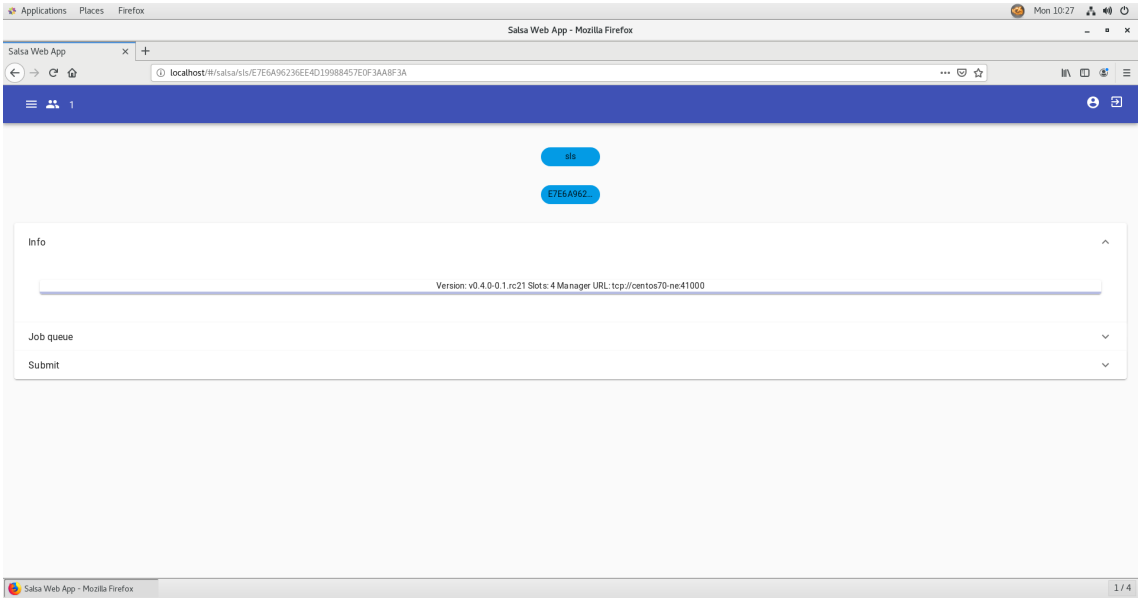
Obr. 24: Skopírovaný príkaz



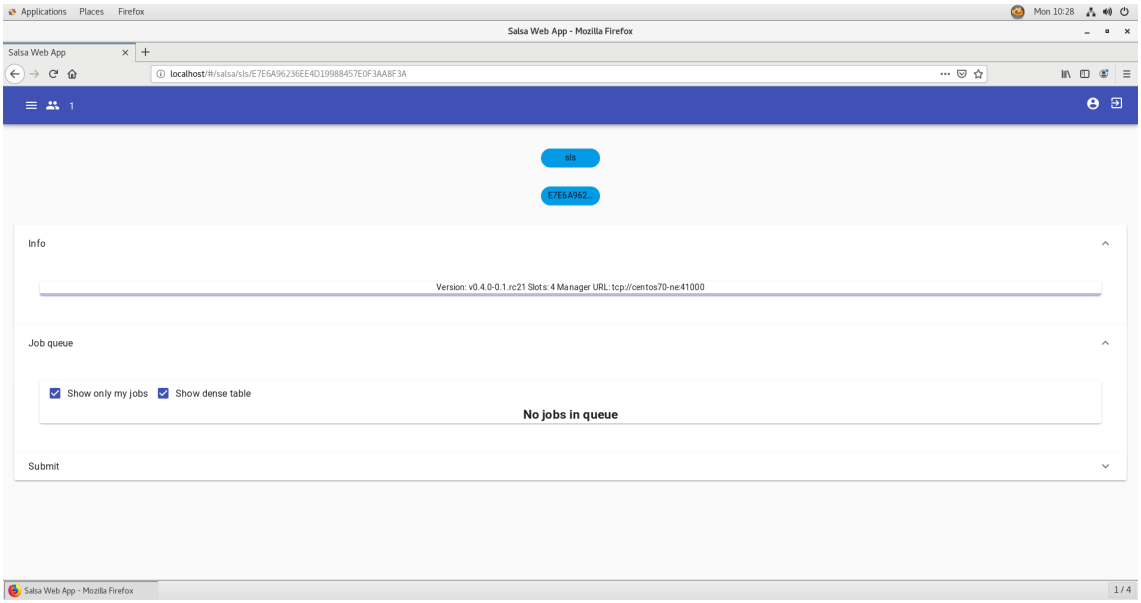
Obr. 25: Spustená úloha



Obr. 26: Hlavné menu



Obr. 27: Ponuka služieb



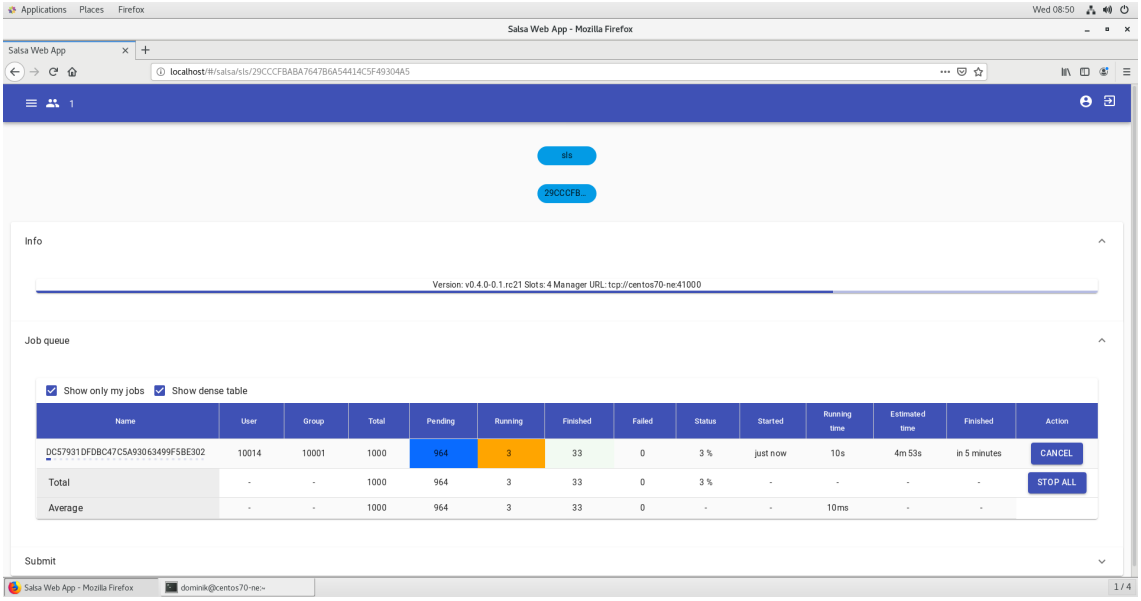
Obr. 28: Žiadne úlohy nie sú spustené

skončilo úspešne, neúspešne, čas štartu, čas od spustenia, približný čas ukončenia a čas ukončenia a možnosť zastavenia úlohy (Obr. 29).

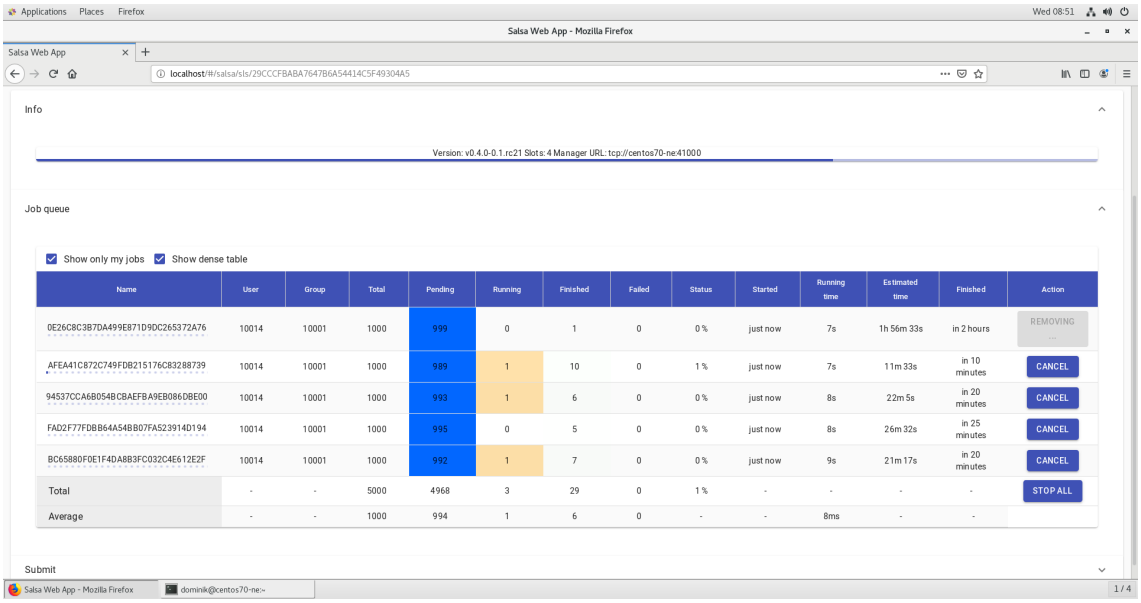
V hornej časti stránky je tiež možné vidieť ukazovateľ vyťaženia klastra pod informáciami o verzii. Taktiež pri prejdení kurzorom nad časom spustenia a časom ukončenia je možné vidieť dátum a čas v klasickom formáte.

Po klinutí na tlačilo **Cancel** (resp. **Remove** v prípade ukončenej úlohy), je možné túto úlohu zmazať z klastra a teda už nebude zobrazovaná (Obr. 30).

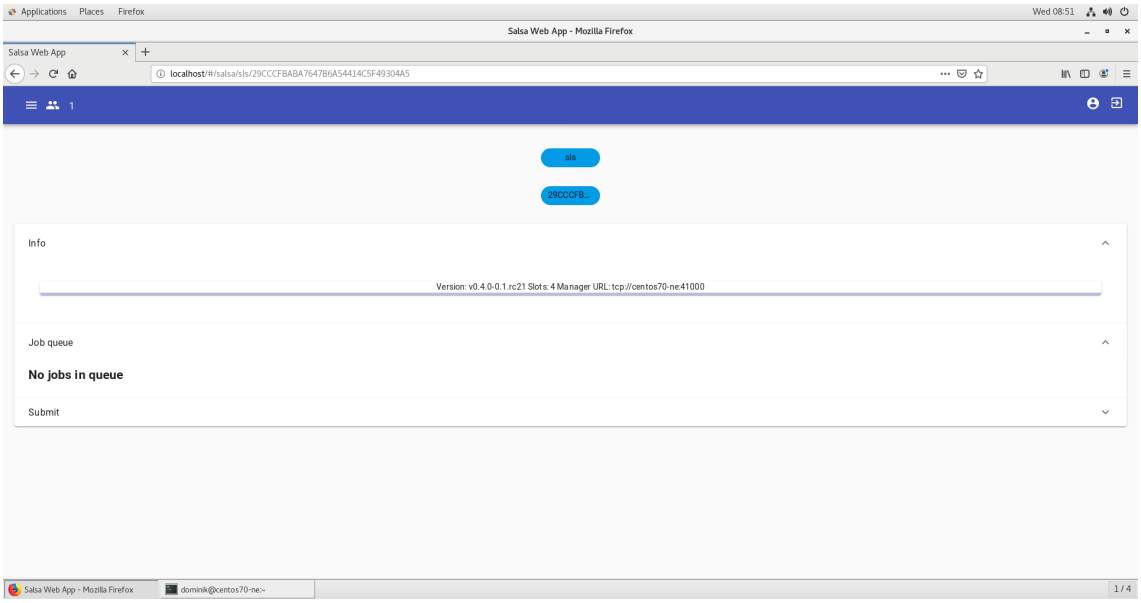
V prípade, že používateľ klikol na tlačidlo **STOP ALL**, resp. vymazal alebo zrušil všetky úlohy, tabuľka bude znovu skrytá a bude zobrazená informácia o tom, že vo fronte nie sú žiadne úlohy (Obr. 31).



Obr. 29: Úloha je spustená



Obr. 30: Rušenie úlohy



Obr. 31: Žiadna úloha vo fronte

Príloha C: Systémová príručka

Dominik Matis

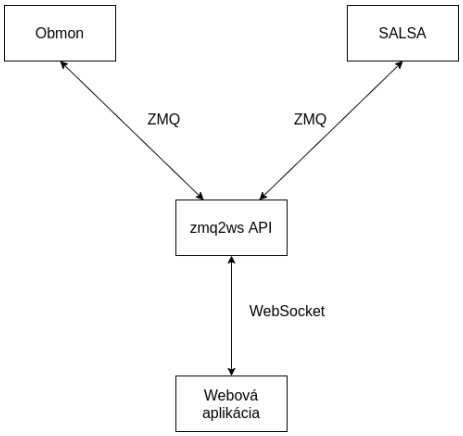
28.05.2020

1 Opis architektúry

Webová aplikácia môže byť spustená pri vývoji pomocou vývojárskeho servera cez príkaz `npm start`, pričom tento HTTP server bude poskytovať túto webovú aplikáciu predvolene na porte 3000. V prípade, že vývojár chce použiť iný port, je potrebné nastaviť environmentálnu premennú `PORT` na číslo požadovaného portu.

Webová aplikácia sa cez WebSocket spojenie pripája na API server, ktorého URL je definovaná v súboroch `env.production.js` pre produkciu, resp. v súbore `env.development.js` pri vývoji. Tieto súbory sa nachádzajú v priečinku `public` a sú importované priamo do `index.html` aplikácie, pričom sa používa `%NODE_ENV%` environmentálna premenná na zistenie, či ide o produkciu alebo vývoj.

API server `zmq2ws` počúva na požiadavky predvolene na porte 8442. Tento server môže mať aj podporu SSL, teda zabezpečenú komunikáciu a tiež môže poskytovať autentifikáciu za pomoci LDAP protokolu. Tieto atribúty je možné meniť pomocou konfigurácie, ktorá sa nachádza v súbore `config.yml`, teda konfigurácia je vo formáte YAML. Okrem týchto atribútov je v tejto konfigurácii meniť aj to, od akých služieb chceme preposielať dáta do webovej aplikácie a tiež, na akom porte počúva tento server na nové informácie od senzorov Obmon, resp. dávkovacieho systému SALSA. Tu je tiež možné zadefinovať URL adresy a názvy služieb, podľa ktorých sú tieto služby triedené vo webovej aplikácii. API server získava nové dáta pomocou knižnice ZMQ, ktorá prepája tento server a Obmon a systém SALSA. Celé prepojenie webovej aplikácie s API serverom a senzorom Obmon a systémom SALSA je možné vidieť na obrázku nižšie.



Obr. 1: Prepojenie častí systému

2 Použité technológie

Je odporúčané, aby vývojár mal verziu Node.js aspoň 10, pričom pri vývoji bola použitá verzia 12.16.3. Verzia NPM použitá pri vývoji je 6.14.4. Pri spúšťaní systému pomocou Dockeru, resp. Podmana, bol použitý Podman verzie 1.9.1. Táto webová aplikácia požaduje viacero NPM balíkov, pričom je vhodné mať kompatibilné verzie týchto balíkov. Zoznam balíkov a ich verzií je spísaný v tomto zozname:

- @material-ui/core 4.9.9
- @material-ui/icons 4.9.1
- @material-ui/lab 4.0.0-alpha.48
- @obl/react-jsroot 0.0.2
- @obl/redux-websocket-middleware 1.0.0
- axios 0.19.2
- google-map-react 1.1.7
- highcharts 8.0.4
- highcharts-react-official 3.0.0
- jwt-decode 2.2.0
- pretty-time 1.1.0
- react 16.13.1
- react-dom 16.13.1
- react-gravatar 2.6.3
- react-redux 7.2.0
- react-router-dom 5.1.2
- react-scripts 3.4.1
- react-time-ago 5.0.7
- redux 4.0.5
- redux-thunk 2.3.0
- uuid 7.0.3

3 Opis vývojového prostredia

Pri vývoji tejto práce bol použitý editor Visual Studio Code, verzia 1.45.1, ktorý je voľne dostupný, má otvorený zdrojový kód a má vstavanú podporu viacerých jazykov, ktorú je možné rozšíriť na ďalšie jazyky pomocou mnohých bezplatných rozšírení.

Okrem samotného jazyka sme použili aj mnohé rozšírenia, ktoré budú spomenuté v nasledujúcich sekciách.

3.1 Debugger for Chrome, Debugger for Firefox

Keďže vývoj nie je vždy jednoduchý a niektoré chyby môžu byť skryté, je potrebné ich odhaľovať čo najskôr. Tieto rozšírenia slúžia na ladenie, pričom povoľujú použitie tzv. bodov prerušenia (angl. *breakpoints*). Výhodou je, že tento editor je priamo spojený s týmito prehliadačmi a ladenie je pomocou týchto rozšírení veľmi jednoduché. Použitá verzia rozšírenia Debugger for Chrome je 4.12.6 a verzia Debugger for Firefox je 2.8.0.

3.2 Git Graph, GitLens

Na správu verzií zdrojového kódu bol použitý nástroj Git. Pre zrýchlenie práce s týmto nástrojom boli použité tieto rozšírenia, ktoré povoľujú vytváranie snímok zdrojového kódu (angl. *snapshot*),

poslanie kódu do vzdialeného repozitára a mnoho ďalšieho, pričom nie je potrebné pre prácu s nástrojom Git otvárať terminál a ani platformu GitLab. V tomto projekte bola použitá verzia Git Graph 1.22.0 a verzia GitLens 10.2.1.

3.3 Prettier

Toto rozšírenie slúži na zjednotenie pravidiel pri písaní zdrojového kódu, pričom formátuje kód v závislosti od týchto pravidiel a je možné ho nastaviť, aby formátoval kód po každom uložení kódu (teda po stlačení Ctrl + S) alebo iba pri spustení príkazu na formátovanie. Použitá verzia tohto rozšírenia je 4.7.0.

3.4 Trailing Spaces

Toto jednoduché rožšírenie má za úlohu upozorniť na biele znaky na konci riadkov, ktoré sú normálne neviditeľné. Upozorňuje na nich tak, že tieto znaky označí červenou farbou. Verzia rozšírenia použitá vo vývoji je 0.3.1.

3.5 Live Share

Celkom populárne rozšírenie od spoločnosti, ktorý VS Code vytvorila, Microsoft. Pri programovaní v skupinách je toto ideálny nástroj na vývoj, pretože toto rozšírenie spôsobí to, že každý kto sa pripojí ku človeku, ktorý spustil zdielanie, bude môcť upravovať v reálnom čase kód v editore toho používateľa. Je to výhodné najmä pri situáciách, keď viacerí programátori programujú spoločne a potrebujú sa koordinovať navzájom, pričom týmto spôsobom pracujú spoločne v jednom editore. Aktuálna verzia rozšírenia, ktorá bola aj použitá je 1.0.2169.

3.6 Settings Sync

V prípade, že vývojár chce konfiguráciu a vzhľad svojho editoru preniesť do iného počítača alebo inému vývojárovi, nie je potrebné aby všetko kopíroval ručne. Toto rozšírenie používa GitHub Gist na uloženie úplnej konfigurácie editoru VS Code spolu so zoznamom nainštalovaných rozšírení a teda po stiahnutí nastavení z Gist sa bude editor správať a vyzerať úplne rovnako ako ten pôvodný. Funguje to aj medzi viacerými verziami editoru, teda klasickou a Insiders verziou (najnovšie aktualizácie, experimentálna verzia). Gist ID pre nastavenie editoru použitého pri vývoji tohto projektu je: **7578aa6c57cec3535805dee702638358**. Verzia Settings Sync je 3.4.3.

3.7 Ostatné nastavenia

Pre písanie práce ku tomuto projektu bol použitý systém LaTeX. Preto pri použití Settings Sync z predchádzajúcej sekcie bude medzi nainsštalovanými rozšíreniami aj LaTeX Workshop. Okrem neho sa tam tiež nachádza Material Icon Theme, ktorými sa zmení štýl ikon súborov v bočnom paneli, ktorý sa nachádza v tomto prípade na pravej strane. Taktiež sú tu rozšírenia na prácu s programom CMake, C/C++, Docker, Python, SSH. Medzi nastaveniami editoru je niekoľko zmien, pričom tieto zahrňujú zmenu písma, skrytie niektorých priečinkov, skrytie omrvinkovej navigácie (angl. *breadcrumbs*) a tiež mini mapy.

4 Zostavenie a nasadenie systému

4.1 Spustenie webovej aplikácie

Pre spustenie webovej aplikácie je potrebné mať nainštalovaný **Node.js** a **NPM**. Inštaláciu balíčkov, ktoré sú potrebné pre chod aplikácie a spustenie vývojárskeho serveru s touto aplikáciou je možné cez nasledujúce príkazy:

```
npm install
npm start
```

Vývojársky server je dostupný na adrese http://localhost:3000

4.2 Spustenie API servera

Získavanie informácií z klastrov je možné za pomoci API servera, pričom nie je potrebná inštalácia, no je potrebné mať nainštalovaný program `docker`. Pre spustenie API servera v pozadí je potrebné spustiť nasledujúci príkaz:

```
docker run --net host registry.gitlab.com/ndmspc/zmq2ws
```

4.3 Spustenie dávkovacieho systému SALSA

Informácie ohľadom klastrov poskytuje dávkovací systém SALSA, ktorý je možné spustiť rovnako pomocou programu `docker` pomocou nasledujúceho príkazu:

```
docker run -d --net host registry.openbrain.sk/salsa/salsa
```

4.4 Spustenie úlohy na klastri

Po tom, ako systém SALSA je spustený, je možné v ňom spustiť úlohu, ktorá bude spustená na klastri. Spustenie je možné príkazom:

```
docker run -it --rm --net host --entrypoint salsa-feeder \
registry.openbrain.sk/salsa/salsa -s tcp://localhost:41000 -t "sleep 1:100"
```

Argument `-s` určuje URL adresu sprostredkovateľa a argument `-t` určuje príkaz na spustenie.

5 Opis kódu

V tejto časti bude opísaný zdrojový kód webovej aplikácie, pričom tu bude zahrnutý každý komponent používaný v aplikácii. Všetky spomenuté komponenty sa nachádzajú v súboroch v priečinku `src/`

5.1 Akcie

Nachádzajú sa v priečinku `actions/`

5.1.1 Súbor `auth.js`

- `loadingLogin` - nastavenie načítavania pri prihlasovaní
- `authError` - nastavenie chyby pri prihlasovaní, chyba je posielaná do funkcie ako argument
- `removeToken` - odstránenie tokenu prihláseného používateľa, odhlásenie

5.1.2 Súbor `salsa.js`

- `onSalsaChange` - nastavenie prijatých dát zo systému SALSA, posielané do funkcie cez argument
- `onSalsaNameChange` - nastavenie sledovanej služby zo systému SALSA, posielané do funkcie cez argument
- `onSalsaSubChange` - nastavenie sledovanej podslužby zo systému SALSA, posielané do funkcie cez argument
- `onSalsaDataChange` - nastavenie prijatých informácií o službe zo systému SALSA, posielané do funkcie cez argument
- `onSalsaReset` - nastavenie predvolených dát o systéme SALSA, posielané do funkcie cez argument

5.1.3 Súbor `websocket.js`

- `writeToSocket` - poslanie dát na API server, dáta posielané do funkcie cez argument

5.1.4 Súbor `zmq2wsServiceInfo.js`

- `getNameService` - nastavenie sledovanej služby, posielané do funkcie cez argument
- `getSubService` - nastavenie sledovanej podslužby, posielané do funkcie cez argument
- `getSrc` - nastavenie zdroju sledovanej služby, posielané do funkcie cez argument
- `getData` - nastavenie prijatých dát od sledovanej služby, posielané do funkcie cez argument
- `getIsSubscribed` - nastavenie informácie o tom, či aplikácia práve sleduje nejakú službu, posielané do funkcie cez argument

5.2 Typy akcií

Nachádzajú sa v priečinku `actionTypes/`

5.2.1 Súbor `auth.js`

- `AUTH_USER`
- `AUTH_ERROR`
- `LOADING_LOGIN`
- `SET_USER_DATA`

5.2.2 Súbor `salsa.js`

- `ON_SALSA_CHANGE`
- `ON_SALSA_RESET`
- `ON_SALSA_NAME_CHANGE`
- `ON_SALSA_SUB_CHANGE`
- `ON_SALSA_DATA_CHANGE`

5.2.3 Súbor `websocket.js`

- `RECEIVE_APP_INFO_SUCCESS`
- `RECEIVED_WEBSOCKET_DATA`
- `WRITE_DATA_WS`

5.2.4 Súbor `zmq2wsServiceInfo.js`

- `GET_ZMQ2SERVICE_NAME`
- `GET_ZMQ2SERVICE_SUB`
- `GET_ZMQ2SERVICE_SRC`
- `GET_ZMQ2SERVICE_DATA`
- `GET_ZMQ2SERVICE_ISSUBSCRIBED`

5.3 Reduktory

Nachádzajú sa v priečinku `reducers/`

- súbor `auth.js`
- súbor `salsa.js`
- súbor `websocket.js`
- súbor `zmq2wsServiceInfo.js`

5.4 Selektory

Nachádzajú sa v priečinku `selectors/`

5.4.1 Súbor `auth.js`

- `getAuthErrorMessage` - získanie chyby pri prihlasovaní
- `getAuthenticatedToken` - získanie autentifikačného tokenu
- `getLoadingLogin` - získanie informácie o načítavaní
- `getUserData` - získanie dát o používateľovi

5.4.2 Súbor `salsa.js`

- `getRedirectorUrl` - získanie URL adresy sprostredkovateľa zo systému SALSA
- `getSalsaName` - získanie sledovanej služby zo systému SALSA
- `getSalsaSub` - získanie sledovanej podslužby zo systému SALSA

5.4.3 Súbor `websocket.js`

- `getAppInfo` - získanie informácií o aplikácií
- `getWsConnected` - získanie informácií o tom, či je aplikácia pripojená ku API
- `getWebsocketId` - získanie informácií o identifikátore WebSocket spojenia

5.4.4 Súbor `zmq2wsServiceInfo.js`

- `getServiceName` - získanie názvu sledovanej služby
- `getServiceSub` - získanie názvu sledovanej podslužby
- `getServiceSrc` - získanie typu sledovanej služby
- `getServiceData` - získanie informácií o službe
- `getServiceIsSub` - získanie informácie o tom, či aplikácia sleduje službu

5.5 Sklad

Nachádza sa v priečinku `store/`

- súbor `configureStore.js` - prepája reduktory a vytvára globálny stav aplikácie

6 Komponenty

V tejto časti sú rozpísané komponenty aplikácie React

6.1 Priečinok `atoms/`

6.1.1 `Button`

- tlačidlo slúžiace na zastavenie / zrušenie úlohy
- prípustné parametre
  - `children` - text tlačidla
  - `jobId` - identifikátor úlohy
  - `task` - text tlačidla

6.1.2 `GpusTooltip`

- zobrazenie informácie o využití grafických kariet, zobrazí sa pri prejdení kurzorom nad bunkou tabuľky s údajmi o GPU
- prípustné parametre
  - `gpus` - pole objektov so všetkými grafickými kartami daného uzlu v klastri

6.1.3 `GpusTotalTooltip`

- zobrazenie informácie o celkovej pamäti grafických kariet, zobrazí sa pri prejdení kurzorom nad bunkou tabuľky s údajmi o GPU
- prípustné parametre
  - `gpus` - pole objektov so všetkými grafickými kartami daného uzlu v klastri

6.1.4 `GpusUsedTooltip`

- zobrazenie informácie o použitej pamäti grafických kariet, zobrazí sa pri prejdení kurzorom nad bunkou tabuľky s údajmi o GPU
- prípustné parametre
  - `gpus` - pole objektov so všetkými grafickými kartami daného uzlu v klastri

6.1.5 `Item`

- položka bočného menu, pričom sa zobrazuje spolu s ikonou
- prípustné parametre
  - `name` - názov položky v menu
  - `icon` - názov ikony položky (používané Material-UI ikony)
  - `url` - adresa, na ktorú je používateľ presmerovaný po kliknutí na položku

6.1.6 `SalsaTableHeader`

- zobrazenie hlavičky tabuľky o dátach zo systému SALSA

6.1.7 `SalsaTooltip`

- zobrazenie informácie o ukončených podúlohách v rámci úlohy, zobrazí sa pri prejdení kurzorom nad bunkou počtu dokončených, resp. počtu neúspešne dokončených úloh
- prípustné parametre
  - `data` - pole identifikátorov podúloh

6.1.8 `TableCell`

- bunka tabuľky so špeciálnym štýlovaním

6.1.9 `TableHeader`

- zobrazenie hlavičky tabuľky o dátach zo senzorov Obmon

6.1.10 `TableSubHeader`

- zobrazenie druhej hlavičky tabuľky o dátach zo senzorov Obmon

6.2 Priečinok `molecules/`

6.2.1 `BytesInAdapter`

- zobrazenie informácie o rýchlosti sťahovania jednotlivých sieťových rozhraní uzlu, zobrazí sa pri prejdení kurzorom nad bunkou o sieti smerom do uzlu
- prípustné parametre
  - `adapter` - pole objektov so sieťovými rozhraniami

6.2.2 `BytesOutAdapter`

- zobrazenie informácie o rýchlosti nahrávania jednotlivých sieťových rozhraní uzlu, zobrazí sa pri prejdení kurzorom nad bunkou o sieti smerom z uzlu
- prípustné parametre
  - `adapter` - pole objektov so sieťovými rozhraniami

6.2.3 `DataRenderer`

- zobrazenie tabuľky s dátami zo senzorov Obmon a tiež zaškrtávacie okno pre zmenu tvaru tabuľky

6.2.4 `DiskRead`

- zobrazenie informácie o rýchlosti čítania jednotlivých diskov uzlu, zobrazí sa pri prejdení kurzorom nad bunkou o čítaní disku
- prípustné parametre
  - `disks` - pole objektov s jednotlivými diskami

6.2.5 `DiskWrite`

- zobrazenie informácie o rýchlosti zápisu na jednotlivé disky uzlu, zobrazí sa pri prejdení kurzorom nad bunkou o zapisovaní na disk
- prípustné parametre
  - `disks` - pole objektov s jednotlivými diskami

6.2.6 `Navbar`

- hlavný navigačný panel umiestnený na hornom okraji aplikácie

6.2.7 `PieChart`

- zobrazenie koláčového grafu s informáciami o jednotlivých dostupných službách

6.2.8 `ProfileCard`

- zobrazenie karty s údajmi prihláseného používateľa
- prípustné parametre
  - `profile` - objekt s údajmi o používateľovi

6.2.9 `SalsaHosts`

- zobrazenie informácie o počtoch jadier jednotlivých uzlov klastra, zobrazí sa pri prejdení kurzorom nad informáciou o klastri
- prípustné parametre
  - `hosts` - pole objektov s údajmi o uzloch

6.2.10 `SalsaSubmit`

- zobrazenie komponentu pre skladanie príkazu úlohy

6.2.11 `SalsaTableContent`

- zobrazenie tabuľky s jednotlivými úlohami na klastri
- prípustné parametre
  - `tasksData` - pole objektov s úlohami na danom klastri

6.2.12 `ServiceSelector`

- zobrazenie tlačidiel pre výber služby a podslužby na sledovanie zmien

6.2.13 `Sidebar`

- bočné menu obsahujúce položky aplikácie
- prípustné parametre
  - `open` - premenná obsahujúca stav viditeľnosti bočného menu
  - `toggle` - funkcia na zmenu viditeľnosti bočného menu

6.2.14 `TableContent`

- zobrazenie tabuľky s informáciami o jednotlivých uzloch klastra

6.2.15 `UsersTable`

- zobrazenie tabuľky s informáciami o jednotlivých používateľoch aplikácie
- prípustné parametre
  - `clientsData` - pole objektov používateľov

6.3 Priečinok `organisms/`

6.3.1 `SalsaInfo`

- zobrazenie informácií o klastri ako sú verzia, počet jadier, URL adresa sprostredkovateľa a tiež ukazovateľ postupu vyťaženia klastra
- prípustné parametre
  - `serviceData` - objektov s údajmi o klastri
  - `jobs` - pole objektov s jednotlivými úlohami

6.3.2 `SalsaPanels`

- komponent s panelmi, ktoré obsahujú informácie o klastri, tabuľku úloh na klastri a tiež komponent pre získanie príkazu na spustenie úlohy

6.3.3 `SalsaTable`

- zobrazenie tabuľky s informáciami o úlohách daného klastra

6.4 Priečinok `pages/`

6.4.1 `ApiInfoPage`

- stránka s informáciou o pripojených používateľoch a dostupných službách
- dostupná na domovskej stránke a adrese `/api/info`

6.4.2 `LoginPage`

- stránka s prihlasovacím formuláom
- dostupná na adrese `/auth`

6.4.3 `ObmonTablePage`

- stránka s tabuľkou o jednotlivých uzloch klastra od senzorov Obmon
- dostupná na adrese `/obmon`

6.4.4 `ProfilePage`

- stránka s používateľským profilom
- dostupná na adrese `/profile`

6.4.5 `RootBrowserPage`

- stránka s prehliadačom súborov softvéru ROOT
- dostupná na adrese `/root`

6.4.6 `SalsaTablePage`

- stránka s komponentmi pre správu úloh na klastri
- dostupná na adrese `/salsa`

6.5 Priečinok `utils/`

6.5.1 Priečinok `obmon/`

6.5.2 `averageRowList`

- funkcia na vygenerovanie priemeru dát pre tabuľku o klastroch z dát zo senzorov Obmon
- prípustné parametre
  - `tableRowsData` - pole objektov s údajmi o uzloch klastra
  - `classes` - objekt s triedami pre štýlovanie
  - `findInsideTableRows` - keďže niektoré uzly obsahujú GPU informácie, je potrebné riadky s uzlami doplniť prázdnymi bunkami

6.5.3 `generateRowData`

- funkcia na transformovanie dát do vhodného formátu pre tabuľku
- prípustné parametre
  - `data` - objekt s informáciou o uzle
  - `tableData` - pole objektov s uzlami, slúži na zistenie, či je nejaký uzol s GPU
  - `id` - jednoznačný identifikátor riadku tabuľky

6.5.4 `gpuTableCellList`

- funkcia na vygenerovanie dát o GPU pre tabuľku o klastroch z dát zo senzorov Obmon
- prípustné parametre
  - `row` - objekt s informáciou o uzle
  - `classes` - objekt s triedami pre štýlovanie

6.5.5 `tableCellList`

- funkcia na vygenerovanie dát pre tabuľku o klastroch z dát zo senzorov Obmon
- prípustné parametre
  - `row` - objekt s informáciou o uzle
  - `classes` - objekt s triedami pre štýlovanie

6.5.6 `totalRowList`

- funkcia na vygenerovanie celkových dátach pre tabuľku o klastroch z dát zo senzorov Obmon
- prípustné parametre
  - `tableRowsData` - pole objektov s údajmi o uzloch klastra
  - `classes` - objekt s triedami pre štýlovanie
  - `findInsideTableRows` - keďže niektoré uzly obsahujú GPU informácie, je potrebné riadky s uzlami doplniť prázdnymi bunkami

6.5.7 Priečinok `salsa/`

6.5.8 `averageTaskList`

- funkcia na vygenerovanie priemeru dát pre tabuľku o klastroch z dát o úlohách
- prípustné parametre
  - `tasks` - pole objektov s úlohami
  - `classes` - objekt s triedami pre štýlovanie

6.5.9 `taskCellList`

- funkcia na vygenerovanie dát pre tabuľku o klastroch z dát o úlohách
- prípustné parametre
  - `task` - objekt s informáciou o úlohe
  - `classes` - objekt s triedami pre štýlovanie
  - `tasksData` - pole objektov s úlohami na danom klastri

6.5.10 `tasksData`

- funkcia `generateTasksData` slúži na transformovanie dát o úlohách do vhodného formátu pre tabuľku
- prípustné parametre
  - `jobs` - pole objekt s informáciou o úlohách

6.5.11 `totalTaskList`

- funkcia na vygenerovanie celkových dátach pre tabuľku o klastroch z dát o úlohách
- prípustné parametre
  - `tasks` - pole objektov s údajmi o úlohách
  - `classes` - objekt s triedami pre štýlovanie

6.6 Súbor `utils.js`

6.6.1 `precisionRound`

- funkcia na zaokrúhlenie čísla na počet desatinných miest zadaným v argumente
- prípustné parametre
  - `number` - číslo na zaokrúhlenie
  - `precision` - presnosť zaokrúhlenia

6.6.2 `bytesToSize`

- funkcia na výpis veľkosti v prijateľnom tvare
- prípustné parametre
  - `bytes` - počet bajtov

6.6.3 `bytesToSizeInSeconds`

- funkcia na výpis rýchlosti toku dát v prijateľnom tvare
- prípustné parametre
  - `bytes` - počet bajtov za sekundu

6.6.4 `decimalPlace`

- funkcia na výpis čísla s určitým počtom desatinných miest
- prípustné parametre
  - `num` - číslo
  - `decimal` - počet desatinných miest
  - `alpha` - či má toto číslo slúžiť ako alfa, napr. pre `rgba` funkciu v CSS

6.6.5 `readableByteInSeconds`

- funkcia na výpis rýchlosti toku dát v prijateľnom tvare
- prípustné parametre
  - `bytes` - počet bajtov za sekundu

6.6.6 `removeDuplicates`

- funkcia na odstránenie duplicitných dát v poli
- prípustné parametre
  - `array` - pole
  - `property` - vlastnosť na odstránenie duplikátov

6.6.7 `bytesInDisk`

- funkcia na získanie hodnoty čítania z diskov, zoznamu diskov a hodnote alfa pre výpis v tabuľke
- prípustné parametre
  - `data` - informácie o diskoch

6.6.8 `bytesOutDisk`

- funkcia na získanie hodnoty zápisu na disk, zoznamu diskov a hodnote alfa pre výpis v tabuľke
- prípustné parametre
  - `data` - informácie o diskoch

6.6.9 `bytesIn`

- funkcia na získanie hodnoty sťahovania zo siete, zoznamu diskov a hodnote alfa pre výpis v tabuľke
- prípustné parametre
  - `data` - informácie o sieti

6.6.10 `bytesOut`

- funkcia na získanie hodnoty nahrávania na sieť, zoznamu diskov a hodnote alfa pre výpis v tabuľke
- prípustné parametre
  - `data` - informácie o sieti

6.6.11 `GpuLoad`

- funkcia na získanie hodnoty využitia GPU a zoznamu diskov pre výpis v tabuľke
- prípustné parametre
  - `data` - informácie o grafických kartách

6.6.12 `GpuUsedMemory`

- funkcia na získanie hodnoty množstva využitej GPU pamäte, zoznamu diskov a hodnote alfa pre výpis v tabuľke
- prípustné parametre
  - `data` - informácie o grafických kartách

6.6.13 `GpuTotalMemory`

- funkcia na získanie hodnoty množstva celkovej GPU pamäte a zoznamu diskov pre výpis v tabuľke
- prípustné parametre
  - `data` - informácie o grafických kartách

6.6.14 `calculatedUsedValue`

- funkcia na získanie hodnoty alfa a využitej pamäte
  - `data` - informácie o grafických kartách

6.6.15 `generateBackgroundColor`

- funkcia na získanie hodnoty RGBA farby pre bunku tabuľky
  - `cores` - počet jadier
  - `load` - celková záťaž

6.6.16 `generateCpuAverage`

- funkcia na získanie priemernej hodnoty atribútu CPU
  - `tableRows` - pole objektov
  - `property` - vlastnosť, na získanie priemernej hodnoty

6.6.17 `generateMemoryAverageAndTotal`

- funkcia na získanie priemernej alebo celkovej hodnoty pamäte
  - `tableRows` - pole objektov
  - `property` - vlastnosť, na získanie priemernej hodnoty
  - `type` - použité na zistenie, či je potrebná celková hodnota alebo nie

6.6.18 `generateNetworkAverageAndTotal`

- funkcia na získanie priemernej alebo celkovej hodnoty atribútu sieťovej premávky
  - `tableRows` - pole objektov
  - `property` - vlastnosť, na získanie priemernej hodnoty
  - `type` - použité na zistenie, či je potrebná celková hodnota alebo nie

6.6.19 `generateDisksAverageAndTotal`

- funkcia na získanie priemernej alebo celkovej hodnoty atribútu diskov
  - `tableRows` - pole objektov
  - `property` - vlastnosť, na získanie priemernej hodnoty
  - `type` - použité na zistenie, či je potrebná celková hodnota alebo nie

6.6.20 `generateTotalNumberLoad1`

- funkcia na získanie celkovej hodnoty záťaže
  - `tableRows` - pole objektov
  - `property` - vlastnosť, na získanie priemernej hodnoty
  - `type` - použité na zistenie, či je potrebná celková hodnota alebo nie

6.6.21 `calculatedCoresTotalAndAverage`

- funkcia na získanie celkového alebo priemrného počtu jadier
  - `tableRows` - pole objektov
  - `property` - vlastnosť, na získanie priemernej hodnoty
  - `type` - použité na zistenie, či je potrebná celková hodnota alebo nie

6.6.22 `arrayToRangeString`

- funkcia na získanie celkového alebo priemrného počtu jadier
  - `vector` - pole identifikátorov úloh

6.7 Hlavný komponent `App`

- obsahuje celú aplikáciu
- obsahuje smerovanie v rámci aplikácie a tiež navigačný panel