MASARYKOVA
UNIVERZITA

FAKULTA INFORMATIKY

Multiagentný systém na detekciu sieťovej prevádzky pomocou RIPE Atlas sond

Diplomová práca

BC. MARTIN KAŽIMÍR

Brno, jar 2023

MASARYKOVA
UNIVERZITA

FAKULTA INFORMATIKY

Multiagentný systém na detekciu sieťovej prevádzky pomocou RIPE Atlas sond

Diplomová práca

BC. MARTIN KAŽIMÍR

Vedúci práce: Ing. Václav Oujezský, Ph.D.

Brno, jar 2023

FACULTAS ARTIS INFORMATICÆ • UNIVERSITAS MASARYKIANA •

Vyhlásenie

Vyhlasujem, že táto diplomová práca je mojím pôvodným autorským dielom, ktoré som vypracoval samostatne. Všetky zdroje, pramene a literatúru, ktoré som pri vypracovaní používal alebo z nich čerpal, v práci riadne citujem s uvedením úplného odkazu na príslušný zdroj.

Bc. Martin Kažimír

Vedúci práce: Ing. Václav Oujezský, Ph.D.

Poďakovanie

Moje poďakovanie patrí vedúcemu diplomovej práce Ing. Václavovi Oujezskému, Ph.D. za ochotu, usmernenia a rady pri tvorbe tejto práce. Ďalej by som sa chcel poďakovať Fakulte informatiky Masarykovej univerzity za poskytnuté zdroje. Ďalšie výpočtové zdroje boli poskytnuté v rámci projektu e-INFRA CZ (ID:90140), ktorý podporilo Ministerstvo školstva, mládeže a telovýchovy Českej republiky. Taktiež by som sa rád poďakoval rodičom, sestre a hlavne mojej priateľke za podporu, ktorú mi poskytujú.

Zhrnutie

Cieľom diplomovej práce je návrh a vývoj aplikácie, ktorá slúži na detekciu sieťovej prevádzky a anomálií pomocou sieťových RIPE sond. V teoretickej časti práca prezentuje projekt RIPE Atlas a analyzuje súčasný stav jeho využitia. Praktická časť práce popisuje motiváciu pre vytvorenie vlastnej aplikácie, špecifikuje jej požiadavky a následne popisuje návrh, implementáciu a nasadenie samotnej aplikácie.

Kľúčové slová

RIPE Atlas, RIPE sonda, sieťová prevádzka, sieťové anomálie, používateľské rozhranie, aplikačné rozhranie, multiagentný systém, framework, Python, React, MongoDB

Obsah

Úvod **4**

1 Súčasný stav **5**
1.1 RIPE Atlas sondy 5
1.1.1 Hardvérové sondy 5
1.1.2 Softvérové sondy 6
1.1.3 Merania na sonde 7
1.2 Aplikačné rozhranie prostredia RIPE Atlas 7
1.3 Anomálie v sieti 8
1.3.1 Základné techniky detekcie anomálií 8
1.4 Systém Disco 10
1.5 Platforma RIPE Alerts 10

2 Vlastné sondy **11**
2.1 Nasadenie softvérovej sondy 11

3 Motivácia a požiadavky pre vlastnú aplikáciu **12**
3.1 Funkčné požiadavky na server 13
3.2 Funkčné požiadavky na agenta 13
3.3 Nefunkčné požiadavky na aplikáciu 14

4 Návrh aplikácie **15**

5 Návrh agenta **17**
5.1 Konfigurácia 17
5.2 Interná štruktúra 18
5.3 Odosielanie výsledkov 19
5.4 Aplikačné rozhranie 19

6 Návrh serveru **21**
6.1 Databáza 21
6.2 Back-end 22
6.2.1 Konfigurácia 22
6.2.2 Aplikačné rozhranie 24
6.3 Front-end 26

6.3.1 Prihlasovacia stránka 26
6.3.2 Stránka – detekčné úlohy 26
6.3.3 Stránka – detail detekčnej úlohy 28
6.3.4 Stránka – agenti 30

7 Implementácia agenta **32**
7.1 Plánovač úloh 32
7.2 Aplikačné rozhranie 33
7.3 Zabezpečenie aplikačného rozhrania 33
7.3.1 JSON Web Token 34
7.4 Prostriedky pre nasadenie 36
7.5 Implementácia ukážkovej úlohy 37

8 Implementácia serveru **38**
8.1 Databáza 38
8.2 Back-end 38
8.3 Front-end 39
8.3.1 React 39
8.3.2 Bootstrap 40
8.3.3 Vite 40

9 Praktické nasadenie a zhodnotenie **43**
9.1 Overenie funkčnosti 43
9.2 Obmedzenia pri vytváraní meraní 45
9.3 Zhodnotenie 45

Záver **46**

Bibliografia **47**

A Elektronická príloha **50**

B Ukážka vytvoreného používateľského rozhrania **51**

Zoznam obrázkov

1.1 Hardvérová RIPE Atlas sonda [5] 6

4.1 Prehľad štruktúry aplikácie 15

6.1 Prehľad prvkov databázy 22
6.2 Návrh prihlasovacej stránky 26
6.3 Návrh stránky zobrazujúcej zoznam detekčných úloh . . 27
6.4 Návrh stránky zobrazujúcej dialóg pre zmazanie úlohy . 27
6.5 Návrh stránky zobrazujúcej dialóg pre vytvorenie novej detekčnej úlohy 28
6.6 Návrh stránky zobrazujúcej detail detekčnej úlohy 29
6.7 Návrh stránky zobrazujúcej zoznam agentov 30
6.8 Návrh stránky zobrazujúcej dialóg pre pridanie nového agenta 31

7.1 JWT v nezakódovanej forme 35
7.2 JWT v zakódovanej forme 36

8.1 Klasický prístup pre budovanie aplikácie [27] 41
8.2 Prístup nástroja Vite pre budovanie aplikácie [27] 42

9.1 Výsledky detekčnej úlohy 44

B.1 Prihlasovacia stránka 51
B.2 Stránka zobrazujúca zoznam detekčných úloh 51
B.3 Stránka zobrazujúca dialóg pre zmazanie detekčnej úlohy 52
B.4 Stránka zobrazujúca dialóg pre vytvorenie novej detekčnej úlohy 52
B.5 Stránka zobrazujúca detail detekčnej úlohy 53
B.6 Stránka zobrazujúca zoznam agentov 54
B.7 Stránka zobrazujúca dialóg pre pridanie nového agenta . 54

Zoznam skratiek

API Application Programming Interface 7, 13, 14, 16, 17, 19, 20, 26, 33, 39, 46

AS Autonomous System 10

ASN Autonomus System Number 11

BSON Binary JavaScript Object Notation 38

CESNET Czech Education and Scientific NETwork 11, 43

CSS Cascading Style Sheets 40

DNS Domain Name System 7

DOM Document Object Model 40

DX Developer Experience 40

EMA Exponential Moving Average 8

ESM ECMAScript Modules 42

HTML HyperText Markup Language 39

HTTP Hypertext Transfer Protocol 7

I/O input/output 33

ICMP Internet Control Message Protocol 8

JS JavaScript 39, 40, 41, 42

JSON JavaScript Object Notation 19, 21, 34, 36, 38

JSX JavaScript Syntax Extension 39

JWT JSON Web Token viii, 17, 23, 33, 34, 35, 36

NAT Network Address Translation 10

NTP Network Time Protocol 7

REST Representational state transfer 7

RFC Request For Comments 9, 34

RIPE NCC Réseaux IP Européens Network Coordination Centre 5

RIR Regional Internet registry 5

RTT Round-trip time 7, 8, 9

SMA Simple Moving Average 8

SSL/TLS Secure Sockets Layer/Transport Layer Security 7

TCP Transmission Control Protocol 9, 10

URL Uniform Resource Locator 7, 18, 23

VDOM Virtual Document Object Model 40

Zoznam ukážok

5.1 Konfiguračný súbor pre agenta 18
5.2 Štruktúra objektu reprezentujúceho výsledok úlohy . . 19
6.1 Konfiguračný súbor pre server 24
7.1 Odpoveď koncového bodu /auth 34
7.2 Autorizačná hlavička 34
7.3 JSON reprezentujúci výsledok úlohy 37
8.1 Súbor vite.config.js 41
9.1 JSON chybová odpoveď 45

Úvod

Sledovanie a analýza sieťovej prevádzky je v súčasnosti už nevyhnutný proces pre zaistenie správneho fungovania moderných počítačových sieti. Proces umožňuje získanie dôležitých informácií o aktuálnom stave siete, ktoré sú nevyhnutné jednak pre samotné identifikovanie problémov, ako sú anomálie, či iné potenciálne nebezpečné aktivity, tak aj pre identifikovanie príčin týchto problémov.

Znalosť informácií je kľúčová nie len pre rýchle riešenie problémov, ale aj k optimalizácii a zaisteniu spoľahlivosti a bezpečnosti siete do budúcnosti. Získavaniu týchto informácií napomáha aj projekt RIPE Atlas[1], ktorého cieľom je meranie rozličných parametrov siete a monitorovanie jej prevádzky v reálnom čase, z rôznych sond po celom svete.

Predmetom práce je návrh a vývoj aplikácie, pozostávajúcej z jednej inštancie komponenty serveru a viacerých inštancií komponenty agenta, ktorá umožňuje detekciu sieťovej prevádzky a anomálií prostredníctvom RIPE sond. Cieľom je vytvorenie aplikačného rámca (angl. framework), ktorý pripraví prostredie s viacerými agentmi a poslúži pre ďalší vývoj a zlepšovanie.

Práca je členená do deviatich kapitol. Prvá kapitola bližšie popisuje projekt RIPE Atlas, ako aj súčasné spôsoby jeho využitia. Nasleduje krátka kapitola predstavujúca vlastné RIPE sondy. Tretia kapitola bližšie popisuje motiváciu pre vytvorenie vlastnej aplikácie a zároveň definuje jej funkčné a nefunkčné požiadavky. Štvrtá kapitola predstavuje návrh celej aplikácie, na ňu následne nadväzujú dve kapitoly, ktoré podrobnejšie predstavujú návrh jednotlivých komponent aplikácie, piata kapitola, venujúca sa návrhu komponenty agenta a šiesta kapitola venujúca sa návrhu komponenty serveru. Siedma kapitola popisuje vybrané aspekty implementácie komponenty agenta a následne ôsma kapitola, popisuje implementáciu jednotlivých vrstiev komponenty serveru. Záverečná deviata kapitola sumarizuje nasadenie a zhodnotenie implementácie celej aplikácie.

1. `https://atlas.ripe.net`

1 Súčasný stav

Projekt RIPE Atlas má v súčasnosti viacero spôsobov využitia pre detekciu anomálií. Infraštruktúru projektu RIPE Atlas tvorí globálna sieť zariadení, ktoré sa nazývajú sondy (angl. probes). Jednotlivé sondy vykonávajú merania rôzneho typu, ktoré sú bližšie popísané v sekcii 1.1.3. Na základe meraní, sondy poskytujú informácie o konektivite a dostupnosti internetu v reálnom čase, čo umožňuje lepšie porozumenie súčasného stavu internetu [1].

Za iniciatívou stojí medzinárodná nezisková organizácia Réseaux IP Européens Network Coordination Centre (RIPE NCC), čo je organizácia fungujúca ako regionálny internetový register (angl. Regional Internet registry) pre Európu, Blízky Východ a centrálnu Áziu [2] [3].

Výsledky z RIPE Atlas meraní sú použité v pomerne mnohých odborných článkoch zameraných na návrh a vytvorenie algoritmu pre detekciu anomálie v sieti. Výsledky sú zväčša použité ako súbor dát pre overenie vytvoreného algoritmu, a teda poskytujú autorom možnosť overiť a demonštrovať funkčnosť algoritmu na reálnych dátach zo siete.

1.1 RIPE Atlas sondy

Infraštruktúra tvoria dva typy sond:

1. Hardvérové – bližšie popísané v podsekcii 1.1.1.
2. Softvérové – bližšie popísané v podsekcii 1.1.2.

1.1.1 Hardvérové sondy

Hardvérové sondy boli pôvodne pri vzniku projektu jediným typom. Sondy sú hardvérové zariadenia, ktoré zhromažďujú a posielajú výsledky meraní do RIPE NCC serveru, kde sú dáta spracované, zlúčené s dátami z ostatných sond a následne zverejnené pre verejnosť na webovej stránke RIPE Atlas projektu[1] [1].

1. `https://atlas.ripe.net/results/maps/network-coverage/`

Hostiteľom sondy sa môže stať ktokoľvek, dobrovoľníkom stačí vyplniť žiadosť o zaslanie sondy na webovej stránke projektu, na základe ktorej im bude bezplatne zaslaná sonda. Sondu si hostiteľ jednoducho integruje do svojej domácej siete – pripojí k svojmu smerovaču a následne zaregistruje na stránkach projektu [4]. Hardvérová sonda je zobrazená na obrázku 1.1.

Obr. 1.1: Hardvérová RIPE Atlas sonda [5]

1.1.2 Softvérové sondy

V roku 2020 boli organizáciou RIPE predstavené softvérové sondy, ktoré poskytujú rovnakú funkcionalitu ako hardvérové sondy, avšak nevyžadujú žiadne špecifické fyzické zariadenie. Hlavnou motiváciou vzniku týchto sond bolo rozšírenie pokrytia a ich nasadenie v oblastiach, kde nie je možné nasadiť fyzické sondy. Ich inštalácia a nasadenie vyžadujú vyššie technické znalosti, slúžia preto hlavne ako doplnkový typ sond a v budúcnosti nie je plánované, aby plne nahradili hardvérové sondy.

Ako už z názvu vyplýva tieto sondy sú v podstate softvérové balíčky, ktoré môžu byť nainštalované na smerovače, servery, virtuálne stroje, atď. Inštalačné balíčky sú v súčasnosti k dispozícií pre linuxové operačné systémy CentOS 7/8, Debian 9/10 a taktiež pre platformu Docker [6].

1.1.3 Merania na sonde

Prostredníctvom sondy je možné vykonávať merania nasledujúcich charakteristík siete [4]:

- Ping – meranie dostupnosti a Round-trip time (RTT) pre danú adresu alebo hostiteľa.
- Traceroute – sledovanie cesty prenosu dát z jedného bodu do druhého.
- Domain Name System (DNS) – meranie času, ktorý je potrebný na vyhľadanie informácií o daných doménach.
- Secure Sockets Layer/Transport Layer Security (SSL/TLS) – meranie času potrebného na vybudovanie bezpečného spojenia pre daných hostiteľov.
- Network Time Protocol (NTP) – synchronizácia času pre daných hostiteľov.
- Hypertext Transfer Protocol (HTTP) – meranie rýchlosti prenosu dát pre danú webovú stránku.

1.2 Aplikačné rozhranie prostredia RIPE Atlas

RIPE Atlas poskytuje prehľadné a dobre zdokumentované aplikačné rozhranie, typu Representational state transfer (REST), pre všetky typy operácií, ktoré umožňuje vykonávanie autentifikácie, správy sond/meraní/kreditového systému výhradne prostredníctvom API volaní. Základná Uniform Resource Locator (URL) adresa pre tieto volania je `https://atlas.ripe.net/api/v2/` [7].

1.3 Anomálie v sieti

Analýza prevádzky sa vykonáva pre mnoho účelov, ako napríklad zistenie anomálie, odhalenie nevyžiadanej prevádzky, odhalenie obsahu škodlivého kódu, atď. Detekcia anomálií je teda proces, ktorým sa zisťujú anomálie na základe príslušných údajov. Anomáliu v sieti, je vo všeobecnosti je možné definovať ako neočakávané správanie a teda niečo, čo je odchýlne od normálneho alebo očakávaného správania. Avšak nato, aby sme mohli identifikovať anomáliu, musíme mať zadefinované čo je normálne správanie a stav danej siete, keďže táto definícia sa môže medzi jednotlivými sieťami značne líšiť [8].

1.3.1 Základné techniky detekcie anomálií

Existuje mnoho prístupov k získaniu sieťových charakteristík, kde základné z nich súvisia s výkonnosťou sieťovej vrstvy. Merajú a analyzujú sa charakteristiky ako oneskorenie, priepustnosť alebo strata paketov. Jedným z najjednoduchších spôsobov je výpočet kĺzavého priemeru (angl. Simple Moving Average (SMA)), ktorý možno použiť napríklad na analýzu toku, resp. na analýzu oneskorenia. V prípade SMA ide o nevážený priemer n čísel v časovom rade, ktorý zvyčajne pracuje s diskrétnymi vzorkami údajov. Pri použití vzoriek údajov možno jednoduchý kĺzavý priemer zapísať ako výraz 1.1.

$$\mathrm{SMA}_k = \frac{\sum_{i=n-k+1}^{n} p_i}{k} = \frac{1}{k} \sum_{i=n-k+1}^{n} p_i, \tag{1.1}$$

kde k predstavuje veľkosť okna, n celkový počet pozorovaných hodnôt, p_i jednu pozorovanú hodnotu.

Existujú aj ďalšie varianty založené na SMA. Sú to kumulatívny priemer, vážený kĺzavý priemer, exponenciálny kĺzavý priemer (angl. Exponential Moving Average (EMA)) a ďalšie. EMA sa používa pri RTT výpočte oneskorenia obojsmernej sieťovej prevádzky programom PING. Na meranie a analýzu siete používa Internet Control Message Protocol (ICMP) a ICMP echo správu. Samotný protokol je definovaný

v Request For Comments (RFC) 0792. Pre rad Y možno hodnotu EMA S_t v ľubovoľnej perióde t vypočítať ako [9]:

$$S_t = \begin{cases} Y_0 & \text{pre } t = 0 \\ \alpha Y_t + (1-\alpha) \cdot S_{t-1} & \text{pre } t > 0 \end{cases}, \tag{1.2}$$

kde koeficient α predstavuje stupeň poklesu váhy a konštantný vyhladzovací faktor medzi 0 a 1.

Bod S_t možno zapísať aj ako vážený súčet bodu Y_t. RTT sa pôvodne odhadoval podľa $RTT = \alpha \cdot$ *starý RTT.* $+ (1-\alpha) \cdot$ *nová vzorka RTT* v TCP. Priepustnosť siete sa môže vypočítať pomocou prijímacieho okna TCP W a RTT súvisiaceho s oneskorením, výraz 1.3.

$$TP \leqslant \frac{W}{RTT} \tag{1.3}$$

Použitie iba kĺzavého priemeru však nie je dostatočné na to, aby odrážalo rôzne aspekty analýzy prevádzky. Nestačí napríklad na meranie úrovne tzv. burstu prevádzkových tokov, kde se používa aj entropia (miera neurčitosti), v informačných technológiách označovaná aj ako Shannon (H) podľa autora Clauda Elwooda Shannona.

Uvedené základné techniky sa bežne používajú pri meraniach pomocou atlasu RIPE. Publikácia [10] sa zaoberá návrhom metódy na detekciu a opísanie sieťových anomálií v RTT časových radoch. Navrhovaná metóda pozostáva z dvoch krokov. V prvom hľadá body zmeny v RTT časových radoch prostredníctvom segmentácie dátových radov, v druhom potom označuje segmenty ako normálne alebo abnormálne na základe analýzy ich štatistickej distribúcie. V prípade nájdenia abnormálneho segmentu vykoná analýzu vzťahov medzi spojmi s odlišnými stavmi a lokalizáciu entity, ktoré sú pravdepodobne zodpovedné za anomáliu. Výsledky z experimentov uskutočnených ako na simulovaných, tak aj reálnych RIPE Atlas dátach ukazujú, že táto metóda je presnejšia a efektívnejšia ako existujúce riešenia.

1.4 Systém Disco

„Disco[2] : Rýchly, dobrý a lacný detektor výpadkov“ [11] je existujúci systém, ktorý používa existujúce dlhodobé spojenia využívajúce protokol Transmission Control Protocol (TCP), na identifikáciu výpadkov siete. Systém v reálnom čase analyzuje zmeny stavu RIPE Atlas infraštruktúry a na základe nich dokáže takmer okamžite detegovať výpadok siete.

Výhodou systému je možnosť monitorovania širokého rozsahu infraštruktúry na internete, ktorú predtým nebolo možné sledovať, pretože bola napríklad skrytá za prekladačom sieťových adries (angl. Network Address Translation). Monitorovanie dokáže vykonávať v takmer reálnom čase (rýchlo), s presnosťou 95 % (dobre) a bez pridávania nových paketov (lacno) [11].

Nevýhodou systému je jeho špecifické zameranie na detekciu výpadku siete. V súčasnosti teda nemá možnosť rozšíriteľnosti na detekciu iných anomálií.

1.5 Platforma RIPE Alerts

RIPE Alerts[3] je monolitická platforma vyvíjaná piatimi študentmi z univerzity Hogeschool Utrecht, ktorá je avšak stále v procese vývoja.

Platforma disponuje používatešlkým rozhraním, ktoré umožňuje zvoliť si aký autonómny systém (angl. Autonomous System) chce používateľ monitorovať a následne, pomocou rôznych médií (web server, e-mail) dostáva upozornenia na anomálie, ktoré systém detegoval z výsledkov meraní na RIPE Atlas infraštruktúre [12]. Platforma taktiež poskytuje rozhranie pre implementáciu vlastných funkcií pre detekciu anomálií. V súčasnosti je totižto implementovaný len jeden ukážkový detektor *anchor down*.

2. `https://github.com/InternetHealthReport/disco`
3. `https://github.com/Wolframfriele/ripe-alerts`

2 Vlastné sondy

Pre potreby práce sú k dispozícií aj dve vlastné sondy, jedna hardvérová, a zároveň jedna softvérová.

Hardvérová sonda bola nasadená už v roku 2015, školiteľom tejto práce. Naopak, softvérová sonda bola nasadená autorom tejto práce v Novembri 2022.

Jednou z hlavných motivácií pre nasadenie vlastnej sondy bol fakt, že v projekte RIPE Atlas funguje kreditový systém. Každé vytvorenie merania, ako aj jeho udržiavanie, denne stojí definovaný počet kreditov, ktorý RIPE NCC vypočíta pri vytvorení merania. Bez získaných kreditov by teda nebolo možné vytvorenie žiadnych vlastných meraní.

Spôsobov ako získať kredity existuje viacero, no jedným z najefektívnejších je práve hosťovanie vlastnej sondy.

2.1 Nasadenie softvérovej sondy

Softvérová sonda bola pre potreby tejto práce nasadená na infraštruktúre združenia Czech Education and Scientific NETwork (CESNET) v prostredí MetaCentrum[1].

V prostredí bola vytvorená virtuálna inštancia využívajúca linuxovú distribúciu CentOS 8, ktorá ma pridelenú verejnú IPv4 adresu 147.251.115.116 a nachádza sa v už spomínanej sieti s číslom autonómneho systému (angl. Autonomus System Number) 2852.

Na tejto inštancii následne prebehla inštalácia a spustenie softvérovej sondy, ktorá tam odvtedy beží. Sonda bola taktiež zaregistrovaná na webových stránkach projektu RIPE Atlas, kde jej bolo pridelené identifikačné číslo (Id) 1004950. Viac informácií o danej sonde je dostupných na webe: `https://atlas.ripe.net/probes/1004950/#tab-general`.

1. `https://metavo.metacentrum.cz/cs/`

3 Motivácia a požiadavky pre vlastnú aplikáciu

Dlhodobým cieľom je vytvorenie aplikácie, ktorá umožní distribuované meranie sieťovej prevádzky a detekciu anomálií v sieti. Zdrojom dát sieťovej prevádzky budú RIPE Atlas sondy, popísané v sekcii 1.1.

V rámci tejto práce je cieľom implementácia základu spomínanej aplikácie, ktorá bude slúžiť ako aplikačný rámec (angl. framework) pre ďalší vývoj a zlepšovanie.

Zámerom je sledovať globálne chovanie siete, k čomu ale nestačí sledovanie jednej zvolenej linky do sieťového uzlu. Môže nastať situácia, že na danej linke správca omylom odpojí nejaký sieťový prvok a daná linka sa stane neaktívnou. Takáto situácia však nedáva komplexnú informáciu o stave celej siete, len na základe jednej informácie by bolo možné mylne usúdiť, že výpadok zasiahol celú sieť.

Pre komplexné vyhodnotenie situácie je potrebné sledovanie viacerých liniek vedúcich do sieťového uzlu. S narastajúcim počtom meraní na jednotlivých linkách bude rásť aj schopnosť presnejšieho vyhodnotenia a určenie, či daná udalosť bola anomálnou, alebo nie.

Pre tento účel bude potrebné vytvorenie špecializovaných sieťových agentov, kde každý agent bude sledovať a vyhodnocovať sieťovú prevádzku a detegovať prípadné anomálie na svojej linke – jeden agent bude spracovávať výsledky práve z jednej sondy. Zaznamenaná sieťová prevádzka a výsledky analýzy budú jednotlivými agentmi následne odoslané a uložené do centrálnej databázy, kde budú tieto dáta prístupné pre ďalšie operácie. Súčasťou aplikácie bude teda taktiež jeden centrálny server, ktorý bude používateľovi sprostredkovávať komunikáciu s jednotlivými agentmi.

Motiváciou pre architektúru využívajúcu agentov je, že takéto riešenie dáva v budúcnosti možnosť integrovať agentov priamo do RIPE Atlas sond. Ďalšou výhodou tohto riešenia je distribúcia výpočtovej záťaže serveru, ktorá bude rozložená medzi jednotlivých agentov. V budúcnosti sa počíta aj s nasadením federatívneho učenia (angl. federated learning). Každý agent si bude, učením sa o svojej časti siete, lokálne aktualizovať svoj model a novo získané informácie zdielať s centrálnym modelom na serveri. Na základe aktualizácií získaných od jednotlivých agentov, bude server následne vedieť upravovať a vylepšovať svoj model.

V súčasnosti takéto riešenie nie je k dispozícií, existujúce platformy využívajúce RIPE Atlas, vrátane oboch popísaných v kapitole 1, sú založené na monolitickej architektúre a teda nie sú vhodné pre tento účel.

Nasledujúce sekcie podrobnejšie špecifikujú funkčné a nefunkčné požiadavky na jednotlivé komponenty aplikácie.

3.1 Funkčné požiadavky na server

- Pridanie nového agenta pomocou IP adresy na ktorej je agent nasadený.
- Odstránenie agenta.
- Vytvorenie nového merania, vrátane možnosti špecifikovať:
  - Cieľový uzol (IP adresa/ Názov hostiteľa).
  - Typ merania, t.j. akú charakteristiku chcem zmerať.
  - Interval opakovania merania.
  - Výber agentov, ktorí sa na meraní budú podieľať.
- Zastavenie merania.
- Uloženie výsledkov a analýzy merania jednotlivých agentov.
- Zobrazenie výsledkov merania z databázy, vrátane ich vizualizácie.

3.2 Funkčné požiadavky na agenta

- Poskytnutie API rozhrania, ktoré serveru umožní komunikáciu s agentom.
- Vytvorenie nového interného merania, opakujúceho sa v zadanom intervale.
- Odstránenie interného merania.

- Komunikácia s RIPE Atlas sondou, prostredníctvom aplikačných volaní.
- Analýza výsledkov RIPE Atlas merania.
- Odoslanie výsledkov a analýzy merania do centrálnej databázy.
- Možnosť konfigurácie agenta pred spustením (port, identifikátor sondy s ktorou bude agent komunikovať, atď.).

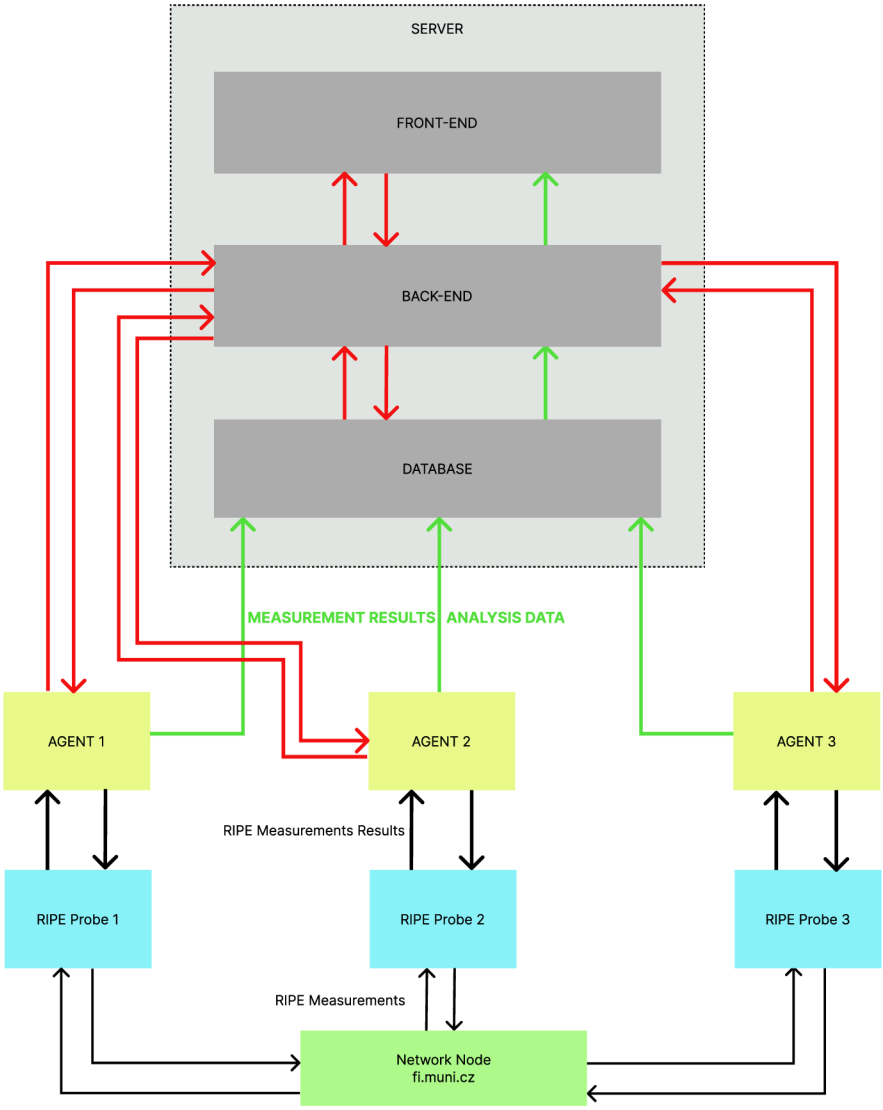
3.3 Nefunkčné požiadavky na aplikáciu

- Podpora pre budúcu možnosť nasadenia frameworku pre federatívneho učenia – flower[1].
- Zabezpečenie agenta, t.j. mechanizmus pre autentifikáciu prístupu k API rozhraniu, ktorý zabráni neoprávnenému využitiu rozhrania.
- Zabezpečenie používateľského a aplikačného rozhrania rozhrania serveru, ktoré zabráni neoprávnenému prístupu.

1. `https://flower.dev`

4 Návrh aplikácie

Celkový prehľad navrhnutej štruktúry aplikácie, zohľadňujúci požiadavky definované v kapitole 3, je zobrazený na obrázku 4.1.



Obr. 4.1: Prehľad štruktúry aplikácie

Aplikáciu budú tvoriť dve samostatné komponenty:

- **Agent** – jedna komponenta, avšak bude možné nasadiť ľubovoľné množstvo inštancií. Návrhu agenta sa podrobnejšie venuje kapitola 5.
- **Server** – architektúra bude tvorená troma vrstvami:
  - **Front-end** – bude predstavovať používateľské rozhranie (UI), ktoré vytvorí prezenčnú vrstvu pre jednoduchšiu interakciu používateľa so serverom.
  - **Back-end** – bude sprostredkovávať komunikáciu medzi jednotlivými vrstvami serveru, ako aj komunikáciu s jednotlivými agentmi.
  - **Databáza** – bude slúžiť na ukladanie dát, ako napríklad zoznam agentov, zoznam meraní, atď.

  Podrobnejšiemu popisu návrhu jednotlivých vrstiev komponenty serveru sa venuje kapitola 6.

Ukazovatele zobrazené na obrázku 4.1 predstavujú nasledujúcu komunikáciu:

- Červené ukazovatele – API komunikácia medzi jednotlivými komponentami aplikácie. Podrobnejšie popisy aplikačných rozhraní sa nachádzajú v sekcii 5.4 pre agenta a v podsekcii 6.2.2 pre back-end vrstvu serveru.
- Zelené ukazovatele – tok dát, predstavujúcich výsledky meraní a výsledky analýzy týchto meraní.
- Čierne ukazovatele – ostatná API komunikácia, t.j predovšetkým komunikácia agenta s RIPE Atlas sondou.

5 Návrh agenta

Agent bude vyvinutý ako konzolová aplikácia a nebude disponovať používateľským rozhraním (UI). Na základe nefunkčnej požiadavky, definovanej v sekcii 3.3, ktorá vyžaduje budúcu možnosť nasadenia aplikačného rámca pre federatívne učenie, sa ako vhodná varianta javí implementácia agenta prostredníctvom programovacieho jazyka Python[1].

Nasledujúce sekcie podrobnejšie popisujú návrh jednotlivých dielčích častí komponenty agenta.

5.1 Konfigurácia

Pre počiatočnú konfiguráciu pred spustením bude slúžiť konfiguračný súbor *.env* , ktorý štandardne slúži na nastavenie premenných prostredia. Pre komponentu agenta umožní nastavenie nasledujúcich premenných:

- AGENT_PORT – číslo portu na ktorom bude agent nasadený.
- AGENT_PASSWORD – heslo pre prístup k aplikačnému rozhraniu agenta.
- AGENT_SECRET_KEY – náhodný 256-bitový tajný kľúč, ktorý bude slúžiť na podpisovanie JSON Web Token (JWT). Bližšiemu popisu štandardu JWT, ako aj zabezpečeniu API rozhrania sa venuje sekcia 7.3. Tajný kľúč môže byť vygenerovaný prostredníctvom príkazového riadku, pomocou príkazu *openssl rand -hex 32*.
- RIPE_PROBE_ID – identifikátor (id) RIPE sondy s ktorou bude agent komunikovať.
- RIPE_API_KEY – API kľuč, ktorý bude slúžiť pre autentifikáciu používateľa pri používaní RIPE Atlas API. Agent bude okrem čítania výsledkov meraní taktiež vytvárať nové merania, je preto potrebné aby mal daný API kľúč správne nastavené práva. Kľúč

1. `https://www.python.org`

je možné vygenerovať v používateľskom profile na webe projektu RIPE Atlas[2].

- MONGO_URL – URL adresa inštancie databázy, do ktorej bude agent zasielať získané výsledky.
- MONGO_USERNAME – používateľské meno pre prístup do inštancie databázy.
- MONGO_PASSWORD – heslo pre prístup do inštancie databázy.

Príklad nakonfigurovaného *.env* súboru je zobrazený v ukážke 5.1.

Ukážka 5.1: Konfiguračný súbor pre agenta

```
AGENT_PORT=8001
AGENT_PASSWORD=password
AGENT_SECRET_KEY=4742217c367c211b2...

RIPE_PROBE_ID=1004950
RIPE_API_KEY=00112233-4455-6677-88...

MONGO_URL=147.251.105.100:27017
MONGO_USERNAME=mongouser
MONGO_PASSWORD=mongopassword
```

5.2 Interná štruktúra

Základným prvkom aplikácie agenta bude jedna inštancia plánovača (angl. scheduler), ktorá bude spravovať a udržiavať zoznam úloh a zároveň zabezpečovať ich vykonávanie v zadaných intervaloch.

Úlohy budú všeobecne reprezentované základnou abstraktnou triedou *MeasurementTask*, ktorá bude, okrem iného, definovať dve abstraktné metódy:

- *measurement_function* – bude definovať kód, ktorý sa má pri zavolaní danej úlohy vykonať a vráti výsledok, vo formáte slovníku (angl. dictionary).

2. https://atlas.ripe.net/docs/howtos/keys.html

- *get_ripe_measurement_type* – bude definovať typ RIPE Atlas merania, ktoré má úloha vytvoriť.

Pre vytvorenie ľubovoľného nového typu úlohy, bude následne stačiť vytvoriť novú triedu, ktorá bude dediť z tejto základnej triedy a implementovať popísané abstraktné metódy.

5.3 Odosielanie výsledkov

V zadanom intervale agent vykoná funkciu, definovanú v metóde *measurement_function* a jej výsledky, vo forme JavaScript Object Notation (JSON) objektu, odošle priamo do databázy, kde budú tieto dáta prístupné pre ďalšie operácie. Základný štruktúra odoslaného objektu je zobrazený v ukážke 5.2.

Ukážka 5.2: Štruktúra objektu reprezentujúceho výsledok úlohy

```
{
  "measurement_task_id": uuid,
  "timestamp": date,
  "results": {}
}
```

Výsledky úlohy a jej analýzy budú odoslané pomocou samostatného spojenia agenta s databázou. Pre získanie výsledkov úlohy teda nebude k dispozícií samostatný koncový bod aplikačného rozhrania.

5.4 Aplikačné rozhranie

Aplikačné rozhranie agenta bude poskytovať prostriedky pre komunikáciu s RIPE Atlas sondou v mene používateľa, ktorému patrí API kľúč zadaný v konfigurácií, preto bude potrebné jeho zabezpečenie. Rozhranie nebude voľne prístupné a pre prístup k nemu bude potrebná autentifikácia.

Prvotná autentifikácia bude prebiehať v momente prvej komunikácie serveru s agentom. K tomuto účelu bude slúžiť koncový bod */auth*, ktorý po úspešnom overení prístupového hesla vráti token.

Na ďalšiu autentifikáciu bude následne slúžiť získaný token, ktorý bude, ako súčasť hlavičky API volania, vyžadovaný pre prístup ku všetkým ostatným koncovým bodom aplikačného rozhrania.

Celkovo bude aplikačné rozhranie agenta poskytovať päť koncových bodov:

- **POST /auth** – bude slúžiť k autentifikácií, po úspešnom overení vráti token.
- **POST /tasks** – bude slúžiť na vytvorenie novej úlohy (interného merania), prijíma nasledujúce atribúty:
  - target: *string* – cieľová IP adresa/ názov hostiteľa
  - description: *string* – vlastný popis zadaný používateľom
  - interval: *int* – interval opakovania
  - measurement_type: *string* – typ merania
  - detection_task_id: *string* – identifikátor hlavnej detekčnej úlohy, t.j úlohy, ktorá bude združovať merania na jednotlivých agentoch, jej identifikátor bude uložený v databáze

  Na základe zadaných atribútov metóda zároveň, na sonde s ktorou komunikuje, vytvorí, pomocou API volania, nové RIPE Atlas meranie. Po úspešnom vykonaní vráti identifikátor novo vytvorenej úlohy.
- **DELETE /tasks/{id}** – bude slúžiť na odstránenie existujúcej úlohy podľa zadaného identifikátora. Metóda zároveň zabezpečí zastavenie priradeného RIPE Atlas merania.
- **GET /verify** – bude slúžiť na jednoduché overenie existencie a dostupnosti agenta.
- **GET /measurementTypes** – bude slúžiť na získanie zoznamu dostupných typov meraní.

6 Návrh serveru

Kapitola podrobnejšie popisuje návrh komponenty serveru, pozostávajúcej z troch vrstiev, konkrétne databázovej vrstvy, popísanej v sekcii 6.1, back-end vrstvy, sekcia 6.2 a front-end vrstvy, sekcia 6.3.

6.1 Databáza

Databáza bude, pre potreby aplikácie, slúžiť na ukladanie dát reprezentujúcich nasledujúce aspekty:

- Zoznam agentov.
- Zoznam detekčných úloh (meraní).
- Výsledky detekčných úloh.

Formát pre ukladanie zoznamu agentov a detekčných úloh, môže byť jasne definovaný, keďže všetky inštancie budú mať rovnaké atribúty.

Na druhú stranu, rozličné inštancie detekčných úloh budú prezentovať rôzny typ merania a teda môžu merať rozličné vlastnosti siete. Ako už bolo popísané v sekcii 5.3, výsledky sa budú odosielať vo formáte JSON, ale presná štruktúra atribútu *results*, ktorý bude tvorený samostatným objektom, nie je pevne daná a je závislá od typu merania.

Pre popísané dôvody sa ako najvhodnejšia varianta javí použiť databázu typu NoSQL namiesto tradičnej relačnej databázy. Konkrétnou variantou databázy by v tomto prípade bola dokumentová databáza MongoDB[1], ktorá nemá pevne danú štruktúru jednotlivých dokumentov, čo implikuje, že štruktúra atribútu *results* nemusí byť naprieč jednotlivými výskytmi zhodná.

1. `https://www.mongodb.com`

Celkový prehľad jednotlivých prvkov databázy, kolekcií, dokumentov a ich atribútov je zobrazený na obrázku 6.1.

Obr. 6.1: Prehľad prvkov databázy

6.2 Back-end

Back-end bude tvoriť prostrednú vrstvu serveru a prostredníctvom aplikačného rozhrania bude sprostredkovávať ako komunikáciu medzi front-endom a databázou, tak aj komunikáciu s agentmi.

6.2.1 Konfigurácia

Pre základnú konfiguráciu vrstvy bude, rovnako ako pri agentovi, slúžiť konfiguračný súbor *.env* , ktorý v tomto prípade umožní nastavenie nasledujúcich premenných:

- USERNAME – používateľské meno, ktoré bude nastavené pre prihlásenie do používateľského rozhrania serveru.

- PASSWORD – heslo, ktoré bude nastavené pre prihlásenie do používateľského rozhrania serveru.
- SECRET_KEY – náhodný 256-bitový tajný kľúč, ktorý bude slúžiť pre podpisovanie JWT.
- MONGO_URL – URL adresa inštancie databázy.
- MONGO_USERNAME – používateľské meno serveru pre prístup do inštancie databázy.
- MONGO_SERVER_PASSWORD – heslo komponenty serveru pre prístup do inštancie databázy, heslo bude nastavené a použité databázovou aj back-end vrstvou.
- MONGO_AGENT_PASSWORD – heslo, ktoré bude nastavené pre prístup komponenty agenta do inštancie databázy, následne bude vyžadované v konfigurácií agenta.
- MONGO_INITDB_ROOT_USERNAME – administrátorské používateľské meno, ktoré bude nastavené pre prístup do inštancie databázy.
- MONGO_INITDB_ROOT_PASSWORD – administrátorské heslo, ktoré bude nastavené pre prístup do inštancie databázy.
- VITE_FRONTEND_PORT – číslo portu na ktorom bude nasadená front-end vrstva.
- VITE_BACKEND_PORT – číslo portu na ktorom bude nasadená back-end vrstva.
- VITE_BACKEND_IP – IP adresa na ktorej bude nasadená back-end vrstva.

Príklad nakonfigurovaného *.env* súboru pre server je zobrazený v ukážke 6.1.

Ukážka 6.1: Konfiguračný súbor pre server

```
USERNAME=username
PASSWORD=password
SECRET_KEY=4742217c367c211b2c6e1...

MONGO_URL=localhost:27017
MONGO_USERNAME=server

MONGO_SERVER_PASSWORD=serverpassword
MONGO_AGENT_PASSWORD=agentpassword

MONGO_INITDB_ROOT_USERNAME=root
MONGO_INITDB_ROOT_PASSWORD=toor

VITE_FRONTEND_PORT=3000
VITE_BACKEND_PORT=8000
VITE_BACKEND_IP=localhost
```

6.2.2 Aplikačné rozhranie

Autentifikácia bude prebiehať obdobným spôsobom ako autentifikácia pre prístup k aplikačnému rozhraniu agenta, popísaná v sekcii 5.4. Koncový bod */login* vráti token ktorý bude následne vyžadovaný pre prístup ku všetkým ostatným koncovým bodom.

Aplikačné rozhranie budú tvoriť nasledujúce koncové body:

- **POST /login** – bude slúžiť pre autentifikáciu používateľa, po úspešnom overení používateľského mena a hesla vráti token.
- **GET /agents** – bude slúžiť na získanie zoznamu dostupných agentov.
- **POST /agents** – bude slúžiť na vytvorenie nového agenta, prijíma atribúty:
  - name: *string* – pomenovanie agenta zadané používateľom, pre jeho jednoduchšiu identifikáciu
  - ip: *string* – IP adresa alebo názov hostiteľa (hostname)

  - port: *int* – číslo portu

- **DELETE /agents{id}** – bude slúžiť na odstránenie existujúceho agenta podľa zadaného identifikátora.

- **GET /detectionTasks** – bude slúžiť na získanie zoznamu detekčných úloh.

- **GET /detectionTasks/{id}** – bude slúžiť na získanie detekčnej úlohy so zadaným identifikátorom.

- **GET /detectionTasks/{id}/results/{agent_task_id}/{offset}/{limit}** – bude slúžiť na získanie výsledkov detekčnej úlohy so zadaným identifikátorom, od konkrétneho agenta (špecifikovaného v *agent_task_id*). Koncový bod vráti maximálne *limit* výsledkov zoradených podľa dátumu, začínajúc výsledkom na pozícií danej atribútom *offset*. Napríklad pre získanie pätnástich najnovších výsledkov bude možné využiť kombináciu atribútov *offset = 0* a *limit = 15*.

- **POST /detectionTasks** – bude slúžiť na vytvorenie novej detekčnej úlohy, prijíma atribúty:

  - target: *string* – cieľová IP adresa/ názov hostiteľa
  - description: *string* – popis zadaný používateľom
  - interval: *int* – interval opakovania
  - measurement_type: *string* – typ merania
  - participating_agents: *list* – zoznam agentov, ktorí sa budú podieľať na meraniach

- **DELETE /detectionTasks/{id}** – bude slúžiť na odstránenie existujúcej detekčnej úlohy podľa zadaného identifikátora.

- **PUT /detectionTasks/{id}/stop/{delete_when_stopping}** – bude slúžiť na zastavenie detekčnej úlohy podľa zadaného identifikátora a na základe atribútu *delete_when_stopping* aj jej prípadne vymazanie.

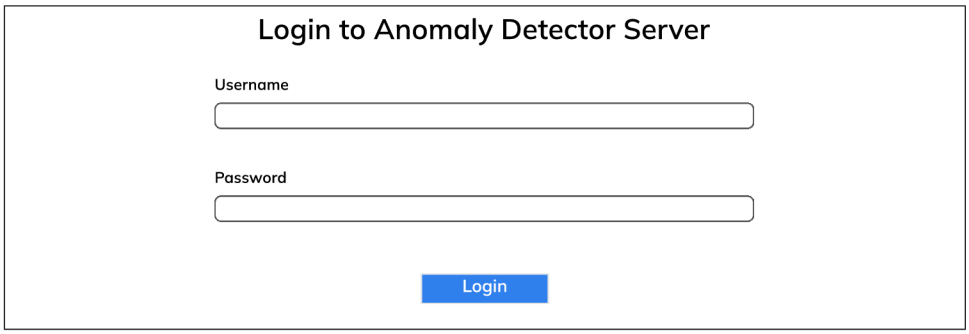
6.3 Front-end

Front-end vrstvu bude tvoriť používateľské rozhranie aplikácie, ktoré používateľovi umožní správu celej aplikácie. Jeho návrh zohľadňuje funkčné požiadavky pre server definované v sekcii 3.1.

Všetky obrázky návrhu jednotlivých stránok a komponentov používateľského rozhrania v nasledujúcich podsekciách, boli vytvorené pomocou online nástroja Figma[2].

6.3.1 Prihlasovacia stránka

Úvodnou stránkou aplikácie bude jednoduchá prihlasovacia stránka, obrázok 6.2, kde používateľ zadá svoje prihlasovacie údaje. Po kliknutí na tlačidlo *Login*, prebehne pomocou API volania autentifikácia používateľa. V prípade úspešnej autentifikácie bude následne používateľ presmerovaný na ďalšiu stránku zobrazujúcu detekčné úlohy, ktorú opisuje nasledujúca podsekcia 6.3.2. V prípade neúspešnej autentifikácie nebude používateľovi umožnený prístup na žiadnu ďalšiu stránku používateľského rozhrania.



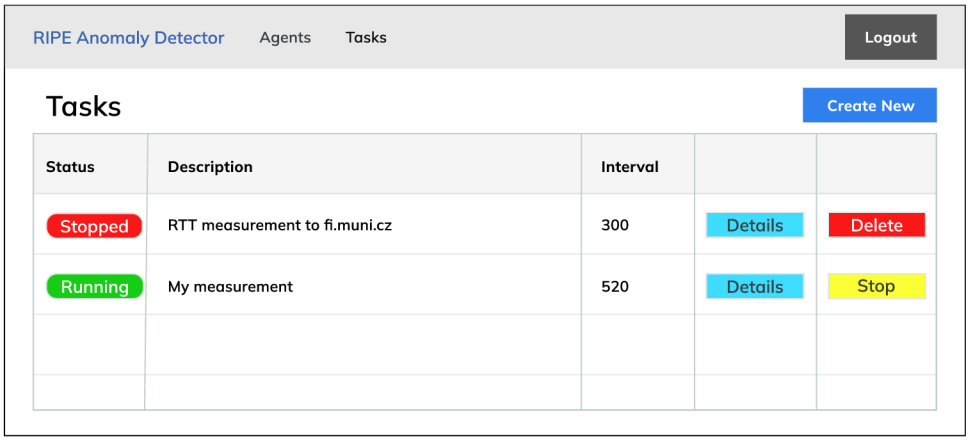
Obr. 6.2: Návrh prihlasovacej stránky

6.3.2 Stránka – detekčné úlohy

Stránka bude slúžiť pre zobrazenie a správu detekčných úloh. Základným prvkom bude tabuľka zobrazujúca zoznam detekčných úloh,

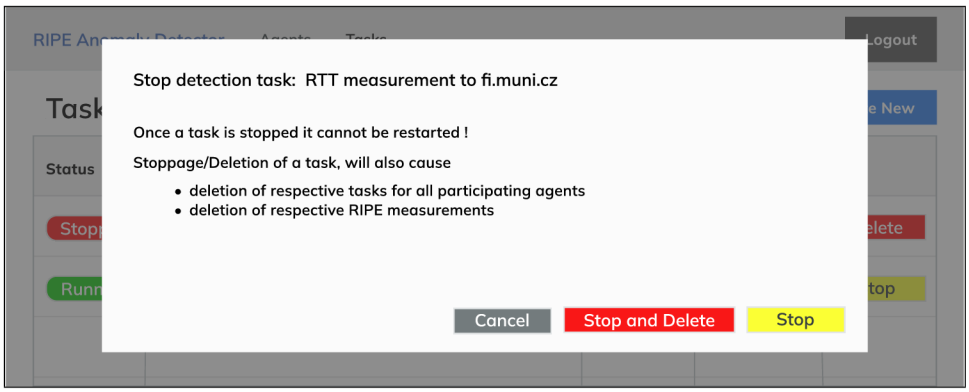
2. `https://www.figma.com`

obrázok 6.3. Tabuľka bude pozostávať z piatich stĺpcov, kde prvé tri budú reprezentovať atribúty jednotlivých meraní, konkrétne: status, popis (angl. description), interval. Zvyšné dva stĺpce budú tvoriť tlačidlá Detaily (angl. Details) a Zastaviť/Vymazať (angl. Stop/Delete) v závislosti od stavu merania.



Obr. 6.3: Návrh stránky zobrazujúcej zoznam detekčných úloh

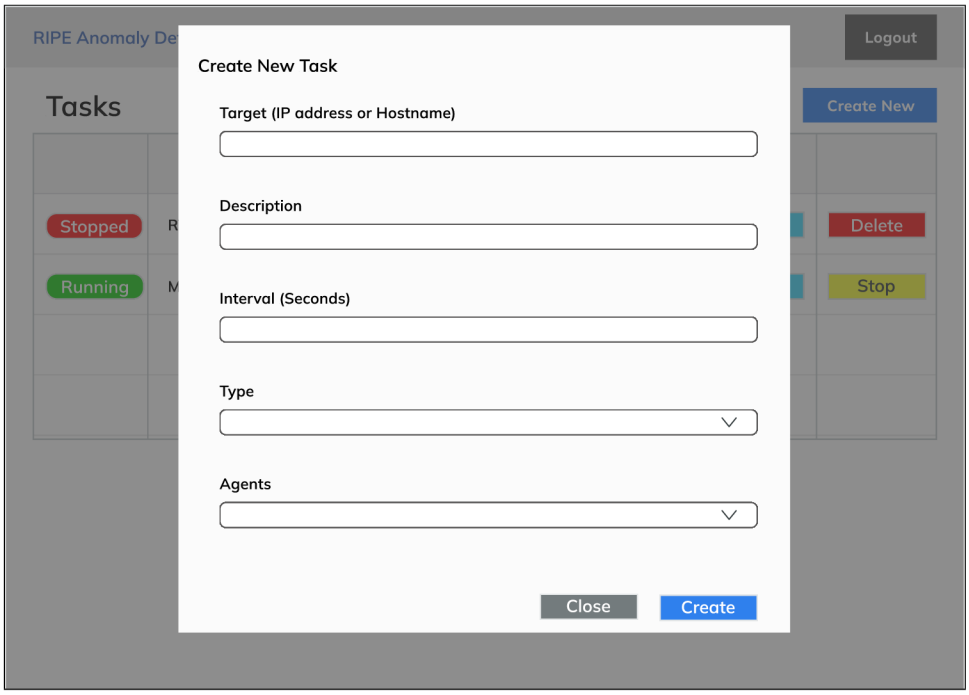
Po stlačení tlačidla Stop bude zobrazené dialógové okno, obrázok 6.4, v ktorom bude mať používateľ možnosť zvoliť zastavenie, alebo zastavenie a vymazanie príslušnej detekčnej úlohy.



Obr. 6.4: Návrh stránky zobrazujúcej dialóg pre zmazanie úlohy

Po stlačení tlačidla Details bude používateľ presmerovaný na stránku zobrazujúcu detail príslušnej detekčnej úlohy, ktorá je popísaná v samostatnej podsekcii 6.3.3.

Navyše, v pravom hornom rohu stránky sa bude nachádzať tlačidlo pre vytvorenie novej úlohy – Vytvoriť (angl. Create), po ktorého stlačení bude používateľovi zobrazené dialógové okno, obrázok 6.5. Okno bude obsahovať formulár s poliami: cieľ (angl. target), popis (angl. description), interval, typ (angl. type) a pole pre zvolenie agentov, ktorí sa budú podieľať na meraní – agenti (angl. agents).

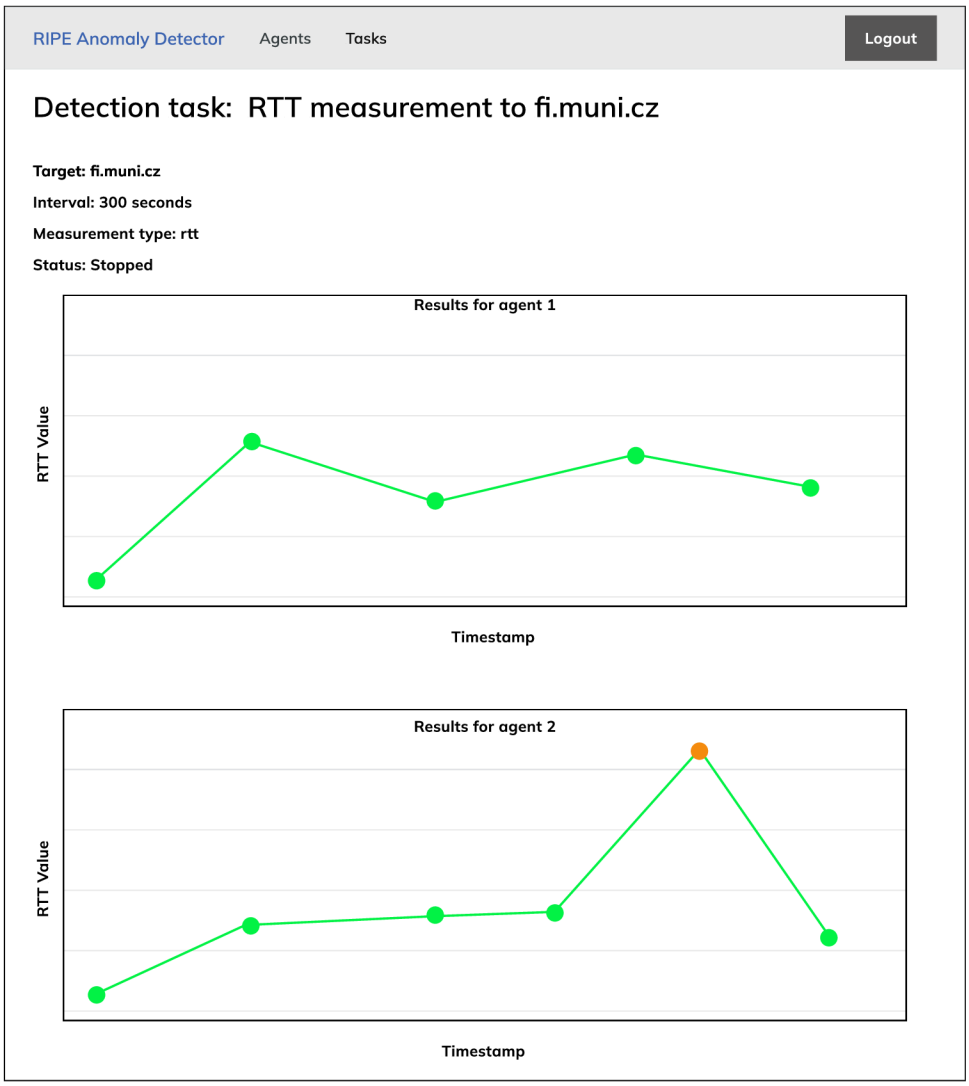


Obr. 6.5: Návrh stránky zobrazujúcej dialóg pre vytvorenie novej detekčnej úlohy

6.3.3 Stránka – detail detekčnej úlohy

Stránka bude slúžiť pre zobrazenie podrobnejších informácií o detekčnej úlohe a taktiež pre zobrazenie jej výsledkov, obrázok 6.6. V hornej časti stránky budú zobrazené atribúty danej úlohy, konkrétne: cieľ

(angl. target), popis (angl. description), interval, typ merania (angl. measurement type) a status. Zvyšnú časť stránky budú tvoriť diagramy, vizuálne reprezentujúce výsledky danej úlohy. Počet diagramov bude závislý od počtu agentov podieľajúcich sa na meraní, každému agentovi bude prislúchať jeden.

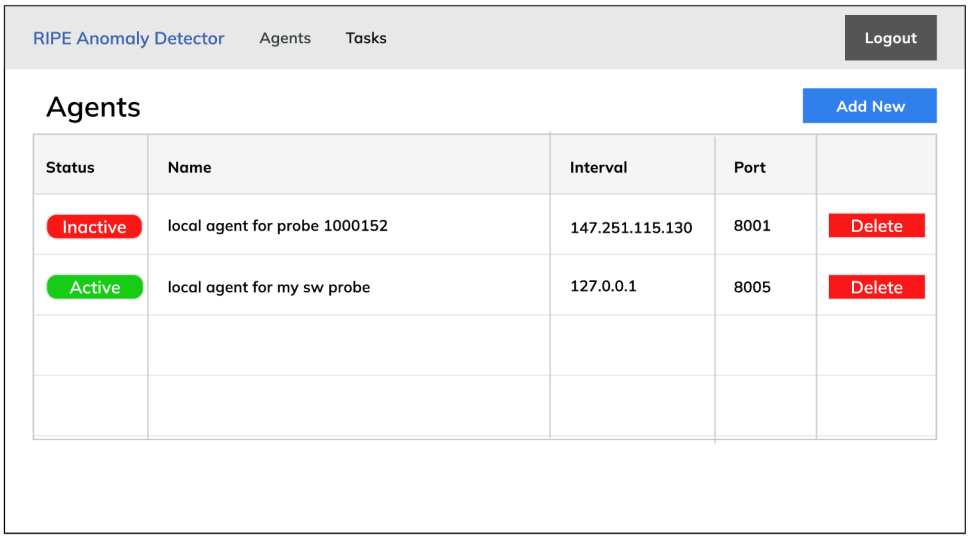


Obr. 6.6: Návrh stránky zobrazujúcej detail detekčnej úlohy

Typ a presný vzhľad diagramu bude závisieť na type detekčnej úlohy, ukážkový diagram na obrázku 6.6 predstavuje typ *rtt*, v ktorom zelené body budú predstavovať normálnu hodnotu, oranžové body budú predstavovať hodnotu, ktorú daný agent vyhodnotil ako anomálnu a prípadné červené body, s nadobudnutou hodnotou *-1*, budú signalizovať chybu merania.

6.3.4 Stránka – agenti

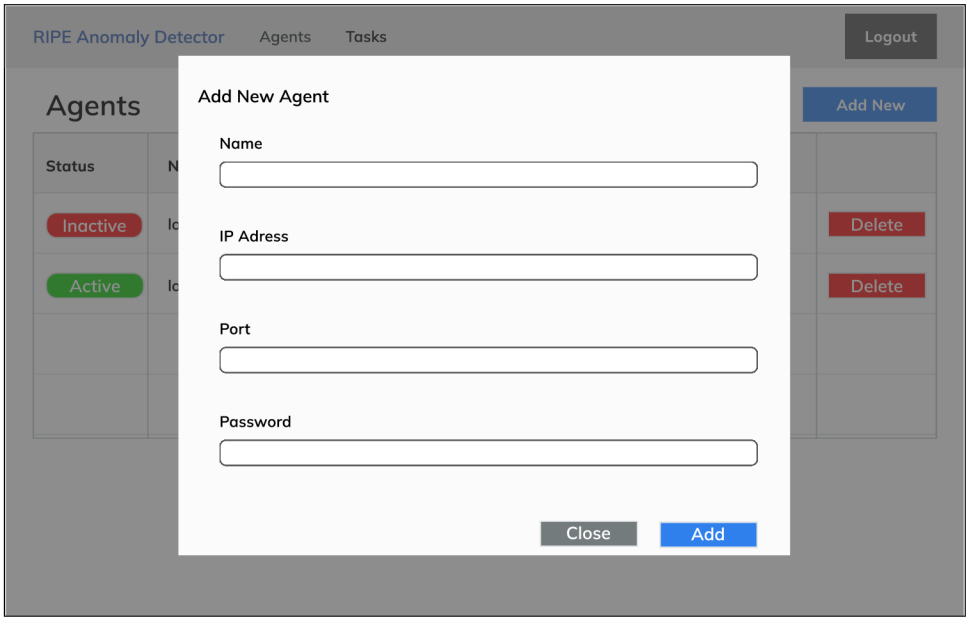
Stránka bude slúžiť pre správu agentov, jej základným prvkom bude tabuľka zobrazujúca zoznam agentov, obrázok 6.7, ktorá bude pozostávať z piatich stĺpcov. Štyri z nich budú reprezentovať atribúty jednotlivých agentov, konkrétne: status, meno (angl. name), IP adresa (angl. IP address) a port. Posledný stĺpec bude tvoriť tlačidlo Vymazať (angl. Delete), po ktorého stlačení bude príslušný agent odstránený zo zoznamu.



Obr. 6.7: Návrh stránky zobrazujúcej zoznam agentov

V pravom hornom rohu stránky sa bude nachádzať tlačidlo pre pridanie nového agenta – Pridať (angl. Add), po ktorého stlačení bude používateľovi zobrazené dialógové okno, obrázok 6.8, obsahujúce for-

mulár s poliami potrebnými pre pridanie nového agenta: meno (angl. name), IP adresa (angl. IP address), port a heslo (angl. password).



Obr. 6.8: Návrh stránky zobrazujúcej dialóg pre pridanie nového agenta

7 Implementácia agenta

Kapitola podrobnejšie popisuje jednotlivé časti agenta a technológie použité pri ich implementácií.

Ako už bolo spomínané pri návrhu agenta v úvode kapitoly 5, pre implementáciu bol použitý programovací jazyk Python. Implementácia je členená do troch balíkov (angl. packages):

- *agent* – okrem modulu *main* obsahuje aj ďalšie základné moduly, vytvárajúce kostru implementácie agenta.
- *measurement_tasks* – obsahuje implementácie tried reprezentujúcich jednotlivé úlohy.
- *ripe_communication* – obsahuje moduly zabezpečujúce komunikáciu s RIPE sondou.

Okrem spomenutých balíkov hlavný adresár implementácie obsahuje taktiež, súbory potrebné pre spustenie pomocou aplikácie Docker: *docker-compose.yml* a *Dockerfile*, súbor *requirements.txt* obsahujúci informácie o všetkých externých knižniciach a moduloch potrebných pre spustenie aplikácie a súbor *README.md*, obsahujúci návod na spustenie aplikácie.

7.1 Plánovač úloh

Ako plánovač úloh bol pri implementácií agenta zvolený framework Rocketry[1], čo je moderný framework pre plánovanie v Python aplikáciách. Nevyžaduje zložité počiatočné nastavenie a je možné ho použiť aj pre väčšie aplikácie. Framework sám o sebe poskytuje množstvo užitočných funkcií, no zároveň je navrhnutý tak, aby bol ľahko rozšíriteľný a poskytoval vývojárom možnosť prispôsobiť si ho svojím potrebám [13].

Framework bol zvolený predovšetkým pre jeho schopnosť modifikovať reláciu (angl. session) počas behu programu. Úlohy nemusia byť definované vopred, pri spustení aplikácie ale môžu byť pomerne jednoducho vytvorené, prípadne odstránené, počas behu aplikácie, čo

1. `https://rocketry.readthedocs.io/en/stable/index.html`

v tomto prípade poskytuje možnosť jednoduchého ovládania agenta pomocou API volaní.

7.2 Aplikačné rozhranie

Pre implementáciu aplikačného rozhrania bol zvolený framework FastAPI[2], čo je vysoko výkonný framework pre vytváranie API rozhraní v jazyku Python, postavený nad webovým serverom Starlette[3]. Obsahuje funkcie, ktoré uľahčujú vytváranie webových aplikácií, ako napríklad spracovanie chýb a validácia údajov. FastAPI taktiež umožňuje automatické vytváranie interaktívnej dokumentácie vytvoreného aplikačného rozhrania pomocou frameworku SwaggerUI[4] [14].

Výhodou frameworku je natívna podpora pre paralelizmus. Koncové body je možné definovať ako asynchrónne funkcie s použitím kľúčového slova *async*, čo umožňuje lepšiu škálovateľnosť aplikácie a lepšiu kontrolu prebiehajúcich input/output (I/O) operácií, ktoré v tomto prípade predstavuje hlavne komunikácia agenta s RIPE Atlas sondou.

7.3 Zabezpečenie aplikačného rozhrania

FastAPI ako súčasť svojej funkcionality poskytuje implementáciu nástrojov pre zabezpečenie aplikačného rozhrania pomocou protokolu OAuth 2.0[5].

Koncový bod aplikačného rozhrania */auth* prijíma formulár *OAuth2PasswordRequestForm*[6]. Formulár v tomto prípade obsahuje meno a heslo používateľa, ktorý sa chce autentifikovať.

Po úspešnom overení hesla koncový bod vráti odpoveď vo formáte zobrazenom v ukážke 7.1, obsahujúcu prístupový token, v štandarde JSON Web Token (JWT) a typ tokenu, v tomto prípade *bearer*. Bližšiemu popisu štandardu JWT sa venuje podsekcia 7.3.1.

---

2. `https://fastapi.tiangolo.com`
3. `https://www.starlette.io`
4. `https://github.com/swagger-api/swagger-ui`
5. `https://oauth.net/2/`
6. `https://github.com/tiangolo/fastapi/blob/master/fastapi/security/oauth2.py`

Ukážka 7.1: Odpoveď koncového bodu /auth

```
{
  "access_token": <token>,
  "token_type": "bearer"
}
```

Autentifikácia pre všetky ostatné koncové body prebieha pomocou takzvanej „bearer authentication“[7], tiež nazývanej „autentifikácia pomocou tokenu“. Klient vyžadujúci prístup ku koncovému bodu musí ako súčasť autorizačnej hlavičky daného volania zahrnúť aj získaný token, viď. ukážka 7.2.

Ukážka 7.2: Autorizačná hlavička

```
Authorization: Bearer <token>
```

7.3.1 JSON Web Token

JSON Web Token (JWT) je štandard definovaný v Request For Comments (RFC) 7519[8], ktorý popisuje spôsob bezpečného prenosu informácií medzi dvoma stranami prostredníctvom JSON objektov [15]. JWT je možné použiť aj na overenie identity používateľa a poskytnutie prístupu k rôznym službám a zdrojom v aplikácii alebo webovej službe, čo je prípad využitia v implementácií tejto práce. JWT sa skladá z troch častí, každá z nich je tvorená atribútmi, nazývanými *JWT Claims*. Štandard definuje atribúty, ktoré majú pevne daný názov a použitie, no zároveň umožňuje aj vytvorenie vlastných atribútov s unikátnym názvom. Jednotlivé časti tvoria [15] [16]:

- **Hlavička** (angl. header) – obsahuje metadáta o tokene, ako typ tokenu a algoritmus použitý na jeho podpisovanie.
- **Náklad** (angl. payload) – obsahuje prenášané dáta. Môže zahŕňať informácie o používateľovi, právach používateľa, časovom obmedzení platnosti tokenu a ďalšie.

7. „Bearer“ voľne preložiteľné ako „držiteľ“. Celé slovné spojenie môže byť chápané ako „udeľ prístup držiteľovi daného tokenu.“
8. https://www.rfc-editor.org/rfc/rfc7519#section-3

- **Podpis** (angl. signature) – používa sa na overenie autenticity tokenu a zabezpečenie, že prenášané dáta neboli zmenené po jeho vytvorení. Podpis sa vytvára z kombinácie hlavičky a nákladu tokenu, ktoré sú podpísané algoritmom špecifikovaným v hlavičke s využitím tajného kľúča zadaného pri konfigurácií agenta.

Ukážka jednotlivých častí tokenu, ich vybraných atribútov v nezakódovanej forme je zobrazená na obrázku 7.1.

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "sub": "123e4567-e89b-12d3-a456-426614174000",
  "name": "Martin",
  "iat": 1681392664
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  7235753878214125442A47
) ☐ secret base64 encoded
```

Obr. 7.1: JWT v nezakódovanej forme

Prenos JWT však neprebieha v otvorenej forme, špecifikovanej v predchádzajúcich odstavcoch. Pre prenos sú všetky tri časti tokenu zakódované a spojené. Spolu tvoria jeden JSON objekt v ktorom sú jednotlivé časti oddelené bodkou (.). Token v zakódovanej forme je zobrazený na obrázku 7.2.

Opätovné dekódovanie tokenu je možné len pomocou tajného kľúča.


```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjNlNDU2Ny1lODliLTEyZDMtYTQ
1Ni00MjY2MTQxNzQwMDAiLCJuYW1lIjoiTWFydG
luIiwiaWF0IjoxNjgxMzkyNjY0fQ.
zPjv02GTVE_KKk5Mpn97y9Pe0u_PvRt7K79h4ZB
baX4
```


Obr. 7.2: JWT v zakódovanej forme

7.4 Prostriedky pre nasadenie

Ako už bolo spomínané, v rámci implementácie boli vytvorené súbory, ktoré umožňujú agenta nasadiť na ľubovoľnom operačnom systéme, prostredníctvom aplikácie Docker[9]. Konkrétne sú to súbory *Dockerfile* – slúžiaci na vytvorenie tzv. Docker Image a *docker-compose.yml* – vykonávajúci kompozíciu kontajnera, a zároveň umožňujúci spustenie agenta pomocou jedného príkazu v príkazovom riadku *docker compose up -d*.

Okrem nasadenia prostredníctvom aplikácie Docker, je agenta možné nasadiť aj ako službu (angl. service) prostredníctvom *systemctl* príkazov. Podrobnejší návod na spustenie agenta ako služby sa nachádza v priloženom *README.md* súbore.

9. `https://www.docker.com`

7.5 Implementácia ukážkovej úlohy

V rámci aplikácie bola implementovaná aj ukážková trieda *RttClassifier*, reprezentujúca konkrétnu úlohu, ktorá dedí zo základnej triedy predstavenej v sekcii 5.2.

Trieda predstavuje úlohu, ktorej cieľom je načítanie posledného výsledku RIPE Atlas merania, typu *ping* a následná predikcia toho, či je nameraný výsledok anomáliou. Predikcia je v tomto prípade vykonávaná pomocou naivnej Bayesovej klasifikácie s použitím knižnice scikit-learn[10], konkrétne triedy GaussianNB[11].

Príklad konkrétneho výsledku úlohy je zobrazený v ukážke 7.3.

Ukážka 7.3: JSON reprezentujúci výsledok úlohy

```
{
  "measurement_task_id": "161f2fa6-c4a1-11ed-
     ab02-0028f868ffbb",
  "timestamp": "2023-03-17T10:19:26.855Z,
  "results": {
    "result": 100.6403136667,
    "prediction_success": True,
    "is_anomaly_prediction": True,
    "probability": 0.9112476637851876
  }
}
```

10. https://scikit-learn.org/stable/
11. https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html#sklearn.naive_bayes.GaussianNB

8 Implementácia serveru

Kapitola bližšie popisuje implementáciu jednotlivých vrstiev serveru a technológie použité pri ich implementácií.

Hlavný adresár implementácie aplikácie je členený do dvoch základných podadresárov, ktoré obsahujú implementáciu jednotlivých komponent: *backend* – viac popísaný v sekcii 8.2 a *frontend* – viac popísaný v sekcii 8.3.

Hlavný adresár taktiež zahŕňa súbor *mongo-init.sh* obsahujúci skript pre inicializáciu databázy a súbory *docker-compose.yml* a *README.md*.

8.1 Databáza

Pre implementáciu bola zvolená MongoDB[1], čo je multiplatformová, open-source databáza typu NoSQL.

Na ukladanie dát nepoužíva relačné tabuľky, ale kolekcie (angl. collections), ktoré pozostávajú z dokumentov (angl. documents) vo formáte podobnom formátu JSON, nazývanom Binary JavaScript Object Notation (BSON). Dokumenty sú dynamické, ich štruktúra nie je pevne daná a môže sa v priebehu času meniť [17], zároveň každý dokument v kolekcii môže mať odlišnú štruktúru, čo umožňuje vytváranie dynamickej dátovej schémy.

V implementácií komunikuje s databázou nielen server, prostredníctvom back-end vrstvy, ale za účelom odosielania výsledkov s ňou samostatne komunikujú aj inštancie agenta. Pre komunikáciu, všetci spomenutý klienti využívajú oficiálny ovládač (angl. driver) pre programovací jazyk Python – PyMongo[2], ktorý poskytuje všetky potrebné nástroje pre prácu s databázou a umožňuje zabezpečenú end-to-end komunikáciu medzi databázou a klientom.

8.2 Back-end

Pre implementáciu back-end vrstvy bol, rovnako ako pre implementáciu agenta, použitý programovací jazyk Python.

---

1. `https://www.mongodb.com`
2. `https://www.mongodb.com/docs/drivers/pymongo/`

Samotná implementácia pozostáva z piatich vlastných Python modulov a dvoch ďalších súborov *Dockerfile* a *requirements.txt*. Primárnou úlohou vrstvy je vytvorenie jednotlivých koncových bodov aplikačného rozhrania. Pre ich tvorbu boli využité rovnaké technológie ako pre implementáciu aplikačného rozhrania agenta, popísanú v sekcii 7.2.

Zabezpečenie autentifikácie pre prístup k API je taktiež zhodné s autentifikáciou pre prístup k API agenta popísanou v sekcii 7.3. Jediným zásadným rozdielom je, že pre prvotnú autentifikáciu slúži v tomto prípade koncový bod */login*.

8.3 Front-end

Pre implementáciu front-end vrstvy bol použitý programovací jazyk JavaScript, konkrétne jeho knižnica React. Podrobnejšiemu popisu použitej knižnice je venovaná podsekcia 8.3.1.

Štruktúra *frontend* adresára zodpovedá štandardnej štruktúre React projektu, ktorú, okrem iného, tvorí adresár *src* obsahujúci zdrojové súbory aplikácie, súbor *package.json* obsahujúci metadáta, závislosti a ďalšie informácie o projekte. Adresár obsahuje taktiež súbor *vite.config.js* slúžiaci pre konfiguráciu nástroja Vite, popísaného v podsekcii 8.3.3 a už skôr spomínaný súbor *Dockerfile*. Snímky obrazovky vytvoreného používateľského rozhrania sa nachádzajú v prílohe B tejto práce.

8.3.1 React

React[3] je open-source knižnica programovacieho jazyka JavaScript (JS), ktorá slúži na vytváranie používateľských rozhraní.

Využíva JavaScript Syntax Extension (JSX), čo je rozšírenie jazyka JavaScript, ktoré navyše umožňuje použitie HTML značiek, čo umožňuje jednoduchšiu definíciu vzhľadu vytvoreného používateľského rozhrania [18]. Základným prvkom pre tvorbu používateľského rozhrania sú tzv. komponenty, čo sú opakovane použiteľné elementy, ktoré majú definovaný vzhľad aj funkcionalitu. Spájaním a interakciu jednotlivých komponentov následne vzniká výsledné používateľské rozhranie [19].

3. `https://react.dev`

Pre dizajn jednotlivých komponent používateľského rozhrania bol v implementácií použitý framework Bootstrap, ktorý je viac popísaný v podsekcii 8.3.2.

Značnou výhodou knižnice React oproti čistej JavaScript aplikácií je rýchlosť. React používa koncept virtuálneho Document Object Model (DOM) tzv. VDOM, ktorého stav si drží vo svojej pamäti. Pri každej zmene ktorá sa udeje v komponente je stav reálneho a virtuálneho DOMu porovnaný, čoho následkom je úprava iba zmenenej, a teda pomerne malej, časti reálneho DOMu. Na druhú stranu JavaScript, pri každej zmene upravuje celý reálny DOM, kvôli čomu zaostáva rýchlosťou [20] [21].

8.3.2 Bootstrap

Bootstrap[4] je open-source framework pre rýchlejší a jednoduchší vývoj front-end aplikácií. Poskytuje vopred definované dizajnové šablóny pre rôzne komponenty používateľského rozhrania ako sú napríklad tlačidlá, formuláre, tabuľky a ďalšie [22]. Použitie týchto komponentov umožňuje vývojárom rýchle vytvorenie dobre vyzerajúceho používateľského rozhrania, bez potreby definovania množstva Cascading Style Sheets (CSS) štýlov [23] .

Vytvorené rozhranie je navyše responzívne, čo znamená, že celkové rozvrhnutie jednotlivých komponent bude automaticky prispôsobené zariadeniu na ktorom je zobrazené. Nie je potrebné vytvorenie osobitného dizajnu pre rôzne veľkosti zariadení, responzívnosť rozhrania zaisťuje jeho konzistenciu v prípade zobrazenia napríklad na mobilnom telefóne [24].

8.3.3 Vite

Vite[5] je front-end nástroj, využívaný pri vývoji aplikácií, ktorý umožňuje rýchle načítavanie zdrojových súborov a spustenie vývojového serveru. Plusom nástroja je zlepšenie Developer Experience (DX), t.j. zážitku vývojára, keďže mu umožňuje rýchlejšie a pohodlnejšie prostredie pre vývoj aplikácie [25]. Pre konfiguráciu nástroja slúži už spomínaný súbor *vite.config.js*, ktorý umožňuje jednoduché nastavenie

4. `https://getbootstrap.com`
5. `https://vitejs.dev`

potrebných aspektov aplikácie. Konkrétna podoba súboru použitá v tejto implementácii je zobrazená v ukážke 8.1.

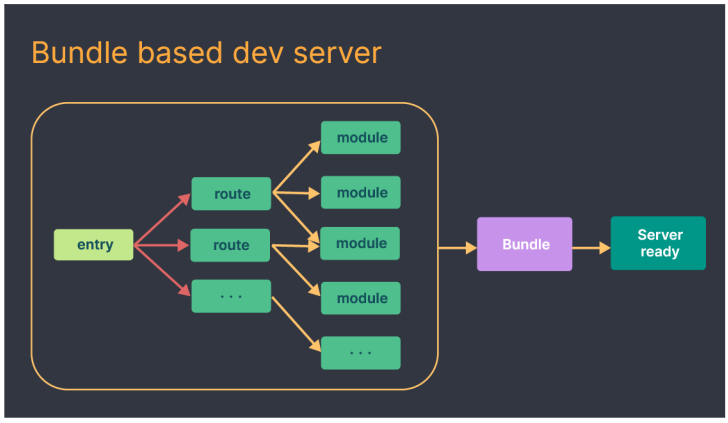
Ukážka 8.1: Súbor vite.config.js

```
export default ({mode}) => {
  const env = {...process.env, ...loadEnv(mode,
     process.cwd())};

  return defineConfig({
    server: {
      host: '0.0.0.0',
      port: parseInt(env.VITE_FRONTEND_PORT),
    },
    plugins: [react()],
  })
}
```

Klasický prístup pre budovanie aplikácie

Klasické nástroje pre vybudovanie aplikácie vytvárajú tzv. bundle, čo je balík obsahujúci všetky komponenty JavaScript kódu spojené do jedného veľkého celku, ktorý je následné upravený a pripravený pre spustenie v prehliadači [26]. Postup je vizualizovaný na obrázku 8.1. Nástroje využívajúce tento prístup sa nazývajú tzv. bundler-based.



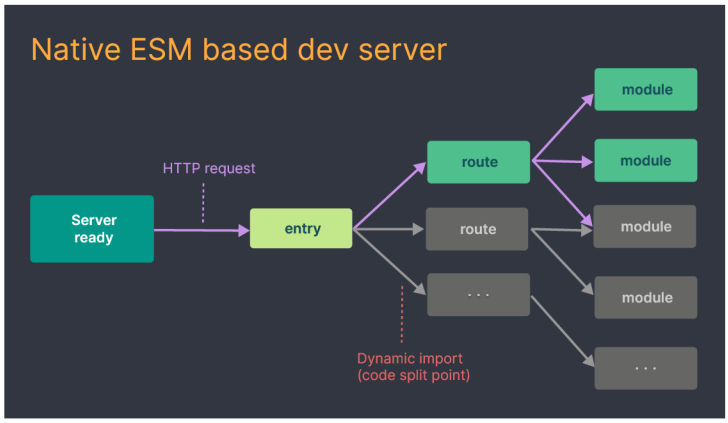
Obr. 8.1: Klasický prístup pre budovanie aplikácie [27]

Problémom popísaného prístupu je, že so zväčšujúcim sa množstvom kódu rapídne rastie aj čas potrebný pre vytvorenie balíka, zároveň v prípade, že je zmenená hoci len malá časť pôvodného kódu, balík sa aj tak musí vytvoriť celý nanovo, čo nie je časovo efektívne.

Prístup Vite

Na druhú stranu Vite, k dosiahnutiu svojej rýchlosti využíva štandardizovaný formát systému modulov ECMAScript Modules (ESM), čo je syntax definujúca formát importov v JS kóde, napríklad *import function from "./library.js";*

Vďaka využitiu ESM modulov a samostatnému načítavaniu jednotlivých komponentov, dokáže Vite časť úlohy bundler-based nástroja prenechať samotnému prehliadaču. Pri zmene kódu tak stačí načítať len zmenenú časť, čo výrazne eliminuje časové oneskorenia [26] [27] . Popísaný prístup nástroja je vizualizovaný na obrázku 8.1.



Obr. 8.2: Prístup nástroja Vite pre budovanie aplikácie [27]

9 Praktické nasadenie a zhodnotenie

Okrem lokálneho nasadenia, bola výsledná implementácia komponenty serveru, pre účely demonštrácie, nasadená aj na virtuálnej inštancii vytvorenej na infraštruktúre združenia CESNET s pridelenou IP adresou 147.251.115.116. Používateľné rozhranie aplikácie je dostupné na porte 3001.

Keďže pre účely nasadenia bola k dispozícií len jedna virtuálna inštancia s verejnou IP adresou, na rovnakej inštancii boli vytvorené aj tri inštancie komponenty agenta, konkrétne na portoch 8004, 8005 a 8006.

Pre umožnenie prístupu boli na virtuálnej inštancii zároveň sprístupnené všetky potrebné porty.

Nasadenie všetkých komponent prebehlo pomocou priloženého návodu popísaného v súbore *README.md* nachádzajúcom sa v adresároch príslušných komponent.

9.1 Overenie funkčnosti

Pre overenie funkčnosti implementácie prebehlo manuálne testovanie na nasadenej infraštruktúre, kde bola overená funkčnosť komunikácie medzi jednotlivými komponentami aplikácie (komunikácie s agentmi, odosielanie výsledkov do databázy, atď.) a taktiež typické prípady použitia aplikácie, špeciálne prípady a pod.

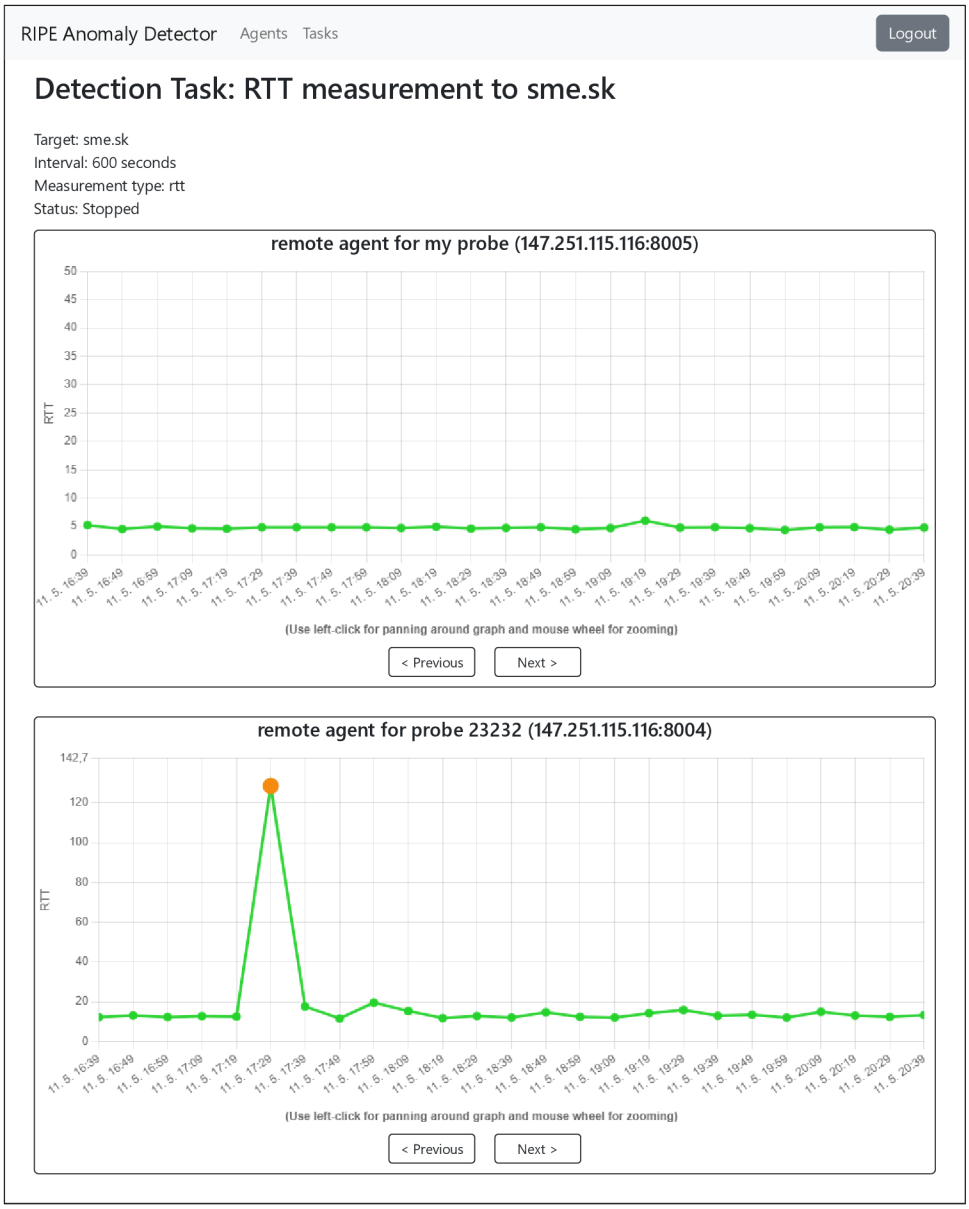
Prostredníctvom používateľského rozhrania aplikácie najprv prebehlo pridanie jednotlivých agentov. Následne prebehlo aj vytvorenie detekčných úloh, zameraných na rôznych hostiteľov, s využitím rôznej kombinácie pridaných agentov. V rámci vytvorenia detekčných úloh bolo skontrolované aj správne vytvorenie samotných RIPE Atlas meraní.

Výsledky z týchto úloh boli následne analyzované a porovnané s výsledkami RIPE meraní, ktoré sú dostupné na webe projektu, pre overenie správnosti preposielaných údajov.

Počas procesu overovania boli nájdené aj rôzne obmedzenia pre vytváranie úloh, ktoré podrobnejšie popisuje sekcia 9.2.

Ukážkový príklad konkrétneho výsledku detekčnej úlohy, ktorý reprezentuje úlohu typu *rtt* pre hostiteľa *sme.sk*, je zobrazený na ob-

rázku 9.1. Výsledok zachycuje aj bod, ktorý bol agentom vyhodnotený ako anomálny, v diagrame je vyznačený odlišnou farbou.



Obr. 9.1: Výsledky detekčnej úlohy

9.2 Obmedzenia pri vytváraní meraní

Počas vytvárania nových úloh bolí nájdené aj rôzne obmedzenia týkajúce sa rozhrania RIPE, s ktorými je potrebné počítať. Okrem spomínaného obmedzenia týkajúceho sa potrebného množstva kreditov, je možné naraziť napríklad na obmedzenie týkajúce sa cieľovej IP adresy pri vytváraní nového RIPE merania. Samotný RIPE Atlas obmedzuje počet meraní na rovnaký cieľ a pri snahe o vytvorenie merania na *google.com* bola, propagáciou od agenta, vrátená chybová odpoveď, ukážka 9.1, ktorá bola následné, vo forme notifikácie zobrazená na používateľskom rozhraní.

Ukážka 9.1: JSON chybová odpoveď

```
{
 "message": "Can not create RIPE measurement: We
     do not allow more than 25 concurrent
    measurements to the same target: google.com."
}
```

9.3 Zhodnotenie

Výsledkom práce je aplikácia na detekciu sieťovej prevádzky a anomálií, vytvárajúca prostredie s viacerými agentmi.

Obe komponenty aplikácie poskytujú priestor pre mnoho ďalších vylepšení, no v konečnom dôsledku, po overení jednotlivých funkcionalít a prípadov použitia, je možné konštatovať, že výsledná aplikácia spĺňa definované požiadavky špecifikované v kapitole 3, ako aj samotné zadanie práce. Implementácia, teda splnila cieľ a vytvorila aplikačný rámec, ktorý bude slúžiť pre ďalší vývoj.

Záver

Predložená diplomová práca sa zaoberala návrhom a vývojom aplikácie slúžiacej na detekciu sieťovej prevádzky a anomálií pomocou sieťových RIPE sond.

Práca postupne popisuje RIPE sondy, ako aj samotný RIPE Atlas projekt a súčasné spôsoby jeho využitia. Predstavuje motiváciu pre tvorbu vlastnej aplikácie a špecifikuje jej požiadavky, no predovšetkým predstavuje detailný návrh, implementáciu a zhodnotenie vyvinutej aplikácie.

Výsledkom práce je funkčná implementácia popisovanej aplikácie, pozostávajúcej z komponenty agenta a komponenty serveru, ktorá spĺňa špecifikované požiadavky.

Ako už bolo spomínané, cieľom tejto práce bolo vytvorenie aplikačného rámca a príprava prostredia s viacerými agentmi. Funkčnosť riešenia bola aktuálne prezentovaná ukážkovou triedou s využitím naivnej Bayesovej klasifikácie, no cieľom do budúcnosti je rozšírenie implementácie agenta o viacero tried (typov meraní) aj s nasadením federatívneho učenia.

Ďalším zaujímavým vylepšením môže byť rozšírenie implementácie o distribuovanú streamovaciu platformu, typu Apache Kafka[1]. Platforma by mohla byť nasadená na úroveň API komunikácie medzi agentom a RIPE sondou (prípadne aj agentom a MongoDB), čo by umožnilo vytvorenie a spracovanie vyššieho počtu meraní (resp. podporu väčšieho počtu agentov) za daný časový úsek.

1. `https://kafka.apache.org`

Bibliografia

1. *What is RIPE Atlas?* [online]. RIPE Atlas [cit. 2022-12-30]. Dostupné z : https://atlas.ripe.net/about/.
2. *Regional Internet Registry* [online]. RIPE NCC [cit. 2022-12-30]. Dostupné z : https://www.ripe.net/about-us/what-we-do/regional-internet-registry.
3. *RIPE NCC Service Region* [online]. RIPE Atlas [cit. 2022-12-30]. Dostupné z : https://www.ripe.net/about-us/what-we-do/ripe-ncc-service-region.
4. *RIPE Atlas Probes* [online]. RIPE Atlas [cit. 2022-12-30]. Dostupné z : https://atlas.ripe.net/about/probes/.
5. [online]. [cit. 2023-04-20]. Dostupné z : https://storage.googleapis.com/abcdc-web-static/content/post/2016/03/host-a-ripe-atlas-probe/images/4.jpg.
6. DAVIES, Alun. *RIPE Atlas Software Probes* [online]. RIPE Labs [cit. 2023-01-03]. Dostupné z : https://labs.ripe.net/author/alun_davies/ripe-atlas-software-probes//.
7. *REST API Manual* [online]. RIPE Atlas [cit. 2023-01-03]. Dostupné z : https://atlas.ripe.net/docs/apis/rest-api-manual/.
8. MELAMPY, Patrick. *Networking anomalies* [online]. Network World [cit. 2023-03-22]. Dostupné z : https://www.networkworld.com/article/3284939/networking-anomalies.html.
9. NIST/SEMATECH. *e-Handbook of Statistical Methods* [online]. [cit. 2022-11-25]. Dostupné z : https://doi.org/10.18434/M32189.
10. HOU, Bingnan; HOU, Changsheng; ZHOU, Tongqing; CAI, Zhiping; LIU, Fang. Detection and Characterization of Network Anomalies in Large-Scale RTT Time Series. *IEEE Transactions on Network and Service Management*. 2021, roč. 18, č. 1, s. 793–806. Dostupné z DOI: 10.1109/TNSM.2021.3050495.

11. SHAH, Anant; FONTUGNE, Romain; ABEN, Emile; PELSSER, Cristel; BUSH, Randy. Disco: Fast, good, and cheap outage detection. In: *2017 Network Traffic Measurement and Analysis Conference (TMA)*. 2017, s. 1–9. Dostupné z DOI: 10.23919/TMA.2017.8002902.
12. FLORIS Wolfram, Sybren; KOEN. *RIPE Alerts* [online]. GitHub [cit. 2023-01-28]. Dostupné z: https://github.com/Wolframfriele/ripe-alerts.
13. KOLI, Mikael. *Rocketry* [online]. Rocketry [cit. 2023-04-03]. Dostupné z: https://rocketry.readthedocs.io/en/stable/#.
14. FASTAPI. *FastAPI* [online]. FastAPI [cit. 2023-04-03]. Dostupné z: https://fastapi.tiangolo.com/#interactive-api-docs.
15. JONES, Michael B.; BRADLEY, John; SAKIMURA, Nat. *JSON Web Token (JWT)* [RFC 7519]. RFC Editor, 2015. Request for Comments, č. 7519. Dostupné z DOI: 10.17487/RFC7519.
16. AHMED, Salman; MAHMOOD, Qamar. An authentication based scheme for applications using JSON web token. In: *2019 22nd International Multitopic Conference (INMIC)*. 2019, s. 1–6. Dostupné z DOI: 10.1109/INMIC48123.2019.9022766.
17. MONGODB. *What Is MongoDB?* [online]. MongoDB [cit. 2023-04-14]. Dostupné z: https://www.mongodb.com/what-is-mongodb.
18. REACT. *Writing Markup with JSX* [online]. React [cit. 2023-04-04]. Dostupné z: https://react.dev/learn/writing-markup-with-jsx.
19. REACT. *Describing the UI* [online]. React [cit. 2023-04-04]. Dostupné z: https://react.dev/learn/describing-the-ui.
20. REACT. *Virtual DOM and Internals* [online]. React [cit. 2023-04-04]. Dostupné z: https://legacy.reactjs.org/docs/faq-internals.html.
21. MISHRA, Ashutosh. *React vs. JavaScript: Why React Web Apps Are Better than Plain Websites* [online]. HackerNoon [cit. 2023-04-04]. Dostupné z: https://hackernoon.com/react-vs-javascript-why-react-web-apps-are-better-than-plain-websites.

22. TEAM, Indeed Editorial. *What is Bootstrap? (With benefits and uses)* [online]. Indeed [cit. 2023-04-06]. Dostupné z : https://uk.indeed.com/career-advice/career-development/what-is-bootstrap.

23. TURING. *What is Bootstrap? The Most Detailed Guide on Bootstrap for 2023* [online]. Turing [cit. 2023-04-06]. Dostupné z : https://www.turing.com/kb/detailed-guide-on-bootstrap.

24. FOUNDATION, Interaction Design. *Responsive Design* [online]. Interaction Design Foundation [cit. 2023-04-06]. Dostupné z : https://www.interaction-design.org/literature/topics/responsive-design.

25. DAVIDSON, Tim. *What is Vite, And Why Is It Awesome?* [online]. CleanCommit [cit. 2023-04-05]. Dostupné z : https://cleancommit.io/blog/what-is-vite/.

26. KIKUCHI, Takuya. *Why is Vite so fast?* [online]. DEV [cit. 2023-05-03]. Dostupné z : https://dev.to/takuyakikuchi/why-is-vite-so-fast-163p.

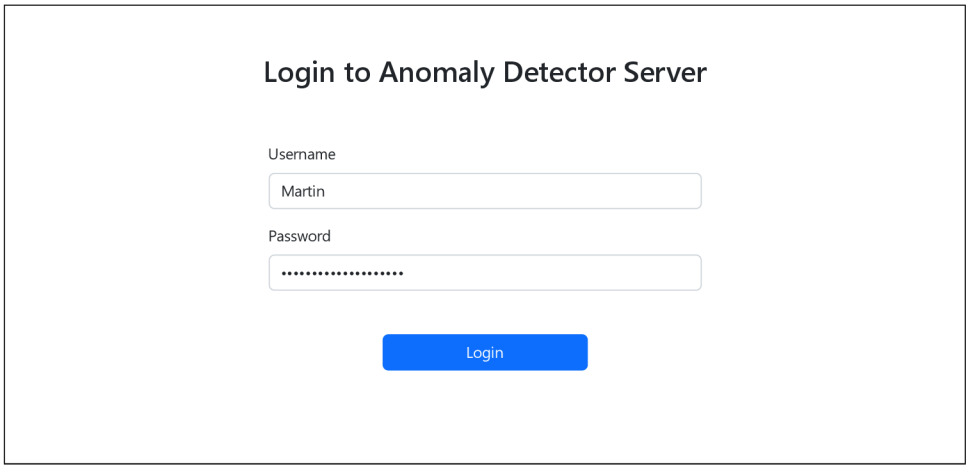
27. VITE. *Why Vite* [online]. Vite [cit. 2023-05-03]. Dostupné z : https://vitejs.dev/guide/why.html.

A Elektronická príloha

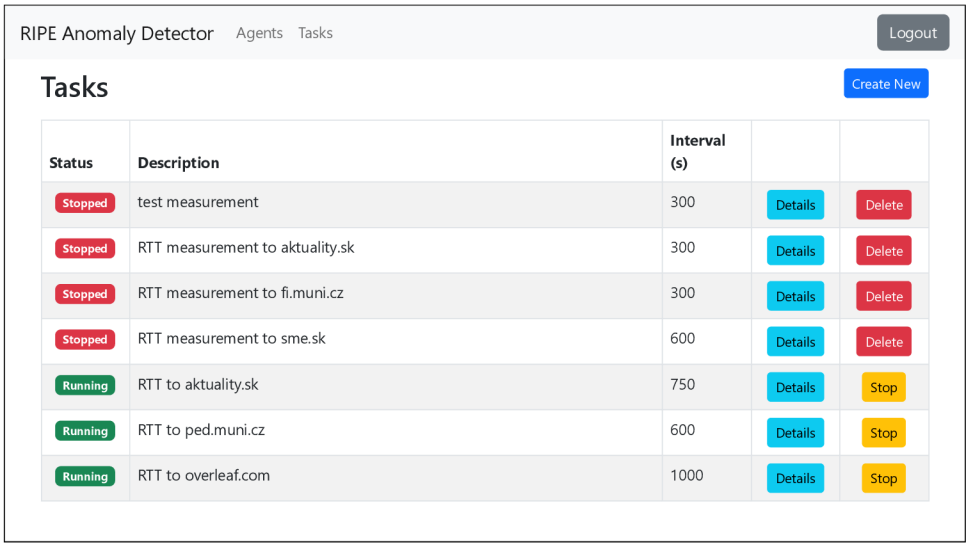
Obsahom archívu priloženého k diplomovej práci sú nasledujúce prílohy:

- *ripe-anomaly-detector-agent* - priečinok obsahujúci zdrojový kód komponenty agenta
- *ripe-anomaly-detector-server* - priečinok obsahujúci zdrojový kód komponenty serveru

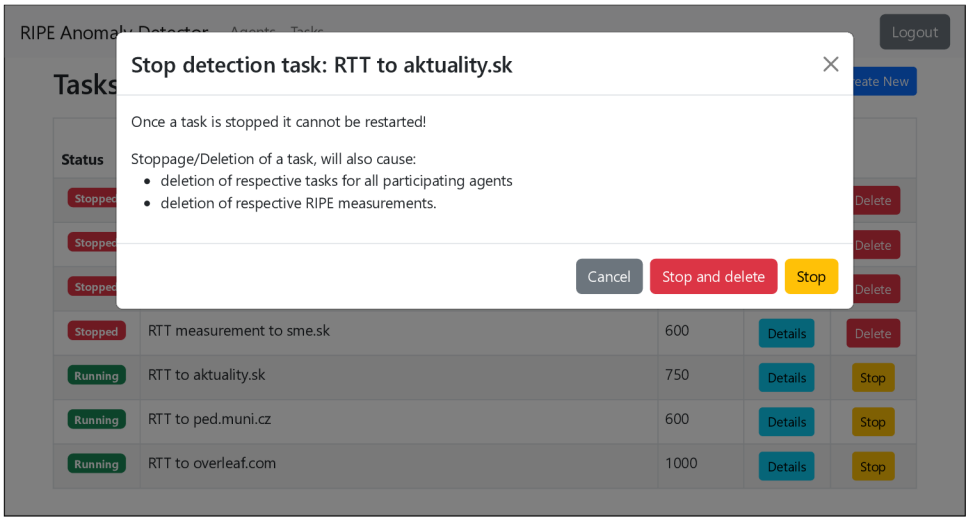
B Ukážka vytvoreného používateľského rozhrania



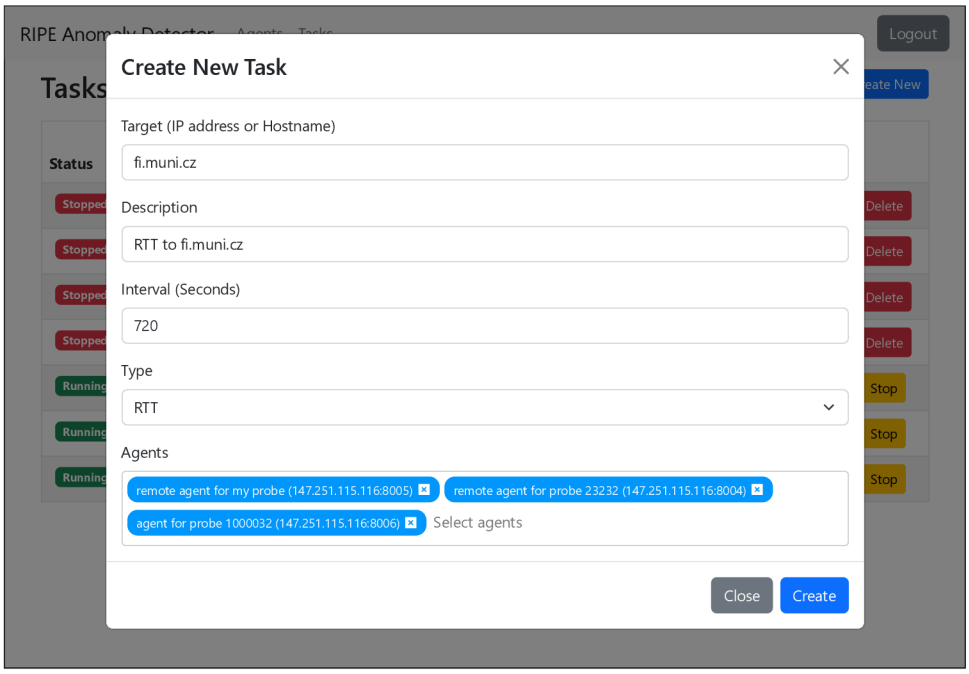
Obr. B.1: Prihlasovacia stránka



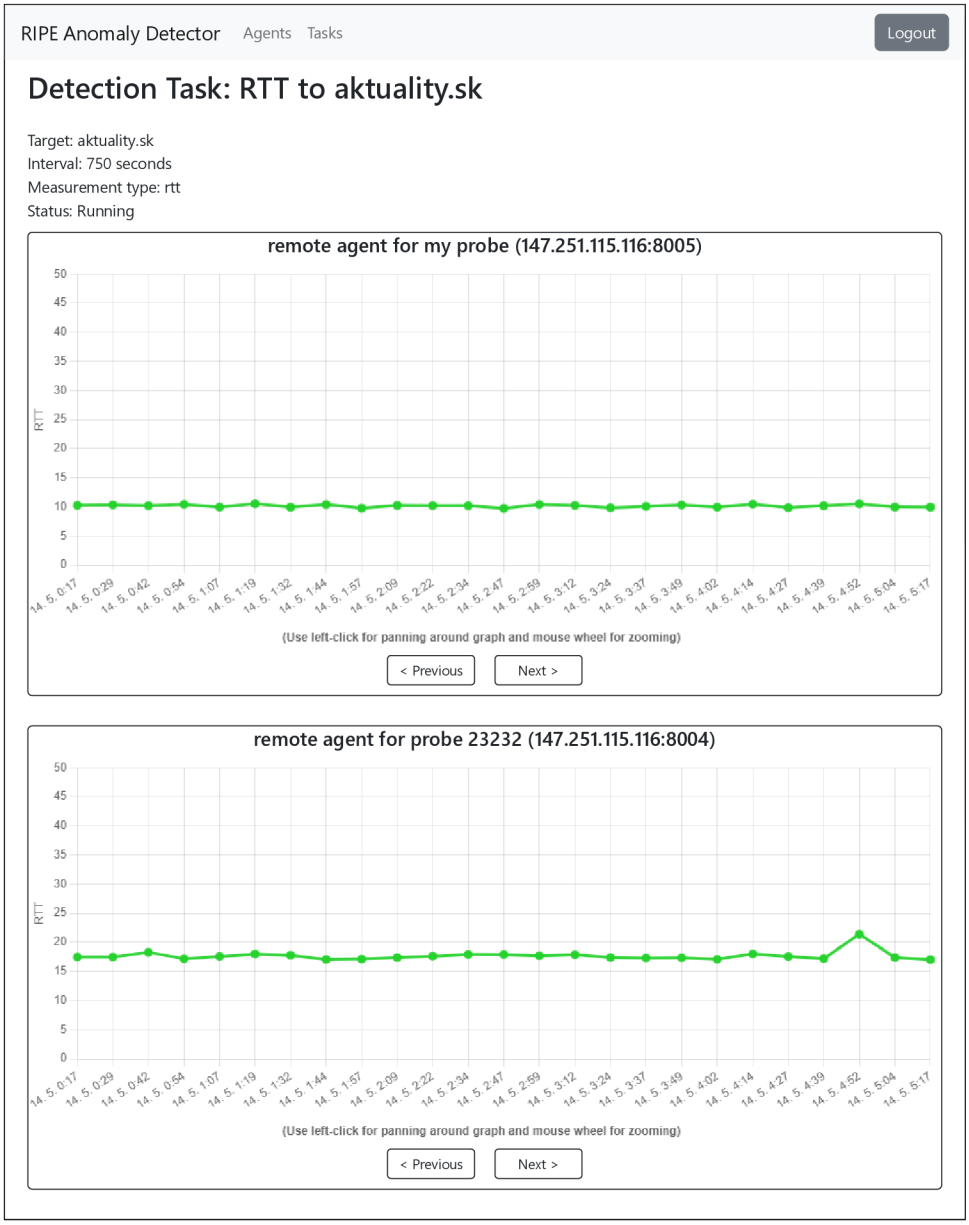
Obr. B.2: Stránka zobrazujúca zoznam detekčných úloh



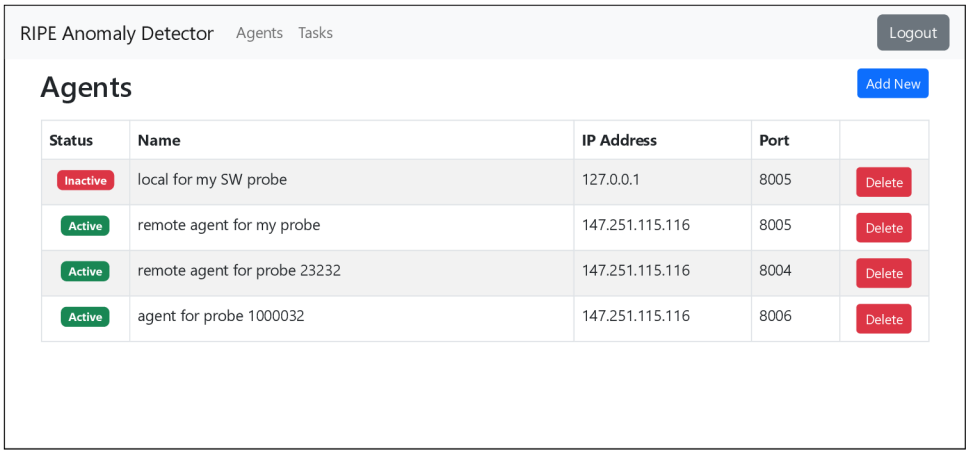
Obr. B.3: Stránka zobrazujúca dialóg pre zmazanie detekčnej úlohy



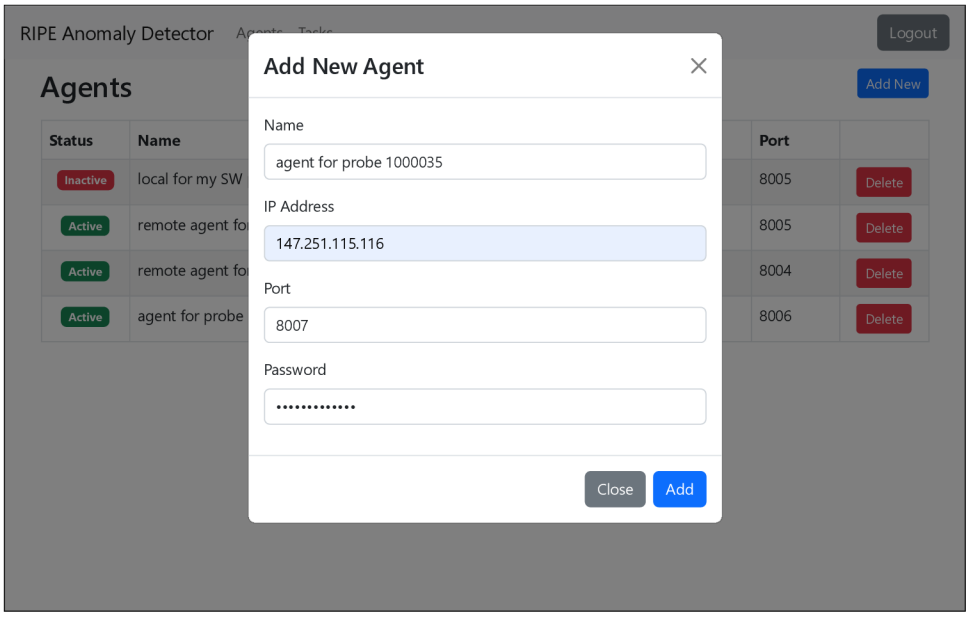
Obr. B.4: Stránka zobrazujúca dialóg pre vytvorenie novej detekčnej úlohy



Obr. B.5: Stránka zobrazujúca detail detekčnej úlohy



Obr. B.6: Stránka zobrazujúca zoznam agentov



Obr. B.7: Stránka zobrazujúca dialóg pre pridanie nového agenta