Masarykova univerzita
Fakulta informatiky



Komponenta pro manipulaci s daty analytického systému CopAS

Bakalárska práca

Jozef Cibík

Brno, jar 2018

Masarykova univerzita
Fakulta informatiky



Komponenta pro manipulaci s daty analytického systému CopAS

Bakalárska práca

Jozef Cibík

Brno, jar 2018

Na tomto mieste sa v tlačenej práci nachádza oficiálne podpísané zadanie práce a vyhlásenie autora školského diela.

Prehlásenie

Prehlasujem, že táto bakalárska 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.

Jozef Cibík

Vedúci práce: RNDr. Tomáš Rebok, Ph.D.

Poďakovanie

Ďakujem vedúcemu bakalárskej práce RNDr. Tomášovi Rebokovi, Ph.D., za motiváciu a odbornú pomoc a svojej blízkym za podporu, ktorú mi poskytli počas doterajšieho štúdia.

Abstrakt

Cieľom tejto práce je pokračovanie vo vývoji existujúcej komponenty „Nástroj na automatizáciu práce so vstupnými dátami pre Elasticsearch [1].“ Tá je súčasťou analytického systému CopAS, jedného z projektov výpočetného centra CERIT-SC. Systém je dnes už naplno využívaný sieťovými analytikmi. Poskytuje im možnosť automatizácie opakujúcich sa pracovných činností. V tejto práci sa nadväzuje na vývoj prototypovej komponenty zodpovednej za interakciu medzi užívateľom a systémom CopAS spolu s popisom dátovej a sieťovej analytiky. K týmto oblastiam sú uvedené aj pomocné nástroje, informácie o nich a ich aplikácia. Popis prototypu implementácie, jeho nedostatkov a potrieb vylepšenia, ktoré boli definované pri jeho používaní, je spojený s popisom novej verzie a krokmi, ktoré riešia tieto nedostatky. Cieľom novej verzie, okrem eliminácie nedostatkov, je aj poskytnúť výhody a softvérovú kvalitu a ukázať potenciál aj pre iné oblasti než len sieťová analýza.

Kľúčové slová

analýza dát, Big Data, sieťová analýza, CopAS, ELK Stack, prototyp, Neck, Bro, Tomcat, kvalita softvéru, implementácia

Obsah

Prehlásenie **i**

Poďakovanie **iii**

Zhrnutie **iv**

Kľúčové slová **v**

1 Úvod **1**

2 Big Data **3**
2.1 *Uchovávanie dát* . . . 3
2.2 *Analytika* . . . 4
2.3 *Služby* . . . 4
2.4 *Sieťová analýza* . . . 5
2.4.1 Analýza PCAP . . . 5
2.4.2 Detekcia sieťových útokov . . . 5
2.4.3 Monitorovanie siete . . . 6

3 ELK Stack **7**
3.1 *Elasticsearch* . . . 7
3.1.1 Popis architektúry . . . 8
3.2 *Logstash* . . . 9
3.3 *Kibana* . . . 9
3.3.1 Popis možností . . . 10
3.4 *Proces použitia ELK Stacku* . . . 10
3.4.1 Logstash nahratie . . . 11
3.4.2 Elasticsearch ukladanie . . . 12
3.4.3 Kibana vizualizácia . . . 13
3.5 *ELK Stack nadstavby* . . . 13

4 Analytický nástroj CopAS **15**
4.1 *Projektové požiadavky* . . . 15
4.1.1 Architektúra CopAS . . . 16
4.1.2 Docker . . . 16
4.1.3 Analýza sieťových záchytov pomocou Bro NSM 17
4.1.4 Aplikačný server Tomcat . . . 17

4.2 *Neck* 18

5 Návrh novej arhitektúry **21**
5.1 *Vývoj požiadaviek* 21
5.2 *Popis základnej verzie* 22
5.2.1 SOLID princípy 23
5.2.2 Čistý kód 23
5.2.3 MVC 24
5.2.4 Spring a Spring Web MVC framework 25
5.2.5 MVVM 25
5.2.6 Knockout 26
5.2.7 RichFileManager 26
5.2.8 Maven a moduly 27
5.2.9 WAR Archív 28

6 Implementácia požiadaviek **29**
6.1 *Nastavenie a spustenie* 29
6.2 *Výber súborov* 30
6.3 *Výber nástroja* 32
6.3.1 Integrácia nového nástroja 33
6.4 *Konfigurácia transformačných polí pre Logstash* 33
6.5 *Nahratie do ES* 34
6.6 *Dodatočná implementácia* 35
6.6.1 Testy 35
6.6.2 Dokumentácia 36

7 Záver **37**
7.1 *Možnosť rozšírenia* 37
7.2 *Rozšírenie do iných oblastí* 38

Bibliografia **39**

Zoznam tabuliek

3.1 Príklad štruktúry invertovaného indexu *12*

Zoznam obrázkov

3.1 Platforma ELK Stack *7*
3.2 Proces spracovania údajov v ELK Stack *11*

4.1 Architektúra CopAS *16*
4.2 Úvodná stránka prototypu *18*
4.3 Konfiguračná stránka prototypu *19*
4.4 Záverenčná stránka prototypu *19*

5.1 Diagram prípadu použitia aplikácie *22*
5.2 MVC model *24*
5.3 MVVM model *26*
5.4 Pôvodná verzia pluginu RichFileManager *27*
5.5 Závislosť modulov *27*

6.1 Proces základnej práce *29*
6.2 Úvodná stránka *30*
6.3 Stránka manipulácie so súborovým systémom *31*
6.4 Výber nástroja *32*
6.5 Výber transformačných funkcií Logstash *34*
6.6 Nahratie do ES *35*

1 Úvod

Rozmanitosť dát, s ktorými je možné v dnešnej dobe manipulovať je rozsiahla. Ich generovanie vo forme dokumentov, obrázkov či videí, ktoré sú vstupom pre služby pracujúcimi s týmito zdrojmi, tiež vytvára údaje vo forme záznamov, poznámok a výstupov. Multiplikátorom je aj faktor toho, že technológie, služby alebo užívatelia všeobecne si medzi sebou potrebujú tieto údaje predávať napríklad cez komunikačné siete a Internet. Internet ako celosvetová sieť svojím obsahom dosahuje takmer nezmerateľný objem [2].

Veľké zdroje informácií rôznych formátov je vhodné spracovať, triediť, uchovať a vybrať z nich len podstatnú množinu na ďalšiu prácu. Analýza vstupov podľa zadaných kritérií je dôležitou vlastnosťou služieb využívaných v oblastiach dátovej analytiky. Získavaním dôležitých poznatkov sa zaoberajú aj ľudia pracujúci v oblasti sieťovej analytiky. Tá sa zaujíma o údaje získané z tokov sietí a zo sieťovej prevádzky, ktoré sa veľkosťou orientujú v jednotkách TB. Existuje mnoho nástrojov, ktoré sa zaoberajú nielen analýzou sieťových tokov alebo detekcií škodných aktivít v nich, ale aj všeobecne ich spracovávaním a transformáciou.

Spracovanie, ukladanie na dátové úložiská a schopnosť znázornenia údajov v čitateľnej alebo logickej forme je významnou vlastnosťou pre softvér využívaný v oblastiach analytiky. Príkladom sú aj nástroje skupiny ELK Stack[1]. Práve táto skupina je využívaná v analytickom systéme CopAS, ktorý využívajú aj sieťový analytici.

Systém slúži na zjednodušenie ich práce. Motiváciou projektu bolo odbremeniť ich od vykonávania náročných krokov inštalácií a nastavení. Účel spočíva v odstránení týchto prekážok a vo vytvorení prostredia vhodného na prácu. V priebehu vývoja projektu vznikol aj prototyp grafického rozhrania slúžiaceho na komunikáciu medzi užívateľom a systémom. Ukázal mnoho podnetov pre ďalšie rozšírenia, ale aj potreby, ktoré by mal používaný softvér v dnešnej dobe spĺňať.

Motiváciou pre vznik tejto práce je vytvorenie novej verzie prototypu, spĺňajúcej nové požiadavky. Nadväzujúca časť spočíva v navrhnutí dizajnu a štruktúry nového, užívateľsky prívetivého spôsobu

1. ELK Stack – skratka pre skupinu nástrojov Elasticsearch, Logstash, Kibana.

práce. S tým je spojená aj schopnosť manipulácie s dátami ako nahrávanie, výber dát, konfigurácia pomocných nástrojov a analýza. Tento návrh bol následne implementovaný vo forme webovej aplikácie, nasadenej do prostredia systému CopAS. Okrem eliminácie nedostatkov sa kladie dôraz aj na prezentačnú vrstvu aplikácie, kvalitu softvéru a budúcnosť tohto projektu.

V tejto práci sa nachádzajú informácie o dátovej a sieťovej analytike, používaných nástrojoch, ELK Stack-u ako jednom z používaných nástrojov a CopAS projekte, využívajúcim túto platformu. Oblasť zaoberajúca sa veľkými dátami je spomenutá v druhej kapitole. Tá bude popisovať aj sieťovú analytiku a nástroje používané v tejto oblasti. Príkladová skupina nástrojov ELK Stack je popísaná v tretej kapitole. Informácie o tom ako je systém navrhnutý sú uvedené v kapitole štvrtej. V piatej kapitole sú predstavené nové požiadavky, nedostatky prototypu, potreba novej verzie rozhrania, a vlastnosti softvéru. Implementačné riešenia sú predstavené v šiestej kapitole spolu s príkladmi používania. Sú popísané vo forme kódov, obrázkov a krokov k správnemu používaniu. Je dôležité spomenúť aj potenciál a ďalšie možné vylepšenia, čo je uvedené v záverečnej kapitole.

2 Big Data

Internet, celosvetová sieť počítačov komunikujúcich medzi sebou, je jeden zo zdrojov obsiahlych a komplexných informácií. Ich nezastaviteľný rast ovplyvnil odvetvia ekonomiky, kultúry a spôsobil vznik nových vedeckých oblastí alebo informačných technológií.

Tradičné databázové systémy, používané na uchovávanie zdrojov, a iné techniky analýzy vo forme relačných databáz sú tlačené na okraj svojich možností pri spracovávaní veľkého obsahu. Reakciou na tento problém bol vznik napríklad technológie NoSQL. V istých smeroch je komplexnejšia ako klasické databázy a v ďalších zase jednoduchšia. Takto navrhnuté systémy sú schopné škálovať väčšiu množinu dát, ale stále je potrebné vytvoriť nové technológie navrhnuté konkrétne pre túto oblasť [2].

Oblasť Big Data sa zaoberá obsiahlymi dátami rôznych formátov a potrebu ich efektívnej analýzy a spracovania, ktoré je finančne a časovo akceptovateľné a inovatívne [3]. Výsledkom analýzy je vznik vzorcov, grafov a formátov vhodných na ľudskú interakciu. Zámer tejto oblasti pozostáva z činností ako zachytávanie, ukladanie, analýza, vyhľadávanie, prenos, zdieľanie a vizualizácia dát. Vzniklo mnoho konceptov o tom, čím sa Big Data zaoberá. Základom boli však iba tri a to objem, rôznorodosť a rýchlosť spracovania dát [4]. Neskôr k nim pribudli koncepty zaoberajúce sa ich pravdivostnou hodnotou či presnosťou [5] a neskôr aj ich významovou hodnotou [6].

2.1 Uchovávanie dát

Pri zaobchádzaní s Big Data vzniká istá skupina prekážok. Prvou z nich je uchovávanie a záloha veľkého obsahu. Dnes už lacné úložiská a s nimi spojené rôzne technológie riešia tento problém. Preto sa postupe začína prechádzať viac k rýchlosti prístupu k dátam. Dopyt po rýchlejších úložiskách viedol k zvýšenej pozornosti smerom k firmám vyrábajúcim SSD disky (FusionIO, Western Digital, Kingston, ...). Dátové úložiská typu NoSQL taktiež pomaly začínajú využívať SSD disky. Budúcnosť tradičných relačných databáz podľa výskumov môže byť hlavne v ich uchovávaní v pamätiach, ku ktorým budú pristupovať aplikácie vyžadujúce častý prístup [7].

2.2 Analytika

Ďalším z problémov je analýza dát. Tu sa na rýchlosť nekladie taký dôraz. Vzhľadom na to, že údaje sú adresované na istom mieste v pamäti, z ktorého ich je možné získať pomocou sofistikovaných algoritmov, je to otázka niekoľkých sekúnd. Výzvou je však manipulácia s nimi v reálnom čase s aktuálnou informačnou hodnotou medzi mnohými distribuovanými zdrojmi. Jednou z kľúčových stratégií je presunúť logiku hlbšie do miest, kde sa nachádzajú uložené dáta, aby sa obmedzilo vyťaženiu siete. To však vyžaduje, aby úložisko bolo úzko prepojené s analytickými algoritmami. Tie často musia byť prepísané podľa užívateľských nárokov v jazyku kompatibilnom s jazykom používaným v dátovej vrstve. Jedno z riešení poskytla firma HBase vo forme koprocesorov schopných konverziu kódu napísaného v jazyku Java do príslušných tabuliek s hodnotami a minimalizovala tak požiadavky prenášané cez sieť. Firma Netezza vylepšila návrh produkcie hardvéru do podoby FPGA[1] jednotiek obsahujúcich rôzne funkcie. Fyzicky sa nachádzajú na rovnakom mieste ako úložné disky.

Zatiaľ čo rozsah možností a nástrojov sa stále rozširuje, väčšina z nich a aj z používaných algoritmov nie je poskytnutá verejnosti. To vytvára priestor, v ktorom vznikajú nové produkty poskytujúce analytické služby pre klientov [7].

2.3 Služby

Posledným z problémov sú služby. Je to priestor, v ktorom je potrebné aby sa stretli produkty so zákazníkmi a firmami. Niekedy sú poskytnuté vo forme API[2] než iba programu. Monitorovanie kreditných kariet, finančných transakcií a zistenie podvodov je jednou z foriem dnes už existujúcich služieb [7]. Ďalším príkladom je monitorovanie siete a prenesených dát na detekciu možných kybernetických útokov pomocou komplexných algoritmov. Slúži to na ich prevenciu a poskytnutie ochrany osobných údajov, ktoré by inak mohlo mať kritické následky.

1. Field programmable gate array – je typ programovateľného logického integrovaného obvodu
2. Application programming interface – Rozhranie pre programovanie aplikácií

2.4 Sieťová analýza

Sieťová analýza je oblasť zaujímajúca sa konkrétne o informáciách prenášaných cez sieť. V posledných rokoch sa pri komunikácií cez Internet obsah prenášaných údajov stále zvyšuje, čo robí prácu sieťových analytikov náročnejšou. Nástroje používané v tejto oblasti sú zodpovedné za monitorovanie siete, detekciu útokov a anomálií. Mali by poskytovať funkcionalitu toho, aby získané znalosti boli analyzované v priebehu niekoľkých sekúnd aj v prípade, že forma vstupov je vo forme vysokorýchlostných prúdov. Mali by byť schopné rozoznávať iba istú konkrétnu časť prúdov. Vysoká rôznorodosť dát otvára priestor aj pre aplikácie využívajúce strojové učenie (*Machine Learning*) na zlepšenie klasifikácie získaných poznatkov alebo útokov.

2.4.1 Analýza PCAP

Analýza sieťových záchytov vo formáte PCAP[3] súborov je činnosť, kedy treba z veľkej množiny údajov vybrať iba požadovanú časť a spracovať ju. Súčasťou týchto súborov sú informácie o balíkoch (*paketoch*) prenesených počas aktívneho monitorovania siete. Na uľahčenie tejto činnosti slúžia nástroje ako TCPDUMP [8] alebo WireShark [9]. Rozdiel v týchto konkrétnych nástrojoch je hlavne v tom, že WireShark poskytuje aj grafické rozhranie.

2.4.2 Detekcia sieťových útokov

V prípade ochrany osobných údajov je potrebné predchádzať útokom prenášaným cez sieť. Aby bolo možné obmedziť vznik akejkoľvek poškodzujúcej aktivity, je nutné rýchlo takýto útok detegovať [10]. Existujú na to mnohé metódy, ktorých súčasťou sú algoritmy schopné detekcie útokov využívajúcich práve aj modely strojového učenia, ktoré zvyšujú sieťovú bezpečnosť. Tými nástrojmi môžu byť aplikácie ako Snort [11], Surikata [12] alebo Big-DAMA[13]. Big-DAMA využíva práve aj strojové učenie na detekciu útokov a poskytuje možnosť prehľadného monitorovanie siete, analýzy veľkých dátových zdrojov v rôznej podobe dávok alebo prúdov.

3. Packet Capture – Súbor obsahujúci dáta zo sieťovej aktivity

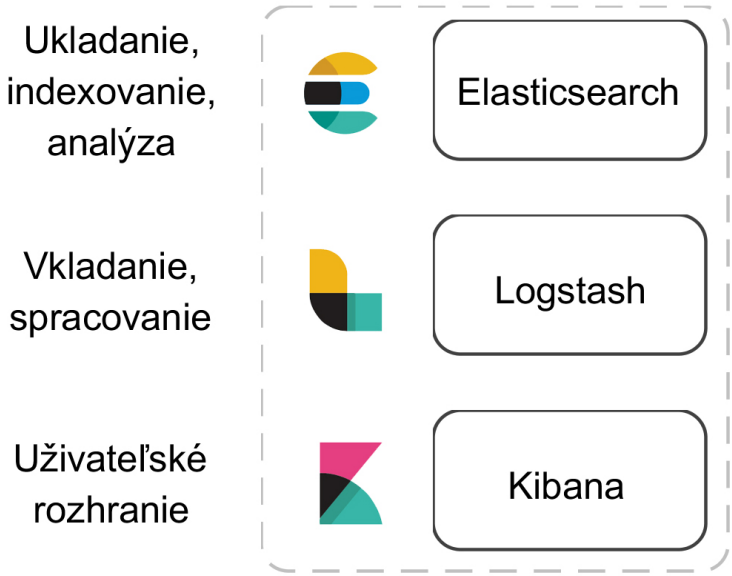
2.4.3 Monitorovanie siete

Ak je motiváciou monitorovanie siete, teda sledovanie počítačovej siete, zaistenie jej bezpečnosti alebo vytváranie štatistík vhodnou voľbou môže byť Stream4Flow. Je schopný analyzovať, monitorovať a zobraziť výsledky v reálnom čase. Je primárne určený na sieťovú bezpečnosť, môže sa však použiť aj v iných oblastiach *Big Data* [14]. Okrem neho je hodné zmieniť aj projekty ako Flowmon [15], Icinga [16], DataDog [17] alebo Solarwinds Network Packet Capture [18].

Rýchly vývoj produktov schopných spracovať veľké objemy dát v poslednej dobe pokročil vpred s čím vznikajú nové techniky kladené na oblasť Big Data. Produkty môžu medzi sebou zdieľať istú podobnosť a zároveň byť odlišné v tom, ako, kde a na aké úlohy sú využívané. Spomínané techniky založené na strojovom učení sú síce efektívne a skrývajú v sebe potenciál, ale zatiaľ sú len v štádiu vzniku alebo testovania než v podobe rozšírenej implementácie do produktov.

3 ELK Stack

Dnes už často využívanou platformou na prácu v oblasti Big Data je aj súprava nástrojov z kolekcie ELK Stack. ELK Stack je skratka pre trojicu voľne dostupných nástrojov Elasticsearch (ďalej už len ES) [19], Logstash [20] a Kibana [21]. Projekty sú vyvíjané firmou Elastic. Vznikli za účelom zobrazovania veľkých objemov dát v reálnom čase. Táto infraštruktúra poskytuje užívateľom vykonávať zber, analýzu podľa vlastných parametrov a vizualizáciu dát v jednom prostredí.



Obr. 3.1: Platforma ELK Stack

3.1 Elasticsearch

Prvou časťou je ES. Flexibilná a škálovateľná vyhľadávajúca služba, využívajúca knižnicu Apache Lucene, ktorá je založená na vyhľadávaní veľkého objemu záznamov hlavne vo formáte JSON[1], čo umožňuje komunikáciu aj s programami napísanými v rôznych programovacích jazykoch.

1. JavaScript Object Notation – forma zápisu objektu v jazyku JavaScript

ES je distribuovaná, RESTful vyhľadávacia knižnica. Primárne sa sústredí na vyhľadávanie a ukladanie informácií rýchlym, bezpečným a robustným spôsobom. Umožňuje kombinovať vyhľadávacie parametre podľa užívateľských požiadaviek, či už sú to textové vyhľadávania, štruktúrované dáta, merania alebo geografické polohy. Jedna z výhod spočíva v rýchlosti vyhľadávania uložených dát pomocou komplexných algoritmov a dátových štruktúr ako napríklad BKD stromy[2].

3.1.1 Popis architektúry

ES pôsobí ako NRT[3] platforma. Znamená to, že existuje malá odozva medzi časom kedy sú záznamy ukladané a časom, kedy ich je možné vyhľadávať. Miestom, kde sú tieto záznamy uchovávané, môže byť jeden alebo viac klastrov.

Klaster (Cluster) je združenie jedného alebo viacerých uzlov, ktoré dohromady v sebe majú uložené dáta. Poskytujú možnosti vkladania (v kontexte ES ďalej už len indexovania) a vyhľadávania vo všetkých uzloch.

Uzol (Node) sa pokladá za jednu inštanciu servera, ktorý je súčasťou klastra. Prispieva k optimálnemu indexovaniu a vyhľadávaniu. V jednom klastri môže existovať ľubovoľný počet uzlov. Obsahom uzla sú indexy.

Index (Index) je kolekcia dokumentov, ktoré medzi sebou zdieľajú istú formu podobných charakteristík. Meno indexu slúži ako identifikátor, pomocou ktorého je možné vyhľadať konkrétne dáta. V jednom klastri môže existovať ľubovoľný počet indexov.

V indexe je možné definovať jeden alebo viac typov (Types). Tie slúžia na reprezentovanie logickej kategórie indexu. Sémantika indexu môže byť akákoľvek. Typ umožňuje definovať vlastnosť polí v dokumente, ktoré držia v sebe istý formát údajov.

Základná jednotka, ktorá dokáže byť indexovaná sa nazýva dokument (Document). Je vyjadrený vo formáte JSON. Takýto formát poskytuje výhodu kompatibility v iných platformách a programovacích jazykoch. Ukladanie veľkého množstva indexovateľných dát do jedného dokumentu prináša riziko prekročenia limitov fyzického zariadenia, na ktorom sú informácie uložené. V prípade 1TB dokumentu

2. BKD Stromy – Dátová štruktúra zvyšujúca efektivitu pri vyhľadávaní údajov
3. NRT – Near real time teda schopná takmer okamžitej odozvy

sa môže táto kvóta ľahko presiahnuť alebo spôsobiť pomalé vyhľadávanie v jednom uzle. Riešením je rozdelenie dokumentu na úlomky (Shards). Počet úlomkov je definovaný pri vytváraní dokumentu. Funkcia úlomku je podobná indexu. Výhodou je rozdelenie obsiahlych dát, prípadne distribuované vykonávanie paralelných požiadaviek, čím sa zvyšuje výkon.

Repliky (Replicas) slúžia ako záložné body v prípade, že jeden z uzlov zlyhá alebo sa poškodí. Taktiež v prípade vyhľadávanie škáluje rozsah vyhľadávaných informácií naprieč všetkými replikami, nie len jednej verzií dát

3.2 Logstash

Druhou komponentou je Logstash. Slúži na vkladanie dát do ES. Nachádza sa na strane servera a slúži ako konzument prichádzajúcich správ a tokov dát. Tie dynamicky spracuje, transformuje do jednotnej podoby a pošle do ES. Následne sú tieto záznamy rozposlané v rôznych formách do mnohých systémov alebo uzlov klastrov. Vstupné informácie na uloženie môžu pochádzať z rôznych zdrojov vo formáte textových logov, výpisov, databáz a prúdov.

Z týchto prúdov je Logstash schopný identifikovať podstatné informácie na spracovanie a transformovať ich do podoby vhodnej na analýzu. Pri nahrávaní je schopný zabezpečiť opakovanie procesu pri akomkoľvek zlyhaní komunikácie alebo vyrovnávať rýchlosti medzi prijímaním a odosielaním prúdov.

3.3 Kibana

K ES a Logstash ešte pribudla Kibana. Je to nástroj na prehľadnú vizualizáciu spracovaných informácií. Pomocou webového rozhrania umožňuje vizualizovať záznamy uložené v ES vo formáte tabuliek a grafov. Forma týchto vizualizácií je natoľko neobmedzujúca, že poskytuje voľnosť akokoľvek záznamy v grafoch usporadúvať, združovať a tvarovať. Táto vlastnosť umožňuje náhľad na tie isté dáta z rôznych uhlov pohľadov a tým možnosť získať rôzne poznatky.

Dáta sú reprezentované v podobe stĺpcových grafov, histogramov alebo koláčových grafov. V prípade geografických dát je ich možné

formovať do teplotných máp (*Heat Maps*) spojených s geografickými miestami. Z hľadiska sieťovej analytiky je možné pozorovanie nie len štatistických údajov, ale aj dát v časových rozmedziach.

3.3.1 Popis možností

Súčasťou Kibany je konzola (Console). Poskytuje rozhranie medzi ES REST API a ostatnými aplikáciami alebo programami, ktoré chcú s touto službou komunikovať. Funguje na princípe zobrazenia ako požiadavky, tak aj odpovede tejto požiadavky na jednej obrazovke.

Dashboard umožňuje prehľad uložených vizualizácií údajov. Umožňuje základné operácie pridávania, editácie a vymazávania grafov a tabuliek podľa vlastného uváženia.

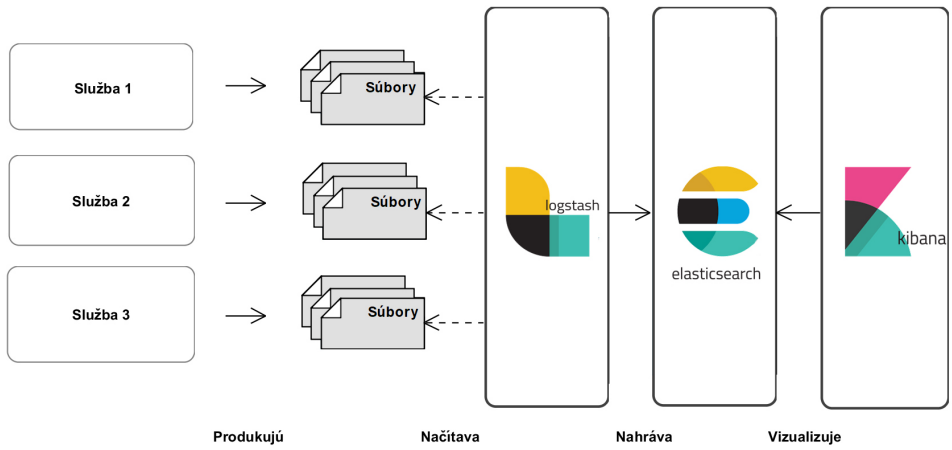
Discover zobrazuje dáta uložené v indexe a poskytuje interaktívnu úpravu. Je v ňom možné aktívne vybrať relatívne polia, časové úseky a rozsahy hodnôt

Ďalšou častou je aj manažment, v ktorom sa nastavuje správanie Kibany. Je tu možnosť pridávania pluginov a nástrojov na rozšírenie schopností vizualizácie dát. Pluginy sú prídavné nadstavby rozširujúce základnú funkcionalitu. Je ich možné rozdeliť na aplikačné (LogTrail), balíkové (X-Pack) alebo vizualizačné (3D Graf).

Vizualizácie (Visualize) sa vytvárajú pomocou preddefinovaných alebo vlastných parametrov formou istého sprievodcu. Výsledkom sú grafy a tabuľky prispôsobené vlastným požiadavkám.

3.4 Proces použitia ELK Stacku

Na to, aby sa mohlo s údajmi jednoducho manipulovať a vizualizovať ich, je najskôr potrebné prejsť postupnosťou inštrukcií.



Obr. 3.2: Proces spracovania údajov v ELK Stack

3.4.1 Logstash nahratie

Prvým krokom je použitie Logstash na načítanie vstupných údajov, či už z disku alebo prijímajúceho portu, ktorý je nastavený v Logstash konfigurácií. Tento krok je možné obohatiť o mnoho pluginov podporujúcich rôzne formáty zdrojov.

Vytvorenie konfigurácie slúži k špecifikácií toho, aké údaje sa majú sledovať, prípadne transformovať alebo filtrovať. Logstash poskytuje niekoľko desiatok funkcií vhodných na transformáciu dát. Nasledujúca ukážka príkladového Logstash súboru s nastaveniami popisuje transformáciu názvu poľa a sledovanie časovej známky v konkrétnom formáte.

```
1  { input {stdin {}}
2    filter {json {source => "message"}
3      date {
4        match => ["ts", "ISO8601"] remove_field => ["ts"]
5      }
6      mutate {
7        rename => {"history" => "past"}
8      }
9      ruby {code => "event.to_hash.keys.each
10             { |k| event.set(k.sub('.','-'),
                   event.remove(k)) if k.include?'.' }"
```

```
11      }
12    }
13    output {elasticsearch {hosts => ["localhost:9200"] }
14    }
15  }
```

Výsledok tejto konfigurácie bude znamenať, že Logstash spracuje dáta zo štandardného vstupu a pošle ich do ES klastra počúvajúceho na porte 9200. Ak vstupné dáta budú obsahovať pole s názvom *history*, tak sa pole premenuje na hodnotu *past*. Zároveň sa určuje pole, ktoré sa bude považovať ako časová známka, spĺňajúca časový formát ISO8601[4].

3.4.2 Elasticsearch ukladanie

Každé slovo v dokumente musí byť vyhľadateľné. Preto je potreba pracovať s dátovou štruktúrou schopnou efektívneho vyhľadávania a podpory držania viacerých hodnôt v jednom poli. Na to slúži štruktúra nazývaná invertovaný index (*inverted index*). Tá obsahuje utriedený list s unikátnymi hodnotami, ktoré sa vyskytujú v dokumente. Pre každú hodnotu je priradený zoznam dokumentov, kde sa táto hodnota alebo výraz nachádza [22].

Tabuľka 3.1: Príklad štruktúry invertovaného indexu

Výraz	Dokument 1	Dokument 2
Lorem		X
ipsum	X	
dolor	X	X

Postup indexovania sa takisto dá ovplyvniť pomocou mnohých definícií textových analyzátorov alebo matematických funkcií, ktoré môžu byť definované napríklad cez rozhranie Kibana konzoly.

4. ISO8601 – Štandard pre zápis času vo formáte rok-mesiac-deň (YYYY-MM-DD)

3.4.3 Kibana vizualizácia

Vizualizácie poskytujú prehľad nad dátami uloženými v ES. Je možné vytvoriť vlastné prehľady (dashboards) na zobrazenie údajov vo vlastne definovanej forme. Tieto sú založené na špeciálnych ES príkazoch (queries). Výsledkom sú grafy, tabuľky alebo 3D vizualizácie vytvorené pomocou mnohých ES agregovaných príkazov na združenie dát. Okrem znázornenia informácií uložených v ES je takisto schopný vizualizovať aj procesy nahrávania alebo analýzy, vďaka prídavným modulom.

3.5 ELK Stack nadstavby

ELK Stack je základom mnohých projektov určených na rôzne úlohy (Yelp, Mozilla, Spotify, ...). Samozrejme jeho funkcionalita sa dá rozšíriť o mnoho modulov. Zopár ponúka aj sama firma Elastic ako napríklad Beats [23], X-Pack [24] a ECE [25].

Pre Kibanu existujú rozšírenia na lepšie triedenie, kategorizáciu a vytváranie vizualizácií dát vo forme grafov, dashboardov a tabuliek. VulnWhisperer [26] slúži na oznamovanie zraniteľných miest v údajoch, LogTrail [27] na vytváranie záznamov o činnostiach alebo Timelion [28] a Siren Investigate (taktiež nazývane aj Kibi) [29], ktoré sú schopné združovať a filtrovať informácie z rôznych zdrojov, databáz a indexov.

Pre ES existujú knižnice rozširujúce vyhľadávanie, indexovanie, mapovanie, bezpečnosť alebo vzhľad rozhrania. Big desk [30] môže zlepšiť prehľad o štatistických údajoch o ES klastri, Search Guard [31] môže zvýšiť bezpečnosť pomocou autentizácie a autorizácie, Cerebro [32] alebo ElasticHQ [33] je schopné pristupovať k ES formou webového rozhrania a mnoho iných.

Nakoniec aj pre Logstash sú rozšírenia, ktoré umožňujú vylepšovať rýchlosť, rozsah spracovania alebo poskytnúť schopnosť monitorovať údaje nad ktorými sa vykonávajú operácie napríklad pomocou spomínaného X-Pack alebo LogIQ [34]. Nie je potrebné však nutne používať iba Logstash. Je možné použiť aj alternatívy spĺňajúce podobné kvality ako LogZoom [35], Filebeat [36], Kafka [37] a Apache Spark [38] ako alternatíva ES alebo GrayLog [39] ako náhrada za skupinu ELK Stack.

4 Analytický nástroj CopAS

CopAS je systém pozostávajúci z nástrojov, ktorých základ tvorí ELK Stack. Je zameraný na pracovanie s dátami primárne sieťovej prevádzky. Je to jeden z projektov výpočetného Centra CERIT-SC, operujúceho na Ústave výpočetnej techniky MU. Okrem ELK využíva nástroje ako Docker [40], Bro [41] a prototypovú verziu Neck ako základné užívateľské rozhranie komunikujúce s týmto systémom. Účel projektu je zrýchlenie a automatizácia procesu spracovania a analýzy dát sieťových analytikov.

4.1 Projektové požiadavky

Aj napriek mnohým prídavným modulom, ktoré sa dajú pridať do základov ELK Stacku tu vzniká situácia, kedy požiadavky kladené na projekt sa nedajú úplne spojiť s už existujúcimi projektami. Sieťoví analytici potrebujú vykonávať opakovaný súbor úkonov pozostávajúcich z inštalácií nástrojov, načítaní dát, konfigurácií nástrojov a niekedy náročnejšou prácou s príkazovým riadkom. V kontexte tohto projektu vzniklo niekoľko návrhov, ktoré bolo treba splniť ako napríklad:

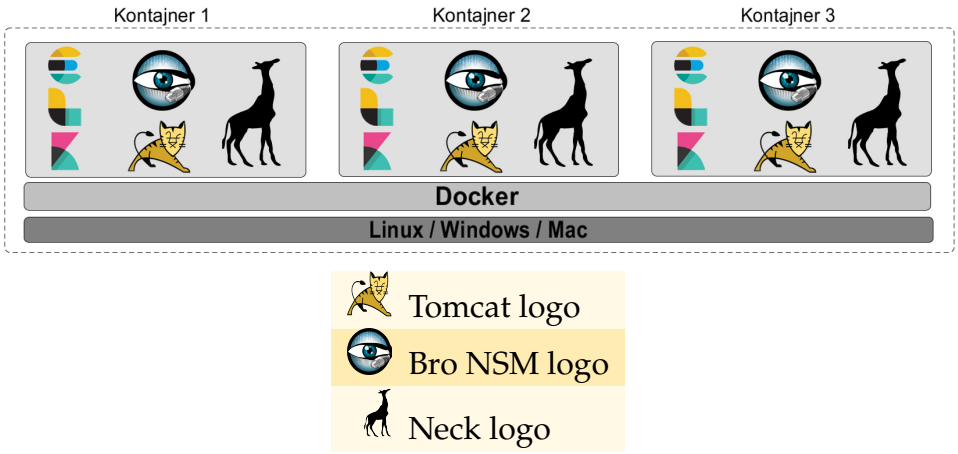
- automatizácia procesu spúšťacích, konfiguračných a nahrávacích príkazov,
- zníženie chybovosti a frekvencie bodov zlyhania pri manuálnych úkonoch,
- prehľad o pracovných údajoch, ich ukladanie a manipulácia s nimi.
- vytvorenie pracovného prostredia s inštalovanými nástrojmi,
- nastavenie nástrojov na spracovanie sieťových záchytov,
- zrýchlenie manuálnych krokov.

Vyššie popísaný zoznam reprezentuje len malú časť z požiadaviek, ktoré vznikli počas realizácie projektu. Tie boli zjednotené do jedného nástroja, vďaka čomu vytvorili jednotné pracovné prostredie

obsahujúce pomocné nástroje. Poslednou úlohou bolo treba vyvinúť komponentu spĺňajúcu úlohu grafického, užívateľsky prívetivého rozhrania, ktoré by bolo koncovým bodom pri komunikácií s týmto prostredím.

4.1.1 Architektúra CopAS

Nástroj slúži na vytvorenie pracovného prostredia obsahujúceho predinštalované programy, ktoré sú neskôr používané pri analýze. Nasledujúci obrázok znázorňuje architektúru CopAS:



Obr. 4.1: Architektúra CopAS

Systém sa skladá z požadovaného množstva medzi sebou nezávislých kontajnerov, obsahujúcich pripravené prostredie. Jeden kontajner je reprezentáciou jedného pracovného prostredia.

4.1.2 Docker

Každé prostredie sa vytvára v kontajneri nástroja Docker vyvíjaného firmou Docker. Kontajnerizácia alebo vizualizácia založená na kontajneroch je stratégia virtualizácie operačného systému, ktorá zahŕňa vytváranie na sebe nezávislých virtuálnych častí požadovaného hardvéru. Docker je implementácia tejto technológie. Systém CopAS je umiestnení v tomto virtuálnom prostredí avšak nie je podmienkou, že to musí byť realizované s použitím iba programu Docker. Tento

systém je možné použiť v akejkoľvek technológií pracujúcej s kontajnermi. Do prostredia bol zavedený operačný systém Linux a s ním predinštalované ELK Stack komponenty spolu s prídavnými programami ako Tomcat [42] a Bro. Docker poskytuje mnoho funkcií na správu a konfiguráciu, čo pomáha riešiť jeden z problémov nastavenia a inštalácie obsiahnutých programov.

4.1.3 Analýza sieťových záchytov pomocou Bro NSM

Bro Network Security Monitor je ďalší z predinštalovaných nástrojov systému CopAS. Analýza sieťových balíkov (*packetov*) sama o sebe nedáva zmysel, preto je potrebné poskladať zo súborov typu *Packet Capture* (PCAP) sieťové spojenie. Bro viac ako dostatočne splňuje úlohu získavania podstatných informácií. Okrem toho je schopný sledovať sieťovú prevádzku, detegovať útoky a produkovať výstupy vo forme záznamov a textových súborov.

Vzhľadom na požiadavky ELK Stacku je nutné, aby formát výstupu bol JSON. Bro umožňuje takúto konverziu pomocou jednoduchého príkazu, ktorý je treba vložiť do konfiguračného súboru:

```
@load policy/tuning/json-logs
redef LogAscii::json_timestamps = JSON::TS_ISO8601;
```

Konfiguračný súbor sa môže predať ako parameter pri spracovaní vstupných dát napríklad takto:

```
> ${CESTA_K_BRO}/bin/bro -r ${VSTUPNY_SUBOR}
   ${CESTA_K_BRO_KONFIGURACNEMU_SUBORU}
```

To spôsobí, že každý vstupný súbor bude transformovaný do formátu JSON. Prepínač -r slúži na načítanie vstupného súboru.

4.1.4 Aplikačný server Tomcat

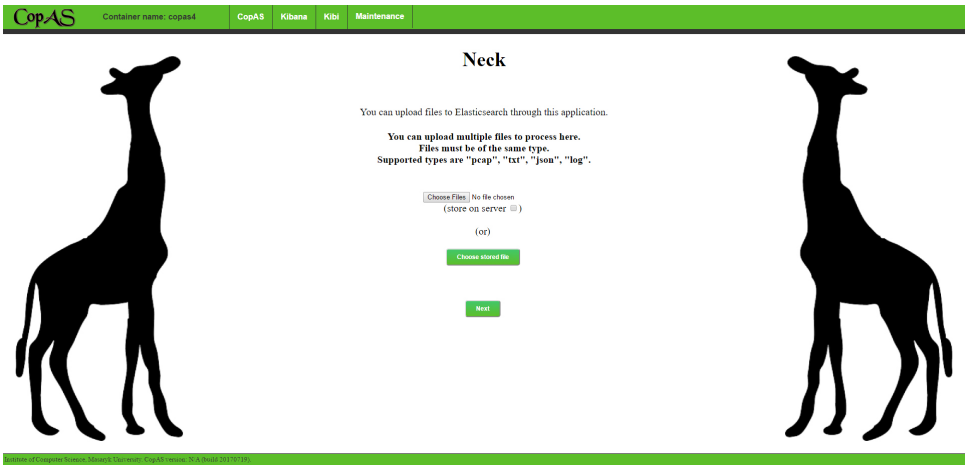
Predposledný z predinštalovaných nástrojov je Tomcat. Apache Tomcat je *open-source Java* servletový[1] kontajner (*servlet conatiner*) vyvíjaný

1. Java Servlet – program alebo trieda, ktorá rozširuje možnosti servera.

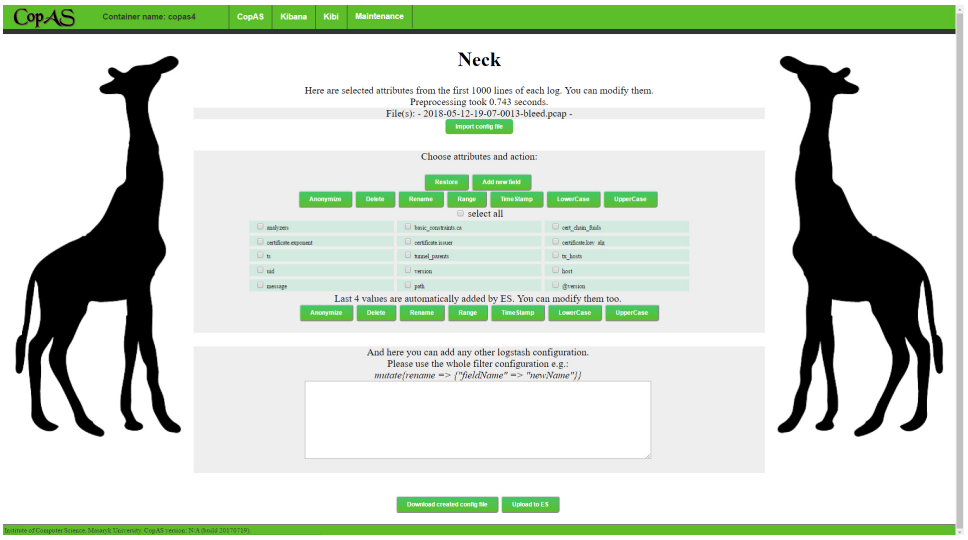
firmou Apache Software Foundation. Implementuje mnohé kontrakty spĺňajúce podmienky Java podnikových aplikácií ako Java servlety, Java stránky (*Java Server Pages*) a hlavne poskytuje možnosť vytvoriť webový server. Vďaka nemu je tak možné nasadiť do prostredia systému CopAS webovú aplikáciu, v ktorej užívateľ bude využívať jeho funkcionalitu.

4.2 Neck

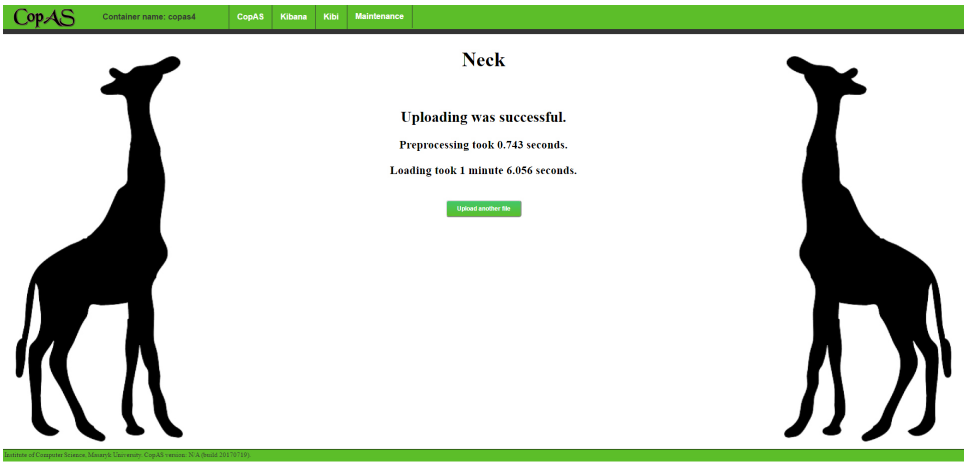
Poslednou časťou systému je Neck. Slúži ako koncový bod pri komunikácií a nahrávaní vstupov do ELK Stacku. Vďaka nemu je možné vytvoriť konfiguračné súbory, zvoliť transformácie nad vstupnými atribútmi, import PCAP súborov vo formáte JSON a všetko nahrať do zvoleného Elasticsearch klastra.



Obr. 4.2: Úvodná stránka prototypu



Obr. 4.3: Konfiguračná stránka prototypu



Obr. 4.4: Záverenčná stránka prototypu

5 Návrh novej arhitektúry

Prvá verzia splnila svoju úlohu prototypu a stala sa používaným rozhraním pri práci. Poskytovalo väčšinu možností, ktoré boli od procesu automatizácie požadované. Avšak používaním postupne začali vznikať situácie, ktoré obmedzovali užívateľa pri práci a preto pribúdali nové návrhy na vylepšenie.

5.1 Vývoj požiadaviek

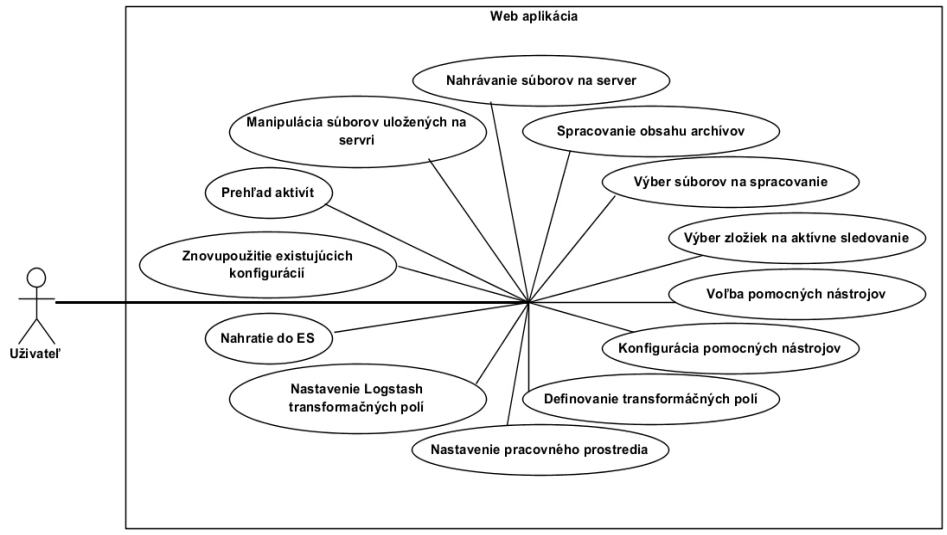
Súčasťou tejto práce je implementácia nových návrhov. Tie sú poskytnuté hlavne z pohľadu koncových užívateľov. Zahŕňa to:

- manipuláciu dát uložených na serveri bez potreby opakovaného nahrávania,
- aktívne sledovanie zložiek s dátami za účelom monitorovania zmien,
- konverziu rôznych typov vstupných dát do JSON formátu,
- znovupoužitie existujúcich konfigurácií pre Logstash,
- elimináciu obmedzenia veľkosti súborov nahrávania,
- integráciu iných pomocných nástrojov než len Bro,
- užívateľskú prívetivosť grafického rozhrania,
- podporu iných formátov než len PCAP,
- manipuláciu so súborovým systémom,
- prehľad súborov uložených na serveri,
- lokalizáciu.

Okrem nich sa bral ohľad aj na potreby týkajúce sa budúcich vývojárov pracujúcich na tomto projekte a zároveň, aby aplikácia spĺňala kvality produkčne využívaného softvéru vo forme prehľadnosti, zjednodušenia adaptácie nad prípadným pokračujúcim vývojom tejto aplikácie a celková kvalita.

5.2 Popis základnej verzie

V mnohých prípadoch je vhodné popísať architektúru aplikácie pomocou grafov, jazykov alebo diagramov na to, aby bolo jednoduché pochopiť jej účelu. Na demonštrovanie rôznych aktivít, ktoré je používateľ schopný vykonávať v aktuálnej verzii webovej aplikácie slúži nasledujúci UML diagram prípadu užitia:



Obr. 5.1: Diagram prípadu použitia aplikácie

Keďže rozhodnutím bolo pokračovanie vo vývoji existujúcej aplikácie, je potrebné, aby pri pridávaní nových funkcií bola aplikácia schopná akceptovať nové časti implementácie bez komplikácií alebo poškodenia korektnosti. Pri adaptácií na prevzatý kód však bola istá činnosť zbytočne zdĺhavá. Dôvodom bolo nepresné rozdelenie záujmov služieb v zdrojovom kóde a nedostatočná dokumentácia. Zároveň nové zmeny spôsobili častú nekorektnosť a chyby. Neistota v pridávaní nových častiach kódu rástla hlavne so situáciou, že prototyp neobsahoval žiaden spôsob kontroly a testovania pri pridávaní nových funkcií. Kvôli týmto prekážkam pribudlo k základným požiadavkám užívateľov aj potreba vytvorenia softvéru, ktorý by spĺňal istú formu kvality zjednodušujúcu vývoj.

5.2.1 SOLID princípy

Jedným z indikátorov dobre navrhnutého softvéru je vlastnosť plnenia istých postupov a princípov. Implementačná časť je napísaná v jazyku Java – objektovo orientovanom jazyku. SOLID princípy boli navrhnuté práve pre objektovo orientované jazyky. Je to skratka pre päť dizajnových princípov slúžiacich na to, aby softvér bol prehľadnejší, udržateľnejší, ľahšie modifikovateľný a lepší na čítanie.

Práve jedna zodpovednosť (*single responsibility*) spočíva v tom, že jedna entita by mala byť zodpovedná iba za jednu vec.

Schopnosť rozšíriť funkcionalitu, ale nie upraviť jej pôvodný kontrakt (*open for extension and closed for modification*).

Liskovov princíp zastúpenia (*Liskov substitution*) znamená, že rodičovské entity by mali byť nahraditeľné ich deťmi bez akejkoľvek zmeny korektnosti programu.

Segregácia rozhraní (*interface segregation*). Definovanie konkrétnych rozhraní je výhodnejšie než definovanie jedného veľkého rozhrania.

Inverzia závislostí (*dependency inversion*) je stav, kedy by entity mali byť závislé na abstrakciách a nie na konkrétnych implementáciách. Dôsledkom je zníženie prepojenosti závislostí entít. Výhodou nízkej prepojenosti je nezávislosť entít a tak aj ich jednoduchá modifikácia.

5.2.2 Čistý kód

Využitím rád definovaných v knihe „Clean Code: A Handbook of Agile Software Craftsmanship [43]" bolo dosiahnuté zlepšenie ďalšej vlastnosti z množiny atribútov softvérovej kvality a to prehľadnosť alebo čitateľnosť kódu. Tá často nie je definovaná užívateľom používajúcim daný produkt, ale výrazne oceňovaná z pohľadu vývojárov pracujúcich s kódom produktu. Konkrétne rady použité v novej verzii sú:

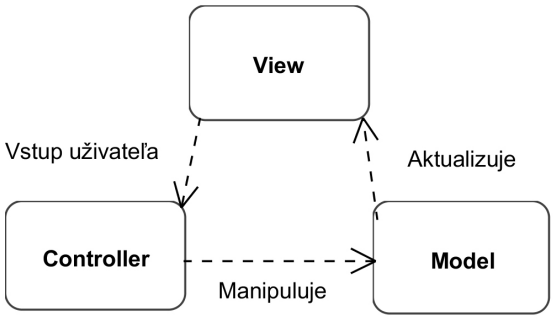
- deskriptívne názvy tried a funkcií,
- výnimky namiesto chybných kódov,
- používanie prídavných knižníc,
- formátovanie zdrojového kódu,

- jednoduché parametre funkcií,
- spracovanie chýb a výnimiek,
- použitie návrhových vzorov,
- eliminácia switch blokov,
- nízka duplikácia kódu,
- separácia záujmov,
- dokumentácia,
- testy.

5.2.3 MVC

V implementácií bol aplikovaný jeden z návrhových vzorov dizajnu softvérovej architektúry s názvom MVC – model(*model*), pohľad (*view*) a kontrolér (*controller*).

Model je špecifická doména aplikácie zahŕňajúca dáta, s ktorou sa manipuluje. Pohľad slúži na zobrazenie týchto dát užívateľovi interaktívnym spôsobom. Kontrolér má za úlohu riadiť udalosti, ktoré vznikli počas používania aplikácie a ako odpoveď na tieto udalosti, vykoná požadovanú akciu vo zvyšných dvoch komponentách. Výhodou takto zvoleného architektonického vzoru je prehľadnejší proces vývoja, nezávislosť častí a možnosť ľahkej zámeny prezentačnej vrstvy. Prezentačná vrstva aplikácie sa tak môže zmeniť bez ovplyvnenia funkcionality zvyšku aplikácie.



Obr. 5.2: MVC model

5.2.4 Spring a Spring Web MVC framework

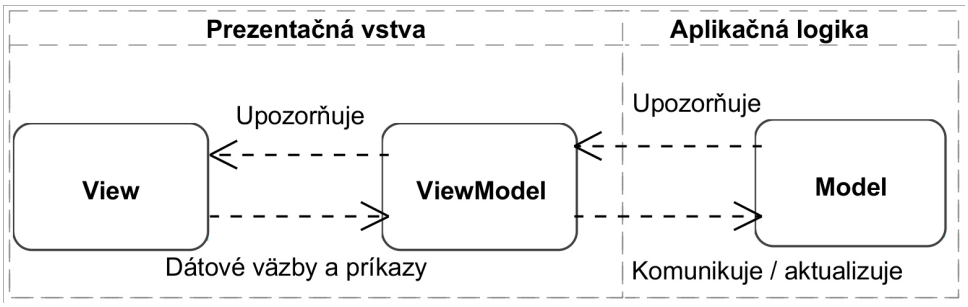
Spring [44] je *open-source framework* určený na vývoj podnikových aplikácií v jazyku Java. V Java komunite sa stáva čoraz viac obľúbenejší vzhľadom na to, že sa pokladá ako vhodná náhrada za *Enterprise JavaBeans* (EJB) komponentu. EJB je softvér na strane servera, ktorý zabaľuje logiku aplikácie a zároveň poskytuje prostredie pre iné webové komponenty (bezpečnosť, transakcie, ...).

Spring Web MVC [45] je webový *framework* postavený na servletovom aplikačnom rozhraní, vďaka ktorému je možné aplikovať dizajn MVC architektúry. Je zahrnutý v Spring *framework*-u už od jeho vzniku. Pokladá sa ako nástroj na jednoduché aplikovanie a implementáciu MVC návrhového vzoru do webových aplikácií.

Jedna z jeho výhod je plnenie dizajnovej kvality softvéru s názvom inverzia kontroly (*inversion of control*). Tá má za následok napríklad zvýšenie modularity programu. Výhodou tohto princípu v rámci tejto aplikácie je odstránenie povinnosti inicializácie objektov a ich životných cyklov a sústredenie sa na implementáciu iba potrebnej logiky.

5.2.5 MVVM

Model (*model*), pohľad (*view*) a pohľadový model (*viewmodel*) je ďalší vzor softvérovej architektúry. Rozdeľuje komponenty grafického rozhrania aplikácie od jej dátového modelu. Túto vlastnosť plní samozrejme aj MVC model, avšak niekedy je nevyhnutné použitie istej logiky aj v prezentačnej vrstve, teda v pohľadoch. Sú to tie časti systému, ktoré vidí užívateľ. MVVM slúži aj na to, aby sme dáta v prezentačnej vrstve správne zobrazili. Rozdiel MVVM oproti MVC spočíva v tom, že na implementáciu zobrazenia dát sa môže použiť dokonca aj iný jazyk než ten, v ktorom je aplikácia napísaná. Transformuje získané modely aplikácie na modely plniace prezentačný účel. Definícia modelu a pohľadu zostáva rovnaká ako v MVC. Pohľadový model je viac modelom ako pohľadom. Tak ako v MVC sa logika nad modelom aplikácie vykonáva v službách a prípadne kontroléri, tak tu sa logika vykonáva v pohľadovom modeli. Logika sa však nevykonáva nad dátami aplikácie, ale nad pohľadom a dátami v ňom. Poskytuje to tak vytváranie prezentačnej vrstvy bez miešania kódu s aplikačnou logikou.



Obr. 5.3: MVVM model

5.2.6 Knockout

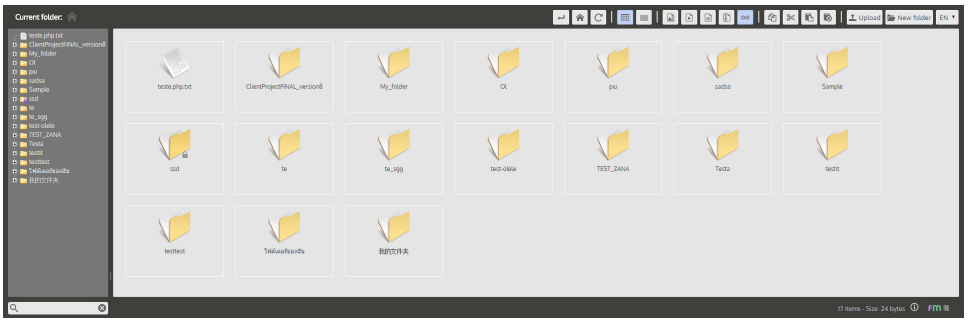
Tak ako Spring je implementáciou MVC modelu softvérovej architektúry, tak aj Knockout [46] je implementáciou MVVM dizajnovej architektúry. Je to samostatná knižnica napísaná v jazyku JavaScript. Rozdeľuje doménové dáta, pohľady a informácie, ktoré majú byť zobrazené užívateľovi a definuje vzťahy medzi komponentami. Zjednodušuje zobrazovanie komplexných dát pomocou jednoduchých príkazov.

5.2.7 RichFileManager

Rich Filemanager [47] je *open-source* projekt dostupný pod licenciou MIT, ktorý bol vytvorený na univerzite MIT. Jedná sa o implementáciu správcu súborov. Či už sú to súbory lokálne na disku alebo serveri, správca umožňuje jednoduchú prácu s týmito súbormi a zložkami. A práve na intuitívnu a jednoduchú prácu sa kladie dôraz, či už v tomto, alebo aj iných projektoch. Prezentačná vrstva tejto knižnice je implementovaná práve pomocou jazyka Knockout.

Dôvodom použitia tohto správcu bolo, že manipulácia so súborovým systémom bola jedna z požiadaviek užívateľov, preto bolo nutné implementovať korektné metódy zodpovedné za túto manipuláciu.

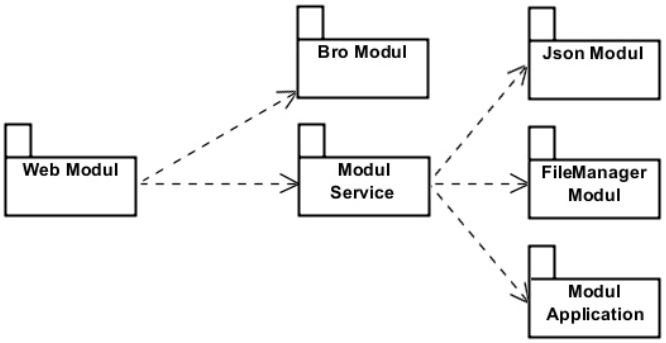
Istá funkcionalita bola prispôsobená na účely tohto projektu. V originálnej verzii pluginu je možné prechádzať súborovým systémom, vytvárať, editovať a mazať súbory. Všetko je intuitívne a konfigurovateľné. Úlohou však bolo obmedziť isté právomoci a pridať možnosť filtrovania a označenia súborov alebo zložiek, s ktorými chce užívateľ následne pracovať.



Obr. 5.4: Pôvodná verzia pluginu RichFileManager

5.2.8 Maven a moduly

Na kompiláciu zdrojového kódu a vytvorenie základnej súborovej štruktúry komponenty bol použitý nástroj Maven [48]. Za účelom vytvorenia modulárneho systému bol jeden z krokov aj rozdelenie zodpovedností a úloh celej aplikácie do modulov bez toho, aby vznikali komplikované závislosti medzi nimi. Schopnosť usporiadania modulov umožňuje vytvoriť prehľad o funkcionalite programu a pomáha vykonávať ľahké výmeny modulov v prípade potreby.



Obr. 5.5: Závislosť modulov

Koncové moduly poskytujú metódy plniace jednoduché úlohy. Ako názvy naznačujú, každý modul je zodpovedný iba za malú časť celej aplikácie. Modul *Application* v sebe zahŕňa príkazy na komunikáciu s operačným systémom. *Bro* modul poskytuje funkcie na používanie nástroja Bro. Podobný význam má modul *Json* na prácu so súbormi typu JSON. Modul *Filemanager* patrí do RichFileManager nadstavby a poskytuje funkcie na manipuláciu súborov a súborovým systémom.

Aplikačná logika kombinujúca funkcie týchto koncových modulov používa *Service* modul. Posledný *Web* modul v sebe zahŕňa kontrolné objekty zodpovedné za transformáciu požiadaviek z pohľadov, reakciu na udalosti, manipuláciu s pohľadmi.

5.2.9 WAR Archív

Okrem vytvorenia modulárnej štruktúry projektu Maven poskytuje možnosť zo zdrojových súborov vytvoriť aj kolekciu obsahujúcu skompilované zdrojové kódy programu na účely distribúcie a nasadenia programu do pracovného prostredia. Existujú mnohé typy používané na rôzne účely. Konkrétne to môžu byť napríklad JAR (Java Archive), EAR (Enterprise Archive) a WAR (Web Archive). V štruktúre sa tieto typy nerozlišujú. Všetko sú to archívy vytvorené pomocou *zip* alebo *jar* kompresie. Ich význam na sa druhú stranu líši v tom, že sa používajú v rôznych oblastiach.

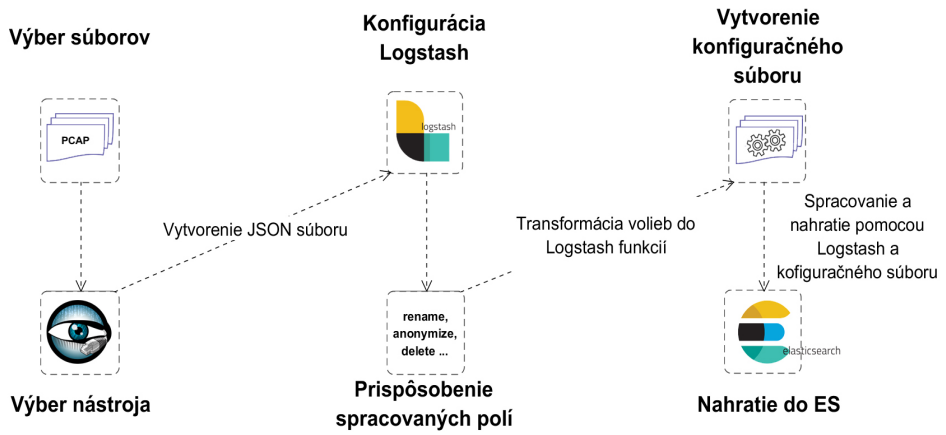
JAR archívy slúžia na to, aby v sebe držali knižnice, triedy a zdrojové súbory. Archív typu WAR je určený na to, aby obsahoval aj kompletnú zbierku súborov tvoriacich webovú aplikáciu teda HTML prípadne JSP stránky, súbory štýlov, skriptov a servlety. Tento typ bol zvolený aj v tejto práci z dôvodu, že v kolekcií predinštalovaných balíkov aplikácie CopAS sa nachádza Tomcat (program, umožňujúci vytvorenie servera). EAR archívy sú súčasťou podnikových aplikácií (*enterprise applications*). Sú definované ako kolekcie JAR a WAR archívov, ktoré môžu byť nasadené na servroch a v podnikových aplikáciách.

6 Implementácia požiadaviek

Po príprave návrhov nasleduje ich realizácia. Riešenie problému ako jedného celku môže byť náročné, preto je vhodné si takýto problém rozdeliť na menšie podproblémy. To isté platí aj pri implementácií funkcií. Ťažko sa pridávajú nové riešenia veľkých návrhov. Preto je vhodné návrhy rozdeliť na menšie podúlohy.

Proces používania aplikácie pozostáva z výberu pracovných dát, ich spracovania a nahratia do ES, teda pomerne veľkých a abstraktných krokov.

Ako pomoc pri vizualizácií týchto krokov slúži aj nasledujúci obrázok znázorňujúci základný pracovný postup používania aplikácie.



Obr. 6.1: Proces základnej práce

6.1 Nastavenie a spustenie

Pred tým, než je možné program používať je najprv potrebné ho spustiť, prípadne správne nastaviť. Ak je správne nastavené pracovné prostredie kontajnera je možné očakávať, že všetky programy a sever sú pripravené na prácu. Prvým krokom k spusteniu je potrebné vytvoriť balík webového archívu aplikácie.

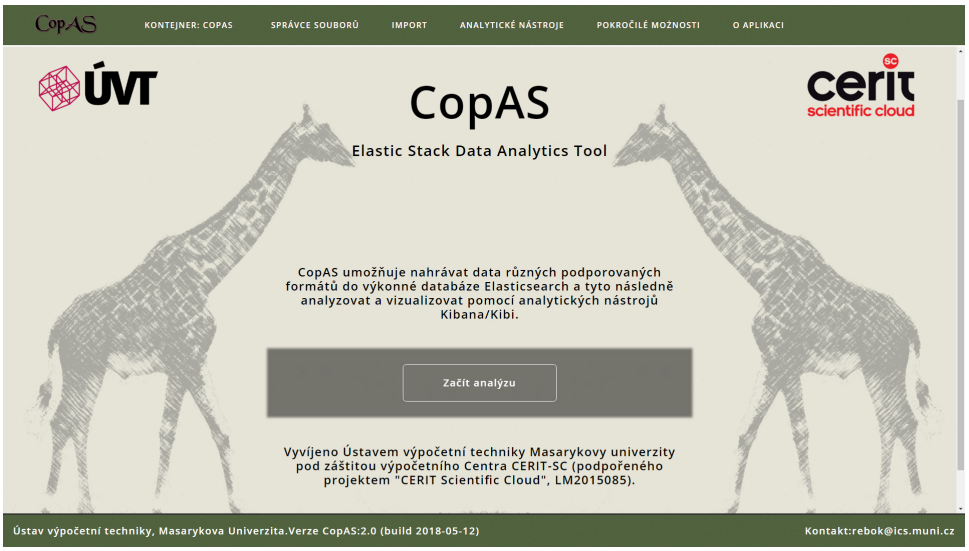
Nasledujúci Maven príkaz, vykonaný v zložke, kde sa projekt nachádza, vytvorí takýto archív v module *Web* s názvom *ROOT.war*.

```
> mvn clean package
```

Súbor je potrebné skopírovať a vložiť do zložky s názvom *webapps*, odkiaľ bude môcť Tomcat tento archív spustiť. Tá sa nachádza na mieste, kde bol nainštalovaný Tomcat.

Pri prvom spustení aplikácie sa vytvorí konfiguračný súbor s názvom *custom.properties*. Ten v sebe drží kľúče a ich hodnoty, ktoré aplikácia používa na načítanie údajov a ciest k súborom potrebným na korektný beh aplikácie. Tieto hodnoty je možné modifikovať. Po modifikácií je potreba zopakovať proces nasadenia archívu na server.

Prístup k aplikácií je možný pomocou webového prehliadača po zadaní adresy, ktorá je závislá na hodnote v konfiguračnom súbore s nastaveniami. Predbežne je premenná nastavená na hodnotu 8080, takže po vstupe na adresu *http://localhost:8080/* je možné začať používať aplikáciu.



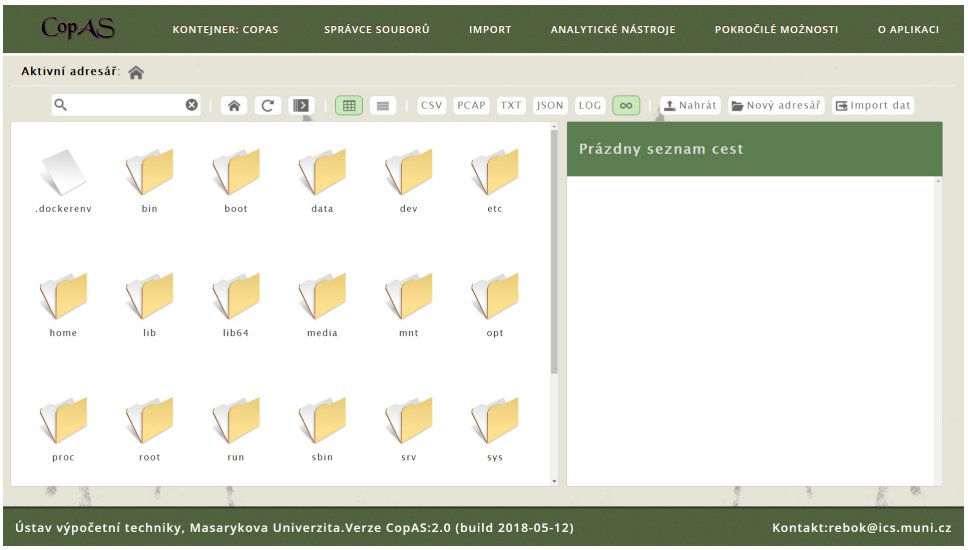
Obr. 6.2: Úvodná stránka

6.2 Výber súborov

Na to aby bolo možné vybrať vstupné dáta na prácu je potrebná interakcia so súborovým systémom. Modifikáciou originálnej verzie

nadstavby RichFileManager a jej integrácií do projektu spolu s pomocou Spring-u bolo dosiahnuté splnenie niekoľkých požiadaviek týkajúcich sa:

- manipulácie s uloženými dátami na serveri a so súborovým systémom (vyváranie, vymazávanie, kopírovanie, filtrovanie, ...),
- eliminácie obmedzenia veľkosti súborov nahrávania pomocou správnej definície Spring servletu,
- zobrazenia a prehľad o súboroch a ich detailných informáciách,
- určenia priečinkov určených na aktívne sledovanie zmien,
- definovania základnej grafickej témy rozhrania,
- lokalizácia preyentačnej vstvy,
- výber dát na spracovanie.



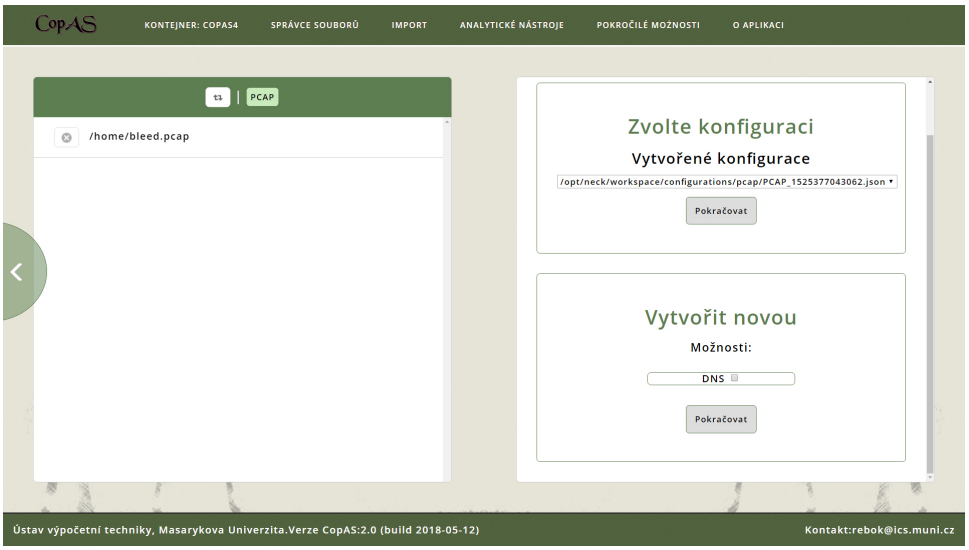
Obr. 6.3: Stránka manipulácie so súborovým systémom

Po selekcií vstupných súborov, prípadne zvolení zložiek, ktoré majú byť aktívne sledované, je možné prejsť ďalej.

6.3 Výber nástroja

V ďalšom kroku je predstavená obrazovka s podrobným zoznamom vybraných súborov, roztriedených podľa typov. Je potrebné vybrať typ vstupu (PCAP), s ktorým sa bude ďalej narábať. K tomuto typu je priradený jeden pomocný nastroj určený na jeho transformáciu na JSON (BRO). Je tu poskytnutá možnosť vytvorenia novej pracovnej konfigurácie alebo použitia už existujúcej v prípade, že užívateľ chce zopakovať presný postup len s inými súbormi.

Konfigurácia je v tomto prípade súbor vo formáte JSON, ktorý táto aplikácia generuje ako výstup pri úspešnom ukončení práce. Obsah tohto súboru je popísaný v kapitole 6.5. Spolu s možnosťou vytvorenia je konkrétne pre Bro pridaná možnosť zvoliť pole s názvom DNS. Je to príkaz pre Logstash, ktorý bude použitý v nasledujúcom kroku. Znamená, že polia s doménovými názvami webových adries zo vstupov sa budú prekladať na ich IP adresy.



Obr. 6.4: Výber nástroja

Zatiaľ je viac ako dostatočné spracovávať iba súbory typu PCAP. Avšak v prípade, že bude potrebné používať aj vstupy iného formátu, je aplikácia navrhnutá na jeho jednoduchú integráciu, prispôsobenie nastavenia a pridanie pohľadu podľa potreby.

6.3.1 Integrácia nového nástroja

V situácií, že by nastala zmena alebo pridanie nového nástroja, bolo dizajnovým zámerom čo najrýchlejšie implementovať túto zmenu. Pred integráciou je dôležité splniť istú množinu prerekvizít. Nasledujúce kroky sú vysvetlené na príklade pridávania Bro modulu. Prerekvizitami na integráciu sú:

- schopnosť nástroja spracovať požadovaný typ súboru na JSON (z PCAP na JSON),
- inštalácia nástroja do prostredia (binárny súbor Bro).

Integrácia nového nástroja spočíva v nasledujúcich krokoch:

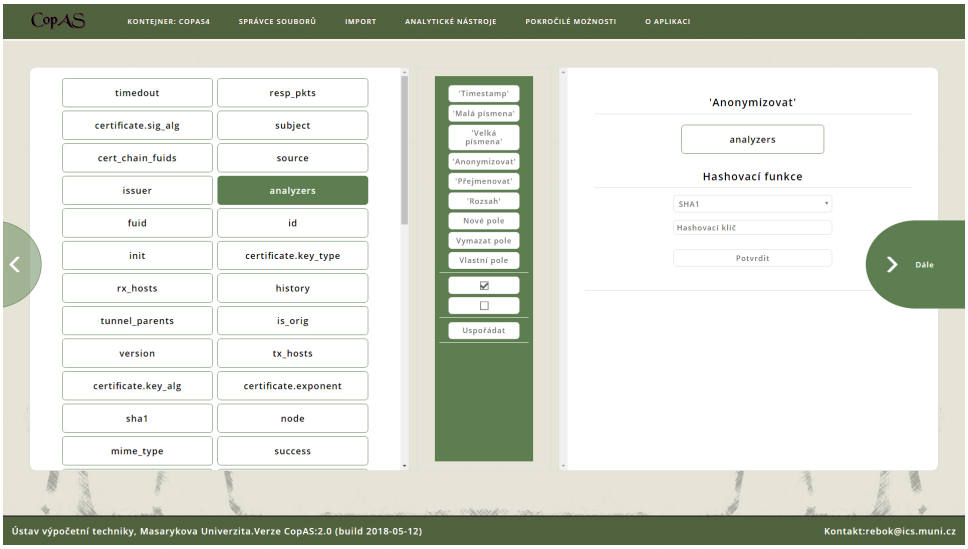
1. Pridanie modulu s implementačnou logikou, plniacou účel komunikácie s nástrojom, do projektovej štruktúry projektu (Modul Bro).
2. Vytvorenie triedy plniacej účel Spring kontroléra v module Web (*PcapController.java*).
3. Potreba zdediť v triede funkcie abstraktnej triedy *FileTypeController.java* (*PcapController.java* rozširuje triedu *FileTypeControler.java*).
4. Vytvorenie JSP stránky s logikou prezentačnej vrstvy v module *Web* (*pcapConfgiuration.jsp*).

Zdedenie od rodičovskej triedy *FileTypeController.java* spôsobí, že je potrebné implementovať funkcie, ktoré používa zvyšok aplikácie pri jej chode. Hlavnými metódami sú *processPaths()* na spracovanie ciest súborov, *getProfileType()* na definovanie typu súborov, s ktorým sa pracuje a *getProcessOutputPath()* s cestou k výsledku spracovania. V prípade Bro, je to zložka obsahujúca výstup spracovania v podobe JSON súborov. JSP stránka slúži na to, aby tieto kroky mohol užívateľ uskutočniť v rozhraní a pokračovať ďalej v práci.

6.4 Konfigurácia transformačných polí pre Logstash

Na ďalšej je obrazovke predstavená možnosť transformácie polí, ktoré užívateľ potrebuje pri analýze. V zozname na ľavej strane obrazovky sú

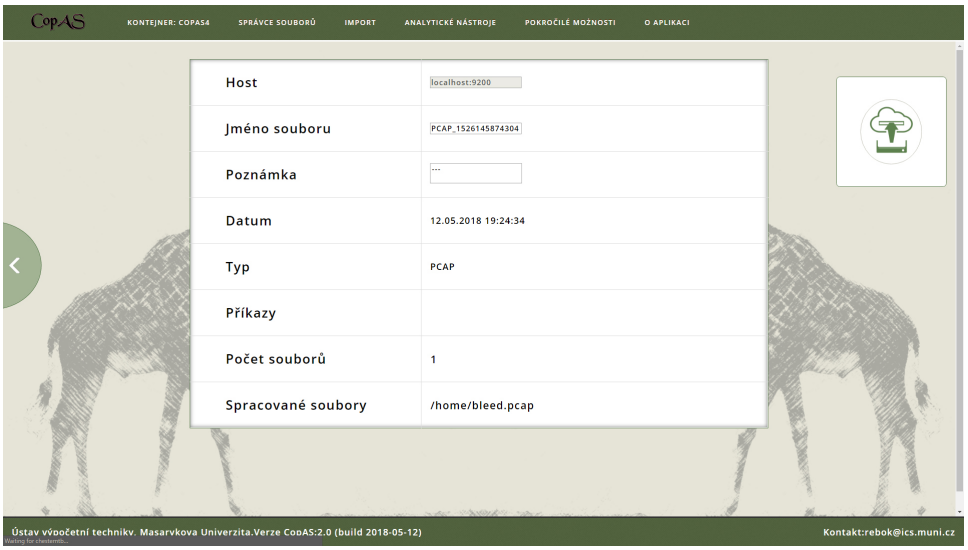
názvy polí, ktoré sa podarilo spracovať pomocou pomocného nástroja z výsledného JSON súboru. Na pravej strane sa zobrazujú formuláre podľa toho, aká Logstash transformačná funkcia sa zvolila v paneli v strede obrazovky. Tieto transformácie sa pridajú do výsledného konfiguračného súboru Logstash, ktorý sa použije pri nahrávaní do ES.



Obr. 6.5: Výber transformačných funkcií Logstash

6.5 Nahratie do ES

Posledným krokom je skontrolovanie všetkých akcií v záverečnej obrazovke procesu a kliknutí na tlačidlo nahratia.



Obr. 6.6: Nahratie do ES

Po úspešnom dokončení nahrávania je vygenerovaný výstupný súbor webovej aplikácie vo formáte JSON obsahujúci detaily o vykonaných akciách ako napríklad názov, dátum vytvorenia, cesty k vstupným súborom, zoznam transformačných funkcií, zoznam zložiek na sledovanie a poznámka. Ten je možné znova použiť pri ďalších dátach rovnakého typu (PCAP).

6.6 Dodatočná implementácia

Okrem implementácie základných funkcií existuje aj možnosti kontroly stavu aplikácie, ktorá informuje o tom, či všetky nástroje sú k dispozícii, vyčistenia pracovného priestoru alebo len vymazania aktuálnych vstupov v prípade chyby pri postupe.

V situácií, že je potrebná zmena alebo aktualizácia akéhokoľvek z predinštalovaných programov stačí pozmeniť obsah súboru *custom.properties* obsahujúceho definície ciest k používaným programom.

6.6.1 Testy

Je potrebné sa uistiť, či vytvorený kód je korektný a spĺňuje požadovanú funkcionalitu. V rámci tejto práce boli vyvinuté jednotkové testy.

Testy sú dôležitou súčasťou softvéru dopomáhajúce k pochopeniu fungovania zdrojových kódov a korektnosti kódu.

Na spustenie testov pomocou Mavenu slúži nasledujúci príkaz:

```
> mvn clean test
```

Pokiaľ tento príkaz bude spustený z príkazového riadku v zložke projektu, spôsobí to spustenie všetkých vytvorených testov (200) zahrnutých v moduloch.

6.6.2 Dokumentácia

Okrem testov bola pridaná dokumentácia s podrobnými informáciami a kontraktmi vytvárajúcimi lepší prehľad o aplikácií. Zvyšuje to tak rýchlosť porozumenia kódu, popisuje účel tried a určuje pravidlá prostredníctvom kontraktov v prípade ďalšieho vývoja. Do projektu bol pridaný Maven *plugin* umožňujúci generáciu dokumentácie pre Java kód s názvom Javadoc. Okrem nej je možné vygenerovať zo zdrojových súborov aj prehľad vo forme webových stránok, ktoré poskytujú informácie o verziách, moduloch, závislostiach projektu a licencií spolu s dokumentáciou Javadoc.

Samotný Javadoc je možné vygenerovať pomocou prvého príkazu, prehľadové stránky o projekte zase pomocou príkazu na druhom riadku:

```
> mvn javadoc:javadoc
> mvn site
```

Vykonaním týchto príkazov v zložke projektu sa vytvoria všetky potrebné súbory použité v stránkach HTML. Cesta k dokumentácií Javadoc je uvedená v prvom riadku a cesta k stránke HTML s informáciami o celom projekte na riadku treťom:

```
${CESTA_K_PROJEKTU}/Neck-${NAZOV_MODULU}/target/site
/apidocs/index.html
${CESTA_K_PROJEKTU}/target/site/index.html
```

7 Záver

Takýto návrh a implementácia už so sebou prináša zopár vlastností softvérovej kvality. Pri úspešnom užívateľskom testovaní, hlavne sieťovými analytikmi, bude rozhodnuté o tom, či nové vlastnosti, ktoré sa nenachádzali v prototype sú poskytnuté v novej verzií s tým, že zvyšné funkcie stále plnia základný návrh a spôsob práce s dátami.

Z pohľadu vývojárov tak vzniká produkt, ktorý je ľahko udržiavateľný a čitateľný. Zároveň však už aj spĺňa kvalitu, ktorá uľahčí ďalšiu manipuláciu s týmto softvérom. Pridávanie nových funkcionalít bude jednoduchšie a bez kritických následkov.

Popísaná komponenta slúži ako rozhranie medzi užívateľom a systémom CopAS. Prototypová verzia spĺňala účel rozhrania ale zároveň predstavila obmedzenia a nové požiadavky. Tie boli vyriešené v novej verzií teoretickou formou, návrhom a uvedením použitých techník a pomocných nadstavieb. Implementačné detaily zase popisujú to, ako vyzerá aplikácia napríklad MVC vzoru, pomocného pluginu RichFileManager alebo integrácia nových nástrojov. Kvalita tejto verzie sa zvyšuje s využitím objektovo orientovaných princípov a techník písania kvalitného kódu, pridaním testov a dokumentácie.

7.1 Možnosť rozšírenia

Pri vývoji nových návrhov, ktoré by aplikácia mohla mať, sa určite nekladú žiadne medze. Niektoré z návrhov boli brané v úvahu už pri vytváraní aktuálnej verzie, ako napríklad lokalizácia do viacerých jazykov. Webová aplikácia poskytuje možnosť práce v anglickom a českom jazyku. Tie však ešte nie je možné prepínať počas behu aplikácie. To je priestor pre vznik prvého vylepšenia.

Responzivita webového rozhrania je tiež vlastnosť, ktorá by v dnešnej dobe pokročilých mobilných technológií s rôznymi veľkosťami obrazoviek pozitívne vylepšila pohľady z prezentačnej vrstvy aplikácie.

V kontexte tejto aplikácie môže byť vhodné pridať aj možnosť vytvorenia pracovných účtov, do ktorých by sa užívatelia prihlasovali a mohli tak zdieľať svoje pracovné prostredia s prípadnými právomocami.

Ako bolo popísané v kapitole 6.3.1 okrem nástroja Bro je možné pridať akékoľvek pomocné programy, ktoré spĺňajú uvedené prerekvizity. Tak môže byť aktuálna analýza, ktorá je schopná spracovať len súbory typu PCAP, obohatená o schopnosť spracovať aj súbory typu CSV, TXT, JPEG a iných. Analýza obrázkov zachytených pri sieťovom prenose je tiež pozitívne hodnotenou vlastnosťou pri získavaní dôležitých údajov zo zdrojov v takýchto formátoch.

Logstash umožňuje transformovať dáta pomocou mnohých funkcií. V aplikácií boli zahrnuté iba základné z nich ako rename, delete, add, anonymize a iné. Ako ďalšie vylepšenie by bolo vhodné pridať ešte viac príkazov, ktoré Logstash poskytuje, alebo dokonca implementovať API pre Logstash.

Prijateľné môže byť aj pridať alebo prispôsobiť farebné témy rozhrania, investovať do testovania a celkovo tak zvýšiť kvalitu alebo bezpečnosť softvéru. Možností je mnoho. Záleží však iba na tom, čo bude potrebné a zrovna požadované.

7.2 Rozšírenie do iných oblastí

Okrem interných vylepšení je však potrebné zmieniť možnosť expanzie aj do iných oblastí. V prípade, že sa do aplikácie a systému budú pridávať nástroje schopné spracovať aj iné formáty dát, je možné aplikáciu používať kýmkoľvek, kto pracuje s veľkým množstvom informácií potrebných na spracovanie a získavanie poznatkov alebo aj na školské účely. Príkladom sú oblasti biológie alebo chémie. Údaje uložené napríklad vo formátoch CSV, ktoré obsahujú tabuľky s tisíckami riadkov a stĺpcov, v ktorých sa dá ťažko orientovať v ich originálnej forme. Vlastnosť expanzie do iných oblastí sa považuje za najhodnotnejší potenciál tejto práce a celého projektu.

Bibliografia

1. LISÝ, Martin. *Nástroj na automatizaci práce se vstupními daty pro Elasticsearch*. 2016. Dostupné tiež z: http://is.muni.cz/th/396280/fi_m/. Diplomová práce. Masarykova univerzita, Fakulta informatiky, Brno. Vedúci práce Tomáš PITNER.
2. MARZ, Nathan; WARREN, James. *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*. 1st. Greenwich, CT, USA: Manning Publications Co., 2015. ISBN 9781617290343.
3. LOSHIN, David. *Big Data Analytics: From Strategic Planning to Enterprise Integration with Tools, Techniques, NoSQL, and Graph*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2013. ISBN 9780124186644.
4. LANEY, Douglas. *3D Data Management: Controlling Data Volume, Velocity, and Variety*. 2001. Dostupné tiež z: http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf. Technická správa. META Group.
5. GOES, Paulo B. Design science research in top information systems journals. *MIS Quarterly: Management Information Systems*. 2014, roč. 38, č. 1, s. iii–viii. ISSN 0276-7783.
6. MARR, Bernard. *The 5 key V's of Big Data*, 2014. Dostupné tiež z: https://www.slideshare.net/BernardMarr/140228-big-data-volume-velocity-variety-varacity-value.
7. O'REILLY MEDIA, I. *Big Data Now: 2012 Edition*. O'Reilly Media, 2012. ISBN 9781449356682. Dostupné tiež z: https://books.google.sk/books?id=DhATUBDFyI0C.
8. JACOBSON, Van; LERES, Craig; MCCANNE, Steven. *Tcpdump/Libpcap [software]*. Dostupné tiež z: http://www.tcpdump.org/. [prístup 18.05.2018].
9. COMBS, Gerald. *Wireshark [software]*. 1998. Dostupné tiež z: https://www.wireshark.org/. [prístup 18.05.2018].

10. MAY, J.; PETERSON, J.; BAUMAN, J. Attack detection in large networks. In: *DARPA Information Survivability Conference amp; Exposition II, 2001. DISCEX '01. Proceedings*. 2001, zv. 1, 15–21 vol.1. Dostupné z DOI: 10.1109/DISCEX.2001.932188.

11. ROESCH, Martin. *Snort [software]*. 1998. Dostupné tiež z: https://www.snort.org/. [prístup 18.05.2018].

12. OISF, Open Information Security Foundation. *Surikata [software]*. 2009. Dostupné tiež z: https://suricata-ids.org/. [prístup 18.05.2018].

13. CASAS, Pedro; SORO, Francesca; VANERIO, Juan Martin; SETTANNI, Giuseppe; D'ALCONZO, Alessandro. Network security and anomaly detection with Big-DAMA, a big data analytics framework. *2017 IEEE 6th International Conference on Cloud Networking (CloudNet)*. 2017, s. 1–7.

14. JIRSÍK, Tomáš; ČERMÁK, Milan; TOVARŇÁK, Daniel; PAULOVIČ, Jakub Samuel; ŠTEFÁNIK, Michal. *Stream4Flow: Software for mining and analysis of the large volumes of network traffic [software]*. GitHub, 2016. Dostupné tiež z: https://github.com/CSIRT-MU/Stream4Flow. [prístup 18.05.2018].

15. FLOWMON. *Flowmon [software]*. Dostupné tiež z: https://www.flowmon.com/. [prístup 18.05.2018].

16. ICINGATEAM; OPEN-SOURCE. *Icinga [software]*. 2009. Dostupné tiež z: https://www.icinga.com/. [prístup 18.05.2018].

17. POMEL, Olivier; LÊ-QUÔC, Alexis. *DataDog [software]*. 2009. Dostupné tiež z: https://www.datadoghq.com/. [prístup 18.05.2018].

18. SOLARWINDS. *Solarwinds Packer Capture [software]*. Dostupné tiež z: https://www.solarwinds.com/topics/packet-capture. [prístup 18.05.2018].

19. ELASTIC. *Elasticsearch [software]*. 2010. Dostupné tiež z: https://www.elastic.co/products/elasticsearch. [prístup 18.05.2018].

20. ELASTIC. *Logstash [software]*. 2009. Dostupné tiež z: https://www.elastic.co/products/logstash. [prístup 18.05.2018].

21. ELASTIC. *Kibana [software]*. 2013. Dostupné tiež z: https://www.elastic.co/products/kibana. [prístup 18.05.2018].

22. ELASTIC. *Inverted index*. Dostupné tiež z: https://www.elastic.co/guide/en/elasticsearch/guide/current/inverted-index.html. [prístup 18.05.2018].
23. ELASTIC. *Beats [software]*. Dostupné tiež z: https://www.elastic.co/products/beats. [prístup 18.05.2018].
24. ELASTIC. *X-Pack [software]*. Dostupné tiež z: https://www.elastic.co/products/x-pack. [prístup 18.05.2018].
25. ELASTIC. *ECE [software]*. 2000. Dostupné tiež z: https://www.elastic.co/products/ece. [prístup 18.05.2018].
26. TAYLOR, Austin. *VulnWhisperer [software]*. GitHub, 2017. Dostupné tiež z: https://github.com/austin-taylor/VulnWhisperer. [prístup 18.05.2018].
27. KALIAPPAN, Sivasamy. *LogTrail [software]*. GitHub, 2016. Dostupné tiež z: https://github.com/sivasamyk/logtrail. [prístup 18.05.2018].
28. ELASTIC. *TimeLion [software]*. GitHub, 2015. Dostupné tiež z: https://github.com/elastic/timelion. [prístup 18.05.2018].
29. SIREN. *Siren Investigate [software]*. GitHub, 2013. Dostupné tiež z: https://github.com/sirensolutions/kibi. [prístup 18.05.2018].
30. VLČEK, Lukáš. *Bigdesk [software]*. GitHub, 2011. Dostupné tiež z: https://github.com/lukas-vlcek/bigdesk. [prístup 18.05.2018].
31. GUARD, Search. *Search Guard [software]*. GitHub, 2015. Dostupné tiež z: https://github.com/floragunncom/search-guard. [prístup 18.05.2018].
32. MENEZES, Leonardo. *Cerebro [software]*. GitHub, 2016. Dostupné tiež z: https://github.com/lmenezes/cerebro. [prístup 18.05.2018].
33. ELASTICHQ. *ElasticHQ [software]*. GitHub, 2013. Dostupné tiež z: https://github.com/ElasticHQ/elasticsearch-HQ. [prístup 18.05.2018].
34. SAVENKO, Yury. *LogIQ [software]*. GitHub, 2017. Dostupné tiež z: https://github.com/sloniki/logiq. [prístup 18.05.2018].
35. PACKETZOOM. *LogZoom [software]*. GitHub, 2015. Dostupné tiež z: https://github.com/packetzoom/logzoom. [prístup 18.05.2018].
36. ELASTIC. *Filebeat [software]*. Dostupné tiež z: https://www.elastic.co/products/beats/filebeat. [prístup 18.05.2018].

37. FOUNDATION, Apache Software. *Kafka [software]*. 2011. Dostupné tiež z: https://kafka.apache.org/. [prístup 18.05.2018].

38. ZAHARIA, Matei. *Apache Spark [software]*. 2012. Dostupné tiež z: https://spark.apache.org/. [prístup 18.05.2018].

39. GRAYLOG. *Graylog [software]*. 2010. Dostupné tiež z: https://www.graylog.org/. [prístup 18.05.2018].

40. HYKES, Solomon; DOCKER INC. *Docker [software]*. 2013. Dostupné tiež z: https://www.docker.com/. [prístup 18.05.2018].

41. PAXSON, Vern. *Bro network and security monitor [software]*. 2010. Dostupné tiež z: https://www.bro.org/index.html. [prístup 18.05.2018].

42. FOUNDATION, Apache Software. *Tomcat [software]*. 1999. Dostupné tiež z: http://tomcat.apache.org/. [prístup 18.05.2018].

43. MARTIN, Robert C. *Clean Code: A Handbook of Agile Software Craftsmanship*. 1. vyd. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2008. ISBN 9780132350884.

44. SOFTWARE, Pivotal. *Spring [software]*. 2002. Dostupné tiež z: https://spring.io/. [prístup 18.05.2018].

45. SOFTWARE, Pivotal. *Spring Web MVC Documentation*. Dostupné tiež z: https://docs.spring.io/spring/docs/. [prístup 18.05.2018].

46. SANDERSON, Steve. *Knockout [software]*. 2010. Dostupné tiež z: http://knockoutjs.com/. [prístup 18.05.2018].

47. SOLOMIENKO, Pavel. *RichFileManager [software]*. GitHub, 2010. Dostupné tiež z: https://github.com/servocoder/RichFilemanager. [prístup 18.05.2018].

48. FOUNDATION, Apache Software. *Maven [software]*. 2004. Dostupné tiež z: https://maven.apache.org/. [prístup 18.05.2018].