



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ

DEPARTMENT OF INFORMATION SYSTEMS

**WEBOVÁ APLIKÁCIA NA VZDIALENÚ SPRÁVU SYS-
TÉMU FITCRACK**

WEB APPLICATION FOR REMOTE ADMINISTRATION OF FITCRACK SYSTEM

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE

AUTHOR

MATÚŠ MÚČKA

VEDOUCÍ PRÁCE

SUPERVISOR

Ing. RADEK HRANICKÝ

BRNO 2018

Abstrakt

Táto práca rieši vzdialenú komunikáciu so systémom na distribuovanú obnovu hesiel - Fitcrack. Zameram sa na vylepšenie súčasného riešenia. Okrem zvýšenia bezpečnosti a rýchlejšej odozvy, návrh riešenia uvedený v tejto práci ponúka aj podporu autentifikácie užívateľov a ich oprávnení. Systém bude automaticky generovať dokumentáciu pri zmene zdrojových kódov. V riešení bola použitá architektúra REST (viď 3.2). Táto architektúra umožňuje oddeliť aplikačnú logiku od databázového systému, a tým pádom umožniť komunikáciu s aplikáciami pomocou zasielanie správ. Vďaka tomu je možné systém Fitcrack ovládať z rôznych prostredí pomocou jednoduchých dotazov protokolu HTTP. V mojej práci som navrhol systém, ktorý zrýchli komunikáciu so systémom Fitcrack, zlepší zabezpečenie a pridá do systému vylepšenia ako podpora viacerých užívateľov a ich oprávnení.

Abstract

This bachelor's thesis resolves remote communication with distributed password recovery system - Fitcrack. I aimed to improve a current implemented solution. In addition to security improvement and faster responses, the design of solution presented in this work offers support for user authentication and authorization. The system automatically generate documentation when source code changes. REST (see 3.2) architecture was used in this solution. This architecture allows to separate the application logic and the database system which consequently allows application communication by sending messages. Because of that, it is possible to control Fitcrack from different environments using simple HTTP queries. I designed the system that accelerates communication with Fitcrack, refines security and adds improvements such as support for multiple user privileges.

Klíčové slová

REST, Fitcrack, API, backend

Keywords

REST, Fitcrack, API, backend

Citácia

MÚČKA, Matúš. *Webová aplikácia na vzdialenú správu systému Fitcrack*. Brno, 2018. Bakalárska práca. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Hranický

Webová aplikácia na vzdialenú správu systému Fitcrack

Prehlásenie

Prehlasujem, že tento semestrálny projekt som vypracoval samostatne pod vedením pána inžiniera Radka Hranického. Uviedol som všetky literárne zdroje a publikácie, z ktorých som čerpal.

.....

Matúš Múčka
28. apríla 2018

Podakovanie

Ďakujem Ing. Radkovi Hranickému za odbornú pomoc a vedenie pri vypracovávaní tejto práce.

Obsah

1	Úvod	3
2	Systém Fitcrack	4
2.1	Hashcat	4
2.2	BOINC	4
2.3	Architektúra systému Fitcrack	4
2.3.1	Webový server	5
2.3.2	Generátor	6
2.3.3	Asimilátor	6
2.3.4	Host	6
3	Popis použitých technológií	7
3.1	Hypertext Transfer Protocol (HTTP)	7
3.1.1	Princíp protokolu HTTP	7
3.1.2	Rozšírenie HTTPS	7
3.1.3	Stavové kódy protokolu HTTP	7
3.1.4	Druhy žiadostí/metódy HTTP	8
3.1.5	Príklad komunikácie	9
3.2	Architektúra REST	9
3.2.1	URI	9
3.2.2	Zásady architektúry REST	10
3.2.3	Metódy pre prístup ku zdrojom	11
4	Návrh systému	12
4.1	Návrh serverovej časti	12
4.1.1	Rozdelenie systému na moduly	12
4.1.2	Autentifikácia a oprávnenia užívateľov	15
4.2	Návrh klientskej časti	16
4.3	Typy útokov v systéme Fitcrack	17
4.3.1	Útok s maskami	17
4.3.2	Slovníkový útok	18
4.3.3	Kombinačný útok	18
4.3.4	Hybridné útoky	19
4.4	Grafy	19
4.5	Notifikácie	19
5	Implementácia	21
5.1	Implementácia databázovej vrstvy	22

5.2	Implementácia aplikačnej vrstvy	23
5.2.1	Implementácia jednotlivých URI	24
5.2.2	Spracovanie útokov	25
5.2.3	Autentizácia užívateľov	26
5.3	Implementácia prezentačnej vrstvy	26
5.3.1	Implementácia komponentov	27
5.3.2	Globálne dáta	29
5.3.3	Navigácia v aplikácii	29
5.3.4	Grafy	30
5.3.5	Prehľad úloh a stránkovanie	31
6	Testovanie a experimenty	33
6.1	Funkcionálne testovanie	33
6.2	Užívateľské testovanie	33
6.3	Experiment	34
7	Záver	37
	Literatúra	38

Kapitola 1

Úvod

V súčasnosti je používanie aplikačného protokolu HTTP pre sieťový prenos textových dát v rámci služby World Wide Web (WWW) veľmi rozšírené. Spoločne s elektronickou poštou je HTTP najviac používaným protokolom, ktorý sa zaslúžil o obrovský rozmach internetu v posledných rokoch. Jedným z autorov protokolu HTTP je Roy Fielding, ktorý vo svojej dizertačnej práci opisuje Representation State Transfer (REST). REST je architektúra, ktorá poskytuje obecné rozhranie pre vzdialené aplikácie, ktoré komunikujú cez sieť. Napriek tomu, že REST nie je určený priamo pre HTTP, je takmer vždy spojovaný s týmto protokolom.

V tejto práci popisujem všeobecný opis a implementáciu aplikačné rozhranie spĺňajúce zásady architektúry REST (viď 3.2.2) pre systém Fitcrack. Jedná sa o výkonný systém na obnovu hesiel, ktorý prerozdeľuje prácu medzi viacerými pripojenými klientmi, a tým pádom zvyšuje výpočetnú silu celého systému (detailnejšie je Fitcrack popísaný v kapitole 2). Konkrétne sa v mojej práci zaoberám jeho serverovou časťou, ktorá slúži na správu celého systému. Mojim cieľom je nahradiť súčasné nevyhovujúce riešenie administrácie systému, ktoré bolo navrhnuté len ako prototyp, novým riešením.

Súčasná implementácia je veľmi ťažko rozšíriteľná. Taktiež obsahuje niekoľko bezpečnostných dier. Nové riešenie prináša do systému Fitcrack väčšiu bezpečnosť a rozšíriteľnosť, umožňuje testovanie systému, a vďaka architektúre klient-server znižuje záťaž na serverovú časť systému.

Táto práca pozostáva z piatich kapitol. Popis systému Fitcrack spolu s nástrojmi, ktoré využíva, sa nachádza v kapitole 2. Opis využitých technológií, ktoré som pri návrhu nového systému na vzdialenú správu Fitcracku použil, sa nachádza v kapitole 3. Samotným návrhom riešenia, ktorý je rozdelený do modulov, sa zaoberám v kapitole 4. V jednotlivých podkapitolách popisujem každý modul. Nakoniec v závere hodnotím výsledky mojej práce.

Kapitola 2

System Fitcrack

Fitcrack je systém, ktorý slúži na obnovu hesiel. Vďaka tomu že ide o distribuovaný systém, je možné rozdeľovať prácu na predom neobmedzený počet staníc, ktoré môžu byť rozmiestnené po celom svete. Je tvorený architektúrou klient-server. Serverová časť sa stará o prerozdeľovanie úloh medzi klientov. Klienti pracujú na výpočte kryptografických hešov a svoje výsledky posielajú na server, kde sa spracujú a vyhodnotia. [5]

2.1 Hashcat

Hashcat¹ je výkonný nástroj na obnovu hesiel, ktorý využíva technológiu OpenCL. Podporuje viac ako 200 typov kryptografických hešov. O jeho rýchlosti svedčí aj to, že v rokoch 2010, 2012, 2014 a 2015, tím, pozostávajúci z členov vývojárov Hashcatu, získal prvé miesto v súťaži *Crack Me If you Can*². Žiaľ, tento nástroj sám o sebe nepodporuje výpočet na viacerých staniciach súčasne.

2.2 BOINC

Systém Fitcrack je postavený na voľne šíriteľnom frameworku *Berkeley Open Infrastructure for Network Computing* (BOINC)³. Vďaka frameworku BOINC systém Fitcrack distribuuje dielčie úlohy medzi pripojených klientov, na ktorých prebieha pokus o nájdenie hesla pomocou nástroja Hashcat.

Systém BOINC tvorí server a klienti. Pri distribuovaní výpočetných úloh prebieha komunikácia medzi klientom a serverom prostredníctvom XML správ, ktoré sa prenášajú pomocou protokolu HTTP alebo HTTPS. Server je hlavnou súčasťou infraštruktúry BOINC. Stará sa o distribuovanie úloh medzi klientov a spracováva výsledky od užívateľov. Každý klient sa periodicky dotazuje na server a žiada si novú úlohu. Po prijatí výpočetnej úlohy ju spracuje a výsledok odošle na serverom. Následne žiada o pridelenie novej úlohy.[1]

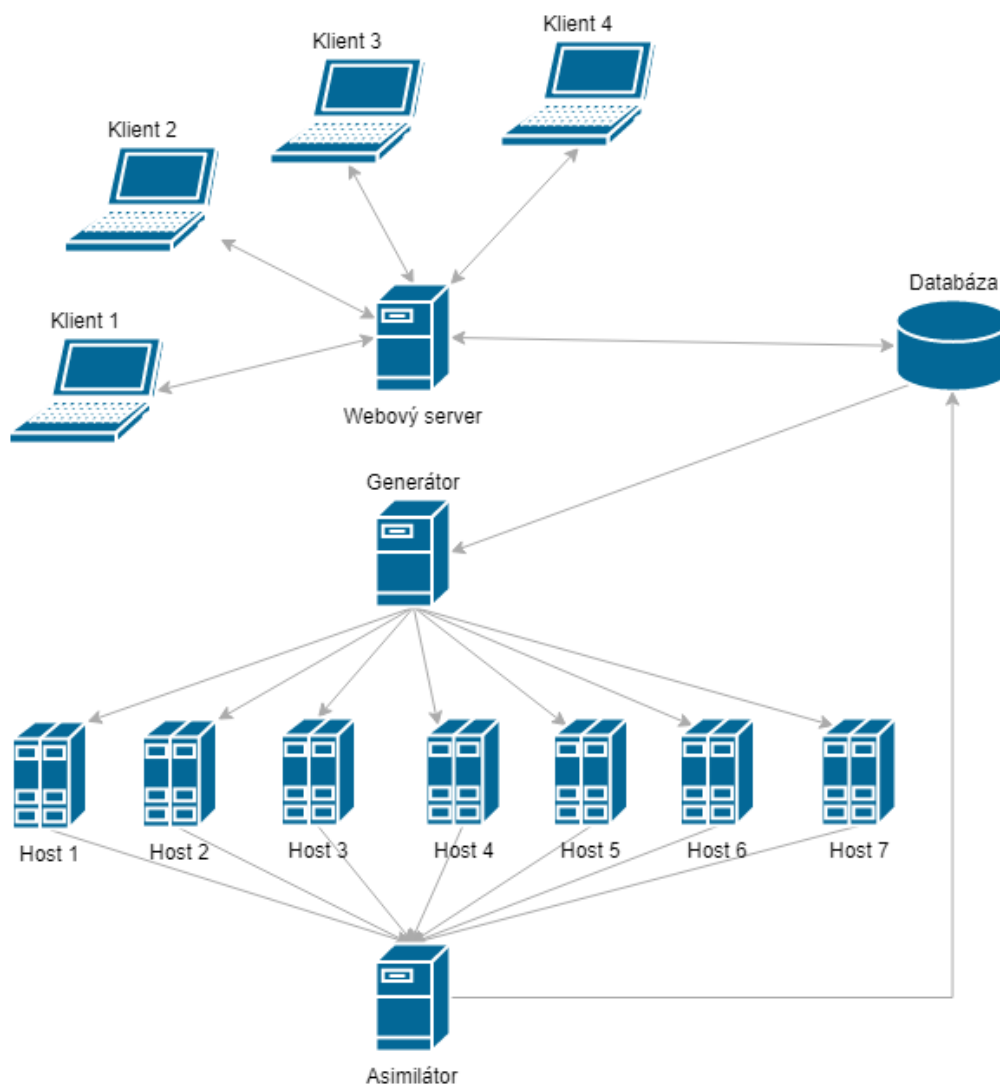
2.3 Architektúra systému Fitcrack

Ako je možné vidieť na obrázku 2.1, architektúra systému Fitcrack je rozdelená na niekoľko častí. Na obrázku je uvedená zjednodušená architektúra bez niektorých systémov BOINC.

¹<https://hashcat.net>

²http://contest-2010.korelogic.com/team_hashcat.html

³<https://boinc.berkeley.edu/>



Obr. 2.1: Architektúra systému Fitrack

2.3.1 Webový server

Cieľom mojej bakalárskej práce je navrhnúť a implementovať webový server, cez ktorý budú môcť užívatelia spravovať celý systém Fitrack. K webovému serveru môže byť naraz pripojených viacero užívateľov, ktorí vykonávajú rôzne činnosti (analyzovanie výsledkov dokončených úloh, sledovanie aktivity jednotlivých hostov, pridávanie nových úloh, sledovanie progresu práve prebiehajúcich úloh atď.). Po pripojení užívateľa na webový server cez internetový prehliadač, sa užívateľovi odošle webová aplikácia, ktorá komunikuje so serverom prostredníctvom REST API (viď 3.2). Prostredníctvom webovej aplikácie je možné do systému pridávať nové úlohy, monitorovať prebiehajúce, analyzovať dokončené úlohy, monitorovať správanie hostov a podobne. Viac o návrhu webového servera sa je možné dočítať v kapitole 4.

2.3.2 Generátor

Keď užívateľ prostredníctvom webovej aplikácie pridá do systému úlohu, generátor, v prípade že nastal čas zahájenia úlohy, začne generovať pracovné jednotky pre priradených hostov k úlohe. Jedná sa o program, ktorý beží v nekonečnom cykle. Generátor systému Fitcrack vytvára dva typy pracovných jednotiek.

- **Meranie výkonu** - tieto pracovné jednotky sú generované hosťom, ktorý sa práve do systému alebo k úlohe pripojili, poprípade hosťom, u ktorých nastala chyba. Slúžia k zmeraniu výkonu daného hosta. Meranie výkonu má veľký význam pri generovaní normálnych pracovných jednotiek, pretože vďaka nameraným údajom vieme odhadnúť správne množstvo práce pre hosta. Výsledkom tejto úlohy je počet vygenerovaných kryptografických šifier určitého formátu za jednu sekundu.
- **Normálne** - obsahujú potrebné informácie pre zvolený spôsob útoku. Môžu obsahovať slovník, masku, rozsah stavového priestoru, súbor s pravidlami, cudzojazyčné znakové sady a podobne. Počet hesiel na overenie v jednej pracovnej jednotke (náročnosť pracovnej jednotky) závisí na zložitosti výpočtu danej kryptografickej šifry a nameraného výkonu hosta. Cieľom je aby sa doba výpočtu jednej pracovnej jednotky čo najviac približovala času zadaného pri vytváraní úlohy. Generátor udržiava v databáze dve naplánované úlohy pre jedného hosta (pokiaľ sa už nevyčerpal celý stavový priestor hesiel). Prvá úloha sa práve počíta a druhá je pripravená k odoslaniu hneď po obdržaní výsledku prvej úlohy.

2.3.3 Asimilátor

Asimilátor slúži na spracovanie výsledkov od hostov. Beží v nekonečnej slučke. Na začiatku každého cyklu kontroluje prítomnosť ešte nespracovaných výsledkov a tieto výsledky overí. Správa pre asimilátor môže obsahovať rôzne informácie (nájdené heslo, správu o chybe, čas výpočtu, typ pracovnej jednotky, stavový kód výsledku a podobne). Podľa obsahu správy asimilátor upravuje databázu. Napríklad v prípade že host heslo našiel hľadané heslo, asimilátor na základe tejto odpovedi upraví databázu. Vďaka tejto úprave ostatní hostia podieľajúci sa na danej úlohe, dostanú správu o zastavení výpočtu.

2.3.4 Host

Host je jeden výpočtový uzol pripojený k systému Fitcrack. Na hostoch beží aplikácia Runner, ktorá prijíma úlohy od Generátora. Samotný výpočet potom riadi Runner a prebieha pomocou nástroja hashcat (viď 2.1).

Kapitola 3

Popis použitých technológií

3.1 Hypertext Transfer Protocol (HTTP)

Pôvodne bol protokol HTTP určený pre výmenu dokumentov vo formáte HTML, ale v súčasnosti sa používa aj pre prenos iných informácií. Vďaka rozšíreniu MIME (Multipurpose Internet Mail Extensions) je tento protokol schopný prenášať akýkoľvek súbor [4].

3.1.1 Princíp protokolu HTTP

Protokol HTTP funguje na princípe dotaz-odpoveď. Užívateľ (klient/user-agent) pošle serveru dotaz. Ten server spracuje a klientovi odpovie. V odpovedi server popisuje výsledok dotazu informáciami, či sa podarilo žiadaný zdroj nájsť, či zdroj existuje, v akom formáte je telo odpovede atď.

3.1.2 Rozšírenie HTTPS

Protokol HTTP neumožňuje zabezpečené spojenie, preto sa často používa protokol TLS (Transport Layer Security) nad vrstvou TCP. Vďaka tomu je možné vytvoriť šifrovaný kanál. Toto spojenie je označované ako HTTPS [6].

3.1.3 Stavové kódy protokolu HTTP

Úspešnosť dotazu vieme zistiť podľa stavových kódov, ktoré sú pribalené v odpovedi servera na dotaz klienta. Vďaka stavovým kódom vieme presne určiť, či behom spracovávania dotazu došlo k chybe. Tým pádom môže klient reagovať na chyby.

Zoznam stavových kódov má na starosti organizácia IANA (Internet Assigned Numbers Authority). Jedná sa o trojciferné číslo v desiatkovej sústave. Prvé číslo určuje kategóriu odpovede a ostatné ju bližšie špecifikujú.

1XX

Kódy, začínajúci číslom 1, sú tzv. informačné. Indikujú že server dotaz spracoval a pochopil. Bližší význam záleží na zvyšných dvoch číslach. V niektorých prípadoch môže klientovi naznačovať, že sa finálna odpoveď ešte spracováva a má na ňu počkať. Tiež môže naznačovať zmenu protokolu.

2XX

Stavové kódy začínajúce číslom 2 indikujú, že dotaz bol serverom obdržaný, pochopený a správne vyhodnotený.

3XX

Tieto kódy naznačujú, že na získanie požadovaného zdroja, je potrebné vykonať ďalšiu akciu. Zvyčajne sa jedná o presmerovanie.

4XX

Stavové kódy, ktoré začínajú číslom 4, indikujú, že nastala chyba na strane užívateľa. Ďalšie 2 čísla presnejšie určujú o akú chybu ide. Najčastejšie sa jedná o chybu **404 Not Found**, ktorá hovorí že žiadaný zdroj nebol na serveri nájdený, ale môže ísť aj o menej časté chyby ako **429 Too Many Requests**, ktorá sa vyskytuje v prípade, že užívateľ žiadal o daný zdroj príliš veľa krát v určitom časovom úseku.

5XX

Stavové kódy začínajúce číslom 5, hovoria o tom, že došlo k chybe na strane servera. Aj keď dotaz mohol byť validný, server ho nedokázal spracovať. Mohlo sa tak stať napríklad kvôli výpadku servera (preťaženie, údržba).

3.1.4 Druhy žiadostí/metódy HTTP

Protokol HTTP využíva niekoľko žiadostí, z ktorých najčastejšie sú:

- **GET** - ide o najbežnejší typ žiadostí. Jej výsledkom je žiadaný zdroj uvedení v dotaze URL. Tento typ dotazu neobsahuje telo správy.
- **POST** - k dotazu je pridané telo správy. Zvyčajne obsahuje hodnoty z HTML formulára.
- **PUT** - využíva sa na nahranie súboru na určitú URI.
- **DELETE** - tento typ žiadosti je len zriedka implementovaný. Zmaže zdroj uvedení v URI.
- **HEAD** - ide o podobný typ žiadosti ako GET, ale na rozdiel od GET, server vracia len hlavičku odpovede.
- **OPTIONS** - v odpovedi vracia metódy, ktoré sú povolené na danej URI.

3.1.5 Príklad komunikácie

Klient začína komunikáciu poslaním dotazu na server. Na výpise 3.1 sa nachádza ukážka dotazu, ktorý obsahuje zvyčajne viac informácií, ale pre zjednodušenie príkladu nie sú uvedené. Server v dotaze špecifikuje metódu žiadosti, svoju totožnosť (Opera verzia 10.60) a podporované kódovanie.

```
GET / HTTP/1.1
Host: www.fit.vutbr.cz
User-Agent: Opera/9.80 (Windows NT 5.1; U; sk) Presto/2.5.29 Version/10.60
Accept-Charset: UTF-8,*
```

Výpis 3.1: príklad HTTP dotazu

Príklad nasledovnej reakcie servera na dotaz sa nachádza na výpise 3.2. Server odpovedá stavovým kódom 200 OK, čo značí, že dotaz sa podarilo úspešne spracovať. Ďalej hlavička odpovedi okrem iného obsahuje dátum a čas vybavenia žiadosti, a informácie o vrátenom zdroji ako typ (text/HTML), použité kódovanie (UTF-8) a dĺžku odpovede.

```
HTTP/1.1 200 OK
Content-Length: 3059
Server: GWS/2.0
Date: Sat, 11 Jan 2003 02:44:04 GMT
Content-Type: text/html
Cache-control: private
Connection: keep-alive
```

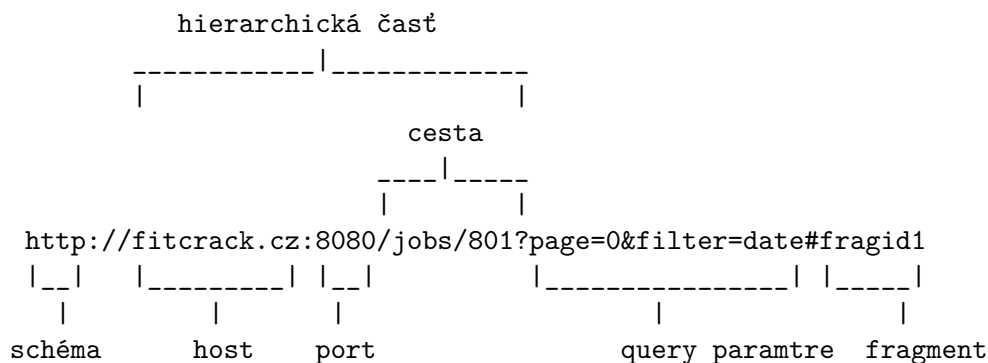
Výpis 3.2: príklad HTTP odpovedi

3.2 Architektúra REST

REST je úzko spojený s protokolom HTTP, keďže ho prvý krát navrhol a popísal Roy Fielding - jeden zo spoluautorov protokolu HTTP, v rámci jeho dizertačnej práce v roku 2000 [3]. Architektúra REST sa používa pre jednotný prístup ku zdrojom. Jedná sa o spôsob, ako klient môže pomocou základných HTTP volaní vytvárať, čítať, editovať alebo mazať informácie zo serveru. Zdrojom môžu byť dáta alebo stav aplikácie, pokiaľ sa dá dátami vyjadriť. K jednotlivým zdrojom sa pristupuje pomocou URI (viď. 3.2.1). Každý zdroj musí mať vlastný identifikátor URI.

3.2.1 URI

URI (Uniform Resource Identifier – jednotný identifikátor zdroja) je veľmi obecný koncept. Základný formát je veľmi voľný. Jedná sa o názov takzvanej schémy, oddelenej dvojbodkou od zvyšku URI. Tento zvyšok tvorí v podstate ľubovoľný reťazec znakov. Záleží na zvolenej schéme [2]. V architektúre REST sa URI používa hlavne s použitím schémy HTTP alebo HTTPS. Príklad takejto URI s jej jednotlivými komponentami je uvedený nižšie na obrázku 3.1. Všeobecne sa URI so schémou http skladá z IP adresy alebo názvu hosta, nepovinne sa za hostom uvádza port (ak nie je uvedený tak sa predpokladá pre http port 80 a pre https 443). Ďalej sa uvádza cesta k zdroju a následne nepovinné query parametre oddelené ampersandom. Na konci URI sa môže objaviť fragment stránky, ktorý často obsahuje id HTML elementu na ktorý sa má prehliadač po otvorení URI zamerať.



Obr. 3.1: Formát URI pri použití schémy HTTP

3.2.2 Zásady architektúry REST

Aby služba mohla byť považovaná za RESTful, musí spĺňať šesť formálnych obmedzení. Vďaka nim sú služby vytvorené pomocou REST architektúry výkonné, škálovateľné, jednoduché, ľahko upraviteľné, prenositeľné a spoľahlivé.

Architektúra klient-server

Jednou z najdôležitejších zásad architektúry REST je klient-server architektúra. Vďaka tomu je možné rovnomernejšie rozložiť záťaž. Server negeneruje pre užívateľa grafické rozhranie a môže obslúžiť viac užívateľov. Taktiež rozloženie záťaže umožňuje jednoduchú prenositeľnosť užívateľského rozhrania na viaceré platformy.

Bezstavová architektúra (Statelessness)

Jedná sa o bezstavovú architektúru z pohľadu servera. Medzi dotazmi sa na serveri nemôžu ukladať žiadne informácie o stave klienta. Každá žiadosť od akéhokoľvek klienta musí obsahovať všetky potrebné informácie na vybavenie dotazu. Svoj stav si klient uchováva sám. Trvalý stav klienta môže server uchovávať v databáze.

Možnosť uchovávať zdroje v medzipamäti (Cacheability)

Pre zlepšenie výkonnosti celého systému musí server označiť zdroje ako uložitelné alebo neuložitelné do medzipamäte. Uložitelné zdroje sú také, ktoré sa často nemenia. Keď si klient požiada o takýto zdroj, server mu odpovie okrem dát z daného zdroja aj informáciou, do kedy má uchovať dané dáta v medzipamäti. Keď bude klient v budúcnosti potrebovať prístup k dátam, ktoré má uložené v medzipamäti, a nevypršala expiračná doba dát, načíta si dáta z medzipamäte. Tým pádom nezatažuje server a pristúpi k dátam rýchlejšie.

Vrstevnatosť (Layered system)

Klient zvyčajne nemôže povedať či je pripojený ku koncovému serveru alebo len k nejakému sprostredkovateľovi. Sprostredkovateľské servery sa používajú na zlepšenie škálovateľnosti systému tým, že umožnia rozložiť záťaž. Môžu tiež uplatňovať bezpečnostné pravidlá (napríklad ochrana proti DDoS útokom).

Zaslanie klientovi spustiteľného kódu (Code on demand)

Jedná sa o voliteľné obmedzenie. Server môže klientovi zaslať spustiteľný kód (zvyčajne v skriptovacom jazyku ako Javascript). Vďaka tomu môže server dočasne rozšíriť alebo prispôbiť funkčnosť klienta.

Jednotné rozhranie

Základom akejkoľvek služby REST je jednotné rozhranie medzi klientom a serverom. Jedná sa hlavne o typy správ, ktoré môžu byť vo viacerých formátoch (HTML, XML, JSON).

3.2.3 Metódy pre prístup ku zdrojom

Architektúra REST definuje štyri základné metódy pre prístup k jednotlivým zdrojom. Tieto metódy sú implementované pomocou zodpovedajúcich metód HTTP protokolu. Významy metód sa líšia v závislosti od toho, či boli zavolané nad kolekciou, alebo nad určitým prvkom.

- **GET** - ak bol zavolaný na kolekciou, odpoveď obsahuje pole prvkov kolekcie. Tiež môže obsahovať doplňujúce údaje (tzv. metadata), napríklad koľko prvkov je v kolekcii. Pri zavolaní nad konkrétnym záznamom, vráti informácie o zázname.
- **POST** - vytvorí nový záznam v kolekcii. ID záznamu je zvyčajne automaticky vytvorené a vrátené v odpovedi dotazu.
- **PUT** - upraví záznam, alebo ak neexistuje tak záznam vytvorí.
- **DELETE** - zmaže celú kolekciu alebo konkrétny záznam.

Kapitola 4

Návrh systému

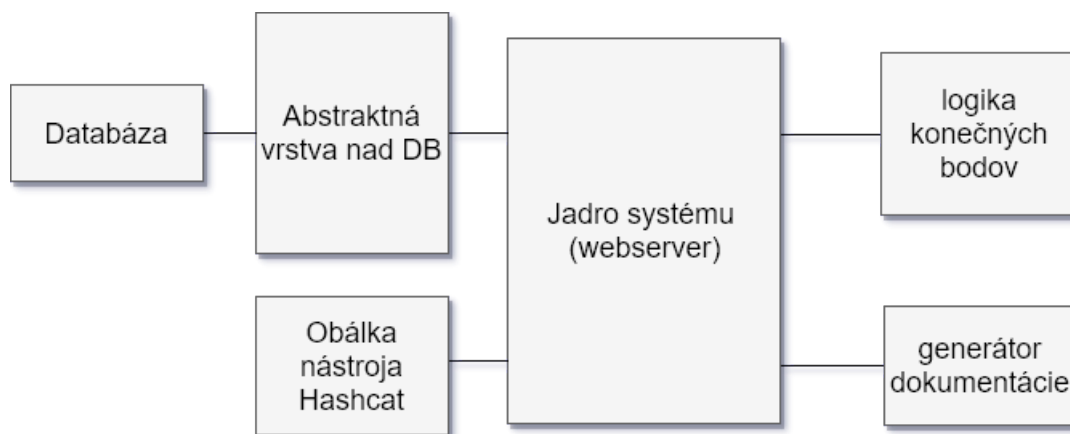
V tejto kapitole je popísaný koncepčný návrh systému. Součástíou tohto návrhu je diagram prípadov užitia, návrh databázy, @TODO: Poslednú časť tejto kapitoly tvorí návrh webovej prezentácie.

4.1 Návrh serverovej časti

V tejto kapitole sa zaoberám návrhom implementácie webového servera pre systém Fitcrack. Ako implementačné prostredie som zvolil Python3. Pri jeho výbere som bral v úvahu už zaužívané technológie v systéme Fitcrack, požadovanú funkčnosť systému a taktiež jeho prenositeľnosť.

4.1.1 Rozdelenie systému na moduly

Pre lepšiu organizáciu som celý systém rozdelil do niekoľko modulov (viď obr. 4.1). Jednotlivé moduly sú detailnejšie rozobrané v nasledujúcich podkapitolách.



Obr. 4.1: Rozdelenie systému do modulov

Jadro systému

Najhlavnejším modulom v systéme je jeho jadro, ktoré spĺňa úlohu webového servera a sú v ňom uložené nastavenia celého systému (prístupové údaje do databázy, zložka pre nahrá-

vane súborov, cesta k nástroju Hashcat atď.). Pri zlyhaní jadra webový server odpovedá HTTP správy s kódom začínajúcim číslom 5 (viď 3.1.3).

Logika konečných bodov

Jedná sa o modul, v ktorom sú obslužené dotazy na konkrétne URI (viď. 3.2.1). Zoznam dostupných URI je uvedený v tabuľke (viď. 4.1). Väčšina konečných bodov podporuje viac typov HTTP metód (viď. 3.2.3). Podľa výberu metódy sa vykoná operácia so zdrojom. Niektoré konečné body podporujú takzvané query parametre za url adresou. Pri chybe v tomto module, webový server odpovedá HTTP správami, ktoré začínajú číslom 4 (viď 3.1.3). Ak užívateľ žiada o neplatnú kombináciu URI adresy a HTTP metódy, je mu vrátená HTTP správa s kódom 404. V prípade že je užívateľ neprihlásený a žiada o zdroj, ktorý vyžaduje autentifikáciu, v odpovedi dostane HTTP správu s kódom 401. Pokiaľ je užívateľ prihlásený, ale na zdroj o ktorý žiada nemá dostačujúce oprávnenia, tento modul vracia HTTP správu s kódom 403. V inakšom prípade je dotaz týmto modulom spracovaný. Ak nedôjde k chybe počas spracovávania dotazu, modul vygeneruje odpoveď vo formáte JSON, ktorú pridá do tela HTTP odpovede s kódom 200 (viď 3.1.3).

Obálka nástroja Hashcat

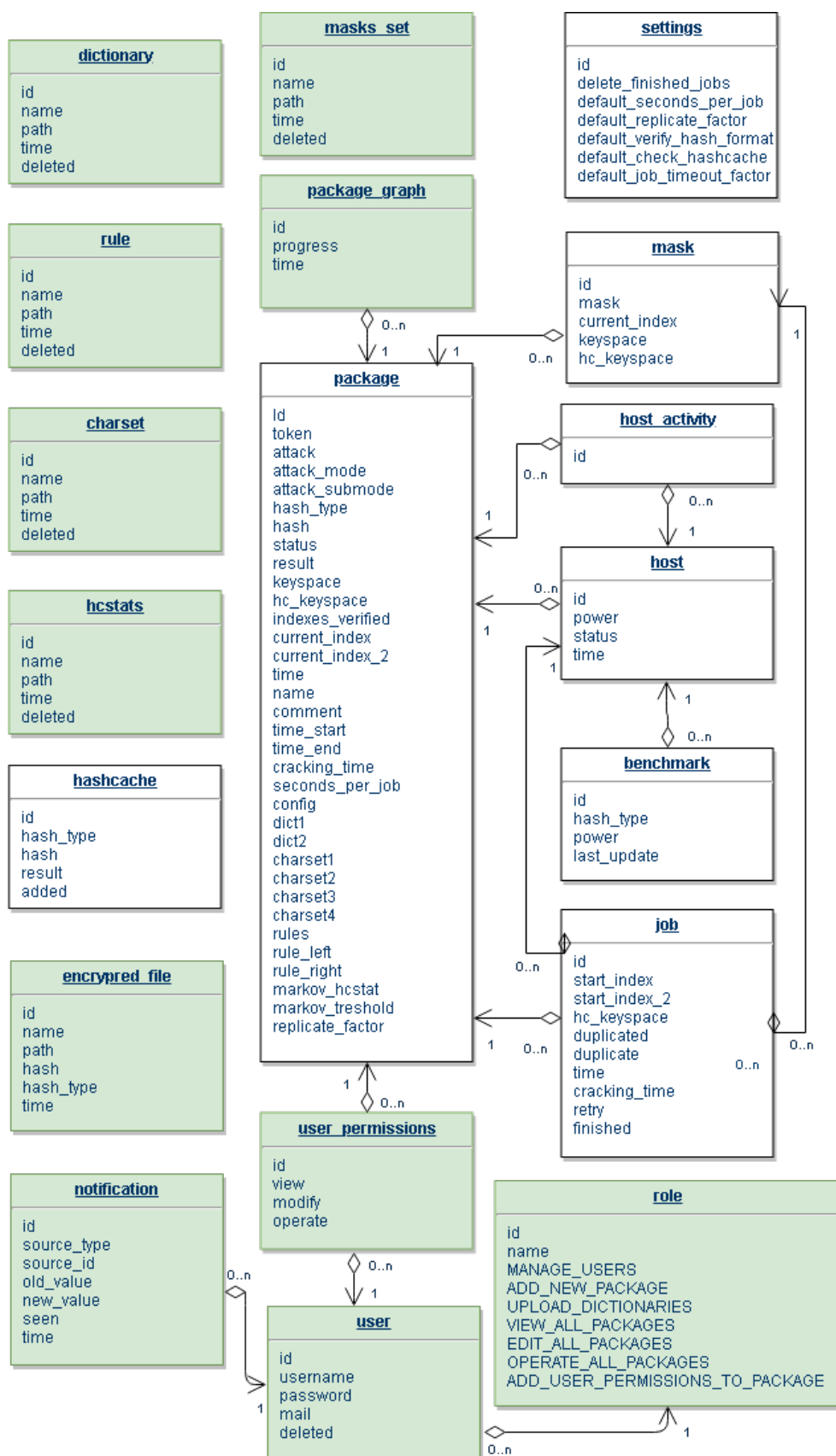
Aj keď serverová časť systému Fitcrack nepoužíva priamo nástroj Hashcat na generovanie hešov kryptografických funkcií, používa ho na rôzne vedľajšie účely, ako napríklad získavanie podporovaných typov útokov a kryptografických hešov, alebo výpočet veľkosti množiny hesiel pre útok. Z tohto dôvodu je potrebné implementovať obálku na tento nástroj vo forme objektu (triedy) s požadovanými funkciami. To nám uľahčí prístup k nástroju Hashcat a taktiež bezpečnejšie narábanie s ním.

Abstraktná vrstva nad databázou

Jedná sa o modul, ktorý uľahčuje prístup k databáze. Zároveň výrazne neovplyvňuje výkon a flexibilitu systému. Vďaka tomu, že systém s databázou nekomunikuje priamo, ale využíva abstraktnú vrstvu, je databáza lepšie chránená (ochrana pred útokmi typu SQL injection).

Databáza

Vzhľadom nato, že systém Fitcrack využíva databázový server MySQL, rozhodol som sa napojiť na túto databázu. Aby bolo možné implementovať všetky rozšírenia systému, bude potrebné modifikovať štruktúru databázy. Na obrázku 4.2 je možné vidieť entitno relačný diagram upravenej databázy. Zelenou farbou sú na ňom označené tabuľky, ktoré bude nutné do systému zaviesť.



Obr. 4.2: Entitno-relačný diagram databázy. Zelenou farbou sú vyznačené nové tabuľky.

Generátor dokumentácie

Generátor dokumentácie po spustení webového servera spracuje zdrojové kódy aj s komentármi a vytvorí pomocou nich interaktívnu dokumentáciu (viď obr. 4.3). Z nej je potom možné zistiť všetky dostupné koncové body spolu s príkladmi odpovede. Generovanie dokumentácie automaticky nastáva aj pri zmene zdrojových súborov.

Fitcrack API
hashcat : Endpointy ktoré priamo využívajú nástroj hashcat

GET /hashcat/attackModes Vracia zoznam podporovaných útokov

GET /hashcat/hashTypes Vracia zoznam podporovaných hashov

Response Class (Status 200)
Success

Model | Model Schema

```
{
  "hashtypes": [
    {
      "code": "string",
      "name": "string",
      "category": "string"
    }
  ]
}
```

Response Content Type application/json ▼
Try it out!

hosts : Operácie s hostami Show/Hide List Operations Expand Operations

GET /hosts/ Vracia list hostov

packages : Operácie s package Show/Hide List Operations Expand Operations

GET /packages/ Vracia list balíčkov

POST /packages/ Vytvorí nový package

Obr. 4.3: Ukážka vygenerovanej dokumentácie

4.1.2 Autentifikácia a oprávnenia užívateľov

Súčasná verzia systému Fitcrack nepodporuje viacerých užívateľov. Aby bolo možné do systému pridať užívateľov, je nutné rozšíriť databázu (viď. 4.1.1). Ako spôsob riešenia právomocí užívateľom som vybral systém oprávnení na základe rolí. Každý užívateľ bude mať uvedenú roľu, ktorá mu udeľuje právomoci na určité akcie. Jedná sa o jednoduchý systém, plne postačujúci pre menší počet užívateľov. Jednotlivé právomoci sú nasledujúce.

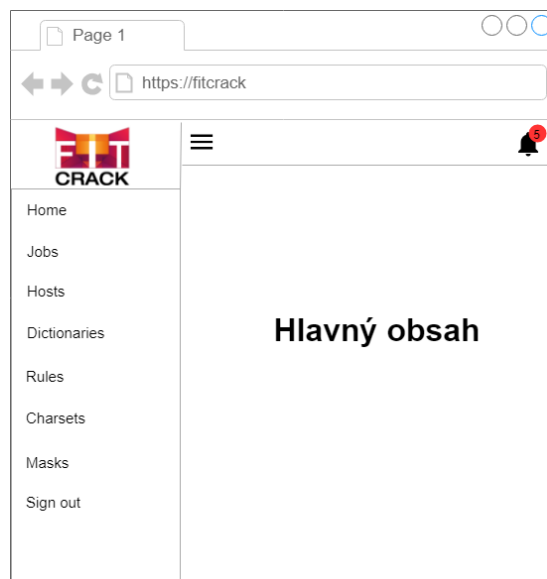
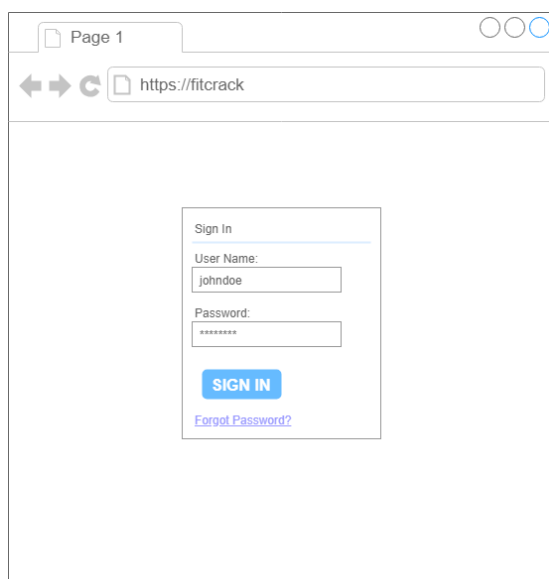
- **Spravovanie užívateľov** - S takýmto oprávnením môže užívateľ vytvárať, mazať a modifikovať iných užívateľov.
- **Pridávanie novej úlohy** - Užívateľ môže pridať do systému novú úlohu

- **Nahrávanie súborov** - Užívateľ má oprávnenie do systému nahráť slovník, cudzojazyčné znakové sady, súbory s pravidlami, masky a súbory s markovými reťazcami.
- **Zobrazenie všetkých úloh** - Užívateľ vidí v systéme všetky vytvorené úlohy.
- **Editovanie úloh** - Užívateľ môže všetky úlohy modifikovať.
- **Operácie s úlohami** - Užívateľ môže vykonávať s úlohami operácie štart, stop a reštart.

4.2 Návrh klientskej časti

Klientskú časť tvorí moderná webová aplikácia, ktorá pomocou HTTP žiadostí komunikuje so serverom (viď 3.1). Analýzou požiadavkov som zistil, že aplikácia by mala byť čo najlepšie ovládateľná a prehľadná pre užívateľa. Taktiež je veľmi dôležité, aby bol vzhľad stránky dobre štruktúrovaný a pútavý. Aplikácia musí byť dostupná aj pre užívateľov s menším rozlíšením displeja ako klasický počítačový monitor. Z tohto dôvodu sa bude aplikácia prispôbovať užívateľskému rozlíšeniu (responzivnosť).

Keďže je systém uzavretý a vyžaduje prihlasovanie, je vhodné, aby úvodná stránka aplikácie obsahovala prihlasovací formulár (viď obrázok 4.4). Systém bude distribuovaný s jedným užívateľom v databáze, ktorý bude slúžiť na prvé prihlásenie. Po prihlásení prostredníctvom tohoto užívateľa je možné vytvárať nových užívateľov vo vnútri aplikácie.



Obr. 4.4: Návrh úvodnej stránky aplikácie. Obr. 4.5: Návrh úvodnej stránky aplikácie.

Všetky ostatné stránky tvorí jednotné rozhranie, ktoré sa skladá z postranného bočného panelu, hornej lišty a hlavného obsahu, ktorý je generický pre každú stránku. Tento bočný panel obsahuje logo Fitcracku, a navigáciu. Pre zachovanie responzivnosti celej aplikácie sa tento panel pri zariadeniach s menším rozlíšením skryje, a zobrazí až po stlačení tlačítka Menu. Horná lišta obsahuje vľavo tlačítka Menu a napravo tlačítka Upozornenia, ktoré indikuje počet nových upozornení. Po stlačení tlačítka Upozornenia sa z pravej strany obrazovky vysunie panel s upozorneniami. Vďaka jednotnému rozmiestneniu navigačných

prvkov pre všetky stránky, má užívateľ vždy prístup k všetkým rubrikám. Návrh štruktúry rozloženia stránky je možné vidieť na obrázku 4.5.

4.3 Typy útokov v systéme Fitcrack

V systéme Fitcrack je možné vytvárať päť druhov útokov. Pôvodný systém Fitcrack podporoval len dva základné. Každý útok je jedinečný a vhodný na iné situácie.

4.3.1 Útok s maskami

Jedná sa o útok hrubou silou. Užívateľ vo webovej aplikácii zadá jednotlivé masky. Masky sú textové reťazce pozostávajúce z pevne daných znakov a zástupných symbolov. Zástupné symboly sa označujú znakom "?" a symbolom znakovkej sady. Napríklad maska `password?d` bude generovať heslá `password0` - `password9`. Zabudované znakové sady v systéme sú nasledovné.

- **?l** - znaková sada obsahuje malé písmená anglickej abecedy (abcdefghijklmnopqrstuvwxyz).
- **?u** - táto znaková sada obsahuje veľké písmená anglickej abecedy (ABCDEFGHIJKLMNOPQRSTUVWXYZ).
- **?d** - znaková sada obsahujúca arabské číslice (0123456789).
- **?h** - hexadecimálna znaková sada s malými písmenami (0123456789abcdef).
- **?H** - hexadecimálna znaková sada s veľkými písmenami (0123456789ABCDEF).
- **?s** - špeciálne znaky ako napríklad medzera `!"#$%&'()*+,-./:;<=>?@`
- **?a** - jedná sa o zjednotenie znakových sád `?l?u?d?s`
- **?b** - v tejto znakovkej sade sa nachádzajú všetky ASCII znaky (0x00 - 0xff).

V systéme je možné vybrať si až 4 ďalšie ľubovoľné znakové sady k jednej úlohe. Tieto znakové sady sa potom zadávajú zástupnými symbolmi `?1` - `?4`.

Okrem znakových sád je možné pri útoku s maskami vybrať súbor s Markovými reťazcami. Tento súbor má zvyčajne príponu `.hcstat`, a je ho možné do systému nahráť, alebo vytvoriť analýzou slovníka s heslami priamo z prostredia webovej aplikácie. Jedná sa o súbor, v ktorom je uložená matica. Prvý stĺpec tejto matice obsahuje všetky znaky, z ktorých sa budú generovať heslá. Riadky matice sú usporiadané podľa pravdepodobnosti výskytu jednotlivých znakov. Príklad takejto matice je možné vidieť na obrázku 4.6. V prvom riadku matice sú znaky, ktorými najčastejšie začínajú heslá. Pri použití Markovho modelu sa negenerujú heslá postupne (napríklad pri použití masky `?1?1?1?1` sa negenerujú heslá v poradí `aaaa-zzzz`), ale práve podľa Markovho modelu. Využitie tohto modelu pri generovaní hesiel je založené na pozorovaní, že ľudia pri tvorení hesiel používajú skrytý Markovský model. Pri vytváraní úlohy je možné taktiež určiť Markov prah, čo je kladné celé číslo, ktoré označuje do ktorého stĺpca v Markovom modeli sa majú heslá generovať.

Tento útok už bol v systéme implementovaný. K útoku som pridal možnosť zadať znakové sady, Markovský model a možnosť načítať masky so súboru.

$$\begin{matrix} \varepsilon \\ a \\ b \\ c \\ \vdots \\ z \end{matrix} \begin{bmatrix} n & p & s & u & \dots \\ m & u & v & k & \dots \\ i & y & e & u & \dots \\ o & e & b & f & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a & e & o & m & \dots \end{bmatrix}$$

Obr. 4.6: Príklad Markovho modelu

4.3.2 Slovníkový útok

Jeden z najzákladnejších útokov je útok pomocou slovníku, ktorý obsahuje veľký počet hesiel. Na každom riadku slovníka sa nachádza jedno heslo. Okrem slovníku je možné zvoliť aj súbor s pravidlami. Každé pravidlo sa aplikuje na každé heslo so slovníka. Pravidlá sú reťazce znakov. Majú pomerne jednoduchú syntax a umožňujú modifikovať heslo rôznymi spôsobmi ¹. Medzi základné pravidlá patria nasledovné znaky.

- **l** - všetky písmená v hesle prevedie na malé písmená.
- **u** - prevedie prímená v hesle na veľké písmená.
- **t** - malé písmená prevedie na veľké a veľké na malé.
- **r** - otočí poradie písmen v hesle.
- **\$X** - na koniec hesla vloží znak X.
- **^X** - na začiatok hesla vloží znak X.
- **[** - zmaže prvý znak hesla.
- **]** - zmaže posledný znak hesla.

Pravidlá je možné medzi sebou aj kombinovať. Ak za každé heslo chceme pridať reťazec 123 a zároveň chceme každé heslo previesť na malé písmená, použijem pravidlo **l\$1\$2\$3**. V súbore s pravidlami sa môžu vyskytovať komentáre, ktoré začínajú znakom #, a prázdne riadky.

Zjednodušená verzia tohto útoku, bez možnosti aplikovať pravidlá, bola v systéme už implementovaná.

4.3.3 Kombinačný útok

Pri kombinačnom útoku je nutné vybrať dva slovníky (ľavý a pravý). Každé heslo z ľavého slovníka sa skonkatenuje s každým heslom z pravého slovníka. Takto vznikajú nové heslá. Z každého hesla v ľavom slovníku vznikne toľko nových hesiel, koľko je hesiel v pravom slovníku. Ku každému slovníku je možné uviesť jedno pravidlo, ktoré upraví heslá v celom slovníku. Tento útok pôvodná webová aplikácia nepodporovala.

¹https://hashcat.net/wiki/doku.php?id=rule_based_attack

4.3.4 Hybridné útoky

Posledné typy útokov, ktoré moje riešenie podporuje sú hybridné útoky. Jedná sa o spojenie slovníkového útoku s útokom s použitím masky. Užívateľ si v systéme vyberie slovník a zadá masku. Potom sa maska so slovníkom skombinuje a vytvorí sa tak kombinačný útok. Napríklad, keď užívateľ zadá masku `?d` a vyberie slovník ktorý obsahuje heslá `aaa`, `bbb`, `ccc`, výsledná množina hesiel je `aaa0`, `aaa1`, `aaa2`, `aaa3`, `aaa4`, `aaa5`, `aaa6`, `aaa7`, `aaa8`, `aaa9`, `bbb0`, ..., `ccc9`. K maske aj k slovníku je možné zadať jedno pravidlo. Hybridné útoky sú dva. Pri prvom je maska naľavo a slovník napravo, a pri druhom je slovník naľavo a maska napravo. Tento útok je nový v systéme Fitcrack.

4.4 Grafy

Keďže má byť nová aplikácia vizuálne prívetivá pre užívateľa, jej dôležitou súčasťou sú grafy. Navrhol som tri typy grafov.

- **Graf progresu úlohy** - tento graf je spojnicový a bude vykresľovať percentuálny pokrok jednej alebo viacerých úloh. Y-ová osa bude v percentách v rozmedzí 0 až 100. Na X-ovej ose bude čas. Tento graf zobrazuje koľko percent hesiel z celkovej množiny hesiel úlohy sa už skontrolovalo. Nezobrazuje progres hľadania hesla, keďže heslo môže byť nájdené hneď na začiatku množiny hesiel, alebo sa heslo nemusí nachádzať vôbec v množine hesiel danej úlohy.
- **Graf aktivity hostov** - jedná sa taktiež o spojnicový graf, na ktorom je možné sledovať aktivitu jednotlivých hostov. Na Y-ovej ose je možné vidieť počet krypto-grafických hešov, ktoré host počítal. Na X-ovej ose sa nachádza čas. Z tohto grafu by mala byť zrejmá aktivita Asimilátora (viď 2.3.3), ktorý prispôsobuje počet hesiel v jednej pracovnej úlohe pre hosta. Tento graf je možné zobraziť v rámci jednej úlohy, alebo v rámci systému.
- **Graf práce hostov** - ide o koláčový graf, ktorý je možné zobraziť pre konkrétnu úlohu. Je z neho možné vyčítať množstvo práce jednotlivých hostov v rámci úlohy.

4.5 Notifikácie

Ďalšou novinkou v systéme je systém notifikácií. Tieto notifikácie slúžia na upozornenie užívateľov pri zmene stavu úlohy. Sú rozdelené do niekoľkých kategórií - informačné, varovné, chybové a notifikácie úspechu. Vďaka tomu je možné notifikácie od seba aj vizuálne (farebne) odlíšiť. Medzi informačné notifikácie patria upozornenia o vytvorení úlohy, o spustení úlohy a o dokončovaní úlohy. Medzi notifikácie úspechu patrí upozornenia o nájdení hesla. Varovná notifikácia sa odošle v prípade, že sa úloha zastavila z dôvodu nastavenia konca doby výpočtu pri vytváraní alebo editovaní úlohy, ale heslo sa ešte nenašlo. Chybová notifikácia nastáva po vyčerpaní celej množiny hesiel danej úlohy a nenájdení hesla. Notifikácie obsahujú príznak videnia daným užívateľom. Tak tiež sa notifikácie o úlohách odosielať len užívateľom, ktorí majú k danej úlohe oprávnenia.

Popis	URI
Operácie s kolekciou znakových sád	/charset/
Operácie s konkrétnou znakovou sadou	/charset/<id>
Stiahnutie znakovej sady	/charset/<id>/download
Operácie s kolekciou slovníkov	/dictionary/
Operácie s konkrétnym slovníkom	/dictionary/<id>
Dáta na vykreslenie grafu podielu práce hostov	/graph/hostPercentage/<job_id>
Dáta na vykreslenie grafu vyťažnosti hostov	/graph/hostsComputing
Dáta na vykreslenie grafu vyťažnosti hosta	/graph/hostsComputing/<id>
Dáta na vykreslenie grafu progresu úloh	/graph/packagesProgress
Dáta na vykreslenie grafu progresu úlohy	/graph/packagesProgress/<id>
Útoky podporované nástrojom hashcat	/hashcat/attackModes
Typy hashov podporované nástrojom hashcat	/hashcat/hashTypes
Operácie s kolekciou hostov	/hosts/
	/hosts/<id>
	/hosts/info
	/job/
	/job/<id>
	/job/<id>/action
	/job/<id>/host
	/job/<id>/job
	/job/crackingTime
	/job/info
	/job/verifyHash
	/markovChains/
	/markovChains/<id>
	/markovChains/makeFromDictionary
	/masks/
	/masks/<id>
	/masks/<id>/download
	/masks/<id>/update
	/notifications/
	/notifications/count
	/protectedFiles/
	/protectedFiles/<id>
	/rule/
	/rule/<id>
	/rule/<id>/download
	/serverInfo/control
	/serverInfo/info
	/user/
	/user/<id>
	/user/isLoggedIn
	/user/login
	/user/logout
	/user/role
	/user/role/<id>

Tabuľka 4.1: Zoznam dostupných URI.

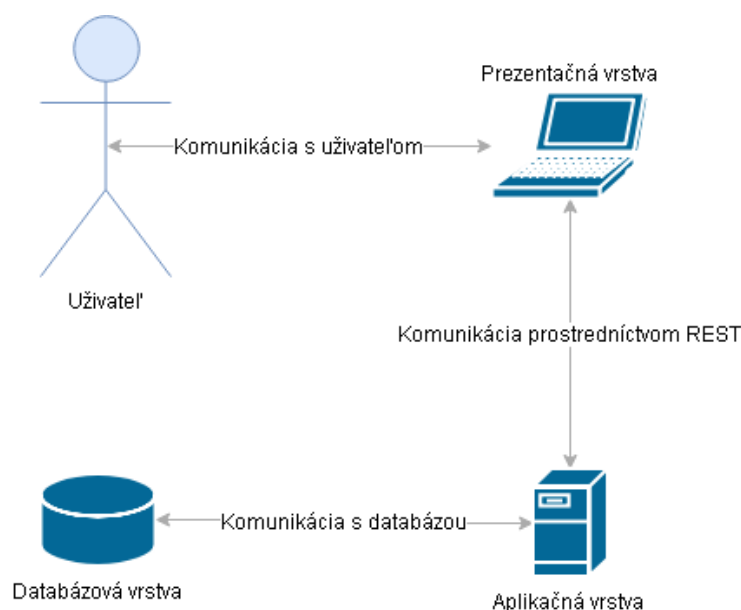
Kapitola 5

Implementácia

Architektúru systému tvoria tri vrstvy - prezentačná, aplikačná a dátová (alebo databázová). Vrstvy medzi sebou spolupracujú a každá vrstva môže bežať na rôznej výpočtovej infraštruktúre. Architektúru systému je možné vidieť na obrázku 5.1. Prezentačná vrstva má za hlavnú úlohu komunikovať s aplikačnou vrstvou a vizualizovať dáta užívateľovi. Zobrazenie vo webovej prezentácii je implementované pomocou SPA (Single Page Application). Hlavnou úlohou SPA je synchronizácia stavu objektov medzi dátovou a prezentačnou vrstvou.

Cieľom aplikačnej vrstvy aplikácie je spracovanie požiadaviek od prezentačnej vrstvy a komunikácia s databázovou vrstvou. Komunikácia s prezentačnou vrstvou prebieha vo formáte JSON.

Databázová vrstva tvorí MySQL databáza, ktorá je vytvorená podľa ER diagramu z obrázka 4.2. Databázová vrstva spracováva požiadavky aplikačnej vrstvy a vykonáva operácie ako ukladanie dát, zobrazovanie dát, vyhľadávanie a mazanie dát.



Obr. 5.1: Architektúra aplikácie

5.1 Implementácia databázovej vrstvy

Databázová vrstva je implementovaná pomocou databázového systému MySQL. Je vytvorená podľa entitno relačného diagramu, ktorý je možné vidieť na obrázku 4.2. Tvorí ju pôvodné tabuľky systému Fitcrack a nové tabuľky, ktoré umožňujú implementáciu rozšírení systému. Tabuľky `dictionary`, `rule`, `masks_set`, `hcstats` a `charset` slúžia na ukladanie informácií o súboroch nahratých do systému. Majú jednotnú štruktúru. Tvorí ju jednotný identifikátor záznamu v rámci tabuľky, názov súboru, cesta k súboru, čas pridania do databázy a príznak zmazania, ktorý slúži na signalizáciu aplikačnej vrstve, aby už súbor označený ako zmazaný, neposielala ďalej prezentačnej vrstve. Súbor označený ako zmazaný ostáva v systéme naďalej uložený kvôli predchádzaniu chýb v prípade že úloha, ktorá súbor využíva stále beží. Ďalšie nové tabuľky sú `user` a `role`, ktoré slúžia umožňujú autentifikáciu užívateľov a správu oprávnení. Pri užívateľoch sa ukladá ich prihlasovacie meno, heslo v zašifrovanej podobe s pridanou soľou, email a odkaz do tabuľky `role`, ktorá obsahuje jednotlivé užívateľov oprávnenia v rámci systému. Oprávnenia k jednotlivým úlohám môžu byť pridané prostredníctvom tabuľky `user_permissions`. Tabuľka `notification` slúži na uchovávanie upozornení pre užívateľov. Údaje o prograse úloh sa uchovávajú v tabuľke `package_graph`. Z týchto údajov je následne možné na prezentačnej vrstve vytvoriť graf progresu (viď 4.4). V databáze sa tiež nachádza trigger, ktorý sa vykoná vždy po upravení úlohy. Tento trigger je možné rozdeliť na dve časti. Prvá časť je možné vidieť na výpise 1 a slúži na počítanie progresu úlohy a pridávanie záznamov do tabuľky `package_graph`. Druhá časť triggeru je vyobrazená na výpise 2. Slúži na rozosielanie notifikácií užívateľom.

Algoritmus 1 Telo triggeru, ktorý pridáva záznamy do tabuľky `package_graph`.

```
1 -- verified passwords changed
2 IF NEW.indexes_verified <> OLD.indexes_verified THEN
3   INSERT INTO package_graph (progress, package_id) VALUES (
4     -- if job finished or exhausted set progress to 100.
5     -- Else calculate progress.
6     IF(NEW.hc_keyspace = 0 OR NEW.status = 1 OR NEW.status = 2,
7       100,
8       ROUND((NEW.indexes_verified / NEW.hc_keyspace) * 100, 2)
9     ),
10    NEW.id
11  );
12 END IF;
```

Je dôležité aby trigger vybral len tých užívateľov, ktorí majú k úlohe prístup (užívateľia s rolou, ktorá má oprávnenie na zobrazenie všetkých úloh, alebo užívateľia s oprávnením k úlohe v tabuľke `user_permissions`).

Algoritmus 2 Trigger, ktorý má na starosti rozposielanie notifikácií užívateľom.

```
1 IF NEW.status <> OLD.status THEN
2   OPEN userWithPermissionCursor;
3   user_loop: LOOP
4     FETCH userWithPermissionCursor INTO user_idCursor;
5     IF done THEN LEAVE user_loop; END IF;
6     INSERT INTO fc_notification
7       VALUES user_id, source_id, old_value, new_value
8       (user_idCursor, NEW.id, OLD.status, NEW.status);
9   END LOOP;
10  CLOSE userWithPermissionCursor;
11 END IF;
```

5.2 Implementácia aplikačnej vrstvy

Aplikačná vrstva je implementovaná v prostredí Python3. Tvorí ju sada balíkov, ktoré plnia rozličné úlohy. Adresárovú štruktúru je možné vidieť na obrázku 5.2.

```
\---src
  +---api
  |   \---fitcrack
  |       +---attacks
  |       \---endpoints
  |           +---charset
  |           +---dictionary
  |           +---graph
  |           +---hashcat
  |           +---host
  |           +---markov
  |           +---masks
  |           +---notifications
  |           +---package
  |           +---protectedFile
  |           +---rule
  |           +---serverInfo
  |           \---user
  \---database
```

Obr. 5.2: Hierarchia balíkov aplikačnej vrstvy.

V koreňovom adresári sa nachádza súbor `app.py`, ktorý obsahuje funkciu `main()`. Táto funkcia najprv prevedie inicializáciu systému, systém nakonfiguruje a načíta všetky koncové body aplikačného rozhrania, a následne spustí webový server. O chod webového servera sa stará framework Flask (viď ??). Ďalší dôležitý súbor v koreňovom adresári je `settings.py`, ktorý slúži na nastavenie servera. Je v ňom možné nakonfigurovať cestu k databázovému systému, debugovací režim, adresu a číslo portu servera. Tiež obsahuje cesty k zložkám

na ukladanie slovníkov, Markovských modelov, cudzojazyčných znakových sád, súborov s pravidlami a zašifrovaných súborov. Ďalej sa v tomto súbore nachádzajú cesty k rôznym pomocným programom, ktoré systém využíva. Jedná sa o program Hashcat (viď 2.1), MaskProcessor¹, XtoHashcat² a HashValidator³. Koreňový adresár obsahuje jediný balík s názvom `src`, v ktorom sú ďalšie balíky `database` a `api`. Balík `database` obsahuje modul s názvom `models.py`, ktorý má na starosti mapovanie databázy na abstraktný databázový model. Tento súbor obsahuje triedy, ktoré predstavujú jednotlivé tabuľky z databázy. Príklad takejto triedy je možné vidieť na výpise 3.

Algoritmus 3 Trieda, ktorá predstavuje mapovanie na databázovú tabuľku.

```

1 class FcPackageGraph(Base):
2     __tablename__ = 'fc_package_graph'
3
4     id = Column(BigInteger, primary_key=True)
5     progress = Column(Numeric(5, 2), nullable=False)
6     package_id = Column(ForeignKey('fc_package.id'), nullable=False, index=True)
7     time = Column(DateTime, server_default=text("CURRENT_TIMESTAMP"))
8
9     package = relationship('FcPackage')
10
11     def as_graph(self):
12         return {
13             'time': str(getattr(self, 'time')),
14             'package_id': round(getattr(self, 'package_id'))
15         }
```

Tieto triedy boli vytvorené prostredníctvom programu `sqlacodegen`⁴. Abstraktná vrstva nad databázou výrazne uľahčuje manipuláciu s databázovými objektami. K triedam je možné doimplementovať hybridné vlastnosti, ktoré bývajú zvyčajne vyjadrené na základe stĺpcov odpovedajúcej tabuľky. Napríklad, keďže je nájdené heslo pri úlohe uložené v stĺpci `result` vo formáte Base64⁵, je možné vytvoriť hybridnú vlastnosť triedy, ktorá automaticky po načítaní dát z databázy dekoduje heslo uložené vo formáte Base64 a priradí ho do hybridnej vlastnosti. Abstraktná vrstva databázy je implementovaná prostredníctvom frameworku `SQLAlchemy` (viď ??).

5.2.1 Implementácia jednotlivých URI

Pri vytváraní funkcií na spracovanie jednotlivých žiadostí používam rozšírenie do frameworku `Flask` s názvom `Flask-RESTPlus` (viď ??). Balík `endpoints` obsahuje ďalšie balíky, v ktorých prebieha ošetrovanie jednotlivých žiadostí. Väčšina týchto balíkov v sebe obsahuje štyri súbory. Jedná sa o `argumentsParser.py`, `responseModels.py`, `functions.py` a modul, s rovnakým názvom ako má balík do ktorého patrí, v ktorom sú definované triedy na ošetrovanie žiadostí. Tieto triedy obsahujú funkcie s názvom HTTP metód (viď 3.2.3). Príklad

¹<https://hashcat.net/wiki/doku.php?id=maskprocessor>

²<https://fitcrack.fit.vutbr.cz/?i=download>

³<https://github.com/Zipperisk/hashValidator>

⁴<https://pypi.org/project/sqlacodegen/1.1.0/>

⁵<https://sk.wikipedia.org/wiki/Base64>

takejto triedy je možné vidieť na výpise 4. V module `argumentsParser.py` sa nachádzajú

Algoritmus 4 Príklad triedy, ktorá ošetruje URI `/jobs`

```
1 @ns.route('/jobs')
2 class jobs(Resource):
3
4     @api.expect(jobList_arguments)
5     @api.marshal_with(pageOfJobs_model)
6     def get(self):
7         args = jobList_arguments.parse_args(request)
8         jobs_page = FcJob.query.paginate(args['page'], args['per_page'])
9         return jobs_page
10
11     @api.expect(addJob_arguments)
12     @api.marshal_with(newJob_model)
13     def post(self):
14         data = request.json
15         job = create_job(data)
16         return { 'message': 'Job ' + job.name + ' succesful created.',
17                 'status': True }
```

objekty, ktoré slúžia na spracovanie argumentov žiadostí. Volajú sa formou dekorátorov pred funkciou. Napríklad na výpise 4 je možné vidieť volanie `jobList_arguments` pomocou dekorátoru `api.expect`. V module `responseModels.py` sa nachádzajú modely odpovedí na žiadosti, ktoré daný balík spracováva. Tieto modely odpovedí sa volajú prostredníctvom dekorátora `api.marshal_with`. Vďaka týmto modelom je možné aby funkcie ošetrojúce žiadosti vracali štruktúry z prostredia python (slovník, pole slovníkov, n-tice alebo množiny). Tieto štruktúry sú potom prekonvertované dekorátorom na textové reťazce vo formáte JSON, ktorými server odpovedá prezentačnej vrstve. V module `functions.py` sú uložené funkcie, ktoré ktoré balík, ošetrojúci sadu URI používa. Z výpisu 4 sa tu napríklad nachádza funkcia `create_job`.

5.2.2 Spracovanie útokov

Spracovanie úlohy na aplikačnej vrstve prebieha v niekoľkých fázach. Najprv sa získa hash určený na lámanie. Ten je možné buď zadať priamo, alebo nahráť šifrovaný súbor, z ktorého systém pomocou programu `XtoHashcat`⁶ hash extrahuje. Následné overí formát hashu pomocou nástroja `hashValidator`. Jedná sa o nástroj, napísaný v jazyku C, ktorý som vytvoril zjednodušením a úpravou programu `Hashcat` (viď 2.1). Týmto spôsobom som sa chcel zbaviť závislostí, ktoré program `Hashcat` vyžaduje pri niektorých funkciách čo nie sú využívané na serverovej časti systému. Po overení hashu sa systém pokúsí nájsť heslo v tabuľke `hashcache`. V prípade, že sa heslo v databáze nenašlo prebieha spracovanie jednotlivých útokov v module `attacks`. Spracovanie jednotlivých útokov sa od seba líši a sú popísané v zozname nižšie.

- **Slovníkový útok** - pri tomto útoku sa v tele žiadosti vyžaduje identifikátor slovníka. Podľa tohto identifikátora sa vyhledá v databáze v tabuľke `dictionary` slovník s

⁶<https://fitcrack.fit.vutbr.cz/?i=download>

príslušným identifikátorom. Potom sa overí či slovník existuje v súborovom systéme. Ak je zadaný súbor s pravidlami, taktiež sa vyhľadá v databáze (v tabuľke `rule`) a overí sa jeho existencia na disku. Na konci spracovávania útoku sa vypočíta veľkosť množiny hesiel danej úlohy. Pri tomto type útokov je veľkosť tejto množiny rovná súčinu počtu pravidiel a počtu hesiel v slovníku (ak nebol vybraný súbor s pravidlami tak len počtu hesiel v slovníku). O podrobnejšom priebehu útoku sa je možné dočítať v podkapitole 4.3.2.

- **Kombinačný útok** - tento útok vyžaduje zadané dva identifikátory slovníkov. Po overení existencie oboch slovníkov sa vypočíta veľkosť množiny hesiel ako súčet počtu hesiel v slovníkoch. Podrobnosti o tomto type útoku sú uvedené v podkapitole 4.3.3
- **Útok s maskami** - spracovanie tohto typu útoku je najzložitejšie. Najprv sa musí skontrolovať syntax každej masky. Potom sa overuje existencia Markovho modelu a súborov s cudzojazyčnými znakovými sadami. Veľkosť množiny hesiel sa pre tento útok počíta pomocou masiek. Za každý zástupný znak v maske sa veľkosť množiny hesiel pre danú masku vynásobí počtu symbolov, ktoré zástupný znak symbolizuje (pre `?d` 10, pre `?l` 26 a pod.). Veľkosti množín hesiel pre jednotlivé masky sa potom spočítajú a výsledok je celková veľkosť množiny hesiel pre všetky masky. Viac o tomto type útoku je uvedené v kapitole 4.3.1.
- **Hybridné útoky** - najprv sa skontroluje syntax zadanej masky a následne sa z nej pomocou programu `MaskProcessor`⁷ vytvorí slovník, ktorý sa uloží na disk. Slovník zadaný užívateľom sa načíta z databázy a skontroluje sa jeho existencia na disku. Veľkosť množiny hesiel sa vypočíta pomocou súčinu počtu hesiel v oboch slovníkoch (tak isto ako u kombinačného útoku). Tento útok je detailnejšie rozobraný v podkapitole 4.3.4.

Po spracovaní útoku je nutné vytvoriť konfiguráciu útoku pre generátor vo formáte TLV⁸. Potom sa už len záznam uloží do databázy a priradia sa k nemu hostia. Aplikačnej vrstve sa odošle správa o úspešnom pridaní úlohy do databázy.

5.2.3 Autentizácia užívateľov

Na autentizáciu užívateľov som vybral modul `Flask-Login`⁹. Jedná sa o rozšírenie do frameworku `Flask`, ktoré poskytuje správu nad sedením užívateľov (prihlásenie, odhlásenie a zapamätanie si užívateľa). Toto rozšírenie pridáva do systému dekorátor `login_required`, ktorý zabezpečuje aby sa neprihlásený užívateľ nedostal k zdrojom, ktoré sú viditeľné len pre prihlásených užívateľov.

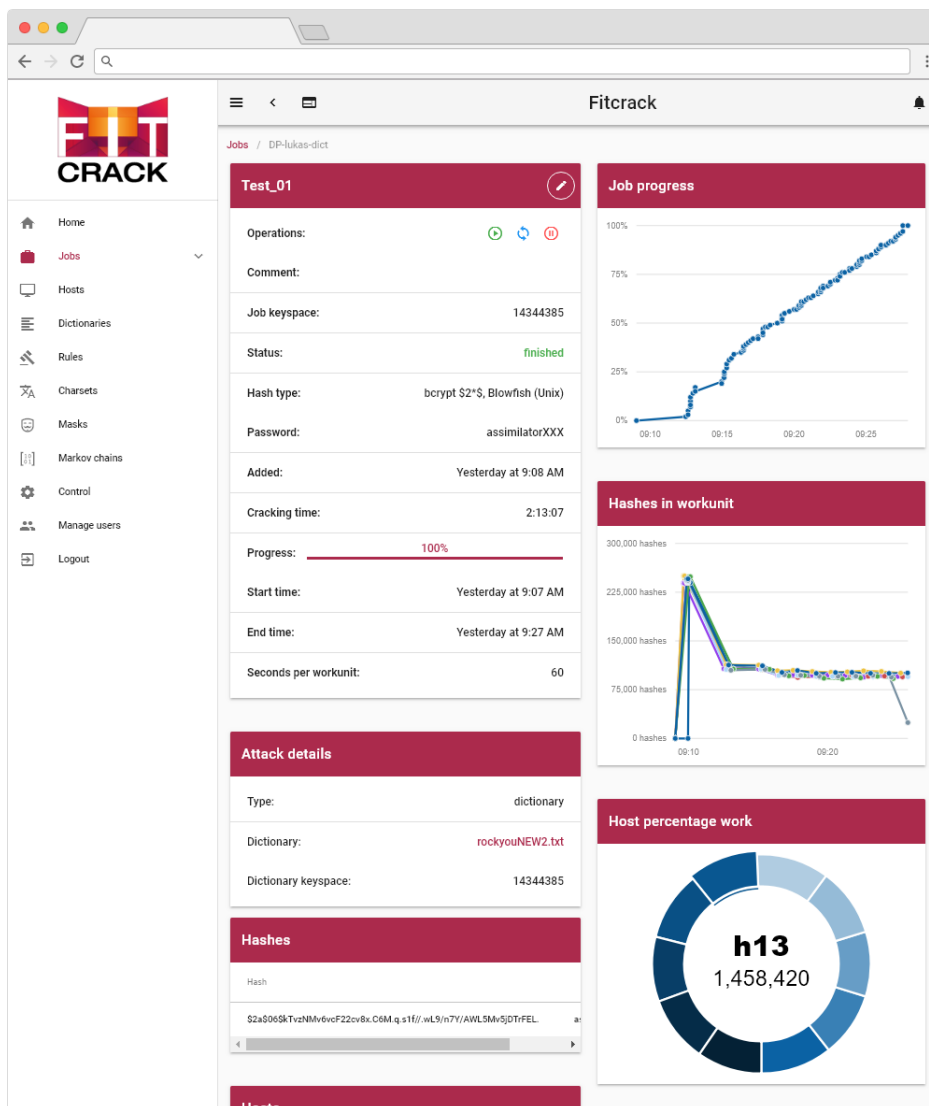
5.3 Implementácia prezentačnej vrstvy

Prezentačná vrstva je implementovaná ako SPA (single Page Application alebo jednostránková aplikácia). Na rozdiel od klasických webových aplikácií poskytujú SPA užívateľsky príjemnejšie prostredie, ale vyžadujú použitie moderných webových prehliadačov. S aplikačnou vrstvou systému komunikuje pomocou asynchronných žiadostí typu AJAX. Vďaka tomu je

⁷<https://hashcat.net/wiki/doku.php?id=maskprocessor>

⁸<https://en.wikipedia.org/wiki/Type-length-value>

⁹<https://flask-login.readthedocs.io/en/latest/>



Obr. 5.3: Detailná stránka úlohy

možné meniť obsah stránok bez potreby ich znovu načítať zo servera. Na zasielanie žiadostí a prijímanie odpovedí AJAX používam framework AXIOS¹⁰. Kvôli ľahkému spracovaniu na prezentačnej vrstve prebieha komunikácia s aplikačnou vrstvou vo formáte JSON. Webová aplikácia je implementovaná pomocou frameworku Vue (viď ??). Výsledok implementácie prezentačnej vrstvy je možné vidieť na obrázkoch 5.3, kde je snímka z detailného pohľadu na úlohu.

5.3.1 Implementácia komponentov

Pre zjednodušenie implementácie som každú stránku rozdelil na znovupoužiteľných komponent, ktoré predstavujú logický celok. Rozklad stránky detailu úlohy na komponenty je možné vidieť na obrázku 5.4. Jednotlivé komponenty sú uložené v súboroch s príponou

¹⁰<https://github.com/axios/axios>



Obr. 5.4: Detailná stránka úlohy rozdelená na komponenty

.vue, ktoré sa nachádzajú v zložke `src/components`. Príklad obsahuje takéhoto súboru je možné vidieť na výpise 5.

Komponenty sa zvyčajne skladajú z troch častí. Prvá časť je kód v jazyku HTML zabalený v elemente `template`. V tejto časti sa popíše štruktúra komponentu a taktiež sa zviažu javascriptové premenné s HTML elementmi. Využíja sa pri tom takzvaný Data-Binding¹¹, ktorý štandardne funguje obojsmerne (pri zmene hodnoty v javascripte sa zmení hodnota HTML elementu a tiež pri zmene HTML elementu užívateľom sa zmení hodnota javascriptovej premennej). V ďalšej časti komponentu sa popíše javascriptová logika. Táto časť je obalená v elemente `script`. Vo poli `props` obsahuje vlastnosti, ktoré sa predávajú z rodičovského komponentu. Premenné si komponent uchováva vo funkcii¹² `data`. Jednotlivé funkcie sú uložené v objekte `methods`. Komponent má životný cyklus zložený z niekoľko fáz, ktoré je možné nájsť v prílohe @TODO. Zvyčajne najdôležitejšie stavy pre komponent sú:

- **created** - stav v ktorom prebieha inicializácia komponentu. Komponent ešte nie je vykreslený na stránke.
- **mounted** - komponent je zobrazený na stránke.
- **destroyed** - fáza, v priebehu ktorej je komponent vymazaný zo stránky. Tento stav je vhodný pre dealokáciu zdrojov, ktoré komponent využíva (napríklad volanie javascriptovej funkcie `clearInterval`).

¹¹<https://vuejs.org/v2/guide/syntax.html>

¹²<https://vuejs.org/v2/guide/components.html#data-Must-Be-a-Function>

Algoritmus 5 Zdrojový kód komponentu notifikácie

```
1 <template>
2   <div class="cont">
3     <router-link :to="'/jobs/' + jobId">
4       <v-alert :type="type" v-bind:class="{ seen: seen}">
5         <span>{{ text }}</span>
6         <span>{{ $moment(time).calendar() }}</span>
7       </v-alert>
8     </router-link>
9   </div>
10 </template>
11 <script>
12   export default {
13     name: 'notification',
14     props: ['type', 'text', 'seen', 'time', 'jobId']
15   }
16 </script>
17 <style scoped>
18   .seen {
19     opacity: 0.5;
20   }
21 </style>
```

Okrem vlastných komponentov, pre zefektívnenie práce používam aj niektoré komponenty zo sady komponentov **Vuetify**¹³. Jedná sa o sadu responzívnych komponentov v modernom dizajne.

5.3.2 Globálne dáta

Dáta, ku ktorým majú prístup všetky komponenty, sú uložené v globálnom skladisku, ktoré sa nazýva **Store**. Nachádzajú sa v ňom dáta u užívateľovi (užívateľské meno, email a jednotlivé oprávnenia), príznak prihlásenia užívateľa, URL adresa serverovej časti a globálne funkcie.

5.3.3 Navigácia v aplikácii

Pri jednostránkových aplikáciách (SPA) musí byť vyriešená adresácia pomocou URL. Ja som si pre navigáciu medzi jednotlivými stránkami vybral knižnicu **Vue Router**, ku ktorému som vytvoril konfiguračný súbor. Ten obsahuje popísané všetky cesty v aplikácii aj s komponentami, ktoré sa majú pri zhode cesty vykresliť. Časť konfiguračného súboru je možné vidieť na výpise 6.

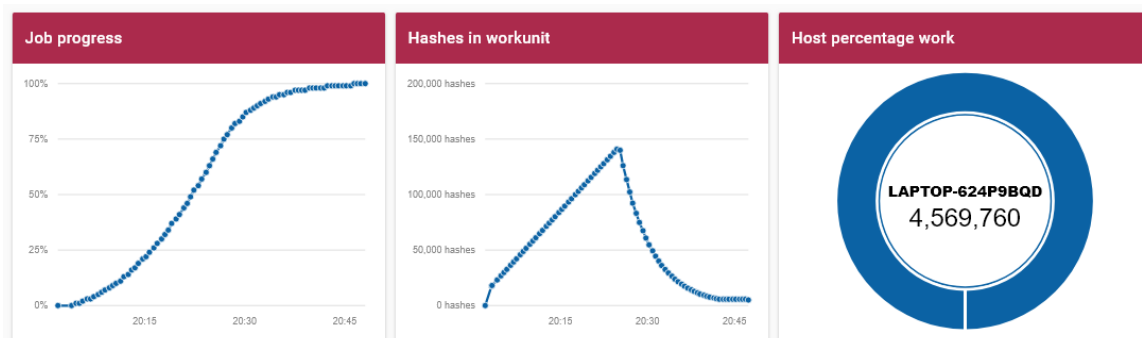
¹³<https://vuetifyjs.com/>

Algoritmus 6 Konfiguračný súbor navigácie

```
1 routes: [  
2   {  
3     path: '/',  
4     name: 'home',  
5     component: home  
6   },  
7   {  
8     path: '/jobs',  
9     name: 'jobs',  
10    component: jobs  
11  },  
12  {  
13    path: '/jobs/:id',  
14    name: 'jobDetail',  
15    component: jobDetail  
16  },  
17  ...  
18  { path: "*",  
19    component: PageNotFound  
20  }  
21 ]
```

5.3.4 Grafy

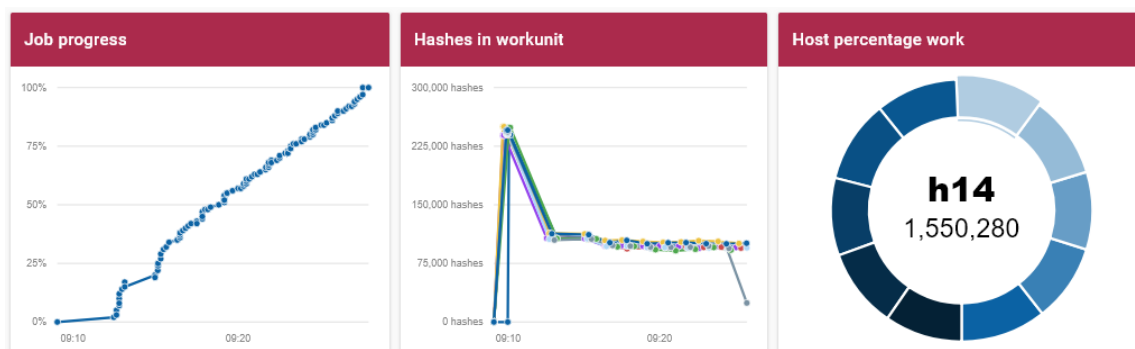
Výraznú časť užívateľského rozhrania aplikácie tvoria grafy, ktoré sú detailnejšie popísané v kapitole 4.4. Každý graf sa periodicky dotazuje servera so žiadosťou o nové dáta. Server odpovedá správou vo formáte JSON, v ktorej sa nachádzajú už spracované dáta grafu. Tým pádom sú grafy aktuálne aj bez nutnosti znovu načítania stránky. Na vykresľovanie grafov využívam javascriptovú knižnicu `morris.js`¹⁴. Na obrázku 5.5 sa nachádzajú grafy jednej z úloh.



Obr. 5.5: Grafy úlohy s jedným hostom

¹⁴<http://morrisjs.github.io/morris.js/index.html>

Z grafu **Hashes in workunit** (počet hashov v pracovnej jednotke) je zrejma práca Asimilátora (viď 2.3.3), ktorý ma za úlohu prispôbovať veľkosť pracovnej jednotky hostom. V tej úlohe bol zapojený len jeden host s názvom **LAPTOP-824P9BQD**. Jednalo sa o pomerne výkonného hosta, čo je možné vidieť aj podľa toho, že mu Asimilátor spolu s Generátorom prideliť čo raz viac hashov na overenie v pracovnej jednotke. Toto zväčšovanie by pokračovalo až kým by host nespracovával pracovné jednotky tak rýchlo, ako bolo stanovené pri vytváraní úlohy, ale od polovice overených hashov nastáva zmena v algoritme pri generovaní nových pracovných jednotiek. Táto zmena nastáva v dôsledku zefektívnenia výpočtu a má za následok postupné znižovanie počtu hesiel na overenie vo všetkých ďalších vygenerovaných pracovných jednotkách. Grafy so zapojením viacerých hostov je možné vidieť na obrázku 5.6. Koláčový diagram značí, že v tomto prípade boli hostia výkonnostne pomerne vyrovnaní.



Obr. 5.6: Grafy úlohy s viacerými hostami

5.3.5 Prehľad úloh a stránkovanie

Stránku prehľadu úloh prehľadná tabuľka, ktorá obsahuje základné informácie o úlohách ako názov, typ útoku, aktuálny stav úlohy, progres, čas pridania, hľadané heslo a tlačítka, ktoré spúšťajú operácie s úlohou. Príklad tejto stránky je možné vidieť na obrázku 5.7. Aby mala stránka rýchlu odozvu pri načítavaní úloh, implementoval som do nej systém stránkovania, kde načítavanie úloh prebieha po jednotlivých stránkach. Pri načítavaní novej stránky s úlohami sa posiela žiadosť na serverú časť aplikácie s parametrami **page** a **per_page**, ktoré označujú číslo stránky s úlohami a počet úloh na stránke. Vďaka stránkovaniu sa znižuje režia na prezentačnej vrstve (pretože nemusí vykreslovať a uchovávať si v pamäti všetky úlohy), na aplikačnej vrstve (pripravuje na odoslanie a prenáša menšie dáta) a aj na databázovej (SQL dotazy s veľkým počtom výsledkov bývajú pomalé).

The screenshot shows a web browser window with the Fitcrack application. The interface includes a search bar, a filter dropdown, and a table of tasks. The table has columns for Name, Attack type, Status, Progress, Added, Result, and Actions. The tasks listed are: test-stress-clients, Test-Retry-Package, package-stop-test, test_mask2, 10masks, DP-rules2, DP-rules2, DP-rules, test_mask_modified, and test_mask. The status of these tasks varies from 'ready' to 'finished' to 'exhausted'. The progress is shown as a percentage in a circular gauge. The 'Added' column shows the date and time. The 'Result' column shows the outcome of the task. The 'Actions' column contains three icons: a green play button, a blue refresh button, and a red stop button.

Name	Attack type	Status	Progress	Added	Result	Actions
test-stress-clients	mask	ready	5%	27.4.2018 14:14:49		
Test-Retry-Package	mask	ready	30%	27.4.2018 13:33:18		
package-stop-test	mask	ready	13%	27.4.2018 12:50:27		
test_mask2	mask	finished	100%	27.4.2018 12:07:59	aerotaner	
10masks	mask	finished	100%	27.4.2018 11:54:23	Not found	
DP-rules2	rules	ready	85%	27.4.2018 11:12:28		
DP-rules2	rules	finished	100%	27.4.2018 11:12:28	!d!!!d!!	
DP-rules	dictionary	exhausted	100%	27.4.2018 11:12:28		
test_mask_modified	markov	finished	100%	27.4.2018 10:36:08	aerotaner	
test_mask	mask	ready	0%	27.4.2018 10:34:19		

Jobs per page: 10 1-10 of 91

Obr. 5.7: Stránka s tabuľkou úloh

Tabuľka taktiež podporuje zoradenie položiek podľa všetkých stĺpcov a vyhľadávanie v úlohách, ale keďže je zavedený systém stránkovania a klientská časť nemá informácie o všetkých úlohách, zoradovanie a vyhľadávanie v úlohách vykonáva serverová časť ktorej sa odošle žiadosť s parametrami `order_by`, ktorý označuje vlastnosť podľa ktorej sa majú úlohy zoradiť, a `descending`, ktorý hovorí o smere zoradenia (vzostupne/zostupne).

Kapitola 6

Testovanie a experimenty

Testovanie je nezbytnou súčasťou vývoja akéhokoľvek projektu. V rámci mojej bakalárskej práce som uskutočnil dva typy testovania. Funkcionálne testovanie, ktoré malo za úlohu overiť funkčnosť systému, a testovanie užívateľského rozhrania, v ktorom som testoval ako rýchlo a efektívne užívateľ zvláda úlohy, a ako je pre užívateľa prostredie prívetivé.

6.1 Funkcionálne testovanie

Funkcionálne testovanie malo predovšetkým odhaliť chyby na aplikačnej vrstve. Testovanie prebehlo spôsobom posielania žiadostí na serverovú časť systému (REST API) a porovnávanie odpovedí servera so správnymi modelmi odpovedí. Týmto spôsobom som automaticky kontroloval syntaktickú správnosť odpovede, ktorá bola vo formáte JSON, a stavové kódy HTTP hlavičiek odpovedí. Toto testovanie odhalilo niekoľko chýb s oprávneniami užívateľov, ktoré boli opravené. Ďalej som cez REST API posielal na server žiadosti, ktoré vytvorili úlohy. Po načítaní detailu úlohy vo formáte JSON som kontroloval správne vypočítanú veľkosť množiny hesiel, priradených hostov, určený typ hashu a ďalšie hodnoty. Tieto hodnoty som následne porovna s hodnotami z databázy, pričom som zistil že sa rovnajú.

6.2 Užívateľské testovanie

Ďalšou fázou testovania bolo testovanie užívateľom. Pre tento typ testovania bolo pozvaných 5 osôb, ktorých úlohou bolo vykonať niekoľko jednoduchých úloh. Jednalo sa o úlohy:

- prihlásenie
- vytvorenie nového užívateľa
- zobrazenie všetkých úloh
- nahranie slovníka do systému
- vytvorenie ľubovolnej úlohy
- spustenie úlohy
- sledovanie progresu úlohy
- zhodnotiť výsledok úlohy

Behom testovania nevznikli u testujúcich takmer žiadne problémy. Keďže sa jedná o pomerne zložitý systém, ktorý si vyžaduje aspoň minimálne znalosti o kryptografickom šifrovaní, nie všetkým testovaným boli princípy jednotlivých útokov jasné. Avšak aj napriek tomu testujúci splnili úlohy za veľmi krátku dobu. Užívateľské prostredie sa všetkým javilo ako jednoduché a intuitívne. Následne, vzhľadom na výsledky testovania som uskutočnil pár grafických zmien, ako napríklad zväčšenie a zvýraznenie niektorých ovládacích prvkov. Taktiež sa väčšine testujúcim páčil grafický dizajn a zvolené farby aplikácie.

6.3 Experiment

Cielom experimentu je otestovať funkčnosť celého systému. Pomocou jednoduchého skriptu napísaného v jazyku Python som vygeneroval slovník, ktorý obsahuje 50 miliónov hesiel. Heslá sú tvorené piatimi znakmi anglickej abecedy. Na konci slovníka som pridal heslo `experiment123`. Obsah slovníka je naznačený na výpise 6.1. Slovník zaberá po vygenerovaní 341 797kB diskového priestoru.

```
1.      aaaaa
2.      aaaab
3.      aaaac
4.      aaaad
      .
      .
      .
49999999. gRFgw
50000000. gRFgx
50000001. experiment123
```

Obr. 6.1: Obsah experimentálneho slovníka

Slovník som nahral prostredníctvom webovej aplikácie do systému Fitrack a pomenoval som ho `experiment.txt`. Overil som pridanie záznamu do databázy a vypočítanie správnej veľkosti množiny hesiel, ktorú slovník obsahuje. Potom som prostredníctvom programu WinRar¹ zkomprimoval a zašifroval skupinu súborov do archívu `faktury.zip`. Ako heslo som zadal `experiment123`. Následne som do systému pripojil dva uzly a vytvoril úlohu do ktorej som nahral zašifrovaný archív a vybral vygenerovaný slovník. Typ hashu sa správne automaticky nastavil. K tejto úlohe som pripojil dva uzly. Systém odhadol dobu výpočtu na 21 minút a 46 sekúnd. Stránku s vytváraním úlohy je možné vidieť na obrázku 6.2.

¹<https://www.win-rar.com/>

Estimated cracking time is 0:21:46.

Create new job

Name:

Comment:

Input settings

Hashtype:

Upload method: ☒ Encrypted file ☐ Hash text ☐ Multiple hashes

[UPLOAD FILE](#)

faktury.zip success

Attack settings

[DICTIONARY ATTACK](#)
[COMBINATION ATTACK](#)
[BRUTE-FORCE ATTACK](#)
[HYBRID W](#)




Select dictionary *

Name	Keyspace	Time
myspaceModified.txt	37142	Yesterday at 9:37 AM
top1000.txt	1000	Yesterday at 9:37 AM
phpbbMod.txt	184388	Yesterday at 10:59 AM
experiment.txt	50000001	Today at 4:00 PM

Dictionaries per page 5 6-9 of 9

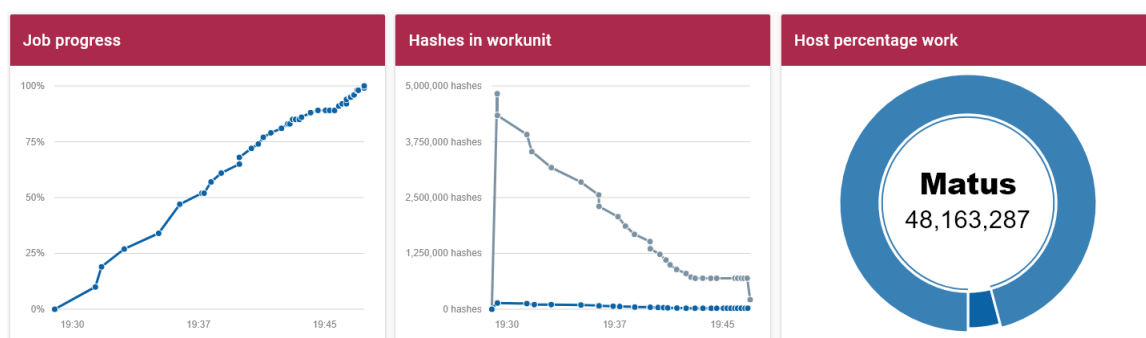
Obr. 6.2: Vytváranie experimentálnej úlohy

Po spustení úlohy sa vyťaženie uzlov prudko zdvihlo. Jeden host svojim výkonom výrazne prevyšoval druhého. Po približne pol hodine sa zobrazila notifikácia o dokončení úlohy. Ako vidno na obrázku 6.3, Fitcrack našiel heslo k archívu za 28 minút a 52 sekúnd.

Matus_experiment		
Operations:		  
Comment:		
Job keyspace:	50000001	
Status:	finished	
Hash type:	Win Zip	
Password:	experiment123	
Added:	Today at 7:25 PM	
Cracking time:	28:52	
Progress:	<div><div></div></div> 100%	
Start time:	Today at 7:25 PM	
End time:	Today at 7:53 PM	
Seconds per workunit:	60	

Obr. 6.3: Vytváranie experimentálnej úlohy

Za pozornosť stoja aj grafy, ktoré je možné vidieť na obrázku 6.4. Koláčový graf podporuje predpoklad o tom, že jeden host je výrazne výkonnejší ako druhý. Na grafe s progresom úlohy vidíme že progres stúpal približne lineárne.



Obr. 6.4: Vytváranie experimentálnej úlohy

Kapitola 7

Záver

V tejto práci sa mi podarilo navrhnuť implementáciu serverovej časti systému Fitcrack s automaticky generovanou dokumentáciou. Zameral som sa na vylepšenia súčasnej verzie webového prostredia Fitcracku a navrhol som rozšírenia, ku príkladu systém viacerých užívateľov, ktoré spríjemnia užívateľom používanie. Taktiež som odstránil nedokonalosti súčasného riešenia, ako napríklad dostupnosť len z webovej platformy. Systém, implementovaný podľa môjho návrhu bude dostupný zo všetkých platforiem, ktoré sú schopné sieťovej komunikácie. Súčasne sa zníži záťaž na server, pretože bude prerozdelená medzi klienta a server. Vďaka implementácii pomocou modulov bude celý systém ľahko rozšíriteľný. Návrh opísaný v tejto práci tiež ošetruje niekoľko bezpečnostných dier súčasnej implementácie. Návrh spĺňa všetky zásady architektúry REST (viď 3.2.2), a taktiež je vhodný na testovacie účely systému.

Keďže som v rámci tejto práce zhromaždil potrebné znalosti, v nasledujúcom semestri budem pracovať na implementácii navrhnutého systému. Tiež sa budem venovať testovaniu spoľahlivosti a experimentom. Systém podrobím výkonnostným testom a výsledky porovnam so súčasným riešením. Mojou snahou je vytvoriť aplikačné rozhranie na vzdialené ovládanie systému Fitcrack takým spôsobom, aby implementácia klientskej časti užívateľského rozhrania bola jednoduchá a vyžadovala čo najmenej informácií o štruktúre serverovej časti.

Literatúra

- [1] Anderson, D. P.: BOINC: a system for public-resource computing and storage. In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, Nov 2004, ISSN 1550-5510, s. 4–10.
- [2] Berners-Lee, T.; Fielding, R. T.; Masinter, L.: Uniform Resource Identifier (URI): Generic Syntax. STD 66, RFC Editor, January 2005,
<http://www.rfc-editor.org/rfc/rfc3986.txt>.
URL <http://www.rfc-editor.org/rfc/rfc3986.txt>
- [3] Fielding, R. T.: *Architectural Styles and the Design of Network-based Software Architectures*. 2000, University of California, Irvine Doctoral dissertation, ISBN: 0-599-87118-0.
- [4] Fielding, R. T.; Gettys, J.; Mogul, J. C.; aj.: Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, RFC Editor, June 1999.
URL <http://www.rfc-editor.org/rfc/rfc2616.txt>
- [5] Hranický, R.; Zobal, L.; Večeřa, V.: Distribuovaná obnova hesel. Technická Zpráva FIT-TR-2017-04, CZ, 2017.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=11568
- [6] Rescorla, E.: HTTP Over TLS. RFC 2818, RFC Editor, May 2000.
URL <http://www.rfc-editor.org/rfc/rfc2818.txt>