



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 návrh a implementáciu webovej aplikácie na vzdialenú správu systému Fitcrack. Jedná sa o systém, ktorý slúži na distribuovanú obnovu hesiel. Serverová časť webovej aplikácie je navrhnutá s použitím architektúry REST, ktorá znižuje záťaž na server a zároveň zjednodušuje testovanie systému. Prezentačná vrstva systému je implementovaná ako moderná jednostránková webová prezentácia (Single page application). So serverovou časťou komunikuje pomocou asynchrónnych žiadostí (AJAX) ktorých obsah je vo formáte JSON. Súčasťou webovej aplikácie je príjemné responzívne užívateľské rozhranie, systém užívateľov s oprávneniami a rôzne pokročilejšie grafické prvky ako napríklad grafy.

Abstract

Anglický abstrakt doplním :)

Kľúčové slová

REST, Fitcrack, API, SPA

Keywords

REST, Fitcrack, API, SPA

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 túto bakalársku prácu som vypracoval samostatne pod vedením pána inžiniera Radka Hranického. Uviedol som všetky literárne zdroje a publikácie, z ktorých som čerpal.

.....

Matúš Múčka

7. mája 2018

Podakovanie

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

Obsah

1	Úvod	2
2	Systém Fitcrack	3
2.1	Hashcat	3
2.2	BOINC	3
2.3	Architektúra systému Fitcrack	4
3	Pôvodné riešenie webovej aplikácie systému Fitcrack	6
4	Popis použitých technológií	8
4.1	Hypertext Transfer Protocol (HTTP)	8
4.2	Architektúra REST	10
4.3	Python	12
4.4	Vue	13
5	Návrh systému	15
5.1	Návrh serverovej časti	15
5.2	Návrh klientskej časti	19
5.3	Typy útokov v systéme Fitcrack	20
5.4	Grafy	22
5.5	Notifikácie	22
6	Implementácia	24
6.1	Implementácia databázovej vrstvy	25
6.2	Implementácia aplikačnej vrstvy	26
6.3	Implementácia prezentačnej vrstvy	30
7	Testovanie a experimenty	37
7.1	Funkcionálne testovanie	37
7.2	Užívateľské testovanie	37
7.3	Experiment	38
8	Záver	41
8.1	Výhľad do budúcnosti	41
	Literatúra	42

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 webovej aplikácie, ktorej serverová časť spĺňa všetky zásady architektúry REST (viď 4.2.2). Táto aplikácia slúži na vzdialenú správu systému Fitcrack. Jedná sa o výkonný systém na obnovu hesiel, ktorý prerozdeľuje prácu medzi viacerými pripojenými klientmi, a tým pádom zvyšuje výpočtnú 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 ôsmich 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 4. Samotným návrhom riešenia, ktorý je rozdelený do modulov, sa zaoberám v kapitole 5. V jednotlivých podkapitolách popisujem každý modul. S implementáciou aplikácie sa zaoberám v kapitole 6. V kapitole 7 som systém otestoval a podrobil experimentu. Nakoniec v záverečnej kapitole 8 hodnotím výsledky mojej práce.

Kapitola 2

System Fitcrack

Fitcrack je systém, ktorý slúži na obnovu hesiel z kryptografického heša, alebo z niekoľkých podporovaných typov súborov (napríklad zip, pdf, rar atď.). 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. [9]

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 dlhšie úlohy medzi pripojených klientov, na ktorých prebieha pokus o nájdenie hesla pomocou nástroja Hashcat.

Systém BOINC tvorí server a klienti. Pri distribuovaní výpočtových 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 výpočtovej úlohy ju spracuje a výsledok odošle na serverom. Následne žiada o pridelenie novej úlohy. [1]

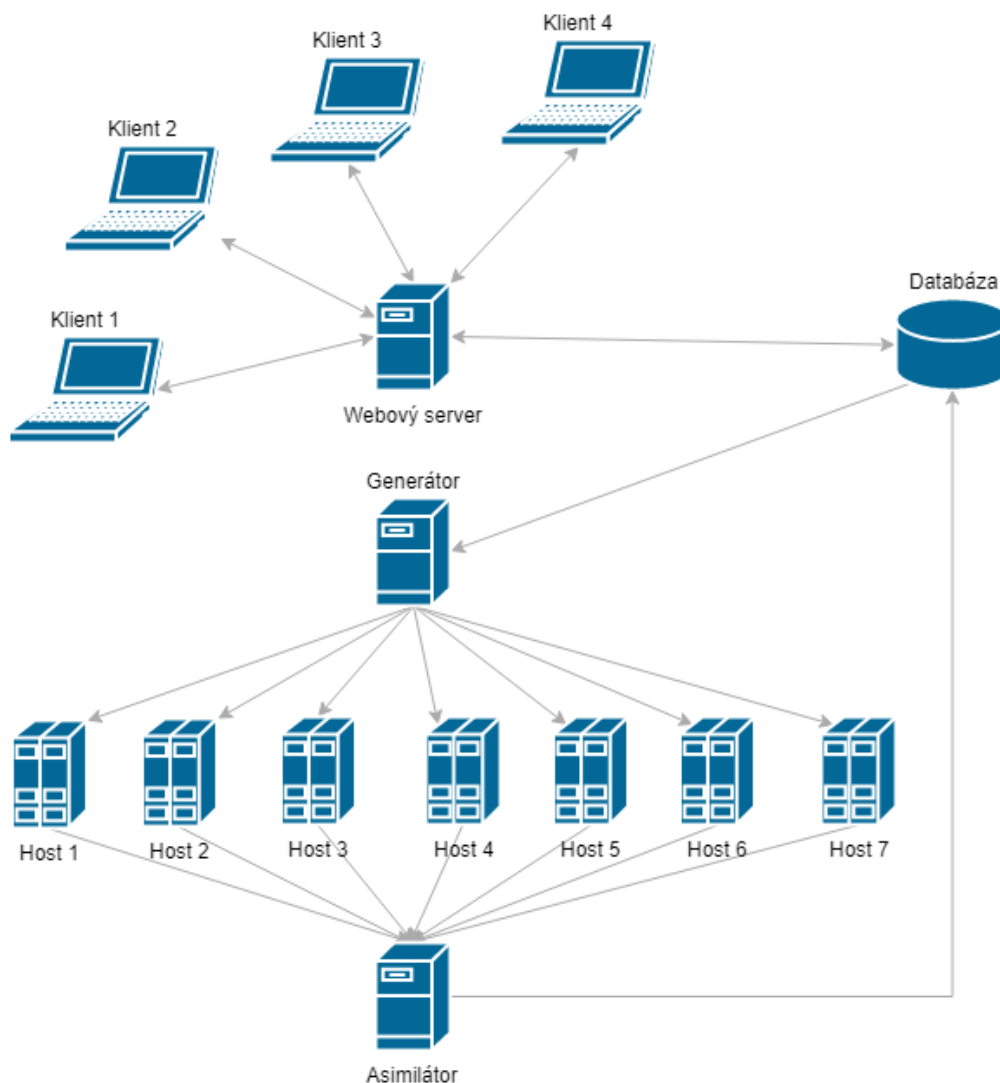
¹<https://hashcat.net>

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

³<https://boinc.berkeley.edu/>

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.



Obr. 2.1: Architektúra systému Fitcrack

2.3.1 Webový server

Cieľom mojej bakalárskej práce je navrhnúť a implementovať webový server, cez ktorý budú môcť užívatelia spravovať celý systém Fitcrack. K webovému serveru môže byť naraz pripojených viacero užívateľov, ktorí vykonávajú rôzne činnosti (analyzovanie výsledkov dokončených úloh, sledovanie aktivity jednotlivých výpočtových uzlov, 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 4.2). Prostredníctvom webovej aplikácie je možné do systému pridávať nové úlohy, monitorovať prebiehajúce, analyzovať dokončené

úlohy, monitorovať správanie výpočtových uzlov a podobne. Viac o návrhu webového servera sa je možné dočítať v kapitole 5.

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 výpočtové uzly priradené 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 výpočtovým uzlom, ktoré sa práve do systému alebo k úlohe pripojili, poprípade uzlom, u ktorých nastala chyba. Slúžia k na meranie výkonu daného uzla. Meranie výkonu má veľký význam pri generovaní normálnych pracovných jednotiek, pretože vďaka nameraným údajom vieme odhadnúť správne množstvo práce pre výpočtový uzol. 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 výpočtového uzla. 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 jeden výpočtový uzol (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 výpočtových uzlov. 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 uzol našiel heslo, ktorého kryptografický heš je rovnaký ako hľadaný heš, asimilátor na základe tejto odpovede upraví databázu tak, že nastaví stav úlohy na dokončený. Vďaka tejto úprave ostatné výpočtové uzly podieľajúci sa na danej úlohe, dostanú správu o zastavení výpočtu.

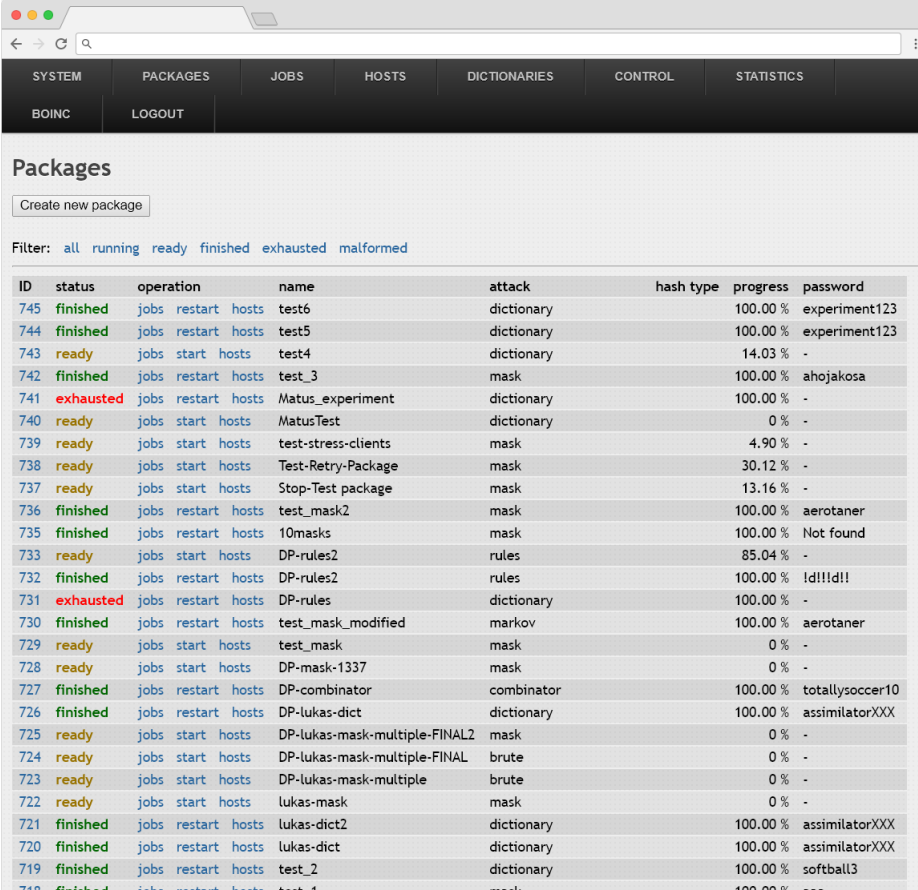
2.3.4 Výpočtový uzol

Výpočtový uzol je fyzický klient pripojený k systému Fitcrack. Na výpočtových uzloch beží aplikácia Runner, ktorá prijíma úlohy od Generátora. Samotný výpočet potom riadi Runner a prebieha pomocou nástroja Hashcat (viď 2.1).

Kapitola 3

Pôvodné riešenie webovej aplikácie systému Fitcrack

Pre systém Fitcrack už bola v minulosti vyvinutá webová aplikácia, ktorá slúžila ako prototyp. Jej webové rozhranie je možné vidieť na obrázkoch 3.1 a 3.2. Táto aplikácia bola napísaná v jazykoch PHP, javascript, HTML a CSS.

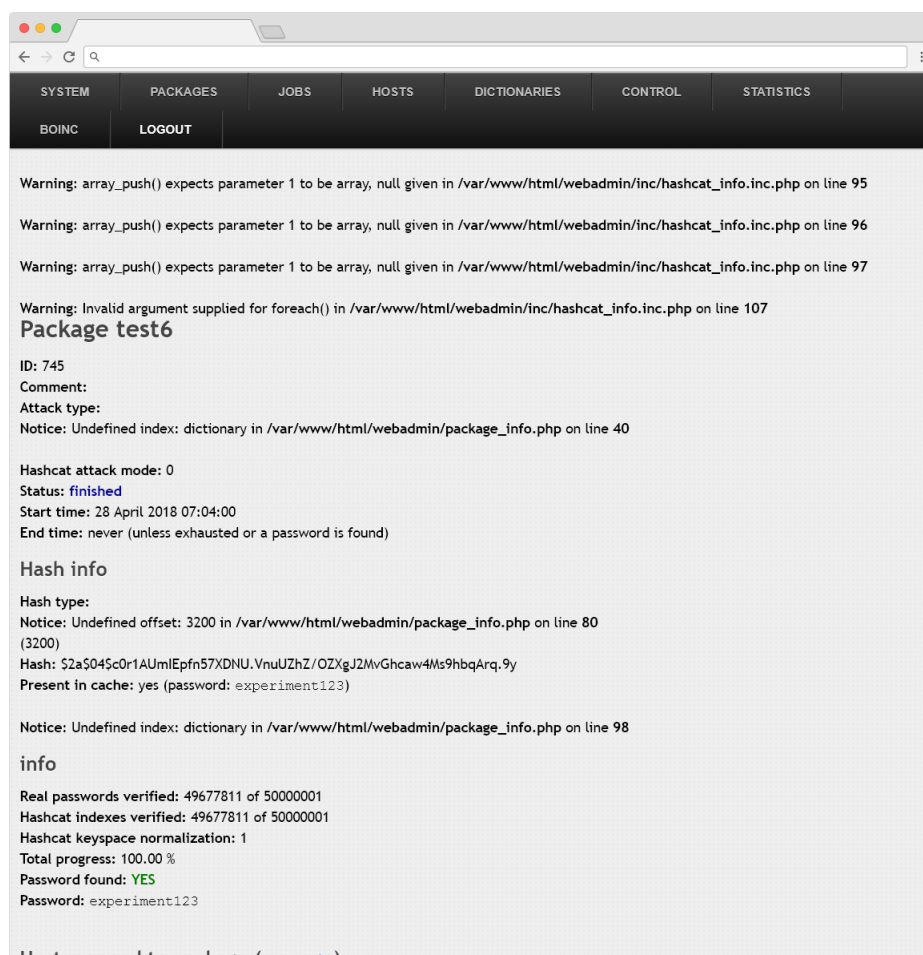


The screenshot shows a web browser window displaying the Fitcrack web application. The interface has a dark navigation bar with tabs: SYSTEM, PACKAGES, JOBS, HOSTS, DICTIONARIES, CONTROL, and STATISTICS. Below the navigation bar, there are buttons for BOINC and LOGOUT. The main content area is titled 'Packages' and includes a 'Create new package' button. A filter bar shows 'all' selected, with other options: running, ready, finished, exhausted, and malformed. Below the filter is a table with columns: ID, status, operation, name, attack, hash type, progress, and password. The table lists various packages with their current status and progress.

ID	status	operation	name	attack	hash type	progress	password
745	finished	jobs restart hosts	test6	dictionary		100.00 %	experiment123
744	finished	jobs restart hosts	test5	dictionary		100.00 %	experiment123
743	ready	jobs start hosts	test4	dictionary		14.03 %	-
742	finished	jobs restart hosts	test_3	mask		100.00 %	ahojakosa
741	exhausted	jobs restart hosts	Matus_experiment	dictionary		100.00 %	-
740	ready	jobs start hosts	MatusTest	dictionary		0 %	-
739	ready	jobs start hosts	test-stress-clients	mask		4.90 %	-
738	ready	jobs start hosts	Test-Retry-Package	mask		30.12 %	-
737	ready	jobs start hosts	Stop-Test package	mask		13.16 %	-
736	finished	jobs restart hosts	test_mask2	mask		100.00 %	aerotaner
735	finished	jobs restart hosts	10masks	mask		100.00 %	Not found
733	ready	jobs start hosts	DP-rules2	rules		85.04 %	-
732	finished	jobs restart hosts	DP-rules2	rules		100.00 %	ld!!!d!!
731	exhausted	jobs restart hosts	DP-rules	dictionary		100.00 %	-
730	finished	jobs restart hosts	test_mask_modified	markov		100.00 %	aerotaner
729	ready	jobs start hosts	test_mask	mask		0 %	-
728	ready	jobs start hosts	DP-mask-1337	mask		0 %	-
727	finished	jobs restart hosts	DP-combinator	combinator		100.00 %	totallysoccer10
726	finished	jobs restart hosts	DP-lukas-dict	dictionary		100.00 %	assimilatorXXX
725	ready	jobs start hosts	DP-lukas-mask-multiple-FINAL2	mask		0 %	-
724	ready	jobs start hosts	DP-lukas-mask-multiple-FINAL	brute		0 %	-
723	ready	jobs start hosts	DP-lukas-mask-multiple	brute		0 %	-
722	ready	jobs start hosts	lukas-mask	mask		0 %	-
721	finished	jobs restart hosts	lukas-dict2	dictionary		100.00 %	assimilatorXXX
720	finished	jobs restart hosts	lukas-dict	dictionary		100.00 %	assimilatorXXX
719	finished	jobs restart hosts	test_2	dictionary		100.00 %	softball3
718	finished	jobs restart hosts	test_1	mask		100.00 %	aaa

Obr. 3.1: Pôvodná webová aplikácia

Aplikácia mala niekoľko závažných problémov. To bol dôvod prečo som sa rozhodol nepokračovať vo vývoji tohto prototypu, ale začať od začiatku s novým riešením. Keďže pôvodná aplikácia nemala od seba oddelenú serverovú a klientskú časť aplikácie, testovanie aplikačnej logiky sa vykonávalo len veľmi ťažko. Server užívateľovi posielal vygenerované HTML dokumenty, ktoré v porovnaní s jednostránkovými webovými prezentáciami¹ (single page application) zvyšujú záťaž na serverovú časť a predlžujú dobu odozvy u užívateľa. Tak tiež sa s pôvodným riešením nedalo jednoducho komunikovať prostredníctvom aplikačného rozhrania z iných platforiem ako webu. To znemožnilo písanie automatizovaných skriptov. Ďalším nedostatkom pôvodného riešenia bola podpora len jedného užívateľa, ktorý nebol uložený v databáze, ale prihlasovacie údaje sa ukladali priamo do zdrojového kódu aplikácie. Veľkým nedostatkom systému bola podpora len dvoch útokov. Podporovaný bol slovníkový útok, avšak bez použitia pravidiel, a útok s maskami bez použitia markovského modelu. Pôvodná webová aplikácia nebola responzívna, užívateľ s menším displejom mal s jej používaním problém. Jednotlivé stránky sa po načítaní samovoľne neaktualizujú, takže sa zmeny v databáze neprejavia na prezentačnej vrstve bez znovunačítania stránky. V riešení chýbajú interaktívne prvky ako grafy a podobne. Celkovo aplikácia pôsobí zastaralo.



Obr. 3.2: Pôvodná webová aplikácia

¹https://en.wikipedia.org/wiki/Single-page_application

Kapitola 4

Popis použitých technológií

4.1 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 [6].

4.1.1 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.

4.1.2 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 [11].

4.1.3 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 počas 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[12].

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).

4.1.4 Druhy žiadostí/metódy HTTP

Protokol HTTP využíva niekoľko žiadostí, z ktorých najčastejšie sú:

- **GET** - ide o najbežnejší typ žiadostí. Jej výsledkom je žiadaný zdroj uvedený 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 uvedený 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.

4.1.5 Príklad komunikácie

Klient začína komunikáciu poslaním dotazu na server. Na výpise 4.1 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 4.1: príklad HTTP dotazu

Príklad nasledovnej reakcie servera na dotaz sa nachádza na výpise 4.2. Server odpovedá stavovým kódom 200 OK, čo značí, že dotaz sa podarilo úspešne spracovať. Ďalej hlavička odpovede 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; charset=UTF-8
Cache-control: private
Connection: keep-alive
```

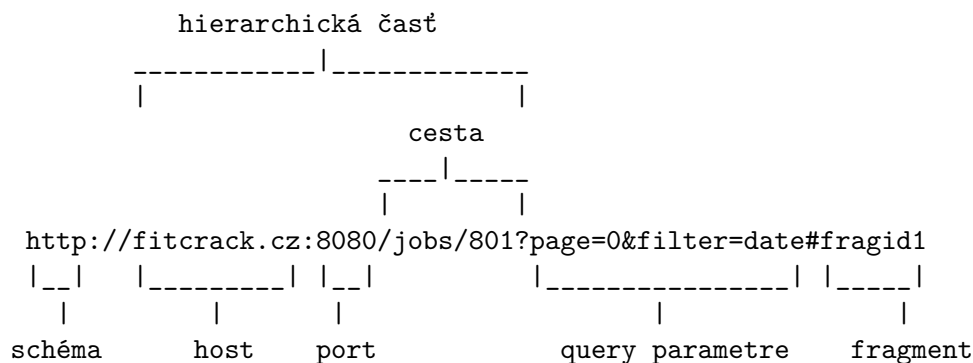
Výpis 4.2: príklad HTTP odpovedi

4.2 Architektúra REST

REST je úzko spojený s protokolom HTTP, keďže ho prvýkrát navrhol a popísal Roy Fielding - jeden zo spoluautorov protokolu HTTP, v rámci jeho dizertačnej práce v roku 2000 [5]. 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. 4.2.1). Každý zdroj musí mať vlastný identifikátor URI.

4.2.1 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 4.1. 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. 4.1: Formát URI pri použití schémy HTTP

4.2.2 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 [10].

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 obsluhovať 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 tieto dáta z medzipamäti. Tým pádom nezafarňuje 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).

4.2.3 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 kolekciu, 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.

4.3 Python

Python¹ je interpretovaný programovací jazyk, ktorý podporuje objektovo orientované, štruktúrované aj funkcionálne programovanie. Ponúka dynamickú typovú kontrolu. Je vyvíjaný ako open source projekt. V dôsledku nedostatkov a chýb pri návrhu jazyka sa v roku 2008 rozdelil vývoj do dvoch častí (Python 2.x a Python 3.x). Verzia 3.x opravuje nedostatky ktoré v sebe mala verzia 2.x a nedali sa odstrániť bez toho, aby došlo k výrazným zásahom do kompatibility. Obe verzie sú v súčasnej dobe vyvíjané a podporované vývojármi.

4.3.1 SQLAlchemy

SQLAlchemy² je knižnica, napísaná v jazyku Python, ktorá slúži na vytvorenie abstraktnej vrstvy nad databázou. Jednotlivé databázové tabuľky sa namapujú na triedy, s ktorými je potom možné robiť operácie filtrovanie, pridávanie a mazanie. Tie zmeny sa následne prejavia v databáze [4].

¹<https://www.python.org/>

²<http://www.sqlalchemy.org/>

4.3.2 Flask

Flask³ je miniatúrny webový framework napísaný v jazyku Python. Narozdiel od svojho najväčšieho konkurenta - Django⁴, Flask v sebe obsahuje len minimum vstavaných funkcií. Napríklad Flask nemá vstavanú abstraktnú databázovú vrstvu ani validáciu formulárov. Avšak Flask podporuje množstvo rozšírení (modulov), ktoré jeho funkčnosť rozširujú [8]. Príklad stránky, ktorá po navštívení vypíše text **Ahoj svet!**, je možný vidieť na výpise 1

Algoritmus 1 Príklad jednoduchej aplikácie s použitím frameworku Flask

```
1 from flask import Flask
2 app = Flask(__name__)
3
4 @app.route("/")
5 def hello():
6     return "Hello World!"
7
8 if __name__ == "__main__":
9     app.run()
```

Flask-RESTPlus

Flask-RESTPlus⁵ rozšírenie pre framework Flask, ktoré zjednodušuje vývoj aplikačného rozhrania s architektúrou REST. Do aplikácie pridáva sadu dekorátorov a nástrojov, a tiež uľahčuje tvorbu dokumentácie.

Flask-Login

Flask-Login⁶ je ďalšie rozšírenie do frameworku Flask, ktoré poskytuje správu sedení užívateľov. Pridáva do systému sadu funkcií, ktorými je možné implementovať prihlasovanie a odhlasovanie užívateľov, zapamätanie si užívateľa a zabezpečenie zdrojov pred užívateľom, ktorý na ne nemá práva.

4.4 Vue

Vue⁷ je voľne šíriteľný framework napísaný v Javascipte, určený na tvorbu užívateľských rozhraní. Zvyčajne sa používa na vývoj jednostránkových aplikácií. Vyznačuje sa svojou nízkou veľkosťou a jednoduchosťou. Do aplikácií používajúce Vue je jednoduché integrovať iné knižnice. Vue taktiež obsahuje množstvo modulov, ktoré rozširujú možnosti aplikácie. Pri väčších aplikáciach je nutné jednotlivé časti rozdeliť do logických celkov - komponentov. Komponent je znovupoužiteľná časť aplikácie, ktorá v sebe obsahuje šablónu v syntaxe HTML, svoju javascriptovú logiku a kaskádové štýly v jazyku CSS. Jednou z výhod Vue je veľmi jednoduchá a intuitívna správa stavu aplikácie vďaka reaktivite, ktorú poskytuje. Pri zmene javascriptových objektov, ktoré sa používajú na vykreslenie komponentu, dôjde

³<http://flask.pocoo.org/>

⁴<https://www.djangoproject.com/>

⁵<http://flask-restplus.readthedocs.io>

⁶<https://flask-login.readthedocs.io>

⁷<https://vuejs.org/>

automaticky k znovu vykresleniu daného komponentu, bez nutnosti vyžadovania prekreslenia manuálne. Každý komponent si vytvorí zoznam závislostí pri prvom vykreslení, takže systém môže jednoducho určiť, kedy je komponent potrebné prekresliť.

4.4.1 Vue CLI

Vue CLI⁸ je jednoduché terminálové rozhranie pre vytváranie nových Vue aplikácií. Cez toto rozhranie je možné vytvoriť aplikáciu podľa šablóny, vyvíjať ju na lokálnom webovom serveri, ktorý má podporu automatického znovunačítania aplikácie po zmene zdrojového súboru (takzvaný hot-reloading), inštalovať do aplikácie rôzne moduly a zostaviť produkčnú verziu aplikácie, ktorá obsahuje minifikovaný zdrojový kód. Príklad práce s Vue CLI je možné vidieť na výpise 4.3. Pre správne fungovanie Vue CLI je nutné mať nainštalovaný node.js⁹.

```
1 # inicializacia aplikacie pomocou sablony webpack
2 vue init webpack fitcrackFE
3 cd fitcrackFE
4 # doinstalovanie zavislosti
5 npm install
6 # spustenie vyvojoveho servera
7 npm run dev
8 # zostavenie produkcej verzie
9 npm run build
```

Výpis 4.3: Príklad práce s Vue CLI

4.4.2 Vue router

Kedže Vue väčšinou slúži na vývoj jednostránkových webových prezentácií, je nutné pri aplikáciách vyriešiť URL adresáciu. Vue router¹⁰ je modul pre framework Vue, ktorý má na starosti navigáciu v aplikácií. Vue router používa konfiguračný súbor, v ktorom sú popísané jednotlivé cesty. Pri každej ceste je uvedený komponent, ktorý je pri zhode cesty vykreslený. Taktiež tento modul pridáva do aplikácie globálny objekt **router**, cez ktorý je možné navigáciu riadiť z javascriptového rozhrania.

4.4.3 JSON

JSON (JavaScript Object Notation) je odľahčený formát, ktorý slúži na výmenu dát. Je založený na podmnožine jazyka javascript. Pre jeho jednoduchú čitateľnosť a ľahkú strojovú generovateľnosť ho v podstate všetky moderné programovacie jazyky v nejakej forme podporujú. JSON je textový a na jazyku nezávislý formát. Pri jednostránkových webových aplikáciach sa používa ako spôsob komunikácie medzi prezentačnou a aplikačnou vrstvou[3].

⁸<https://vuejs.org/2015/12/28/vue-cli/>

⁹<https://nodejs.org>

¹⁰<https://router.vuejs.org>

Kapitola 5

Návrh systému

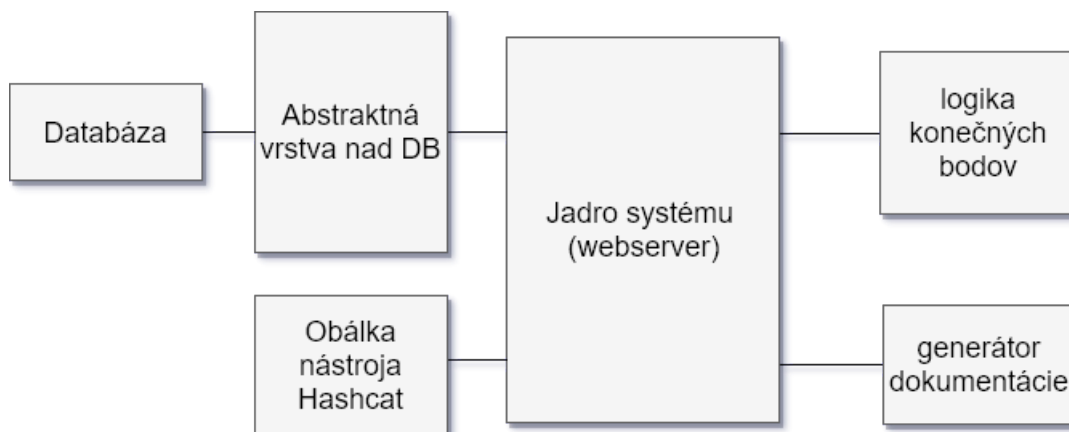
V tejto kapitole je popísaný koncepčný návrh systému. Kvôli ľahšej rozširiteľnosti a jednoduchšiemu testovaniu, je vhodné od seba klientskú a serverovú časť systému oddeliť. Súčasťou tohto návrhu je návrh databázy, užívateľského rozhrania, typov útokov v systéme Fitcrack a návrh niekoľkých rozšírení do systému.

5.1 Návrh serverovej časti

Serverová časť aplikácie má za úlohu sprostredkovať komunikáciu medzi databázou a prezentačnou vrstvou. Funguje ako webový server, ktorý na žiadosti prezentačnej vrstvy odpovedá HTTP správou, so správne nastaveným stavovým kódom v hlavičke a telom správy vo formáte JSON^{4.4.3}.

5.1.1 Rozdelenie systému na moduly

Pre lepšiu organizáciu som celý systém rozdelil do niekoľko modulov (viď obr. 5.1). Jednotlivé moduly sú detailnejšie rozobrané v nasledujúcich podkapitolách.



Obr. 5.1: 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 nahrávanie súborov, cesta k nástroju Hashcat atď.). Pri zlyhaní jadra webový server odpovedá HTTP správy s kódom začínajúcim číslom 5 (viď 4.1.3).

Logika konečných bodov

Jedná sa o modul, v ktorom sú obslužené dotazy na konkrétne URI (viď 4.2.1). Zoznam dostupných URI je uvedený v tabuľke 5.1. Väčšina konečných bodov podporuje viac typov HTTP metód (viď 4.2.3). 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 4.1.3). 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 dostatočujúce 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 4.4.3, ktorú pridá do tela HTTP odpovede s kódom 200 (viď 4.1.3).

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 5.2 je možné vidieť entitno relačný diagram upravenej databázy. Zelenou farbou sú na ňom označené tabuľky, ktoré som do systému pridal ja. Bez nich by nebolo možné implementovať všetky rozšírenia.

Generátor dokumentácie

Generátor dokumentácie po spustení webového servera spracuje zdrojové kódy aj s komentármi a vytvorí pomocou nich interaktívnu dokumentáciu (viď obr. 5.3). Z nej je potom možné zistiť všetky dostupné koncové body spolu s príkladmi odpovede. Generovanie dokumentácie automaticky nastáva aj pri zmene zdrojových súborov.

The screenshot displays the Fitcrack API documentation. At the top, it says "Fitcrack API" and "hashcat : Endpointy ktoré priamo využívajú nástroj hashcat". Below this, there are two main sections. The first section is for the "/hashcat/attackModes" endpoint, which is a GET request and returns a list of supported attacks. The second section is for the "/hashcat/hashTypes" endpoint, which is a GET request and returns a list of supported hash types. This section is expanded to show a "Response Class (Status 200)" with a "Success" status. It includes a "Model Schema" tab showing a JSON response structure:

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

. Below the JSON, there is a "Response Content Type" dropdown set to "application/json" and a "Try it out!" button. The bottom section of the image shows two more endpoints: "/hosts/" (GET, returns a list of hosts) and "/packages/" (GET, returns a list of packages). Below these, there is a "packages : Operácie s package" section with a "POST /packages/" endpoint that creates a new package.

Obr. 5.3: Ukážka vygenerovanej dokumentácie

5.1.2 Autentifikácia a oprávnenia užívateľov

Súčasná verzia systému Fitcrack nepodporuje viacerých užívateľov. Aby bolo možné do systému pridať užívateľov, je nutné rozšíriť databázu (viď. 5.1.1). Ako spôsob riešenia právomocí užívateľom som vybral systém oprávnení na základe rolí. Každý užívateľ bude mať uvedenú rolu, ktorá mu udeľuje právomoci na určité akcie. Jedná sa o jednoduchý systém, plne postačujúci pre menší počet užívateľov. Jednotlivé právomoci sú nasledujúce.

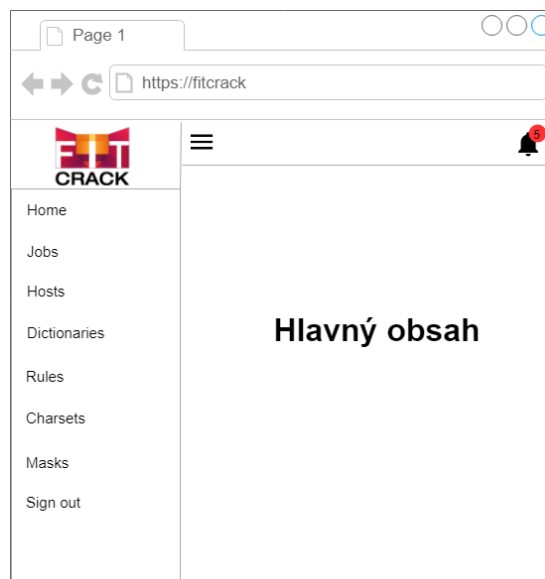
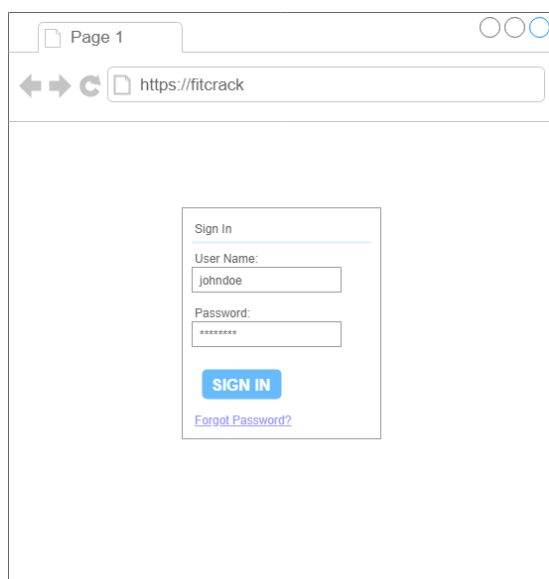
- **Spravovanie užívateľov** - S takýmto oprávnením môže užívateľ vytvárať, mazať a modifikovať iných užívateľov.
- **Pridávanie novej úlohy** - Užívateľ môže pridať do systému novú úlohu

- **Nahrávanie súborov** - Užívateľ má oprávnenie do systému nahrávať 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.

5.2 Návrh klientskej časti

Klientskú časť tvorí moderná webová aplikácia, ktorá pomocou HTTP žiadostí komunikuje so serverom (viď 4.1). 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 5.4). 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. 5.4: Návrh úvodnej stránky aplikácie. Obr. 5.5: 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 5.5.

5.3 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.

5.3.1 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 Markovovými 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 5.6. 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 Markovovho 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 Markovovho modelu sa majú heslá generovať [7].

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. 5.6: Príklad Markovovho modelu

5.3.2 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 písmená 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.

5.3.3 Kombinačný útok

Pri kombinačnom útoku je nutné vybrať dva slovníky (ľavý a pravý). Každé heslo z ľavého slovníka sa spojí 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

5.3.4 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.

5.4 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 os bude v percentách v rozmedzí 0 až 100. Na X-ovej osi 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 uzlov** - jedná sa taktiež o spojnicový graf, na ktorom je možné sledovať aktivitu jednotlivých výpočtových uzlov. Na Y-ovej osi je možné vidieť počet kryptografických hešov, ktoré uzol vypočítal v rámci pracovnej jednotky. Na X-ovej osi sa nachádza čas. Z tohto grafu by mala byť zrejماً aktivita Asimilátora (viď [2.3.3](#)), ktorý prispôsobuje počet hesiel v jednej pracovnej úlohe pre uzol. Tento graf je možné zobrazovať v rámci jednej úlohy, alebo v rámci systému.
- **Graf práce uzlov** - ide o koláčový graf, ktorý je možné zobrazovať pre konkrétnu úlohu. Je z neho možné vyčítať množstvo práce jednotlivých výpočtových uzlov v rámci úlohy.

5.5 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 znakovkej sady	/charset/<id>/download
Operácie s kolekciou slovníkov	/dictionary/
Operácie s konkrétnym slovníkom	/dictionary/<id>
Dáta na vykreslenie grafu podielu práce uzlov	/graph/hostPercentage/<job_id>
Dáta na vykreslenie grafu vyťažnosti uzlov	/graph/hostsComputing
Dáta na vykreslenie grafu vyťažnosti uzla	/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 hešov podporované nástrojom hashcat	/hashcat/hashTypes
Operácie s kolekciou výpočtových uzlov	/hosts/
Operácie s konkrétnym uzlom	/hosts/<id>
Získanie informácií o uzloch	/hosts/info
Operácie s kolekciou úloh	/job/
Operácie s konkrétnou úlohou	/job/<id>
Štart/stop/reštart úlohy	/job/<id>/action
Operácie s uzlami, ktoré sa podielajú na úlohe	/job/<id>/host
Získanie pracovných jednotiek úlohy	/job/<id>/workunit
Výpočet odhadu doby trvania úlohy	/job/crackingTime
Získanie informácií o úlohách	/job/info
Overenie formátu hešu	/job/verifyHash
Operácie s kolekciou Markovských modelov	/markovChains/
Operácie s konkrétnym Markovským modelom	/markovChains/<id>
Vytvorenie Markovského modelu zo slovníka	/markovChains/makeFromDictionary
Operácie s kolekciou masiek	/masks/
Operácie s konkrétnou maskou	/masks/<id>
Stiahnutie binárneho Markovho modelu	/masks/<id>/download
Operácie s upozorneniami	/notifications/
Získanie počtu nových upozornení	/notifications/count
Operácie s kolekciou šifrovaných súborov	/protectedFiles/
Operácie s konkrétnym šifrovacím súborom	/protectedFiles/<id>
Operácie s kolekciou pravidiel	/rule/
Operácie s konkrétnym pravidlom	/rule/<id>
Operácie so serverovými nastaveniami	/serverInfo/control
Získanie informácií o behu servera	/serverInfo/info
Operácie s kolekciou užívateľov	/user/
Operácie s konkrétnym užívateľom	/user/<id>
Zistenie či je užívateľ prihlásený v systéme	/user/isLoggedIn
Prihlásenie užívateľa do systému	/user/login
Odhlásenie užívateľa zo systému	/user/logout
Operácie s kolekciou užívateľských rolí	/user/role
Operácie s konkrétnou užívateľskou rolou	/user/role/<id>

Tabuľka 5.1: Zoznam dostupných URI.

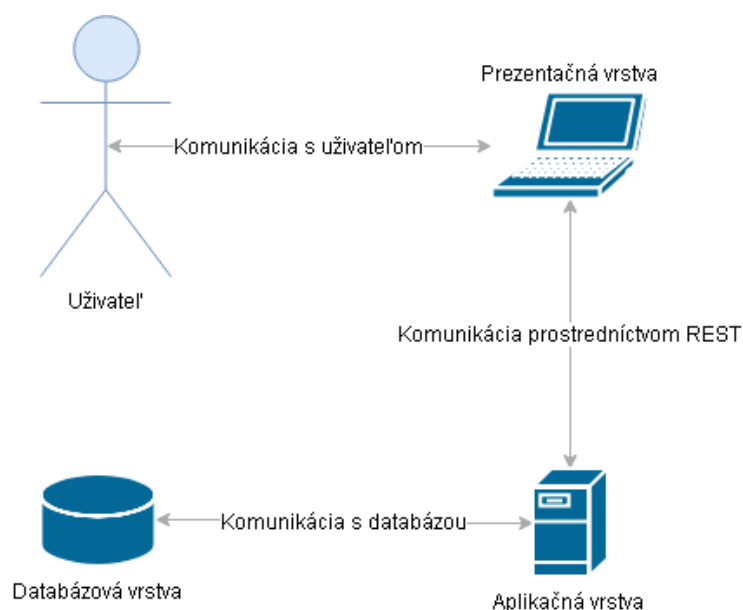
Kapitola 6

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 6.1. 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 4.4.3.

Databázovú vrstvu tvorí MySQL databáza, ktorá je vytvorená podľa ER diagramu z obrázka 5.2. Databázová vrstva spracováva požiadavky aplikačnej vrstvy a vykonáva operácie ako ukladanie dát, zobrazovanie dát, vyhľadávanie a mazanie dát.



Obr. 6.1: Architektúra aplikácie

6.1 Implementácia databázovej vrstvy

Databázová vrstva je implementovaná pomocou databázového systému MySQL. Je vytvorená podľa entitno relačného diagramu, ktorý je možné vidieť na obrázku 5.2. 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 zahešovanej podobe s pridanou soľou, email a odkaz do tabuľky `role`, ktorá obsahuje jednotlivé oprávnenia užívateľov v rámci systému. Oprávnenia k jednotlivým úlohám môžu byť pridané prostredníctvom tabuľky `user_permissions`. Tabuľka `notification` slúži na uchovávanie upozornení pre užívateľov. Údaje o postupe úloh sa uchovávajú v tabuľke `package_graph`. Z týchto údajov je následne možné na prezentačnej vrstve vytvoriť graf progresu (viď 5.4). V databáze sa tiež nachádza spúšťač, ktorý sa vykoná vždy po upravení úlohy. Tento spúšťač je možné rozdeliť na dve časti. Prvú časť je možné vidieť na výpise 2 a slúži na počítanie progresu úlohy a pridávanie záznamov do tabuľky `package_graph`. Druhá časť spúšťača je vyobrazená na výpise 3. Slúži na rozosielanie notifikácií užívateľom.

Algoritmus 2 Telo spúšťača, 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 spúšťač vybral len tých užívateľov, ktorí majú k úlohe prístup (užívateľa s rolou, ktorá má oprávnenie na zobrazenie všetkých úloh, alebo užívateľa s oprávnením k úlohe v tabuľke `user_permissions`).

Algoritmus 3 Spúšťač, 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;
```

6.2 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 6.2.

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

Obr. 6.2: Hierarchia balíkov aplikačnej vrstvy.

V koreňovom adresári sa nachádza súbor `app.py`, ktorý obsahuje funkciu `main()`. Táto funkcia najprv vykoná inicializáciu systému, nakonfiguruje systém a načíta všetky koncové body aplikačného rozhrania, a následne spustí webový server. O chod webového servera sa stará framework Flask (viď 4.3.2). 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 4.

Algoritmus 4 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             getattr(self.package, 'id'): round(getattr(self, 'progress'))
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 4.3.1).

6.2.1 Implementácia jednotlivých URI

Pri vytváraní funkcií na spracovanie jednotlivých žiadostí používam rozšírenie do frameworku `Flask` s názvom `Flask-RESTPlus` (viď 4.3.2). 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 4.2.3). 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 5. V module `argumentsParser.py` sa nachádzajú

Algoritmus 5 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 5 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 JSON4.4.3, ktorými server odpovedá prezentačnej vrstve. V module `functions.py` sú uložené funkcie, ktoré používa balík ošetrojúci sadu URI. Z výpisu 5 sa tu napríklad nachádza funkcia `create_job`.

6.2.2 Spracovanie útokov

Spracovanie úlohy na aplikačnej vrstve prebieha v niekoľkých fázach. Najprv sa získa heš určený na lámanie. Ten je možné buď zadať priamo, alebo nahráť šifrovaný súbor, z ktorého systém extrahuje heš pomocou programu `XtoHashcat`⁶. Následné overí formát hešu 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 hešu sa systém pokúsi 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

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

s 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. Následne sa vypočíta veľkosť množiny hesiel danej úlohy. Pri slovníkovom útoku 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 je možné sa dočítať v podkapitole 5.3.2.

- **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 5.3.3
- **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 5.3.1.
- **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 5.3.4.

Po spracovaní útoku je nutné vytvoriť konfiguráciu útoku pre generátor vo formáte `TLV`⁸. Potom sa už len záznam uloží do databázy a priradia sa k nemu výpočtové uzly. Aplikačnej vrstve sa odošle správa o úspešnom pridaní úlohy do databázy.

6.2.3 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.

6.2.4 Odhadovanie času výpočtu

Systém pri vytváraní úlohy poskytuje užívateľovi informácie o odhadovanom čase výpočtu, ktorý úloha zaberie. Prezentačná vrstva pre zobrazenie odhadovaného času pošle žiadosť na aplikačnú vrstvu s potrebnými údajmi na jeho vypočítanie. Na získanie odhadu času pre výpočet je potrebné niekoľko informácií.

- **Typ hešu** - keďže časy výpočtu hešu sa od seba líšia pri rôznych kryptografických algoritmoch, je nutné vybrať typ hešovacej funkcie.

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/>

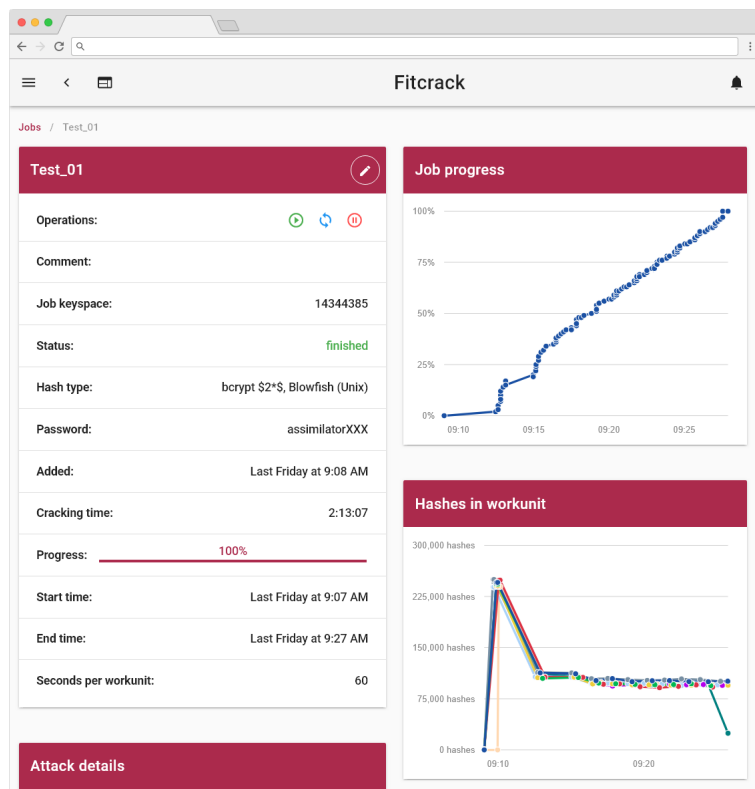
- **Výpočtové uzly** - taktiež je nutné vybrať výpočtové uzly, ktoré sa budú na úlohe podieľať. Potom je možné spočítať celkovú silu uzlov pre určitú kryptografickú funkciu a získať tak celkové množstvo vypočítaných hešov za jednu sekundu.
- **Informácie o útoku** - pre výpočet odhadovaného času je taktiež nutné vedieť veľkosť množiny hesiel danej úlohy. Tá sa pri rôznych typoch útokov získava inak, preto je potrebné uviesť v informáciach typ útoku a ďalšie informácie, ktoré sú potrebné na jej výpočet. Pri slovníkovom útoku je nutné poslať identifikátor slovníka, ktorý sa potom načíta z databázy kde je uvedená jeho veľkosť (počet hesiel, ktoré slovník obsahuje). Pri útokoch s maskami je potrebné poslať v žiadosti aj všetky zadané masky, z ktorých sa vypočíta veľkosť množiny hesiel, ktoré masky zastupujú. Pri kombinačnom útoku sa odosielaajú v žiadosti na server identifikátory oboch slovníkov. Celkový počet hesiel zistíme vynásobením počtu hesiel ktoré jednotlivé slovníky obsahujú. Pri hybridnom útoku sa posiela v informáciach o útoku zadaná maska a identifikátor slovníku. Z týchto informácií je potom možné vypočítať veľkosť množiny hesiel vynásobením počtu hesiel v slovníku a počtu hesiel, ktoré zastupuje zadaná maska.

Odhadovaný čas úlohy v sekundách sa vypočíta podielom veľkosti množiny hesiel s celkovým výkonom uzlov. Jedná sa o odhadovaný čas, ktorý by úloha trvala, keby sa heslo našlo až úplne na konci množiny všetkých hesiel. V dôsledku toho končia väčšinou úlohy skôr. Niekedy sa však môže stať že úloha bude trvať dlhšie, ako je jej odhadovaný čas v dôsledku vyťaženia niektorých z uzlov.

6.3 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 žiadostí typu AJAX. Vďaka tomu je 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^{4.4.3}. Webová aplikácia je implementovaná pomocou frameworku Vue (viď 4.4), ktorý výrazným spôsobom uľahčuje členenie aplikácie na jednotlivé komponenty, ktoré sú znovupoužiteľné. Výsledok implementácie prezentačnej vrstvy je možné vidieť na obrázkoch 6.3, kde je snímka z detailného pohľadu na úlohu.

¹⁰<https://github.com/axios/axios>



Obr. 6.3: Detailná stránka úlohy

6.3.1 Implementácia komponentov

Pre zjednodušenie implementácie som každú stránku rozdelil na znovupoužiteľné komponenty, ktoré predstavujú logický celok. Rozklad stránky detailu úlohy na komponenty je možné vidieť na obrázku 6.4.



Obr. 6.4: Detailná stránka úlohy rozdelená na komponenty

Jednotlivé komponenty sú uložené v súboroch s príponou `.vue`, ktoré sa nachádzajú v zložke `src/components`. Príklad obsahuje takéhoto súboru je možné vidieť na výpise 6.

Algoritmus 6 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>
```

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. Využije 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 niekoľkých fáz. 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>

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

6.3.2 Globálne dáta

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

6.3.3 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 ktorej 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 7.

Algoritmus 7 Konfiguračný súbor navigácie

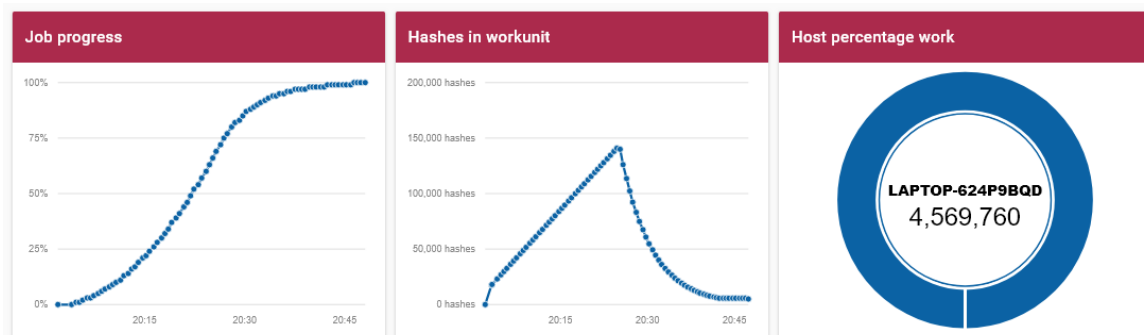
```
1 routes: [  
2   {  
3     path: '/',  
4     name: 'home',  
5     component: home  
6   },  
7   {  
8     path: '/jobs',  
9     name: 'jobs',  
10    component: jobs  
11  },  
12  {  
13    path: '/jobs/:id',  
14    name: 'jobDetail',  
15    component: jobDetail  
16  },  
17  ...  
18  {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

6.3.4 Grafy

Výraznú časť užívateľského rozhrania aplikácie tvoria grafy, ktoré sú detailnejšie popísané v kapitole 5.4. Každý graf sa periodicky dotazuje servera so žiadosťou o nové dáta. Server

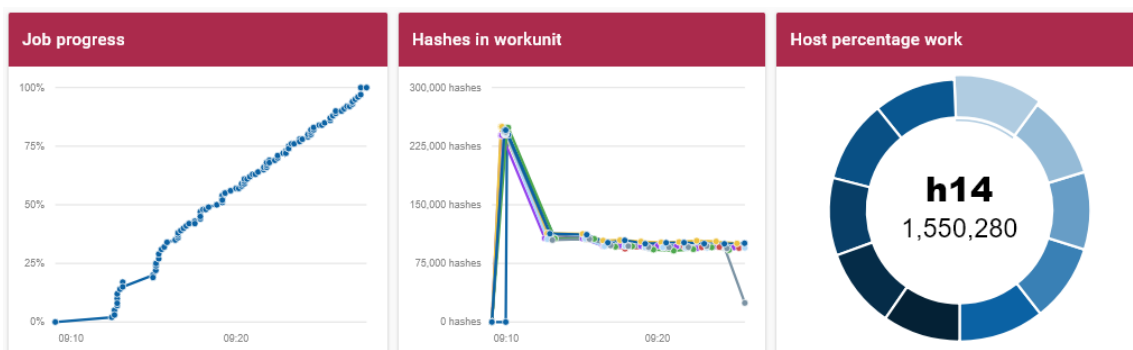
¹³<https://vuetifyjs.com/>

odpovedá správou vo formáte JSON^{4.4.3}, 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 6.5 sa nachádzajú grafy jednej z úloh.



Obr. 6.5: Grafy úlohy s jedným výpočtovým uzlom

Z grafu **Hashes in workunit** (počet hešov v pracovnej jednotke) je zrejmá práca Asimilátora (viď 2.3.3), ktorý ma za úlohu prispôbovať veľkosť pracovnej jednotky výpočtovým uzlom. V tejto úlohe bol zapojený len jeden uzol s názvom **LAPTOP-824P9BQD**. Jednalo sa o pomerne výkonný uzol, čo je možné vidieť aj podľa toho, že mu Asimilátor spolu s Generátorom prideliť v pracovnej jednotke čoraz viac hešov na overenie. Toto zväčšovanie by pokračovalo až kým by výpočtový uzol nespracovával pracovné jednotky tak rýchlo, ako bolo stanovené pri vytváraní úlohy, ale od polovice overených hešov 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 výpočtových uzlov je možné vidieť na obrázku 6.6. Koláčový diagram značí, že v tomto prípade boli uzly výkonnostne pomerne vyrovnané.



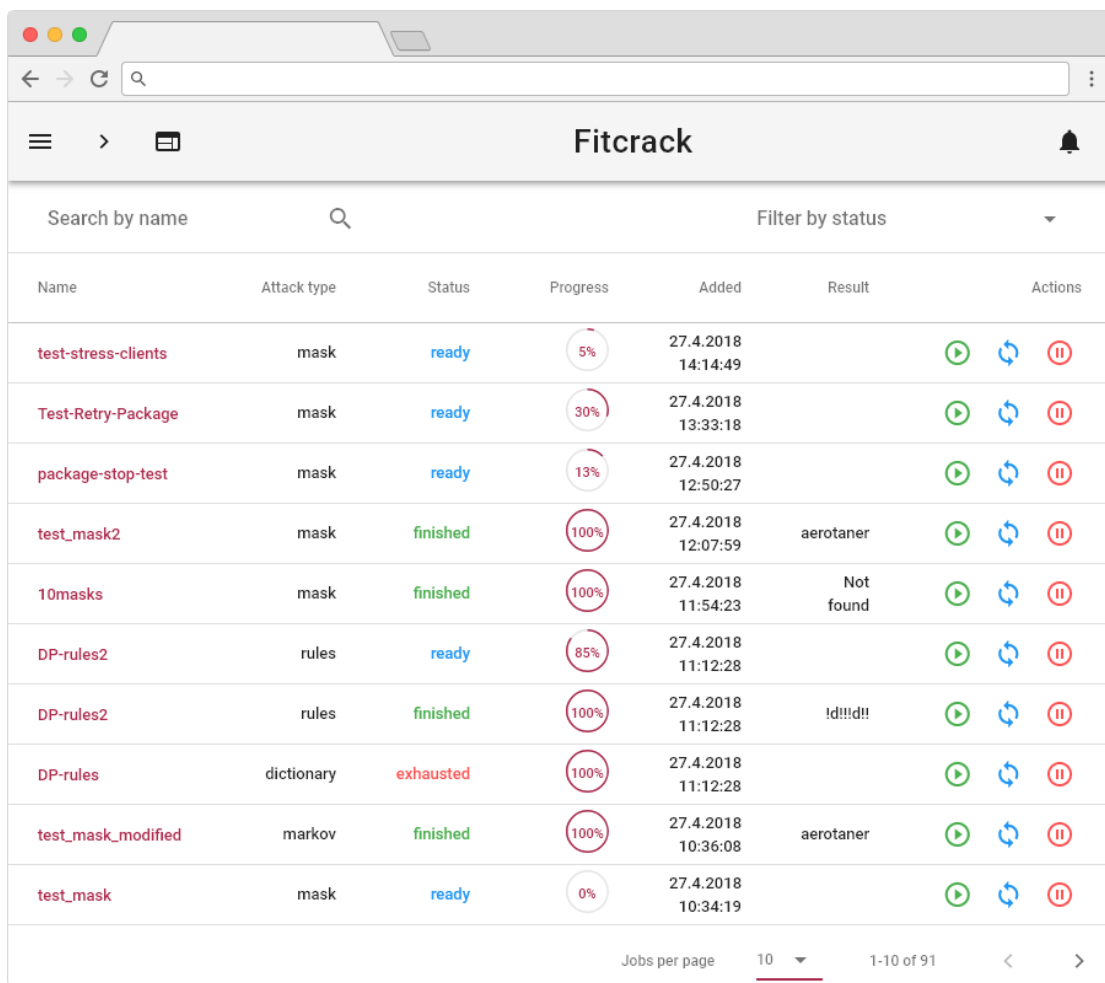
Obr. 6.6: Grafy úlohy s viacerými uzlami

6.3.5 Prehľad úloh a stránkovanie

Stránku prehľadu úloh tvorí prehľadná tabuľka, ktorá obsahuje základné informácie o úlohách ako názov, typ útoku, aktuálny stav úlohy, progres, čas pridania, hľadané heslo a

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

tlačítka, ktoré spúšťajú operácie s úlohou. Príklad tejto stránky je možné vidieť na obrázku 6.7. 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 serverovú 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 are listed with their respective progress bars and status indicators.

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	Id!!!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

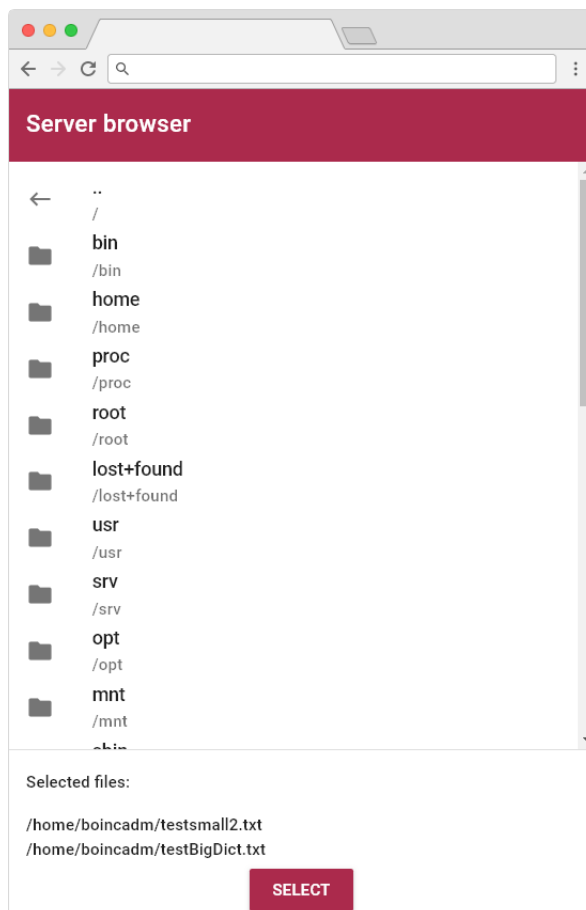
Obr. 6.7: Stránka s tabuľkou úloh

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).

6.3.6 Nahrávanie slovníkov z diskového priestoru servera

Kvôli limitáciám webového servera Apache, bolo nutné vymyslieť alternatívny spôsob nahrávania slovníkov (alebo iných súborov) väčších ako niekoľko gigabajtov. Taktiež je vhodné,

aby sa dal slovník vybrať priamo z diskového priestoru servera, alebo z prenosného úložiska pripojeného k serveru. Z toho dôvodu som do systému implementoval jednoduchý prehliadač súborov servera, v ktorom je možné navigovať sa v rámci súborového systému a označovať súbory. Po potvrdení sa tieto súbory spracujú ako slovníky (pridajú sa do databázy a spočíta sa počet hesiel, ktoré obsahujú) a pre každý súbor sa v predvolenej zložke na ukladanie slovníkov vytvorí zástupca, ktorý odkazuje na tento súbor. Snímku prehliadača súborov servera je možné vidieť na obrázku 6.8.



Obr. 6.8: Prehliadač súborov servera.

6.3.7 Databáza známych hešov

Aby sme predišli opakovanému lámaniu heša, ktorý sme už v minulosti počítali, aplikácia podporuje históriu nájdených kryptografických hešov spolu s heslami. Po úspešnom dokončení úlohy sa heš spolu s heslom pridá do databázy. História nájdených hesiel je vizualizovaná pomocou prehľadnej tabuľky, v ktorej je možné vyhľadávať a zoradiť záznamy podľa času nájdenia, alebo abecedného poradia hesiel.

Kapitola 7

Testovanie a experimenty

Testovanie je neoddeliteľnou 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.

7.1 Funkcionálne testovanie

Funkcionálne testovanie malo predovšetkým odhaliť chyby v 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 uzlov, určený typ hešu a ďalšie hodnoty. Tieto hodnoty som následne porovnal s hodnotami z databázy, pričom som zistil, že sa rovnajú.

7.2 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. Uživateľské 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.

7.3 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 7.1. Slovník zaberá po vygenerovaní 341 797kB diskového priestoru.

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

Obr. 7.1: 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`, čo je posledné heslo v slovníku `experiment.txt`, takže systém bude musieť prejsť celý slovník. Následne som do systému pripojil dva uzly a vytvoril úlohu, do ktorej som nahral zašifrovaný archív a vybral slovníkový útok s vygenerovaným slovníkom `experiment.txt`. Typ heša 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 7.2.

¹<https://www.win-rar.com/>

The screenshot shows the Fitcrack web application interface. At the top, a status bar indicates "Estimated cracking time is 0:21:46." Below this, there are two main sections: "Create new job" and "Input settings".

Create new job

Name:

Comment:

Input settings

Hashtype:

Upload method:

- ☒ Encrypted file
- ☐ Hash text
- ☐ Multiple hashes

faktery.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

Dictionary per page 5 6-9 of 9

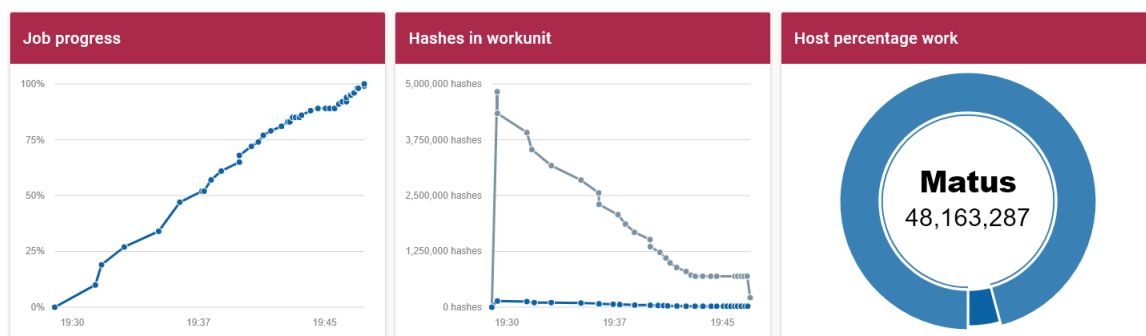
Obr. 7.2: Vytváranie experimentálnej úlohy

Po spustení úlohy sa vyťaženie uzlov prudko zdvihlo. Jeden výpočtový uzol svojim výkonom výrazne prevyšoval druhý. Po približne pol hodine sa zobrazila notifikácia o dokončení úlohy. Ako vidno na obrázku 7.3, 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. 7.3: Vytváranie experimentálnej úlohy

Za pozornosť stoja aj grafy, ktoré je možné vidieť na obrázku 7.4. Koláčový graf podporuje predpoklad o tom, že jeden uzol je výrazne výkonnejší ako druhý. Na grafe s progresom úlohy vidíme že progres stúpal približne lineárne.



Obr. 7.4: Vytváranie experimentálnej úlohy

Kapitola 8

Záver

V rámci tejto práce som urobil rozbor systému Fitcrack a analyzoval požiadavky na webovú aplikáciu, ktorá má slúžiť na vzdialenú správu systému. V rámci návrhu systému bola výrazne upravená schéma databázy, navrhnutý systém užívateľov a rozloženie prvkov na klientskej časti, popísané typy útokov ktoré bude webová aplikácia podporovať a navrhnuté druhy grafov.

Po návrhu systému, prebehla jeho implementácia, ktorej súčasťou je moderné webové rozhranie, serverová časť komunikujúca prostredníctvom REST API a databázový systém MySQL. Oproti prechádzajúcemu riešeniu, poskytuje moja implementácia rôzne rozšírenia ako napríklad podpora viacerých užívateľov s ich oprávneniami, 3 nové typy útokov, systém upozornení, príjemné a prehľadné užívateľské rozhranie so zaujímavými grafickými prvkami ako sú napríklad grafy. Taktiež systém podporuje odhad času potrebného na dokončenie úlohy a overenie správneho formátu hešu. Keďže som od seba oddelil klientskú a serverovú vrstvu, systém je výkonnostne menej náročný. Vďaka použitiu jednostránkovej webovej prezentácie a komunikácie so serverovou časťou prostredníctvom asynchrónnych žiadostí (AJAX), je možné periodicky aktualizovať dáta na prezentačnej vrstve a tým zabezpečiť konzistentnosť s databázovou vrstvou bez nutnosti znovunačítania stránky.

Po implementácii webovej aplikácie nasledovalo overenie funkčnosti prostredníctvom užívateľského a funkcionálneho testovania. Testovanie objavilo niekoľko drobných chýb, ktoré boli opravené. Potom nasledoval experiment, ktorý overil správnu funkčnosť celého systému Fitcrack.

8.1 Výhľad do budúcnosti

Keďže si myslím, že systém Fitcrack môže byť v budúcnosti úspešný, mám záujem webovú aplikáciu ďalej rozširovať. Plánujem pridať možnosť nahrávania slovníkov priamo z diskového priestoru serveru, pridať logovací systém, možnosť zobrazenia databázy hešov, ktoré sa už v minulosti našli a množstvo ďalších vylepšení. Taktiež by bolo vhodné zbaviť sa závislosti serverovej časti na nástroji Hashcat, ktorý server používa už len na výpočet veľkosti množiny hesiel.

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] Bray, T.: The JavaScript Object Notation (JSON) Data Interchange Format. RFC 7159, RFC Editor, March 2014, <http://www.rfc-editor.org/rfc/rfc7159.txt>.
URL <http://www.rfc-editor.org/rfc/rfc7159.txt>
- [4] Copeland, R.: *Essential sqlalchemy*. Ö'Reilly Media, Inc.", 2008.
- [5] 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.
- [6] 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>
- [7] Gazdík, P.: *Využití heuristik při obnově hesel pomocí GPU*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2016.
- [8] Grinberg, M.: *Flask web development: developing web applications with python*. Ö'Reilly Media, Inc.", 2018.
- [9] Hranický, R.; Zabal, L.; Večeřa, V.: Distribuovaná obnova hesel. Technická Zpráva FIT-TR-2017-04, CZ, 2017.
- [10] Masse, M.: *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. Ö'Reilly Media, Inc.", 2011.
- [11] Rescorla, E.: HTTP Over TLS. RFC 2818, RFC Editor, May 2000.
URL <http://www.rfc-editor.org/rfc/rfc2818.txt>
- [12] Shiflett, C.: *HTTP developer's handbook*. Sams Publishing, 2003.