VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

AUTOMATIZOVANÁ ANALÝZA A ARCHIVACE DAT Z WEBU

AUTOMATED WEB ANALYSIS AND ARCHIVATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. TOMÁŠ KOCMAN**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. LIBOR POLČÁK, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2018/2019

Zadání diplomové práce

Student: **Kocman Tomáš, Bc.**
Program: Informační technologie Obor: Počítačové sítě a komunikace
Název: **Automatizovaná analýza a archivace dat z webu**
Automated Web Analysis and Archivation
Kategorie: Data mining

Zadání:

1. Seznamte se s nástroji vhodnými pro automatizovanou analýzu webu, extrakci dat a ukládání dat.
2. Identifikujte data obsažená na webu vhodná pro vyšetřování trestné činnosti.
3. S využitím existujících nástrojů navrhněte automatické získávání podkladů pro vyšetřování trestné činnosti z webu.
4. Navržené prostředí implementujte.
5. Implementaci otestujte se zaměřením na výkonnost, škálovatelnost a množství ukládaných dat.
6. Zhodnoťte vytvořené nástroje a navrhněte možná rozšíření.

Literatura:

- SEREČUN, Viliam. *Automatizovaná rekonstrukce webových stránek*. Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií.
- GOODMAN, Marc. *Future crimes: inside the digital underground and the battle for our connected world*. Anchor, 2016. ISBN: 978-0804171458
- JOHNSON, Faustina, GUPTA, Santosh Kumar. Web Content Mining Techniques: A Survey. International Journal of Computer Applications, Volume 47- č.11, 2012.
- Ministerstvo vnitra České Republiky, odbor bezpečnostní politiky a prevence kriminality: Audit národní bezpečnosti. dostupné online, URL https://www.vlada.cz/assets/media-centrum/aktualne/Audit-narodni-bezpecnosti-20161201.pdf

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Polčák Libor, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 22. května 2019
Datum schválení: 23. října 2018

Abstrakt

Tato práce je zaměřena na kyberkriminalitu, získávání důkazních materiálů a tvorbu platformy pro získávání, analýzu a archivaci dat z webových stránek, která by vyhověla vyšetřovatelům a bezpečnostním expertům Policie České republiky. Cílem je poskytnout platformu s otevřeným zdrojovým kódem, která bude volně šiřitelná a uspokojí požadavky právních institucí. Výstupem diplomové práce budou tedy dvě verze platformy – plnohodnotná, splňující všechny požadavky stanovené zadáním diplomové práce a odlehčená verze pro vyšetřovatele Policie České republiky.

Abstract

This thesis is focused on cybercrime, acquisition of evidence and development of a platform for retrieval, analysis and archiving web site data. The goal is to satisfy the investigators and security experts of the Czech police. The aim is to provide an open source platform that will be freely disseminable and in compliance with the requirements of legal institutions. The output of the thesis is two platform versions – full-fledged, fulfilling all the requirements set out in the diploma thesis and a light version for the police investigators of the Czech Republic.

Klíčová slova

kyberkriminalita, archivace webu, dolování dat, analýza webu, Scrapy, Lemmiwinks, Python, Redis, PostgreSQL

Keywords

cybercrime, web archivation, data mining, web analysis, Scrapy, Lemmiwinks, Python, Redis, PostgreSQL

Citace

KOCMAN, Tomáš. *Automatizovaná analýza a archivace dat z webu.* Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Libor Polčák, Ph.D.

Automatizovaná analýza a archivace dat z webu

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana doktora Libora Polčáka. Další informace mi poskytl pan inženýr Viliam Serečun. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Tomáš Kocman
18. května 2019

Poděkování

Rád bych poděkoval vedoucímu své diplomové práce panu Ing. Liboru Polčákovi Ph.D. a panu Ing. Viliamu Serečunovi za odborné rady a čas, které mi při tvorbě této práce poskytli.

Obsah

1 Úvod **3**
1.1 Přehled kapitol . . . 3

2 Kyberkriminalita **4**
2.1 Projevy kyberkriminality . . . 5
2.1.1 Pirátský obsah . . . 8
2.1.2 Drogy . . . 9
2.1.3 Kradené bankovní účty a karty . . . 9
2.1.4 Zbraně . . . 10
2.1.5 Obchodování s lidskými orgány . . . 10
2.1.6 Dětská pornografie . . . 11
2.2 Vyšetřování kyberkriminality . . . 11
2.2.1 Digitální stopa . . . 12
2.2.2 Postup při vyšetřování . . . 14
2.2.3 Dokazování kyberkriminality . . . 14

3 Sběr dat na internetu **16**
3.1 Legálnost . . . 18
3.2 Existující metody a nástroje . . . 19
3.2.1 XPath, CSS selektor . . . 19
3.2.2 Requests/urllib3 a Beautiful Soup . . . 20
3.2.3 Selenium a PhantomJS . . . 21
3.3 Existující řešení . . . 22
3.3.1 Apache Nutch . . . 22
3.3.2 HTTrack . . . 23
3.3.3 Scrapy . . . 23
3.3.4 Lemmiwinks . . . 24

4 Návrh řešení **26**
4.1 Výběr technologií . . . 26
4.2 Ovládání platformy . . . 29
4.3 Konzultace s vyšetřovateli PČR . . . 30
4.4 Architektura platformy . . . 31

5 Implementace **33**
5.1 Databázová část . . . 33
5.1.1 PostgreSQL . . . 33
5.1.2 Redis . . . 35

5.2 Scrapy pro sběr dat . . . 37
5.2.1 Architektura . . . 37
5.2.2 Možnosti konfigurace . . . 40
5.3 Lemmiwinks pro archivaci stránek . . . 41
5.3.1 Architektura . . . 41
5.3.2 Nutné modifikace . . . 43
5.4 Aplikační programové rozhraní . . . 44
5.4.1 Architektura . . . 46
5.5 Docker . . . 47
5.6 Verze platformy pro účely vyšetřování . . . 49

6 Testování **51**
6.1 Jednotkové testy . . . 52
6.2 Manuální testování . . . 54
6.3 Výkonnost . . . 55
6.4 Množství uložených dat . . . 57
6.5 Hledání důkazního materiálu . . . 58

7 Závěr **59**
7.1 Možnosti rozšíření této diplomové práce . . . 59

Literatura **60**

A Ukázka obsahu tabulek databáze PostgreSQL **62**

B Instalace, konfigurace, nasazení **64**

Kapitola 1

Úvod

Je důležité brát na vědomí, že ve světě se páchají zločiny od nepaměti a na této skutečnosti se v budoucnosti nejspíš nic nezmění. Stejným tempem, kterým se vyvíjejí zločinci a zločinecké organizace se musí vyvíjet rovněž síly na druhé straně. Za tímto účelem vznikla tato diplomová práce – pomáhat, chránit a usvědčovat pachatele.

Ačkoli si to lidé neuvědomují nebo tu skutečnost nechtějí vidět či ji ignorují, ve světě se vedou kybernetické války. Nejde o válku konvenční, která přijde člověku na mysl jako první, nýbrž o válku v kyberprostoru. Jednotlivci, zločinecké organizace či tajné státní kybernetické útvary útočí jak na běžné obyvatele, tak i na celé státy. Na řadu přichází kvalitní obrana – specialisté v oblasti kybernetické bezpečnosti. Ti ovšem potřebují patřičné nástroje pro ulehčení a automatizaci své investigativní práce.

Tato diplomová práce má sloužit převážně pro vyšetřovatele České republiky při vyšetřování kybernetických trestných činů. Může nastat případ, kdy například probíhá soudní řízení a čeká se na důkazní materiál. V tomto případě může posloužit diplomová práce k vyhledání a k získání požadovaného důkazního materiálu z webových stránek. Mějme například obžalovaného za překupnictví drog. Pokud vyšetřovatelé znají webové stránky (pravděpodobně půjde o web umístěný na síti Tor), popřípadě fórum, kde pachatel obchodoval, můžou platformu využít k nalezení důkazního materiálu – internetovou přezdívku pachatele, obsah zpráv, čas a místo obchodu a další pro soud cenné informace. Podobná situace může nastat například tehdy, pokud pachatel poskytoval nelegální obsah skrze síť BitTorrent. Potom je nasnadě vyhledat na webu daný torrent soubor podle jeho infohaše.

1.1 Přehled kapitol

V kapitole 2 se hovoří o hrozbách a o rizicích v kyberprostoru. Kapitola rozebírá znaky projevu kyberkriminality, srovnání s běžnou kriminalitou a její typy. Je rovněž stručně popsán proces vyšetřování kyberkriminality. Kapitola 3 popisuje konkrétně a s technickými detaily techniky, způsoby a možnosti hledání informací, jejich analýzu a následnou archivaci. Popisuje, do jaké míry je sběr informací legální, rovněž představuje existující metody a nástroje pro sběr informací a jejich analýzu. V kapitole 4 jsou rozebrány myšlenky návrhu, jakým směrem by se měla diplomová práce ubírat a jaké by měla mít vlastnosti, strukturu, způsob nasazení, či testování. Kapitola 5 popisuje konkrétní implementaci obou verzí platformy, čili způsob řešení sběrače dat, archivátoru webových stránek a databázových částí. V kapitole 6 je popsán způsob testování platformy – jednotkové, manuální a výkonnostní testy spolu s praktickou ukázkou získávání důkazních materiálů.

Kapitola 2

Kyberkriminalita

V této kapitole budou evaluovány hrozby a rizika v kyberprostoru jak obecně, tak i vyplývající pro ČR a její obyvatele. Sekce 2.1 rozebere znaky projevu kyberkriminality, srovnání s běžnou kriminalitou a její různé typy. Dále v sekci 2.2 bude stručně popsán proces vyšetřování kyberkriminality se všemi podstatnými aspekty. Jelikož je tato práce tvořena v rámci projektu TARZAN, který je součástí bezpečnostního výzkumu České republiky, bude z části zaměřena na prostředí České Republiky, ačkoli principy, které budou prezentovány jsou platné obecně.

V dnešní době není před hrozbami v kyberprostoru plně chráněn žádný stát, ten náš nevyjímaje [16]. Vzhledem k tomu, že se zhoršuje bezpečnostní situace ve státech bezprostředně sousedících s členskými státy NATO a EU, jsou vyvíjeny zvyšující se nároky na schopnost ČR reagovat na případné bezpečností hrozby v kyberprostoru. Lze sledovat rostoucí snahy nestátních i státních aktérů ve vyvíjení a nasazování kybernetických ofenzivních prostředků, které cílí jak na kritickou infrastrukturu, respektive její části, které jsou přítomny v kyberprostoru, tak i na běžné obyvatele, kteří mají pouze omezenou možnost se bránit. Rozlišujeme pět konkrétních hrozeb:

- kybernetická špionáž,
- narušení nebo snížení odolnosti IT infrastruktury,
- nepřátelské kampaně,
- narušení nebo snížení bezpečnosti eGovernmentu,
- kyberkriminalita.

Pro účely této diplomové práce je nejpodstatnější poslední, pátý, bod. Za kyberkriminalitu lze považovat jakékoli protiprávní jednání. Kyberterorismus je poté podmnožina tohoto pojmu, který značí takové teroristické aktivity v kyberprostoru, které narušují počítačové sítě a zařízení. Při útocích tohoto typu může docházet k lidským úmrtím nebo k závažným ekonomickým ztrátám, jejichž důsledky jsou jen těžko předvídatelné.

Dle bezpečnostního auditu ministerstva vnitra ČR [16] zde hrají významnou roli tzv. nová média (média založená na digitálním kódování dat – software i hardware). Kyberkriminalitu nelze považovat za pouhý hypotetický fenomén. Útoky probíhají v čím dál větším měřítku nejen na běžné uživatele internetu, čili jednotlivce, ale rovněž na celé státy (například pro manipulaci s politickou situací). V současnosti však velkou část útoků a incidentů, často mediálně popisovaných a prezentovaných jako kyberkriminalita, můžeme označit spíše

za využívání kyberprostoru (internetu) teroristy. Teroristické organizace nejspíše nedisponují kapacitami a schopnostmi k uskutečnění kybernetických útoků s vážnými dopady [16]. Ovšem není obtížné tyto schopnosti a kapacity nakoupit ve formě služby. Například Islámský stát dokázal provést kybernetické útoky (avšak nikterak sofistikované), které jiné teroristické organizace nebyly schopny dlouho uskutečnit.

Dalším problémem je využívání prostředí dark webu v ČR k ilegálním činnostem mířeným jak na běžné obyvatele, tak na vrcholné představitele státu, odcizování e-mailové komunikace a ovlivňování veřejného mínění [16]. Rovněž byly zjištěny úspěšné pokusy o znepřístupnění služeb či webových stránek, případně sdělovacích prostředků nebo sociálních profilů politických subjektů. Je logické předpokládat, že se tyto situace budou s největší pravděpodobností opakovat a stupňovat z hlediska závažnosti. Výčet nejdůležitějších rizik a problémů:

- Koordinovaný kybernetický útok za účelem vydírání obchodních korporací, státních orgánů či vystrašení společnosti.
- Útoky se snahou ukořistit citlivé informace zpravodajského charakteru (příprava pro napadení určitých subjektů, informace pro výběr cílů).
- Teroristé hojně používají kyberprostor k šíření propagandy, náboru a radikalizaci stoupenců. Rovněž zveřejňují soukromé údaje zájmových osob na internetu a podněcují proti nim případné útočníky.
- Nedostatek ICT expertů v bezpečnostních složkách a jejich nízká připravenost na digitální prostředí, obzvláště v dark webu, který je stále ve větší míře využíván teroristy a organizovanými zločineckými strukturami.

Poslední bod je pro tuto diplomovou práci stěžejní. Práce by měla v této oblasti pomoci bezpečnostním složkám automatizovaně prohledávat weby a vyhledávat v nich stěžejní informace. Práce se teď odprostí od problémů naší země jako takové k problémům jednotlivců. Dále se v textu budou objevovat pojmy jako slovník a regulární výraz. Je důležité porozumět jejich síle v kontextu vyhledávání v textu. Slovník je jednoduchý – obsahuje slova, ve kterých se vyhledává aktuálně zkoumaný řetězec v textu. Oproti tomu regulární výraz[1] obecně definuje množinu vyhovujících řetězců, čili je podstatně složitější vymyslet takový regulární výraz, který nebude příliš konkrétní, ani obecný.

2.1 Projevy kyberkriminality

Tato sekce bude hovořit o různých typech projevů kyberkriminality spolu se srovnáním k běžné kyberkriminalitě. Každá další sekce rozebírá specifický druh kyberkriminality zaměřený spíše na jednotlivce.

Přestože se kyberkriminalita projevuje převážně prostřednictvím kybernetických útoků, k úspěšnému uskutečnění řady útoků je zapotřebí využít i netechnických aspektů [11]. Musíme být obezřetní při definování toho, co je a co už není kyberkriminalitou. Jednání související s kyberkriminalitou či určitá protiprávní jednání v kyberprostoru jsme schopni zařadit pod příslušná ustanovení trestního zákoníku. Existují však takové typy jednání, jejichž označení za trestné činy je nemožné, případně podstatně obtížné. V mnoha případech se totiž jedná pouze o nemorální jednání.

[1] https://www.princeton.edu/~mlovett/reference/Regular-Expressions.pdf

Kyberkriminalita je velmi často označována za nový typ kriminality, nicméně podstatná část kyberkriminality přenáší známé druhy protiprávního jednání (porušování autorských práv, šikana, podvody, krádeže) do digitálního prostředí, ve kterém je pachatel páchá lépe, rychleji a efektivněji, nežli v reálném světě [11]. Za ryze kybernetické útoky můžeme považovat například hacking, DoS, DDoS nebo botnety.

V dnešní době je pro virtuální svět typické, že většina nevědomých uživatelů v něj má téměř bezmeznou důvěru [11]. To jde ruku v ruce s tvrzením, že se pro nás virtuální svět stává čím dál tím významnějším. Při využívání různých internetových služeb mnoho lidí vůbec nepřemýšlí o možných hrozbách a rizicích – jsou uchváceni, skoro až hypnoticky, nekonečnými možnostmi internetu a všech nových technologií. Jak jinak si potom máme vysvětlit absenci elementárních obranných mechanismů a principů ve virtuálním světě, když v reálném světě bychom se chovali zcela jinak. Dalším typem uživatelů v kyberprostoru jsou lidé, kteří se v reálném světě chovají slušně a nevzbuzují minimální podezření, ovšem v pseudoanonymním prostředí kyberprostoru se projevují bez jakýchkoli morálních nebo legálních zábran. Můžeme tak narazit například na případ soudce, jenž si stahuje dětskou pornografii či uživatele, kteří by nikdy v reálném světě nic neukradli, ale v tom virtuálním jsou zloději [11].

Možným příkladem [1], který se stal v době psaní této práce, je zpráva o francouzském policistovi, který byl zatčen za prodej důvěrných dat na dark webu. Platidlem byl samozřejmě Bitcoin, jak je v této sféře zvykem. Francouzská policie jej zatkla po tom, co odhalili „Černou ruku“ dark web trhu. Prověřením všech získaných dat našla policie informace o prodeji tajných francouzských policejních dokumentů. Všechny tyto dokumenty měly unikátní identifikátory, které posloužily k vyhledání francouzského policisty, který tato data prodával pod jménem Haurus. Kromě prodeje těchto dokumentů rovněž zjistili, že provozoval službu ke sledování lokace mobilních zařízení podle telefonního čísla. Službu prodával za účelem sledování svých partnerů či členů nepřátelských soupeřících gangů. Vyšetřovatelé věří, že Haurus využíval francouzské policejní zdroje určené ke sledování kriminálníků a jiných podezřelých. Rovněž nabízel službu, která zákazníkovi sdělila informaci o tom, zdali je sledován policií či nikoli, popřípadě informace, které již policie vypátrala.

Parametr	**Průměrné ozbrojené přepadení**	**Průměrný kybernetický útok**
Riziko	Pachatel riskuje, že bude zraněn či zabit.	Bez rizika fyzické újmy.
Zisk	Průměrně 3–5 000 $.	Průměrně 50–500 000 $.
Pravděpodobnost dopadení	Dopadeno 50–60 % útočníků.	Dopadeno cca 10 % útočníků.
Pravděpodobnost odsouzení	Odsouzeno 95 % dopadených útočníků.	Z dopadených útočníků dojde k soudnímu projednávání pouze u 15 % útočníků a z nich je odsouzeno jen 50 %.
Trest	Průměrně 5–6 let, pokud pachatel při loupeži nikoho nezranil.	Průměrně 2–4 roky.

Tabulka 2.1: Srovnání běžné a kybernetické loupeže [11].

Tabulka 2.1 představuje statistiku FBI, která srovnává běžnou loupež s jednáním, které má povahu phishingového útoku.

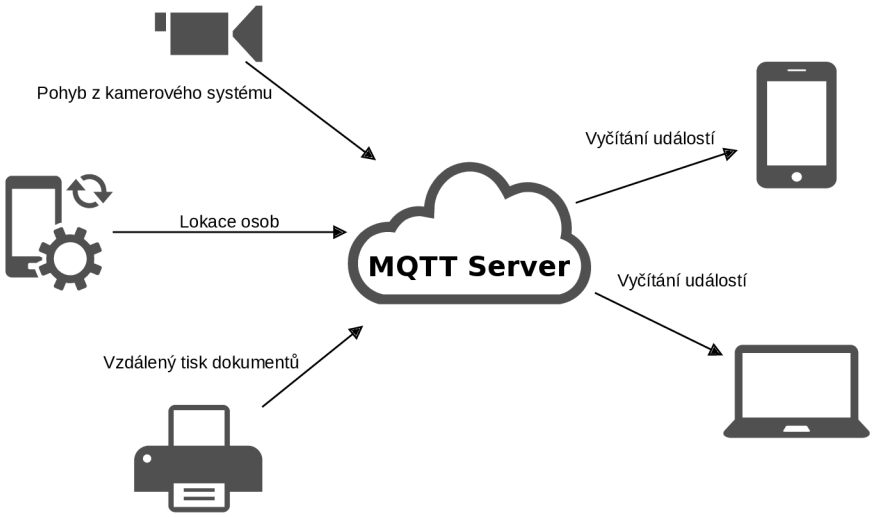
V současnosti dochází k masivnějšímu a stále většímu propojování různých počítačových systémů do kyberprostoru. Dá se tvrdit, že čím víc je připojených systémů a zařízení, tím větší je jejich zranitelnost a větší počet útoků. To lze znázornit graficky, dokonce v reálném čase. Například na adrese http://map.norsecorp.com/#/ můžeme pozorovat právě probíhající útoky.

Nelze pochybovat o tom, že kyberkriminalita je na vzestupu a představuje celosvětový problém [11]. Různé statistiky uvádějí částečně rozdílné škody způsobené kyberkriminalitou, mají však jeden společný jmenovatel. Všechny do těchto škod započítávají škody primární (nefunkčnost nebo výpadek služby, systému nebo celé infrastruktury) a sekundární (znovu připojování uživatelů, záchrana dat nebo obnova systémů). Europol ve své zprávě z roku 2014 uvádí, že kyberkriminalita stojí ekonomiku v globálním měřítku přibližně 300 miliard $ ročně. Od masového rozšíření internetu se komunita útočníků značně změnila. Už se většinou nejedná o jednotlivce, kteří páchali nezákonné jednání pro slávu, zábavu či překonávání překážek. V současnosti se zpravidla jedná o experty, kteří dělají svoji činnost s cílem profitovat a často jsou zapojeni do organizovaných skupin. Tato změna je spojena se třemi aspekty:

- závislost společnosti na internetu (na službách, technologiích aj.),
- kyberkriminalita se stala výnosným byznysem,
- žalostná gramotnost uživatelů, kteří využívají informační a komunikační technologie.

S rozvojem služeb postavených na principu as-a-service vzniklo v prostředí kyberkriminality mnoho platforem (typicky z digitálního podsvětí, darknet fór), kde lze nalézt služby, které je možné označit jako Crime-as-a-service [11]. To dalo za vznik černé digitální ekonomii, která za poplatek poskytne kterémukoli uživateli prostředky k páchání trestných činů. Rozmach těchto negativních aktivit jde ruku v ruce s fenoménem internetu věcí (IoT), který propojuje počítačové systémy (jednotlivá zařízení) s internetem, což představuje další významnou hrozbu. Mnoho distributorů či výrobců počítačových systémů, které lze řadit pod pojem IoT, neřeší bezpečnostní stránku věci. Jejich cíl je prostý – co nejdříve uvést na trh a prodat co možná největší počet zařízení. Této skutečnosti samozřejmě útočníci hojně využívají. Není výjimkou, že náklady spojené s vývojem v oblasti bezpečnosti produktu jsou těmi nejnákladnějšími součástmi vývoje, nicméně je to oblast, které je potřeba se věnovat. Mezi tyto oblasti můžeme konkrétně zahrnout například nezabezpečený přenos dat u kardiostimulátoru, letadlo, loď či auto, jež lze ovládat na dálku nebo chytrý dům (zabezpečovací systém, lednice, televize, rolety, jež lze ovládat vzdáleně).

Společnost Avast udělala v této oblasti výzkum a přináší velmi zajímavou zprávu [15]. Martin Hron se svým analytickým týmem popisuje velké mezery v zabezpečení chytrých domácností a to, jak útočník sestavil kompletní socioekonomický profil rodiny. Bezpečnostní inženýři z Avastu zkoumali síťový protokol MQTT (Message Queuing Telemetry Transport), který slouží pro výměnu stručných zpráv, jež používají nejrůznější zařízení internetu věcí jako základní komunikační protokol. Princip v jednoduchosti spočívá v tom, že například kávovar, lednice či teploměr publikuje data do MQTT serveru, ze kterého klienti (například mobilní aplikace) tyto zprávy vyčítají, viz. obrázek 2.1.



Obrázek 2.1: Zjednodušené schéma principu fungování MQTT protokolu.

Tento server je klíčovým bodem, který se stará o to, aby se zprávy z chytrých krabiček dostaly ke koncovým odběratelům. Tato komunikace může být šifrovaná a práva ke zprávám chráněna autorizací. Problém ovšem je, že se tak v praxi často neděje. Stačí si jen zjistit IP adresu nezabezpečené domácnosti a serveru přikázat, aby zaslal všechny své zprávy. Příkladem je rodina z Brightonu. Během pár sekund se útočníkům podařilo získat vše ze života mladého britského páru. Zprávy obsahovaly data ze senzorů v místnostech (útočník ví, jestli je dům prázdný), informace o pohybu mobilního telefonu v blízkosti domu (včetně jména majitele telefonu a stavu baterie) nebo přímo GPS souřadnice členů domácnosti. S rostoucím počtem chytrých krabiček v domácnostech si lidé zadělávají na velký problém. Přitom jen stačí splňovat základní bezpečnostní standardy.

Útočník může s daty provádět v podstatě tři základní věci:

- krást je (narušení principu důvěrnosti),
- měnit je (narušení principu integrity),
- zabraňovat vlastníkům v přístupu (narušení přístupu dostupnosti).

Dá se předpokládat, že s masivním rozšířením IoT se poslední dva typy útoků stanou velmi účinnými. V následujících sekcích budou představeny některé z nejhorších projevů kyberkriminality. Všechny případy a útoky nelze z důvodu rozsahu této práce vymezit, není to ani účel této práce. Budou ovšem popsány stěžejní témata, na základě kterých vznikala tato práce a která by se měla řešit ve spolupráci s akademiky a policií ČR [11].

2.1.1 Pirátský obsah

Existuje nespočet ilegálních torrentů nebo webů ke sdílení souborů jako například Pirate Bay, které dokonce patří mezi sto nejnavštěvovanějších webů na internetu. Nejlepší příklad lze vidět na teď už bývalé službě Megaupload [6]. Tento web při největším vytížení navštívilo padesát milionů uživatelů za den a jeho provoz činil čtyři procenta z celkového provozu na internetu. Majitelem byl hacker německé národnosti, který používal přezdívku Kim Dotcom. Hlavním produktem služby bylo padesát petabajtů kradených filmů, hudby, videoher, knih a softwaru. Megaupload vykazoval ročně zisk přibližně 25 000 000 $ z online reklam a 150

000 000 $ z poplatků uživatelů, aby mohli stahovat pirátský obsah vysokými rychlostmi. Díky těmto příjmům vedl Kim nadstandardní život [18].

Na pirátství nelegálně profituje mnoho kriminálníků, bohužel jim to často prochází a nejsou dopadeni. Při policejním vyšetřování se může stát, že důkazní materiál vedoucí k řešení případu je nelegálně získaný soubor. V takovém případě nás zajímá, na jakém serveru se daný soubor nachází, kdo a kdy ho tam vložil, popřípadě lidé, kteří si jej stáhli. Často se jedná o BitTorrent síť. Potom můžeme chtít vyhledat určitý magnet link. Regulární výraz pro vyhledání obecného magnet linku je například `\"magnet\:\?.*(xl|dn|xt|as|xs|kt|mt|tr).*\"`.

2.1.2 Drogy

Asi nejznámějším prodejním místem drog je Silk Road [6]. Jedná se o černý trh s drogami všeho typu, který lze nalézt na dark webu. Prodej sahá od malých individuálních objednávek, až po velké prodeje mezi jednotlivými prodejci. Na Silk Road lze nalézt mnohem nebezpečnější látky než jsou drogy, například skopolamin, který má výrazné účinky na psychiku, vyvolává poruchy paměti a neschopnost úsudku. Spekuluje se o použití této látky při výsleších Jana Bydžovského [6].

Obchodování s drogami má svůj specifický systém. Většinou to funguje tak, že si zákazník zaregistruje účet na nějakém webu, kde se drogy prodávají. K tomu potřebuje určitou částku nejčastěji bitcoinu, případně jiné kryptoměny. Při dokončení objednávky je potřeba zadat adresu příjemce, kam bude zboží dodáno. Většina prodejců doporučuje zadat svou reálnou adresu, což jde proti intuici každého, kdo se nechce nechat zatknout. Při provádění transakce je adresa šifrována, tudíž je znesnadněna práce policie, která chce dopadnout pachatele. Zboží je rovněž převáženo „tajně“, čímž se myslí to, že je zásilka vakuově zabalena, popřípadě skryta jiným způsobem, aby bylo zabráněno odhalení [6].

Zákazník může být zatčen policií při objednávání drog na internetu stejně snadno, jako kdyby si drogy kupoval osobně na ulici přímo od dealera. Z pohledu policie se může zdát zajímavé například s využitím jmen ve slovníku (`mike456|Ijohnnn`), hledat jméno uživatele, zdali si na nějakém podobném místě nezakupoval nějaké zboží.

2.1.3 Kradené bankovní účty a karty

Pravděpodobně žádná věc není v digitálním podsvětí více ceněna než kradené bankovní karty [18]. Ty lze získat na tzv. karetních fórech, kde lidé prodávají a nakupují debetní nebo kreditní karty většiny bank a zemí na světě. Většina kradených finančních dat je odcizena skrze malware, hacking nebo skimmery bankovních karet. Tato data jsou následně prodávána na dark webu pomocí tzv. *dumps*. Pod pojmem *dumps* si lze představit data z magnetických pásek, které jsou součástí karet. To může být například jméno držitele, číslo karty, doba expirace a CVV (verifikační číslo karty). Tyto cenné informace jsou poté zneužity kriminálníky k online nakupování nebo dokonce k zakódování dat na jinou padělanou kartu, kterou lze následně používat k zakoupení drahého zboží, následnému přeprodání a získání fyzických peněz.

Ačkoli se počet kradených karet rapidně zvyšuje, cena, za kterou si lze odcizenou kartu pořídit narůstá stejně tak. V roce 2010 byla průměrná cena takové karty 3 $, v roce 2017 to už bylo v průměru 38 $ [6]. Řeč je o běžných Visa a Mastercard kartách. Kradené karty jsou prodávány například na webech Mazafaka, Tortuga, CarderPlanet, ShadowCrew, Approven.su nebo na IAACA (mezinárodní asociace pro rozvoj kriminální aktivity). Téměř všechny tyto weby vyžadují registraci a jednotliví členové jsou prověřeni, zda nemají nějaké

styky s policií. Prodejci slibují karty s dlouhou dobou expirace a nabízejí garanci vrácení peněz, pokud jejich kradené karty nebudou fungovat. Takoví prodejci mají obrovské zisky. Globální ztráty sahají k 11 miliard $ ročně. Nejvíce obětí je z USA, což činí přibližně polovinu z celkové ztráty [6].

Do budoucna lze očekávat zlepšení situace minimálně co se týče kradených bankovních účtů, jestliže se rozšíří software bezpečnostních firem, jako je například ThreatMark[2]. V případě policie může být užitečné vyhledat na černém trhu například číslo konkrétní bankovní karty jakožto důkazní materiál k soudu použitím regulárního výrazu `(?:5[1-5][0-9]{2}`. pro MasterCard.

2.1.4 Zbraně

Pokud si chce někdo nelegálně zakoupit zbraň, může tak učinit online skrze dark web na stránkách jako Armory, Black Market Reloaded nebo LiberaTor [6]. Zbraně stylu Glock, Baretta a jiné 9 mm pistole s tlumiči se prodávají na denní bázi. Stejně tak útočné pušky včetně ruských AK-47 nebo Bushmaster M4 používané speciálními silami v Afghánistánu nevyžadují žádné čekací lhůty nebo jakýkoli typ kontroly. K sehnání jsou rovněž C-4 výbušniny, přičemž prodejci hrdě prohlašují, že dodávají do celého světa. Doprava je tady zásadní problém. Člověk si zkrátka nemůže nechat zaslat poštou AK-47 bez vzbuzení jakéhokoli podezření nebo kontroly. Tím pádem se dodavatelé z dark webu přizpůsobili a vymysleli různé strategie, jak zákazníkům dodat své zboží. Nejběžnějším způsobem je používat stíněné obaly, které mají rovněž vypadat jako jiné běžné zboží. Zbraně se také rozkládají na menší díly a zasílají zvlášť. Dalším způsobem jsou tzv. "dead drops", což jsou místa ve veřejném prostranství, kde dodavatel skryje složenou zbraň. Může ji například zahrabat pod zem, schovat mezi odpadky, popřípadě vybrat jinou skrýš. Po provedené platbě zákazník obdrží GPS souřadnice a popis, kde má jít hledat svou skrytou zásilku. V dnešní době jsou na prodej dokonce těžké vojenské zbraně. Například v těchto kruzích známý uživatel se jménem Bohica nabízí většinou ruční zbraně, ovšem dodává, že v případě potřeby dělostřeleckých zbraní, MANPADS, APC nebo jiných těžkých zbraní, jsou na skladě a za poplatek poskytne bližší informace, stačí poslat zprávu s PGP šifrováním. Pokud by chtěli policisté najít důkazní materiál o prodeji nějakého typu zbraně, využili by například slovník `AK-47|C-4|Bushmaster|Glock|Baretta`.

2.1.5 Obchodování s lidskými orgány

Na celém světě je obrovský černý trh s částmi lidského těla [6]. Ačkoli jsou uvedené ceny orientační, například ledviny lze prodat za 200 000 $, srdce za 120 000 $, játra za 150 000 $, rameno za 500 $ a pár očí za 1 500 $. Velký zájem je dokonce i o kůži. Měla by vyvstat logická otázka, odkud tyto části lidského těla pochází? Od živých i mrtvých. Přestože žijeme v 21. století, stále se hojně vykrádají hroby a mnoho márnic po celém světě prodává části těla mrtvých lidí, kteří jim jsou svěřeni s předpokladem, že rodina zesnulého se o tom nedozví. Ještě horší situace je u chudých lidí, na které je aktivně cíleno jak online, tak osobně a kteří jsou v takové finanční nouzi, že dobrovolně prodávají své orgány. Na druhé straně jsou lidé se špatným zdravotním stavem, kteří beznadějně čekají na dárce orgánů. Jen v USA je v pořadníku na transplantaci ledvin více než 100 000 čekajících pacientů, což vyústilo v čekací lhůty až deseti let. Většina z těchto pacientů se bohužel nedočká. Důsledek je takový, že mnoho velmi bohatých lidí hledá dárce a stal se z toho neoprávněný obchod s lidskými

[2]Česká firma pohybující se na poli bankovního zabezpečení (https://www.threatmark.com/).

orgány. Crime Inc. má celou divizi obchodníků, kteří slouží jako prostředníci skrze všechny kontinenty a snaží se spojovat prodávající, kupující a zprostředkovávají lékaře, chirurgy a nemocniční zařízení. World Health Organization odhaduje, že přibližně každou hodinu je prodán nějaký orgán v digitálním podsvětí. Orgány můžou pocházet od popravených vězňů v Číně, žen z Indie, které jsou manžely přinuceny nebo Syrskými uprchlíky. Ačkoli má například ledvina pro dárce cenu přibližně 200 000 $, těmto dárcům je vyplacena pouze malá část ve prospěch kriminálníků, kteří obchod zprostředkovávají. Tím hůř, že se těmto orgánům nedostává patřičné pooperační péče a odumírají, než najdou potřebného [6].

Stále více se tyto aktivity odehrávají online, v chatovacích místnostech a skrze dark web. Často chudoba v zemích jako Indie, Bulharsko nebo Srbsko vyprovokuje obyvatele k tomu, aby si zjistili svou krevní skupinu a následně podávali nabídky svých orgánů na černém trhu. Například v Číně kontaktují překupníci mladé lidi na internetových fórech se slogany "Daruj ledvinu, kup si nový iPad". Pro policejní účely může být zajímavé prohledat takové fórum a najít všechna vlákna, kde se nelegálně obchoduje s lidskými orgány. Postačil by lehce rozšiřitelný slovník `kidney|heart|liver|skin|shoulder|eye`, vůči kterému by se analyzovala všechna nalezená slova.

2.1.6 Dětská pornografie

Stejně jako v Danteho Infernu, dark web má rovněž devátý kruh pekla. Je to místo, kde lze nalézt ty nejodpornější akty násilí na těch nejslabších a nejzranitelnějších z naší společnosti. V hluboce znepokojující zprávě Europolu (Evropský policejní úřad) zaznamenaly orgány činné v trestním řízení zvyšující se počet webů v digitálním podsvětí, které provozují online přenos zneužívání a znásilňování dětí [6]. Crime Inc. a jiné pedofilní sítě organizují tzv. pay-per-view (platba za zhlédnutí) znásilnění dítěte na požádání. Organizované kriminální sítě zejména v Asii poskytují pedofily s možností připojit se živě skrze Tor do video kanálů s vestavěnou funkcionalitou chatu, kde můžou uživatelé z celého světa za poplatek zadávat požadavky násilníkovi a řídit tím jeho kroky. V jednom z incidentů vyšetřovaném policií si uživatelé připojení skrze Tor objednali skupinu mužů na znásilnění osmiletého dítěte, kdy v reálném čase řídili kroky těchto mužů a udávali tak scénář a to vše za 100 $.

Jelikož se tyto aktivity odehrávají na dark webu, a protože je video streamováno namísto stahování, důkaz o spáchání trestného činu není nikde permanentně zaznamenán, kromě samotných týraných obětí, které přežily brutalitu svých trýznitelů [8]. Všechny nelegální činnosti tohoto typu jsou nabízeny na prodej v digitálním podsvětí a generují obrovský zisk organizacím jako je Crime, Inc. Tento trend za poslední dobu rychle akceleroval především díky tajnému nelegálnímu financování. Realita je taková, že se pravděpodobně nikdy nepodaří zcela vymítit tyto odporné aktivity, ale každý by měl nějakým způsobem přispět minimálně k jejich omezení. Jak již bylo řečeno, najít důkazy o trestné činnosti tohoto typu je netriviální problém, můžeme se ovšem zaměřit alespoň na fóra, případně internetové diskuze, kde bychom mohli pomocí slovníku (například `child|rape|girl`) začít svá pátrání. Takový typ vyhledávání slov ve slovníku pravděpodobně bude generovat mnoho *false positive* případů, tudíž by bylo vhodné vytvořit buď konkrétnější regulární výraz nebo upravit slovník tak, aby se v něm nevyskytovala častá a běžná slova.

2.2 Vyšetřování kyberkriminality

Tato sekce vymezí některé kriminalistické a procesněprávní aspekty souvisejících s odhalováním, prověřováním a vyšetřováním kyberkriminality. Sekce 2.2.1 hovoří obecně o digitální

stopě, jaké jsou její druhy a jak se jí zbavit. Dále sekce 2.2.2 rozebírá všechny kroky postupu vyšetřování a nakonec sekce 2.2.3 popisuje typy důkazů a procesy při dokazování kyberkriminality. Kyberkriminalita může být namířena proti počítačům, softwaru, hardwaru, sítím, datům, nebo v ní vystupuje pouze počítač jako nástroj k páchání trestného činu [2]. Další možností je síť a k ní připojená zařízení, ve které se odehrává trestná činnost. Nejen pro policii je velmi obtížné sledovat projevy kyberkriminality, jelikož prostředí, ve kterém se odehrávají je obtížně vnímatelné – dění v kyberprostoru můžeme sledovat pouze za pomocí jiného zařízení. Jak již bylo řečeno, značná část kyberkriminality přenáší známé druhy protiprávního jednání do prostředí kyberprostoru, kde je lze chápat rychleji, lépe, efektivněji, než v reálném světě. Jak již bylo řečeno, za ryze kybernetické útoky, které nemají v reálném světě obdobu, můžeme zařadit například hacking, DoS, DDoS nebo botnety.

S ohledem na specifický charakter kyberkriminality bylo třeba v průběhu posledních let vyvinout novou zvláštní metodiku, která upřesňuje, rozvíjí a specifikuje typické prostředky dokazování trestného činu. Tento postup je stále doplňován tak, jak dochází k dalším novým způsobům kybernetických útoků. Samotné vyšetřování by měly vést specializované týmy složené z odborníků na danou problematiku [11].

2.2.1 Digitální stopa

Stále častěji se součástí důkazního materiálu, a to nejen v oblasti počítačové kriminality, stávají rovněž digitální stopy [11]. Definicí digitální stopy je mnoho, různí autoři často používají synonyma pro označení digitální stopy, nám zde však postačí krátká výstižná definice – *digitální stopa je informace zanechaná uživatelem v prostředí internetu nebo jako součást souborů*. Pojem digitální stopa je pro účely diplomové práce stěžejní, jelikož se snaží právě o sběr aktivních digitálních stop. Je proto žádoucí pochopit, co to digitální stopa je, se všemi vlastnostmi a náležitostmi. Informace, které po sobě uživatelé na internetu zanechávají, se dělí na aktivní:

- profily nebo příspěvky zanechané na sociálních sítích,
- e-maily, sms zprávy, historie chatu,
- různé typy úředních údajů.

a pasivní:

- IP adresa,
- vyhledávané výrazy na internetu,
- cookies (údaje uložené ve spojitosti s konkrétním webem),
- lokace, poskytovatel připojení.

Jak je jistě patrno, mezi aktivní se řadí ty informace, které o sobě uživatel zveřejní prostřednictvím různých služeb vědomě a dobrovolně [11]. Naopak pasivní informace vznikají v prostředí internetu bez našeho přímého záměru. Realita je taková, že jakákoliv aktivita v online prostředí může být zaznamenána a uložena. Tyto informace se často zneužívají například pro:

- krádež osobních informací (údaje z kreditních karet, rodné číslo, e-mailová adresa),
- kyberšikana,

- kyberstalking,
- zdroj informací pro personalisty,
- sledování návyků uživatelů (realizováno třetími stranami, sběrateli dat a reklamními společnostmi).

Existují však způsoby a nástroje pro správu a ochranu digitálních stop. Pro správu aktivních stop je účinné:

- používat více přihlašovacích jmen,
- rozvážně (nejlepe vůbec) publikovat fotografie, videa a osobní údaje,
- vhodně nastavit soukromí, obzvláště u sociálních sítí, které poskytují základní konfiguraci,
- vhodně nastavit zabezpečení svého internetového prohlížeče,
- Me on the web (služba, která sleduje nově zveřejněné informace o uživateli v určité oblasti).

Pro správu pasivních stop potom:

- Do not Track (umožňuje částečně zamezit nechtěnému sledování),
- Opt-outs (umožňuje uživateli vyvázání se z nechtěné služby),
- různá softwarová řešení (například doplňky internetového prohlížeče),
- anonymní režim prohlížení.

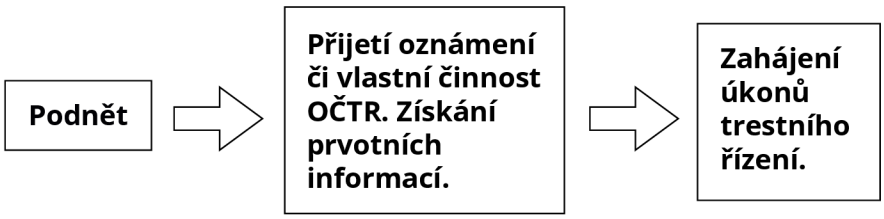
Žádná z těchto technik není dokonalá a uživatel si může být jist, že v každém případě na některých místech zanechá svou digitální stopu. Zpráva o webovém sledování [14] konkrétně rozebírá rozmanitý ekosystém webových služeb třetích stran, které sbírají informace o uživatelích a dále je využívají pro cílenou reklamu. Jedno z takových expandujících odvětví internetu jsou sledovací služby třetích stran, které sledují uživatele na každém kroku. Poskytují inzertní a analytické služby a služby, které nabízejí sledování a profilování uživatelů. Ačkoli některé uživatelské interakce s těmito službami mohou být úmyslné a explicitní (například sdílení obsahu nebo kliknutí na *líbí se mi*), většina takových interakcí s těmito službami není explicitní a úmyslná. Uživatelé si většinou neuvědomují přítomnost takových služeb vůbec. Největší obavy ohledně soukromí vzbuzuje narůstající obchod s osobními informacemi mezi těmito službami a jejich všudypřítomnost a možnost sledovat uživatele z jedné stránky na jinou, často na různých webech.

Smazat takovou stopu je v dnešní době prakticky nemožné [11]. Závisí na samotném uživateli, jakým způsobem bude své aktivní stopy monitorovat. Omezení aktivních digitálních stop je možné docílit použitím vícero přihlašovacích jmen a e-mailových adres. Další účinnou metodou je použít Tor, což je software zajišťující anonymitu uživatele při pohybu na internetu tím, že zabrání serverům získání reálné IP adresy počítače. Oproti aktivním, pasivní stopy mají určitou dobu platnosti uchování dat. Uživatelé mohou zabránit maximálně tak dodatečnému sběru dat, například softwarovým řešením nebo správou cookies. Poté zbývá vyčkat, než uplyne doba uložení dat.

2.2.2 Postup při vyšetřování

V této sekci budou z policejního hlediska jednoduše a srozumitelně popsána specifika úkonů spojených se zjišťováním, zda se skutek stal, zda je tento skutek trestným činem a kdo je jeho pachatelem. Každé trestní řízení začíná sepsáním záznamu o zahájení trestního řízení [12]. Orgány činné v trestním řízení se o skutečnosti, že byl spáchán trestný čin v rámci internetu nejčastěji dozvídají na základě trestních oznámení. Při řešení kyberkriminality je nejdůležitější zajištění prvotních informací a důkazů. V této fázi trestního řízení je potřeba co nejpreciznějí zajistit informace týkající se kybernetického útoku. Pokud je to možné, je třeba zajistit data v nezměněné podobě (originály e-mailových zpráv, paměťová média či celý počítač, popřípadě jen kopie nebo printscreeny) [12].

Velmi důležitým zdrojem informací je způsob připojení počítače do sítě a přidělení IP adresy [11]. Pro správnou identifikaci počítače je potřeba znát kromě IP adresy i datum s přesným časem připojení do sítě. K vlastnímu zjištění koncového počítačového systému je dále zapotřebí znát technické principy připojování do sítě (přidělení IP adresy, konfigurace NAT). Na základě procesu připojování do sítě či k různým službám (e-mail, cloudová uložiště, sociální sítě) je možné najít zdroj útoku. Další informace lze získat od jednotlivých ISP (poskytovatelů připojení k internetu). Ti uchovávají informace o zařízeních, včetně informace o IP adrese, délce a času používání dané služby. Pokud je z obsahu trestního oznámení patrno, že jde o trestný čin, případně kdo je jeho pachatelem, je sepsán záznam o zahájení úkonů trestního řízení.



Obrázek 2.2: Zjednodušené schéma zahájení úkonů trestního řízení.

Celý popsaný postup lze shrnout do tří kroků, viz. obrázek 2.2.

2.2.3 Dokazování kyberkriminality

Jak už samotný název sekce napovídá, bude řeč o důkazech a jak kyberkriminalitu dokázat. Za důkaz můžeme označit vše, co může přispět k objasnění věci [12]. Například výpověď obviněného, svědků, znalecké posudky nebo různé listiny. Při objasňování, odhalování a vyšetřování kyberkriminality je možné užít všechny dostupné prostředky dokazování. V této sekci budou popsány některé důkazy a důkazní prostředky, které nejčastěji slouží k odhalování kyberkriminality. Následuje výčet typů důkazů:

- Věcné důkazy – to jsou takové důkazy, na kterých byl trestný čin spáchán. Co se týče kyberkriminality, nejčastěji jsou věcnými důkazy počítačové systémy a paměťová média obsahující data. Data můžou být rovněž uložena v cloudu namísto paměťových médií.
- Listinné důkazy – za listinný důkaz lze považovat takové listiny, které potvrzují či vyvracejí kyberkriminalitu. Takovým typickým důkazem jsou data či informace po

jejich vytištění. Může se jednat o papír nebo jiné médium schopné zaznamenat psaný či kreslený projev.

- Digitální důkazy – tento typ důkazů zatím není zaveden mezi kategoriemi důkazů. Lze ovšem nalézt hlasy některých bezpečnostních odborníků, kteří tvrdí, že by tento typ důkazu měl být oficiálně zahrnut. Za digitální důkaz by se potom považovala jakákoli data či informace, jež byly vytvořeny, přeneseny, uloženy či modifikovány za použití počítačového systému a mohou být prostředkem k odhalení trestného činu. Diplomová práce se opírá o tento poslední bod a snaží se pomoci získat digitální důkazy, konkrétně z webových stránek.

Dokazování kyberkriminality obnáší několik úkonů [12]. První činností při dokazování kyberkriminality je zajištění digitálních stop. Zajišťování digitálních stop důležitých pro trestní řízení je zpravidla prováděno orgány činnými v trestním řízení. Nejčastějšími úkony jsou domovní a osobní prohlídka, vydání a odnětí věci a ohledání místa činu.

Domovní prohlídka je jedním ze zajišťovacích úkonů umožňujících velmi intenzivní zásah do základních lidských práv a svobod. Prohlídku je možné vykonat, existuje-li důvodné podezření, že by se v bytě nebo jiném bydlení mohla nacházet věc nebo osoba důležitá pro trestní řízení. Domovní prohlídku je oprávněn nařídit předseda senátu nebo soudce a vykonává jej policejní orgán.

Vydáním věci se rozumí povinnost předložit či vydat věc důležitou pro trestní řízení na výzvu orgánů činných v trestním řízení. Oprávnění vyzvat k vydání věci má předseda senátu, státní zástupce a policejní orgán. Pokud osoba výzvy neuposlechne, může jí být věc odňata, popřípadě uložena pokuta. V rámci kyberkriminality se v praxi bude nejčastěji jednat o zajišťování počítačových systémů (počítače, servery, tablety, mobilní telefony, datová uložiště, síťová zařízení) včetně periferií (3D tiskárna), paměťových médií a dalších prostředků sloužících ke komunikaci (SIM karta).

Kapitola 3

Sběr dat na internetu

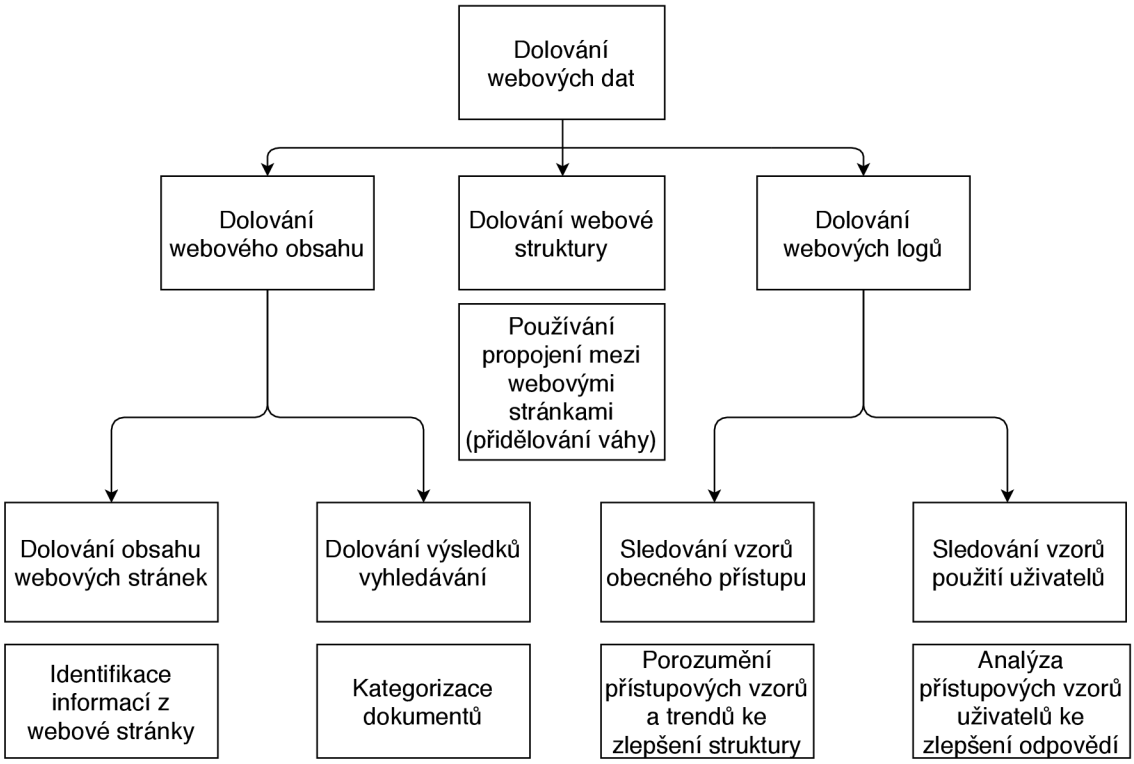
Během posledních několika let se *World Wide Web* stal doslova napumpovaný informacemi a v některých případech se stalo velmi obtížným najít konkrétní informaci. Jakožto řešení tohoto problému vznikla disciplína *web mining* [9]. Pod pojmem *web mining* si lze představit aplikaci různých technik pro dolování dat. Je to proces objevování potenciálně užitečných a doposud neznámých informací. *Web mining* [9] se využívá pro:

- získávání relevantních informací,
- vytváření nových znalostí z relevantních dat,
- perzonalizaci informací,
- nauku o zákaznících a obecně o uživatelích, pokud jde o nějaký typ služby.

Dolování webového obsahu je podmnožina tohoto pojmu. Se zvyšující se složitostí webů se rovněž enormně zvyšuje množství informací na nich obsažených. Tudíž extrakci dat, kterou by uživatel mohl požadovat se stala náročným a často zdlouhavým úkolem [9]. Výsledkem je to, že se dolování dat z webů stalo základní technikou pro extrakci hodnotných informací z internetu. *Web mining* [13] je dále rozčleněn do tří kategorií – dolování webového obsahu, dolování struktury webu a dolování užívání webu:

- při dolování webového obsahu zkoumáme objekty jako text, obrázky nebo multimedia,
- při dolování webové struktury pracujeme na základě struktury webu, kdy například hledáme odkazy specifikované pomocí URL,
- v případě dolování webového užívání jsou oblastí zájmu logovací soubory, které obsahují navigační vzorce uživatelů, čili jak se uživatelé pohybovali na daném webu.

Pro účely této práce je podstatný první a druhý bod, jelikož nám jde o analýzu textu obsaženého na webech a o strukturu webu, respektive o odkazy obsažené na webu. *Web mining* obecně pomáhá porozumět chování zákazníka, nalézt úzké hrdlo ve výkonnostní stránce webu a nepřímo podporuje byznys firem, které tyto techniky praktikují [9]. Dolování obsahu z webu má rovněž svou vlastní taxonomii, viz obrázek 3.1. Základní popis již byl zmíněn v předchozím výčtu. Následovat bude podrobnější popis typů dolování dat [13].



Obrázek 3.1: Taxonomie dolování webových dat [9].

První dvě úrovně byly popsány, čili rozdělení dolování webových dat na dolování webového obsahu, webové struktury a webových logů. Dolování webového obsahu dále zkoumá obsah webu, jakožto i výsledky vyhledávání [9]. To je klasifikováno jako dolování obsahu webových stránek a dolování výsledků vyhledávání.

Dolování obsahu webových stránek spočívá v analýze dané stránky a vyhledání textu nebo jiné formy informace, zatímco dolování výsledků vyhledávání zkoumá další vyhledávání stránek na základě stránky zkoumané v předchozím kroku. Dolování webové struktury analyzuje strukturu dokumentu jako HTML nebo XML a získává informace o organizaci stránky. Tato technika používá propojení mezi webovými stránkami na základě váhy, která je jim v průběhu zpracování přiřazena. Dolování webového užití (webových logů) je aplikace technik za účelem porozumění vzorů webového používání uživateli. Tato technika doluje data z logovacích souborů, profilů uživatelů, informací o jejich sezení, souborů cookies, uživatelských dotazů, záložek, kliků myši a dalších uživatelských interakcí.

Dolování webového užití má tři fáze – předzpracování, odhalení vzorů a analýza vzorů. Tento typ dolování je dále klasifikován na sledování vzorů obecného přístupu a sledování vzorů použití uživatelů. První z těchto dvou doluje informace na základě historie používání dané stránky, kterou uživatelé navštíví. Takovým způsobem lze porozumět vzorům a trendům v používání webu a zlepšit tak strukturu webu. Pokud se zaměřujeme na konkrétního uživatele, jedná se o druhý typ. Analyzují se přístupové vzory konkrétního uživatele vedoucí ke zlepšení specifických odpovědí [9].

Pro tuto práci je podstatné dolování webového obsahu, konkrétně textu obsaženého na specifických stránkách. Proto bude nyní soustředění zaměřeno převážně na tuto oblast. Obecně jde o běžnou techniku, kdy prohledáváme web skrze jeho obsah. Rovněž vyhledávací procesory dělají navíc ke své činnosti dolování webového obsahu. Tento typ dolování

odkazuje na objevení užitečných informací z webového obsahu (text, obrázky, videa). Při dolování dat využíváme dvou přístupů – použití databáze nebo agenta [13]. Databázový přístup se skládá z databáze obsahující schémata a atributy s definovanými doménami. Při přístupu s agenty existují tři druhy agentů:

- agenti inteligentního vyhledávání automatizovaně hledají informace na základě specifického dotazu (princip agenta inteligentního vyhledávání je využit v této práci),
- filtrující agenti používají různé techniky pro filtraci dat dle předem definovaných instrukcí,
- perzonalizovaný agent se učí preference uživatelů, vytváří uživatelské profily a na jejich základě modifikuje vyhledávání relevantních dokumentů.

V oblasti dolování webového obsahu jsou pro tuto diplomovou práci stěžejní převážně techniky dolování nestrukturovaného obsahu (převážně textu). Dolováním nestrukturovaných dat získáváme neznámé informace. Dolování textu spočívá v extrakci dříve neznámých informací extrakcí informací z různých textových zdrojů [13]. Mezi nejznámější techniky patří extrakce informací, sledování témat, sumarizace, kategorizace, shlukování a vizualizace informací. V diplomové práci se užívá techniky extrakce informací, která používá vzory k získání shody textu. Vhodnými metodami jsou vyhledávání ve slovníku (hledání klíčových slov) a regulární výrazy (vyhledání celých frází dle specifického předpisu). Tato technika je vhodná při velkých objemech dat [13].

V následujících sekcích budou konkrétněji a s technickými detaily představeny techniky, způsoby a možnosti hledání informací, jejich analýza a následná archivace. Nejprve bude v sekci 3.1 objasněno, zdali a do jaké míry je sběr a archivace informací z webových stránek legální. Následuje sekce 3.2, kde budou představeny existující metody a nástroje pro sběr informací a jejich analýzu. V této sekci půjde o ukázky různých technik stažení statických a dynamických webových stránek a jejich analýzy. V poslední sekci 3.3 budou popsány již existující nástroje pro procházení webů, hledání, analýzu a archivaci dat. Na rozdíl od předchozí sekce půjde o hotové komplexní systémy schopné stahovat, analyzovat a archivovat několik webových stránek paralelně, čímž se přibližujeme reálné praxi.

3.1 Legálnost

Když se vytváří nástroj na sběr webových dat, jsou zde citlivé oblasti, na které musí vývojář brát zřetel. Nezodpovědné sbírání dat může být pro druhou stranu minimálně otravné, ovšem může to v některých případech hraničit až s ilegalitou. Je potřeba se vyvarovat dvěma primárním věcem – útoku DoS (odepření služby) a porušení autorských práv [10].

V prvním případě, DoS útoku, musíme brát na vědomí, že typický uživatel může navštěvovat novou stránku například každé tři sekundy [10]. Oproti tomu, typický sběrač webových dat stahuje během sekundy desítky stránek, čili může vygenerovat několikanásobný provoz oproti běžnému uživateli. To je naprosto validní důvod minimálně k obavám na straně majitele webu. Proto se jako dobrá praktika považuje přiškrcení provozu natolik, aby generovaný provoz byl porovnatelný s provozem běžných uživatelů. Doporučuje se rovněž monitorovat dobu odezvy a jestliže se zvyšuje, redukovat intenzitu generovaného provozu.

Dalším omezením jsou autorská práva. Doporučuje se přečíst si oznámení o autorském právu na dané webové stránce před zahájením jakékoli operace a ujistit se, zda bylo dobře

porozuměno tomu, co daný web schvaluje a co už ne. Většina webů dovoluje zpracovávat informace ze svých stránek tehdy, pokud nebudou reprodukovány dále s tvrzením, že data jsou vlastnictvím majitele sběrače dat [10].

Mezi dobré praktiky se dále řadí použití *User-Agent* pole v hlavičce každého HTTP dotazu [10]. Tím dáme majiteli webu informaci o tom, kdo jsme a co máme v plánu dělat se získanými informacemi. Tohle pole může být vyplněno například webovou adresou nebo jménem, což identifikuje majitele sběrače dat. Majitel webu si poté může zjistit, kdo a jak s jeho daty nakládá. Další dobrou praktikou je umožnění majiteli webu omezit přístup na některá místa svého webu. K tomu slouží soubor *robots.txt*, který je čitelný pro člověka a obsahuje pravidla specifikující, která místa mají být sběrači dat dostupná a která nikoli. Příklad tohoto souboru se nalézá například na adrese `https://www.google.com/robots.txt`. Co se týče zákonů, ty jsou v každé zemi jiné, tudíž se doporučuje vyhledat právní profesionály v případě rozsáhlého používání webových sběračů dat [10].

Diplomová práce se snaží držet všech dobrých praktik souvisejících se sběrem webových dat. Důležitá je především konfigurace, kde si uživatel může naspecifikovat parametry sběru – odezvu mezi dotazy, patřičné nastavení *User-Agent* či následování pravidel obsažených v souboru *robots.txt*.

3.2 Existující metody a nástroje

Hlavní pointou této sekce je poukázat na to, že není třeba vymýšlet kolo. Jestliže existují open source nástroje, které již mají uživatelskou základnu, jsou spolehlivé a dostatečně otestované, byla by chyba vyvíjet nástroj se stejnou nebo obdobnou funkcionalitou. Dále budou v této sekci popsány metody, techniky a knihovny pro sběr a analýzu dat z webu [7]. V sekci 3.2.1 bude objasněna navigace a manipulace s HTML elementy. Následující dvě sekce budou rozebírat techniky získávání dat z webu. Konkrétně sekce 3.2.2 bude řešit HTML stránky bez dynamického Javascript obsahu, zatímco sekce 3.2.3 osvětlí získání HTML stránek s dynamickým obsahem generovaným pomocí Javascriptu.

3.2.1 XPath, CSS selektor

Když webový prohlížeč zobrazuje konkrétní stránku, vytváří si model obsahu stránky v reprezentaci *document object model* (DOM). DOM[1] je hierarchická reprezentace obsahu celé stránky, informací o struktuře, stylu, skriptů a odkazů na jiný obsah. Je kritické správně porozumět této struktuře, aby mohla být efektivně sbírána data z jednotlivých stránek.

Nejprve bude popsán XPath[2], což je dotazovací jazyk pro hledání uzlů v XML dokumentech. Znalost tohoto jazyka je esenciální pro každého, kdo se chce zabývat sbíráním dat z webů. Mezi jeho hlavní přednosti patří:

- jednoduchá navigace v modelu DOM,
- sofistikovanější a praktičtější než jiné selektory jako například CSS selektory nebo regulární výrazy,
- nabízí velkou množinu vestavěných funkcí a rozšiřitelnost pro další uživatelské funkce,
- má širokou podporu mezi analyzačními knihovnami a platformami pro sběr dat z webů.

[1] `https://www.w3.org/DOM/`
[2] `https://www.w3.org/standards/techs/xpath`

Výsledkem XPath výrazů může být řetězec, logická hodnota, číslo nebo množina uzlů. Množina uzlů jakožto návratová hodnota XPath výrazu je nejběžnější a nejužitečnější. Taková množina obsahuje pouze ty uzly, jejichž poloha je relativní k aktuálnímu uzlu. Příklady užití:

```
xpath("/html/body/div/table/tr")
xpath("/html/body/div/table/tr[@class='planet']")
xpath("/html/body/div[@id='planets']/table/tr[@id!='planetHeader']")
```

První XPath výraz vrátí množinu uzlů, kterým odpovídá cesta k tělu HTML, dále element *div*, *table* a následně všechny řádky *tr* tabulky. Pokud je v HTML dokumentu takových tabulek vícero, budou všechny zahrnuty do výsledné množiny. Druhý příkaz má stejné chování jako první, ovšem s tím rozdílem, že do výsledné množiny zahrne pouze ty řádky, které mají atribut *class* roven řetězci *planet*. Poslední příkaz zahrne pouze ty tabulky, které spadají pod element *div* s atributem *id* rovnu *planet*. Následně se vyberou pouze ty řádky, které nemají atribut *id* roven *planetHeader*.

Oproti tomu, CSS selektory[3] jsou vzory používané pro hledání elementů a obvykle se používají pro definici elementů, na které je aplikován libovolný CSS styl. Mohou být rovněž použity pro hledání elementů v modelu DOM. CSS selektory se hojně používají díky kompaktnosti, jednoduchosti a znovupoužitelnosti v kódu, kde má XPath díky své široké funkcionalitě nedostatky. Příklady použití:

```
cssselect("tr.planet")
cssselect("tr#planet3")
cssselect("tr[name='Pluto']")
```

První CSS selektor vybere všechny tabulky a z jejich obsahu pouze ty řádky, které obsahují atribut *class* roven řetězci *planet*. Druhý selektor se liší od prvního pouze tím, že nezkoumá atributy *class*, nýbrž atributy *id*. Třetí selektor po vzoru předchozích hledá ty řádky, kde se vyskytuje atribut *name* roven řetězci *Pluto*.

Na příkladech lze vidět, že CSS selektory jsou jednodušší a kratší než XPath příkazy, ovšem nemají takové vyjadřovací schopnosti a funkcionalitu. V případě složitých dotazů se tudíž většinou uchyluje k použití XPath. Je rovněž nutno podotknout jeden implementační detail, kdy CSS selektory při tomto způsobu použití interně využívají XPath výrazy, které se následně aplikují na dokument. Díky této režii CSS selektory nedosahují takové výkonnosti jako XPath.

3.2.2 Requests/urllib3 a Beautiful Soup

Tato sekce prezentuje jednoduchost sběru HTML stránek a jejich analýzu v programovacím jazyku Python. Nejprve bude popsán způsob stažení stránky s knihovnami Requests a urllib3, následně analýza a vyhledávání informací ze stažených dat pomocí knihovny Beautiful Soup.

Knihovna Requests[4] pod licencí Apache2 slouží pro práci s HTTP protokolem. Cílem knihovny je jednoduchá práce s HTTP dotazy a především přívětivost pro uživatele. Příklad získání obsahu stránky:

[3] https://www.w3.org/TR/2011/REC-css3-selectors-20110929/
[4] http://docs.python-requests.org/en/master/

```
r = requests.get(
        'https://api.test.service.com/user',
        auth=('user', 'pass')
).text
```

V proměnné *r* se nachází obsah stažené stránky. Knihovna urllib3 stejně jako Requests slouží pro práci s HTTP protokolem. Vydána je pod licencí MIT. urllib3 je přítomna ve velké části Python ekosystému. Jedná se o pokročilejší knihovnu, která se vyznačuje:

- podporou několika vláken,
- optimalizací sdružováním mnoha připojení,
- podporou SSL/TLS,
- nahráváním souborů s *multipart* kódováním,
- podporou HTTP přesměrování,
- podporou *gzip* a *deflate* kódování,
- podporou proxy serverů.

Příklad užití:

```
r = urllib3.PoolManager().request(
        'GET',
        'https://google.com/robots.txt'
).data
```

Stejně jako v předchozím případě se v proměnné *r* nachází obsah stažené stránky. Po tom, co je k dispozici obsah HTML stránky, přichází na řadu analýza a hledání informací. K tomu slouží knihovna Beautiful Soup pod MIT licencí, což je Python knihovna pro hledání dat v HTML a XML objektech. Podporuje navigaci, vyhledávání i modifikaci stromu modelu DOM. Příklad použití:

```
s~= BeautifulSoup(r, 'html.parser')
s.find_all('a')
s.find(id='link3')
```

Po inicializaci jsou uvedeny dva příklady hledání elementů v DOM modelu. První příklad vyhledá všechny značky *a*. Ty definují odkazy na další stránky. Druhý příklad hledá všechny elementy s atributem *id*, jehož hodnota se rovná řetězci *link3*.

3.2.3 Selenium a PhantomJS

Selenium je velmi užitečný a rozšířený nástroj na získávání dat z webů, který byl původně vyvinut pro testování webových stránek [17]. Dnes se rovněž používá k získání přesného vzhledu webové stránky tak, jak je vykreslena ve webovém prohlížeči. Selenium pracuje způsobem automatizace webového prohlížeče ke stažení stránek, získání požadovaných dat, pořizování snímků stránky a upozornění na určité akce prováděné na načtené stránce. Selenium neobsahuje žádný webový prohlížeč, namísto toho se integruje s prohlížeči třetích stran (automatizuje prohlížeče). Pokud se například spustí selenium s webovým prohlížečem

Firefox, otevře se nová instance prohlížeče, která načte požadovanou stránku a provede předem nakonfigurované akce v programu. Ačkoli je pěkné pozorovat stránku při vykonávání požadovaných akcí, v praxi chceme, aby program nebo skript běžel na pozadí a nevytvářela se žádná grafická instance webového prohlížeče. Pro takové učely lze použít například nástroj PhantomJS, jakožto alternativa k běžným prohlížečům.

Nástroje, jako je například PhantomJS, se označují jako *headless* prohlížeče [17]. Dalšími takovými prohlížeči jsou například Google Chrome, Firefox, HtmlUnit nebo SimpleBrowser. Pracují tak, že načtou webovou stránku do paměti a vykonají JavaScript, který se na stránce nachází, ovšem bez jakéhokoli grafického zobrazení. Kombinací selenia a *headless* prohlížeče získáme sběrač webových dat, který je schopen manipulovat s cookies, JavaScript kódem, hlavičkami nebo analyzovat text jako v předchozí sekci například Beautiful Soup. Knihovna selenium je API (aplikační rozhraní) volané na *WebDriver*. *WebDriver* se zvenku jeví jako prohlížeč, který dokáže načíst stránku, ale může být rovněž použit na oblasti analýzy textu, hledání elementů nebo interakci s elementy na stránce (vyplnit text do textového pole, kliknout na tlačítko). Selenium využívá na hledání elementů v html souborech XPath a CSS selektory. Touto cestou se dá jednoduše pracovat se stránkami obsahující dynamický JavaScript obsah.

3.3 Existující řešení

Stejně jako v předchozí sekci zde bude stručný popis existujících nástrojů, ovšem s tím rozdílem, že už nepůjde o metody a postupy, nýbrž o hotové komplexní systémy, knihovny a programy, které mají širší funkcionalitu. Na některých z nich je rovněž založená tato diplomová práce, konkrétně je využit Scrapy a Lemmiwinks (v kapitole 4 budou vysvětleny důvody k tomuto rozhodnutí). Technologií tohoto druhu bude popsáno více, aby si čtenář vytvořil širší pohled na existující komplexní řešení. V sekci 3.3.1 bude stručně představena technologie Apache Nutch sloužící pro automatizované procházení webových stránek, dále sekce 3.3.2 obsahuje popis programu HTTrack pro archivaci webových stránek. V sekci 3.3.3 bude rozebrána knihovna Scrapy pro automatizované procházení webu a v poslední sekci 3.3.4 knihovna Lemmiwinks, která umožňuje archivaci webových stránek.

3.3.1 Apache Nutch

Apache Nutch[5] je open source software pro automatizované procházení webů [25]. Profesionální uživatelé jsou schopni s pomocí tohoto nástroje vytvořit plnohodnotný webový vyhledávač, kterým je například Google. Apache Nutch nabízí vlastní webový vyhledávač, který může zvýšit ohodnocení konkrétní aplikace ve výsledcích vyhledávání a přizpůsobit vyhledávání aplikace daným požadavkům. Je rovněž rozšiřitelný, dobře škáluje a umožňuje parsování, indexaci, vytváření vlastního webového vyhledávače, přizpůsobení vyhledávání požadavkům a velkou robustnost. Dovoluje pracovat se *ScoringFilter* mechanismem, který umožňuje vlastní implementaci k ohodnocování proměnných (jako proměnná může sloužit celá webová stránka). *ScoringFilter* je Java třída, která je využívána k vytváření rozšíření pro Apache Nutch.

Apache Nutch může fungovat na jednom systému stejně tak jako v distribuovaném prostředí, kterým může být například Apache Hadoop [25]. Celé řešení je napsáno v programovacím jazyku Java. Při vytváření indexů můžeme využít Apache Nutch například

[5] http://nutch.apache.org/

pro hledání nefunkčních odkazů a k vytváření kopií navštívených stránek pro budoucí vyhledávání. Hledání odkazů na webových stránkách probíhá automatizovaně. Systém nabízí bohaté možnosti integrace s jinými Apache nástroji, jako například s Apache Solr, což je vyhledávací platforma postavená na Apache Lucene, která může být použita pro vyhledávání jakéhokoli typu dat, i webových stránek. V této kombinaci Apache Solr ukládá všechny indexy webových stránek, které jsou získány pomocí Apache Nutch. Poté vyhledáváme webové stránky na základě indexů získaných z Apache Solr, který v tomto případě slouží jako NoSQL databáze.

3.3.2 HTTrack

HTTrack[6] je open source nástroj pro offline manipulaci s webovými stránkami [24]. Umožňuje stránku stáhnout do lokálního adresáře, kde vytvoří rekurzivní adresářovou strukturu, získá všechny soubory HTML, obrázky a další soubory, ze kterých se skládá daná stránka. Rovněž upraví všechny relativní odkazy původní stránky tak, aby odkazovaly na uložené soubory. Poté stačí stránku zobrazit v prohlížeči a uživatel se může pohybovat na stránce stejně, jakoby byla online. HTTrack je rovněž schopen aktualizovat stránku, která je již stažená nebo pokračovat v přerušovaném stahování. Je plně konfigurovatelný a obsahuje možnosti pro filtrování stránek.

HTTrack se dá používat jak v prostředí příkazové řádky, tak s grafickým uživatelským rozhraním [24]. Verze s grafickým rozhraním se jmenují WinHTTrack nebo WebHTTrack. Interně HTTrack používá automatizovaný prohledávač webových stránek s respektováním souboru robots.txt, ačkoli lze tuto funkcionalitu zakázat. Dokáže následovat odkazy generované jednoduchým JavaScript kódem nebo Applety, popřípadě Flash. Nedokáže už ovšem následovat kompletní odkazy generované například JavaScript funkcemi, výrazy nebo obrázkovou mapou.

3.3.3 Scrapy

Scrapy[7] je robustní webová Python knihovna pro dolování dat z různých zdrojů [4]. Z vysokoúrovňového pohledu Scrapy exceluje obzvláště při dvou případech užití:

- Když běžný uživatel webu požaduje stáhnout určitá data ze stránky, kterou zrovna prohlíží a libovolně data formátovat. Například je uložit ve formátu JSON nebo CSV či je uložit do databáze za účelem offline prohlížení nebo provedení dalších výpočtů.
- Pokud si uživatel přeje kombinovat data z různých zdrojů a extrahovat je.

S knihovnou Scrapy jsme schopni při jedné konfiguraci provést úlohy, na které bychom s jinými nástroji nebo knihovnami potřebovali mnoho tříd, rozšíření a konfigurací [4]. Z pohledu programátora stojí za zmínku *event-based* architektura. Ta umožňuje vytvářet kaskády operací, které můžou čistit, formátovat, obohacovat nebo data ukládat například do databáze, přičemž nezaznamenáme žádnou degradaci výkonu.

Technicky řečeno, touto architekturou se Scrapy dokáže zbavit přílišné latence, kterou generuje síť, zpracování dat nebo databáze, zatímco pracuje s tisíci otevřenými spojeními [4]. Jako extrémní příklad můžeme zvolit extrahování například nadpisů z webové stránky, která má sumarizující stránku se stovkou nadpisů v každém odkazu. Scrapy ve výchozí konfiguraci provádí 16 dotazů na webový server paralelně. Pokud by jeden dotaz čekal na dokončení

[6] https://www.httrack.com/
[7] https://scrapy.org/

sekundu, stahoval by 16 stránek za sekundu, čili by generoval 1600 nadpisů za sekundu. Scénář může pokračovat tak, že po stažení a extrahování nadpisu jej chceme uložit do cloudového úložiště, které má velmi špatnou odezvu, téměř 3 sekundy. Za předpokladu, že má být udržena rychlost stahování 16 stránek za sekundu, musíme generovat 4800 paralelních zápisů do databáze ($1600 * 3$). Pro klasickou více vláknovou aplikaci by to znamenalo alokovat a obsluhovat paralelně 4800 vláken, což by mohlo být pro operační systém zničující, nehledě na to, že by to bylo velmi nevýkonné. Ve světě aplikací Scrapy je ovšem 4800 dotazů do databáze za sekundu naprosto běžnou praxí, nehledě na nároky na paměť, které mají složitost O(n), kde n je počet nadpisů. Oproti tomu vícevláknová architektura alokuje ke každému vytvořenému vláknu signifikantní množství paměti.

Ve zkratce, pomalé nebo nepředvídatelné webové servery, databáze nebo jiné API třetích stran nebudou mít devastující efekt na výkon Scrapy programu, jelikož provádí paralelně mnoho dotazů a všechny interní operace řídí z jednoho vlákna [4]. To přináší další výhodu, pokud hodláme vedle Scrapy programu vykonávat další programy na stejném systému. Dále není zapotřebí žádná synchronizace v kódu, tudíž jde o značné zjednodušení celé aplikace oproti vícevláknovým aplikacím.

Od první verze Scrapy uběhla již dekáda, komunitou je dobře otestovaný a použitý v mnoha projektech pro dolování dat z webů. Mezi jeho další přednosti patří:

- práce s poškozenými HTML soubory, Scrapy se dokáže vypořádat se syntaktickými chybami a soubor zpracovat,
- nativní podpora Beautiful Soup, XPath a CSS selektorů,
- Scrapy má rozsáhlou komunitu uživatelů,
- dobře organizovaný a udržovaný kód komunitou,
- stále stoupající počet nových vlastností se soustředěním na kvalitu.

3.3.4 Lemmiwinks

Lemmiwinks je programový rámec poskytující funkcionality získávání dat z webů a jejich archivaci [22]. Velkým přínosem tohoto rámce je možnost zpracovat také webové stránky s dynamickým obsahem. Lemmiwinks archivuje celé webové stránky a to do formátu MAFF (Mozilla Archive Format), jehož předností je schopnost archivovat více oteřených záložek do jednoho výsledného dokumentu. Mezi slabé stránky patří chybějící podpora CAPTCHA mechanismu a navštěvování privátního obsahu, který je dosažitelný pouze po provedení autentizace.

Architektura Lemmiwinks je modulární, umožňuje tudíž přidávat libovolně nové moduly, případně měnit implementaci těch stávajících [22]. Jelikož je architektura modulární, má generický design, který definuje a používá rozhraní pro řešení částečných problémů. Jako programovací jazyk je použit Python. Určité části rámce jsou paralelní kvůli zvýšení výkonu a lepšímu využití zdrojů. Tento paralelismus není implementován pomocí vláken, ale po vzoru Scrapy 3.3.3 byla využita asynchronní architektura, kterou zajišťuje standardní knihovna asyncio (Scrapy ovšem využívá asynchronní knihovnu Twisted, která je méně výkonná a aplikačně náročnější než asyncio).

Asyncio je aktuálně v Pythonu nejlepší volba, jak naimplementovat asynchronní systém, obzvláště síťové aplikace, které musí odpovídat velkému množství klientů [3]. Jak již bylo řečeno v předchozí sekci u knihovny Scrapy, paralelní výpočty použitím vícero vláken jsou

náročné, obzvláště při řešení *race condition* u mnoha vláken. Kód napsaný pomocí asyncio je mnohem více odolný proti chybám – pouhým pohledem do kódu lze identifikovat ty části kódu, které jsou pod kontrolou programátora a ty, které vykonává asyncio smyčka a u kterých je dovoleno přerušení a provádění jiných událostí. Program je logicky rozdělen do úloh a rutin (úloha se může skládat z jedné nebo vícero rutin), které jsou řízeny manažerem úloh. Takový přístup reaguje na přerušení způsobené voláním operačního systému, převážně při I/O (vstup/výstup) operacích a přepne kontext provádění na jinou část kódu. Tím je dosaženo asynchronního provádění. V otázce vláken má Python obrovský problém – globální zámek interpretu (GIL[8]). Tento zámek dovoluje provádění pouze jednoho vlákna v čase, čili alokací vícero vláken při výpočtech, které nevyžadují I/O, se kód viditelně nezrychlí.

Samotný Lemmiwinks[9] je open source a obsahuje testovací aplikace *pharty* a *pharty2*, které slouží jako nápověda pro vlastní použití rámce. Aplikace *pharty* očekává jako vstupní parametr URL adresu a název výsledného archivu. *pharty2* je volaná z webového grafického uživatelského rozhraní a funguje obdobně.

[8] https://wiki.python.org/moin/GlobalInterpreterLock
[9] https://github.com/nesfit/Lemmiwinks

Kapitola 4

Návrh řešení

Jak již bylo řečeno, tato diplomová práce se zabývá sběrem, analýzou a následnou archivací dat z internetových stránek. V této kapitole budou rozebrány myšlenky návrhu, jakým směrem by se měla platforma ubírat jako taková a jaké by měla mít vlastnosti, strukturu, způsob nasazení, či testování.

Nejprve bude v sekci 4.1 obhájeno rozhodnutí pro výběr daných technologií. Sekce obsahuje výčet, popis a výběr vhodného programovacího jazyka, zhodnocení vhodných knihoven nebo rámců vhodných pro tuto práci a obhájení volby databáze se všemi klady i zápory. V další sekci 4.2 bude řeč o manipulaci s celou platformou, čili jakým způsobem se očekává, že by měl uživatel s platformou zacházet. Sekce dále obsahuje popis nejběžnějších aplikačních programových rozhraní a výběr toho nejvhodnějšího pro tento případ. Sekce 4.3 popisuje setkání s vyšetřovateli Policie České republiky ve Skalském Dvoře a z toho plynoucí opatření a závěr pro diplomovou práci. Poslední sekce 4.4 hovoří o celkové architektuře platformy, o všech částech, ze kterých se skládá, o způsobu nasazení do produkce a nakonec o návrhu testování dílčích částí.

4.1 Výběr technologií

V této sekci se nachází popis nejběžnějších technologií, které lze pro tuto práci využít, jejich klady a zápory, které jsou v tomto případě podstatné a nakonec výběr samotné vhodné technologie. Nejprve bude objasněna volba programovacího jazyka v širokém spektru jazyků, které se nám nabízejí, dále volba knihoven, které již byly zmíněny v předchozí kapitole a nakonec databáze s jednoduchým návrhem databázové struktury.

Programovací jazyk

Výběr vhodného programovacího jazyka hraje velmi důležitou roli když přijde na manipulaci s webovými stránkami, proto této oblasti bude věnován větší rozsah. Obtížnost sběru dat z internetu mimo jiné souvisí právě s jazykem, který je pro práci použit a s knihovnami, které daný jazyk nabízí. Nejdůležitější kritéria pro výběr vhodného jazyka jsou:

- flexibilita,
- schopnost práce s databázemi,
- efektivita sběru dat,
- úroveň obtížnosti programování,

- škálovatelnost a modularita,
- udržovatelnost kódu.

Často diskutovaným tématem je výsledný výkon programovacích jazyků. Mnoho začínajících programátorů se mylně domnívá, že co se týče aplikací, tak je nejdůležitější výkon, kterým daný jazyk oplývá. Reálně ovšem v mnoha typech aplikací hraje procesorový výkon pouze vedlejší roli. Hlavním argumentem pro vysoký výkon webových aplikací je rychlost provádění I/O (vstup/výstup) operací. Skutečným úzkým hrdlem je zde komunikace přes internet (rychlost přenosu po síti se totiž nemůže rovnat procesorovým rychlostem) – tuto nevýhodu ovšem dokáží překonat aplikace, které podporují asynchronní způsob provádění.

Prvním jazykem ze čtyř, který je zahrnut do výběru je C/C++. Jazyk nabízí nejvyšší výkon ze všech zkoumaných jazyků, je ovšem velmi náročné jej použít pro práci s webovými stránkami. Proto se nedoporučuje [7] využívat jazyky podobné C/C++ pro vytváření webových sběračů dat, pokud tedy cílem není vyloženě extrakce dat a ne jejich sběr, v té oblasti by se vysoký výkon vyplatil. Kdyby se využila knihovna libcurl pro stahování webových stránek, zbývala by pouze oblast analýzy HTML dokumentu a ta samotná je netriviální. C/C++ není dobrou volbou pro jakýkoli projekt týkající se webových stránek, jelikož dynamické jazyky dokáží tuto práci stejně dobře a s mnohem menším úsilím programátora.

Dalším jazykem je PHP. Ačkoli jde o dynamický jazyk, jde o nejméně preferovanou volbu. Dokonce pokud by bylo cílem extrahovat grafiku, videa nebo například fotografie z různých webových zdrojů, byla by lepší volba použít nástroj cURL – ten dokáže přenášet soubory protokoly HTTP nebo FTP. Velkou slabinou PHP je vícevláknové nebo asynchronní provádění či plánování úloh, což je pro případ této práce nepřijatelné.

NodeJS je další jazyk vhodný pro stahování webových stránek, převážně těch s dynamickým obsahem. Rovněž podporuje distribuovaný sběr webových stránek. NodeJS používá JavaScript, ve kterém interně běží smyčka pro události, což znamená neblokující provádění I/O operací, ze kterých aplikace těží maximální výkon. Každý NodeJS proces běží pouze na jednom procesorovém jádře, čili je vhodné spouštět takových procesů více. Tento jazyk oplývá rovněž velkým množstvím knihoven jak vestavěných, tak knihoven třetích stran. NodeJS se ovšem nedoporučuje pro dlouhý běh rozsáhlých aplikací [7].

Posledním jazykem, o kterém bude zmínka je dynamický jazyk Python. Co se týče sběru dat z internetu, jde o nejpopulárnější jazyk. S trochou nadsázky by se dalo říct, že jde o kompletní produkt, jelikož zvládá většinu operací souvisejících se zpracováním a extrakcí dat. Hlavním důvodem, proč je Python v této oblasti tak populární je přítomnost knihoven (viz. kapitola 3) jako Scrapy nebo Beautiful Soup, což programátorovi velmi usnadní práci. Nevýhoda je například oproti jazyku R ve vizualizaci dat, která nemusí být v některých případech vyhovující. Další nevýhodou je jednovláknové provádění, které již bylo v předchozí kapitole popsáno. Obecně jde ale díky své jednoduchosti a přítomnosti mnoha existujících nástrojů o jednu z nejlepších voleb jak pro sběr webových dat, tak pro jejich extrakci.

Programovacím jazykem pro všechny dílčí části platformy byl zvolen Python, díky jeho jednoduchosti a hlavně přítomnosti mnoha knihoven a rámců pro práci s webovými daty. Je ovšem nutno podotknout, že nejvhodnější alternativou by byl nejspíše jazyk Go, který spojuje výhody C/C++ a Pythonu, a který přináší zcela nový způsob vícevláknového provádění tak, aby byl procesor využit na maximum. Nevýhodou je ovšem jeho zatím malá rozšířenost a nedostatek kvalitních knihoven.

Knihovny

K programovacímu jazyku Python, který byl vybrán pro implementaci platformy je potřeba vybrat vhodné knihovny tak, aby co nejvíce ulehčily práci v těchto oblastech:

- automatizovaný sběr webových stránek,
- zpracování stažených dat,
- analýza webových dat,
- archivace stažených stránek,
- práce s databázemi.

V kapitole 3 byly představeny existující řešení pro automatizovaný sběr webových dat a pro jejich archivaci. Nyní bude řečeno, které a proč byly vybrány pro implementaci platformy. Prvním představeným nástrojem byl Apache Nutch. Jeho nepřekonatelnou výhodou je škálování a použití ve velkých aplikacích, což ovšem není případ této práce. Pro samotný sběr dat a hledání informací například pomocí regulárních výrazů není zapotřebí takového velkého systému a bylo by to spíše ku škodě. Další nevýhodou je Java jakožto programovací jazyk Apache Nutch, což je pro účely této práce zcela nepřijatelné, protože by bylo velmi nevhodné a nepraktické kombinovat v jedné platformě vícero jazyků.

Dalším uvedeným nástrojem pro sběr webových dat (viz. kapitola 3) je Scrapy, což je Python knihovna, poskytující automatizovaný sběr webových stránek. Tato knihovna byla vybrána ze zcela logických důvodů – jde o nejrozšířenější knihovnu ve světě Pythonu, co se týče sběru dat a jejich manipulace. Scrapy obsahuje například podporu pro práci s databázemi, konfiguraci agentů, kteří stahují webové stránky a jeho velkou předností je asynchronní provádění, čímž překonává nevýhody jak pomalých sítí, tak jednovláknového provádění v Pythonu. Za Scrapy stojí velká komunita i mnoho velkých společností, které knihovnu používají pro automatizovaný sběr dat, tudíž je zde velká podpora a mnoho dohledatelných informací.

Pro archivaci webových stránek byl představen HTTrack. Ten je pro zálohování webů pro tuto platformu nevhodný z toho důvodu, že dokáže pracovat pouze synchronně. Za předpokladu, že by bylo potřeba v jednom čase zálohovat mnoho webových stránek, výkon platformy by byl degradován právě na tomto místě. Další nevýhodou je skutečnost, že HTTrack nedokáže archivovat stránky s odkazy generovanými například JavaScript funkcemi.

Posledním zmíněným nástrojem pro archivaci webových dat byl Lemmiwinks, který poskytuje funkcionalitu získávání dat z webů a jejich archivaci. Hlavním důvodem proč byl tento nástroj vybrán pro archivaci webů je ten, že je to programový rámec, který poskytuje pouze základní funkcionalitu a je na programátorovi, aby se zamyslel nad způsobem, jakým chce, aby Lemmiwinks pracoval. Další nespornou výhodou je schopnost archivace stránek s dynamickým obsahem a výkon samotného rámce. Jelikož Lemmiwinks pracuje asynchronně, výborně doplňuje knihovnu Scrapy, která pracuje na podobné bázi a tím se teoreticky zachová vysoký výkon platformy.

Databáze

Databázová část aplikací je často tou kritickou, která může být úzkým hrdlem mnoha aplikací. Přestože tato platforma není jednou z takových aplikací, při špatném návrhu nebo nepochopení databázových datových struktur by se jí mohla lehce stát. Proto je důležité

zvolit vhodné databáze a hlavně správné datové struktury a korektně je v aplikacích používat. V platformě se nachází dvě databáze:

- SQL databáze PostgreSQL,
- NoSQL databáze Redis.

První z nich, PostgreSQL, často přezdívaná jako Postgres, je objektově-relační databází s možností použít objektového návrhu s důrazem na rozšiřitelnost. Relační databáze je jedním ze způsobů, jak organizovat data do kolekce relací. Stejně jako například MariaDB je open-source a využívá vlastního relačního schématu. Výhodou Postgres je relativně jednoduchá možnost konfigurace a velká schopnost škálování. Používá se u malých aplikací běžících na jednom systému s nízkou zátěží až po velké světově známé aplikace s mnoha konkurentními databázovými operacemi a s obrovskými datovými úložišti. Postgres je multiplatformní databáze napsaná v jazyku C. V diplomové práci Postgres slouží k perzistentnímu ukládání archivovaných informací.

Druhou databází je Redis (Remote Dictionary Server), což je open-source NoSQL databáze sponzorována Redis Labs. V NoSQL databázích se jedná o nerelační model dat, které jsou charakteristické tím, že nepoužívají dotazovací jazyk SQL, čímž nedosahují takové flexibility jako relační databáze. Takové databáze pracují bez striktního schématu, což umožňuje přidávat nové informace bez nutnosti měnit datové struktury. Konkrétně Redis je Key-Value databáze napsaná v jazyku C, která je založena na hash funkci, primárně při přístupech do databáze přes klíč. Velkou výhodou oproti relačním databázím je větší výkon a flexibilita ukládání dat. Redis pracuje jako in-memory databáze, čili záznamy ukládá do paměti, volitelně pak na disk. V diplomové práci Redis slouží právě jako in-memory fronta.

4.2 Ovládání platformy

V této sekci budou uvedeny možnosti, jak by se dalo s platformou manipulovat a výběr té nejvhodnější. Nejprve je potřeba si objasnit, jaké hlavní možnosti jsou k dispozici:

- ovládání skrze příkazovou řádku,
- manipulace s platformou přes grafické uživatelské rozhraní.

Běžný uživatel by mohl navrhnout, že by postačovalo pouze grafické uživatelské rozhraní, ale praxe říká opak – vždy musí existovat hlavní přístup k platformě jako celku i k jejím dílčím částem skrze příkazovou řádku, aby se zajistil rychlý a efektivní přístup, vývoj a ladění. Grafické uživatelské rozhraní je potom už jen vrstva navíc, která má běžným uživatelům ulehčit a zrychlit manipulaci s platformou. Stejný přístup byl akceptován pro tuto práci, kdy bude vystavěno aplikační programové rozhraní pro komunikaci jak s celou platformou, tak i s jejími jednotlivými částmi. Grafické uživatelské rozhraní už může být volitelně vystavěno nad tímto aplikačním programovým rozhraním.

Aplikační programové rozhraní

V nejjednodušším pojetí je aplikační programové rozhraní (API) softwarový zprostředkovatel, který dovoluje, aby spolu mohly dvě aplikace komunikovat. Existují hlavní dva typy API – SOAP a REST. SOAP (Simple Object Access Protocol) je sám o sobě protokol, který je komplexnější než REST (definuje mnoho standardů jako například bezpečnost nebo jak

se jednotlivé zprávy zasílají). Oproti tomu REST (Representational State Transfer) je API, které je ve většině případů užívané webovými službami a je založeno na URI (Uniform Resource Identifier) a HTTP protokolu. REST API může být jednoduché vytvořit a škálovat, může se z něj ovšem stát rovněž API masivní a velmi komplikované. V následující tabulce 4.1 je popsáno základní rozlišení:

Vlastnost	**SOAP**	**REST**
Styl	Protokol.	Architektura.
Funkce	Funkčně řízený (přenáší strukturované informace).	Řízený daty (přístup a zdroj k datům).
Formát dat	Používá pouze XML.	Povoluje mnoho formátů dat (text, XML, HTML, JSON)
Bezpečnost	Podporuje WS-Security a SSL.	Podporuje SSL a HTTPS.
Náročnost na zdroje	Vyžaduje více zdrojů.	Jednoduchý a nenáročný na zdroje.
Cache date	Nepodporuje cache.	Může využívat cache dat.

Tabulka 4.1: Srovnání SOAP a REST API [21].

Pro platformu bylo vybráno REST API převážně z důvodu jednoduchosti (podpora JSON formátu). Nad každou důležitou službou, ze které se bude platforma skládat bude vystavěno REST API pro přímou komunikaci s jednotlivými komponenty.

4.3 Konzultace s vyšetřovateli PČR

Tato diplomová práce spadá pod projekt TARZAN. Jde o integrovanou platformu pro zpracování digitálních dat z bezpečnostních incidentů [23]. Projekt je zaměřen na analýzu a detekci nových forem kybernetické kriminality především v prostředí mobilních a komunikačních aplikací a v prostředí internetu věcí. Cílem projektu je výzkum nových technologií a metod pro efektivní vyšetřování bezpečnostních incidentů. Výsledky budou demonstrovány na případech z praxe, například na detekci provozu P2P sítí [20], bezpečnostní analýze mobilních zařízení či řešení incidentů v oblasti Bitcoinů.

V rámci projektu TARZAN se uskutečnila konference ve Skalském Dvoře s vyšetřovateli policie České republiky, což pomohlo upřesnit požadavky na platformu jako takovou. Policie České republiky se bohužel potýká s nedostatkem financí a kvalifikovaných odborníků. Z toho vyplývají následující komplikace:

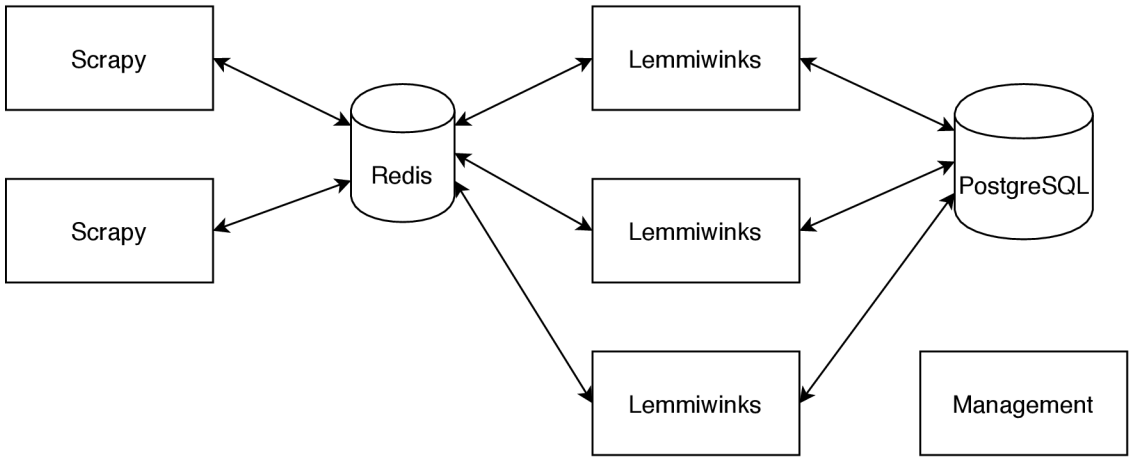
- požadují, aby platforma byla spustitelná na systému Windows,
- školení nových uživatelů platformy by mělo být minimalistické a co nejjednodušší,
- celá platforma by měla být spustitelná pouze z jediného spustitelného souboru a tudíž by mělo jít architektonicky o monolitickou aplikaci.

Na základě těchto požadavků bylo usneseno rozhodnutí, že v rámci diplomové práce bude vytvořena druhá verze platformy, speciálně pro účely Policie České republiky. Jak bude popsáno v sekci 4.4, navrhovaná architektura platformy se diametrálně liší od představy vyšetřovatelů. Přináší to ovšem řadu problémů, které znamenají, že nemusí být „odlehčená“

verze stejně kvalitní a nemusí mít všechny vlastnosti a výhody plnohodnotné platformy. V následující sekci budou rozebrány obě verze co do architektury a rozdíly mezi nimi.

4.4 Architektura platformy

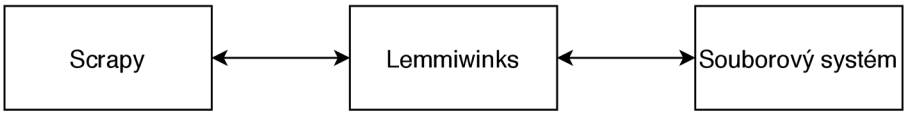
V této sekci bude podrobněji vysvětlena a vyobrazena architektura platformy obou verzí – plnohodnotné verze podporující systémy Unix a druhé verze s omezenou funkcionalitou pro systém Windows.



Obrázek 4.1: Architektura plnohodnotné platformy pro systémy Unix.

Jak lze vidět na obrázku 4.1, plnohodnotná verze se skládá z pěti částí:

- Scrapy je služba pro sběr a analýzu webových dat. Umožňuje dynamickou konfiguraci pavouků a tzv. potrubí, přes která tečou stažená data a v každém tomto bodě lze s daty provádět odlišné operace. Procesů Scrapy může být v rámci platformy více, podle aktuálního zadání.
- Redis je distribuovaná fronta pracující pouze v rámci paměti nezávisle od ostatních komponent platformy. Zde stačí pouze jeden databázový datový typ list, který slouží jako fronta. Procesy Scrapy vkládají do fronty data a procesy Lemmiwinks z fronty tato data odebírají. Do fronty se ukládají celé HTML dokumenty spolu s metadaty.
- Lemmiwinks je webový archivátor, ukládající výstupní archivy ve formátu MAFF. Proces z Redisu načítá celé HTML dokumenty a ty rekurzivně archivuje. V rámci platformy může být instancí procesu Lemmiwinks více, jelikož fronta v Redisu se může plnit mnohem rychleji, nežli Lemmiwinks může data konzumovat.
- PostgreSQL je perzistentní databáze, která přiřazuje k jednotlivým doménám archivovaných webových stránek cesty k jejich uloženému MAFF archivu. Schéma databáze je obohaceno o metadata související s archivovanou webovou stránkou.
- Management slouží jako aplikační programové rozhraní celé platformy. Skrze tohle REST API je možné manipulovat s jednotlivými částmi platformy – spouštět/ukončovat procesy Scrapy nebo Lemmiwinks a získávat lokaci archivu dle zadaného dotazu. V rámci obrázku 4.1 je Management část spojena s každou z ostatních částí.



Obrázek 4.2: Architektura platformy podporující systém Windows.

Na obrázku 4.2 lze vidět architekturu aplikace vyhovující požadavkům vyšetřovatelů. Požadují vynechání obou databází, aby celá platforma byla co nejjednodušší a obsahovala jen ty části, které jsou nezbytně nutné. Jako další nadbytečnou částí bylo označeno REST API pro manipulaci s platformou.

Docker

Co se týče verze platformy pro systémy Unix, každá z jednotlivých komponent bude fungovat v kontejneru Docker. Uživatel bude mít možnost výběru, zdali si platformu zprovozní sám nebo použije předpřipravené Docker soubory. V případě verze pro vyšetřovatele se Docker používat nebude, jelikož je to další systém, který je potřeba obsluhovat a je označen za nadbytečný.

Iniciativa ohledně Dockeru usnadnila uchopení paradigmatu kontejnerizace. Docker je open-source kontejnerizační systém, který automatizuje zabalení, distribuci a nasazení jakékoli softwarové aplikace. Tyto aplikace jsou potom prezentovány jako lehké a přenositelné kontejnery.

Docker kontejner [19] si lze představit jako softwarový balík obsahující vše potřebné pro nezávislý běh daného softwaru. Na jednom systému může běžet mnoho kontejnerů. Ty jsou na sobě nezávislé a jsou vůči sobě i hostujícímu systému zcela izolovány. Jinými slovy, takový kontejner obsahuje softwarové komponenty spolu se všemi potřebnými závislostmi (binární soubory, knihovny, konfigurační soubory, skripty a další potřebné soubory). Docker kontejnery plynule běží na x64 Linux jádru. Jsou zde ovšem mechanismy a postupy k tomu, aby Docker kontejner běžel i na systémech jako Windows nebo Mac. Kontejner má svůj vlastní procesorový prostor a síťové rozhraní.

Testování

Platforma bude testována automatizovaně pomocí Python knihovny *ptest*. Nejprve budou otestovány všechny dílčí části (Scrapy a Lemmiwinks), poté platforma jako celek. Scrapy podporuje *Spiders Contracts*, čímž Scrapy nabízí integrovanou cestu, jak testovat pavouky definicí vstupů, výstupů a jednoduchých pravidel. Lemmiwinks bude testován pouze pomocí *ptest*.

ptest je jednoduchý testovací rámec v jazyku Python, který využívá značení tříd a testovacích funkcí pomocí dekorátorů. Testy lze spouštět z příkazové řádky a zpráva o vykonaných testech je jasná a přehledná. Pomocí testů bude snaha o co největší pokrytí kódu testy (Code Coverage). To je míra používaná k vyjádření toho, kolik zdrojového kódu programu je vykonáno, když proběhnou testy. Program s nejvyšším pokrytím má 100 %, což implikuje, že všechen zdrojový kód je otestován. To je ideální stav, ke kterému se chceme přiblížit.

Kapitola 5

Implementace

V této kapitole bude podrobněji popsána implementace obou verzí integrované platformy pro zpracování digitálních dat – plnohodnotné verze podporující systémy Unix a druhé verze s omezenou funkcionalitou pro systémy Windows.

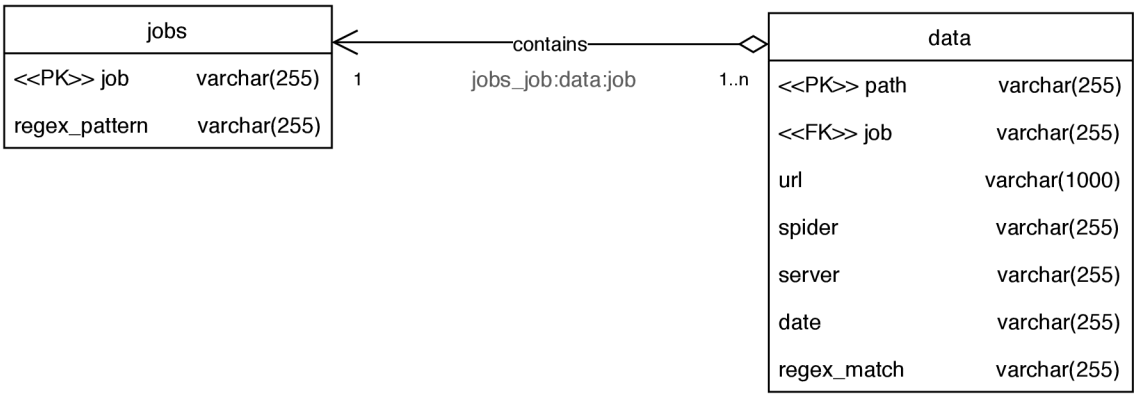
Nejprve budou v sekci 5.1 rozebrány detaily o použitých databázích spolu s databázovými strukturami. Jak již bylo řečeno v kapitole 4, řeč bude konkrétně o SQL databázi PostgreSQL a NoSQL databázi Redis. V další sekci 5.2 bude prezentován sběrač dat Scrapy se všemi jeho náležitostmi. Konkrétní popis nástroje Scrapy, obecná architektura sběrače dat a konkrétní architektura Scrapy, jeho použití v platformě a následně základní možnosti konfigurace. Následuje sekce 5.3, kde bude představen nástroj Lemmiwinks jako takový s jeho architekturou i implementačními detaily. Důležitou částí této sekce jsou nutné modifikace, které byly zapotřebí udělat na straně nástroje Lemmiwinks, aby pracoval způsobem vhodným pro platformu. Poté bude v sekci 5.4 objasněn návrh a implementace aplikačního programového rozhraní, jeho modulární systém s možnostmi rozšíření a způsobem použití v rámci platformy. Následně bude v sekci 5.5 popsáno nasazení platformy v prostředí Docker s použitím nástroje Docker Compose. Poslední sekce 5.6 bude hovořit o verzi platformy pro vyšetřovatele České republiky spolu s její architekturou a rozdíly oproti plnohodnotné verzi.

5.1 Databázová část

V této sekci budou objasněny implementační detaily ohledně použitých databází. Důraz bude kladen na datové typy, struktury a na operace nad nimi, stejně jako na konkrétní způsob jejich použití v diplomové práci. Sekce 5.1.1 bude hovořit o SQL databázi PostgreSQL spolu se způsobem využití v platformě. V sekci 5.1.2 bude popsána NoSQL databáze Redis a stejně jako v předchozím případě i způsob užití v platformě.

5.1.1 PostgreSQL

Databáze PostgreSQL je v diplomové práci použita jako uložiště metadat. S těmito metadaty je poté operováno pomocí SŘBD (systém řízení báze dat), což si lze představit jako kolekci programů, které umožňují konstrukci databází a aplikací, které tyto databáze využívají. Databázové schéma v diplomové práci obsahuje dvě tabulky, viz. obrázek 5.1.



Obrázek 5.1: ERD schéma v PostgreSQL databázi.

Tabulka *jobs* slouží k uložení pouze základních informací o úloze Scrapy. Proto obsahuje jen dva sloupce:

- primární klíč *job* je název úlohy, pod kterým byl spuštěn proces Scrapy,
- *regex_pattern* je regulární výraz, který Scrapy využil při analýze stažených dat z webových stránek.

Každý záznam v tabulce *jobs* má k sobě asociován jeden nebo více záznamů z tabulky *data*:

- *path* jakožto primární klíč slouží k uložení cesty v souborovém systému, kam byl uložen MAFF archiv,
- *job* slouží jako cizí klíč do databáze *jobs*,
- *url* je adresa konkrétní webové stránky, která byla archivována,
- *spider* je název pavouka, který byl použit pro prohledávání webu,
- *server* je název systému, na kterém byl Scrapy spuštěn,
- *date* je datum, kdy byla stažená stránka zanalyzována a uložena do databáze Redis,
- *regex_match* obsahuje středníkem oddělené řetězce vyhovující regulárnímu výrazu.

Tato metadata jsou sbírána pro ulehčení tvorby budoucího grafického uživatelského rozhraní, které není součástí této diplomové práce. Uživatel, který vytvoří úlohu na sběr a archivaci dat z nějaké webové stránky bude očekávat, že existuje i nějaký způsob pro získání výsledků, to je celý účel ukládání metadat. Ideálně by mělo grafické uživatelské rozhraní fungovat jako webová stránka, kde uživatel vyplní všechny důležité informace pro zadání úlohy popsané v sekci 5.4 a poté si bude moci přehlednou formou zobrazit tato metadata spolu s cestou k archivované webové stránce, kterou by si měl být schopen pomocí webového prohlížeče rovněž zobrazit v dalším okně webového prohlížeče. Takové uživatelské rozhraní je bráno v potaz jako možné rozšíření platformy.

5.1.2 Redis

Jak již bylo řečeno, Redis je open-source NoSQL databáze, která pracuje s nerelačním modelem dat. Ty jsou charakteristické tím, že nepoužívají jazyk SQL, naopak pracují bez striktního schématu. Redis je Key-Value databáze, která využívá při přístupech do databáze hash funkci.

Redis podporuje pět datových typů – řetězec, seznam, množina, hash a seřazená množina. Popsány budou jen řetězec a seznam, jelikož tyto dva typy platforma využívá pro ukládání HTML stránky spolu s metadaty. Řetězec je v databázi Redis nejzákladnějším datovým typem. Takový řetězec je binárně bezpečný, což je neocenitelná vlastnost při práci s multimediálními daty. Tato vlastnost zajišťuje, že řetězec může obsahovat jakýkoli druh dat, například JPEG obrázky jebo serializované objekty vyšších programovacích jazyků. Řetězec může obsahovat data o velikosti až 512 MB. S řetězcem mohou být prováděny například následující operace:

- použití jakožto atomický čítač (*INCR*, *DECR*, *INCRBY*),
- konkatenace dvou řetězců (*APPEND*),
- libovolný přístup k různým částem řetězce (*GETRANGE*, *SETRANGE*),
- komprimace velkého objemu dat (*GETBIT*, *SETBIT*).

Dalším datovým typem je seznam. Seznam se skládá z jednotlivých řetězců seřazených pořadím, podle kterého jsou do seznamu vkládány. Vkládat položky je možné pomocí operací *LPUSH* nebo *RPUSH*, což vloží prvek na začátek (doleva) respektive na konec seznamu (doprava), viz. příklad:

```
RPUSH my_list abc # my_list obsahuje "abc"
RPUSH my_list def # my_list obsahuje "abc","def"
LPUSH my_list ghi # my_list obsahuje "ghi","abc","def"
```

Maximální délka seznamu je $2^{32}-1$ prvků (4294967295). Hlavní výhodou seznamu z pohledu časové složitosti je vkládání a mazání prvků ze začátku nebo konce seznamu v konstantním čase $O(1)$. Přístup k prvkům je rychlý na začátku nebo konci seznamu, ovšem velmi pomalý v tom případě, kdy chceme přistupovat například k prvku, který leží uprostřed seznamu, jelikož je potřeba provést $O(N)$ operací. Se seznamem mohou být prováděny například následující operace:

- vkládat prvky na začátek seznamu pomocí operace *LPUSH* a poté získat jen část prvků operací *LRANGE*,
- *LPUSH* může být použit v kombinaci s operací *LTRIM*, kdy délka seznamu nikdy nepřekročí požadovanou délku. Seznam bude vždy obsahovat pouze posledních N prvků,
- seznam může být využit pro posílání zpráv mezi jednotlivými procesy běžících na pozadí.

V diplomové práci je využit datový typ seznam, který plní funkci fronty mezi dvěma procesy běžícími na pozadí.

Obrázek 5.2: Fronta po výměnu dat mezi procesy.

Jak naznačuje obrázek 5.2, Redis zde slouží jako fronta mezi procesy Scrapy a Lemmiwinks, které si skrze tuto frontu zasílají data. Fronta je implementována pomocí datového typu seznam, přičemž prvky seznamu jsou řetězce. Je zde využito vlastnosti binární bezpečnosti řetězců, jelikož prvek tohoto seznamu je ve skutečnosti objekt ve formátu JSON. Komunikace s frontou je zajištěna operacemi *RPUSH* (procesy Scrapy vkládají prvky na konec fronty) a *LPOP* (procesy Lemmiwinks vyčítají prvky ze začátku fronty). Na obrázku 5.3 je zobrazen vzorek dat, která jsou v Redisu uložena.

```
1)
"{
\"url\": [\"http://www.fit.vutbr.cz/\"],
\"job\": [\"basicplatform\"],
\"spider\": [\"basic\"],
\"server\": [\"8363f38fa125\"],
\"date\": [\"2019-04-05 17:15:50.100025\"],
\"headers\": [\"{b'Date': [b'Fri, 05 Apr 2019 17:15:49 GMT'], b'Server': [b'Apache'],
b'Content-Location': [b'index.php.en'], b'Vary': [b'negotiate,accept-language'],
b'Tcn': [b'choice'], b'Pragma': [b'no-cache'], b'X-Frame-Options': [b'deny'],
b'Content-Type': [b'text/html; charset=iso-8859-2'], b'Content-Language': [b'en']}\"],
\"htmlcontent\": [\"<!DOCTYPE HTML PUBLIC \\\"-//W3C//DTD HTML
4.01//EN\\\">.....\\n</body>\\n\"],
\"regex_match\": [\"Brno University of Technology\"],
\"regex_pattern\": [\"[a-zA-Z]{4}\s.{10}\sof\s.{4}n.{5}\"]
}"

2)
"{
\"url\": [\"http://www.fit.vutbr.cz/events/bissit/\"],
\"job\": [\"basicplatform\"],
\"spider\": [\"basic\"],
\"server\": [\"8363f38fa125\"],
\"date\": [\"2019-04-05 17:27:35.650944\"],
\"headers\": [\"{b'Date': [b'Fri, 05 Apr 2019 17:27:35 GMT'], b'Server': [b'Apache'],
b'Set-Cookie': [b'logoShown=1; path=/events/bissit/; domain=www.fit.vutbr.cz'],
b'Content-Type': [b'text/html; charset=utf-8']}\"],
\"htmlcontent\": [\"<!DOCTYPE HTML PUBLIC \\\"-//W3C//DTD HTML
4.01//EN\\\">.....\\n</body>\\n\"],
\"regex_match\": [\"best-rated IT faculty\"],
\"regex_pattern\": [\"[a-zA-Z]{4}-.{5}\sIT\s[a-zA-Z]{7}\"]
}"
```

Obrázek 5.3: Vzorek uložených dat v databázi Redis.

Vzorek obsahuje pouze dva prvky v seznamu pojmenovaného *queue*. Tento krátký výčet byl získán operací *LRANGE queue 0 2*. Příkaz začne na nultém indexu a vybere do výčtu pouze dva následující prvky. Obsahem prvku je datová struktura JSON, která obsahuje escape sekvence, aby se předešlo nechtěné interpretaci dat. JSON obsahuje metadata spolu s celou HTML stránkou, která bude následně archivována. Záměrně byly vybrány dva odlišné prvky, aby si čtenář mohl povšimnout rozdílných regulárních výrazů se specifickými řetězci, které vyhovují daným regulárním výrazům ze dvou různých webových stránek. Tato metadata jsou podrobněji popsána v sekci 5.1.1.

5.2 Scrapy pro sběr dat

Jak již bylo řečeno v sekci 3.3.3, Scrapy je knihovna v programovacím jazyku Python a slouží k dolování webových dat z různých zdrojů na internetu. Hlavní výhoda knihovny je výrazné ulehčení práce – programátor nemusí psát základní logiku sběrače dat, pouze definuje pavouky a ty nakonfiguruje. Jak bylo řečeno v sekci 3.3.3, Scrapy je postavený na *event-based* architektuře, překonává tedy známé nedostatky s latencí spojenou se sítěmi.

Tato sekce se bude věnovat převážně architektuře (viz sekce 5.2.1) knihovny Scrapy a modulů, které na jejím základě v rámci platformy byly vytvořeny. Jako modul je zde myšlen pavouk pro sběr dat a potrubí pro další práci s těmito daty. Bude kladen důraz na ty klíčové vlastnosti Scrapy, díky nimž byl vybrán jakožto klíčová komponenta platformy. Poté v sekci 5.2.2 bude konkrétně vysvětlen princip konfigurace pavouků a přehled konfiguračních parametrů, které lze v rámci definovaných pavouků použít.

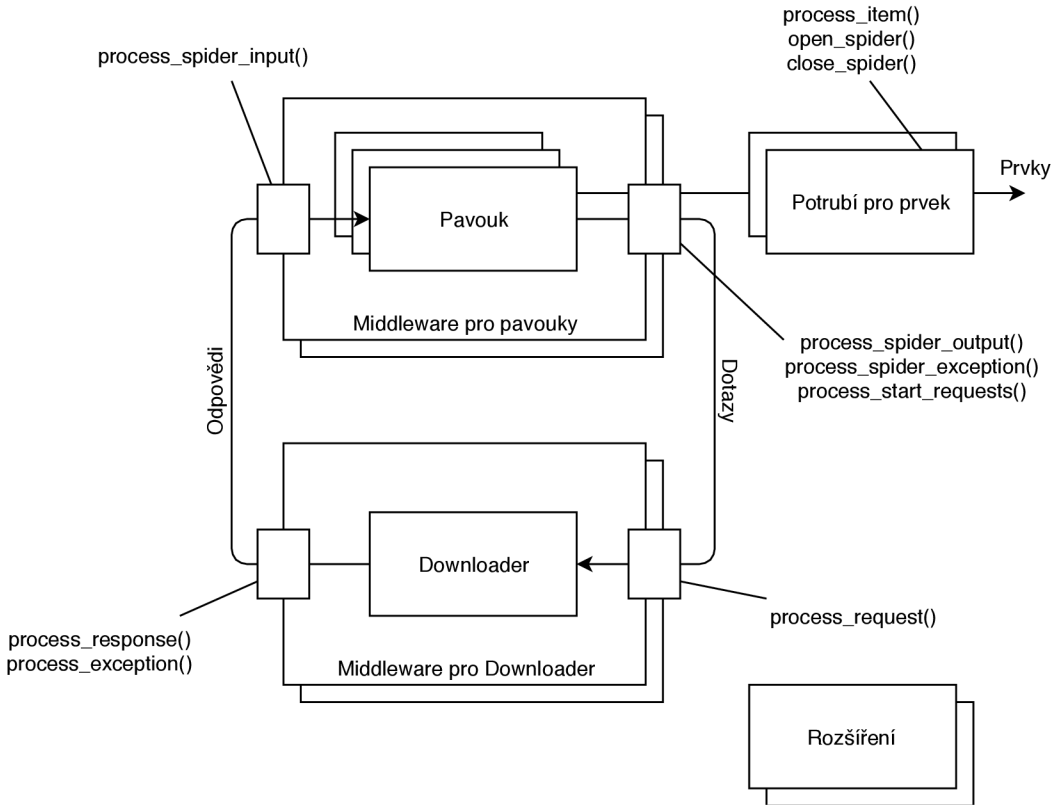
5.2.1 Architektura

Architektura vlastní implementace se opírá o obrázek 5.4, který znázorňuje architekturu nástroje Scrapy. Lze si povšimnout tří podobných typů objektů, nad kterými Scrapy operuje – dotazy, odpovědi a prvky. Uživatelští pavouci jsou jádrem celé architektury, jelikož v nich leží logika procházení webu a manipulace s prvky. Další podstatnou částí jsou potrubí, skrze které putují prvky a jsou s nimi prováděny další operace.

Pavouci vytváří HTTP dotazy, zpracovávají HTTP odpovědi a poté generují jednotlivé prvky spolu s dalšími dotazy na další webové stránky. Prvek v tomto kontextu může znamenat:

- extrahovaný element z právě stažené webové stránky (například nadpisy úrovně h1 nebo všechny elementy vyhovující definovanému CSS selektoru),
- systémové a konfigurační informace (například název serveru, na kterém je proces spuštěn nebo použitý regulární výraz),
- libovolná uživatelská proměnná vytvořená v rámci běhu pavouka.

V sekci 5.1.1 byla představena metadata, která se v konečném důsledku zpropagují až do PostgreSQL databáze. V kontextu Scrapy a jeho pavouků jsou tato metadata právě prvky, které pavouk vytvoří a v potrubí jsou nad nimi prováděny definované operace. Dalším klíčovým pojmem jsou potrubí. To si lze představit jako posloupnost funkcí, kdy každá z funkcí v potrubí jednoduše přijímá prvek jakožto parametr a může nad prvkem provádět libovolné operace.



Obrázek 5.4: Scrapy architektura.

Adresářová struktura projektu Scrapy vypadá následovně:

- basicplatform/
  - pipelines/
    - ___init___.py
    - redis.py
  - spiders/
    - ___init___.py
    - basic.py
    - login.py
  - ___init___.py
  - items.py
  - middlewares.py
  - settings.py
- scrapy.cfg

Z obrázku 5.4 lze vidět jisté podobnosti s adresářovou strukturou. Projekt je logicky rozčleněn na potrubí, pavouky, prvky, middleware a konfiguraci.

Pavouci

Projekt obsahuje definici dvou pavouků – *basic* a *login*. Oba dva fungují v podstatě totožně až na jeden významný rozdíl, kterým je podpora autentizace v případě druhého pavouka. Nejprve bude popsán základ fungování obou pavouků, jelikož mají většinu logiky společnou.

Při startu pavouka jsou nejprve inicializovány důležité proměnné z konfiguračního souboru. Těmi jsou:

- název pavouka,
- seznam povolených domén – všechny ostatní domény jsou implicitně na černé listině a budou filtrovány,
- počáteční URL adresa, kde Scrapy začne pracovat,
- regulární výraz, který je z výkonnostních důvodů zkompilován a který bude využit při analýze webové stránky,
- definice pravidel, která se definují pomocí tříd *Rule* a *LinkExtractor*.

Na základě definovaných pravidel potom pavouk navštěvuje (nebo filtruje) jednotlivé stránky. Dalším kritickým bodem je metoda *parse_item*, ve které se odehrává celá logika chování pavouka. Pomocí třídy *ItemLoader* je vytvořen objekt, do kterého lze přidávat jednotlivé prvky (tento objekt naplněný prvky bude dále putovat skrze potrubí). Následující prvky jsou získány buď z konfigurace nebo ze systémových proměnných – *url*, *job*, *spider*, *server*, *date*, *regex_pattern*. Prvky získané na základě stažené webové stránky jsou potom *headers*, *htmlcontent* a *regex_match*. *regex_match* je seznam řetězců oddělených středníkem, které vyhovují danému regulárnímu výrazu.

Pavouk *login* obsahuje navíc mechanismus autentizace. Ten probíhá následovně:

- Třída s definicí pavouka obsahuje metodu *start_requests*, která je invokována jakožto první, ještě před vygenerováním prvního HTTP dotazu. Ta vytváří specifický HTTP dotaz na *LOGIN_URL* definovanou v konfiguračním souboru. Scrapy takto obdrží HTML stránku s přihlašovacím formulářem.
- Následně se invokuje metoda *login*, ve které se nachází samotná logika přihlašování. Scrapy automaticky zanalyzuje přihlašovací stránku, vyhledá formulář a vyplní jeho údaje opět na základě konfiguračních parametrů. Zde je ovšem komplikovanější situace kvůli přihlašovacímu jménu a heslu, jelikož citlivé údaje se nesmí nacházet v konfiguračních souborech. Z toho důvodu jsou tyto dva parametry předány na příkazové řádce při spuštění procesu.
- Po provedení autentizačních kroků začne pavouk fungovat jako v předchozím případě, ovšem už s tím rozdílem, že má naplněnou datovou strukturu s platnou cookie a tak se dostane i na ty webové stránky, které jsou bez autentizace nepřístupné.

Potrubí

Potrubí jsou nedílnou součástí fungování Scrapy, jelikož dokážou jednoduše modifikovat prvky, které pavouk vytvořil. V projektu se využívá pouze jednoho potrubí definovaného v souboru *pipelines/redis.py*. Třída *RedisCache* se zde stará o uložení prvku do databáze Redis.

Nachází se zde tři podstatné metody. První z nich je *from_crawler*, která se invokuje automaticky jako první, jelikož je to třídní metoda a Scrapy ji automaticky invokuje ještě před inicializací samotné instance třídy. V této metodě se inicializují třídní proměnné, které budou v dalším kroku sloužit k práci s databází – doménové jméno (popřípadě IP adresa) instance databáze Redis a její port.

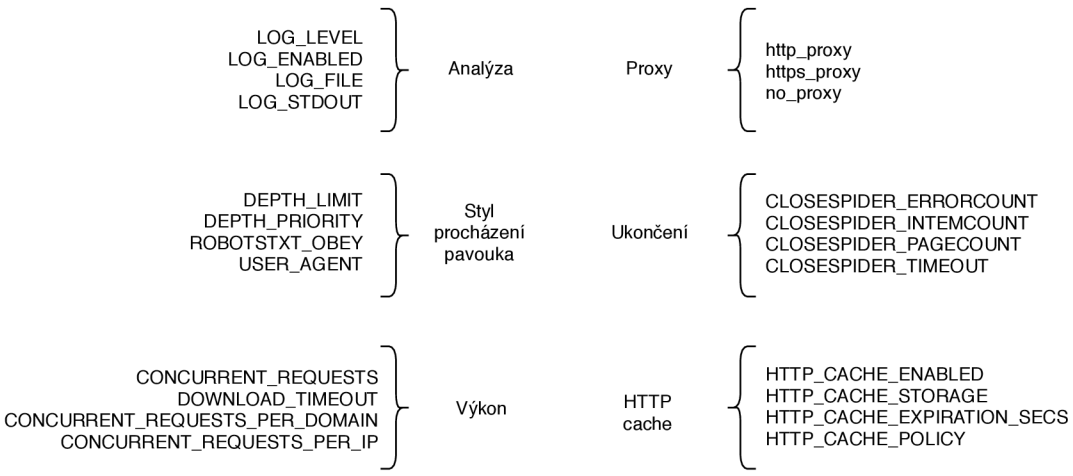
Následně se invokuje metoda *init_redis___pool*, ve které se vytvoří fyzické spojení s databází. Spojení se nevytvoří pouze jedno. Naopak se vytvoří soubor spojení, která pracují paralelně pro zvýšení výkonu, jelikož síťové latence se můžou stát jednoduše úzkým hrdlem celé platformy.

Poslední invokovanou metodou je *process_item*, která slouží k samotnému uložení prvku do databáze. Jak již bylo řečeno v úvodu této sekce, funkce v potrubí přijme prvek jakožto parametr, což je případ této funkce. Proměnná je typu *Item* a obsahuje seznam všech jednotlivých prvků, které pavouk vytvořil a uložil do této datové struktury. Následně je zkontrolován atribut *regex_match* a jestliže je neprázdný, provede se uložení do databáze Redis. Jestliže pavouk za pomoci regulárního výrazu nenašel žádný řetězec, funkce končí bez další akce.

5.2.2 Možnosti konfigurace

Scrapy přichází s velkou funkcionalitou a nástroji, které jsou dostupné skrze konfiguraci. Ta hraje v softwaru obecně velmi důležitou roli, jelikož je to možnost, jak jednoduše měnit chování aplikace a ladit ji. Scrapy má více úrovní konfigurace, ale nejužitečnější je modifikovat konfiguraci specifickou pro celý projekt (*<project_name>/settings.py*). Tahle konfigurace je platná pouze pro aktivní projekt a neovlivní jiné projekty v rámci fungování Scrapy.

Na obrázku 5.5 je znázorněn diagram konfiguračních sekcí. Parametrů v každé sekci je mnohem více, než je zde zobrazeno[1]. V diagramu jsou naopak vyobrazeny ty parametry, které jsou v komunitě Scrapy nejpoužívanější a které rovněž využívá paltforma.



Obrázek 5.5: Diagram konfiguračních kategorií Scrapy.

V kategorii analýza lze nakonfigurovat ladicí informace skrze logovací soubory, statistiky nebo možnost připojit se na běžícího pavouka pomocí nástroje telnet.

[1] https://docs.scrapy.org/en/latest/topics/settings.html

Kategorie výkon umožňuje uzpůsobit výkonnostní charakteristiky různým situacím. Můžeme nakonfigurovat například počet paralelních dotazů, počet paralelních dotazů na jednu doménu, IP adresu nebo počet paralelně generovaných prvků ze stránek vstupujících do potrubí. Tyto vlastnosti chrání vzdálený server poskytující webovou službu před nadbytečnou zátěží, která v mnoha případech není zapotřebí.

Kategorie stylu procházení pavouka umožňuje volbu například toho, která stránka se získá jako první nebo definovat maximální hloubku zanoření. Pokud by například uživatel vyžadoval procházení webů algoritmem BFS (Breadth-first search), zvolí hloubku zanoření na kladnou hodnotu (nula značí žádný limit) a jako frontu pro plánovač vybere FIFO (First in, first out).

Nastavení z kategorie proxy je povolena automaticky a využívá proměnných prostředí systému podle konvence Unix (*http_proxy*, *https_proxy* a *no_proxy*).

Kategorie ukončení umožňuje zastavit činnost pavouka po splnění specifických podmínek definovaných právě v této kategorii. Uživatel může nakonfigurovat například ukončení pavouka po nějakém časovém úseku, po získání určitého počtu prvků, chyb při zpracování nebo po obdržení daného počtu HTTP odpovědí z webu.

Poslední kategorie HTTP cache poskytuje konfiguraci pro cachování HTTP dotazů a odpovědí. Můžeme například nastavit chování podle pravidel uvedených v RFC2616 použitím *scrapy.contrib.httpcache.RFC2616Policy* jakožto hodnotu pro parametr *HTTP_CACHE_POLICY* [5].

5.3 Lemmiwinks pro archivaci stránek

Jak již bylo řečeno v sekci 3.3.4, Lemmiwinks je programový rámec umožňující archivaci webových stránek do formátu MAFF (Mozilla Archive Format). Dokáže archivovat webové stránky i s dynamickým JavaScript obsahem. Velkou výhodou je modulární architektura, což umožňuje (jak již bylo řečeno v architektuře Scrapy) libovolně přidávat nové chování nebo měnit již existující. Platforma této modulární architektury plně využívá a uzpůsobila si chování celého rámce ke svým specifickým požadavkům.

V sekci 5.3.1 bude detailně popsána architektura části platformy využívající Lemmiwinks pro archivaci webových stránek. Bude rozebrána jak architektura rámce Lemmiwinks, tak celé části platformy odpovědná za archivace webových stránek. Jelikož je Lemmiwinks programový rámec, je úlohou programátora, aby takový rámec využil způsobem, který bude vhodný pro specifickou úlohu. Toto uzpůsobení rámce pro požadavky platformy bude rozebráno v sekci 5.3.2.

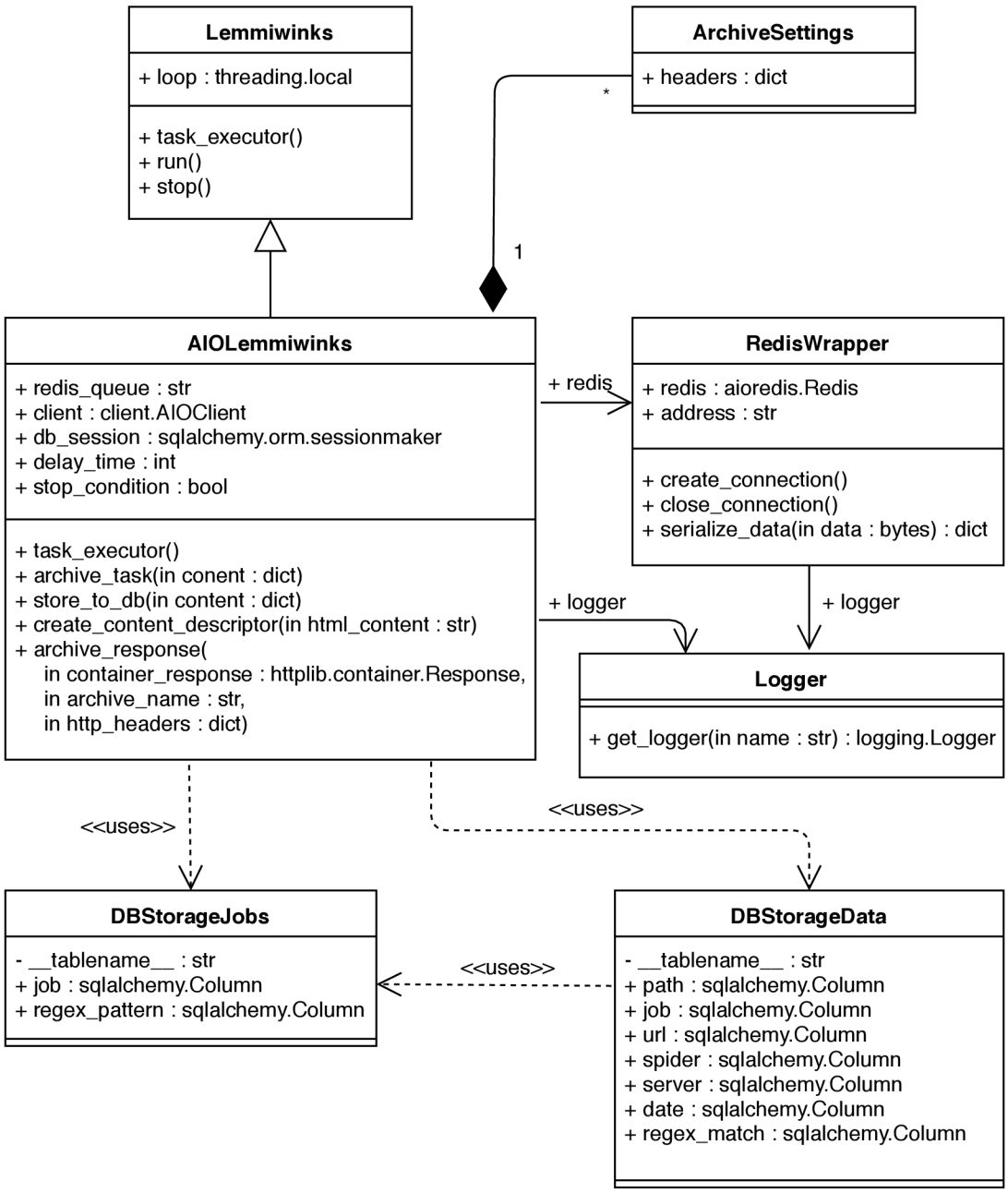
5.3.1 Architektura

Na diagramu 5.6 je vyobrazena základní architektura části platformy, která leží mezi databází Redis, samotným rámcem Lemmiwinks a databází PostgreSQL. Základem je třída *AIOLemmiwinks*, která dědí ze třídy *Lemmiwinks*, který je již součástí rámce. Klíčová je metoda *task_executor*, kterou je nutné definovat, jelikož touto metodou je spuštěna smyčka událostí, která zajišťuje asynchronní způsob provádění.

Smyčka událostí spustí běh archivátoru, který pomocí objektu třídy *RedisWrapper* periodicky vyčítá data z Redis databáze. Jestliže v Redisu nejsou přítomna žádná data, zdvojnásobí se prodleva, se kterou je proveden další dotaz (původní hodnota pro prodlevu je nastavena na jednu sekundu). S každým neúspěšným čtením je tato prodleva zdvojnásobena

až na konečnou hodnotu 64 sekund. Tento mechanismus zamezí zbytečnému dotazování do databáze, pokud proces Scrapy negeneruje žádná data.

V kladném scénáři jsou data z Redisu úspěšně vyčtena. Tato data jsou objektem ve formátu JSON, který obsahuje pole se shodnými názvy, jakou jsou v SQL tabulkách *jobs* a *data* diagramu 5.1 (v objektu se nachází navíc podstatný atribut, čímž je celá HTML stránka). Pomocí metody *serialize_data* jsou data přetypována na *dict*, což je reprezentace objektu v jazyce Python. Další účel této metody je odstranit escape sekvenci, protože v tomhle místě chceme data interpretovat.



Obrázek 5.6: Diagram tříd Lemmiwinks.

Následujícím krokem je provedení samotné archivace, kterou má v režii rámec Lemmiwinks. Archivace je invokována v metodě *archive_task*, kdy je z objektu použita HTML stránka, která je archivována. Metody *create_content_descriptor* a *archive_response* jsou pomocné funkce sloužící k přípravě dat před samotnou archivací. První z nich má za úkol převést HTML data do vnitřní objektové reprezentace, kterou Lemmiwinks používá pro manilulaci s HTML dokumenty. Druhá z nich má potom za úkol volat jednotlivé metody zajišťující archivaci jednotlivých komponent stránky.

Po provedení archivace se vykoná poslední úloha, kterou je uložení metadat do PostgreSQL databáze. Schéma již bylo vyobrazeno v diagramu 5.1. K tomu slouží třídy *DBStorageJobs* a *DBStorageData* pro totožně pojmenované databázové tabulky. K práci s databází slouží ORM (objektově relační mapování), což je moderní a bezpečný způsob, jak manipulovat s objekty databáze z prostředí kódu aplikace. Tuto funkcionalitu zde zajišťuje SQLAlchemy, což je komplexní balík nástrojů pro databáze pro jazyk Python. Tento nástroj (v názvosloví Pythonu se jedná o knihovnu) umožňuje pracovat s databází jako s objekty a tím pádem se dovoluje vyvarovat přímému použití SQL dotazů, které by se vykonávaly z uživatelského kódu. Další výhodou je podpora všech známých databází, takže vyměnit PostgreSQL za libovolnou jinou databázi není problém – část platformy zajišťující archivaci nepozná žádný rozdíl, což je účelem modulární architektury.

5.3.2 Nutné modifikace

Lemmiwinks jako takový je programový rámec sloužící k archivaci webových stránek, tudíž je potřeba jej doplnit o své vlastní moduly a přizpůsobit si jej podle svých požadavků. Lemmiwinks obsahuje v základu dva základní případy užití, pro každý případ je přítomen jeden soubor (skript):

- Prvním a jednodušším případem je *Lemmiwinks/pharty/pharty.py*. Jedná se o konzolovou aplikaci, která slouží k archivaci webové stránky specifikovanou URL adresou. Podporuje dva režimy archivace – s dynamickým JavaScript obsahem a bez něj (serverem renderované stránky). Výkon archivace v těchto dvou případech se taky významně liší. Zatímco stránky s dynamickým obsahem se mohou archivovat dlouhou dobu (v průměru 4-6 sekund, záleží ovšem na velikosti a složitosti stránky), ty bez dynamického obsahu se archivují podstatně kratší dobu (v průměru 2-4 sekundy, opět záleží na konkrétní stránce a výkonnosti stroje, na kterém skript běží).

- Druhým a složitějším případem je *Lemmiwinks/pharty2/pharty2.py*, jehož účelem je pracovat v rámci aplikace MultiFIST[2]. Tento skript vyžaduje při spuštění parametr ve formátu JSON, ve kterém je specifikována URL adresa a další informace nutné pro archivaci. Skript vytváří při archivaci další záložky – první je snapshot webové stránky a druhá je tabulka s metainformacemi.

Pro použití v této práci ovšem nemají tyto aplikace význam, neboť řeší zcela odlišné problémy. Platforma vyžaduje v rámci archivování webových stránek kontinuální běh aplikace, kdy bude archivátor periodicky vyčítat data z Redisu a pokud nějaká data obdrží, archivuje je. Vyvstává ovšem další problém, který se týče integrity dat. Webová stránka se totiž v čase mění a pokud by platforma fungovala na principu, kdy by Scrapy spustil skript na archivaci *pharty.py* a ten by ji archivoval, originální webová stránka by se mohla na straně webového serveru nějakým způsobem změnit a tudíž by byla porušena integrita dat. Toto riziko je

[2] https://github.com/nesfit/MultiFIST

eliminováno na úrovni architektury platformy tím, že je vytvořen skript *pharty3.py*, který přímo využívá moduly Lemmiwinks pro archivaci.

Je řeč o třídě *AIOLemmiwinks* z diagramu 5.6. Využívá ke své činnosti dalších tříd, které musely být implementovány pro splnění dalších požadavků, například logování nebo manipulace s databází Redis a PostgreSQL. Archivátor tedy funguje následovně:

- V prvním kroku je využita třída *RedisWrapper*, která vyčítá data z fronty databáze Redis.
- Poté je provedena archivace webové stránky ovšem s tím rozdílem, že rámec Lemmiwinks archivuje HTML data obdržená z Redisu namísto archivace pouze podle URL stránky. Tím se zachová integrita dat.
- Cesta k archivované webové stránce spolu s dalšími metadaty z Redisu jsou uloženy do PostgreSQL využitím příslušných tříd, viz. 5.6.

Další změnou, která musela být provedena, je zrušení optimalizace na úrovni cache při archivaci stránky. Původně archivace fungovala tím způsobem, že se všechny zdroje ze stránky rekurzivně stahovaly a lokálně ukládaly pro případ, kdy stejný zdroj bude potřeba stáhnout opětovně. Takový způsob optimalizace dobře fungoval při archivaci jedné stránky během jednoho běhu aplikace, ne už při archivaci vícero stránek během jednoho běhu. Problém byl v mechanismu, kterým se ukládaly cesty k souborům, jelikož tyto cesty neměly unikátní bezkolizní názvy. Nebylo to ani potřeba vzhledem k tomu, že rámec původně sloužil k archivaci pouze jedné stránky. Při vytváření rámce se nepočítalo s tím případem užití, kdy bude rámec pracovat opakovaně nad podobnými úlohami (nad archivací stránek, kde jsou sdílené soubory, například obrázky).

Problém tedy nastal při archivaci stránky, která obsahovala například právě obrázek, který byl archivován u jedné z předchozích stránek. Do cache se v tomto případě zapisoval soubor se stejným názvem a vznikla kolize. Proto byla tato optimalizace zakázána, dokud nevyjde nová verze rámce Lemmiwinks, která bude tento problém řešit a bude počítat s tím, že při startu archivace může být cache již naplněna a tak první operací by mělo být vyhledání v cache namísto pokusu o stažení zdroje.

5.4 Aplikační programové rozhraní

Je důležité mít možnost nějakým způsobem s platformou komunikovat. Tím je myšleno schopnost zadávat úlohy, vyčítat běžící úlohy nebo je mazat. Proto je aplikační programové rozhraní přidruženo k části Scrapy a funguje tím stylem, kdy při zavolání konkrétního koncového bodu se správnými parametry spustí proces Scrapy, který začne vykonávat danou úlohu.

Toto rozhraní je postaveno na rámci Flask. Jedná se o mikro webový rámec vytvořený v Pythonu. Označován jako mikro rámec je z toho důvodu, že pro svůj běh nevyžaduje spoustu různých nástrojů a knihoven, narozdíl například od rámce Django, který v konečném důsledku plní v podstatě podobnou úlohu jako Flask. Nemá žádnou databázovou abstraktní vrstvu, validaci formulářů nebo další předpřipravené komponenty. Flask ovšem podporuje velké množství rozšíření, které plní libovolnou funkci. Jedná se o plně modulární systém – Flask v základu obsahuje pouze nejdůležitější komponenty, lze ovšem jednoduše přidat další, které budou rozšiřovat funkcionalitu.

Aplikační programové rozhraní je rovněž vytvořeno jako modulární systém, protože případů užití platformy může být mnoho a není účelem práce je všechny pokrýt. Naopak

cílem je vytvořit takový systém, který bude jednoduchý, intuitivní pro uživatele a rozšiřitelný. Následující přehled obsahuje výčet aktuálně podporovaných operací. Přehled obsahuje aplikační programové rozhraní dvou tříd – *Job* pro manipulaci nad konkrétní úlohou a *Jobs* pro manipulaci nad všemi úlohami:

- Koncový bod: /jobs/<id>
  - Metoda: GET
    - Popis: získání informace o procesu s identifikátorem <id>. Návratová hodnota může obsahovat tři různé typy odpovědi – proces nenalezen, proces aktuálně probíhá nebo proces skončil s daným návratovým kódem.
    - Návratová hodnota: ({"message": "process <job> not found"}, 404), ({"message": "process <job> is in progress"}, 200), ({"message": "process <job> ended with return code {ret_code}"}, 200).
  - Metoda: DELETE
    - Popis: zastavení běžícího procesu s identifikátorem <id> a jeho smazání z registru běžících procesů. Návratová hodnota může značit buď neznámý název procesu nebo úspěšné zrušení procesu.
    - Návratová hodnota: ({}, 404), ({}, 204).
- Koncový bod: /jobs
  - Metoda: GET
    - Popis: získání názvu všech aktuálně běžících úloh. Návratová hodnota obsahuje pole řetězců – identifikátorů úlohy.
    - Návratová hodnota: ({"jobNames": "array"}, 200).
  - Metoda: POST
    - Popis: Vytvoření nové úlohy. Na základě vstupní JSON datové struktury se provede nejprve nakonfigurování pavouka a knihovny Scrapy, následně se spustí proces s danou konfigurací. Ve vstupním JSONu je podstatný atribut ïd;, což značí identifikátor pavouka. Pomocí tohoto identifikátoru je dále možno získat stav běžícího procesu nebo proces úplně zastavit. Návratová hodnota může být oznámení, že proces s daným identifikátorem již běží, některý z JSON atributů chybí nebo oznámení o neznámém názvu pavouka.
    - Vstupní JSON: {"robotstxtObey": "boolean", "dynamicJavaScript": "boolean", "job": "string", "spider": "string", "regexPattern": "string", "logLevel": "string", "logFile": "string", "loginUrl": "string", "formName": "string", "formLogin": "string", "formPassword": "string", "username": "string", "password": "string", "allowedDomains": "array", "startUrls": "array"}
    - Návratová hodnota: ({"message": "process <job> is already running"}, 409), ({"message": "some field is missing"}, 400), ({"message": "process <job> unknown spider"}, 400), ({}, 200)
  - Metoda: DELETE
    - Popis: zastavení všech běžících procesů a jejich smazaní z registru běžících procesů. Návratová hodnota se očekává vždy jako úspěch.
    - Návratová hodnota:({}, 200).

Platforma aktuálně podporuje dva pavouky – *basic* a *login*. Při použití prvního z nich nemusí být v metodě POST přítomny atributy *loginUrl*, *formName*, *formLogin*, *formPassword*, *username* a *password*. Jsou to totiž atributy sloužící k autentizaci a ta probíhá pouze při použití pavouka *login*.

5.4.1 Architektura

Aplikační programové rozhraní je přidruženo do Scrapy repozitáře z toho důvodu, že spolu tyto dvě domény úzce souvisí. Když chceme invokovat nějakou úlohu v platformě, počátek je vždy na straně Scrapy, jelikož se musí nejprve spustit proces Scrapy pro sběr dat. Proto se nachází implementace REST API vedle implementace Scrapy. Celá struktura vypadá následovně:

- api/
  - models/
    - ___init___.py
    - job.py
  - resources/
    - ___init___.py
    - job.py
  - app.py
  - settings.py
  - utils.py

Jedná se o Flask aplikaci, jejímž nejdůležitějším souborem je *app.py*, kde se nachází definice koncových bodů a k nim přidružené třídy, které tyto koncové body implementují. Jedná se o koncový bod */jobs*, který implementuje třída *Jobs* a druhý koncový bod */jobs/<id>*, který implementuje třída *Job*.

Obě dvě třídy se nachází v adresáři *resources*, který je určen pro třídy implementující koncové body. Kromě definice obsluhy koncových bodů soubor *job.py* obsahuje rovněž pomocné funkce pro práci s konfigurací. Když přijde požadavek pro spuštění nového procesu, nejprve se z JSONu vyberou parametry a provede se konfigurační fáze. Tyto pomocné funkce vyžadují jako parametr cestu ke konfiguračnímu souboru v adresáři s projektem Scrapy, parametr označující levou stranu přiřazení a následně hodnotu, které se má rovnat levá strana. Tímto způsobem lze nastavit například mód sběru dat (podpora dynamického JavaScript obsahu), nastavení názvu úlohy, regulární výraz, seznam povolených domén, počáteční doménu a další konfigurační parametry v souboru *settings.py*.

Soubor *models/job.py* je volitelný, jelikož obsahuje definice modelu pro serializaci/deserializaci vstupní/výstupní struktury JSON. Při obdržení vstupní JSON struktury je jakožto první operace provedena deserializace do nativní Python datové struktury *dict*. Flask k tomu využívá svých vlastních mechanismů implementovaných v knihovně *flask_marshmallow*, pomocí kterých lze definovat svůj vlastní model na základě primitivních datových typů. Pokud tedy přijde při invokaci koncového bodu JSON, který neobsahuje všechny povinné atributy nebo jsou špatného typu, další zpracování nenastane a uživateli je ihned navrácet návratový kód s chybovou hláškou.

V souboru *settings.py* je uvedena základní konfigurace REST API. Je zde obsažena:

- cesta k repozitáři s projektem Scrapy,
- cesta k logovacímu souboru (původně nastavena na */tmp/scrapy_web_api.log*),
- úroveň logování do konzole (původně nastavena na *logging.DEBUG*),
- úroveň logování do souboru (původně nastavena na *logging.INFO*).

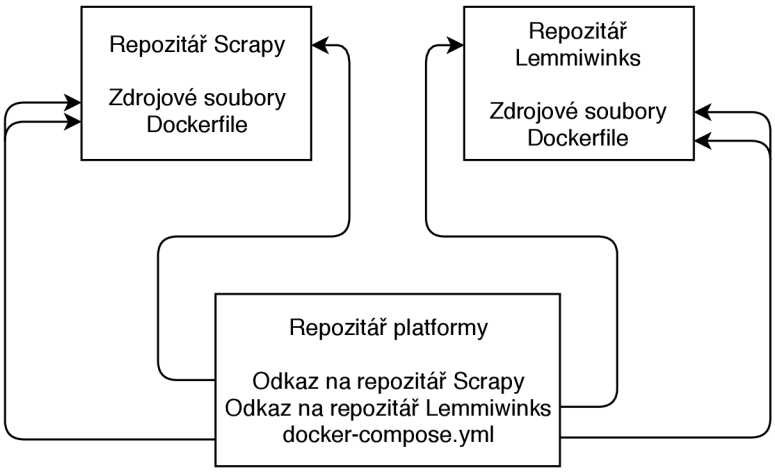
Posledním souborem obsaženým v adresáři je *utils.py*, který obsahuje definici logovací třídy a dekorátoru pro deserializaci, který byl představen v předešlém odstavci. Třída *Logger* definující logovací mechanismus pracuje následujícím způsobem:

- Statickou metodou *get_logger* se provede inicializace nové instance třídy *Logger*.
- Nově vzniklé instanci se nastaví identifikátor, který slouží pro informování uživatele, z jakého souboru vznikl daný řádek v logovacím souboru.
- Této instanci se nastaví chování pro logování do souboru a do konzole. Chováním je zde myšlen formát výstupu, který bude logger tisknout na konzoli nebo do souboru a logovací úroveň, která je definovaná v konfiguračním souboru *settings.py*.

Základ aplikačního rozhraní se tedy nachází v souborech *scrapit/api/app.py* a *scrapit/api/recources/job.py*. Jedná se o plně modulární systém, tudíž v případě libovolných rošíření jsou důležité pouze tyto dva soubory, do kterých lze snadno doplnit kód pro další koncové body a další HTTP metody, přičemž lze čerpat inspiraci z již existujícího kódu. Praktické příklady použití jak s aplikačním programovým rozhraním manipulovat (například pomocí nástroje *curl*) se nachází v souboru *README.md* přiloženého v repozitáři.

5.5 Docker

Docker jakožto technologie byla blíže popsána již v sekci 4.4. V této sekci bude podrobně rozebrán způsob, jakým platforma využívá funkcionality Dockeru. Na obrázku 5.7 je zobrazen systém užití Dockeru pomocí Docker a Docker Compose souborů.



Obrázek 5.7: Schéma Docker souborů.

Zásadním souborem, který se stará o management kontejnerů v platformě je *proof_platform/docker-compose.yml*. Docker Compose je nástroj pro definici a správu běhu multikontejnerových aplikací. Jakožto konfigurační soubor využívá YAML soubory pro definici vlastností jednotlivých služeb. Následně lze jednoduše s jedním příkazem vytvořit všechny služby v podobě kontejnerů. Soubor *docker-compose.yml* v platformě definuje následující:

- Definice kontejneru lemmit – obsahuje relativní cestu ke zdrojovým souborům, odkazy na jiné kontejnery, na kterých závisí a které bude pro svůj běh potřebovat (postgresdb a redisdb) a cestu k hostujícímu adresáři, kam bude ukládat výsledné archivované soubory.

- Definice kontejneru scrapit – stejně jako u lemmit obsahuje cestu ke zdrojovým souborům, odkaz na kontejner redisdb a definici portů (mapování portu hostujícího systému na port uvnitř kontejneru) kvůli korektnímu fungování REST API.

- Definice kontejneru postgresdb – obsahuje pouze Docker obraz, podle kterého Docker Compose zhotoví kontejner.

- Definice kontejneru redisdb – stejně jako postgresdb obsahuje pouze Docker obraz.

- Definice kontejneru pgadmin – kromě obrazu obsahuje odkaz na kontejner postgresdb, na kterém závisí spolu s proměnnými prostředí a definicí portu. Tento kontejner slouží pouze pro správu databáze PostgreSQL.

Docker Compose při vytváření lemmit a scrapit kontejnerů přejde do adresáře se zdrojovými soubory, vyhledá soubor Dockerfile, ve kterém je uveden recept ke zhotovení kontejneru na základě přiložených zdrojových souborů a podle instrukcí v něm obsažených zhotoví požadovaný kontejner.

Oproti souboru *docker-compose.yml*, *Dockerfile* obsahuje konkrétní recept k vytvoření Docker obrazu a z tohoto obrazu se následně vytváří konkrétní kontejner. Scrapy *Dockerfile* obsahuje následující recept:

- Docker skládá obrazy podle vrstev. Jako nižší vrstva, na kterou bude aplikována vrstva Scrapy je použita *python:3.6-slim*, což je odlehčený obraz obsahující Python verze 3.6.

- Příkaz *EXPOSE <port_number>* dovolí kontejneru poslouchat na uvedeném portu. Scrapy konkrétně bude poslouchat na portu 5000. Jak bylo řečeno výše u Docker Compose, v případě Scrapy se protuneluje vnější port (konkrétně 80) na vnitřní port v kontejneru 5000. Tím pádem pro okolní svět se bude REST API procesu Scrapy nacházet na portu 80, ačkoli aplikace uvnitř poslouchá na portu 5000.

- Poté se zkopíruje celý obsah adresáře, ve kterém se uživatel aktuálně nachází při volání příkazu *docker-compose build* do */root/scrapit/*.

- V dalším kroku se zkopíruje geckodriver do adresáře */usr/local/bin/*. geckodriver je webový procesor vyvíjený Mozilla Foundation, který platforma potřebuje pro propojení Selenium rámce s webovým prohlížečem Firefox. Díky tomuto propojení je platforma schopná zpracovávat webové stránky i s dynamickým JavaScript obsahem.

- Následně se změní pracovní adresář na */root/scrapit/* a poté se provede aktualizace systémových balíčků. V tomto kroku probíhá instalace některých balíčků, které jsou potřeba pro běh Scrapy nebo pro ulehčení práce a ladění přímo uvnitř kontejneru.

Konkrétně se instaluje gcc a vim, lze ale *Dockerfile* lehce rozšířit o další, pokud by uživatel platformy požadoval některé další nástroje.

- Nyní se vytvoří Python virtuální prostředí, do kterého se nainstalují všechny závislosti ze souboru *requirements.txt*.
- Posledním bodem v receptu Scrapy je *CMD*, definující příkaz, který se provede bezprostředně po vytvoření a spuštění kontejneru. V kontextu Scrapy je to aktivace virtuálního prostředí a spuštění REST API.

V případě souboru *Dockerfile* archivátoru Lemmiwinks je situace velmi obdobná, jsou proto uvedeny pouze rozdíly:

- Lemmiwinks nepotřebuje poslouchat na žádném portu, jelikož neinteraguje napřímo s vnějším světem. Oproti tomu je zde potřeba se nějakým způsobem spojit s hostujícím souborovým systémem. Bylo by totiž nepraktické, kdyby Lemmiwinks archivoval soubory, ale ty by byly přístupné pouze napřímo z kontejneru a při smazání kontejneru by se všechny archivované webové stránky smazaly. K tomuto propojení slouží příkaz *volumes* v souboru *docker-compose.yml*. To by ovšem nefungovalo, pokud by samotný kontejner nepovolil sdílení patřičného adresáře, podobně jako to bylo u konfigurace portů. K vystavení adresáře hostujícímu systému slouží příkaz *VOLUME <directory>*. V případě Lemmiwinks je to adresář */root/lemmit/archived/*.
- Další kroky jsou totožné jako v případě Scrapy až na poslední příkaz *CMD*, který v případě Lemmiwinks startuje z virtuálního prostředí skript *pharty3.py*.

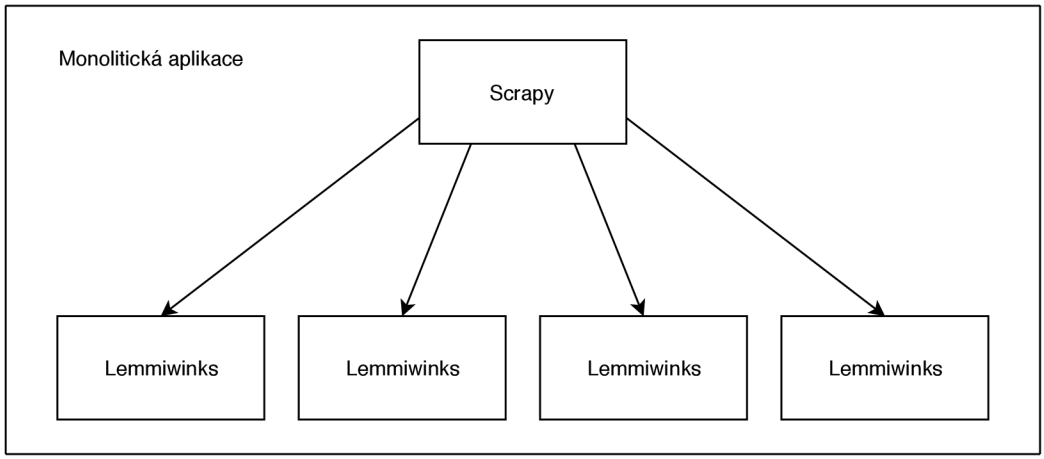
Nasazení platformy pomocí Docker Compose je velmi jednoduché. Stačí se přesunout do adresáře s platformou a pomocí příkazu *docker-compose build* se vytvoří všechny potřebné Docker obrazy, jak bylo vysvětleno výše. Následně se příkazem *docker-compose up -d* vytvoří všechny kontejnery a spustí se na pozadí. Pokud by se uživatel potřeboval dostat přímo dovnitř kontejneru například pro potřeby ladění, lze tak snadno učinit příkazem *docker exec -it <container_id> /bin/bash*. Pro zastavení všech kontejnerů potom slouží příkaz *docker-compose stop*.

5.6 Verze platformy pro účely vyšetřování

Policie České republiky shledává platformu pro své účely užitečnou, ovšem ne v podobě, jakou byla doposud prezentována. Vyšetřovatelé dali na platformu několik striktních omezení, což vedlo k logickému rozhodnutí zhotovit platformu ve dvou verzích. Architektura první, plnohodnotné verze, podporující systémy Unix již byla představena. V této sekci bude naopak představena verze platformy pro účely vyšetřování. Omezení jsou následující:

- platforma musí být spustitelná na systému Windows,
- důraz na minimalistické a jednoduché školení nových uživatelů platformy,
- jeden spustitelný soubor, tudíž pouze jedna komponenta.

Za požadavky stojí pro policii zcela validní důvody. Nepotřebují ke svým účelům tak velkou platformu, která by dokázala libovolně horizontálně škálovat. Obrázek 5.8 popisuje architekturu této verze.



Obrázek 5.8: Architektura platformy pro systémy Windows.

Tato verze platformy funguje diametrálně odlišně. Namísto modulárního systému se jedná o monolitickou aplikaci, kdy je přítomen jeden proces Scrapy, který nezasílá data o webové stránce k archivaci do databáze Redis. Namísto toho vytváří nové procesy, které obdrží URL adresu a tu webovou stránku archivují. Tato kritická část aplikace se odehrává v souboru *winit/scrapitlite/basicplatform/pipelines/proxy.py*. Pro správu procesů je využita Python knihovna *apscheduler*, která je nakonfigurována následovně:

- Jak je vidno na obrázku 4.2, maximální počet instancí procesů Lemmiwinks je v základní konfiguraci omezen na čtyři.
- *apscheduler* si ukládá všechny příchozí požadavky na archivaci do fronty a jakmile nějaký z procesů skončí, tehdy vytváří proces nový.

Všechny parametry pro archivaci jsou konfigurovatelné v souboru *proxy.py* a parametry pro sběr dat pomocí procesu Scrapy v souboru *winit/scrapitlite/basicplatform/settings.py*. Tato verze platformy kvůli přísným požadavkům na architekturu trpí nedostatkem zajištění konzistence dat, o které se hovoří v sekci 5.3.2.

Kapitola 6

Testování

V této kapitole bude popsáno testování, které bylo s platformou provedeno. Celá platforma jako taková je velmi rozsáhlý projekt a bez stejně rozsáhlého testování si nemůže být uživatel jist, že platforma funguje skutečně očekávaným a korektním způsobem. Platforma má následující části, které musí být pomocí testů ideálně plně pokryty:

- pavouci, potrubí a fungování Scrapy sběrače dat jako takového,
- práce s databází Redis,
- archivační rozhraní rámce Lemmiwinks,
- práce s databází PostgreSQL.

Následující sekce představí pokaždé jiný způsob testování, v konečném důsledku budou ovšem testy pokryty všechny aspekty platformy. Nejprve budou v sekci 6.1 popsány jednotkové testy, které jsou základem každého projektu a je skoro až povinností programátora je v průběhu tvorby aplikace tvořit. Bez správně napsaných jednotkových testů se projekt nedá považovat za relevantní. Následně bude v sekci 6.2 popsán manuální způsob testování, který si může vyzkoušet s platformou prakticky kdokoli. Bude se jednat o konfiguraci dílčích částí, sběr dat z libovolné stránky, její následná archivace a uložení metadat. Výsledná data dokážou správnost fungování platformy jako celku. V sekci 6.3 bude vykonáno komplexní výkonnostní testování všech částí platformy, aby si uživatel mohl udělat obrázek o tom, zdali při dané úloze má čekat na výsledek v řádu sekund, minut nebo hodin. V této sekci bude rovněž kladen důraz na testování škálovatelnosti (a zvýšení výkonu, jenž škálovatelnost přináší), kterou Docker umožňuje velmi jednoduchým způsobem. V sekci 6.4 budou ukázány paměťové nároky, aby si mohl uživatel udělat představu o tom, kolik paměti RAM a perzistentního datového uložiště bude pro úlohy s platformou potřebovat. Poslední sekce 6.5 ukáže praktický postup při hledání důkazního materiálu, který byl popsán v kapitole 2. Testování proběhlo na dvou strojích:

- Ubuntu, Intel Core i5 (2 jádra) 2.5 GHz, 4 GB RAM, HDD disk, internetové připojení 20Mbit/s download, 10Mbit/s upload,
- macOS, Intel Core i7 (4 jádra) 2.7 GHz, 16 GB RAM, SSD disk, internetové připojení 150Mbit/s download, 150Mbit/s upload.

6.1 Jednotkové testy

Jednotkové testy slouží k automatizovanému testování a ověření korektnosti implementace dílčích systémů. Jednotka v kontextu jednotkového testování znamená samostatně testovatelná část programu. Jednotkou může být program, funkce nebo například proměnná. Jednotkový test by měl ideálně testovat pouze jednu malou oddělenou funcionalitu. Takové testy jsou vhodné převážně v těch případech, kdy se software tvoří postupně. Když přijde situace, kdy přibývá nová funkcionalita, popřípadě se modifikuje stávající, chce si být programátor jist, že všechny části softwaru fungují stále podle očekávání.

Scrapy

Testování Scrapy pavouků je netriviálním problémem, jelikož pavouků může být definováno mnoho a každý se může chovat v čase rozdílným způsobem, jelikož samotné webové stránky se v čase mění. Scrapy má pro tyto účely integrovaný mechanismus testování pavouků skrze kontrakty. Kontrakt by se dal přirovnat k jednotkovému testu. Dovolují programátorovi relativně jednoduše a rychle zjistit, zdali nějaká část konkrétního pavouka nepracuje dle očekávání.

Případ užití může být například takový, kdy vytvoříme projekt s mnoha pavouky a po několika týdnech si chceme ověřit, zda se změnila struktura webové stránky, kterou má pavouk prohledávat. Pokud se změnilo cokoli ve struktuře webové stránky, ze které pavouk doluje data, kontrakt selže a programátor je obeznámen, že pavouk nemůže vyhledat například konkrétní element *div*, ve kterém se měl nacházet odkaz na další stránku. Jestliže autoři webu změnili strukturu své webové stránky natolik, že logika pavouka neodpovídá aktuální situaci, kontrakt nám to přehledně sdělí. Kontrakty jsou definovány v komentáři funkce, která se stará o dolování dat. V platformě je testován pavouk *basic* a *login* na vlastním automaticky vygenerovaném testovacím webu.

Tento web generuje stránky, které mají dynamický obsah a při každém novém spuštění se mění. Stránky obsahují obrázky anglických měst a budov s náhodným textem a náhodnou cenou hotelů v librách. Oba dva pavouci (*basic* i *login*) mají definovány totožné kontrakty:

```
@url http://localhost:9312/properties/property_000000.html
@returns items 1
@scrapes title price description address image_urls
@scrapes url project spider server date
```

Oba dva pavouci mají definovány stejné kontrakty s tím rozdílem, že pavouk *login* musí provést autentizaci, aby se k webové stránce dostal. Zmíněné kontrakty definují následující:

- První pravidlo kontroluje, zdali se pavouk dostal na stránku */properties/property_000000.html*.
- Druhé pravidlo definuje podmínku, kdy musí pavouk získat ze stažené stránky pouze jeden prvek.
- Třetí pravidlo říká, které dílčí atributy se musí v prvku nacházet. Jedná se o nadpis, cenu, popis, adresu a odkaz na obrázek hotelu.
- Poslední pravidlo definuje opět další dílčí atributy, které musí výsledný prvek obsahovat. V tomto případě se jedná o atributy, které neobsahují obsah stránky, ale jde o metadata – URL adresa, název projektu, pavouka, serveru a datum.

Jestliže kterýkoli z definovaných kontraktů selže, celý běh pavouka se zastaví a Scrapy přehledně zobrazí, který z kontraktů byl porušen. V souboru *README.md* projektu *scrapit* je napsán postup, jak testy spustit.

Lemmiwinks

Testování archivačního rámce Lemmiwinks proběhlo za pomoci *pytest*, což je rámec podporující komplexní funkcionální testování pro projekty v jazyce Python. Výhody tohoto rámce spočívají především v následujících vlastnostech:

- Poskytuje detailní informace o selhaných testech, programátor tudíž nemusí složitě pátrat po příčině chyby.
- Automaticky prohledá celý projekt a nalezne testovací scénáře, není proto potřeba žádné konfigurace.
- Obsahuje systém managementu pro parametrizovatelné dlouho žijící zdroje (soubory, sockety).
- Dokáže spustit testy vytvořené rámcem *unittest*.
- Podporuje Python 2.7 a Python 3.4+.
- Bohatý výběr doplňků od open-source komunity.

Testy mají oddělenou adresářovou strukturu nezávislou na zbytku projektu *lemmit*:

- lemmit/
  - tests/
    - test_archive.py
    - test_db.py
    - test_redis.py

První ze souborů obsahující testy, *test_archive.py*, testuje samotný proces archivace webové stránky. Jedná se o testovací metodu *test_archive_page* třídy *TestArchivation*. Jako parametry obsahuje názvy metod *lemmiwinks* a *mock_html_data*. Tyto parametry slouží jako prerekvizity k samotnému testu. První prerekvizita spustí smyčku událostí Lemmiwinks a inicializuje databázi PostgreSQL. Druhá prerekvizita poté simuluje databázi Redis a přichystá HTML data k archivaci. Samotný test vytvoří událost archivace tohoto falešného HTML dokumentu a vloží tuto událost do smyčky Lemmiwinks. Následně test ověří, že se skutečně provedla archivace HTML dokumentu a na disku se nachází MAFF archiv.

Druhý ze souborů, *test_db.py*, testuje správnost modulů sloužících pro ukládání metadat do databáze PostgreSQL. Stěžejní je zde test *test_storage_jobs*, který volá metodu z rámce Lemmiwinks odpovědnou za správné uložení metadat. První prerekvizita *lemmiwinks* má stejný účel jako v prvním testovacím souboru. Druhá prerekvizita, *mock_db_data*, připraví strukturu naplněnou falešnými metadaty, které následně test uloží. Stejně jako v předchozím případě test spouští smyčku událostí, do které vloží událost uložení metadat. Následně je proveden dotaz do databáze a provedeno ověření, zda se v ní data skutečně nacházejí.

Poslední soubor, *test_redis.py*, testuje správnost třídy *RedisWrapper*, která pracuje s databází Redis. První prerekvizita, *test_connection*, vytváří instanci třídy *RedisWrapper*,

která následně inicializuje spojení do databáze Redis. Druhá prerekvizita, *mock_redis_input*, připraví falešná data, která vyhovují formátu, v jakém data ukládá do databáze Scrapy. Následně *test_serialization* provede serializaci a zkontroluje, zdali jsou data zbavena escape sekvencí a jsou připravena k archivaci.

6.2 Manuální testování

Manuální testování je rovněž důležitou součástí testování produktu. Tím, že si programátor sám ručně vyzkouší, jak jeho program pracuje, dokáže nalézt spoustu chyb pouhým pohledem. V této sekci bude vyzkoušena funkcionalita platformy jako celku. Scrapy bude sbírat webové stránky na serveru *fit.vutbr.cz*. Následně bude zobrazena výsledná archivovaná webová stránka spolu s ukázkou databáze PostgreSQL, zdali jsou uložena korektní metadata. Byla použita následující konfigurace:

```
JOB = 'holky'
PATTERN = r'.*holky.*'
SPIDER_MODULES = ['basicplatform.spiders']
START_URLS = ['http://www.fit.vutbr.cz']
LOG_LEVEL = 'DEBUG'
ROBOTSTXT_OBEY = True
LOG_FILE = '/tmp/scrapy_spider.log'
CLOSESPIDER_ITEMCOUNT = 50
```

Jak lze vyčíst z konfigurace, zadaná úloha prohledá pouze prvních 50 stránek počínaje *http://www.fit.vutbr.cz*. Hledáme a archivujeme všechny ty stránky, které obsahují řetězec *holky*. Do souboru */tmp/scrapy_spider.log* Scrapy loguje všechnu svoji aktivitu s úrovní *DEBUG*.

Nejprve je provedeno ověření správnosti archivace pohledem na archivovanou stránku, zdali vypadá stejně jako ta původní.

Obrázek 6.1: Ukázka výsledné archivované stránky.

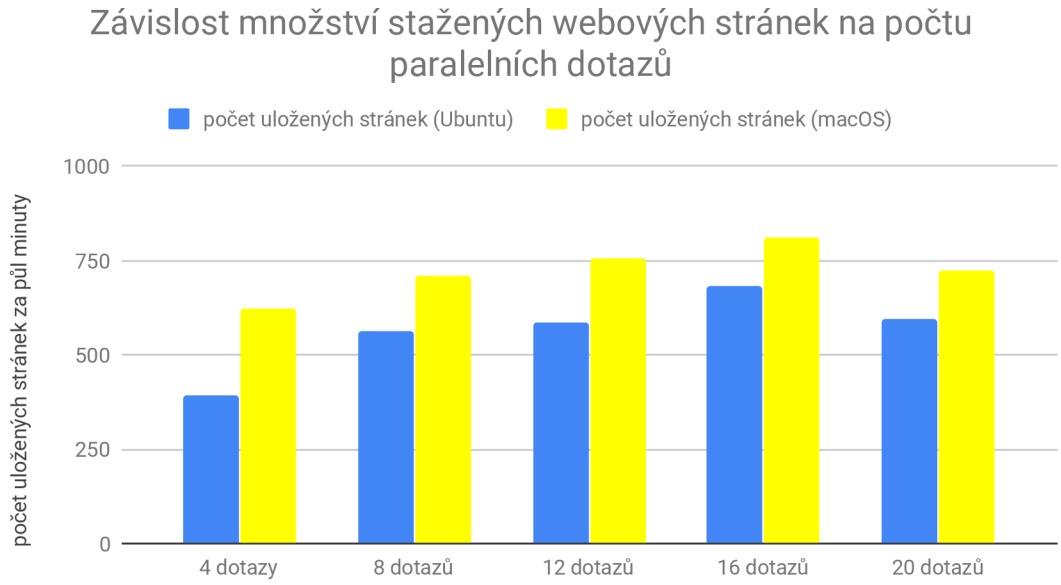
Na obrázku 6.1 lze vidět archivovaná stránka, jejíž původ je na adrese *http://www.fit.vutbr.cz/*. Stránka byla archivována z toho důvodu, jelikož obrázek vpravo, na kterém je napsáno *FIT sluší dívkám* je ve skutečnosti odkaz, který obsahuje řetězec *holky*, což vyhovuje regulárnímu výrazu. Originál je od archivované stránky k nerozeznání, což bylo cílem. Další část validace spočívá v PostgreSQL databázi. Tabulky *jobs* a *data* musí obsahovat korektní metadata.

Z důvodu přehlednosti byly následující dva obrázky přesunuty do přílohy. Na obrázku A.1 lze vidět tabulku obsahující metadata *job* a *regex_pattern*. Obsah tabulky odpovídá počáteční konfiguraci Scrapy. Na obrázku A.2 lze vidět tabulku obsahující metadata *path*, *job*, *url*, *spider*, *server*, *data* a *regex_match*. Obsah tabulky se shoduje s reálnými daty.

6.3 Výkonnost

V této sekci bude provedeno výkonnostní testování obou částí platformy – sběrače dat Scrapy a archivátoru Lemmiwinks. Testování proběhne odděleně kvůli povaze platfromy. Kdyby bylo provedeno výkonnostní testování celé platformy jako takové, nebylo by vzhledem k celé architektuře dostatečně vypovídající, jelikož zde hraje roli horizontální škálovatelnost dílčích částí. Tím pádem může být libovolná instance přítomna v platformě v mnoha instancích. Jednotlivé instance mohou být přitom rozprostřeny přes vícero fyzických systémů.

Oficiální Scrapy dokumentace doporučuje pro dosažení maximálního výkonu zvyšovat počet paralelně běžících HTTP dotazů, dokud se nebude zatížení CPU pohybovat v průměru na 80 %. Tím je zajištěno, že úzkým hrdlem není procesor a nenastane CPU saturace. Na obrázku 6.2 je vyobrazen graf znázorňující závislost množství stažených webových stránek uložených do databáze Redis na počtu paralelních dotazů. Počet dotazů začíná na čtyřech a končí na dvaceti, jelikož více dotazů na těchto dvou testovacích strojích již nevede ke zlepšení výkonu.



Obrázek 6.2: Závislost množství stažených webových stránek na počtu paralelních dotazů.

Jako testovací scénář byl vybrán ten nejhorší možný, kdy je každá stránka zpracována a uložena do databáze Redis. V praxi by takový případ mohl nastat pravděpodobně pouze tehdy, pokud by byl zadaný regulární výraz příliš obecný. Z grafu vyplívá, že ideální konfigurace spočívá v 16 paralelních HTTP dotazech, přičemž na CPU nenastává saturace. Testování části Scrapy byla provedena na webu *http://www.fit.vutbr.cz* a test běžel při každé konfiguraci 30 sekund. Při 16 paralelních dotazech nastala na Ubuntu téměř saturace CPU a další zvýšení paralelních dotazů již vedlo k degradaci celkového výkonu. Oproti tomu při 16 paralelních dotazech měl systém macOS CPU v průměru pouze na 50 %. Ovšem při dalším zvýšení paralelních dotazů došlo ke snížení počtu posbíraných stránek. Příčina leží překvapivě na protilehlém webovém serveru, vůči kterému byla platforma testována – web server nebyl schopen dodat paralelně více webových stránek než bylo maximum při 16 dotazech. Tudíž lze předpokládat, že stroj se systémem macOS má potenciál překonat s téměř maximálním využitím CPU uložení více než tisíce webových stránek během 30 sekund.

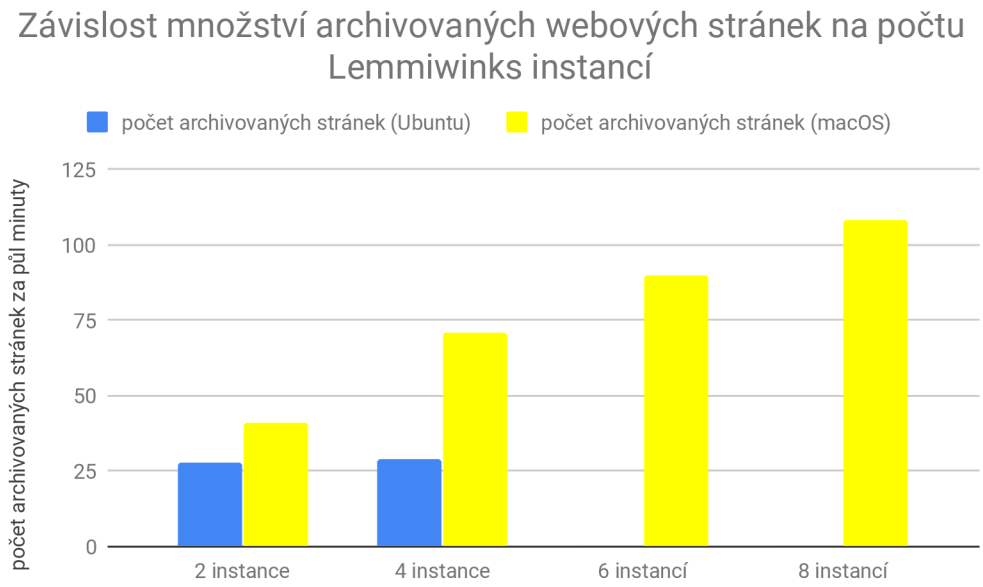
Výkonnostní testování archivační části Lemmiwinks je podstatně jednodušší, jelikož Lemmiwinks nemá možnosti konfigurace. Postačí naplnit databázi Redis metadaty a HTML dokumenty, které může Lemmiwinks archivovat. Testy byly provedeny na totožné doméně jako při výkonnostním testování Scrapy. Výsledky ukazují, že za 30 sekund dokáže stroj s Ubuntu archivovat v průměru 17 a macOS 18 stránek. Vzhledem k tomu, o kolik je stroj se systémem macOS výkonější než stroj se systémem Ubuntu, lze jednoznačně říci, že při úloze archivace záleží především na I/O síťových operacích, jelikož procesor byl zatížen minimálně.

Škálovatelnost

Jak bylo řečeno v sekci 4.1, platforma dokáže libovolně horizontálně škálovat až do toho momentu, kdy se úzkým hrdlem stane databáze Redis nebo PostgreSQL. Jelikož úzké hrdlo částí Scrapy i Lemmiwinks spočívá v I/O operacích, je velmi nepravděpodobné, že by celou platformu brzdily právě databáze. Testování proběhne stejně jako v předchozí sekci, pouze s tím rozdílem, že bude postupně navyšován počet instancí Lemmiwinks o dvě až do konečného počtu logických CPU. Testování výkonu Scrapy při horizontálním škálování bude vynecháno, jelikož jak bylo patrno na obrázku 6.2, Scrapy dokáže na dostatečně výkonném stroji vygenerovat tolik paralelních HTTP dotazů, že úzkým hrdlem se může stát protilehlý webový server namísto samotné platformy.

Na obrázku 6.3 jsou zobrazeny výsledky, kolik se archivovalo webových stránek vzhledem k počtu instancí Lemmiwinks. U Ubuntu systému lze jasně vidět, že počet instancí nezvýší výkon celé platformy. Úzkým hrdlem při archivaci není výkon procesoru, ale I/O síťové operace. Internetové připojení systému Ubuntu bylo natolik nekvalitní, že více než 29 webových stránek za 30 sekund nemohlo být archivováno. Ovšem mezi jednou a dvěma instancemi je téměř lineární nárůst výkonu, což by odpovídalo teorii, že nárůst výkonu Lemmiwinks je lineárně závislý na počtu instancí až do maximálního počtu logických CPU.

Na druhé straně, u systému macOS vidíme stálý nárůst výkonu. To je zapřičiněno převážně kvalitním internetovým připojením s nízkou latencí. Při pohledu na počet archivovaných stránek při jedné až čtyřech instancích vidíme jako u systému Ubuntu téměř lineární nárůst výkonu. Následně u šesti a osmi instancích už je nárůst výkonu minoritní. Vzhledem k tomu, že při osmi instancích bylo každé z jader CPU vytíženo na průměrných 30 %, lze opět předpokládat, že v případě systému macOS nebylo úzké hrdlo na straně platformy, ale na straně webového serveru, který nezvládl paralelně dodat tolik webových zdrojů.



Obrázek 6.3: Závislost množství stažených webových stránek na počtu Lemmiwinks instancí.

6.4 Množství uložených dat

V závislosti na povaze a složitosti úlohy se velmi mění požadavky na paměť. V tomto typu testování byl zvolen ten nejhorší možný případ, tedy uložení každé stránky, na kterou pavouci narazí do databáze Redis. Toho bylo docíleno volbou velmi obecného regulárního výrazu. Jelikož je Redis konfigurován tak, aby ukládal záznamy pouze do paměti, bude monitorováno množství paměti RAM, které je potřeba pro uložení všech nasbíraných záznamů. Dále budou tyto stránky archivovány, čili bude monitorována databáze PostgreSQL a její nároky na perzistentní datové uložiště.

Testování proběhlo na serveru *fit.vutbr.cz*, přičemž během krátkého běhu testovací úlohy v platformě bylo nasbíráno do databáze Redis 7264 záznamů, jinými slovy bylo staženo 7264 HTML stránek s metadaty, které jsou připraveny na archivaci. V databázi Redis byly v rámci zobrazení paměťových nároků provedeny následující operace:

```
127.0.0.1:6379> llen queue
(integer) 7264
127.0.0.1:6379> MEMORY USAGE queue
(integer) 59424495
```

Z výsledku operace *MEMORY USAGE queue* vyplývá, že pro uložení 7264 záznamů Redis potřebuje 59424495 bytů, což je téměř 57 MB. V rámci optimalizace bylo z celkového počtu 7264 archivováno pouze 844 webových stránek. Pro zobrazení paměťových nároků databáze PostgreSQL byly provedeny následující SQL dotazy:

```
SELECT pg_size_pretty(pg_relation_size('public.jobs'));
SELECT pg_size_pretty(pg_relation_size('public.data'));
```

Návratová hodnota prvního z příkazů je 8192 bytů a druhého příkazu 352 KB. Oproti databázi Redis se jedná pouze o zlomek paměťových nároků, jelikož se do PostgreSQL ukládají pouze metadata.

Posledním ukazatelem je celková velikost uložených MAFF archivů na disku. Archivováno bylo 844 webových stránek, přičemž dohromady zabírají 463 MB. Nejvíce místa ovšem vyžadují obrázky a jiná multimédia obsažená na webových stránkách.

6.5 Hledání důkazního materiálu

V této sekci bude krátce popsán postup při možném hledání důkazního materiálu na webu. Mějme situaci, kdy vyšetřovatele zajímá trestná činnost na serveru *pastebin.com*, kam mohou lidé vkládat libovolný text a sdílet jej s ostatními. Vyšetřovatele zajímají tři oblasti kyberkriminality – uniklá čísla bankovních karet, adresy sítě Tor, na kterých se odehrává trestná činnost a adresy Bitcoin peněženek. Všechny testy, které proběhly trvaly 30 minut a byly provedeny na stroji se systémem Ubuntu.

V prvním případě, kdy vyšetřovatel hledá bankovní karty, použije pro vyhledání regulární výraz *(?:4[0-9]{12}(?:[0-9]{3})?|[25][1-7][0-9]{14}|6(?:011|5[0-9][0-9])[0-9]{12}|3[47-][0-9]{13}|3(?:0[0-5]|[68][0-9])[0-9]{11}|(?:2131|1800|35\d{3})\d{11})*, který vyhledá bankovní karty Visa, MasterCard, American Express, Diners Club, Discover a JCB. Bylo nalezeno 2054 případů, kdy se shodoval řetězec z webové stránky s regulárním výrazem. Je nutno ovšem připomenout, že je potřeba u takového regulárního výrazu počítat i s *false positive* výskyty, čili že na stránce ve skutečnosti žádné číslo bankovní karty není a přesto byla tato stránka archivována.

Ve druhém případě hledá vyšetřovatel adresy sítě Tor. Během 30 minut bylo nalezeno pouze 32 výskytů. Použitý regulární výraz byl *(?:https?://)?(?:www)?(\S*?\.onion)\b* a jak už samotný výraz napovídá, je hodně konkrétní a bude tedy obsahovat minimum *false positive* výskytů (po prozkoumání bylo zjištěno, že neobsahuje žádný).

Poslední případ, kdy vyšetřovatel hledá adresy Bitcoin peněženek není tak zcela zdařilý, jelikož regulární výraz pokrývající tyto řetězce, *[13][a-km-zA-HJ-NP-Z0-9]26,33*, je velmi obecný. Z toho důvodu bylo během stanového časového intervalu nalezeno 10310 výskytů, přičemž lze předpokládat, že většina z těchto výskytů bude *false positive*. V tomto případě je potřeba zamyslet se nad konkrétnějším regulárním výrazem, který najde skutečně pouze výskyty adres Bitcoin peněženek.

Kapitola 7

Závěr

Tato práce popisuje řešení pro automatizované stažení, analýzu a archivaci webových stránek. Výsledkem práce je platforma určená pro vyšetřovatele a bezpečnostní experty České republiky. Platforma se skládá ze čtyř částí – Scrapy pro stažení a analýzu dat, databáze Redis sloužící jako fronta, Lemmiwinks pracující jako archivační nástroj webových stránek a databáze PostgreSQL pro perzistentní uložení dat. Všechny komponenty pracují v kontejnerech Docker. Cílovou skupinou této práce jsou bezpečností experti, ale i běžní uživatelé, kteří potřebují z nějakého důvodu vyhledat na webu požadovanou informaci a zaarchivovat stránku, na které byla informace nalezena.

Práce byla zahrnuta do projektu TARZAN. Jde o integrovanou platformu pro zpracování digitálních dat z bezpečnostních incidentů. Projekt je zaměřen na analýzu a detekci nových forem kybernetické kriminality především v prostředí mobilních a komunikačních aplikací a v prostředí internetu věcí. Cílem projektu je výzkum nových technologií a metod pro efektivní vyšetřování bezpečnostních incidentů. Výsledky budou demonstrovány na případech z praxe, například na detekci provozu P2P sítí, bezpečnostní analýze mobilních zařízení či řešení incidentů v oblasti Bitcoinů.

V rámci řešení diplomové práce jsem se zúčastnil konference inovací, technologií a vědy v IT Excel@FIT 2019[1], kde jsem získal ocenění odborné komise za přínos prevence kriminality na webu. Dále byl v rámci této práce publikován článek na serveru *root.cz* na téma sběr informací na webu a jejich analýza[2].

7.1 Možnosti rozšíření této diplomové práce

Primární rozšíření platformy by mohlo spočívat ve vyvoření grafického uživatelského rozhraní, které by zaujímalo pozici *Management* ve znázorněné architektuře, viz obrázek 4.1. Mělo by se jednat o SPA (*single page application*), kde by si uživatel navolil vlastnosti dané úlohy a odeslaný HTTP dotaz by byl odeslán na aplikační programové rozhraní přiloženému k procesu Scrapy. Tím by se spustil celý proces archivace. Uživatel by měl rovněž aktualizovanou informaci o stavu jeho úlohy (zdali stále běží nebo již skončila a s jakým návratovým kódem). Následně by si uživatel mohl zobrazit metadata z databáze PostgreSQL přidružené k jeho provedené úloze.

[1] http://excel.fit.vutbr.cz/submissions/2019/005/5.pdf
[2] https://www.root.cz/clanky/predstaveni-knihovny-scrapy-pro-tvorbu-web-crawleru/

Literatura

[1] BeauHD: *French Officer Caught Selling Access To State Surveillance System On the Darkweb*. [Online; navštíveno 11.10.2018].
URL https://yro.slashdot.org/story/18/10/07/1741230/french-officer-caught-selling-access-to-state-surveillance-system-on-the-darkweb

[2] Brenner, S. W.: *Cybercrime: Criminal Threats from Cyberspace*. Praeger, 2010, ISBN 978-0313365461.

[3] *Caleb Hattingh*: *Using Asyncio in Python 3*. O'Reilly Media, 2018, ISBN 9781491999691.

[4] *Dimitrios Kouzis-Loukas*: *Learning Scrapy*. Packt Publishing, 2016, ISBN 978-1-78439-978-8.

[5] Fielding, R. T.; Gettys, J.; Mogul, J. C.; aj.: Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, RFC Editor, June 1999, http://www.rfc-editor.org/rfc/rfc2616.txt.
URL http://www.rfc-editor.org/rfc/rfc2616.txt

[6] Goodman, M.: *Future Crimes: Everything Is Connected, Everyone Is Vulnerable and What We Can Do About It*. Doubleday, 2015, ISBN 978-0385539005.

[7] Heydt, M.: *Python Web Scraping Cookbook*. Packt Publishing, 2018, ISBN 978-1-78728-521-7.

[8] Hinduja Sameer K.: *Bullying Beyond the Schoolyard*. Sage Publications Ltd, 2014, ISBN 9781483349930.

[9] Johnson, F.: *Web Content Mining Techniques: A Survey. International Journal of Computer Applications*, ročník 47, č. 11, 7 2012.

[10] *Katharine Jarmul, Richard Lawson*: *Python Web Scraping*. Packt Publishing, 2017, ISBN 9781786462589.

[11] Kolouch, J.: *CyberCrime*. CZ.NIC, 2016, ISBN 978-80-88168-15-7.

[12] Kročil, B.: *Praxe dokazování kyberkriminality [online]*. 2015 [cit. 2018-09-29].
URL https://is.muni.cz/th/xnz63/

[13] Liu, B.: *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data*. Springer Science & Business Media, 2011, ISBN 9783642194603.

[14] Marjan Falahrastegar, Hamed Haddadi, Steve Uhlig: *The Rise of Panopticons: Examining Region-Specific Third-Party Web Tracking*. Technická zpráva, Queen Mary University of London, 2014.
URL http://mort.io/publications/pdf/tma14-panopticons.pdf

[15] Martin Hron: *Are smart homes vulnerable to hacking?* [Online; navštíveno 11.10.2018].
URL https://blog.avast.com/mqtt-vulnerabilities-hacking-smart-homes

[16] Ministerstvo vnitra ČR, odbor bezpečnostní politiky a prevence kriminality: *Audit národní bezpečnosti*. [Online; navštíveno 27.09.2018].
URL https://www.vlada.cz/assets/media-centrum/aktualne/Audit-narodni-bezpecnosti-20161201.pdf

[17] Mitchell, R.: *Web Scraping with Python*. O'Reilly Media, 2015, ISBN 978-1-491-91027-6.

[18] Mitnick, K. D.: *The Art of Deception: Controlling the Human Element of Security*. Wiley, 2007, ISBN 978-0-471-23712-9.

[19] *Pethuru Raj*: *Learning Docker*. Packt Publishing, 2015, ISBN 9781786462923.

[20] Polčák, L.: Co skrývá síť BitTorrent? Technická zpráva, 2018.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=11885

[21] Potti, P.; of North Florida. School of Computing, U.; University of North Florida. College of Computing, E. . C.: *On the Design of Web Services: SOAP Vs. REST*. 2011.
URL https://books.google.cz/books?id=awtFtwAACAAJ

[22] Serečun, V.: *Automatizovaná rekonstrukce webových stránek*. Diplomová práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2018.
URL http://www.fit.vutbr.cz/study/DP/DP.php?id=21138

[23] Vysoké učení technické, Fakulta informačních technologií, Brno: *Integrovaná platforma pro zpracování digitálních dat z bezpečnostních incidentů*. [Online; navštíveno 19.12.2018].
URL http://www.fit.vutbr.cz/units/UIFS/grants/index.php.cs?id=1063

[24] Xavier Roche: *HTTrack website copier*. [Online; navštíveno 24.11.2018].
URL https://www.httrack.com/

[25] *Zakir Laliwala, Abdulbasit Shaikh*: *Web Crawling and Data Mining with Apache Nutch*. Packt Publishing, 2013, ISBN 978-1-78328-685-0.

Příloha A

Ukázka obsahu tabulek databáze PostgreSQL

Obrázek A.1: Ukázka obsahu tabulky jobs.

```
1  SELECT * FROM public.data
2
```

Data Output | Explain | Messages | Notifications

	path [PK] character varying (255)	job charact	url character varying (1000)	spider characte	server character varyin	date character varying (255)	regex_match character varying (255)
1	/root/lemmit/archived/hZHiTys...	holky	http://www.fit.vutbr.cz/	basic	8363f38fa125	2019-04-13 14:14:57.9...	<a href="/holky/"><img src=...
2	/root/lemmit/archived/q4bN0Z...	holky	http://www.fit.vutbr.cz/verejn...	basic	8363f38fa125	2019-04-13 14:14:58.4...	<tr valign=top><td align=rig...
3	/root/lemmit/archived/zm8sZ5...	holky	https://www.fit.vutbr.cz/	basic	8363f38fa125	2019-04-13 14:14:59.3...	<a href="/holky/"><img src=...
4	/root/lemmit/archived/vWULxH...	holky	http://www.fit.vutbr.cz/.en	basic	8363f38fa125	2019-04-13 14:15:01.8...	<a href="/holky/.en"><img s...
5	/root/lemmit/archived/2T1hxa2...	holky	https://www.fit.vutbr.cz/verej...	basic	8363f38fa125	2019-04-13 14:15:01.9...	<tr valign=top><td align=rig...

Obrázek A.2: Ukázka obsahu tabulky data.

Příloha B

Instalace, konfigurace, nasazení

Nasazení je s použitím Docker Compose velmi jednoduché. Postačí přejít do adresáře projektu[1] a postupovat podle návodu:

- Je možné modifikovat v souboru docker-compose.yml atribut <volumes> podle požadavků uživatele.
- docker-compose build
- docker-compose up -d

První příkaz vytvoří pomocí nástroje Docker Compose obrazy vyhovující Dockeru. Druhý příkaz potom vytvoří samotné kontejnery na základě zhotovených Docker obrazů a spustí je na pozadí.

Následující příkazy nejsou povinné, mohou ovšem přijít vhod při ladění služeb, či při řešení různých problémů s platformou:

- docker-compose ps -a
- docker exec -it <container> /bin/bash
- docker-compose stop
- docker-compose start

První z příkazů vypíše všechny kontejnery, které aktuálně běží, ale i ty, které neběží a jsou jen vytvořené a připravené k použití. Druhý příkaz uživateli dovolí přistoupit přímo do terminálu běžícího kontejneru, což je velmi výhodné při ladění služby, která běží v kontejneru a nechová se podle očekávání. Třetí a čtvrtý příkaz zastaví, respektive spustí všechny kontejnery přidružené k aktuálnímu projektu, ve kterém se uživatel nachází (ve skutečnosti je detekován pouze soubor *docker-compose.yml* v aktuálním adresáři).

[1] https://gitlab.com/tomaskocman/proof_platform