VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

VYLEPŠENÍ SYSTÉMOVÝCH TESTŮ PROJEKTU JSHELTER

BETTER JSHELTER SYSTEM TESTING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **DAVID KONEČNÝ**
AUTHOR

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

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

144997

Ústav:	Ústav informačních systémů (UIFS)
Student:	**Konečný David**
Program:	Informační technologie
Specializace:	Informační technologie
Název:	**Vylepšení systémových testů projektu JShelter**
Kategorie:	Web
Akademický rok:	2022/23

Zadání:

1. Seznamte se se systémovými testy projektu JShelter a analyzátorem volaných webových API vzniklého rozšířením diplomové práce Marka Schauera *Oblíbenost JavaScriptových API internetového prohlížeče*.
2. Seznamte se s metodami klasifikace zobrazené webové stránky (např. pro detekci stránek s kontaktním formulářem, přihlášením uživatele, výběrem zboží, košíkem apod.). Vyhledejte metody založené jak na analýze dokumentového objektového modelu stránky, tak na analýze odkazů z hlavní stránky webové prezentace (zobrazený text, alternativní text, jméno cílového dokumentu).
3. Navrhněte mechanismus řízení procházení webových stránek založený na metodách nastudovaných v bodě 2 zadání tak, aby byl použitelný v nástrojích z bodu 1 zadání. Navrhněte vylepšení systémových testů. Návrhy konzultujte s vedoucím práce.
4. Návrh implementujte.
5. Vytvořenou implementaci spusťte a analyzujte výsledky. Nedostatky zjištěné systémovými testy reportujte projektu JShelter.
6. Vyhodnoťte dosažené výsledky a navrhněte možná pokračování práce.

Literatura:

- Martin Bednář. *Automatické testování projektu JavaScript Restrictor*. Brno, 2020. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií
- Marek Schauer. *Oblíbenost JavaScriptových API internetového prohlížeče*. Brno, 2021. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií.
- Libor Polčák, Marek Saloň, Giorgio Maone a kol. JShelter: Give Me My Browser Back, ArXiV https://arxiv.org/abs/2204.01392, 2022.

Při obhajobě semestrální části projektu je požadováno:
První 3 body zadání včetně technické zprávy.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Polčák Libor, Ing., Ph.D.**
Vedoucí ústavu:	Kolář Dušan, doc. Dr. Ing.
Datum zadání:	1.11.2022
Termín pro odevzdání:	10.5.2023
Datum schválení:	4.5.2023

Abstrakt

Tato práce se zabývá návrhem a implementací vylepšení systémových testů projektu JShelter. Systémové testy projektu JShelter obsahují jisté nedostatky, které budou odstraněny. Dále budou také systémové testy vylepšeny a rozšířeny o nové funkcionality založené na metodách analýzy webových stránek. V rámci této práce bude také navržen nový mechanismus procházení webových stránek založený na již existujicím mechanismu Marka Schauera. Tento mechanismus bude založený na nástrojích OpenWPM, Selenium a Kubernetes. Do tohoto mechanismu budou také začleněny vylepšené systémové testy. Tento mechanismus bude implementován. Výsledky získané pomocí tohoto mechanismu budou také vyhodnoceny.

Abstract

This thesis deals with the proposal of improving and implementation the system tests of the JShelter project. The system tests of the JShelter project contain certain shortcomings that will be removed. Furthermore, the system tests will also be improved and expanded with new functionalities based on website analysis methods. As part of this thesis, a new website crawling mechanism will also be proposed based on the already existing mechanism of Marek Schauer. This mechanism will be based on OpenWPM, Selenium and Kubernetes. Improved system tests will also be integrated into this mechanism. This mechanism will be implemented. The results obtained using this mechanism will also be evaluated.

Klíčová slova

Testování, OpenWPM, Selenium, Kubernetes, Docker, Webový prohlížeč, Firefox

Keywords

Testing, OpenWPM, Selenium, Kubernetes, Docker, Web browser, Firefox

Citace

KONEČNÝ, David. *Vylepšení systémových testů projektu JShelter*. Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Libor Polčák, Ph.D.

Vylepšení systémových testů projektu JShelter

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Libora Polčáka, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
David Konečný
9. května 2023

Poděkování

Rád bych poděkoval panu Ing. Liboru Polčákovi, Ph.D. za cenné rady a odborné vedení, které mi poskytoval během psaní této práce. Mé poděkování patří také mé rodině za neutichající podporu, kterou mi poskytovala během psaní této práce.

Obsah

1 Úvod **2**

2 Systémové testy projektu JShelter a analyzátor volaných webových API **3**
2.1 Systémové testy projektu JShelter 3
2.2 Analyzátor volaných webových API 5

3 Metody klasifikace zobrazených webových stránek **7**
3.1 Analýza rozložení stránky 7
3.2 Analýza odkazů z hlavní stránky webové prezentace 10

4 Mechanismus řízení procházení webových stránek a vylepšení systémových testů **12**
4.1 Mechanismus řízení procházení webových stránek 12
4.2 Vylepšení systémových testů 12

5 Implementace vylepšení mechanismu řízení procházení webových stránek a systémových testů **15**
5.1 Vylepšení mechanismu řízení procházení webových stránek 15
5.2 Detekce odkazů na zajímavé podstránky 18
5.3 Integrace algoritmu VIPS 19
5.4 Integrace systémových testů projektu JShelter 21
5.5 Sestavení a modifikace Docker kontejneru 22

6 Analýza výsledků získaných implementovaným nástrojem **24**
6.1 Výkonnost implementovaného nástroje 24
6.2 Nedostatky implementovaného nástroje 28
6.3 Budoucí vývoj 29

7 Závěr **30**

Literatura **31**

A Obsah přiloženého paměťového média **33**

Kapitola 1

Úvod

Každý softwarový produkt prochází při vývoji různými fázemi, jednou z nich je testování. Testování je nedílnou součástí vývoje každého softwarového produktu, díky testování je možné určit, zda byl produkt vytvořen správně a nevykazuje žádné nežádoucí chování.

Správnost chování produktu jako celku je ověřována systémovými testy, které mohou mít různý charakter v závislosti na testovaném produktu. Ve většině případů se jedná o testy automatické, testování tedy probíhá vždy podle stejných postupů a se stejnými daty.

Na FIT VUT v Brně je vyvíjen výzkumný projekt JShelter, jehož cílem je poskytnou uživateli větší soukromí při procházení webových stránek. Projekt JShelter se snaží chránit uživatele před sledováním jeho aktivity (například omezením vytváření otisku prohlížeče) a omezuje webovým stránkám přístup k vybraným informacím uživatele a prohlížeče (například informace o grafické kartě a poloze uživatele).

Cílem této práce bude navrhnout vylepšení již existujících systémových testů projektu JShelter, které disponují určitými nedostatky. V rámci této práce budou nedostatky systémových testů odstraněny a testy budou dále vylepšeny a rozšířeny o nové funkcionality pomocí metod analýzy webových stránek. Tyto vylepšené testy budou implementovány a začleněny do již existujícího mechanismu procházení webových stránek, čímž dojde k vytvoření univerzálního nástroje pro automatizované testování projektu JShelter. Výsledky získané pomocí tohoto mechanismu budou analyzovány a vyhodnoceny.

Díky tomuto nástroji bude možné provádět automatizované testování projektu JShelter na velkém množství vstupních dat. Přínosem bude také sjednocení dvou mechanismů – systémové testy projektu JShelter a mechanismu procházení webových stránek do jednoho univerzálního nástroje, který bude možné využít i v jiných nástrojích souvisejících s projektem JShelter.

Práce je členěna do několika kapitol, které popisují jednotlivé kroky návrhu vylepšení. Ve druhé kapitole jsou představeny systémové testy projektu JShelter a mechanismus procházení webových stránek. Třetí kapitola popisuje metody analýzy rozložení webové stránky. Ve čtvrté kapitole je pak představen návrh vylepšení systémových testů projektu JShelter a návrh nového mechanismu procházení webových stránek. Samotná implementace nového mechanismu procházení webových stránek je popsána v páté kapitole. V šesté kapitole jsou analyzovány a vyhodnoceny výsledky získané pomocí tohoto mechanismu. Celá práce je shrnuta v sedmé závěrečné kapitole.

Kapitola 2

Systémové testy projektu JShelter a analyzátor volaných webových API

JShelter[1] je rozšíření webového prohlížeče, které poskytuje uživateli kontrolu nad událostmi, jež probíhají v prohlížeči, a dále také poskytuje uživateli ochranu před sledováním jeho aktivity.

Všechny moderní webové prohlížeče mají implementovanou podporu jazyka JavaScript, což je objektově orientovaný a událostmi řízený skriptovací jazyk, který je hojně využíván pro tvorbu webových stránek. Kromě podpory jazyka JavaScript implementují webové prohlížeče také poměrně velké množství JavaScriptových API, skrze která lze přistupovat i do částí operačního systému, v němž běží webový prohlížeč. Některá JavaScriptová API umožňují přístup ke kameře, mikrofonu nebo také k různým senzorům, kterými zařízení disponuje. Díky těmto API může mít webová stránka, respektive aplikace, která zde běží, přístup k citlivým datům uživatele. Na základě informací získaných z těchto API mohou webové stránky vytvářet tzv. otisk webového prohlížeče, nebo dokonce otisk konkrétního zařízení uživatele, ze kterého webové stránky navštěvuje [4]. Nejen tyto poznatky stály za vznikem projektu JShelter.

2.1 Systémové testy projektu JShelter

Cílem systémových testů je především otestovat funkční projekt jako celek. Při systémovém testování jsou testovány funkční a nefunkční požadavky. Funkční požadavky mohou zahrnovat výpočty, technické detaily, manipulaci, zpracování dat a další specifické funkce, jež definují, co má systém plnit [21]. Jedná se množinu funkcí, které jsou pro uživatele aplikace (systému) dostupné, a které tedy může používat.

Nefunkční požadavky definují kritéria, na základě kterých je možné rozhodnout o kvalitě systému [23]. Mezi vlastnosti systému, jež mohou nefunkční požadavky vyžadovat pro splnění požadované kvality, patří mimo jiné výkon, omezení (provozní či jiná), modifikovatelnost, přenositelnost, spolehlivost, použitelnost a další [5].

Lze tedy říci, že funkční požadavky definují, *co* má systém umět, respektive *čím* má disponovat, a nefunkční požadavky definují, *jak* má být systém vytvořen, aby dosahoval požadované kvality.

[1] https://jshelter.org/

Systémové testy projektu JShelter, tehdy ještě pod názvem JavaScript Restrictor, byly vytvořeny jako součást diplomové práce Martina Bednáře v roce 2020. Jak již bylo zmíněno, cílem systémových testů projektu JShelter je otestovat projekt jako celek. Vzhledem k tomu, že JShelter je rozšíření webového prohlížeče, které může poměrně značně ovlivnit chování a vzhled webové stránky, je nutné ověřit, zda je obsah navštívené webové stránky s aktivním rozšířením JShelter odlišný od obsahu webové stránky, která byla zobrazena bez tohoto rozšíření. V případě že je zobrazený obsah navštívené webové stránky s rozšířením JShelter odlišný od webové stránky navštívené bez tohoto rozšíření, je nezbytné tento stav zaznamenat. Na základě výsledků systémových testů je možné projekt JShelter neustále zdokonalovat. Systémové testy se skládají ze dvou hlavních částí – získávání dat a analýzy získaných dat.

2.1.1 Získávání dat

Získávání dat spočívá v automatizovaném navštívení určitého množství nejnavštěvovanějších webových stránek podle žebříčku Tranco[2]. V rámci systémových testů je navštěvováno 500 webových stránek. Samotné automatizované navštívení (procházení) webových stránek je realizováno paralelně a distribuovaně pomocí nástroje Selenium, konkrétně pomocí jeho prostředí Selenium Grid, které je implementováno v jazyce Java.

Existuje více možností, jak lze prostředí Selenium Grid využít. V rámci systémových testů je využita varianta *hub and node*. Prostředí obsahuje jeden prvek *hub*, který slouží jako hlavní část, jež řídí samotné procházení webových stránek a optimálně využívá prvky typu *node*. V rámci varianty *hub and node* zde může figurovat jeden či více prvků typu *node*, který detekuje nástroj Selenium WebDriver. Nástroj Selenium WebDriver komunikuje a řídí činnost webového prohlížeče pomocí HTTP požadavků [22]. V tomto případě byl použit webový prohlížeč Google Chrome. Jednotlivé prvky nástroje Selenium Grid mohou běžet současně i na různých zařízeních s různými operačními systémy; jejich vzájemná komunikace probíhá skrze síť. Prvky typu *node* jsou identifikovány pomocí IP adresy a síťového portu [11].

Automatické procházení webových stránek pomocí Selenium Grid s požadovanými parametry je spouštěno a řízeno skripty implementovanými v jazyce Python. Tyto skripty také provádí úvodní konfiguraci testovacího prostředí na základě hodnot, které jsou v nich obsaženy.

Testy běží ve dvou průchodech, nejprve bez aktivního rozšíření JShelter, následně s aktivním rozšířením JShelter. Při navštívení webové stránky se zachycuje snímek obrazovky a záznamy v konzoli. Tato data jsou později využita k samotné analýze. URL adresy webových stránek určených k navštívení jsou uloženy ve formátu CSV. Tyto URL adres následně výše zmíněné skripty, které řídí chod testů, předávají nástroji WebDriver [1].

Velká část moderních webových stránek je tvořena dynamickým obsahem, při opakovaném načtení, se jejich obsah mění – jedná se o přirozenou vlastnost. Tento fakt systémové testy projektu JShelter nijak nezohledňují. Další nevýhodou je fakt, že při procházení webových stránek v rámci systémových testů nejsou dále procházeny podstránky.

2.1.2 Analýza dat

Analýza získaných dat se dělí na dva samostatné celky – analýza záznamů v konzoli a analýza snímků obrazovky. Při analýze záznamů v konzoli se porovnávají záznamy získané

[2] https://tranco-list.eu/

z webového prohlížeče bez rozšíření JShelter a záznamy z webového prohlížeče s aktivním rozšířením JShelter. Na základě jejich shody nebo podobnosti je možné určit, zda konrétní záznamy vznikly jako reakce na přítomnost rozšíření JShelter či nikoliv. Při analýze záznamů v konzoli se využívá tři metod:

- přesná shoda,
- Levenshteinova vzdálenost,
- kosinová podobnost.

V případě že je Levenshteinova vzdálenost nebo kosinová podobnost vyšší než stanovená úroveň, je záznam prohlášen za shodný. Stejně tak dojde-li k přesné shodě záznamů. U takto shodných záznamů není nutné uvažovat o jejich souvislosti s rozšířením JShelter.

Analýza snímků obrazovky, tedy vizuální analýza, využívá metody rozdílu snímků obrazovky. Snímky jsou porovnávány po jednotlivých pixelech, následně vznikne nový snímek, složený pouze z rozdílu jednotlivých pixelů. Tento rozdílový snímek je převeden do odstínů šedi. Následně je celý snímek agregován do jediné hodnoty vypočtením průměrné hodnoty pixelu. Tímto postupem je získáno číselné ohodnocení, které udává, jak moc se od sebe snímky liší [1].

2.2 Analyzátor volaných webových API

Analyzátor volaných webových API byl vytvořen v rámci diplomové práce pana Marka Schauera v roce 2021. Tento analyzátor funguje na podobném principu jako systémové testy projektu JShelter představené v kapitole 2.1. Automatizované procházení webových stránek je implementováno v prostředí OpenWPM[3], které v sobě zapouzdřuje také již zmíněný nástroj Selenium, dále pak webový prohlížeč Mozilla Firefox [8].

2.2.1 Princip webového crawleru

Automatizované procházení webových stránek je realizováno pomocí tzv. webového crawleru. Webový crawler (v češtině je také možné se setkat s výrazem webový prohledávač) je automatizovaný program, který prochází webové stránky a sbírá z nich informace. Principem webového crawleru je procházení webového prostoru pomocí URL odkazů a následné stahování obsahu webových stránek za účelem dalšího zpracování.

Při procházení jednotlivých webových stránek webový crawler shromažďuje různé informace, například názvy stránek, obsah, klíčová slova, popisy nebo obrázky. Tyto informace mohou být následně využívány mj. pro tvorbu webových vyhledávačů [17].

2.2.2 Princip funkce analyzátoru webových API

Při automatizovaném procházení webových stránek bylo v prohlížeči aktivní rozšíření Web-API Manager, jehož hlavním cílem je blokování uživatelem zvolených JavaScriptových API. Toto rozšíření bylo v rámci diplomové práce ještě rozšířeno za účelem získávání dat o volaných JavaScriptových API, která byla následně použita k tvorbě statistik. Informace o volaných API jsou získávány pomocí tzv. Proxy objektů. Díky těmto Proxy objektům je možné zachytit a také předefinovat chování existujících objektů v JavaScriptu [16]. Pro výběr

[3] https://github.com/openwpm/OpenWPM

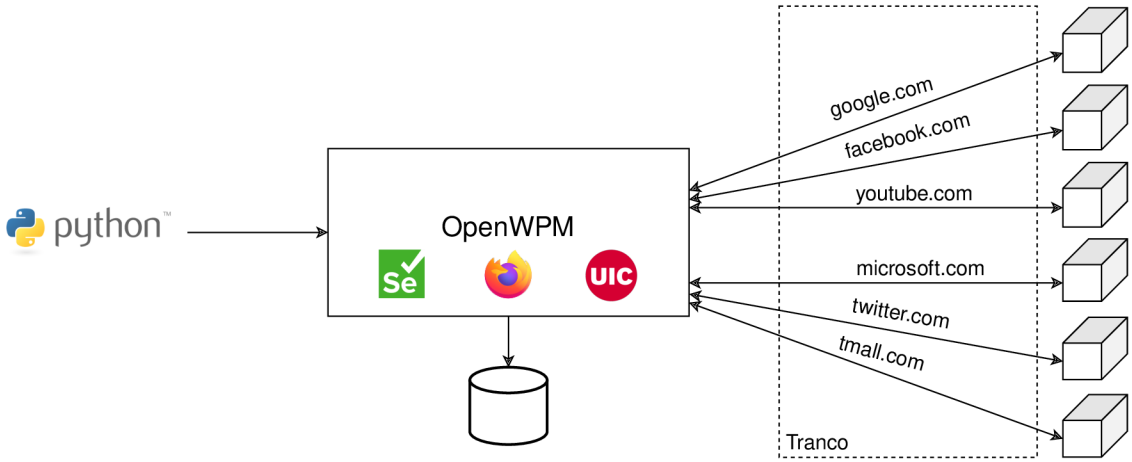
vhodné sady webových stránek byl použit, stejně jako u systémových testů projektu JShelter, žebříček Tranco s rozdílem, že webových stránek bylo navštíveno více konkrétně 10000. Dalším rozdílem je způsob uložení URL adres webových stránek určených k navštívení. Samotné URL adresy stránek se nachází v souboru formátu JSON. Cesta k tomuto souboru je zadávána jako parametr při spuštění nástroje OpenWPM. Procházení bylo spouštěno na více zařízeních současně za účelem navštívení více stránek v menším čase [20].

Samotný nástroj OpenWPM je koncipován pro běh v kontejneru Docker[4]. Docker je open-source platforma pro vývoj, nasazení a běh aplikací v kontejnerech. Kontejner je izolované prostředí, které obsahuje aplikaci a všechny její závislosti, což znamená, že se aplikace může spouštět na jakémkoli počítači bez nutnosti instalace závislostí na daný počítač. Docker umožňuje jednotný způsob balení a distribuci aplikací v kontejnerech. To je užitečné zejména v cloudovém prostředí, kde mohou být aplikace snadno škálovatelné [7].

2.2.3 Kubernetes

Tento přístup je poměrně náročný na výpočetní výkon, později byl proto výsledek diplomové práce pana Schauera převeden do prostředí Kubernetes. Prostředí, respektive nástroj Kubernetes, slouží jako systém pro orchestraci virtualizace na úrovni operačního systému. Tento nástroj podporuje mj. právě kontejnery Docker. Kubernetes zajišťuje neustálou dostupnost – v případě selhání jednotky, je vytvořena jednotka nová, případně je proveden pokus o obnovení původní jednotky.

Pomocí Kubernetes lze zvýšit efektivitu procházení webových stránek díky přístupu *StatefulSets*. Tento přístup vytváří a udržuje v běhu určitý počet běžících jednotek. Tyto jednotky jsou nazývány *pod*. Každý *pod* využívá jeden kontejner Docker. Právě určitý počet těchto *podů* vykonává samotné procházení webových stránek [15]. Pro uchování seznamu webových stránek, které jsou určeny k navštívení, je použita databáze Redis[5]. Veškerá inicializace, naplnění Redis databáze daty a logika procházení je implementována pomocí skriptů v jazyce Python. Architekturu prostředí s analyzátorem volaných webových API zobrazuje obrázek 2.1.



Obrázek 2.1: Architektura prostředí s analyzátorem volaných webových API. Inspirace z [14].

[4] https://www.docker.com/
[5] https://redis.io/

Kapitola 3

Metody klasifikace zobrazených webových stránek

Hlavním cílem metod klasifikace zobrazených webových stránek je získání užitečných informací o obsahu a vzhledu webové stránky. Těchto metod je využíváno velmi často při automatizovaném procházení webových stránek, jehož cílem je průchod a analýza velkého vzorku webových stránek. V těchto případech není možné projít webové stránky ručně z důvodu velké časové náročnosti a možného subjektivního hodnocení.

Tyto metody je možné na základě způsobu klasifikace rozdělit na tři kategorie:

- metody založené na analýze vzhledu,
- metody založené na analýze dokumentového objektového modelu,
- metody založené na strojovém učení.

Metody založené na analýze vzhledu mohou být heuristické, v případě manuální analýzy mohou vycházet z lidské intuice. Při automatizované analýze mohou využívat různé algoritmy, které mohou být založené na strojovém učení.

Metody založené na analýze dokumentového objektového modelu (angl. *Document Object Model*), zkráceně DOM. Pomocí DOM je možné k webové stránce přistupovat jako k datové struktuře strom, což je vhodné zejména pro programovací jazyky. DOM se skládá z objektů, reprezentující jednotlivé elementy, které jsou tvořeny značkami [13]. Tyto metody se zaměřují na konkrétní značky a jejich atributy, pracují se strukturou značkovacího jazyka HTML pomocí DOM. Příkladem takové značky, která je analyzována, muže být hodnota atributu `href` značky `<a>`.

Poslední kategorií jsou metody založené na strojovém učení. Zde se využívá algoritmů strojového učení pro komplexní analýzu obsahu webové stránky. Pomocí algoritmů strojového učení je analyzován nejen vzhled stránky, ale i její obsah a další vlastnosti. Pro dosažení uspokojivých výsledků těchto metod je nezbytné využít kvalitní sadu trénovacích dat [9].

3.1 Analýza rozložení stránky

Tato metoda byla představena v článku od Lei Fu et al. [9]. Metoda vychází z kategorie metod založených na analýze dokumentového objektového modelu. Cílem této metody je

detekce sekcí rozložení webové stránky (angl. *layout*), z jejichž obsahu jsou následně získány informace. Motivací pro vznik této metody bylo zajímavé zjištění autorů – klasická analýza DOM modelu v případech velkého výskytu hypertextových odkazů vykazuje chyby. Autoři se tedy rozhodli vylepšit již existující algoritmus Vision-based Page Segmentation (VIPS) [3].

Metoda provádí analýzu a extrakci dat pomocí následujících kroků:

1. analýza dokumentového objektového modelu,
2. analýza rozložení stránky pomocí algoritmu VIPS,
3. filtrování okrajových bloků,
4. detekce hranic kandidátních bloků obsahu,
5. extrakce obsahu.

Pro analýzu DOM modelu je nejprve nutné samotný DOM model vytvořit. Většina vysokoúrovňových programovacích jazyků disponuje rozhraním pro převod HTML dokumentu do DOM a umožňuje s DOM modelem dále pracovat. V rámci převodu do DOM je také provedena syntaktická kontrola.

3.1.1 Algoritmus VIPS

Algoritmus VIPS provádí analýzu rozložení webové stránky na základě DOM modelu. Nejprve jsou extrahovány všechny bloky na základě stromové struktury HTML v DOM. Následně jsou vyhledávány ve stromové struktuře DOM vhodné HTML elementy, které by mohly představovat oddělovače mezi jednotlivými sekcemi stránky. Na základě těchto oddělovačů, jež jsou reprezentovány svislými a horizontálními čarami, které neprotínají žádný blok, je vytvořena struktura webové stránky. V rámci algoritmu VIPS je webová stránka W reprezentována jako trojice:

$$W = (B, S, \delta) \tag{3.1}$$

$B = \{B_1, B_2, \dots B_n\}$ je množina bloků, které nejsou překryty. $S = \{S_1, S_2, \dots S_n\}$ je konečná množina oddělovačů, kde každý oddělovač má svoji váhu, definující jeho viditelnost. δ značí relaci každých dvou bloků v množině bloků B. Relaci každých dvou bloků δ lze vyjádřit jako:

$$\delta : B \times B \to S \cup \{NULL\} \tag{3.2}$$

Máme-li dva bloky – B_i a B_j z množiny bloků B takové, že $\delta(B_i, B_j) \neq NULL$, lze říci, že tyto dva bloky jsou od sebe odděleny oddělovačem $\delta(B_i, B_j)$, nebo se nachází vedle sebe. V opačném případě lze říci, že mezi bloky B_i a B_j se nachází další bloky. Díky tomuto mechanismu, lze pomocí algoritmu VIPS rozdělit webovou stránku na několik nezávislých bloků, které jsou odděleny oddělovači z množiny S; tyto bloky je možné lokalizovat pomocí souřadnic. Množina bloků tvořící rozložení webové stránky je dále označována jako S_{tb} [9, 3].

Vzniklo několik implementací tohoto algoritmu, mezi které patří implementace v jazyce Java, jež vznikla v rámci diplomové práce pana Tomáše Popely [18]. Existuje také implementace[1] v jazyce Python, vycházející z původní implementace od firmy Microsoft.

Dalším krokem metody analýzy rozložení stránky je filtrování okrajových bloků, které pracuje s výstupem předchozího kroku, tedy analýzy pomocí algoritmu VIPS. Cílem tohoto

[1] https://github.com/wushuartgaro/VipsPython

kroku je odstranění okrajových bloků, které většinou neobsahují žádný zajímavý obsah, může se jednat o reklamní bloky, navigační lišty aj. Při filtrování jsou z množiny S_{tb} odstraněny bloky, jejichž šířka pokrývá celou stránku. Po odstranění těchto bloků je u všech bloků nacházejících se v množině S_{tb} provedena ještě jedna filtrace. Jejím cílem je detekce bloků, které se nachází v levé části stránky. Takto detekované bloky jsou vloženy do množiny levých bloků S_{lb}. Při detekci je využito dvou prahových hodnot – α a β. Autoři této metody zvolili hodnoty $\alpha = 10$ px a $\beta = \frac{1}{3}$ [9]. Tento krok je popsán také v následujícím pseudokódu:

Algoritmus 1 Třetí krok analýzy rozložení stránky [9]

```
function TOP_BOTTOM_SCAN(S_tb)
    if current_block.width == webpage.width then
        Remove current_block from S_tb
function BOTTOM_TOP_SCAN(S_tb)
    body
    if current_block.width == webpage.width then
        Remove current_block from S_tb
for all block ∈ S_tb do
    if (block.left < α || block.right < α) && block.width / webpage.width < β then
        continue
    else
        Add block into S_lb
```

Další krok je zaměřen na detekci obsahových bloků v množině S_{lb}. Cílem je nalézt bloky, které ohraničují bloky obsahové. Tato detekce probíhá na základě tří parametrů:

- Délka textu v bloku
- Poměr mezi délkou textu hyperlinkových odkazů a celkovou délkou textu
- V případě že je v bloku obsažen interaktivní objekt, kterým může být například `<form>`, blok není chápán jako obsahový

V tomto kroku se podobně jako v předchozím využívá prahových hodnot α a β, dále zde figuruje množina S_{tcb}, která obsahuje dočasné kandidátní obsahové bloky. Tato množina slouží pro pomocné účely. Množina S_{cb} obsahuje bloky, které jsou konečnými kandidátními bloky obsahu. Nakonec zde ještě figurují bloky B_f a B_l, mezi nimiž se nachází právě kandidátní bloky obsahu z množiny S_{cb}. Algoritmický popis tohoto kroku se nachází níže:

Algoritmus 2 Čtvrtý krok analýzy rozložení stránky [9]

```
for all block ∈ S_lb do
    if block.text_length > α & block.link_text_ration < β & block.no_form then
        Add block into S_tcb
Sort block in S_cb by block.width
Find the blocks which have the most and the same width, then add them to S_cb
Pick out the beginning block B_f and ending block B_l from S_tb
return B_f and B_l
```

V posledním kroku jsou z bloků obsahu v množině S_{tb}, které se nachází mezi bloky B_f a B_l, extrahovány informace ve formě textu. Zde je ještě tento krok popsán pomocí algoritmu:

Algoritmus 3 Pátý (finální) krok analýzy rozložení stránky [9]

```
1: for all block ∈ S_tb do
2:     if block is between B_f and B_l then
3:         content += block.text
4: return content
```

3.2 Analýza odkazů z hlavní stránky webové prezentace

Cílem analýzy odkazů z hlavní stránky webové prezentace v rámci této práce bude detekce zajímavých sekcí webové stránky. Zajímavé sekce často obsahují interaktivní prvky – například formuláře. Jedná se o sekce webových stránek, které uživatelé často vyhledávají a používají, je tedy vhodné tyto sekce detekovat a zaměřit se na jejich chování při aktivním rozšíření JShelter. Dále je také žádoucí rozšíření JShelter testovat na různých typech webových stránek (e-shopy, zpravodajské weby…), je tedy důležité analyzovat chování různých zajímavých sekcí různých typů webových stránek. Mezi tyto zajímavé sekce můžeme zařadit například nákupní košík v případě e-shopů, sekce přihlášení uživatele, kontaktní formulář a další.

Tato metoda provádí analýzu hypertextových odkazů v jazyce HTML. Hypertextový odkaz umožňuje uživateli přejít na jinou webovou stránku kliknutím. V jazyce HTML je tento odkaz reprezentován elementem `<a>` a jeho atributem `href`, který obsahuje URL adresu stránky, na níž je proveden přechod. Obsahem atributu `href` může být také zobecnění URL adresy – řetězec URI. Řetězec URI (Uniform Resource Locator) se skládá z několika částí, z nichž nejdůležitější jsou:

- Schéma – způsob přenosu informací, který je identifikován síťovým protokolem nebo klíčovým slovem.
- Doména nejvyššího řádu, doména druhého řádu, případně domény nižších řádů, ze kterých se skládá adresa webové stránky

Mezi další části, z nichž se skládá URI, patří mimo jiné: síťový port, cesta k souboru, název souboru, dotaz na formulářová data a další. Příkladem řetězce URI může být: `https://www.fit.vut.cz/search/?q=UIFS` [12, 2].

Tato metoda obsahuje dva hlavní kroky:

1. Analýza dokumentového objektového modelu
2. Analýza hypertextových odkazů

V rámci prvního kroku je postupováno podobně jako u prvního kroku metody analýzy rozložení stránky představené v kapitole 3.1. V tomto kroku jsou také získány URL adresy všech odkazů z hlavní stránky webové prezentace.

Ve druhém kroku je provedena samotná analýza odkazů. Z množiny získaných odkazů je nejprve nutné odstranit odkazy, které nesměřují na webové stránky. Jedná se například o odkazy, nevyužívající protokolů HTTP a HTTPS, nebo odkazy obsahující schéma `mailto`,

`file` aj. Dále jsou také odstraněny odkazy, které směřují mimo doménu webové prezentace. Následně je analyzován samotný text odkazu, z nějž je možné detekovat zajímavou sekci webové stránky pomocí klíčových slov. Pro tuto analýzu je stěžejní zejména umístění na serveru a také název souboru. V určitých částech odkazu je tedy nutné vyhledat klíčová slova, která mohou identifikovat zajímavou sekcí webové stránky. Pro vyhledávání klíčových slov v odkazu je využita Levenshteinova vzdálenost.

3.2.1 Levenshteinova vzdálenost

Levenshteinova vzdálenost uvádí, kolik operací se znakem (vkládání, mazání a záměna) je nutné provést k tomu, aby vznikl z jednoho porovnávaného řetezce řetězec druhý [10]. Pro využití Levenshteinovy vzdálenosti při detekci existence klíčových slov v odkazu je vhodné její výslednou hodnotu c_a převést do relativní reprezentace c_r pomocí vztahu:

$$c_r = \frac{c_a}{max(len(string1), len(string2))}. \tag{3.3}$$

Predikát $len(string)$ udává délku textového řetězce $string$. Relativní reprezentace c_r může nabývat hodnot $< 0, 1 >$, kde 0 značí úplnou shodu řetězců, naopak 1 značí úplnou odlišnost řetězců [1].

Implementace výpočtu Levenshteinovy vzdálenosti existují mimo jiné v jazycích Python[2] a PHP[3].

Stanovením podobnosti dvou řetězců – textu odkazu a klíčového slova, a porovnáním s určitou prahovou hodnotou c_t, lze detekovat odkaz na zajímavou sekci webové stránky.

[2] https://pypi.org/project/Levenshtein/
[3] https://www.php.net/manual/en/function.levenshtein.php

Kapitola 4

Mechanismus řízení procházení webových stránek a vylepšení systémových testů

V této kapitole bude představen mechanismus řízení procházení webových stránek s využitím metod představených v kapitole 3. Dále zde bude také představeno vylepšení systémových testů projektu JShelter.

4.1 Mechanismus řízení procházení webových stránek

Navržený mechanismus bude vycházet z mechanismu Marka Schauera představeném v rámci kapitoly 2.2 z důvodu, že již představený mechanismus byl navržen a implementován poměrně robustně. Dalším důvodem pro využití tohoto mechanismu byl požadavek vedoucího této práce, jehož cílem je sjednocení mechanismu pro procházení webových stránek a mechanismu výběru podstránek. Cílem bude zaintegrovat existující systémové testy projektu JShelter do již zmíněného existujícího mechanismu. Systémové testy budou ještě v rámci této práce vylepšeny. Kód vytvořený v rámci této práce bude využívám i v jyných nástrojích souvisejících s projektem JShelter.

V rámci systémových testů již nebude potřebná část zajišťující získávání dat. Tato část je již implementována v rámci prostředí OpenWPM, respektive prostředí Selenium, které je jeho součástí. Ze systémových testů bude tedy zachována pouze část obstarávající analýzu dat. Jelikož se analyzovaná data skládají ze snímků obrazovky a záznamů v konzoli, bude nutné upravit konfiguraci prostředí OpenWPM tak, aby byla tato data ukládána. Tato úprava je možná díky využití prostředí Selenium, které tyto funkcionality podporuje. Výhodou tohoto řešení je také fakt, že systémové testy i obsluha a konfigurace prostředí OpenWPM jsou implementovány v jazyce Python. Při vzájemné komunikaci mezi těmito komponentami bude možné využít nativní rozhraní, které jazyk poskytuje.

4.2 Vylepšení systémových testů

V rámci vylepšení systémových testů projektu JShelter budou do testů implementovány následující funkcionality:

- analýza rozložení webové stránky metodou představenou v kapitole 3.1,

- analýza odkazů z hlavní stránky webové prezentace metodou představenou v kapitole 3.2,
- rozhraní pro konfiguraci systémových testů.

Systémové testy s tímto rozšířením budou začleněny do mechanismu řízení procházení webových stránek navrženém v kapitole 4.1

Cílem metody analýzy rozložení webové stránky, jejímž jádrem je algoritmus VIPS, je především extrakce obsahu z obsahových částí webové stránky. Pro účely detekce zajímavých sekcí webové stránky by bylo vhodné detekovat spíše sekce webových stránek, které obsahové nejsou, a to zejména pro to, že zajímavé sekce webových stránek se většinou nachází právě v částech neobsahových. Za tímto účelem bude implementován pravý opak této metody, respektive pravý opak kroků 3 a 4. Díky modifikaci této metody je možné detekovat okrajové bloky obsahu, ve kterých se mohou nacházet právě zajímavé sekce webových stránek. Pro druhý krok této metody bude využita již zmíněná existující implementace algoritmu VIPS v jazyce Python.

Pozorováním bylo zjištěno, že odkazy vedoucí na zajímavé sekce webových stránek velmi často obsahují klíčová slova, která tyto sekce identifikují. Tato klíčová slova se nachází v části odkazu, obsahující umístění podstránky. URL adresa sekce e-shopu s nákupním košíkem může mít například tento tvar: `https://www.czc.cz/kosik`. Metoda bude implementována v jazyce Python, a to zejména z důvodu vytvoření homogenního prostředí – ostatní komponenty v rámci této práce také jsou nebo budou implementovány v jazyce Python. Výhodou jazyka Python je také poměrně velká podpora knihoven, které umožňují analýzu a zpracování obsahu HTML stránky do formátu DOM. Zde bude použita knihovna Beautiful Soup[1], vycházející z populárních analyzátorů (angl. *parsers*) lxml a html5lib. Beautiful Soup mimo jiné umožňuje vyhledávání elementů v DOM a podporuje formát Unicode [19].

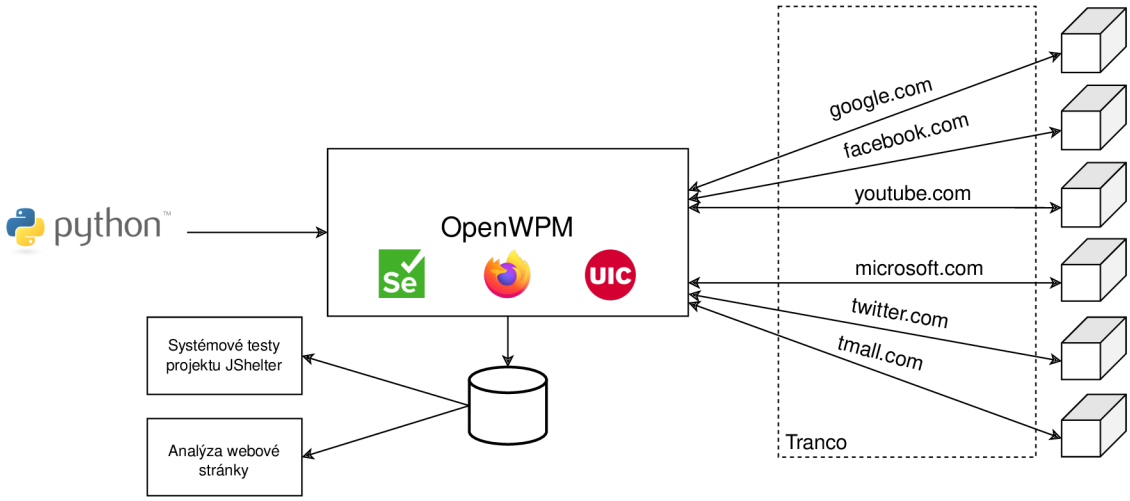
S textem odkazu jsou následně provedeny řetězcové operace popsané v druhém kroku metody v kapitole 3.2. Výsledný řetězec odkazu je následně vstupem pro výpočet Levenshteinovy vzdálenosti, druhým vstupem je řetězec klíčového slova z množiny klíčových slov reprezentujících určitou zajímavou sekci webové stránky. Pomocí vztahu 3.3 je poté výsledná hodnota Levenshteinovy vzdálenosti c_a mezi těmito řetězci převedena do relativní reprezentace c_r. Je-li hodnota relativní reprezentace c_r větší než prahová hodnota c_t, lze tento odkaz prohlásit jakou vedoucí na zajímavou sekci webové stránky.

Hlavní nevýhodou systémových testů projektu JShelter je špatná modifikovatelnost. Zejména prahové hodnoty Levenshteinovy vzdálenosti a kosinové podobnosti je možné změnit pouze úpravou zdrojového kódu, což může být například v rámci experimentování poměrně nevhodné. Za účelem modifikovatelnosti bude implementováno rozhraní, které umožní lepší nastavení prahových hodnot a dalších parametrů systémových testů. Implementace rozhraní bude spočívat v jeho začlenění do konfiguračního souboru nástroje OpenWPM. Do tohoto souboru budou přidány další konfigurační proměnné, jež budou reprezentovat jednotlivé konfigurační parametry.

Nevýhodou systémových testů je také způsob pojmenování vygenerovaných adresářů pro data určená k analýze a výstupní data. Názvy těchto adresářů jsou při každém spuštění stejné. Existují-li již tyto adresáře a je provedeno spuštění systémových testů, adresáře s daty jsou přepsány. Tento stav bude vyřešen přidáním časového razítka (angl. *timestamp*) do názvů adresářů, čímž se docílí vytvoření adresářů s unikátními názvy při každém spuštění.

[1] https://pypi.org/project/beautifulsoup4/

Nový mechanismus řízení procházení webových stránek s integrovanými systémovými testy popisuje obrázek 4.1 níže:



Obrázek 4.1: Schéma navrženého mechanismu procházení webových stránek s integrací systémových testů projektu JShelter

Kapitola 5

Implementace vylepšení mechanismu řízení procházení webových stránek a systémových testů

Vylepšení mechanismu řízení procházení webových stránek a vylepšení systémových testů bylo implementováno jako rozšíření nástroje OpenWPM. V rámci implementace byl také nástroj OpenWPM používaný v mechanismu řízení procházení webových stránek aktualizován na nejnovější verzi 0.21.1. Stávající skripty nástroje byly rozšířeny, a také vytvořeny skripty nové.

5.1 Vylepšení mechanismu řízení procházení webových stránek

Pro účely systémových testů bylo nutné mechanismus řízení procházení webových stránek upravit tak, aby umožňoval dvojí průchod, a tedy dvojí navštívení každé webové stránky. Pro tyto účely byl implementován parametr `--jshelter`. Přítomnost tohoto parametru při spouštění mechanismu zajistí právě již zmíněné dvojí navštívení každé webové stránky. Druhé navštívení se liší od prvního přítomností aktivního rozšíření JShelter. Obě navštívení mají společnou přítomnost rozšíření uBlock Origin[1], jehož hlavním cílem je blokování zobrazení reklam na webové stránce. Díky tomuto rozšíření by měl být zredukován dynamicky se měnící obsah webové stránky, který by mohl znehodnocovat data, jež jsou později využita jako vstupní data systémových testů.

5.1.1 Běh mechanismu

Mechanismus využívá nástroje Kubernetes, který aplikuje virtualizaci pomocí nástroje Docker. Virtualizace pomocí nástroje Docker byla zvolena pro svoji nenáročnost, dále také proto, že zařízení, na němž Kubernetes běží, nemusí disponovat podporou hardwarové virtualizace, která je vyžadována například nástrojem QEMU. Prostředí Kubernetes je ovládáno

[1] https://ublockorigin.com/

nástrojem minikube[2], který umožňuje běh lokálního prostředí v rámci jednoho zařízení. Tento nástroj zajišťuje také lepší správu tohoto prostředí.

Pro první spuštění mechanismu je vyžadováno běžící prostředí Kubernetes s již aplikovanou konfigurací. Aplikace konfigurace je provedena načtením konfiguračních souborů ve formátu YAML, obsahujících mj. údaje o počtu *podů* určených k navštívení webových stránek, konfiguraci *podu*, ve kterém běží databáze Redis či název jmenného prostoru, v němž tyto *pody* běží.

Nástroj OpenWPM umožňuje získávat data potřebná pro účely systémových testů. Při běhu nástroje OpenWPM jsou tedy získávána následující data:

- snímky obrazovky,
- snímky celé obrazovky (snímek celého zobrazeného DOMu),
- zdrojové kódy zobrazené webové stránky v jazycích HTML, CSS a JavaScript,
- záznamy nástroje OpenWPM.

Soubory obsahující data, jež se vztahují ke konkrétní webové stránce nesou název: `<id návštěvy>-<otisk URL stránky><případné přípony><přípona souboru>`, kde ID návštěvy je unikátní vygenerované číslo návštěvy webové stránky, otisk URL stránky je vytvořen algoritmem MD5 a případná přípona obsahuje dodatečné informace o souboru (například informace, že data, nacházející se v souboru vznikla při aktivním rozšíření JShelter).

Nástroj OpenWPM využívá webový prohlížeč Firefox. Vzhledem k tomu, že funckionalita pro získávání záznamů z konzole webového prohlížeče nebyla dosud pro Firefox implementována[3] a jiný webový prohlížeč není v současné době nástrojem OpenWPM podporován, není možné tyto záznamy získávat.

Veškerá data, která za běhu nástroje OpenWPM vznikají (záznamy nástroje, snímky obrazovky, ...), jsou uložena na tzv. persistentních jednotkách (angl. *Persistent Volumes*). Tyto jednotky jsou dostupné i po ukončení mechanismu procházení a lze k nim přistupovat pomocí tzv. *dummy podů*. K persistentní jednotce je možné připojit *dummy pod* podobným způsobem, jakým jsou připojovány jednotky v unixových systémech (příkaz `mount`). Platí tedy, že ke každému *podu*, který vykonává navštěvování webových stránek, náleží také jeden *dummy pod* a také jedna persistentní jednotka.

Činnost vykonávána mechanismem řízení procházení webových stránek je popsána následujícím algoritmem:

[2] https://github.com/kubernetes/minikube
[3] https://github.com/mozilla/geckodriver/issues/1698

Algoritmus 4 Příkazy vykonávané mechanismem řízení procházení webových stránek

```
interesting_sites = []
for all site ∈ sites do
    site.GetCommand()
    site.SaveScreenshotCommand()
    site.ScreenshotFullPageCommand()
    site.VipsAnalysis()
    site.DumpPageSourceCommand()
    if run_with_jshelter == True then
        site.ScreenshotAnalysisCommand()
    interesting_sites.append(site.GetInterestingSitesLinks())
for all inter_site ∈ interesting_sites do
    inter_site.GetCommand()
    inter_site.SaveScreenshotCommand()
    inter_site.ScreenshotFullPageCommand()
    if run_with_jshelter == True then
        inter_site.ScreenshotAnalysisCommand()
```

Výše popsaná sekvence příkazů je vykonávána pro jeden běh navštívení webových stránek – při navštívení s rozšířením JShelter je tedy vykonána dvakrát.

5.1.2 Příkazy nástroje OpenWPM

Nástroj OpenWPM poskytuje svoji funkcionalitu v podobě tzv. příkazů. Pomocí těchto příkazů je možné modifikovat činnost mechanismu řízení procházení webových stránek. Jednotlivé příkazy popsané v algoritmu 4 jsou zapouzdřeny v příkazu `InterceptJavaScriptCommand()`. Tento příkaz vznikl v rámci diplomové práce Marka Schauera pro účely analýzy volaných JavaScriptových API [20].

V rámci implementace vylepšení systémových testů projektu JShelter a mechanismu řízení procházení webových stránek byl příkaz `InterceptJavaScriptCommand()` rozšířen o příkazy popsané výše. V rámci nástroje OpenWPM jsou dostupné tyto příkazy: `GetCommand()`, `SaveScreenshotCommand()`, `ScreenshotFullPageCommand()` a `DumpPageSourceCommand()`. OpenWPM poskytuje také další příkazy, ty však nebyly v rámci této implementace využity. Příkazy `VipsAnalysis()`, `ScreenshotAnalysisCommand()` a `SaveLogsCommand()` byly implementovány jako rozšíření nástroje OpenWPM. Funkcionality jednotlivých příkazů jsou popsány v následující tabulce:

Příkaz	Funkcionalita
`GetCommand()`	Navštívení webové stránky dle zadané URL
`SaveScreenshotCommand()`	Pořízení snímku obrazovky
`ScreenshotFullPageCommand()`	Pořízení snímku celé obrazovky
`DumpPageSourceCommand()`	Získání zdrojových kódu webové stránky
`VipsAnalysis()`	Analýza snímku obrazovky algoritmem VIPS
`ScreenshotAnalysisCommand()`	Obrazová analýza ze systémových testů od M. Bednáře
`SaveLogsCommand()`[4]	Získávání záznamů z konzole webového prohlížeče

Tabulka 5.1: Funkcionality příkazů nástroje OpenWPM

5.1.3 Podpůrný skript pro získávání dat z nástroje OpenWPM

Nevýhodou stále zůstává skutečnost, že data vznikající při běhu OpenWPM jsou v případě opakovaného spuštění přepsána. Pro odstranění tohoto nedostatku byl implementován v jazyce Python podpůrný skript `get_datadir`. Cílem tohoto skriptu je zkopírovat adresář s názvem `datadir`, jenž obsahuje data vzniklá při běhu OpenWPM, do hostitelského prostředí, a to vše až po dokončení běhu nástroje OpenWPM.

Skript každých 30 sekund kontroluje stav jednotlivých *podů*, jež vykonávají navštěvování webových stránek. Dokončí-li některý *pod* svou činnost, skript zajistí zkopírování adresáře `datadir` z persistentní jednotky skrze *dummy pod*. Adresář `datadir` je při kopírování přejmenován na název ve formátu: `datadir_<časové razítko>_<id podu>`, vznikne tak unikátní název, čímž je vyřešen problém představený v kapitole 4.2.

Tento skript je vhodné spustit při běhu prostředí OpenWPM. Jeho spuštění není nutnost, řeší však problém popsaný výše.

5.1.4 Struktura adresáře `datadir`

Struktura adresáře `datadir` je následující:

```
/
├── <název podu>-<id podu>.sqlite ... výstup analyzátoru volaných webových
│   API
├── openwpm.log ... záznamy nástroje OpenWPM
├── screenshots
│   ├── *.png ... snímky obrazovky, rozdílové snímky, výstupní snímky algoritmu
│   │   VIPS
│   └── *.html ... výstupy systémových testů nástroje JShelter
└── sources
    └── *.html ... zdrojové kódy navštívených webových stránek
```

Záznamy nástroje OpenWPM je možné zobrazit také při běhu *podu* – tedy v době, kdy *pod* provádí navštěvování a analýzu dat z webových stránek. Pořizování těchto záznamů je realizováno knihovnou logging[5], jež je standardní knihovnou jazyka Python.

Záznamy jsou zpravidla formátu: `<původce záznamu> - <závažnost> - <zpráva>`. Do souboru `openwpm.log` jsou navíc ukládány s časovým razítkem. Závažnost identifikuje míru vlivu události na samotný chod nástroje OpenWPM. Míry závažnosti jsou přebírány právě z knihovny logging, jedná se například o `DEBUG`, `INFO` či `ERROR`. Příkladem takového záznamu může být:

```
intercept_javascript_command - INFO - I am going to visit following subpage
of http://google.com: https://www.google.com/about/.
```

5.2 Detekce odkazů na zajímavé podstránky

Ze zdrojových kódů zobrazené webové stránky je možné získat odkazy, které je možné analyzovat. Pro samotnou analýzu je nejprve nutné načíst HTML soubor se zdrojovými kódy. Soubor je následně zpracován knihovnou Beautiful Soup s pomocí jejichž funkcionality jsou vyhledávány absolutní odkazy na webové stránky, tedy takové odkazy, které využívají

[4] Tento příkaz není z důvodů popsaných v kapitole 5.1.1 využíván.

[5] https://docs.python.org/3/library/logging.html

protokoly HTTP nebo HTTPS. Schéma těchto odkazů, jakožto řetězců URI, tedy obsahuje klíčová slova `http` nebo `https`.

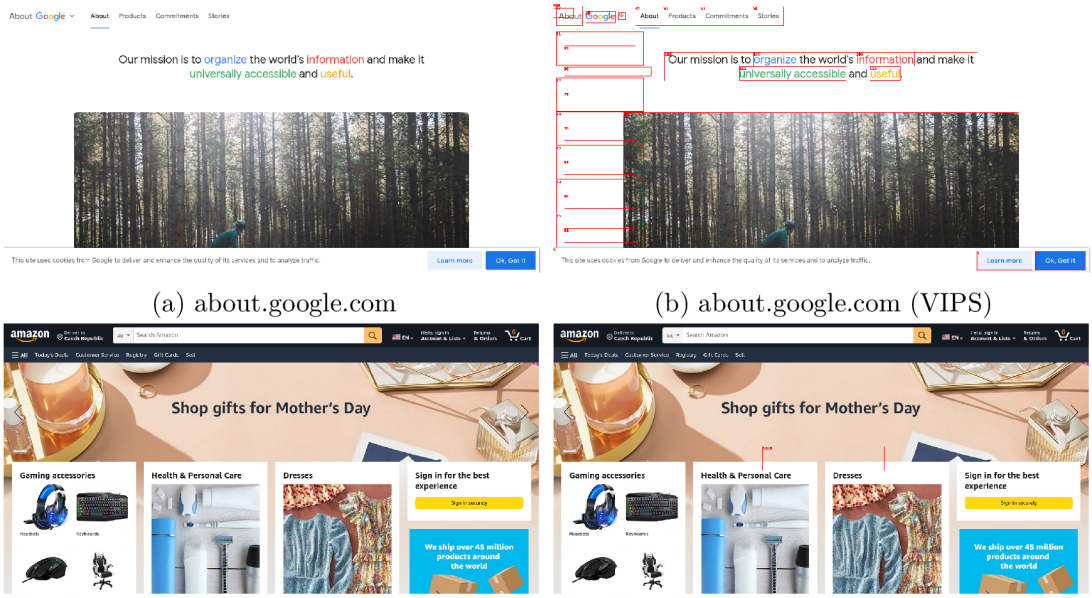
Z těchto odkazů jsou dále vyřazeny takové odkazy jejichž URL adresa obsahuje jiné doménové jméno než to, které náleží aktuální webové stránce. Tímto způsobem jsou eliminovány odkazy na jiné webové stránky – dále se tedy analyzují pouze odkazy na podstránky aktuální webové stránky. Detekovány jsou také relativní odkazy, a to na základě počátečního znaku „/“. Detekce klíčových slov, případně počátečního znaku „/“, jsou prováděny pomocí regulárních výrazů.

V dalším kroku jsou v odkazech na podstránky vyhledávána klíčová slova na základě dvou kritérií: výskytem klíčového slova v odkazu na základě regulárního výrazu a na základě hodnoty Levenshteinovy vzdálenosti mezi odkazem a klíčovým slovem. Pokud se nachází klíčové slovo v odkazu a zároveň je jeho relativní reprezentace hodnoty Levenshteinovy vzdálenosti menší než 0,9, je odkaz vyhodnocen jako odkaz vedoucí na zajímavou podstránku aktuální webové stránky. Prahová hodnota relativní reprezentace hodnoty Levenshteinovy vzdálenosti byla stanovena experimentováním. Tuto hodnotu je možné měnit v konfiguraci nástroje OpenWPM, stejně tak je možné měnit i seznam klíčových slov.

Odkazy vedoucí na zajímavé podstránky jsou následně naplánovány pro budoucí navštívení. U webových stránek navštívených z těchto odkazů se již tato analýza neprovádí – zamezí se tak velmi zdlouhavému a teoreticky i nekonečnému procházení podstránek. Je-li stejný odkaz na zajímavou podstránku detekován vícekrát, zajímavá podstránka je navštívena pouze jednou, díky tomu není zbytečně provedeno několik návštěv stejné podstránky.

5.3 Integrace algoritmu VIPS

Do mechanismu byla integrována implementace algoritmu VIPS v jazyce Python zmíněná v kapitole 3.1.1. Vstupem tohoto algoritmu je snímek obrazovky pořízen během navštívení webové stránky, algoritmus následně provede analýzu. Výstupem je série snímků s vyznačenými obsahovými bloky a textové soubory s obsahy detekovaných bloků obsahu. V rámci integrace bylo rozhraní implementace algoritmu VIPS mírně upraveno tak, aby přijímalo jako vstup soubory s názvem ve formátu, jenž vytváří nástroj OpenWPM.

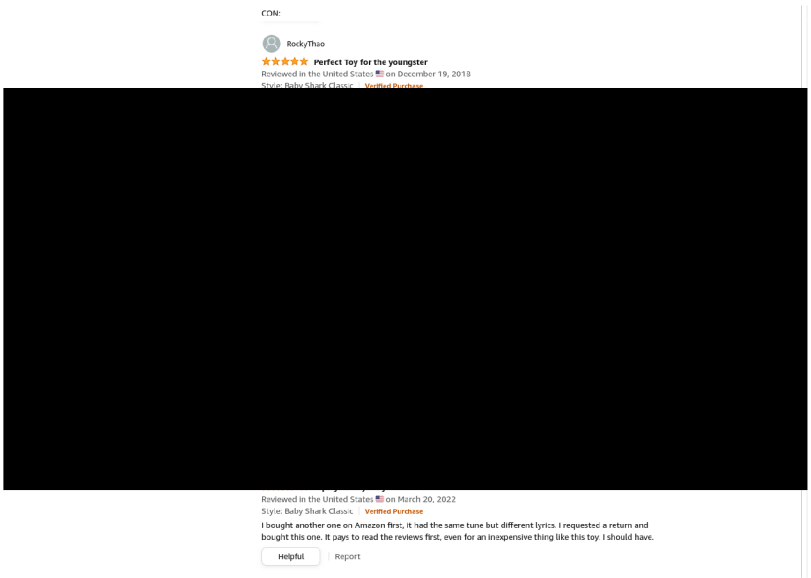


(a) about.google.com (b) about.google.com (VIPS)

(c) amazon.com (d) amazon.com (VIPS)

Obrázek 5.1: Ukázka výstupu algoritmu VIPS

Původně byly zvažovány pro vstup algoritmu VIPS snímky celé obrazovky (snímky celého zobrazeného DOMu), tyto snímky se však ukázaly být nevhodné pro tyto účely. Funkcionalita získávání snímků celé obrazovky dosud nebyla v nástroji Selenium, jenž je zapouzdřen a využit v nástroji OpenWPM, implementována. Autoři OpenWPM využívají vlastní implementaci, která nevytváří příliš dobré výsledky. Snímky celé obrazovky jsou skládány ze snímků obrazovky zobrazených ve výřezu stránky viditelném v okně webového prohlížeče (angl. *viewport*). Tímto skládáním však ve výsledném snímku vzniknou poměrně široké části obsahující pouze černou barvu. Jednu z těchto částí je možné vidět na obrázku 5.2. Z tohoto důvodu byly použity pro vstup právě snímky *viewportu*.



Obrázek 5.2: Nedostatky snímku celé obrazovky na jeho části

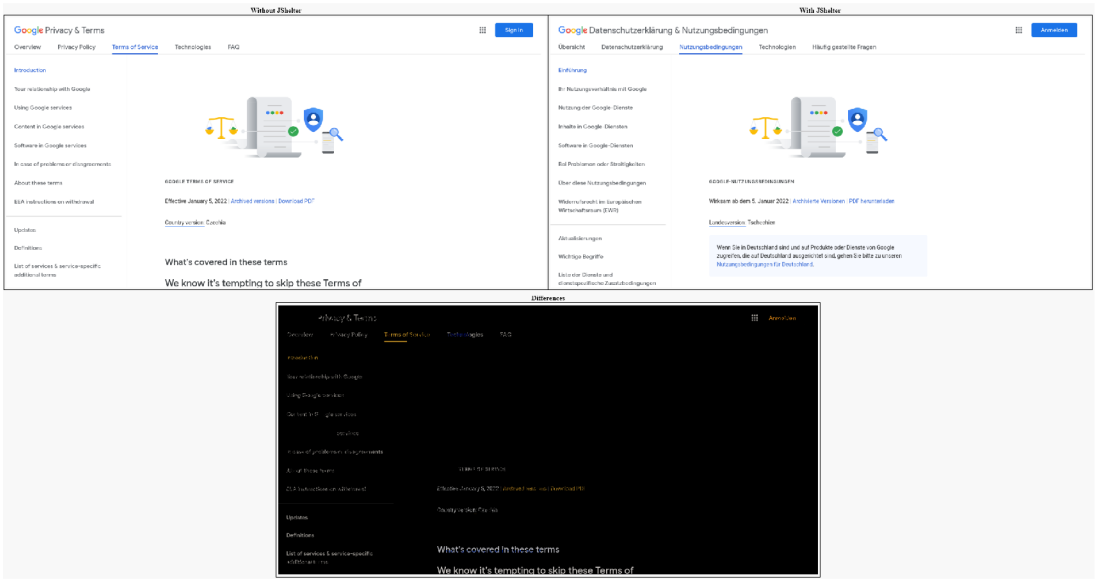
Po sérii testování bylo zjištěno, že algoritmus VIPS není příliš vhodný pro získávání dat z moderních webových stránek. Jak je možné vidět na obrázku 5.1d, algoritmus detekuje pouze jediný blok obsahu, který neobsahuje žádné informace. Z tohoto důvodu nebyl algoritmus VIPS použit pro získávání zajímavých podstránek a sekcí webové stránky.

5.4 Integrace systémových testů projektu JShelter

V rámci integrace systémových testů projektu JShelter do nástroje OpenWPM bylo vhodné integrovat pouze tu část systémových testů zajišťující analýzu získaných dat. Část obstarávající získávání dat nebylo smysluplné integrovat – OpenWPM obsahuje vlastní mechanismus pro získávání dat jako jsou například snímky obrazovky. Nevýhodou tohoto řešení však je, že není možné získat záznamy z konzole webového prohlížeče z důvodů popsaných v kapitole 5.1.1.

Integrována byla tedy pouze analýza snímků obrazovky. V rámci této integrace byly použity a upraveny původní zdrojové kódy Martina Bednáře využívající knihovnu OpenCV[6]. V rámci těchto systémových testů jsou porovnávány snímky obrazovky pořízené bez aktivního rozšíření JShelter a s aktivním rozšířením JShelter, následně je vypočtena průměrná hodnota pixelů rozdílového snímku. Tato hodnota je vypisována do záznamů nástroje OpenWPM a také do výstupu systémových testů.

Výstupem systémových testů je soubor ve formátu HTML zobrazující tři snímky: snímek obrazovky pořízený bez aktivního rozšíření JShelter, s aktivním rozšířením JShelter a rozdílový snímek snímků předchozích. Dále je zde zobrazena průměrná hodnota pixelů rozdílového snímku. Tento výstup je možné vidět na obrázku 5.3.



Obrázek 5.3: Výstup systémových testů rozšíření JShelter

[6] https://docs.opencv.org/4.x/index.html

5.5 Sestavení a modifikace Docker kontejneru

Jak bylo zmíněno v kapitole 5.1.1, celý mechanismus řízení procházení webových stránek využívá Docker kontejnerů. Je tedy vhodné popsat základní úkony pro práci s nimi, zejména sestavení a modifikace. Nástroj, aplikace či úloha běžící v kontejneru Docker je izolována od okolního prostředí a nelze do ní za jejího běhu příliš zasahovat. To přináší nevýhodu v podobě nutnosti znovu sestavit Docker kontejner i při velmi malé modifikaci.

Pro sestavení kontejneru Docker se využívá instrukcí obsažených ve zvláštním souboru, který obvykle nese název `Dockerfile`. Tento soubor obsahuje instrukce, tedy příkazy, jež jsou provedeny při sestavování, může se jednat například o instalaci dodatečných balíčků do kontejneru, vkládání souborů nebo provedení příkazů uvnitř kontejneru. Výsledkem procesu sestavení je soubor obsahující tzv. obraz kontejneru (angl. *image*). Tento obraz je po sestavení uložen v instanci nástroje Docker [6].

Pro modifikaci kontejneru Docker není nutné provádět kompletní proces sestavení. Je možné sestavit nový obraz kontejneru s využitím obrazu již existujícího – určeného k modifikaci. Tímto způsobem jsou vytvářeny tzv. vrstvy (angl. *layers*), každou modifikací (například změnou nebo přidáním souboru) vznikne jedna vrstva. Toto sestavení, jež provádí modifikaci, je prováděno také pomocí souboru `Dockefile`, ten však obsahuje pouze příkazy provádějící modifikace. Příkladem takového souboru `Dockerfile` může být například výpis 5.1.

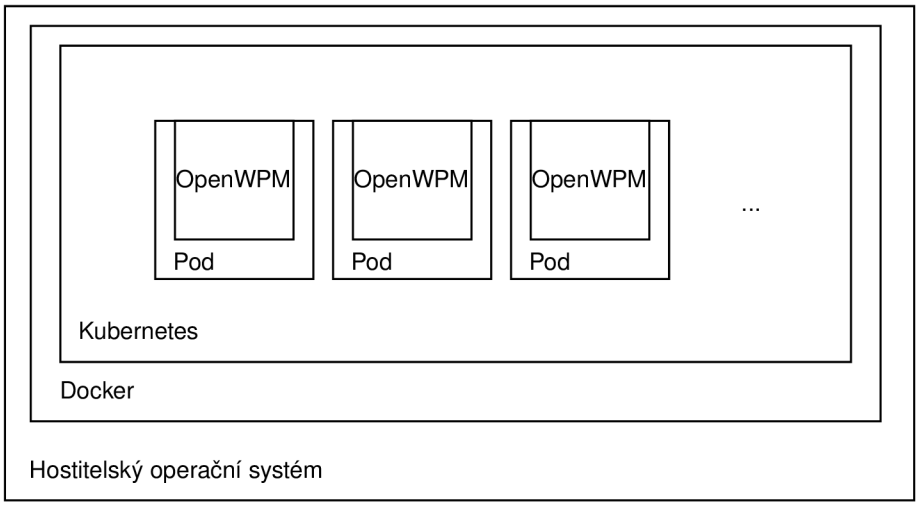
```
1  # Dockerfile for Docker image modification
2  FROM xkonec83/web_crawler:v1.0_to_be_modifficated
3
4  RUN modiffication_command
5  COPY modiffication_file.py /opt/OpenWPM/modiffication_file.py
6
```

Výpis 5.1: Příklad souboru `Dockerfile` pro modifikaci existujícího obrazu

Výhoda v tomto způsobu modifikace spočívá především v rychlosti samotného procesu modifikace, respektive sestavení obrazu kontejneru s využitím obrazu již existujícího.

Je-li obraz sestaven, je nutné jej exportovat z instance nástroje Docker běžící na hostitelském operačním systému do souboru ve formátu `tar`. Následně je nutné tento soubor přesunout do instance nástroje Kubernetes, který běží v Docker kontejneru. Zde je nutné obraz v podobě `tar` souboru importovat do zde přítomné instance nástroje Docker, která slouží jako zdroj Docker obrazů pro běh samotného nástroje Kubernetes. Funkcionalitu přesouvání souborů mezi hostitelským operačním systémem a Docker kontejnerem, ve němž běží nástroj Kubernetes, poskytuje nástroj minikube. V samotném nástroji OpenWPM běžícím v *podu* běží více instancí webového prohlížeče Firefox (ve výchozím stavu 3). Počet těchto instancí je možné škálovat.

Struktura celého provozu mechanismu řízení procházení webových stránek zobrazující také vrstvy tvořené nástrojem Docker je zobrazena níže na obrázku 5.4.



Obrázek 5.4: Struktura provozu mechanismu řízení procházení webových stránek

Při procesu sestavování jsou kladeny poměrně vysoké nároky na rychlost úložiště – zejména při exportu a importu obrazu. Při sestavování obrazu je vhodné, aby zařízení, na kterém je tento proces prováděn, disponovalo dostatečně rychlou internetovou konektivitou. Obraz kontejneru Docker, v němž běží nástroj OpenWPM, je založen na obrazu, který je při sestavování stažen z repozitáře Docker Hub[7].

Pro částečnou automatizaci byl vytvořen skript `build.sh`, který provede sestavení obrazu kontejneru Docker, jeho export a přesun do kontejneru v němž běží nástroj Kubernetes.

[7] https://hub.docker.com/

Kapitola 6

Analýza výsledků získaných implementovaným nástrojem

Získávání a analýza dat probíhala na virtuálním serveru běžícím v prostředí Proxmox[1]. Server disponoval parametry nacházejícími se v tabulce 6.1.

Operační systém	Procesor	Operační paměť	Úložiště
Ubuntu 22.04.2 LTS	4 x AMD	18 GB	120 GB

Tabulka 6.1: Parametry virtuálního serveru

Nástroj Kubernetes měl k dispozici pouze 16 GB RAM. Zbývající 2 GB RAM byly ponechány volné pro potřeby operačního systému a jeho služeb.

6.1 Výkonnost implementovaného nástroje

Cílem implementovaného nástroje je navštívení pokud možno největšího počtu webových stránek a jejich zajímavých podstránek v pokud možno nejmenším čase. Odkazy na webové stránky a jejich podstránky byly získány z žebříčku Tranco. Pro nalezení optimální konfigurace, která docílí optimální výkonnosti, bude nutné vhodně nastavit následující parametry:

- počet webových stránek a jejich podstránek určených k navštívení,
- počet *podů* vykonávajících samotné navštívení,
- počet instancí webového prohlížeče Firefox běžících v jednom *podu*.

Tyto parametry byly modifikovány v následujících experimentech, jejichž cílem bylo nalezení jejich optimálních hodnot. U experimentů budou sledovány a zaznamenávány následující parametry:

- doba potřebná k navštívení webových stránek a jejich analýze,
- vytížení procesoru,
- vytížení operační paměti.

[1] https://www.proxmox.com/en/

Experimenty poběží tak dlouho, dokud nebudou navštíveny a analyzovány všechny webové stránky a jejich podstránky, které byly nástroji zadány. Pro detekci zajímavých podstránek bylo využito klíčové slovo „cart“, byla očekávána detekce sekce košík na webových stránkách e-shopů.

6.1.1 Experiment 1

Cílem toho experimentu je získání výchozích hodnot, na základě kterých budou voleny parametry v dalších experimentech.

Webové stránky (a podstr.)	*Pody*	Instance Firefoxu
2 (15)	1	3

Tabulka 6.2: Parametry experimentu 1

Během experimentu byly zaznamenány následující hodnoty:

Doba běhu [*hh:mm:ss*]	Vytížení procesoru	Vytížení operační paměti	Zaj. podstr.
00:10:11	15 %	25 %	1

Tabulka 6.3: Výsledky experimentu 1

Z hodnot v tabulce 6.3 můžeme vidět, že navštívení a analýza poměrně malého množství webových stránek, respektive jejich podstránek zabere nástroji více než 10 minut. Můžeme si také povšimnout detekce jedné zajímavé podstránky.

6.1.2 Experiment 2

Tento experiment si klade za cíl ověřit, zdali povede navýšení počtu *podů* ke snížení doby potřebné pro navštívení a analýzu webových stránek. Zvoleny byly tyto parametry:

Webové stránky (a podstr.)	*Pody*	Instance Firefoxu
2 (15)	4	3

Tabulka 6.4: Parametry experimentu 2

Experimentem 2 bylo dosaženo těchto hodnot:

Doba běhu [*hh:mm:ss*]	Vytížení procesoru	Vytížení operační paměti	Zaj. podstr.
00:10:27	17 %	28 %	1

Tabulka 6.5: Výsledky experimentu 2

Ze získaných hodnot v tabulce 6.5 je patrné, že experiment 2 trval dokonce o 16 sekund déle než experiment 1. Důvodem je s největší pravděpodobností systém distribuce odkazů

na webové stránky z databáze Redis. Odkazy na webové stránky jsou distribuovány jednotlivým *podům*, respektive instancím webového prohlížeče, běžícím uvnitř *podu* včetně jejich podstránek. Při spuštění nástroje tedy došlo k téměř okamžité distribuci odkazů dvěma instancím webového prohlížeče v prvním *podu*. Ostatní *pody* zůstaly nevyužity. Doba běhu mírně vzrostla, pravděpodobně kvůli režii mezi více *pody*.

6.1.3 Experiment 3

Tento experiment bude sloužit jako reference k experimentu 4. Dojde k navýšení počtu webovách stránek. Z důvodu využití pouze jednoho *podu* se očekává poměrně dlouhá doba běhu.

Webové stránky (a podstr.)	*Pody*	Instance Firefoxu
10 (98)	1	3

Tabulka 6.6: Parametry experimentu 3

Experiment 3 poskytl tyto hodnoty:

Doba běhu [*hh:mm:ss*]	Vytížení procesoru	Vytížení operační paměti	Zaj. podstr.
00:55:23	31 %	16 %	2

Tabulka 6.7: Výsledky experimentu 3

Doba běhu dle očekávání vzrostla, její nárust však nebyl přímo úměrný zvýšení počtu webových stránek. Díky většímu vzorku webových stránek byly také nalezeny dvě zajímavé podstránky. Tyto výsledky budou použity pro porovnání v experimentu 4.

6.1.4 Experiment 4

V tomto experimentu bude navýšen počet *podů*, lze tedy očekávat nižší dobu potřebnou pro navštívení a analýzu všech webových stránek.

Webové stránky (a podstr.)	*Pody*	Instance Firefoxu
10 (98)	4	3

Tabulka 6.8: Parametry experimentu 4

Experiment 4 přinesl tyto výsledky:

Doba běhu [*hh:mm:ss*]	Vytížení procesoru	Vytížení operační paměti	Zaj. podstr.
00:41:33	82 %	51 %	2

Tabulka 6.9: Výsledky experimentu 4

Dle výsledků experimentu v tabulce 6.9 poklesla doba potřebná pro navštívení a analýzu webových stránek téměř o 14 minut. Poměrně znatelně však vzrostlo vytížení procesoru. Dále bylo zjištěno, že jeden ze čtyř *podů* nebyl využit, z tohoto důvodu bude v následujícím experimentu zvýšen počet webových stránek.

6.1.5 Experiment 5

Cílem tohoto experimentu je získání hodnot při procházení velkého množství webových stránek.

Webové stránky (a podstr.)	*Pody*	Instance Firefoxu
100 (≈ 950)	4	3

Tabulka 6.10: Parametry experimentu 5

V rámci experimentu 5 byly do tabulky 6.11 zaznamenány tyto hodnoty:

Doba běhu [*hh:mm:ss*]	Vytížení procesoru	Vytížení operační paměti	Zaj. podstr.
01:48:47	90 %	51 %	21

Tabulka 6.11: Výsledky experimentu 5

Prestože bylo navštíveno a analyzováno téměř desetkrát více webový stránek než v předchozím experimentu, doba potřebná pro tuto činnost vzrostla pouze přibližně 2,6 krát. Pozitivní vliv více *podů* na dobu běhu nástroje se tedy projevil až při velkém množství procházených webových stránek. Bylo také nalezeno 21 zajímavých podstránek.

6.1.6 Experiment 6

V tomto experimentu bude zvýšen počet instancí webového prohlížeče Firefox. Doba potřebná pro navštívení a analýzu webových stránek by tedy měla být kratší než v experimentu 5.

Webové stránky (a podstr.)	*Pody*	Instance Firefoxu
100 (≈ 950)	4	5

Tabulka 6.12: Parametry experimentu 6

Byly zaznamenány tyto hodnoty:

Doba běhu [*hh:mm:ss*]	Vytížení procesoru	Vytížení operační paměti	Zaj. podstr.
01:08:30[2]	100 %	79 %	21

Tabulka 6.13: Výsledky experimentu 5

[2]Experiment skončil neúspěchem.

Tento experiment skončil neúspěchem, jenž se projevil zastavením navštěvování a analýzy webových stránek. V době zastavení bylo již navštíveno a analyzováno více než 90% webových stránek. Příčinou neúspěchu tohoto experimentu byl s největší pravděpodobností nedostatečný výpočetní výkon serveru – procesor byl po celou dobu experimentu vytížen ze 100 %.

6.1.7 Shrnutí experimentů

Z výsledků těchto experimentů vyplývá, že využití více *podů* je smysluplné při větším počtu webových stránek určených k navštívení a analýze. V experimentu 3 popsaném v kapitole 6.1.3 byly při počtu 10 webových stránek efektivně využity 3 *pody*. Neefektivní využití *podů* působí na výkonnost nástroje spíše negativně, proto je nutné zvolit vhodný počet *podů* tak, aby byly všechny efektivně využity. Nejvhodnější hodnoty parametrů byly zvoleny v experimentu 5 v kapitole 6.1.5. V tomto experimentu bylo také dosaženo téměř maximálního vytížení procesoru serveru. I přesto byl proveden experiment 6, při kterém bylo dosaženo maximálního vytížení procesoru serveru. Tím bylo dosaženo limitu škálování tvořeného výpočetním výkonem serveru.

Obecně tedy platí, že zvyšování počtu *podů* je smysluplné pouze tehdy, je-li nutné navštívit a analyzovat větší množství webových stránek. S tímto navyšováním však vzrůstají požadavky na výpočetní výkon, zejména procesoru. Požadavky na výpočetní výkon tak mohou být poměrně vysoké.

Navštívením 100 webových stránek, respektive přibližně 950 jejich podstránek, bylo detekováno 21 zajímaných podstránek. Manuální kontrolou bylo zjištěno, že z těchto podstránek vedlo 7 z nich do sekce košíku. Úspěšnost detekce nebyla tedy nijak vysoká ($\approx 33\%$), určitý podíl zajímavých podstránek však detekován byl.

6.1.8 Činnost rozšíření

Po dokončení experimentů byly analyzovány výstupy systémových testů projektu JShelter a záznamy nástroje OpenWPM. Při analýze a pozorování záznamů nástroje OpenWPM bylo zjištěno, že při pořizování snímků celé obrazovky příkazem `ScreenshotFullPageCommand()` dojde k chybě, jež je způsobena neúspěšných spuštěním JavaScriptového kódu. Cílem tohoto kódu je navrátit hodnotu `window.innerHeight`, kvůli tomu nemůže být snímek celé obrazovky pořízen.

Dále bylo pozorováním snímků obrazovky zjištěno, že vzhled webových stránek s aktivním rozšířením JShelter i bez něj se zásadně neliší.

Analyzátor volaných webových API po celou dobu experimentů pracoval korektně. Své výstupy zapisoval do výstupních souborů formátu SQLite.

6.2 Nedostatky implementovaného nástroje

Jako nedostatek se ukázala nevhodnost algoritmu VIPS pro detekci zajímavých sekcí webové stránky. Algoritmus VIPS byl publikován v roce 2004 [9]. Segment webových stránek však od té doby prošel velmi intenzivním vývojem. Z tohoto důvodu již algoritmus VIPS nedosahuje při analýze moderních webových stránek uspokojivých výsledků.

Jako nedostatek implementovaného nástroje může být považována jeho „roztříštěnost“. Každý jeden *pod*, ve kterém nástroj OpenWPM běží, produkuje svá výstupní data. Běžel-li

nástroj s využitím čtyř *podů*, vznikly čtyři adresáře `datadir` nacházející se na příslušných persistentních jednotkách.

Poměrně velkým nedostatkem jsou vysoké nároky na výpočetní výkon a doba běhu nástroje. Navštívení a analýza velkého množství (stovek) webových stránek trvá několik hodin. V případě ještě většího množství se doba pohybuje v řádů dnů, potažmo týdnů nepřetržitého běhu. Z těchto důvodů není příliš vhodné provádět navštívení a analýzu webových stránek na PC nebo notebooku, je vhodnější využít server.

6.3 Budoucí vývoj

Vývoj implementovaného nástroje není zdaleka u konce. Budoucí vývoj by měl být zaměřen na odstranění jeho nedostatků a zlepšení jeho funkcionality.

Pro detekci zajímavých sekcí webové stránky by měl být zvolen vhodnější algoritmus než je algoritmus VIPS. Nabízí se zde možnost využití již existujících komplexních algoritmů založených na strojovém učení, případně je možné navrhnout algoritmus zcela nový. Tyto moderní algoritmy by měly dosahovat podstatně lepších výsledků než algoritmus VIPS.

Algoritmy založené na strojovém učení by mohly najít uplatnění také při analýze odkazů získaných z webové stránky, jež by mohly vést na zajímavé podstránky.

Prostor pro zlepšení je možné také pozorovat z hlediska výkonnosti. Z tohoto důvodu by bylo žádoucí optimalizovat nástroj OpenWPM. Díky velkému množství procházených a analyzovaných webových stránek by se i malá optimalizace mohla znatelně projevit na době běhu. V budoucnu je také žádoucí navýšit výpočetní výkon zařízení, na kterém nástroj poběží. Tímto navýšením bude snížena doba běhu nástroje. Bude také možné využít více *podů* a instancí prohlížeče Firefox díky vyššímu výpočetnímu výkonu.

V budoucnu by bylo vhodné implementovat do nástroje OpenWPM podporu webového prohlížeče Chrome. Tím by byl mj. vyřešen problém popsaný v kapitole 5.1.1. Díky této podpoře by bylo také možné provádět navštěvování a analýzu webových stránek pomocí webových prohlížečů s různými renderovacími jádry.

Díky dobře zvolenému konceptu rozšiřování funkcionality nástroje OpenWPM pomocí příkazů bude možné implementovat i další nové funkcionality různé povahy.

Využití Docker kontejnerů s sebou přináší možnost použít pro implementovaný nástroj také technologie jako jsou například Docker Swarm[3] nebo Nomad[4]. Existují také zpravidla placené cloudové služby, které poskytují nástroj Kubernetes. Tyto služby poskytují mj. společnosti Google, IBM nebo Microsoft.

[3] https://docs.docker.com/engine/swarm/
[4] https://www.nomadproject.io/

Kapitola 7

Závěr

V této práci byly představeny systémové testy projektu JShelter včetně jejich nedostatků a mechanismus procházení webových stránek Marka Schauera. Dále zde byly představeny metody analýzy webových stránek na různých principech, které budou použity k detekci zajímavých sekcí webových stránek. S pomocí těchto metod bylo navrženo vylepšení systémových testů, jež zároveň odstraní jejich nedostatky. Mezi tyto nedostatky lze zařadit fakt, že systémové testy duplikují mechanismus procházení stránek jiného nástroje. Dále také skutečnost, že mechanismus procházení webových stránek nefunguje na webových stránkách s dynamickým obsahem a že jsou analyzovány pouze titulní stránky. V neposlední řadě je nutné zmínit také špatnou a složitou konfiguraci samotných testů.

V této práci byl sjednocen mechanismus procházení webových stránek v rámci systémových testů a nástroj pro zjišťování používaných webových API. Vznikl tak univerzální nástroj, který může být využit i v jiných nástrojích souvisejících s projektem JShelter. Dále byl také implementován mechanismus výběru podstránek s ohledem na diverzitu navštívených stránek z pohledu očekávaného obsahu. Do tohoto nástroje byl také integrován algoritmus VIPS. Konfigurace systémových testů byla integrována do konfigurace nástroje OpenWPM, čímž vzniklo jednotné rozhraní pro konfiguraci nástroje jako celku.

Výsledky získané pomocí implementovaného nástroje byly analyzovány a vyhodnoceny. Byly představeny jeho nedostatky a navržena možná budoucí rozšíření a vylepšení tohoto nástroje.

Díky tomuto nástroji je možné automaticky procházet a analyzovat webové stránky. Tento nástroj také umožňuje detekci zajímavých podstránek. Data, která jsou získávána procházením a analýzou webových stránek, jsou využita k testování projektu JShelter pomocí systémových testů, jež jsou součástí implementovaného nástroje.

Literatura

[1] Bednář, M. *Automatické testování projektu JavaScript Restrictor*. Brno, CZ, 2020. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/22376/.

[2] Berners Lee, T. *Uniform Resource Identifier (URI): Generic Syntax* [Internet Requests for Comments]. RFC 3986. RFC Editor, leden 2005. Dostupné z: https://www.rfc-editor.org/rfc/rfc3986.

[3] Cai, D., Yu, S., Wen, J.-R. a Ma, W.-Y. Extracting Content Structure for Web Pages Based on Visual Representation. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, sv. 2642, s. 406–417. Lecture Notes in Computer Science. ISBN 9783540023548.

[4] Cao, Y., Li, S. a Wijmans, E. (Cross-)Browser Fingerprinting via OS and Hardware Level Features. In:. Leden 2017. DOI: 10.14722/ndss.2017.23152.

[5] Dalbey, J. *Nonfunctional Requirements* [online]. [cit. 2022-12-27]. Dostupné z: http://users.csc.calpoly.edu/~jdalbey/SWE/QA/nonfunctional.html.

[6] Docker Inc.. *Dockerfile reference* [online]. [cit. 2023-05-01]. Dostupné z: https://docs.docker.com/engine/reference/builder/.

[7] Docker Inc.. *What is a Contianer?* [online]. [cit. 2023-04-24]. Dostupné z: https://www.docker.com/resources/what-container/.

[8] Englehardt, S. a Narayanan, A. Online tracking: A 1-million-site measurement and analysis. In: *Proceedings of ACM CCS 2016*. 2016.

[9] Fu, L., Meng, Y., Xia, Y. a Yu, H. Web Content Extraction based on Webpage Layout Analysis. In: *2010 Second International Conference on Information Technology and Computer Science*. IEEE, 2010, s. 40–43. ISBN 9781424472932.

[10] Haldar, R. a Mukhopadhyay, D. Levenshtein Distance Technique in Dictionary Lookup Methods: An Improved Approach. 2011.

[11] Kolektiv autorů. *Getting started with Selenium Grid* [online]. Revidováno 1.7.2021 [cit. 2022-12-28]. Dostupné z: https://www.selenium.dev/documentation/grid/getting_started/.

[12] Kolektiv autorů. *HTML links* [online]. [cit. 2022-12-31]. Dostupné z: https://www.w3schools.com/html/html_links.asp.

[13] Kolektiv autorů. *Introduction to the DOM* [online]. [cit. 2022-12-29]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction.

[14] Kolektiv autorů. *Measurement platform* [online]. [cit. 2023-01-02]. Dostupné z: https://jshelter.org/pt/crawling/crawling-architecture.png.

[15] Kolektiv autorů. *Pods* [online]. [cit. 2023-01-01]. Dostupné z: https://kubernetes.io/docs/concepts/workloads/pods/.

[16] Kolektiv autorů. *Proxy* [online]. [cit. 2022-12-28]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Proxy.

[17] Masanès, J. *Web Archiving*. 2006th edition. Springer, 2006. ISBN 3540233385.

[18] Popela, T. *Implementace algoritmu pro vizuální segmentaci www stránek*. Brno, CZ, 2012. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/14163/.

[19] Richardson, L. *Beautiful Soup* [online]. [cit. 2023-01-02]. Dostupné z: https://www.crummy.com/software/BeautifulSoup/.

[20] Schauer, M. *Oblíbenost JavaScriptových API internetového prohlížeče*. Brno, CZ, 2021. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/23312/.

[21] United States Government US Army. *Systems Engineering Fundamentals*. DAU PRESS, 2001. ISBN 978-1484120835.

[22] Vaidya, N. *All You Need to Know About Selenium WebDriver Architecture* [online]. [cit. 2022-12-28]. Dostupné z: https://www.edureka.co/blog/selenium-webdriver-architecture/.

[23] Wada, H., Suzuki, J. a Oba, K. Modeling Non-Functional Aspects in Service Oriented Architecture. In: *2006 IEEE International Conference on Services Computing (SCC'06)*. 2006, s. 222–229. DOI: 10.1109/SCC.2006.74. ISBN 0-7695-2670-5.

Příloha A

Obsah přiloženého paměťového média

```
/
├── crawler/ ... zdrojové soubory implementovaného nástroje
│   ├── Extension/ ... zdrojové soubory analyzátoru volaných webových API
│   ├── scripts/ ... skripty využívané pro sestavení Docker obrazu
│   ├── openwpm/ ... zdrojové soubory nástroje OpenWPM
│   ├── k8s/ ... konfigurační soubory nástroje Kubernetes
│   ├── *.py ... zdrojové soubory crawleru a algoritmu VIPS
│   └── IMAGE_BUILD.md ... návod pro sestavení, nasazení a spuštění
│       implementovaného nástroje
├── tex/ ... zdrojové soubory technické zprávy ve formátu LaTeX
└── xkonec83_bp.pdf ... tato technická zpráva ve formátu PDF
```