VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

OBLÍBENOST JAVASCRIPTOVÝCH API INTERNETOVÉHO PROHLÍŽEČE

ON POPULARITY OF WEB BROWSER JAVASCRIPT APIS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. MAREK SCHAUER**
AUTHOR

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

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2020/2021

Zadání diplomové práce

Student: **Schauer Marek, Bc.**
Program: Informační technologie a umělá inteligence
Specializace: Informační systémy
Název: **Oblíbenost JavaScriptových API internetového prohlížeče**
On Popularity of Web Browser JavaScript APIs
Kategorie: Web
Zadání:

1. Nastudujte literaturu týkající se rozšíření používání jednotlivých API klientského JavaScriptu, zaměřte se na výzkum Petera Snydera.
2. Seznamte se s nástroji, které se používají pro analýzu webu, zaměřte se na nástroje použitelné pro detekci využívaných API. Prostudujte možnosti webového rozšíření Web API Manager.
3. Navrhněte testovací prostředí, která vám umožní navázat na výzkum z článku Browser Feature Usage on the Modern Web. Zaměřte se také na API přidaná do prohlížečů po uskutečnění původního výzkumu.
4. Implementujte testovací prostředí a dokumentujte jeho použití.
5. Za využití implementovaného prostředí navažte na výzkum z článku Browser Feature Usage on the Modern Web.
6. Vyhodnoťte výsledky práce a navrhněte možnosti pokračování.

Literatura:

- Peter Snyder, Cynthia Taylor a Chris Kanich, "Most Websites Don't Need to Vibrate: A Cost-Benefit Approach to Improving Browser Security," in Proceedings of the 2017 ACM Conference on Computer and Communications Security, str. 179-194, 2017.
- Peter Snyder, Lara Ansari, Cynthia Taylor a Chris Kanich, "Browser Feature Usage on the Modern Web," in Proceedings of the 2016 Internet Measurement Conference, str. 97-110, 2016.
- Pierre Laperdrix, Nataliia Bielova, Benoit Baudry a Gildas Avoine. 2020.Browser fingerprinting: A survey. ACM Trans. Web, roč. 14, č. 2, str. 8:1-8:33.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3 zadání.

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. listopadu 2020
Datum odevzdání: 19. května 2021
Datum schválení: 26. října 2020

Abstrakt

V tejto práci prezentujeme návrh a implementáciu platformy pre automatizované meranie používania JavaScriptových API vo webovom prehliadači. Táto platforma je založená na OpenWPM, ktoré slúži na inštrumentáciu webového prehliadača. V našej architektúre je prehliadač obohatený o upravené rozšírenie Web API Manager, ktoré umožňuje zachytávať volania jednotlivých JavaScriptových API a zaznamenávať o príslušných volaniach informácie. Uvedená platforma bola použitá pre vykonanie meraní na počte 10000 webových stránok. Z analýzy dát získaných meraním sme zistili, že na navštívených webových stránkach sú najpoužívanejšie API pre prácu s HTML, DOM, High Resolution Time API a Web Cryptography API. V rámci API, ktoré boli v prehliadači Mozilla Firefox implementované po roku 2016, sme ako najčastejšie používané identifikovali Intersection Observer API, Background Tasks API a Resize Observer API.

Abstract

In this work we present the design and implementation of a platform for automated measurement of the use of JavaScript APIs in a web browser. This platform is based on OpenWPM, which is used to instrument the web browser. In our architecture, the browser is extended with a modified Web API Manager extension, which allows to capture calls to JavaScript methods and log information about these calls. The platform was used to perform measurements on a 10,000 websites. From the analysis of the data obtained by the measurement, we found that the most used APIs over measured websites are APIs specified in HTML and DOM standards, High Resolution Time API and Web Cryptography API. Within the APIs that were implemented in Mozilla Firefox after 2016, we identified the Intersection Observer API, Background Tasks API and Resize Observer API as the most frequently used.

Kľúčové slová

JavaScript, ECMAScript, Web API, Meranie webu, OpenWPM, Prehliadač

Keywords

JavaScript, ECMAScript, Web API, Web measurement, OpenWPM, Browser

Citácia

SCHAUER, Marek. *Oblíbenost JavaScriptových API internetového prohlížeče*. Brno, 2021. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Libor Polčák, Ph.D.

Oblíbenost JavaScriptových API internetového prohlížeče

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením pána Ing. Libora Polčáka, Ph.D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som pri písaní tejto práce čerpal.

.....
Marek Schauer
18. mája 2021

Poďakovanie

Za cenné rady, usmernenia a informácie počas celého roka by som veľmi rád poďakoval pánovi Ing. Liborovi Polčákovi, Ph.D., ktorý mi poskytoval svoje odborné vedenie.

Obsah

Kapitola 1

Úvod

S príchodom internetu, ktorý je dnes dostupný v podstate zo všetkých kútov sveta, vzniklo nespočetné množstvo nových služieb a aplikácií, ktoré sú postavené na rozšírení internetu. S internetom sa nesporne spájajú aj webové technológie, ktoré dnes predstavujú veľmi komplexný systém rôznych technológií, ktoré spolu tvoria infraštruktúru pre fungovanie pomerne veľkého množstva webových stránok a aplikácií. Medzi základné stavebné kamene infraštruktúry zabezpečujúcej fungovanie webových stránok a aplikácií môžeme bezpochyby zaradiť napríklad webové servery, ktoré vykonávajú celý rad rôznych dynamických výpočtov a servírujú webové stránky klientom, alebo službu DNS, ktorá prekladá jednoducho zapamätateľné doménové mená na IP adresy. Ďalej môžeme do tejto infraštruktúry určite začleniť aj webové prehliadače, ktoré tvoria prostredie a rozhranie pre zobrazovanie webových stránok užívateľom.

V tejto práci sa zameriame na webové prehliadače, konkrétne na tú časť webových prehliadačov, ktorá ponúka prostredie pre beh jazyka JavaScript. JavaScript, ktorý je dnes v rámci prehliadača spravidla obohatený o rôzne nadstavby základnej množiny jazyka, nemusí byť použitý len pre prácu so štruktúrou webovej stránky pomocou DOM[1] alebo pre animácie. S príchodom rôznych nových možností, ktoré sú častokrát štandardizované, ponúka dnešný JavaScript vo webovom prehliadači veľké množstvo vstavaných objektov a funkcií, vďaka ktorým je napríklad možné komunikovať s webovým serverom, používať lokálnu databázu priamo v prehliadači, zisťovať polohu užívateľa alebo reagovať na hodnoty namerané rôznymi senzormi užívateľovho zariadenia. V rámci tejto práce sme identifikovali 38 nových JavaScriptových API, do ktorých sme kategorizovali 211 metód.

Cieľom tejto práce je zhrnúť poznatky z oblasti používania rôznych funkcionalít jazyka JavaScript vo webovom prehliadači. S pribúdajúcim počtom nových možností jazyka JavaScript môže byť užitočné vedieť, či a ako často sú tieto možnosti v praxi využívané, prípadne, ktoré API jazyka JavaScript vo webovom prehliadačí sú využívané viac a ktoré menej. S cieľom zistenia odpovedí na predošlé otázky sme v rámci tejto práce navrhli prostredie pre automatizované meranie, v rámci ktorého bude umožnené zmerať používanie JavaScriptových API vo webovom prehliadači danými webovými stránkami. Navrhnuté prostredie bolo implementované a použité pre návštevu 10000 webových stránok, na ktorých sme zbierali dáta o používaní JavaScriptu v bežne používanom prehliadači.

V kapitole 2 najprv uvádzame stručnú históriu vzniku a štandardizácie jazyka JavaScript, pričom tiež uvádzame, akým spôsobom vznikajú JavaScriptové Web API a kto ich

[1] *Document Object Model*, rozhranie, ktoré umožňuje pristupovať k častiam dokumentu ako k stromovej štruktúre

štandardizuje. Ďalej sa v časti 2.4 zameriavame na zhrnutie existujúceho výskumu v oblasti používania JavaScriptu a JavaScriptových API na internete. V kapitole 3 potom popisujeme návrh prostredia pre automatizovanú návštevu webových stránok, ktoré je založené na platforme OpenWPM pre automatizovanú inštrumentáciu webového prehliadača. V kapitole 3 sa tiež zaoberáme výberom vhodného doplnku do prehliadača, pomocou ktorého môžeme realizovať meranie dát a tiež problematikou získania informácií o tom, aké možnosti daný prehliadač ponúka a prečo je to pre našu prácu dôležité. V kapitole 4 sa zaoberáme implementáciou uvedenej platformy, ktorá bola využitá na získanie dát. Nakoniec, v kapitole 5 uvádzame informácie o používaní JvaSctiptových API na 10000 webových stránok.

Kapitola 2

JavaScript

JavaScript je programovací jazyk, ktorý je známy najmä v spojení s webovými technológiami a skriptovaním vo webových prehliadačoch. S príchodom technológií ako Node.js sa však JavaScript stáva jazykom, ktorý už nie je len výsadou front-endu webových aplikácií, ale čoraz častejšie môžeme JavaScript nájsť aj na serverovej časti technologického zásobníka rôznych služieb, čo podľa Davida Flanagana, autora knihy JavaScript: The Definitive Guide, robí z JavaScriptu najpoužívanejší jazyk medzi softvérovými vývojármi [14].

2.1 Vývoj a štandardizácia jazyka JavaScript

Prvá verzia JavaScriptu vznikla pod záštitou spoločnosti Netscape, ktorá v roku 1995 najala Brendana Eicha, ktorého úlohou bolo implementovať jazyk Scheme do prehliadača Navigator, ktorý bol v rámci Netscape vyvíjaný. Neskôr sa však na úrovni spoločnosti Netscape učinilo rozhodnutie vytvoriť nový skriptovací jazyk, ktorý mal byť inšpirovaný syntaxou jazyka Java. Tento jazyk sa vo svojej prvej verzii, ktorá bola vydaná v rámci prehliadača Navigator, volal LiveScript. Tento názov sa o niekoľko mesiacov neskôr zmenil na JavaScript.

Názov jazyka JavaScript vyvolal zmätenie, pretože by sa mohlo zdať, že JavaScript je odnožou jazyka Java, ktorý bol v čase vzniku JavaScriptu pomerne populárny. Možno práve preto bola syntax jazyka JavaScript inšpirovaná Javou, s ktorou však okrem podobnosti v názve, podobnou syntaxou a konceptu primitívnych hodnôt a objektov nemá JavaScript takmer nič spoločné [36].

V roku 1996 uviedol Microsoft tretiu verziu svojho webového prehliadača Internet Explorer, v rámci ktorého Microsoft zabudoval interpret vlastnej verzie jazyka JavaScript, ktorá niesla názov JScript. JScript vychádzal z jazyka JavaScript, ktorý bol v rámci Microsoftu preskúmaný reverzným inžinierstvom a naprogramovaný vlastným spôsobom. Implementácia JScriptu bola však pomerne odlišná od implementácie JavaScriptu v rámci prehliadača Navigator, čo viedlo k tomu, že vývojári webových stránok a aplikácií si museli častokrát vyberať, pre ktorý prehliadač svoju stránku alebo aplikáciu optimalizujú. Odlišnosti medzi implementáciami JScriptu a JavaScriptu boli triviálne (napríklad rozličné pomenovanie metód a vlastností, ktoré ponúkali rovnakú funkcionalitu), ale aj pomerne fundamentálne a principiálne (napr. inverzný model delegovania udalostí) [40].

2.1.1 ECMAScript

V roku 1996 sa spoločnosť Netscape rozhodla štandardizovať svoj jazyk JavaScript v spolupráci s organizáciou ECMA (*European Computer Manufacturers Association*). Pri štandardizácii jazyka však nemohol byť použitý názov JavaScript, pretože práva na používanie tohto názvu vlastnila spoločnosť Sun (dnes Oracle) a pre účely štandardizácie jazyka bolo tým pádom nutné vytvoriť pre jazyk nový názov. Spojením slov ECMA a JavaScript vznikol názov EcmaScript, ktorý sa dnes používa najmä pri odkazovaní na rôzne verzie štandardu jazyka, avšak vo všeobecnosti sa stále používa názov JavaScript. V rámci tejto práce sa budeme držať názvu JavaScript a názov EcmaScript, prípadne skratku ES budeme používať len v súvislosti s presnými verziami štandardov jazyka.

Jazyk EcmaScript je špecifikovaný štandardom, ktorý nesie názov ECMA-262. Tento štandard je dodnes udržiavaný a k dnešnému dátumu (25.9.2020) bola vydaná už jeho jedenásta verzia. Štandard ECMA-262 je spravovaný skupinou TC39 (*Technical Committee 39*). Členmi skupiny TC39 sú rôzne spoločnosti, napríklad Mozilla Foundation, Apple, Facebook, Google alebo Microsoft [10].

V tabuľke 2.1 môžeme vidieť prehľad verzií štandardu jazyka EcmaScript aj s časovými údajmi, kedy boli jednotlivé štandardy vydané. Z tabuľky môžeme vidieť, že nové verzie jazyka spočiatku vychádzali pomerne nepravidelne, avšak od verzie 6 vychádzajú nové verzie spravidla každý rok.

V tabuľke 2.1 môžeme tiež vidieť, že verzia 4 nebola nikdy vydaná. Dôvodom bolo, že skupina TC39 sa nezhodla na štandarde, ktorý by mal novú verziu jazyka štandardizovať, pretože do jazyka prinášal príliš veľa zásadných zmien, pri ktorých sa členom TC39 nepodarilo prísť ku spoločnému konsenzu. Nakoniec sa namiesto prijatia štvrtej verzie štandardu podarilo skupine TC39 prijať rozhodnutie, v rámci ktorého sa členovia TC39 zhodli na vydaní inkrementálnej aktualizácie predošlej verzie EcmaScriptu 3, neskôr vydaného ako ECMAScript 5. Zároveň bolo ohlásené, že neskôr bude vyvinutá nová verzia jazyka, ktorá bude obsahovať menej zmien, ako ECMAScript 4, avšak omnoho viac ako ECMAScript 5 a bude rozdelená do dvoch verzií, ktoré budú postupne vydané ako ECMAScript 6 a ECMAScript 7.

Verzia	Názov verzie	Vydanie
1	ECMAScript 1	Jún 1997
2	ECMAScript 2	August 1998
3	ECMAScript 3	December 1999
4	ECMAScript 4	Nevydané
5	ECMAScript 5	December 2009
5.1	ECMAScript 5.1	Jún 2011
6	ECMAScript 2015	Jún 2015
7	ECMAScript 2016	Jún 2016
8	ECMAScript 2017	Jún 2017
9	ECMAScript 2018	Jún 2018
10	ECMAScript 2019	Jún 2019
11	ECMAScript 2020	Jún 2020

Tabuľka 2.1: Verzie EcmaScriptu v poradí, v akom boli vydané

2.2 Web API

Štandardy, ktoré sú sformulované a špecifikované skupinou TC39 a organizáciou EcmaScript štandardizujú všeobecnú podobu jazyka JavaScript, resp. EcmaScript. Je dôležité uviesť, že v rámci jednotlivých štandardov JavaScriptu vydaných organizáciou ECMA je špecifikovaná len štandardná knižnica jazyka. Štandardná knižnica jazyka ponúka rozhranie, ktoré umožňuje prácu nad štruktúrami dostupnými v každom prostredí, v ktorom môže JavaScript bežať, ako napríklad prácu s objektami, poliami, dátumom či regulárnymi výrazmi. Príkladom rozhrania, ktoré nie je poskytnuté v štandardnej knižnici JavaScriptu je napríklad rozhranie objektu DOM (*Document Object Model*), ktorý reprezentuje HTML dokument vykreslený v okne prehliadača a ktorý je pre skriptovanie v prehliadači pomerne dôležitý a esenciálny.

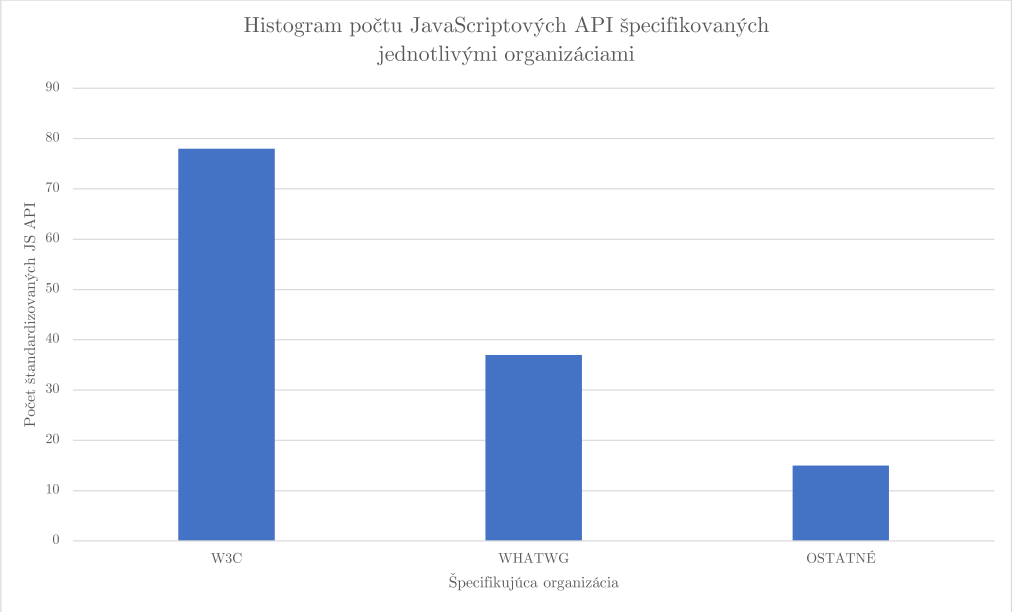
Veľké množstvo rozhraní, ktoré nie sú súčasťou štandardnej knižnice JavaScriptu je špecifikovaných inými organizáciami, než ECMA. Príkladmi takých organizácií sú napríklad organizácie W3C (*World Wide Web Consortium*) alebo WHATWG (*Web Hypertext Application Technology Working Group*). Ide o organizácie, ktoré sú autormi rôznych webových štandardov, ktoré môžu tvorcovia prehliadačov implementovať do svojich prehliadačov a ktoré môžu následne vývojári webových stránok a aplikácií využiť pri ich vývoji. Tieto štandardy sú často označované ako Web API. Je dôležité podotknúť, že medzi Web API patria aj štandardy, ktoré nie sú rozšíreniami JavaScriptu, ale určujú napríklad štruktúru a vlastnosti jazyka CSS alebo značkovacieho jazyka HTML. V tejto práci sa budeme v súvislosti s Web API zaoberať najmä tými štandardmi, ktoré rozširujú jazyk JavaScript.

Autoritatívna dokumentácia jednotlivých Web API má formu špecifikácie pre programátorov, ktorí jednotlivé Web API implementujú v prehliadačoch, tj. vo väčšine C++ programátorov. Rolu dokumentácie pre používateľov Web API, tj. pre webových vývojárov, spĺňa platforma MDN spoločnosti Mozilla[1], ktorá tieto Web API dokumentuje vo forme vhodnej pre programátorov využívajúcich Web API pri vývoji webových stránok a aplikácií.

V čase písania tejto práce existuje pomerne veľké množstvo rôznych webových štandardov, ktoré pri programovaní webových stránok ponúkajú vývojárom veľa možností. Jedným z najdôležitejších Web API je už spomenutý DOM, ktorého autorom je v jeho pôvodnej verzii organizácia W3C [18], avšak jeho aktuálnu podobu definujú štandardy, ktoré vydáva WHATWG [47]. Medzi ďalšie rozhrania, ktoré môžeme považovať za súčasť Web API patrí rozhranie objektu XMLHttpRequest, pomocou ktorého môže prehliadač komunikovať so serverom. Autorom špecifikácie rozhrania objektu XMLHttpRequest je WHATWG [49].

Na obrázku 2.1 môžeme vidieť histogram počtu JavaScriptových API, ktoré boli špecifikované rôznymi organizáciami. Dáta pre tento obrázok pochádzajú z portálu caniuse.com [8]. Z týchto dát vyplýva, že najvyšší počet špecifikácií vytvorila organizácia W3C, ktorá je autorom 78 špecifikácií, pričom z dielne organizácie WHATWG pochádza 37 štandardov. Na obrázku 2.1 tiež môžeme vidieť, že 15 zo skúmaných štandardov pochádza od ostatných organizácií. Medzi tieto ostatné organizácie spadá napríklad skupina WICG (*Web Platform Incubator Community Group*), ktorá vznikla pod záštitou W3C a združuje vývojárov, ktorí nie sú súčasťou niektorej zo štandardizačných W3C skupín, avšak zaujímajú sa o štandardizáciu v oblasti webových technológií. WICG poskytuje týmto vývojárom platformu pre špecifikáciu nových štandardov. Medzi ďalšie organizácie, ktoré spadajú do skupiny ostatných tvorcov špecifikácií patrí napríklad FIDO Alliance, ktorá špecifikovala FIDO U2F API pre prácu s Universal Second Factor zariadeniami [9].

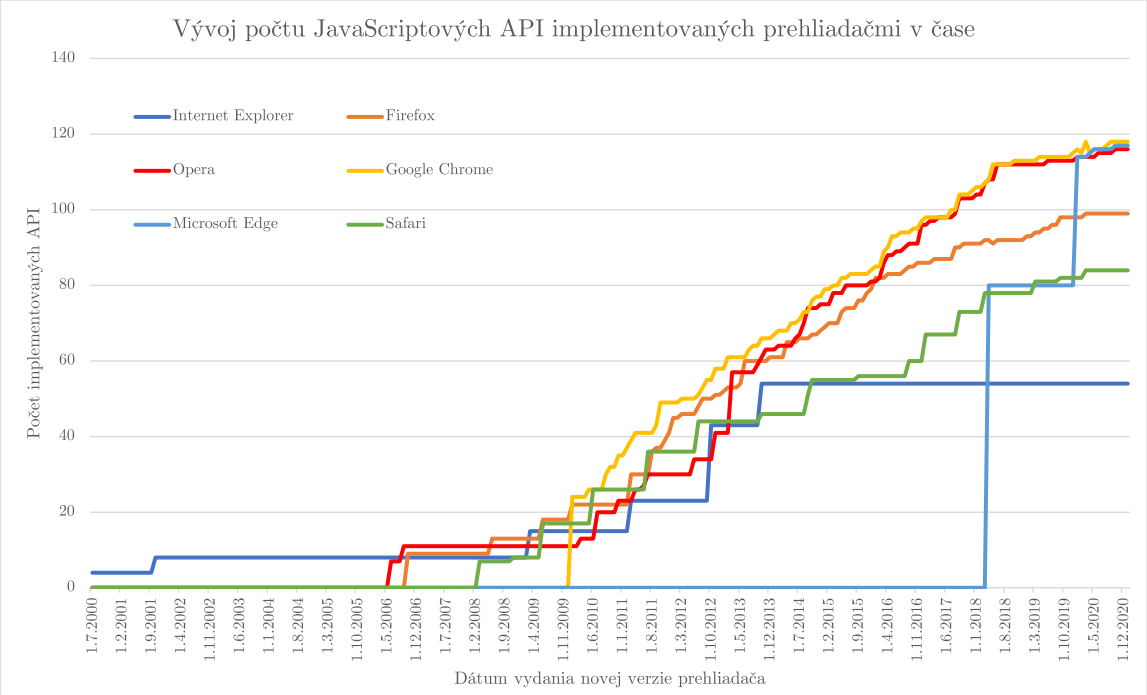
[1] https://developer.mozilla.org/docs/Web/API



Obr. 2.1: Histogram počtu JavaScriptových API špecifikovaných jednotlivými organizáciami, dáta pochádzajú z portálu caniuse.com [8]

Vďaka tomu, že ponuka webových prehliadačov je pomerne široká, existuje viacero implementácií jednotlivých Web API v závislosti od prehliadača. Na obrázku 2.2 môžeme vidieť časový vývoj počtu JavaScriptových API implementovaných na úrovni jednotlivých najpoužívanejších a najrozšírenejších prehliadačov. Ďalej, v prílohách A.1, A.2, A.3 a A.4 môžeme vidieť časové osy, ktoré vyjadrujú, kedy boli v štyroch vybraných prehliadačoch (konkrétne Google Chrome, Firefox, Opera a Safari) implementované jednotlivé API, pričom tieto obrázky zachytávajú API implementované v týchto prehliadačoch od roku 2016. Dáta, ktoré boli použité pri tvorbe obrázka 2.2 a príloh A.1, A.2, A.3 a A.4 pochádzajú z portálu caniuse.com[2]. Rastúci charakter vývoja počtu implementovaných JavaScriptových API, ktorý môžeme vidieť na obrázku 2.2 napovedá, že odstraňovanie už implementovaných API nie je bežné. Aj napriek tomu sa v prehliadačoch vyskytli niektoré API, ktoré boli neskôr odstránené. Príkladom takého API môže byť napríklad Battery Status API [28], ktoré bolo odstránené z prehliadača Mozilla Firefox od verzie 52. Zo strany tvorcov prehliadača išlo vo všeobecnosti o bezprecedentný krok [35]. Uvedené API bolo odstránené z dôvodu príliš častého využívania daného API pre iné účely, než pre tie účely, pre ktoré bolo pôvodne navrhnuté. Zo štúdie vypracovanej autormi Englehardt a Narayanan vyplýva, že Battery Status API bolo (resp. v časti prehliadačov ešte stále je) využívané ako jedna z viacerých vlastností užívateľa, ktoré boli následne použité pre identifikáciu užívateľov webových stránok [12].

[2]https://github.com/fyrd/caniuse



Obr. 2.2: Vývoj počtu JavaScriptových API implementovaných rôznymi webovými prehliadačmi v čase, dáta pochádzajú z portálu caniuse.com

2.2.1 Zneužívanie JavaScriptových API

S rastúcim množstvom možností jazyka JavaScript štandardizovaného organizáciou ECMA rástol dopyt vývojárov webových stránok na to, aby bol JavaScript akýmsi mostom medzi webovým prehliadačom a operačným systémom, na ktorom je príslušný prehliadač spustený. Tento dopyt bol spôsobený vidinou priblíženia užívateľskej skúsenosti pri používaní webových aplikácií tej skúsenosti, ktorú majú užívatelia spojenú s používaním natívnych aplikácií [21]. Takúto tendenciu môžeme vidieť už v rámci prvej verzie jazyka ECMAScript, v ktorej bol uvedený objekt `Date` ponúkajúci rozhranie sprístupňujúce informácie o čase, dátume a časovej zóne užívateľovho počítača. Pri navrhovaní takýchto API, či už sú súčasťou vstavanej knižnice JavaScriptu alebo ide o Web API, je dôležité, aby autori špecifikácií týchto API brali na vedomie, že JavaScriptové API nemusia byť vždy použité len pre tie účely, pre ktoré boli pôvodne navrhnuté a následne implementované tvorcami prehliadačov.

Využívanie rôznych API pre iné účely než tie, pre ktoré boli pôvodne navrhnuté je pomerne bežné, najmä v spojení s ich používaním pre účely identifikácie a sledovania užívateľov webových stránok. Pre tieto účely sa spravidla využíva kombinácia viacerých API, pomocou ktorých sa dá odhaliť pomerne vysoká unikátnosť užívateľov [11, 17].

Pravdepodobne najznámejším príkladom API, ktoré je často spájané s používaním pre iný účel, než ten pôvodný je Battery Status API, ktoré bolo spomenuté vyššie. Pôvodným a plánovaným spôsobom použitia tohto API je možnosť tvorcov webových stránok reagovať na vybíjanie batérie zariadenia, napríklad častejším posielaním formulárových hodnôt z prehliadača na server, aby sa minimalizovala šanca vybitia užívateľovho zariadenia pri neuloženej práci. Výskumy však naznačujú, že Battery Status API je relatívne často využívané pre identifikáciu užívateľov [17, 35]. K tomuto účelu sa využíva pomerne á informácia o stupni nabitia užívateľovho zariadenia, ktorú prehliadač priamo vystavuje príslušnému

API. Pomocou tejto informácie je potom možné identifikovať užívateľa, ktorý v danom momente, prípadne v istom krátkom časovom okne navštívi dve rôzne webové stránky z toho istého zariadenia [34].

Ďalším príkladom API, ktoré môže byť zneužité pre identifikáciu užívateľov je Audio API [32]. Audio API bolo navrhnuté pre prácu so zvukom na webových stránkach tak, aby bol tvorcom stránok umožnený výber z rôznych zdrojov zvuku či pridávanie zvukových alebo priestorových efektov. Súvislosť využívania tohto API pre účely identifikácie užívateľov popísal Steven Englehardt vo svojej práci Automated discovery of privacy violations on the web [11]. Základná myšlienka pri využívaní tohto API pre identifikáciu užívateľov je založená na tom, že výsledné zvukové signály spracované rôznymi počítačmi a prehliadačmi môžu byť mierne rozdielne, pričom signály spracované tou istou kombináciou počítača a prehliadača budú zaručene rovnaké.

Podobná základná myšlienka ako pri identifikácii užívateľov pomocou Audio API môže byť použitá aj pri identifikácii užívateľov pomocou Canvas API [29]. Canvas API umožňuje tvorcom webových stránok kresliť rôzne 2D obrazce na webových stránkach pomocou JavaScriptu a elementu `<canvas>`. Podobne ako pri Audio API, pri procese vykresľovania (po angl. *rendering*), potlačenia rozmazania (po angl. *antialiasing*) alebo vyhladzovania (po angl. *smoothing*) môžu byť do vykresleného výsledku vnesené rôzne indície o vlastnostiach užívateľovho systému [34]. Autori štúdie Pixel Perfect: Fingerprinting Canvas in HTML5, Keaton Mowery a Hovav Shacham, v ktorej bolo poukázané na možné zneužitie Canvas API, deklarujú, že relatívne vysokú identifikáciu užívateľa je možné pomocou Canvas API zaistiť už v zlomku sekundy bez vedomia užívateľa. Konkrétne, pri identifikácii užívateľa autori štúdie používajú techniku vykresľovania rôznych fontov a scén s použitím WebGL a následnú analýzu pixelov vykreslených do elementu `<canvas>`.

Medzi API používané aj pre iné účely, než tie pôvodné, môžeme zaradiť aj Permissions API [30], ktoré umožňuje tvorcom webových stránok získavať informácie o prístupe k jednotlivým API prehliadača, resp. informácie o povoleniach, ktoré boli rôznym API v prehliadači udelené. Kolektív autorov Iqbal, Englehardt a Shafiq vo svojom výskume ukazuje, že rôzne skripty pre identifikáciu užívateľov snímajú informácie o stave a povoleniach rôznych API, napríklad Notification API, Geolocation API a Camera API [17].

Súčasťou procesu identifikácie užívateľov môže byť aj použitie API pre prácu so senzormi [7]. Z práce s názvom The Web's Sixth Sense: A Study of Scripts Accessing Smartphone Sensors od kolektívu autorov Das, Acar, Borisov a Pradeep, v rámci ktorej skúmali používanie JavaScriptových API slúžiacich pre prácu so senzormi na mobilných zariadeniach vyplýva, že väčšina použití takýchto API nie je v súlade s použitím, pre ktoré sú tieto API určené. Uvedená štúdia ukazuje, že aj napriek tomu, že niektoré skripty pristupovali k týmto API za účelom zlepšenia užívateľskej prívetivosti (napr. pre zistenie, či je zariadenie orientované na výšku alebo šírku), vo väčšine prípadov boli tieto API využívané pre sledovanie užívateľov. Autori štúdie uvádzajú, že zo skúmaného počtu 100000 najznámejších stránok zoznamu Alexa sa až na 1695 webových stránkach pristupuje k niektorému z objektov, ktoré ponúkajú rozhranie pre prácu so senzormi príslušného zariadenia, pričom hodnoty získané zo senzorov sa v istých prípadoch posielajú na server prevádzkovateľov príslušných webových stránok. Autori ďalej uvádzajú, že pomerne veľké množstvo z prístupov k API pre prácu so senzormi je vykonaných skriptami, ktoré patria rôznym spoločnostiam spojeným s reklamou na internete.

2.2.2 Nové JavaScriptové API

V tejto podkapitole sú predstavené vybrané JavaScriptové API, ktoré boli implementované vo webových prehliadačoch od roku 2016. Do vybraných API začleňujeme aj tie API, ktoré ešte nie sú ustálené vo všetkých prehliadačoch a je možné, že ich presná špecifikácia sa môže meniť.

Jedným z takýchto relatívne nových JavaScriptových API je Payment Request API, ktoré je špecifikované organizáciou W3C [43]. Toto API je určené pre použitie na webových stránkach, v rámci ktorých môže dochádzať k procesu objednávania nejakého tovaru (ďalej budeme o tomto procese hovoriť ako o objednávkovom procese), pri ktorom užívatelia naprieč rôznymi webovými stránkami častokrát vypĺňajú rovnaké, opakujúce sa informácie. Podstatou tohto API je umožniť webovým stránkam čerpať informácie, ktoré už raz užívateľ v rámci nejakého objednávkového procesu v minulosti zapísal, čím sa vo výsledku môže významne skrátiť čas, ktorý užívateľ strávi pri objednávkovom procese. Na druhej strane, využitie tohto API môže byť výhodné aj pre tvorcov webových stránok zahŕňajúcich objednávkový proces, pretože podľa štatistík je nedokončenie objednávkového procesu častokrát zapríčinené vysokým počtom krokov, ktoré musí užívateľ pre jeho dokončenie vykonať [20]. Payment Request API vo svojej podstate počet týchto krokov pri opakovaných nákupoch znižuje, čím sa zvyšuje pravdepodobnosť, že frustrácia užívateľov bude pri objednávkovom procese nižšia. Pri návrhu tohto API bolo myslené aj na bezpečnosť a už v samotnej špecifikácii určenej pre vývojárov webových prehliadačov je vyžadované, aby bolo toto API vystavené prehliadačom len v tzv. bezpečnom kontexte (z angl. *secure context*). Podľa špecifikácie je dokument v bezpečnom kontexte vtedy, ak ide o aktívny dokument[3], ktorého top–level prehliadací kontext[4] je tiež bezpečný kontext.

Ďalším zástupcom pomerne nového API je Intersection Observer API, ktorého špecifikácia pochádza od W3C [50]. Vo všeobecnosti je Intersection Observer API určené pre asynchrónne získavanie informácií o viditeľnosti a pozícii daného elementu na webovej stránke voči jeho rodičovským elementom, prípadne voči viditeľnej časti stránky (po anglicky nazývanej ako *viewport*). Zavedenie tohto API do prehliadačov je motivované viacerými prípadmi užitia, pre ktoré môže byť Intersection Observer API dobrým základom pre implementáciu. Napríklad, v minulosti bolo pomerne obťažné a vo výsledku nespoľahlivé detegovať viditeľnosť reklamy, pričom takéto informácie sú dôležité pre kalkuláciu výnosov zo zobrazených reklám na webových stránkach. Podľa štúdie, ktorá sa zaoberala výskumom detekcie zobrazenia reklamy na webových stránkach sú metódy, ktoré sú založené na použití Intersection Observer API pomerne spoľahlivé a vykazujú vyššiu účinnosť, než implementácie založené na iných API [13]. Ďalej, Intersection Observer API môže byť použité napríklad pri rozhodovaní, či vykonať nejaký náročnejší výpočet alebo animáciu na webovej stránke na základe toho, či užívateľ vidí alebo nevidí výsledok príslušnej operácie. Na uvedenej myšlienke môže byť založená implementácia načítania ďalších prvkov na webovú stránku podľa toho, akú časť stránky má užívateľ v príslušnom momente zobrazenú.

Ako ďalší príklad nového API môžeme uviesť Network Information API, ktorého cieľom je ponúknuť webovým vývojárom možnosť získať informácie o internetovom pripojení užívateľa príslušnej webovej stránky, vďaka čomu môže webová stránka reagovať napríklad reguláciou množstva dát, ktoré k užívateľovi prúdia v závislosti od typu užívateľovho pripojenia (WiFi, mobilné dáta a pod.). Network Information API je špecifikované skupinou

[3] https://html.spec.whatwg.org/multipage/browsers.html#active-document

[4] https://html.spec.whatwg.org/multipage/browsers.html#top-level-browsing-context

WICG [16] a hoci ide zatiaľ ide len o draft špecifikácie a je možné, že vo finálnej špecifikácii môže dôjsť k zmenám, rôzne prehliadače už dnes toto API implementujú.

Do kategórie relatívne nových API, ktoré súvisia s prenosom dát, môžeme bez pochýb zaradiť Streams API špecifikované organizáciou WHATWG [48], ktoré slúži na spracovanie prúdov prijímaných dát. Hlavnou motiváciou pre vznik Streams API je riešenie prenosu a spracovania väčšieho množstva dát. V minulosti bolo pre spracovanie dát jazykom JavaScript potrebné počkať na stiahnutie celého súboru, ktorý bolo potom možné celý načítať do pamäte a následne spracovať pomocou JavaScriptu. Streams API prináša do spracovania dát prijímaných zo siete nový spôsob, pomocou ktorého sú prúdiace dáta rozdelené na menšie časti, ktoré je možné spracovať samostatne hneď, ako sú tieto jednotlivé časti prijaté webovým prehliadačom. Špecifikácia Streams API tiež definuje možnosť vytvárania vlastných prúdov dát a možnosť vzájomne tieto prúdy dát reťaziť. Hoci koncept prúdov dát nie je úplne nový a vznikol už v období okolo roku 2015, Streams API vo svojej pôvodnej podobe nešpecifikovalo možnosť reťazenia dátových prúdov a preto sme medzi novšie API zaradili aj Streams API, ktoré bolo o podporu reťazenia dátových prúdov rozšírené až po roku 2016.

Medzi novými API môžeme nájsť aj API, ktoré priamo súvisia so zlepšovaním užívateľských prostredí v rámci webových stránok. Napríklad Streams API zjednodušuje prácu s dátami najmä programátorom webových stránok a užívatelia si možno častokrát ani neuvedomujú, že na pozadí ich obľúbenej webovej stránky došlo k zmenám, po ktorých im je obsah doručovaný a spracovávaný pomocou dátových prúdov s použitím Streams API. Existujú však API, ktorých implementácia súvisí priamo s užívateľským prostredím webovej stránky. Takýmto API je napríklad pomerne nové Picture-in-Picture API špecifikované organizáciou W3C [5]. Toto API slúži na zobrazenie videa mimo okna webového prehliadača s tým, že video je zobrazené vždy v hornej vrstve tak, aby mohol užívateľ pracovať s inými webovými stránkami alebo aplikáciami a stále mal video zobrazené. Medzi ďalšie API, ktoré bolo implementované väčšinou webových prehliadačov v ostatných rokoch patrí Resize Observer API, ktoré umožňuje tvorcom webových stránok programovať ich užívateľské prostredia tak, aby vedeli reagovať na zmeny veľkosti nejakého elementu na stránke.

S rastúcim počtom rôznych platforiem, na ktorých je možné prezerať webové stránky rastie aj potreba vzniku takých API, ktoré budú brať ohľad aj na takéto nové platformy. Príkladom takého API môže byť napríklad nové Pointer events API špecifikované organizáciou W3C [6], ktoré vychádza z myšlienky, že webové stránky dnes už nie sú prehliadané len na zariadeniach, ku ktorým je pripojená myš alebo touchpad, ale množstvo webových stránok je používaných napríklad na mobilných zariadeniach s dotykovým displejom. Pointer events API prináša celý rad udalostí, ktoré sú jednotné pre akékoľvek zariadenia umožňujúce prácu s kurzorom. Kurzor (po angl. *pointer*) je v súvislosti s Pointer events API definovaný ako zariadenie nezávislé od hardvérovej implementácie, ktoré umožňuje určiť konkrétnu súradnicu (alebo množinu súradníc). Vo všeobecnosti ide teda o API, ktoré webovému prehliadaču sprístupňuje udalosti pre prácu s myšou, touchpadom, trackpadom, dotykovým displejom a rôznymi ďalšími nástrojmi pre ovládanie zariadení, na ktorých môže byť eventuálne zobrazená webová stránka.

Z uvedených API môžeme vidieť, že štandardizačné organizácie špecifikujú rôzne nové API, pomocou ktorých sú webovým vývojárom sprístupnené nové možnosti, vďaka ktorým môžu užívateľom svojich webových stránok v čoraz väčšom rozsahu sprostredkovať užívateľskú skúsenosť, ktorú poznajú z natívnych aplikácií. Môžeme tiež vidieť, že vývoj nových API nie je sústredený len na také API, ktoré sú zamerané na efektívnejšiu prácu s dátami a ich prenosom po sieti (ako napríklad Streams API, Network Information API

alebo aj Intersection Observer API), ale vidíme, že ešte stále pribúdajú nové API aj v oblasti, ktorá ponúka možnosti pre pohodlnejšiu prácu s užívateľským prostredím prehliadača (napr. Picture-in-Picture API, Resize Observer API alebo Pointer Events API).

2.3 Implementácie jazyka JavaScript

JavaScript je špecifikovaný štandardom, ktorý je verejne dostupný a preto nie je prekvapivé, že existuje viacero jeho konkrétnych implementácií. V tejto podkapitole uvedieme niektoré vybrané implementácie jazyka JavaScript a popíšeme ich hlavné charakteristiky a vlastnosti.

Pred samotným vymenovaním vybraných implementácií JavaScriptu je vhodné uviesť, že časť moderných JavaScriptových interpretov sú v skutočnosti just–in–time (JIT) kompilátory, ktoré sú založené na princípe, kedy sú frekventované časti kódu prekladané za behu programu priamo do strojového kódu [4]. Súčasťou JIT kompilátorov je spravidla mechanizmus, ktorý analyzuje zdrojový kód, ktorého preklad priamo do strojového kódu ušetrí väčšie množstvo času pri vykonávaní príslušného strojového kódu, než sa spotrebuje na preklad do strojového kódu.

2.3.1 SpiderMonkey

SpiderMonkey je open-source JavaScript Engine, ktorý bol vyvinutý spoločnosťou Mozilla Foundation. SpiderMonkey je engine, ktorý je napísaný v jazykoch C a C++ [31]. Engine SpiderMonkey je súčasťou rôznych aplikácií od Mozilly, ako napríklad Mozilla Firefox alebo Mozilla Thunderbird, avšak nie je používaný len samotnou Mozillou. SpiderMonkey je možné nájsť napríklad aj v MongoDB [33] alebo Adobe Acrobat [1].

2.3.2 V8

V8 je JavaScriptový engine vyvíjaný v spoločnosti Google, ktorý je napísaný v jazyku C++. Ide o engine, ktorý je súčasťou prehliadača Google Chrome a je tiež použitý v rámci platformy Node.js, ktorá umožňuje beh JavaScriptu mimo webového prehliadača, napríklad na webových serveroch. Engine V8 je open–source[5].

2.3.3 JavaScriptCore

Spoločnosť Apple vyvíja engine pre webové prehliadače s názvom WebKit, ktorého súčasťou je JavaScriptový engine JavaScriptCore. Vďaka tomu je JavaScriptCore súčasťou rôznych aplikácií ako napríklad Safari, Mail alebo App Store, ktoré sú na engine WebKit postavené [2].

JavaScriptCore má niekoľko ďalších pomenovaní, ktoré sa naň odkazujú, ako napríklad SquirrelFish, SquirrelFish Extreme, Nitro alebo Nitro Extreme.

2.4 Používanie JavaScript API na internete

V tejto kapitole sa zameriame na existujúci výskum v oblasti obľúbenosti rôznych API jazyka JavaScript, ktoré sú využívané na webových stránkach verejne dostupných na internete.

[5] https://github.com/v8/v8/blob/master/LICENSE.v8

Výskumnou otázkou obľúbenosti JavaScriptových API v prehliadači sa vo svojej práci s názvom Browser Feature Usage on the Modern Web zaoberá kolektív výskumníkov z University of Illinois at Chicago, ktorí vo svojej práci skúmajú, ako populárne sú jednotlivé API jazyka JavaScript dostupné v bežne používanom webovom prehliadači [41], konkrétne v prehliadači Mozilla Firefox. Vo svojej práci sa venujú návrhu metodológie, ktorá im umožnila zmerať používanie rôznych API jazyka JavaScript, ktoré sú dostupné v prehliadači a následne vyhodnoteniu meraní.

Základnou myšlienkou pri meraní, ako rozšírené je používanie jednotlivých API bolo vytvorenie meracieho rozšírenia do prehliadača Mozilla Firefox, ktoré bolo schopné merať volania jednotlivých metód rôznych API. Toto rozšírenie bolo zároveň do istej miery schopné merať prácu s vlastnosťami objektov, ktoré tieto API poskytovali. Konkrétne, autori boli schopní detegovať zápis do vlastností objektov, ktoré sú v prehliadači reprezentované ako singleton (napr. `window`, `window.document`, `window.navigator`). Hoci rozšírenie nebolo verejne publikované, vďaka jeho popisu je známe, že jeho fungovanie je založené na princípe, kedy sú jednotlivé metódy implementujúce rôzne JavaScriptové API nahradené špeciálne nakonfigurovanými Proxy objektami, ktorým sa bližšie venujeme v časti 3.2. Toto meracie rozšírenie bolo použité pri automatizovanej návšteve 10 000 rôznych webových stránok, ktoré boli súčasťou tzv. Alexa rankingu obsahujúceho až milión najpopulárnejších webových stránok internetu.

Každá webová stránka z uvedenej množiny stránok bola navštívená dvoma spôsobmi. Prvý spôsob návštevy reprezentoval zobrazenie stránky v prehliadači, ktorý mal nainštalovaný jediný doplnok, ktorým bolo už spomínané meracie rozšírenie. Druhý spôsob návštevy tejto stránky bol obohatený o dva ďalšie doplnky prehliadača: doplnok určený pre blokovanie reklamy (AdBlockPlus) a doplnok pre blokovanie sledovania užívateľov webových stránok (Ghostery). So zámerom zaznamenať čo najviac prístupov k rôznym JavaScriptovým API nebola navštívená len úvodná stránka každého webu zo zoznamu, ale okrem úvodnej stránky boli navštívené ďalšie tri podstránky, na ktoré bolo odkazované z titulnej stránky. Z každej z týchto troch stránok boli navštívené ďalšie tri stránky. Vo výsledku mohlo byť teda analyzovaných až 13 podstránok každej webovej stránky. Na fakt, že analýza domovských stránok nie je pri obdobných meraniach dostatočná poukazujú aj výskumní pracovníci Aqeel et al., ktorí sa vo svojej štúdii zamerali na porovnanie domovských stránok s podstránkami [3]. Z ich záverov vyplýva, že tieto dva typy stránok sa môžu pomerne významne líšiť vo veľkosti ale aj čase načítania a vykresľovania.

Z výsledkov výskumu Browser Feature Usage on the Modern Web vyplýva, že niektoré API sú veľmi populárne a využíva ich viac než 90% stránok, ktoré boli v rámci meracieho procesu navštívené (napr. metóda `Document.prototype.createElement`). Na druhej strane sú však API, ktoré sú využívané pomerne málo: takmer 50% implementovaných API, ktoré prehliadač v príslušnom čase ponúkal, neboli navštívenými stránkami využité ani raz. Ďalších 29% API využívalo menej, než 1% stránok, nad ktorými bolo popísané meranie vykonané.

Vďaka meraniu používania API v dvoch režimoch – bez rozšírenia určeného na blokovanie reklám a sledovanie užívateľov a potom s ich použitím sa autorom výskumu podarilo zistiť, že blokovanie rôznych API nie je rovnomerné a niektoré API sú blokované omnoho častejšie, než iné. Konkrétne, 10% API bolo blokovaných vo vyše 90% prípadoch pri použití blokovacích rozšírení. Autorom štúdie sa tiež podarilo zistiť, že viac než 83% API prehliadača bolo pri použití blokovacieho rozšírenia použitých na menej než 1% webových stránkach.

Uvádzaný výskum sa tiež zaoberal vzťahom medzi tým, kedy boli štandardy jednotlivých API vydané a ich obľúbenosťou, resp. ich používaním naprieč webovými stránkami rebríčka Alexa. Z analýzy vyplýva, že medzi týmito dvoma veličinami neexistuje priama spojitosť. Existujú štandardy, ktoré sú špecifikované pomerne dlhú dobu a ich obľúbenosť je naprieč rebríčkom vysoká (napr. *XMLHttpRequest*). Na druhej strane existujú štandardy, ktoré sú rovnako staré, avšak sú používané menej (napr. *HTML: Plugins*), než niektoré novšie štandardy (napr. *Selectors API Level 1*).

Autori štúdie s názvom VisibleV8: In-browser Monitoring of JavaScript in the Wild, Kapravelos a Jueckstock, sa postavili pred výskumnú otázku vytvorenia platformy, pomocou ktorej je možné zaznamenávať informácie o vykonávaní JavaScriptu v prehliadači [19]. Na rozdiel od autorov vyššie spomínanej štúdie, pri ktorej bolo meranie inštrumentované pomocou rozšírenia do prehliadača, sa Kapravelos a Jueckstock venovali modifikovaniu JavaScriptového engine V8 a tvorbe frameworku, ktorý nazvali VisibleV8. Pomocou tejto metódy je možné zaznamenať nielen invokácie metód JavaScriptových API a zápis vlastností singleton objektov vystavujúcich príslušné API, ale je možné zaznamenávať prístupy aj k vlastnostiam takých objektov, ktoré nie sú v rámci prehliadača implementované ako singleton. Autori sa vo svojej štúdii venujú najmä popisu vytvoreného frameworku, pričom výsledky meraní, ktoré by sme mohli porovnať s výsledkami publikovanými v článku Browser Feature Usage on the Modern Web, neboli publikované.

VisibleV8, ktorý je popísaný vyššie, bol použitý pri výskume realizovanom autormi Sarker, Jueckstock a Kapravelos. Tento výskum sa sa zaoberal detekciou obfuskácie jazyka JavaScript na webe [37]. Autori vo svojej štúdii považujú kód za obfuskovaný vtedy, ak je pri dynamickej analýze kódu zistený taký prístup k API, ktorý nebolo možné objaviť statickou analýzou príslušného kódu. Z výsledkov výskumu, v rámci ktorého autori získali dáta pre 77423 webových stránok, vyplýva, že iba v prípade približne 4% webových stránok sa na príslušnej stránke nenachádzal obfuskovaný kód jazyka JavaScript. Autori štúdie tiež ukazujú, že najväčší počet obfuskovaného kódu pochádza z externých URL adries (až 98%). Z výsledkov meraní tiež vyplýva, že veľká časť obfuskovaného JavaScriptu je servírovaná stránkami, ktoré sú zamerané na spravodajstvo. Tento trend autori vysvetľujú tým, že na spravodajských portáloch sa nachádza veľké množstvo reklamných skriptov, ktoré sú často spájané so sledovaním a identifikáciou užívateľov.

V tabuľke 2.2 môžeme vidieť názvy metód rôznych JavaScriptových API, ktoré boli volané najčastejšie z obfuskovaného kódu, kde percentil rank je hodnota vyjadrujúca popularnosť využívania príslušnej metódy v obfuskovanom kóde. Percentil rank každej metódy je vypočítaný ako rozdiel percentilu (popularity) príslušnej metódy v obfuskovanom kóde a percentilu príslušnej metódy v kóde, ktorý autormi štúdie nebol vyhodnotený ako obfuskovaný. Rovnakým spôsobom sú vypočítané hodnoty percentil ranku aj v tabuľke 2.3, ktorá uvádza tie vlastnosti objektov rôznych API, ku ktorým sa v obfuskovanom kóde pristupuje najčastejšie.

Z uvedených tabuliek môžeme vidieť, že pomerne veľká časť týchto metód a vlastností slúži na simuláciu a detekciu užívateľovej interakcie s webovou stránkou alebo pre prácu s formulárovými prvkami na stránke. V tabuľke 2.3 tiež môžeme vidieť, že prístup k Battery API, diskutovanom v časti 2.2.1, je pomerne často vykonávaný v rámci obfuskovaného kódu.

V prvej polovici roku 2018, dňa 25.5.2018, vstúpilo do platnosti nariadenie Európskej únie s názvom Všeobecné nariadenie o ochrane údajov (známe tiež ako GDPR, *General Data Protection Regulation*), ktoré má fyzickým osobám pomôcť získať vyššiu mieru kontroly nad získavaním a distribúciou ich osobných informácií rôznymi službami, napríklad webovými stránkami. Zmenu v používaní JavaScriptu distribuovaného tretími stranami po uvedení

Názov metódy	Percentil rank
`Element.scroll`	50,21
`HTMLSelectElement.remove`	36,39
`Response.text`	34,24
`HTMLInputElement.select`	31,97
`ServiceWorkerRegistration.update`	30,23
`Window.scroll`	27,76
`PerformanceResourceTiming.toJSON`	27,53
`HTMLElement.blur`	26,78
`Iterator.next`	26,35
`Navigator.registerProtocolHandler`	26,03

Tabuľka 2.2: Metódy rôznych JavaScriptových API, ktoré sú najčastejšie invokované z obfuskovaného kódu podľa Sarker et al. [37]

Názov vlastnosti	Percentil rank
`UnderlyingSourceBase.type`	67,56
`HTMLInputElement.required`	38,02
`Navigator.userActivation`	36,35
`StyleSheet.disabled`	35,05
`CanvasRenderingContext2D.imageSmoothingEnabled`	34,68
`Document.dir`	33,00
`HTMLElement.translate`	32,14
`HTMLTextAreaElement.disabled`	32,01
`Document.fullscreenEnabled`	30,77
`BatteryManager.chargingTime`	30,34

Tabuľka 2.3: Vlastnosti rôznych JavaScriptových API, ku ktorým je najčastejšie pristupované z obfuskovaného kódu podľa Sarker et al. [37]

GDPR do platnosti sledovala vo svojej práci dvojica výskumníkov Sørensen a Kosta [44]. V rámci ich výskumu bolo vykonaných 21 automatizovaných meraní na reálnych webových stránkach v priebehu siedmych mesiacov, pričom nariadenie GDPR vošlo do platnosti približne v polovici tohto intervalu. Pri interpretácii výsledkov meraní autori konštatujú mierny pokles používania skriptov tretích strán, vo všeobecnosti sa im však nepodarilo preukázať priamy vzťah a koreláciu medzi uvedením GDPR do platnosti a poklesom využívania skriptov tretích strán.

Autori uvedenej štúdie, Sørensen a Kosta, vykonávali meranie v rámci webových stránok s pôvodom v rámci Európskej únie, avšak pre poskytnutie širšieho pohľadu na používanie skriptov tretích strán vykonávali merania aj na stránkach, ktoré nepochádzali len s prostredia z Európskej únie, ale aj z ostatných krajín, napríklad USA. Pri kategorizácii analyzovaných webových stránok autori neostali len pri rozdelení podľa krajiny pôvodu príslušných webových stránok, ale webové stránky kategorizovali aj podľa druhu zriaďovateľa na verejné a privátne (ako verejné označili tie, ktoré sú zriaďované z verejných financií, napríklad webové stránky rôznych vládnych inštitúcií, verejných univerzít a pod.) alebo do kategórií podľa obsahu príslušnej webovej stránky (napr. spravodajstvo, zábava, nakupovanie, cestovanie, počasie). Zatiaľ čo vo väčšine kategórií bol počet skriptov tretích strán

takmer nemenný, v kategóriách privátneho spravodajstva, zábavy, nakupovania a cestovania autori štúdie zaznamenali pokles počtu skriptov tretích strán. Zároveň však dodávajú, že nemusí ísť o pokles v dôsledku zavedenia GDPR a zdôrazňujú, že môže ísť o dôsledok rôznych ďalších faktorov.

Kapitola 3

Návrh platformy pre zber dát

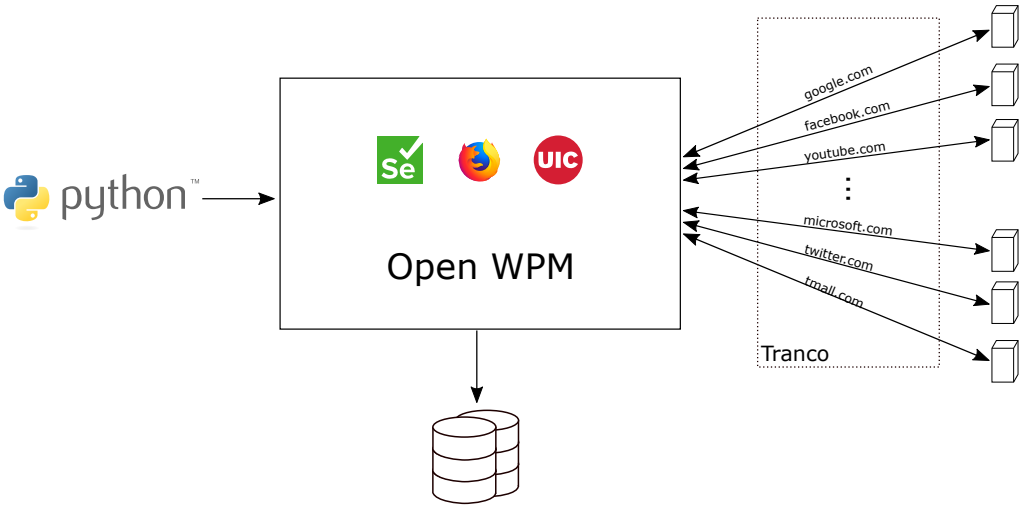
V tejto časti práce je zdokumentovaný návrh prostredia, v ktorom sme vykonávali zber dát pre neskoršiu interpretáciu a analýzu. Platformu pre zber dát ďalej v texte nazývame aj ako testovaciu platformu, pretože slúži na testovanie toho, aké JavaScriptové API sú danými stránkami používané.

Základným cieľom zberu dát je získať informácie o tom, ako často sú používané jednotlivé API sprístupnené prehliadačom cez JavaScript. Vzhľadom na veľký počet webových stránok, ktoré chceme v rámci našich meraní navštíviť, je vhodné, aby bola platforma pre zber dát realizovaná tak, aby bolo možné navštíviť webové stránky automatizovaným spôsobom. Ďalej, zoznam navštívených stránok by mal obsahovať také stránky, ktoré budeme môcť považovať za reprezentatívnu vzorku verejného internetu. Dáta zozbierané pri tomto meraní budeme ďalej analyzovať.

Na obrázku 3.1 je načrtnutá schéma štruktúry testovacieho prostredia, ktoré sa skladá z viacerých stavebných kameňov a technológií. Medzi hlavné komponenty tohto testovacieho rozhrania patrí nástroj OpenWPM, ktorý budeme používať pre inštrumentáciu celého procesu návštev jednotlivých webových stránok. Predstaveniu nástroja OpenWPM sa budeme venovať bližšie v podkapitole 3.1. Ďalej v podkapitole 3.2 predstavíme koncept rozšírenia webového prehliadača, pomocou ktorého budeme merať používanie jednotlivých API jazyka JavaScript na webových stránkach, ktoré pri meraní navštívime. V podkapitole 3.3 uvedieme a priblížime rebríčky webových stránok založené na ich popularite v rámci internetu a uvedieme zoznam webových stránok, ktorý pri našom meraní využijeme. Nakoniec v podkapitole 3.4 uvedieme spôsob, ktorým získame informácie o JavaScriptových API, ktoré budeme pri našom meraní sledovať.

3.1 Inštrumentácia automatizovanej návštevy stránok

Vykonanie automatických návštev jednotlivých stránok, ktoré budú podliehať nášmu meraniu, bude vykonávané pomocou prostredia OpenWPM. Ide o prostredie navrhnuté pre účely automatizovaného testovania a návštev veľkého počtu webových stránok, pomocou ktorého je možné toto testovanie a návštevy inštrumentovať [12]. OpenWPM je nástroj, ktorý bol vytvorený za účelom vykonania merania, ktoré bolo sústredené na fingerprinting a sledovanie užívateľov webových stránok za použitia prehliadača. Autori však uvádzajú, že tento nástroj je možné použiť aj pre merania, ktoré majú iný účel. Na obrázku 3.2 je načrtnutá zjednodušená architektúra nástroja OpenWPM tak, ako ju prezentujú jeho autori.



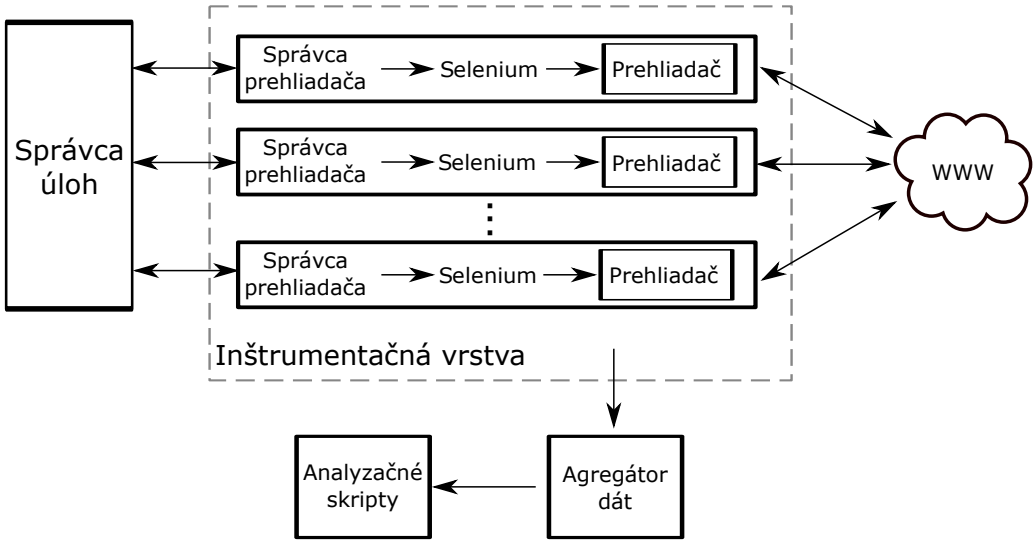
Obr. 3.1: Architektúra prostredia pre meranie používania JavaScriptových API

OpenWPM sa skladá z troch hlavných častí, ktoré sú tvorené správcami prehliadača, správcom úloh a agregátorom dát [12]. Správcovia prehliadača tvoria vrstvu abstrakcie pre jednotlivé prehliadače a pre ich inštrumentáciu. Užívateľ nástroja OpenWPM interaguje so správcom úloh, ktorý je zodpovedný za prideľovanie pokynov správcom prehliadača. Interakcia užívateľa nástroja OpenWPM so správcom úloh prebieha pomocou jazyka Python. Agregátor dát je autormi nástroja prezentovaný ako abstrahovaná vrstva, ktorá je dostupná inštrumentácii prehliadača.

Pre konkrétnu inštrumentáciu prehliadača sa v rámci OpenWPM používa nástroj Selenium, ktorý je vyvinutý za účelom automatického testovania webových stránok. Ide o tzv. webový ovládač (z angl. *web driver*), pomocou ktorého je možné riadiť a inštrumentovať webový prehliadač počas testovania webovej stránky. Selenium deklaruje podporu viacerých prehliadačov, napríklad Mozilla Firefox, Safari, Edge, Chrome, Internet Explorer a iné [39]. Pri výbere prehliadača sa autori OpenWPM rozhodli využiť podporu pre prehliadač Mozilla Firefox.

Na meranie prístupu jednotlivých stránok z našej testovacej sady k rôznym JavaScript API budeme používať samostatné rozšírenie do prehliadača, ktoré nie je súčasťou OpenWPM. Vďaka faktu, že Selenium a poťažmo aj OpenWPM umožňuje inštrumentáciu plnej verzie webového prehliadača, z ktorého potom budú jednotlivé webové stránky navštívené, môžeme týmto spôsobom dané rozšírenie do prehliadača pridať a pozmeniť tým jeho prostredie. Samotnému rozšíreniu, pomocou ktorého budeme vykonávať naše meranie, sa venujeme v podkapitole 3.2.

OpenWPM umožňuje zapisovať rôzne dáta namerané počas návštevy príslušnej webovej stránky pomocou agregátora dát. Agregátor dát umožňuje zapisovať dáta do SQLite databázy. Nástroje pre podporu samotnej analýzy, ktoré sú na obrázku 3.2 znázornené blokom *Analyzačné skripty* nie sú súčasťou OpenWPM.



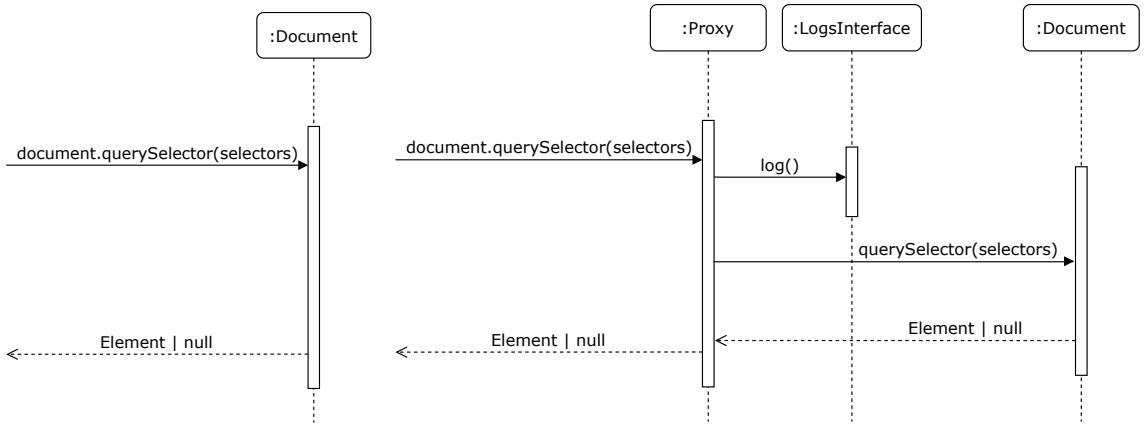
Obr. 3.2: Zjednodušená architektúra nástroja OpenWPM [12]

3.2 Meracie rozšírenie do prehliadača

Na získanie informácií o tom, ktoré API jazyka JavaScript sa na jednotlivých stránkach využívajú, je potrebné pozmeniť prostredie webového prehliadača tak, aby nám bolo umožnené volania jednotlivých API zaznamenávať. Tento cieľ môžeme docieliť vytvorením rozšírenia do webového prehliadača.

Pri tvorbe rozšírenia, ktorým pozmeníme prostredia prehliadača tak, aby sme boli schopní zaznamenávať volania JavaScriptových API, budeme vychádzať z rozšírenia Web API Manager, ktoré vzniklo v rámci výskumu realizovaného Petrom Snyderom [41, 42]. Ide o rozšírenie, ktorého hlavným účelom je blokovať API poskytnuté webovým prehliadačom, ktoré bývajú používané na sledovanie a identifikáciu návštevníkov webových stránok. Medzi tieto API patrí napríklad *High Resolution Time Level 2*, *Canvas* alebo *Web Audio API* [42]. Vo všeobecnosti je však možné rozšírenie využiť pre blokovanie rôznych API, aj takých, ktoré nie sú bežne využívané v súvislosti so sledovaním užívateľov webu. Rozšírenie obsahuje prostredie, v ktorom je užívateľovi umožnené definovať vlastné pravidlá, ktoré určujú následné blokovanie príslušných API prehliadača. Tieto pravidlá môžu byť definované na úrovni jednotlivých webových stránok podľa URL adresy, prípadne na úrovni skupiny webových stránok definovanej regulárnymi výrazmi reprezentujúcimi URL adresy stránok, pri ktorých má byť blokovanie definovaných API aplikované.

Verzia jazyka ECMAScript 2016, taktiež zvaná ES6, uvádza do JavaScriptu dostupnosť tzv. Proxy objektov, ktoré umožňujú pozmeniť základné správanie objektov v JavaScripte. Táto technika metaprogramovania je využitá aj v rámci doplnku Web API Manager, kde sú objekty poskytujúce rozhranie blokovaných JavaScriptových API nahradené špeciálne konštruovanými a inštruovanými Proxy objektami. Tieto špeciálne Proxy objekty následne prijímajú volania rôznych metód spadajúcich pod príslušné API, ktoré sú využité klientským kódom navštívenej webovej stránky. Proxy objekty umožňujú definovať tzv. pasce (z angl. *traps*), pomocou ktorých je možné registrovať obslužné funkcie vyvolané v prípade,



(a) Bez Proxy objektu (b) S delegáciou a záznamom prístupu k metóde cez Proxy objekt

Obr. 3.3: Sekvenčné diagramy volania metódy `querySelector`

že je Proxy objekt invokovaný napríklad ako funkcia, je pristúpené k jeho vlastnostiam (z angl. *properties*) alebo je nad príslušným Proxy objektom vykonaná nejaká iná operácia [14]. V rozšírení Web API Manager sú Proxy objekty reprezentujúce objekty blokovaných API inštruované tak, aby nedelegovali volania metód na pôvodné objekty, ale aby vrátili hodnoty, ktoré nebudú vystavovať príslušné API prehliadača a zároveň v čo najväčšej miere minimalizujú riziko, že kód spoliehajúci sa na dané API spôsobí chybu vo vykonávaní JavaScriptu.

Pri našom meraní využijeme infraštruktúru, ktorá je už v uvedenom rozšírení vytvorená. Pre naše účely budeme musieť vykonať v rozšírení zmeny, ktoré nám umožnia merať prístupy k API. Pri našom meraní nie je žiaduce, aby boli jednotlivé API blokované a preto budeme Proxy objekty konfigurovať tak, aby boli volania metód delegované na pôvodné objekty. Ďalšou zmenou, ktorú budeme v rámci konfigurácie Proxy objektov implementovať je záznam prístupu k jednotlivým Proxy objektom, čím budeme realizovať meranie prístupu k metódam implementujúcim príslušné JavaScriptové API, ktoré je prehliadaču poskytnuté. Dáta z meraní môžeme následne spracovať agregátorom dát a uložiť do perzistentnej podoby. Hlavná myšlienka spôsobu využitia Proxy objektov pri našom meraní je znázornená na obrázku 3.3, kde v časti 3.3a môžeme vidieť volanie metódy `querySelector` vystavenej objektu `Document`, pričom v časti 3.3b môžeme vidieť, že metóda `querySelector` je zavolaná nad Proxy objektom, ktorý je inštruovaný, aby po invokovaní príslušnej metódy zaznamenal prístup k metóde a následne delegoval volanie na objekt `Document`, ktorého návratová hodnota je následne navrátená Proxy objektom. Rozšírenie Web API Manager vo svojej súčasnej verzii umožňuje blokovanie najmä tých JavaScriptových API, ktoré boli v čase vytvárania doplnku spájané s používaním pre účely sledovania a identifikácie užívateľov. Pre účely merania prístupu k rozličným API budeme ďalej musieť rozšírenie Web API Manager rozšíriť o podporu všetkých JavaScriptových API, ktoré príslušný prehliadač implementuje.

Výhodou merania prístupu k rôznym API pomocou Proxy objektov je, že takéto meranie je imúnne voči minifikácii a obfuskácii zdrojového kódu.

3.3 Výber vhodnej sady webových stránok

Pre mieru výpovednej hodnoty získaných dát je výber webových stránok, ktoré v rámci automatizovaného navštívime pomerne dôležitý. Výberu vhodnej sady stránok je preto venovaná celá táto sekcia.

Pomerne veľké množstvo výskumných tímov a prác, ktoré sa zaoberajú témami vyžadujúcimi prístup k reprezentatívnej vzorke webových stránok na internete, používa dáta poskytnuté v zozname Alexa Top Sites. Tento rebríček obsahuje informácie až o jednom milióne webových stránok, ktoré sú zoradené podľa tzv. Alexa Traffic Rank – hodnotenia webových stránok na základe obľúbenosti naprieč užívateľmi s nainštalovaným doplnkom v prehliadači. Obsah uvedeného rebríčka je denne aktualizovaný a prístupný cez oficiálne API. Prístup k Alexa Top Sites je spoplatnený. Výhodou rebríčka Alexa Top Sites je pomerne veľké množstvo informácií, ktoré sú v rebríčku obsiahnuté, pretože dáta obsahujú rôzne ďalšie informácie o jednotlivých stránkach, napríklad počet užívateľov v prepočte na milión užívateľov alebo priemerný počet stránok, ktoré užívateľ na danej stránke navštívi. Nevýhodou tohto rebríčka je jeho prirodzená nestabilnosť a zraniteľnosť voči pomerne jednoduchým útokom [22], vďaka čomu môže dochádzať k častým zmenám v rebríčku a k strate jeho výpovednej hodnoty.

Okrem rebríčka Alexa Top Sites existuje niekoľko ďalších rebríčkov, ktoré sa rôznymi spôsobmi snažia identifikovať najznámejšie webové stránky, prípadne domény na internete. Ako príklady môžeme uviesť napríklad Cisco Umbrella 1 Million [46] alebo Majestic [24]. Uvedené zoznamy sú však náchylné na rôzne chyby, kvôli ktorým môžu byť obsahy jednotlivých zoznamov pomerne nestále a nevýpovedné [38].

Z dôvodov uvedených vyššie sme sa rozhodli ako reprezentatívny zoznam najpoužívanejších webových stránok použiť zoznam Tranco [23]. Obsah zoznamu Tranco je založený na kombinácii viacerých zoznamov (konkrétne Alexa, Cisco Umbrella a Majestic) a je denne aktualizovaný. Zoznam neobsahuje okrem poradia a domény príslušnej webovej stránky žiadne ďalšie informácie.

3.4 Zoznam JavaScript API

Pri zmene prostredia prehliadača, pomocou ktorého vykonáme návštevu webových stránok zo zoznamu Tranco musíme vytvoriť Proxy objekty, ktorými nahradíme objekty vystavujúce API v prehliadači. Pre tento účel budeme potrebovať zoznam všetkých metód, ktoré sú vystavené objektami reprezentujúcimi API jazyka JavaScript.

V kapitole 2.2 sme uviedli, že oficiálna dokumentácia, ktorá špecifikuje rozhrania objektov jednotlivých JavaScriptových API prehliadača je definovaná najmä pre tvorcov prehliadačov. Táto dokumentácia má formu tzv. WebIDL (z angl. *interface definition language*) dokumentov, ktorých účelom je vo všeobecnosti popis rozhraní rôznych Web API, ktoré majú byť implementované vo webových prehliadačoch. Štruktúra WebIDL dokumentov je nezávislá od akéhokoľvek programovacieho jazyka. V oblasti webových prehliadačov existujú dva typy WebIDL dokumentov. Prvým typom sú WebIDL dokumenty vytvorené v rámci špecifikácie príslušných Web API, ktoré slúžia ako referenčná definícia daného API. Druhý typ WebIDL dokumentov reprezentujú WebIDL dokumenty, ktoré sú vytvorené v rámci tvorby webového prehliadača a vypovedajú o implementovaných API v danom prehliadači [26].

Pri vytváraní zoznamu všetkých JavaScript API, ktoré Firefox ako webový prehliadač poskytuje, budeme vychádzať z WebIDL dokumentov, ktoré popisujú implementované API.

WebIDL dokumenty definujú vystavované API prehliadača pomocou tzv. rozhraní (z angl. *interfaces*), ktoré ďalej obsahujú definície metód a vlastností (z angl. *properties*) definovaných daným rozhraním. Pre naše účely budeme potrebovať WebIDL súbory prehliadača Mozilla Firefox, ktoré sú verejne dostupné[1]. WebIDL dokumenty, ktorých analýzou získame zoznam všetkých JavaScriptových metód vystavených prehliadaču majú príponu `.webidl`.

[1] https://dxr.mozilla.org/mozilla-central/source/dom/webidl/

Kapitola 4

Implementácia testovacieho prostredia

V tejto časti sú popísané implementačné detaily a úpravy, ktorými bolo potrebné modifikovať nástroje použité pri meraní používania JavaScript API na verejne dostupných webových stránkach.

Testovacie rozhranie bolo plne implementované a použité pri meraní používania JavaScriptu na 10000 webových stránkach. Behovým prostredím pre vytvorenú platformu boli servery od spoločnosti DigitalOcean, ktorá ponúka možnosť vytvorenia tzv. Dropletov. Pod pojmom Droplet môžeme rozumieť ekvivalent pojmu VPS (*Virtual Private Server*). V rámci vykonávania meraní bolo vytvorených 14 Dropletov, ktoré mali parametre uvedené v tabuľke 4.1.

Názov parametra	**Hodnota parametra**
Typ CPU	Zdieľaný
Počet vCPU	4
Veľkosť operačnej pamäte	8 GB
Veľkost pevného disku	160 GB
NVMe SSD	Nie
Operačný systém	Ubuntu 18.04 (LTS) x64
Poloha datacentra	Frankfurt

Tabuľka 4.1: Parametre Dropletov, na ktorých bolo vykonané meranie používania JavaScriptu webovými stránkami verejne dostupnými na internete

Pri testovacích meraniach sme zaznamenali vysokú pamäťovú náročnosť testovacieho prostredia a zistili sme, že v dôsledku toho dochádzalo k nedostatku pamäte pre prehliadače, ktoré sú v rámci implementovanej platformy ovládané a pomocou ktorých sú webové stránky navštevované. Príčinu problému s vyčerpávaním pamäťových prostriedkov sa nám nepodarilo identifikovať, avšak uvedený problém sme vyriešili postupným spúšťaním meracej platformy po menších častiach. Konkrétne, po navštívení desiatich stránok a ich podstránok, tj. po navštívení potenciálne až 130 webových stránok, platformu reštartujeme a spustíme znova tak, aby navštívila podstránky ďalších desiatich webových stránok.

4.1 Konfigurácia OpenWPM

V tejto časti je uvedený popis konfigurácie a rozšírenia platformy OpenWPM, ktorú sme použili pre automatizovanú návštevu webových stránok. V rámci tejto práce bolo použité OpenWPM v0.14.1. Po vykonaní našich meraní autori platformy OpenWPM zverejnili informáciu o možnej strate niektorých nameraných dát pri použití verzie OpenWPM v0.14.0 a v0.14.1. Straty dát by sa však mali týkať len tých meraní, ktoré boli uskutočnené v tzv. stavovom (z angl. *stateful*) režime platformy, pričom naše merania boli vykonávané v tzv. bezstavovom režime (z angl. *stateless*).

Výsledná aplikácia, ktorá reprezentuje naše meracie prostredie a je založená na používaní OpenWPM vyžaduje niekoľko parametrov. Jedným z najpodstatnejších parametrov je parameter `--sites=path`, kde je `path` cesta k vstupnému súboru. Vstupný súbor je textový súbor obsahujúci informácie o webových stránkach, ktoré majú byť v rámci merania navštívené. Vstupný súbor musí byť vo formáte JSON (*JavaScript Object Notation*) a musí byť nasledovnej štruktúry:

```
{
   "sites": [
      ...
      {
         "site_url": "http://google.com",
         "links_count": 13,
         "links": [
            "https://www.google.com/about/",
            "https://mail.google.com/mail/&ogbl",
            "https://www.google.com/policies/terms/",
            "https://support.google.com/accounts?p=signin_pr...",
            "https://accounts.google.com/ServiceLogin?hl=de&pa...",
            "https://accounts.google.com/TOS?loc=DE&hl=de",
            "https://www.google.com/search/howsearchworks/?fg=1#j...",
            "https://www.google.com/search/howsearchworks/mission/",
            "https://www.google.com/intl/de/gmail/about/policy/",
            "https://support.google.com/accounts?hl=de",
            "https://google.com/search/howsearchworks/?fg=1",
            "http://google.com",
            "https://accounts.google.com/SignUp?service=mail..."
         ],
      },
      ...
   ]
}
```

Môžeme vidieť, že uvedená štruktúra predstavuje objekt s jediným kľúčom `sites`, pomocou ktorého je možné identifikovať kolekciu objektov reprezentujúcich jednotlivé webové stránky, ktoré majú byť pri meraní navštívené.

Medzi ďalšie dôležité povinné parametre meracej platformy patria nasledovné parametre:

- `--browsers=n`, kde `n` určuje počet prehhliadačov, ktoré môžu byť spustené paralelne,

- `--start=n`, kde `n` určuje index prvej webovej stránky z poľa definovaného kľúčom `sites`, ktorá má byť v rámci merania navštívená a
- `--offset=n`, kde `n` určuje počet stránok, ktoré majú byť v rámci merania navštívené.

Pri vykonaných meraniach bola hodnota `n` parametra `--browsers=n` nastavená na hodnotu 3, čo znamená, že v jednom okamihu mohli byť spustené maximálne tri prehliadače. Pri testovacích meraniach bolo zistené, že pri konfigurácii serverov uvedenej v tabuľke 4.1 a pri nastavení parametra `browsers` na hodnotu 4, bola platforma pomerne nestála a dochádzalo k vyčerpaniu operačnej pamäte omnoho skôr.

Pre časovo náročné merania, medzi ktoré môžeme zaradiť aj naše meranie, umožňuje OpenWPM aktivovať funkcionalitu dvoch druhov tzv. strážnych psov (z angl. *watchdogs*). Úlohou strážnych psov v OpenWPM je monitorovať pamäťovú náročnosť prehliadačov spustených samotnou platformou a taktiež ukončovať procesy, ktoré by mohli byť v konflikte s procesmi používanými OpenWPM. Pri našich meraniach boli druhy uvedených strážnych psov automaticky aktivované.

Platforma OpenWPM umožňuje výber z niekoľkých módov prehliadačov, z ktorých sú následne dané webové stránky navštevované. Pri našich meraniach sme zvolili mód prehliadača bez GUI (*Graphical User Interface*). Takýto mód prehliadača sa po anglicky nazýva ako tzv. *headless* mód a pri použití tohto módu je webová stránka navštívená prehliadačom bez reálneho viditeľného grafického užívateľského rozhrania. Mód bez GUI so sebou prináša niekoľko obmedzení. Napríklad, pri návštevách bez GUI nie je možné identifikovať všetky volania WebGL API. Výhodou využitia módu bez GUI je jednoduchšia inštrumentácia, pri ktorej nie je nutné poskytnúť platforme OpenWPM virtuálny displej, napríklad použitím aplikácie XVFB. Zároveň platí, že pomocou bezhlavého módu je možné zachytiť naprostú väčšinu JavaScriptových API.

Metodológia nášho merania zahŕňa použitie dvoch doplnkov prehliadača, pomocou ktorého sú webové stránky navštevované. Konkrétne, rozšírenie Web API Manager pre odchytávanie volaní JavaScriptových funkcií a rozšírenie Ghostery pre blokovanie reklám. Prvé z uvedených rozšírení, Web API Manager, musí byť v prehliadači nainštalované pri každej návšteve, ktorá je pri našom meraní vykonaná. Na druhú stranu, rozšírenie Ghostery musí byť aktívné len pri behoch určených na porovnanie využívaných API bez a s nainštalovaným doplnkom.

Platforma OpenWPM umožňuje dva spôsoby inštalácie rozšírení prehliadača. Prvým možným spôsobom je inštalácia pomocou webového ovládača (z angl. *web driver*) nástroja Selenium, ktorý je inštruovaný pomocou OpenWPM. Týmto spôsobom je do prehliadača nainštalované rozšírenie Web API Manager. Druhým možným spôsobom inštalácie rozšírení do prehliadača inštruovaného OpenWPM je inštalácia pomocou profilu, ktorý je možné načítať pri spustení samotného prehliadača. Profil určený na načítanie do prehliadača inštruovaného pomocou OpenWPM môže okrem iného obsahovať rôzne rozšírenia. Túto možnosť využívame pri návštevách webových stránok, pri ktorých má byť aktívne rozšírenie pre blokovanie reklám Ghostery. Dôvodom použitia uvedeného spôsobu inštalácie rozšírenia Ghostery je, že Ghostery pri svojom prvom použití využíva komunikáciu so servermi[1] pre zostavenie databázy, ktorá sa následne používa pri blokovaní rôznych volaní JavaScriptu. Zostavovanie novej databázy pri každom reštartovaní webového prehliadača môžeme v našom prípade označiť za nežiaduce, pretože výsledná databáza sa môže v rámci toho istého merania líšiť. Uvedené správanie je možné eliminovať prvotnou inicializáciou, ktorá sa

[1] https://github.com/mozilla/OpenWPM/issues/25

vykoná manuálne a následným vyrobením nového profilu, ktorý môže obsahovať už inicializované rozšírenie Ghostery a následným použitím tohto profilu pri spúšťaní prehliadača. Nevýhodou tohto prístupu k inštalácii rozšírení do prehliadača je nižšia výsledná rýchlosť meraní, pretože načítanie profilu je samostatná operácia, ktorá vyžaduje ďalšie časové a pamäťové prostriedky. Použitie doplnku Ghostery, resp. použitie profilu obsahujúceho nainštalované rozšírenie Ghostery je podmienené prítomnosťou parametra `--ghostery` pri spúšťaní meracej platformy.

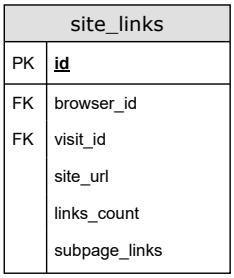
Nasledujúce dve časti uvádzajú spôsob implementácie dvoch vlastných príkazov platformy OpenWPM slúžiacich na zostavenie zoznamu podstránok webov uvedených v zozname Tranco (resp. v zozname akýchkoľvek daných stránok) a pre ich automatizovanú návštevu za účelom vykonaná meraní používania JavaScriptu.

4.1.1 Rozšírenie zoznamu Tranco o podstránky

Vzhľadom na to, že meranie na webových stránok prebiehalo vo viacerých režimoch, konkrétne s rozšírením pre blokovanie reklám a potom bez jeho použitia, bolo potrebné zostaviť sadu webových stránok, ktorá bude pre oba behy meraní totožná. V prípade, že by sme podstránky vyberali nezávisle v rámci každého behu, je možné, že by sme navštívili odlišné stránky a namerané dáta by nemuseli byť konzistentné. Z tohto dôvodu sme sa rozhodli pre zostavenie zoznamu webových stránok ešte pred samotným meraním. Pre tento účel sme platformu OpenWPM rozšírili o vlastný príkaz, použitím ktorého sme navštívili každú z prvých 10 tisíc stránok zoznamu Tranco a zozbierali sme odkazy na ich podstránky.

Vlastný príkaz, ktorý sme nazvali `GetLinksCommand` je naprogramovaný tak, aby navštívil zadanú webovú stránku a vybral na nej náhodným spôsobom tri odkazy, ktoré smerujú na stránky patriace danému webu. Stránkami patriacimi danému webu rozumieme také stránky, ktoré sa nachádzajú na rovnakej doméne prvého rádu alebo na subdoméne (tj. doméne tretieho rádu). V ďalšom kroku sa navštívia všetky takto získané odkazy na podstránky a opäť sa náhodným spôsobom vyberú odkazy na ďalšie tri podstránky patriace danému webu. Takýmto spôsobom sme na každom webe mohli získať až trinásť podstránok daného webu, vrátane odkazu na domovskú stránku.

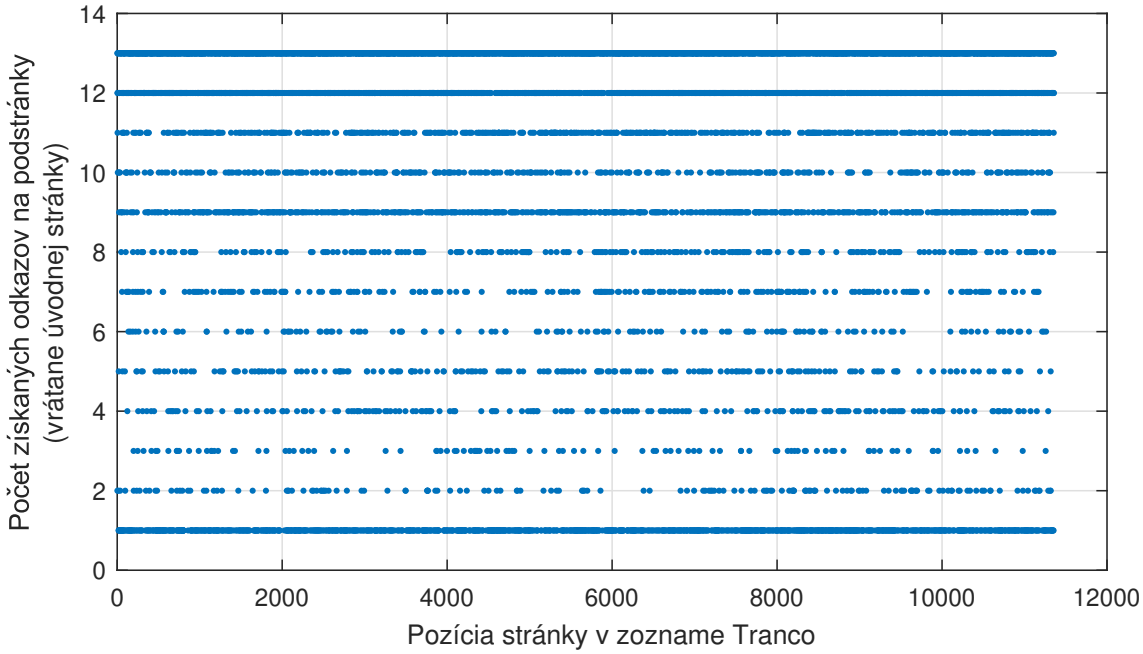
Výsledný zoznam stránok, ktorý bol uvedeným príkazom získaný, je zapisovaný do databázy, v ktorej sme pre tento účel vytvorili vlastnú tabuľku s názvom `site_links` a štruktúrou, ktorá je uvedená na obrázku 4.1. Cudzie kľúče `browser_id` a `visit_id` odkazujú do ďalších tabuliek, ktoré definuje OpenWPM (konkrétne `site_visits` a `crawls`) a ktoré uchovávajú ďalšie informácie, ktoré pre nás nie sú dôležité. Zoznam podstránok stránky uloženej v stĺpci `site_url` je uložený v stĺpci s názvom `subpage_links` vo formáte JSON.



Obr. 4.1: Štruktúra databázovej tabuľky `site_links`, ktorá uchováva zoznam podstránok webových stránok zoznamu Tranco

Pri získavaní zoznamu podstránok pre stránky uvedené v zozname Tranco sme identifikovali domény, na ktorých nie je vystavená žiadna webová stránka. Tieto domény väčšinou slúžia pre rôzne služby internetovej alebo inej infraštruktúry, ako napríklad rôzne DNS služby (napr. *akadns.net*), servery slúžiace pre sťahovanie aktualizácií operačného systému (napr. *windowsupdate.com*) a iné. Tieto adresy boli z výsledného zoznamu webových stránok určeného na navštívenie za účelom merania používania JavaScriptu vyradené. Výsledný zoznam webových stránok obsahuje odkazy na 10000 webových stránok, ktoré sa nám podarilo pomocou príkazu `GetLinksCommand` navštíviť. Pre zostavenie zoznamu podstránok sme použili adresy webových stránok obsiahnuté v zozname Tranco vygenerovanom dňa 15.4.2021[2].

Naprieč 10000 webovými stránkami sa nám podarilo spôsobom uvedeným vyššie identifikovať priemerne 9,78 podstránok (vrátane titulnej stránky, ktorú v tomto kontexte počítame tiež ako podstránku). Medián počtu identifikovaných podstránok má hodnotu 12 a modus počtu identifikovaných podstránok je rovný hodnote 13. Na obrázku 4.2 je znázornený počet podstránok pre 10000 webových stránok zoznamu Tranco, pre ktoré sme získavali podstránky. Z obrázka 4.2 nie je zjavné, že by existovala nejaká závislosť medzi umiestnením v zozname Tranco a počtom podstránok príslušnej webovej stránky. Pre ilustráciu ďalej na obrázku 4.3 uvádzame histogram počtu webových stránok, z ktorého môžeme vidieť počty webových stránok, na ktorých sme získali príslušný počet podstránok.

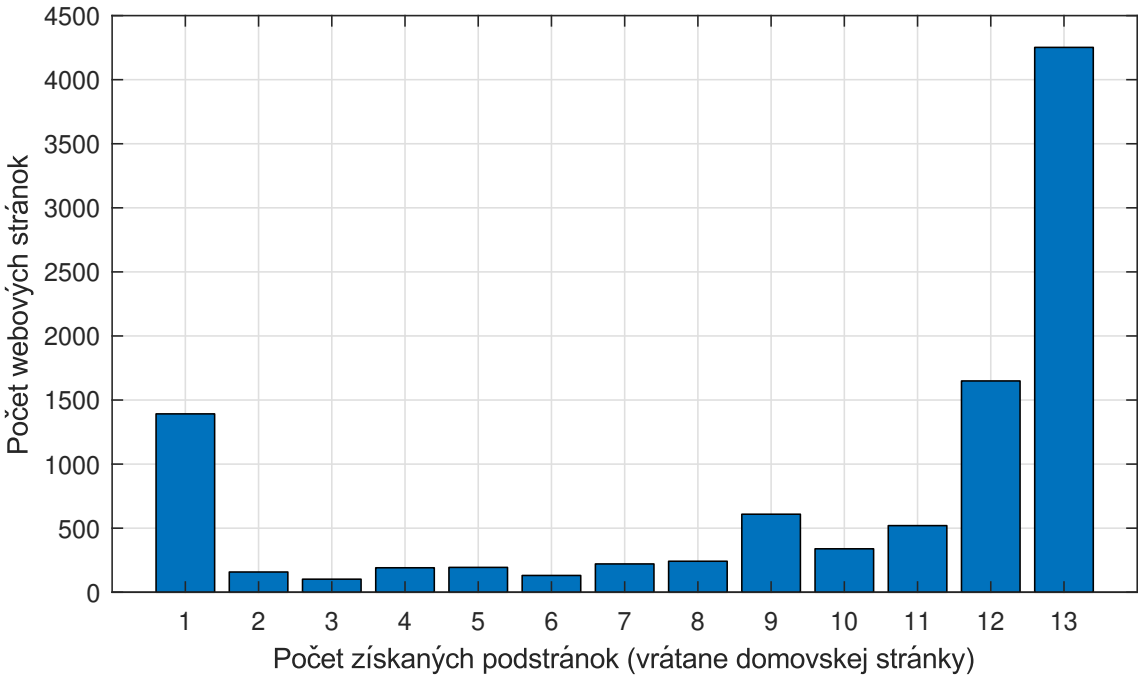


Obr. 4.2: Vzťah pozície webovej stránky v zozname Tranco a počtu získaných odkazov na podstránky danej webovej stránky

4.1.2 Automatizované návštevy webových stránok a ich podstránok

Pre návštevy webových stránok za účelom merania JavaScriptových volaní sme implementovali vlastný príkaz `InterceptJavaScriptCommand` prijímajúci dva argumenty. Jedným

[2] https://tranco-list.eu/download/4ZWX/1000000



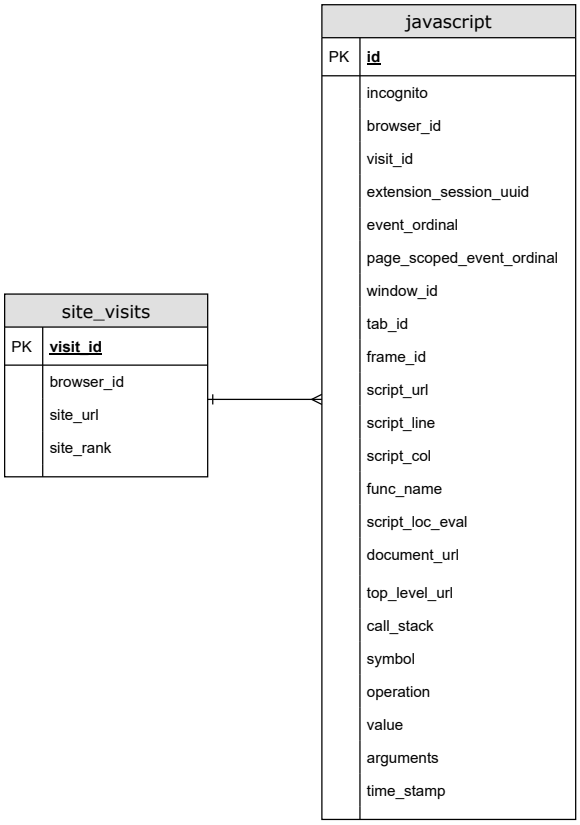
Obr. 4.3: Histogram počtu webových stránok, na ktorých sme získali daný počet podstránok

z dvoch argumentov príkazu `InterceptJavaScriptCommand` je objekt, ktorý reprezentuje webovú stránku určenú na navštívenie a jeho štruktúra je zhodná s objektami v kolekcii objektov s kľúčom `sites` vo vstupnom súbore popísanom vyššie. Druhým argumentom príkazu `InterceptJavaScriptCommand` je číselná hodnota reprezentujúca počet sekúnd, počas ktorých má prehliadač odchytávať volania JavaScriptu na každej podstránke daného webu. Pre všetky návštevy bola táto hodnota pri vykonaných meraniach nastavená na 30 sekúnd.

Počas vykonávania príkazu `InterceptJavaScriptCommand` je vo webovom prehliadači aktívne rozšírenie Web API Manager, ktoré je zodpovedné za odchytávanie JavaScriptových volaní a za následné ukladanie informácií o príslušných volaniach do databázy. Viac informácií o implementácii ukladania informácií z rozšírenia Web API Manager uvádzame v časti 4.2. Štruktúra databázovej tabuľky `javascript`, do ktorej sú ukladané informácie o JavaScriptových volaniach je znázornená na obrázku 4.4. Štruktúra tabuľky je definovaná autormi OpenWPM a je primárne používaná pre ukladanie dát pochádzajúcich z meraní používania JavaScriptu, ktoré sú implementované a pripravené na použitie autormi OpenWPM a ktorých implementácia pre naše účely nebola vhodná. Štruktúra databázy však vyhovuje našim požiadavkám a z toho dôvodu sme ju využili.

Implementácia príkazu `InterceptJavaScriptCommand` je založená na použití vstavaného príkazu `GetCommand` pre každú podstránku danej stránky. Výhodou takejto formy enkapsulácie viacnásobného volania príkazu `GetCommand` v samostatnom príkaze je fakt, že všetky návštevy vykonané v rámci príkazu `InterceptJavaScriptCommand` majú vo výslednej databáze obsahujúcej namerané dáta nastavenú rovnakú hodnotu atribútu `visit_id`, vďaka čomu nie je pri následnej interpretácii nutné párovať návštevy podstránok napríklad podľa domény, pretože návštevy podstránok danej stránky môžu byť zoskupené podľa `visit_id`.

Pri spúšťaní príkazu určeného pre OpenWPM je vo všeobecnosti možné špecifikovať počet sekúnd n, po ktorých uplynutí bude príkaz ukončený bez ohľadu na to, či bol príkaz



Obr. 4.4: Časť schémy databázy používanej nástrojom OpenWPM

ukončený úspešne sám od seba alebo nie. Pri vykonaných meraniach bola táto hodnota v rámci použitia príkazu `InterceptJavaScriptCommand` nastavená na hodnotu $n = 60c$, $0 < c \leq 13$, kde c, reprezentuje počet podstránok danej stránky.

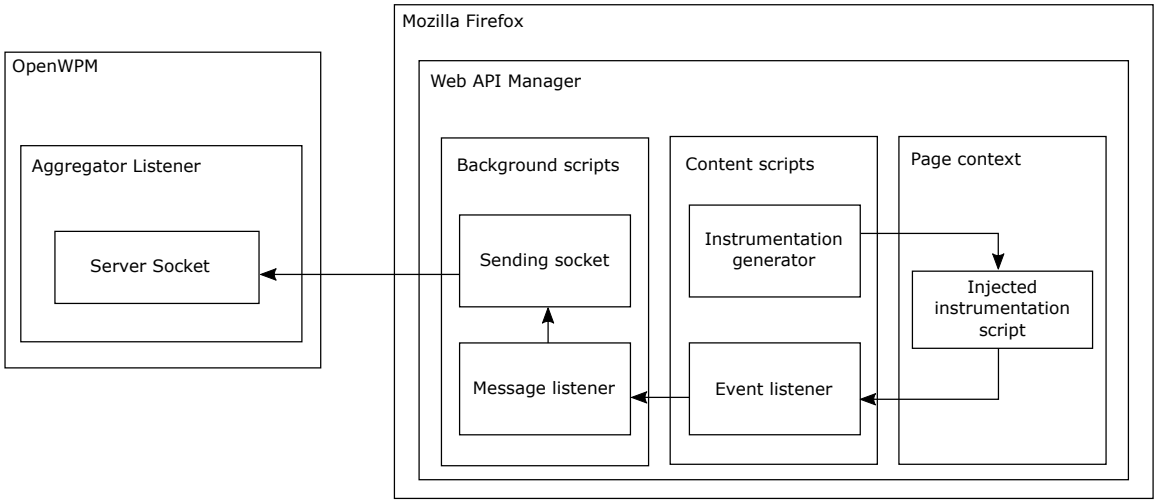
4.2 Úpravy rozšírenia Web API Manager

Pre využitie rozšírenia Web API Manager pre účely nášho merania bolo potrebné rozšírenie upraviť. Medzi najdôležitejšie úpravy rozšírenia Web API Manager patrili úpravy vedúce k tomu, aby sa zachytené volania metód rôznych JavaScriptových API neblokovali ale aby sa pri ich zachytení ich volanie zaznamenalo a následne bolo toto volanie delegované ich pôvodnému príjemcovi. Hlavné úpravy rozšírenia teda môžeme rozdeliť na dva samostatné podproblémy:

1. Implementácia mechanizmu pre zaznamenávanie informácií o volaní metód a
2. delegovanie volania metód ich pôvodnému príjemcovi.

Pre zaznamenávanie volaní jednotlivých metód JavaScriptových API využívame rozhranie agregátora dát, ktoré OpenWPM vystavuje pomocou soketu. Naša úloha pri implementácii zaznamenávania volaní metód z rozšírenia Web API Manager spočívala v pripojení sa na soket a následnom poslaní informácie o volaní príslušnej metódy. Pre dosiahnutie uvedeného sme rozšírenie Web API Manager doplnili o možnosť komunikovať cez sokety. Pri implementácii sme vychádzali z implementácie, ktorá je použitá pri predvolenom rozšírení

OpenWPM. Princíp fungovania záznamu informácií o volaných metódach je znázornený na obrázku 4.5.



Obr. 4.5: Princíp fungovania zaznamenávania volania metód zachytených rozšírením Web API Manager

Na obrázku 4.5 môžeme vidieť, že blok Page content, ktorý reprezentuje obsah stránky obsahuje blok s názvom Injected instrumentation script, ktorý reprezentuje skript injektovaný do každej stránky navštívenej s rozšírením Web API Manager. Tento skript je zodpovedný za generovanie Proxy objektov, nad ktorými následne stránka volá JavaScriptové funkcie tak, ako keby to boli klasické objekty rôznych JavaScriptových API. Inštrumentačný skript je injektovaný do každej stránky pri každej jej návšteve, pretože obsahové skripty (z angl. *content scripts*) nemajú prístup ku globálnemu objektu `window`, ktorý obsahuje referencie k rozhraniam rôznych API a ktoré sú v našom prípade nahradzované Proxy objektami.

V prípade, že v rámci injektovaného skriptu dôjde k detekcii volania nejakej metódy, skript injektovaný do obsahu stránky vyvolá udalosť so špeciálnym názvom. Obsahový skript je inštruovaný tak, aby na tieto udalosti reagoval poslaním správy, ktorá je následne odpočutá na úrovni skriptu bežiaceho na pozadí rozšírenia (po anglicky tzv. *background script*). Tento skript umožňuje rozšíreniu komunikovať s agregátorom dát pomocou soketu.

Uviedli sme, že okrem implementácie mechanizmu, pomocou ktorého môžeme informovať agregátor dát o volaní metódy v JavaScripte je potrebné delegovať volanie tejto metódy na jej pôvodného príjemcu. Tento krok je zabezpečený využitím JavaScript Reflect API[3].

Vo svojej bežnej verzii rozšírenie Web API Manager po inštalácií očakáva od užívateľa vykonanie prvotného nastavenia. V rámci toho môže užívateľ určiť konkrétne JavaScript API, ktoré sa budú pri návštevách webových stránok blokovať (resp. v našom prípade zaznamenávať prístup k týmto API a následne zavolať príslušnú funkciu tak, ako keby bola zavolaná na skutočnom objekte príslušného API a nie na Proxy objekte). Toto nastavenie je možné definovať odlišne pre rôzne webové stránky pomocou regulárneho výrazu odpovedajúcemu jednej konkrétnej alebo viacerým webovým stránkam. Pre účely našich meraní bolo potrebné upraviť rozšírenie tak, aby dochádzalo k odchytávaniu volaní všetkých metód,

[3] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Reflect

ktoré sú rozšíreniu známe a zároveň aby rozšírenie prístup k týmto metódam odchytávalo na každej webovej stránke.

4.3 Získavanie zoznamu implementovaných JavaScript API

V tejto časti uvádzame popis automatizácie získavania zoznamu metód, ktoré sú v príslušnej verzii prehliadača Mozilla Firefox implementované. Ako už bolo uvedené v sekcii 3.4, budovanie zoznamu implementovaných JavaScriptových API a ich metód v príslušnom prehliadači je založené na získavaní informácií z tzv. WebIDL súborov, ktoré reprezentujú aktuálny stav prehliadača.

Proces získavania zoznamu implementovaných JavaScript API pozostáva z nasledovných štyroch krokov:

1. Získanie URL adries WebIDL dokumentov,
2. stiahnutie WebIDL dokumentov z adries získaných v predošlom kroku,
3. prečítanie stiahnutých WebIDL súborov a získanie informácií o tzv. mixin rozhraniach a
4. opätovné prečítanie stiahnutých WebIDL súborov a získanie informácií pre generovanie ciest k jednotlivým metódam JavaScript API

Uvedený proces je plne automatizovaný pomocou vlastnej aplikácie naprogramovanej v jazyku JavaScript a vyvinutej pre behové prostredie Node.js. Syntaktická analýza stiahnutých WebIDL súborov je vykonávaná s použitím knižnice `webidl2.js`[4], ktorá po vykonaní syntaktickej analýzy daného WebIDL dokumentu umožňuje pristúpiť k príslušnému syntaktickému stromu.

V rámci uvedeného procesu môžeme vidieť dvojitý priechod WebIDL súbormi, konkrétne v treťom a štvrtom bode. Tento prístup sme zvolili kvôli tomu, že WebIDL súbory môžu obsahovať tzv. mixin rozhrania, ktoré pomáhajú eliminovať duplicity v definícii rozhraní v prípade, že niektoré rôzne rozhrania obsahujú atribúty alebo metódy s rovnakou signatúrou. Ako príklad môžeme uviesť rozhrania `HTMLInputElement` a `HTMLTextAreaElement`, ktorých časť metód a atribútov metód majú zhodné signatúry. Vďaka využitiu mixin rozhraní sú však tieto atribúty a metódy definované len raz, konkrétne v rámci mixin rozhrania s názvom `MozEditableElement`. Pri prvom priechode WebIDL súbormi sú zbierané informácie o mixin rozhraniach, pričom v rámci následného druhého priechodu čítania WebIDL dokumentov už máme vytvorenú databázu mixin rozhraní, ktorú môžeme využiť pri zostavovaní zoznamu metód patriacich k príslušným rozhraniam jednotlivých API.

Pri budovaní výsledného zoznamu metód, ktoré jednotlivé JavaScript API ponúkajú prechádzame syntaktický strom, pričom sa zaoberáme len tými položkami, ktoré reprezentujú metódy. Rozhrania definované vo WebIDL súboroch vo všeobecnosti obsahujú aj atribúty a metódy, ktoré sú určené len na volanie z interného systémového kódu. Takéto atribúty a metódy sú pomocou tzv. rozšírených atribútov (z angl. *extended attributes* označené ako `ChromeOnly`. Pre naše účely takto označené metódy nezahŕňame do výsledného zoznamu všetkých metód, pretože tieto metódy nie sú dostupné webovým stránkam.

V prípade, že niektoré zo stiahnutých WebIDL súborov neodpovedajú špecifikácii WebIDL súborov [25], tieto súbory nie sú analyzované a metódy rozhraní, ktoré sú v týchto

[4] https://github.com/w3c/webidl2.js

súboroch definované nie sú súčasťou výsledného zoznamu metód. Na prípadné chyby pri analýze WebIDL súborov je užívateľ upozornený. V takom prípade sa očakáva manuálna úprava stiahnutých WebIDL súborov a opätovné spustenie programu, pričom prvý a druhý krok sa už nevykonáva.

Niektoré časté odlišnosti od špecifikácie, ktoré sme pri analýze WebIDL súborov identifikovali riešime automaticky. Konkrétne, odstraňujeme všetky rozhrania, ktoré nemajú definované telo. Ďalej, v prípade, že identifikujeme návratový typ akejkoľvek operácie alebo atribútu ako tzv. sekvenčný typ (z angl. sequence type), zmeníme návratový typ príslušnej operácie alebo atribútu na všeobecnejší typ `object`. Vzhľadom na to, že informácie obsiahnuté vo WebIDL súboroch používame len pre účely zostavenia zoznamu metód rôznych API implementovaných vo webovom prehliadači, ani jedna z uvedených automatických zmien WebIDL súborov nedegraduje informácie, ktoré sú pre nás dôležité. Všetky ďalšie odchýlky, ktorými sa WebIDL súbory prehliadača Mozilla Firefox vymikajú špecifikovanému štandardu je potrebné napraviť manuálne. Príkladom takej odchýlky môže byť napríklad prítomnosť rôznych makier, ktorými bola v rámci WebIDL súborov spracovávaných pre účely našich meraní podmienená prítomnosť rôznych metód alebo vlastností rozhraní podľa toho, o akú verziu prehliadača Mozilla Firefox ide (rôzne verzie prehliadača môžu obsahovať rôzne rozhrania, prípadne rôzne metódy alebo vlastnosti rozhraní).

4.4 Abstrakcia nameraných dát

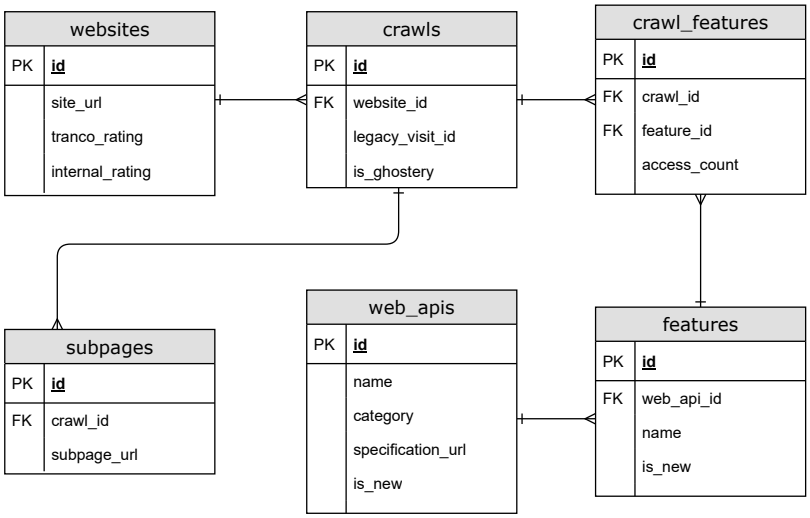
OpenWPM v rámci svojej základnej verzie ponúka hotovú infraštruktúru pre zaznamenávanie rôznych dát z meraní a obsahuje tiež preddefinované databázové tabuľky, do ktorých je možné ukladať informácie z meraní vykonávaných pomocou OpenWPM. Pre účely zaznamenávania prístupu k JavaScriptovým API sme využili vstavanú databázovú schému. Časť databázovej schémy, ktorá bola pre naše účely významná je naznačená na obrázku 4.4. Tabuľka site_visits slúži pre zaznamenanie návštev webových stránok. V rámci našich meraní je do tejto tabuľky ukladaný jeden záznam pre každú webovú stránku a jej podstránky, ktoré sme navštívili. Ďalej, tabuľka `javascript` uchováva informácie o prístupe ktorejkoľvek podstránky danej webovej stránky k nejakej metóde JavaScript API, pričom pre každé volanie metódy je v tabuľke `javascript` uložený samostatný riadok. Výhodou uvedeného prístupu k ukladaniu informácií o volaní jednotlivých metód je možnosť následnej analýzy postupnosti volaní metód danej stránky.

Pre zvýšenie rýchlosti analýzy nameraných dát sme namerané dáta abstrahovali, konkrétne, volania rovnakých metód sme zlúčili do jedného záznamu s explicitne uvedeným počtom volaní príslušnej metódy. Abstrakcia nameraných dát bola implementovaná v samostatnom programe vytvorenom v jazyku Python.

Štruktúra databázy pre ukladanie abstrahovaných dát je uvedená na obrázku 4.6. Tabuľka `websites` slúži na uloženie zoznamu všetkých stránok, ktoré sme pri meraní navštívili. Ďalej, tabuľka `crawls` slúži na uloženie informácie o tom, či bola webová stránka určená atribútom `website_id` navštívená v režime s blokátorom reklám alebo bez neho a uchováva tiež identifikátor návštevy z OpenWPM ako atribút `legacy_visit_id`. Tabuľka `subpages` je určená pre uloženie zoznamu podstránok navštívených pri meraní a tabuľka `crawl_features` slúži pre uloženie počtu volaní metód, ktoré boli v rámci merania určeného atribútom `crawl_id` invokované.

Štruktúra abstrahovaných dát tiež obsahuje tabuľky `features` a `web_apis`, ktoré slúžia na uloženie kategorizácie jednotlivých metód do rôznych API podľa toho, ktoré API danú metódu ponúka. Kategorizácia metód bola z časti prevzatá z doplnku Web API Manager a

metódy, ktoré neboli obsiahnuté v uvedenom doplnku boli kategorizované manuálne. Týmto spôsobom sme kategorizovali 211 nových metód do 38 nových API. Ďalších 410 nových metód sme kategorizovali do 31 API, ktoré existovali už aj v čase výskumu vykonaného v roku 2016 Petrom Snyderom. Zoznam API, ktoré sme identifikovali ako nové uvádzame v prílohe C.1.



Obr. 4.6: Schéma databázy abstrahovaných dát

Pri vykonávaní meraní pomocou príkazu `InterceptJavaScriptCommand` popísanom bližšie v časti 4.1.2 mohla nastať situácia, kedy bol v rámci režimu prehliadania stránok bez blokátora reklám a potom s blokátorom reklám navštívený rôzny počet podstránok. Rôzny počet podstránok mohol byť navštívený napríklad z dôvodu rôznej záťaže webového servera v dvoch rôznych okamihoch, kedy boli jednotlivé merania vykonávané (meranie s blokátorom reklám nemuselo byť vykonávané v rovnakom čase ako meranie bez blokátora) a v rámci jedného z meraní mohlo tým pádom dôjsť k predčasnému ukončeniu vykonávania príkazu z dôvodu vypršania časového limitu určeného pre odchytávanie volaní JavaScriptu. Pri abstrakcii dát sme preto abstrahovali len dáta zaznamenané na tých podstránkach, ktoré boli navštívené v rámci oboch režimov meraní. Uvedeným spôsobom sme získali informácie o používaní JavaScriptu na 9420 webových stránkach.

Pri abstrakcii dát došlo k redukcii informácií voči pôvodným dátam. Konkrétne, vďaka štruktúre tabuľky `javascript` z obrázka 4.4 je možné jednoznačne identifikovať počet invokovaných metód na daných konkrétnych podstránkach. Abstrahované dáta však takúto identifikáciu neumožňujú, pretože všetky údaje o počte metód invokovaných na ktorejkoľvek podstránke sú naviazané na domovskú stránku. Ďalšou informáciou, ktorá nie je v abstrahovaných dátach obsiahnutá je informácia o poradí, v ktorom boli jednotlivé metódy volané, pretože informácie o volaní príslušnej metódy boli v rámci abstrakcie dát agregované do jedného riadka s explicitne uloženým počtom volaní danej metódy.

Abstrakcia dát bola vykonaná na serveroch spoločnosti DigitalOcean. Konfigurácia serverov bola odlišná od konfigurácie serverov použitých pri meraní pomocou OpenWPM. Parametre Dropletov, ktoré sme použili pre abstrahovanie dát sú uvedené v tabuľke 4.2. Rozdiel, ktorý sme pri testovaní abstrakcie dát identifikovali ako kľúčový je typ pevného disku, pričom pri abstrakcii dát boli použité SSD disky s technológiou NVMe.

Názov parametra	**Hodnota parametra**
Typ CPU	Zdieľaný
Počet vCPU	2
Veľkosť operačnej pamäte	4 GB
Veľkost pevného disku	80 GB
NVMe SSD	Áno
Operačný systém	Ubuntu 18.04 (LTS) x64
Poloha datacentra	Frankfurt

Tabuľka 4.2: Parametre Dropletov, na ktorých bola vykonaná abstrakcia nameraných dát

Kapitola 5

Výsledky meraní

V tejto kapitole uvádzame výsledky a analýzu našich meraní realizovaných pomocou meracej platformy popísanej v kapitolách 3 a 4. Uvedeným spôsobom sme nazbierali dáta o používaní JavaScriptu na 10000 webových stránkach. V tejto časti analyzujeme používanie JavaScriptových API na 9420 webových stránkach, pretože na niektorých webových stránkach dochádzalo k rôznym chybám, kvôli ktorým boli tieto stránky vyradené z analyzovanej množiny. Príkladom stránok, ktoré sme vyradili z analyzovanej množiny sú napríklad webové stránky, ktoré sa nám podarilo navštíviť len v jednom z meracích režimov (buď s aktívnym blokátorom reklám alebo naopak len bez neho), prípadne sa nám stránku z rôznych dôvodov nepodarilo navštíviť vôbec.

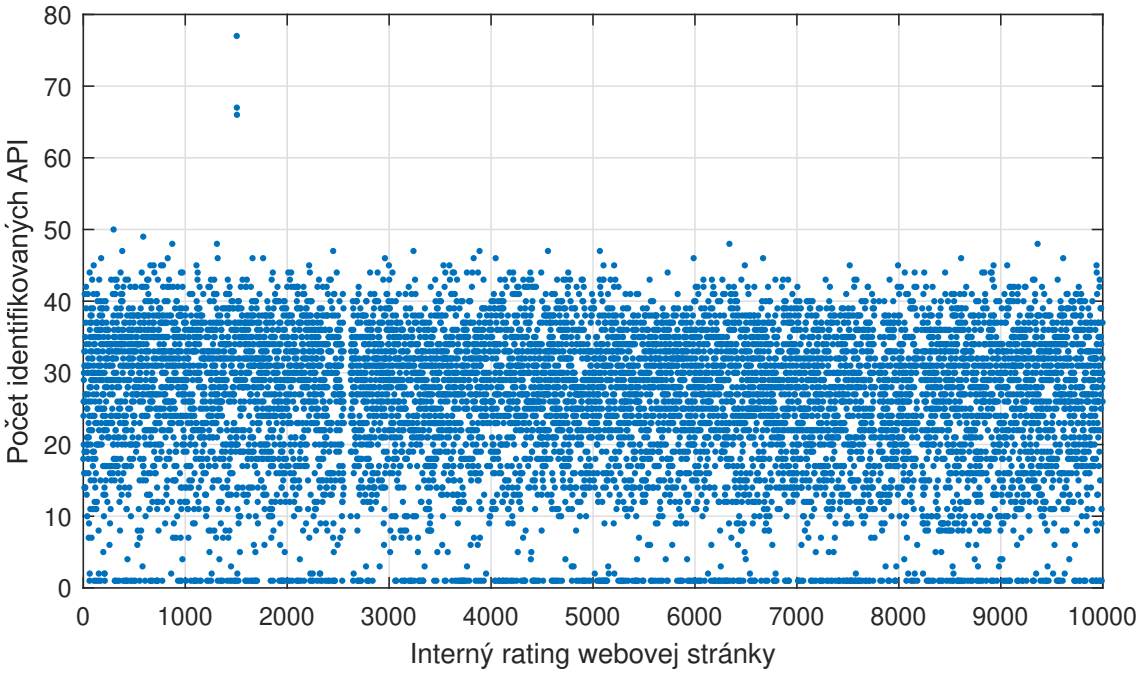
Naprieč touto kapitolou používame pojem *interný rating webovej stránky*. Interný rating webovej stránky predstavuje poradové číslo pridelené každej analyzovanej webovej stránke, pričom s rastúcou hodnotou interného ratingu rastie umiestnenie danej webovej stránky v zozname Tranco. V prípade, že by sme analyzovali len prvú a poslednú webovú stránku zoznamu Tranco, tak prvá webová stránka bude mať pridelenú hodnotu interného ratingu 1 a posledná hodnotu 2.

V nasledujúcej časti 5.1 uvádzame výsledky našich meraní, v rámci ktorých sme merali používanie JavaScriptových API na webových stránkach. Analyzujeme tiež volania metód a uvádzame informácie o tom, na akej časti webových stránok boli príslušné metódy volané. V časti 5.2 sa potom venujeme analýze používania nových API a analýze volania konkrétnych metód nových API. Ďalej uvádzame informácie o používaní nových metód, ktoré sú špecifikované v rámci starších API. V časti 5.3 sú uvedené informácie o meraní používania JavaScriptových API s aktívnym doplnkom Ghostery. Nakoniec v časti 5.4 porovnávame výsledky našich meraní s výskumom realizovanom v roku 2016.

5.1 Používanie JavaScriptových API

Na obrázku 5.1 uvádzame graf závislosti interného ratingu a počtu API, ktoré boli na príslušných webových stránkach identifikované. Pri meraniach sme na každej webovej stránke v priemere identifikovali 25, 93 API. Na obrázku 5.1 si môžeme všimnúť tri hodnoty, ktoré sa od ostatných bodov odlišujú počtom API, ktoré boli na daných stránkach identifikované. Po manuálnej analýze sme zistili, že tieto webové stránky majú hodnoty interného ratingu 1505, 1506 a 1507 a meranie na týchto webových stránkach prebiehalo súčasne na tom istom serveri. Dôvod takéhoto vysokého počtu zaznamenaných API na týchto stránkach sa nám nepodarilo identifikovať ani po bližšej analýze, v rámci ktorej sme dané stránky nav-

štívili manuálne. Pri manuálnej návšteve daných stránok sa nám taký vysoký počet API nepodarilo identifikovať.



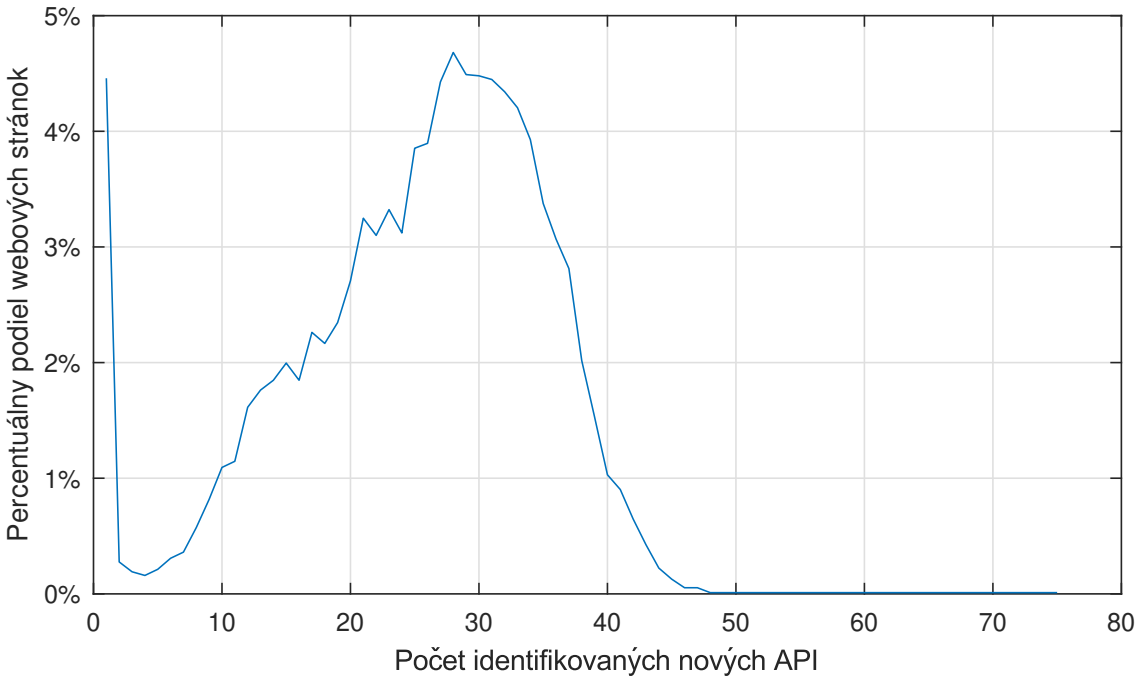
Obr. 5.1: Závislosť hodnoty interného ratingu voči počtu API identifikovaných na danej webovej stránke

Na obrázku 5.2 ďalej uvádzame vzťah medzi počtom API identifikovaných na rôznych stránkach a percentuálnou časťou webových stránok, na ktorých sme identifikovali príslušný počet API. Z obrázka 5.2 môžeme vidieť, že na najväčšej časti webových stránok, konkrétne na $4,68\%$ webových stránok, bolo identifikované použitie 28 rôznych API, pričom hodnota 28 odpovedá mediánu použitých API naprieč všetkými webovými stránkami.

Z uskutočnených meraní vyplýva, že niektoré API sú používané takmer na každej webovej stránke navštívenej v rámci našich meraní. Najpoužívanejším API, ktoré sme v rámci našich meraní zaznamenali je API špecifikované v rámci špecifikácie HTML[1]. Konkrétne, HTML API bolo identifikované na $99,86\%$ webových stránkach. Zoznam API, ktoré sme identifikovali na väčšine webových stránok, t. j. na viac ako 50% webových stránok, je uvedený v tabuľke 5.1. Kompletný zoznam API, ktoré sme identifikovali na viac, než 1% webových stránok je uvedený v prílohe B.1. Môžeme vidieť, že vo všeobecnosti patria medzi najčastejšie identifikované API najmä tie API, ktoré slúžia pre prácu s HTML dokumentom ako napríklad štandard HTML alebo DOM rôznych úrovní.

Medzi často používanými API tiež môžeme vidieť API pre prácu s asynchrónnymi požiadavkami posielanými na server z klientského JavaScriptu. Konkrétne, minimálne jedno z XMLHttpRequest a Fetch API boli identifikované na $89,72\%$ skúmaných webových stránok. Z tabuľky 5.1 vidíme, že staršie XMLHttpRequest API bolo identifikované na väčšej časti webových stránok aj napriek tomu, že modernejšie Fetch API poskytuje možnosť pohodlnejšej práce s asynchrónnymi požiadavkami. Súčasné použitie XMLHttpRequest a Fetch API bolo identifikované na $53,49\%$ webových stránkach. Do kategórie API pre prácu

[1] https://html.spec.whatwg.org/



Obr. 5.2: Vzťah počtu API voči relatívnej časti webových stránok, kde bol príslušný počet API identifikovaný

s asynchrónnymi požiadavkami posielanými na server patrí aj Beacon API, ktoré slúži pre posielanie takých požiadaviek, ktoré nevyžadujú odpoveď servera (napríklad pre posielanie rôznych analytických dát). Použitie Beacon API bolo identifikované na 54, 64% webových stránok.

V tabuľke 5.1 tiež môžeme vidieť, že použitie API s názvom Timing Control for Script-Based Animations bolo identifikované na 73, 45% analyzovaných webových stránkach. Toto API dáva webovým vývojárom možnosť pracovať s animáciami riadenými pomocou JavaScriptu a poskytuje metódy, ktoré sú alternatívou k metódam `window.setInterval` a `window.setTimeout`. Konkrétne, súčasťou Timing Control for Script-Based Animations API je metóda `window.requestAnimationFrame`, pomocou ktorej je možné eliminovať problém so správnym nastavením hodnoty obnovovacieho intervalu pri animáciách, ktorý vzniká pri riadení prekreslovania animácií pri použití metód `window.setInterval` a `window.setTimeout` [15]. Z výsledkov našich meraní vyplýva, že obe metódy `window.setTimeout` a `window.setInterval` sú pri používaní populárnejšie, než metóda `window.requestAnimationFrame`. Vyššia obľúbenosť metód `window.setTimeout` a `window.setInterval` však môže byť zapríčinená tým, že ich použitie nie je vždy spojené len s prekreslovaním animácií riadených JavaScriptom. Obe metódy môžu byť použité pri širšej škále prípadov užitia, vo všeobecnosti vždy, keď potrebujeme, aby prehliadač vykonal nejakú akciu až o nejaký čas [14]. V tabuľke 5.2 uvádzame percentuálnu časť webových stránok, na ktorých sme identifikovali použitie uvedených metód.

5.1.1 Najpoužívanejšie metódy pre prácu s HTML a DOM

Vzhľadom k tomu, že najčastejšie identifikovanými API boli pri našich meraniach API určené primárne na pácu so štruktúrou HTML dokumentov a s DOM, v tejto časti uvádzame

Názov API	Percentuálna časť webových stránok
HTML	99,86%
DOM Level 1	95,24%
DOM	95,11%
DOM Level 2: Events	94,98%
Selectors API (Level 1)	93,85%
DOM Level 2: Core	93,66%
DOM Level 2: Style	89,48%
XMLHttpRequest	88,86%
DOM Level 3: Core	87,12%
HTML: Web Storage	86,80%
CSS Object Model (CSSOM)	84,39%
High Resolution Time (Level 2)	80,01%
Web Cryptography API	79,89%
HTML: DOM	78,58%
DOM Level 4	77,26%
HTML 5	74,17%
Timing Control for Script-Based Animations	73,45%
HTML: Channel Messaging	70,69%
Performance Timeline	64,11%
CSSOM View Module	62,76%
HTML: Canvas Element	59,97%
URL	55,41%
Beacon	54,64%
Fetch	54,44%
Intersection Observer API	52,70%
User Timing (Level 2)	50,23%

Tabuľka 5.1: Zoznam API, ktoré boli v rámci našich meraní identifikované na väčšine webových stránok

analýzu najčastejšie používaných metód v rámci týchto API. Konkrétne, v tejto časti analyzujeme volania metód vypísaných v tabuľke 5.3.

Najčastejšie identifikovanou metódou bola naprieč analyzovanými webovými stránkami metóda `Document.createElement` špecifikovaná v DOM API, ktorá slúži na dynamické vytváranie HTML elementov na webovej stránke. Použitie metódy `Document.createElement` sme identifikovali na viac než 95% webových stránkach, pričom priemerný počet volaní metódy `Document.createElement` na jednom webe je $2872,8$ krát.

Druhou najčastejšie identifikovanou metódou je v rámci vybraných API metóda pre obsluhu rôznych udalostí `EventTarget.addEventListener`, ktorá tvorí pomerne esenciálnu súčasť jazyka JavaScript. Použitie metódy `EventTarget.addEventListener` definovanej v DOM Level 2 API v časti definujúcej API pre prácu s udalosťami (v našej kategorizácii považujeme túto metódu za samostatné API s názvom DOM Level 2 API: Events) sme identifikovali na $94,98\%$ stránkach.

Treticu najčastejšie identifikovaných použití metód v rámci API uvedených v tabuľke 5.3 uzatvára metóda `Node.appendChild`, ktorá umožňuje tvorcom webových stránok pripájať HTML elementy k existujúcim elementom na danej webovej stránke. Použitie metódy

Názov metódy	Percentuálna časť webových stránok
`window.setTimeout`	93,62%
`window.setInterval`	82,4%
`window.requestAnimationFrame`	73,37%

Tabuľka 5.2: Porovnanie množstva webových stránok, na ktorých boli identifikované metódy používané pre obnovu snímku animácie vykreslovanej pomocou JavaScriptu

Názov API	Percentuálna časť webových stránok
HTML	99,86%
DOM Level 1	95,24%
DOM	95,11%
DOM Level 2: Events	94,98%
Selectors API (Level 1)	93,85%
DOM Level 2: Core	93,66%
DOM Level 3: Core	87,12%
HTML: DOM	78,58%
DOM Level 4	77,26%
HTML 5	74,17%
HTML: Canvas Element	59,97%

Tabuľka 5.3: Zoznam API určených pre manipuláciu so štruktúrou HTML dokumentu a DOM, ktoré sme vybrali pre bližšiu analýzu používania metód

`Node.appendChild` sme identifikovali na 94, 45% webových stránok. Kompletný zoznam najpoužívanejších metód špecifikovaných v rámci API uvedených v tabuľke 5.3, ktoré sme identifikovali aspoň na 1% webových stránok uvádzame v prílohe D.1.

V tejto časti tiež analyzujeme použitie metód definovaných v Selectors API (Level 1), ktoré autorom webových stránok zjednodušujú získavanie inštancií elementov daného dokumentu pomocou CSS selektorov pre ďalšiu manipuláciu pomocou JavaScriptu. Konkrétne, analyzujeme metódy `querySelector` a `querySelectorAll`, ktoré sú v rámci tohto API definované v rozhraniach objektov `Document`, `Element` a `DocumentFragment`. Hlavným rozdielom medzi metódami `querySelector` a `querySelectorAll` je návratová hodna daných funkcií, pričom `querySelector` vracia prvý nájdený element, ktorý vyhovuje danému selektoru (prípadne skupine selektorov) a metóda `querySelectorAll` vracia zoznam všetkých elementov, ktoré danému selektoru (prípadne skupine selektorov) vyhovujú.

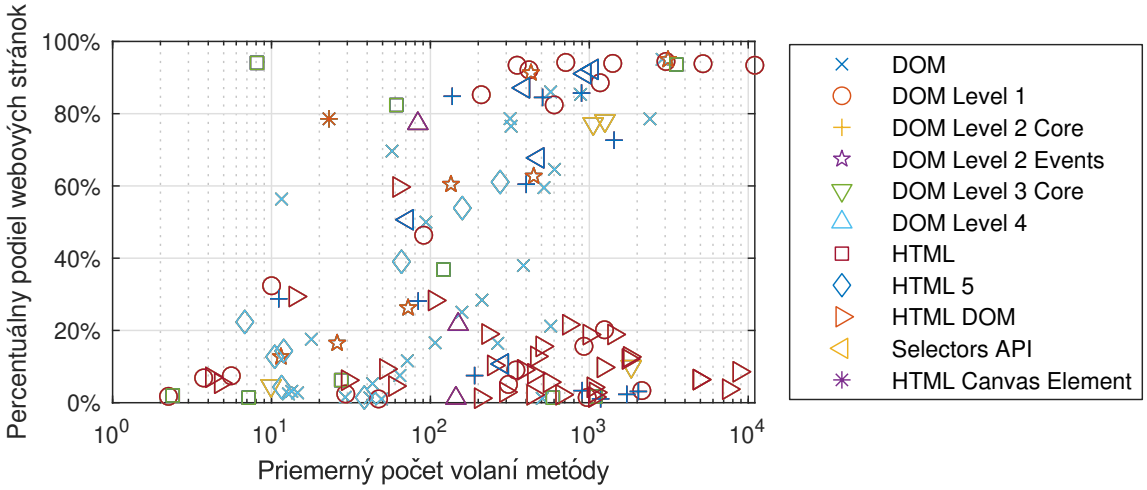
Použitie metód definovaných v Selectors API (Level 1) bolo identifikované na necelých 94% analyzovaných webových stránkach. Množstvo webových stránok, na ktorých sme použitie daných metód zaznamenali uvádzame spolu s priemernými počtami volaní v tabuľke 5.4. V tabuľke 5.4 môžeme vidieť, že najčastejšie identifikovanou metódou je v rámci Selectors API (Level 1) metóda `querySelectorAll` definovaná na rozhraniach `Document` a `Element`, pričom môžeme vidieť, že priemerný počet invokácií týchto metód je približne podobný. Zároveň môžeme konštatovať, že použitie metódy `querySelectorAll` prevažuje použitie metódy `querySelector`.

Na obrázku 5.3 môžeme vidieť zobrazenie závislosti priemerného počtu identifikovaných volaní metód a percentuálnej časti webových stránok, na ktorých bolo príslušné volanie zaznamenané, pričom jednotlivé body na obrázku 5.3 reprezentujú jednotlivé metódy príslušných API. Z obrázka 5.3 môžeme vidieť, že väčšina bodov reprezentujúcich rôzne me-

Názov API	P	U
`Document.querySelectorAll`	92,26%	1043,51
`Element.querySelectorAll`	91,07%	938,97
`Document.querySelector`	87,14%	386,59
`Element.querySelector`	67,76%	475,60
`DocumentFragment.querySelectorAll`	50,64%	71,68
`DocumentFragment.querySelector`	10,72%	284,37

Tabuľka 5.4: Používanie metód definovaných v Selectors API (Level 1), kde stĺpec U značí percentuálnu časť webových stránok, kde bolo identifikované použitie príslušnej metódy a P značí priemerný počet volaní na každej stránke, kde bola invokácia príslušnej metódy identifikovaná

tódy daných API je zobrazených v časti, ktorá napovedá, že invokácie väčšiny metód boli identifikované rádovo v počte stoviek až tisícov.



Obr. 5.3: Vzťah medzi priemerným počtom volaní rôznych metód pre prácu s HTML a DOM (v logaritmickej mierke) a percentuálnou časťou webových stránok, na ktorých boli volania daných metód identifikované

5.1.2 Analýza volania metód ďalších vybraných API

Pri interpretácií výsledkov našich meraní sme zaznamenali použitie metód rôznych ďalších API, ktoré boli použité na pomerne veľkej časti webových stránok a preto im vyčleňujeme samostatný priestor.

Použitie API s názvom High Resolution Time Level 2 sme identifikovali na viac než 80% webových stránok, pričom prakticky takmer vždy išlo na daných stránkach o použitie metódy `Performance.now`. Uvedená metóda slúži na získanie pomerne presnej informácie o čase, ktorý uplynul od počiatku vzniku daného dokumentu, resp. o čase, ktorý uplynul od tzv. *time origin*. Metóda `Performance.now` môže byť používaná aj v oblasti identifikácie užívateľov, pretože umožňuje získať pomerne presné hodnoty, ktoré môžu tvoriť časť identifikácie daného užívateľa [14, 17]. V čase písania tejto práce môžeme vidieť, že tvorcovia

prehliadača Mozilla Firefox si možnosť zneužitia tejto metódy pravdepodobne uvedomujú a redukujú presnosť hodnôt vrátených metódou `Performance.now`.

Ďalším API, ktorého použitie sme identifikovali na pomerne veľkej časti analyzovaných webových stránok je Web Cryptography API, ktorého použitie sme identifikovali na viac než 79% stránok. Podobne ako pri High Resolution Time Level 2 a metóde `Performance.now`, v podstate všetky použitia Web Cryptography API spočívali vo volaní jednej metódy s názvom `Crypto.getRandomValues` slúžiacej pre generovanie pseudonáhodných čísel, ktoré sú kryptograficky silné. V rámci našich meraní bola metóda `Crypto.getRandomValues` v priemere volaná 275 krát na každej stránke, kde sme identifikovali jej použitie. Druhou najpoužívanejšou metódou definovanou v Cryptography API je metóda `SubtleCrypto.digest`, ktorej volania sme identifikovali na necelých 239 webových stránkach, čo reprezentuje necelé $2,5\%$ všetkých analyzovaných webových stránok.

V tejto časti tiež uvádzame informácie o používaní metód tých API, ktoré slúžia pre prácu s objektovým modelom v rámci CSS. V tabuľke 5.1 môžeme vidieť, že na relatívne veľkej časti webových stránok sme zaznamenali použitie API definovaných štandardami CSS Object Model (CSSOM) a CSSOM View Module. Najčastejšie identifikované metódy definované v rámci týchto dvoch uvedených API uvádzame v tabuľke 5.5. Môžeme vidieť, že ako najčastejšie invokovanú metódu sme identifikovali `Element.getBoundingClientRect`, ktorá slúži na získanie informácií o viditeľnej časti stránky voči danému elementu. V tabuľke 5.5 môžeme vidieť, že priemerný počet volaní tejto metódy v rámci jedného webu je voči ostatným metódam dvoch skúmaných API pomerne vysoký. Pričínou by mohli byť napríklad rôzne implementácie napodobňujúce správanie novšieho Intersection Observer API, ktoré zahŕňajú cyklické použitie metódy `Element.getBoundingClientRect` [27].

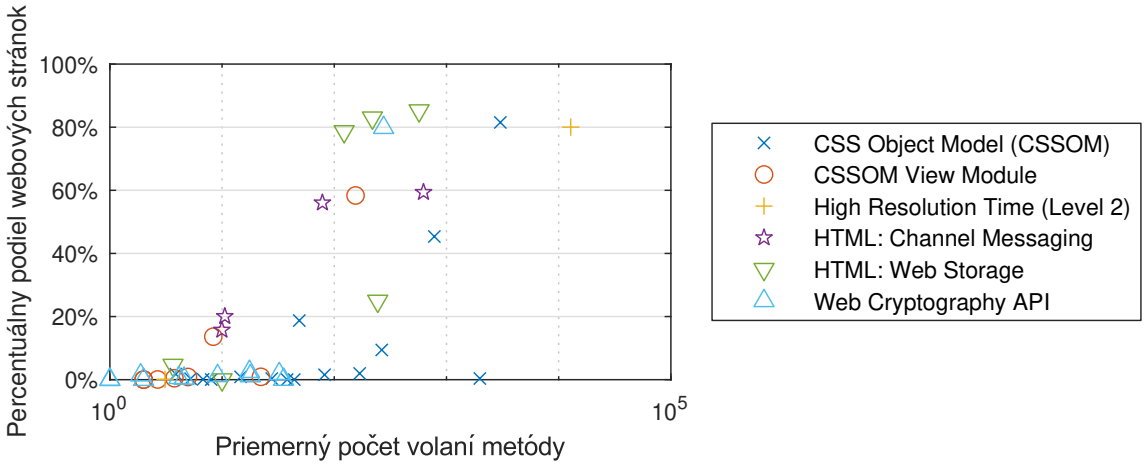
Názov API	**API**	**P**	**U**
`Element.getBoundingClientRect`	CSS Object Model (CSSOM)	81,51%	3020,60
`Element.getClientRects`	CSS Object Model (CSSOM)	45,36%	779,81
`MediaQueryList.addListener`	CSS Object Model (CSSOM)	18,73%	48,78
`window.matchMedia`	CSSOM View Module	58,37%	154.76
`window.scrollTo`	CSSOM View Module	13,63%	8.36

Tabuľka 5.5: Používanie metód definovaných v CSS Object Model (CSSOM) a CSSOM View Module (LVL 1), kde stĺpec P značí percentuálnu časť webových stránok, kde bolo identifikované použítie príslušnej metódy a U značí priemerný počet volaní na každej stránke, kde bola invokácia príslušnej metódy identifikovaná

Na pomerne veľkej časti webových stránok sme identifikovali volania metód definovaných v rámci časti špecifikácie HTML, ktorá sa venuje špecifikácii metód pre prácou s dátami uloženými v prehliadači. Toto API je v rámci našej kategorizácie zaradené do samostatného API s názvom HTML: Web Storage. Z analýzy nameraných dát vyplýva, že najčastejšie invokovanou metódou tohto API je metóda `Storage.prototype.getItem`, ktorá bola identifikovaná takmer na každej stránke, kde sme použitie tohto API identifikovali. Konkrétne, volanie metódy `Storage.getItem` sme identifikovali na $85,25\%$ stránkach. Ďalšími metódami, ktoré boli invokované na veľkej časti webových stránok využívajúcich toto API sú metódy `Storage.setItem` a `Storage.removeItem`.

Podobne ako v predošlej časti, kde sme diskutovali používanie API používaných najmä pre manipuláciu s HTML dokumentom a DOM, aj pre API diskutované v tejto časti uvádzame na obrázku 5.9 vzťah závislosti priemerného počtu volaní rôznych metód daných

API (v logaritmickej mierke) a percentuálnou časťou webových stránok, na ktorých boli volania príslušných metód identifikované. Môžeme vidieť, že na rozdiel od API pre prácu s DOM a HTML v rámci každého z ďalších vybraných API existuje len zopár metód, ktoré sú často používané na veľkej časti stránok a ostatné metódy daného API sú používané v pomerne malej miere. Zároveň môžeme vidieť, že priemerné počty volaní veľkej časti metód, ktoré sú používané na väčšine webových stránok sa pohybujú v rozmedzí niekoľko stoviek až desiatok tisíc.



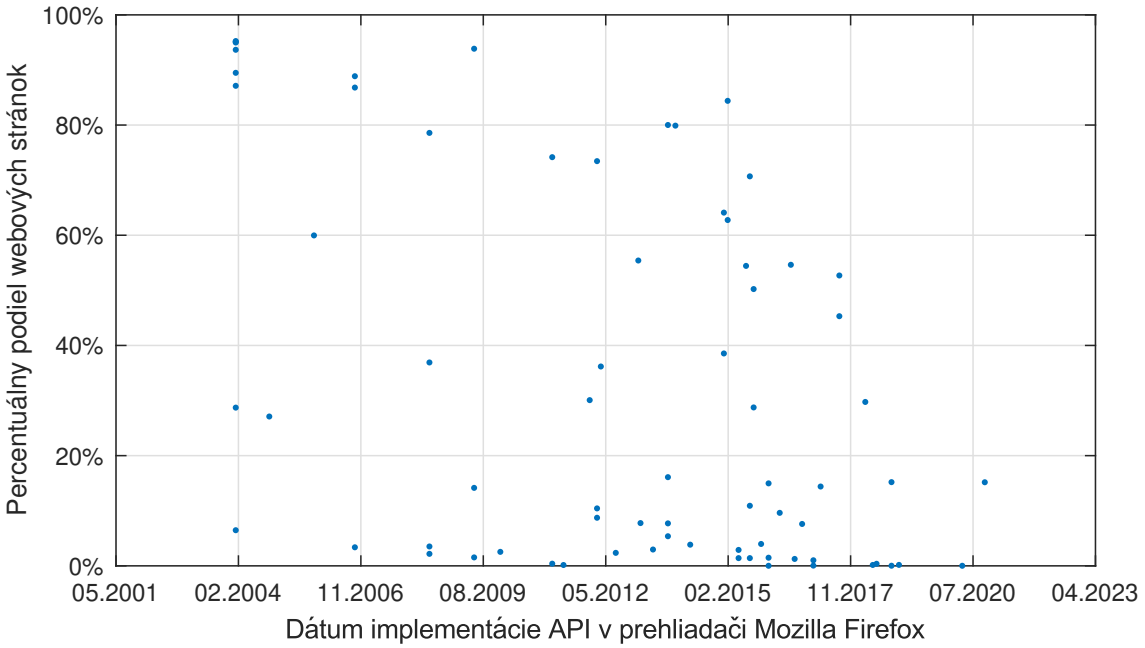
Obr. 5.4: Vzťah medzi priemerným počtom volaní rôznych metód vybraných API (v logaritmickej mierke) a percentuálnou časťou webových stránok, na ktorých boli volania daných metód identifikované

5.1.3 Vzťah medzi časom implementácie a obľúbenosťou API

Pri analýze používania rôznych API sme skúmali aj vzťah medzi tým, ako dlho sú dané API dostupné v prehliadači a tým, aká je ich popularita pri používaní. Na obrázku 5.5 môžeme vidieť vzťah medzi časom implementácie daného API v prehliadači a medzi tým, na akej časti webových stránok sme použitie príslušného API zaznamenali. Február roka 2004 odpovedá prvému vydaniu prehliadača s názvom Mozilla Firefox a v rámci tohto prvého vydania bolo implementovaných niekoľko API. Je dôležité uviesť, že tieto API už existovali a boli špecifikované už aj pred tým, než boli implementované v prehliadači Mozilla Firefox. V dnešnej dobe môžeme sledovať trend tzv. živých štandardov (z angl. *living standards*), ktoré sú stále vyvíjané a autori prehliadačov implementujú jednotlivé API podľa toho, ako sa mení živý štandard daného API. Príkladom takého API môže byť napríklad Streams API, ktorého špecifikácia má podobu živého štandardu[2], ktorý je stále vyvíjaný, avšak podporu tohto API môžeme nájsť v bežných prehliadačoch už aj dnes. Ďalším príkladom môže byť aj štandard HTML, ktorý mal najprv podobu štandardu vydávaného postupne v rôznych verziách organizáciou W3C, pričom dnešný HTML štandard[3] je spravovaný skupinou WHATWG prostredníctvom živého štandardu [45].

Keď sa zameriame na obrázok 5.5, môžeme vidieť, že niektoré pomerne staré API sú používané na veľkej časti webových stránok. Medzi tieto API patria najmä API pre prácu

[2] https://streams.spec.whatwg.org/
[3] https://html.spec.whatwg.org/



Obr. 5.5: Vzťah implementácie API v prehliadači Mozilla Firefox a obľúbenosťou API

s DOM a s HTML, ktorým sa venujeme v časti 5.1.1. Ďalším príkladom pomerne starého API, ktoré je používané na veľkom množstve webových stránok je už tiež diskutované XMLHttpRequest API pre prácu s asynchrónnou komunikáciou s webovými servermi implementované v roku 2006 alebo HTML Web Storage API pre ukladanie dát na strane webového prehliadača ponúkajúc alternatívu k ukladaniu pomocou cookies. HTML Web Storage API bolo v prehliadači Mozilla Firefox implementované v roku 2006 a v rámci našich meraní sme identifikovali volanie metód tohto API na 86, 80% webových stránkach.

Z našich meraní vyplýva, že okrem starších a vo veľkej miere používaných API existuje aj kategória takých starších API, ktoré sú používané pomerne malou časťou webových stránok. K takýmto API môžeme zaradiť napríklad Scalable Vector Graphics (SVG) 1.1 (Second Edition) API, ktoré bolo v prehliadači Mozilla Firefox implementované v roku 2006 a pri analýze našich meraní bolo toto API identifikované len na 3, 51% webových stránok. Ďalším starším a málo používaným API je napríklad DOM Level 3: XPath API slúžiace pre prístup k DOM pomocou XPath. Použitie DOM Level 3: XPath API, ktoré bolo v prehliadači Mozilla Firefox tiež implementované v roku 2006 sme identifikovali len na 2, 18% webových stránok.

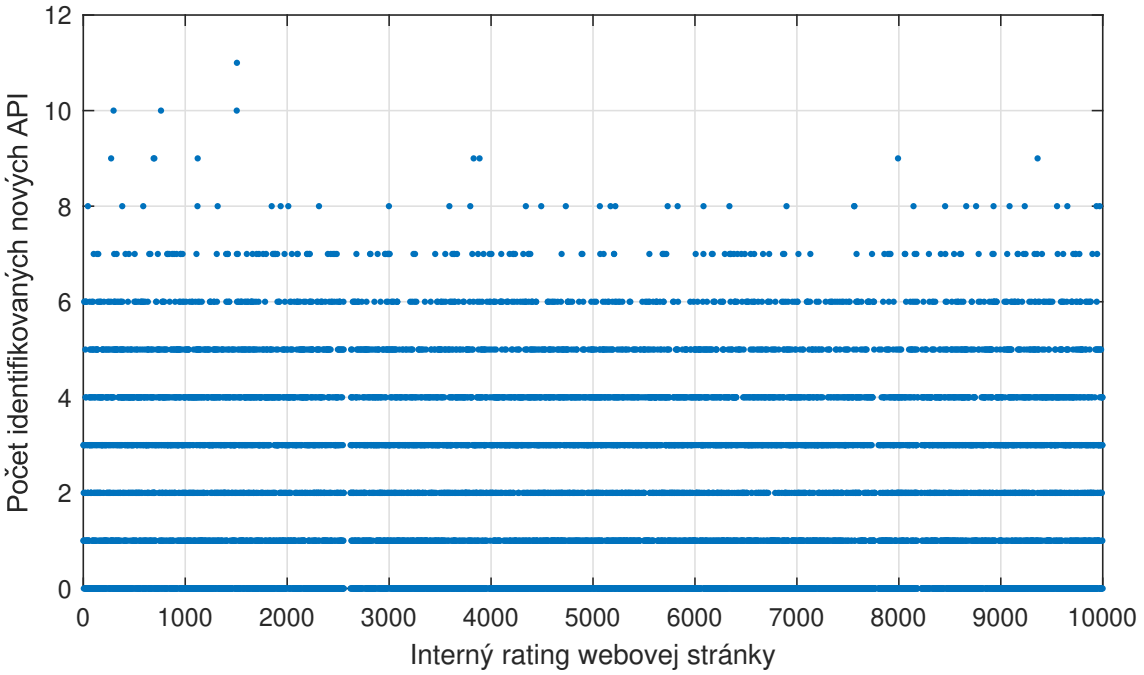
Na druhej strane môžeme nájsť aj novšie API, ktorých použitie sme identifikovali na pomerne veľkej časti webových stránok. Do tejto skupiny môžeme zaradiť napríklad Intersection Observer API diskutované v časti 2.2.2, ktoré bolo v prehliadači Mozilla Firefox implementované v roku 2017, pričom invokáciu metód poskytnutých týmto API sme zaznamenali na viac, než polovici webových stránok, konkrétne na 52, 07% webových stránok. Ďalším príkladom API, ktoré považujeme za relatívne nové a ktoré je aj napriek tomu použité na 54, 64% webových stránok je Beacon API, ktoré slúži pre asynchrónne a neblokujúce posielanie požiadaviek na webový server. Posledným API, ktoré uvedieme ako príklad je API pre prácu s CSSOM (CSS Object Model), pomocou ktorého je možné pracovať s CSS

daného HTML dokumentu podobne, ako s DOM a umožňuje dynamicky meniť štýly daného dokumentu.

5.2 Analýza používania nových API

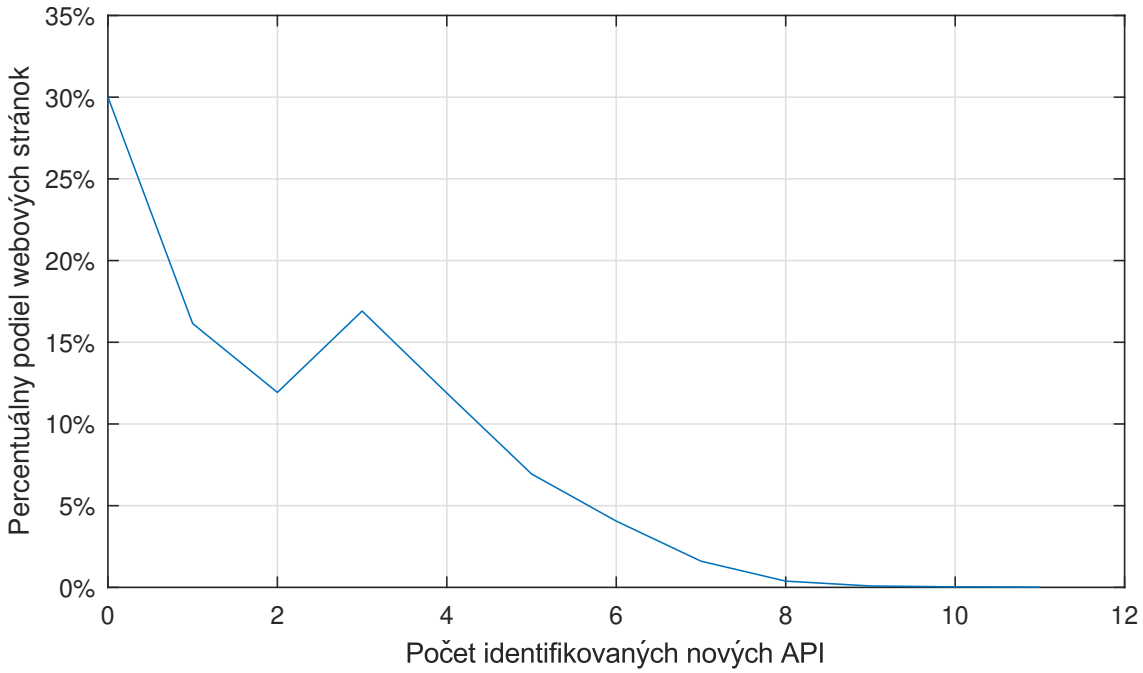
V tejto časti práce sa venujeme analýze používania nových API, pričom nové API definujeme ako také API, ktoré neboli zahrnuté v meraniach výskumu publikovanom v roku 2016 Petrom Snyderom et al. Ako už bolo spomenuté vyššie v texte, v rámci kategorizácie metód do jednotlivých API, ktorú sme vykonávali manuálne sme kategorizovali 211 nových metód do 38 nových API a ďalších 410 metód do starších API, ktoré boli v prehliadači implementované už aj v čase spracovávania výskumu publikovanom v roku 2016. Zoznam API, ktoré sme identifikovali ako nové uvádzame v prílohe C.1.

Na obrázku 5.6 uvádzame vzťah medzi interným ratingom webovej stránky a počtu nových API, ktoré sme na danej stránke identifikovali. Priemerný počet nových API, ktoré sme naprieč analyzovanými webovými stránkami identifikovali je rovný hodnote $2,01$. V prípade, že priemerný počet nových API počítame len zo stránok, kde sme identifikovali použitie nenulového počtu nových API, tak priemerný počet nových API identifikovaných na jednej stránke stúpne na hodnotu $3,04$. Na obrázku 5.7 ďalej uvádzame vzťah počtu použitých nových API voči percentuálnej časti stránok, ktoré daný počet nových API využívajú.



Obr. 5.6: Závislosť hodnoty interného ratingu voči počtu nových API identifikovaných na danej webovej stránke

V tabuľke 5.6 uvádzame zoznam nových API, ktoré sme identifikovali na viac, než 1% webových stránok. Z tabuľky 5.6 môžeme vidieť, že najčastejšie identifikovaným novým API je Intersection Observer API, ktoré bolo v prehliadači Mozilla Firefox implementované vo verzii 55 vydanej v auguste 2017. Hlavná myšlienka, ktorá stála za vznikom tohto API už bola diskutovaná v časti 2.2.2 a na tomto mieste len pripomenieme, že ide o API po-



Obr. 5.7: Vzťah počtu použitých nových API voči relatívnej časti webových stránok, kde bol príslušný počet noých API identifikovaný

užívané pre asynchrónne získavanie informácií o viditeľnosti a pozícii daného elementu na webovej stránke voči jeho rodičovským elementom, prípadne voči viditeľnej časti stránky (po anglicky nazývanej ako *viewport*). V priebehu obdobia necelých štyroch rokov od implementácie v prehliadači našlo toto API využitie na necelých 53% webových stránok.

Druhým najčastejšie identifikovaným novým API je Background Tasks API, ktorého použitie sme identifikovali na 41, 97% webových stránok. Hlavným cieľom Background Tasks API je poskytnúť vývojárom webových stránok možnosť zvýšenia užívateľskej prívetivosti ich webových stránok tým, že umožňuje rôzne úlohy zaradiť do fronty, z ktorej sú tieto úlohy vyberané a spracované až v momente, keď prehliadač deteguje, že má na vykonanie danej úlohy dosť voľného času. Týmto spôsobom je chránený plynulý beh cyklu udalostí (z angl. *event loop*), ktorého prípadné spomalenie môže viesť k dlhším odozvám danej webovej stránky a k zníženej spokojnosti užívateľov. Rovnako ako Intersection Observer API, Background Tasks API bolo v prehliadači Mozilla Firefox implementované od verzie 55 vydanej v auguste roku 2017.

V tabuľke 5.6 môžeme vidieť, že tretím najpoužívanejším API je Resize Observer API, ktoré tvorcom webových stránok umožňuje reagovať na zmeny veľkosti zobrazených elementov, ktoré sú zapríčinené napríklad obrátením obrazovky mobilného zariadenia. Existuje viacero spôsobov, ako takéto zmeny detegovať, Resize Observer API však prináša možnosť detekcie takýchto zmien bez príliš veľkého zaťaženia prehliadača, ktoré by mohlo nastať napríklad cyklickým volaním metódy `window.matchMedia`. Použitie Resize Observer API sme identifikovali na necelej tretine analyzovaných webových stránok, ktoré sme analyzovali.

Tabuľka 5.6 obsahuje API, ktoré sme nazvali Not specified yet. API s týmto názvom v skutočnosti neexistuje a slúžilo nám len pre kategorizáciu metód, ktoré zatiaľ nie sú súčasťou žiadnej špecifikácie. Metódam, ktoré sme kategorizovali do tejto skupiny sa venujeme v nasledovnej časti.

Názov API	P
Intersection Observer API	52,70%
Background Tasks API	41,97%
Resize Observer API	29,75%
Navigation Timing Level 2	16,09%
Media Session API	15,17%
Service Workers	14,18%
Permissions API	9,62%
Not specified yet	6,00%
Resource Timing Level 2	1,40%
Web Animations API	1,25%
Storage API	1,01%

Tabuľka 5.6: Zoznam nových API, ktoré boli v rámci našich meraní identifikované na viac, než webových stránok, kde U značí percentuálnu časť webových stránok používajúcich dané API

5.2.1 Analýza volaní metód nových API

Vo všeobecnosti môžeme povedať, že Intersection Observer API a Resize Observer API poskytujú funkcionalitu, ktorá je veľmi podobná. Obe API sú súčasťou množiny viacerých API, ktoré sú založené na použití modelu Observer, ktorý umožňuje detegovať rôzne zmeny v rámci webovej stránky. Všetky tieto API, vrátane Intersection Observer API a Resize Observer API, definujú metódy `observe`, `unobserve` a `disconnect`. Informácie o používaní týchto metód v rámci oboch diskutovaných API uvádzame v tabuľke 5.7. Môžeme vidieť, že v oboch prípadoch bolo najčastejšie identifikované použitie metódy `observe`. Zatiaľčo pri volaní metód Intersection Observer API môžeme vidieť, že priemerný počet volaní príslušnej metódy rastie s počtom webových stránok, na ktorých boli volania danej metódy identifikované, pri Resize Observer API to neplatí. Zaujímavým zistením je, že v prípade, že sme na nejakej webovej stránke identifikovali použitie metódy `ResizeObserver.unobserve`, bola volaná viac krát, než metóda `ResizeObserver.observe`, ktorej použitie by malo predchádzať použitiu `ResizeObserver.unobserve`.

Názov API	Názov metódy	U	P
`IntersectionObserver.observe`	Intersection Observer API	52,52%	364,53
`IntersectionObserver.unobserve`	Intersection Observer API	31,00%	119,33
`IntersectionObserver.disconnect`	Intersection Observer API	25,01%	27,82
`IntersectionObserver.takeRecords`	Intersection Observer API	16,82%	3582,98
`ResizeObserver.observe`	Resize Observer API	29,71%	55,45
`ResizeObserver.unobserve`	Resize Observer API	5,58%	277,43
`ResizeObserver.disconnect`	Resize Observer API	1,45%	15,92

Tabuľka 5.7: Používanie metód Intersection Observer API a Resize Observer API

Pri analýze volaní metód špecifikovaných v Background Tasks API sme zistili, že takmer všetky použitia tohto API identifikované našimi meraniami spočívali vo volaní metódy `requestIdleCallback` definovanej na rozhraní globálneho objektu `window`. V rámci našich meraní sme na pomerne malej časti webových stránok zaznamenali aj volania metód

`IdleDeadline.timeRemaining` a `window.cancelIdleCallback`, pričom volanie metódy `IdleDeadline.timeRemaining`, resp. `window.cancelIdleCallback` sme identifikovali na 3, 33%, resp. 1, 13% webových stránok.

Ako sme uviedli v predošlom texte, tabuľka 5.6 obsahuje API, ktoré sme nazvali Not specified yet a ktoré slúži len ako kategória, kam sme zaradili metódy, ktoré zatiaľ nie sú súčasťou žiadneho štandardu. Hoci sme do tejto skupiny kategorizovali niekoľko rôznych metód (konkrétne `Document.hasStorageAccess`, `Document.requestStorageAccess` a `HTMLElement.attachInternals`), v rámci našich meraní sme identifikovali len použitie metódy `Document.hasStorageAccess`, ktorá bola v prehliadači Mozilla Firefox dostupná od verzie 65 vydanej v januári roku 2019.

5.2.2 Analýza volania nových metód starších API

V tejto časti sa zameriavame na analýzu volaní metód, ktoré sú síce nové, avšak sú súčasťou starších API, ktoré boli vo webových prehliadačoch implementované už dávnejšie a neskôr boli rozšírené o nové metódy. V tabuľke 5.8 uvádzame informácie o používaní takýchto metód, pričom uvedenú tabuľku obmedzujeme len na tie metódy, ktorých použitie sme detegovali na viac, než 10% analyzovaných webových stránok. Dlhší zoznam nových metód starších API potom uvádzame v prílohe E.1, kde je možné nájsť informácie o používaní metód identifikovaných na aspoň 1% webových stránok.

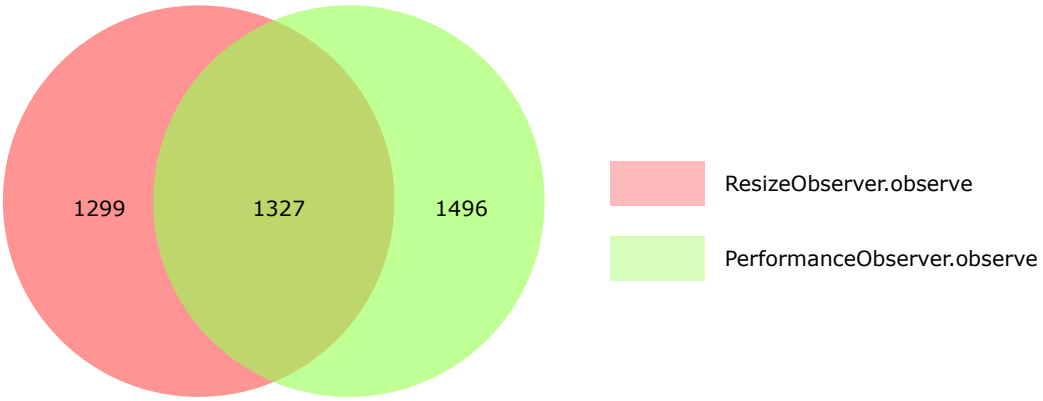
Názov metódy	**API**	**U**	**P**
DOMTokenList.supports	DOM	49,97%	93,66
Node.getRootNode	DOM	37,97%	384,91
PerformanceObserver.observe	Performance Timeline 2	27,89%	16,53
PerformanceObserverEntryList.getEntries	Performance Timeline 2	24,70%	51,45
Element.insertAdjacentElement	DOM	11,60%	71,60

Tabuľka 5.8: Používanie nových metód pridaných do starších API, stĺpec U značí percentuálnu časť webových stránok, kde bolo identifikované použitie príslušnej metódy a P značí priemerný počet volaní na každej stránke, kde bola invokácia príslušnej metódy identifikovaná, uvedené sú len tie metódy, ktoré boli identifikované na viac ako 10% webových stránok

V tabuľke 5.8 môžeme vidieť, že na najväčšej časti webových stránok sme identifikovali použitie metód, ktoré sú špecifikované ako súčasť štandardu DOM. Metódou, ktorej použitie sme identifikovali na najväčšej časti webových stránok je metóda `DOMTokenList.supports`, ktorá slúži pre získanie informácie ohľadom podpory daného atribútu príslušným elementom. Metóda `DOMTokenList.supports` je v prehliadači Mozilla Firefox implementovaná od verzie 49, ktorý bol vydaný v druhej polovici roka 2016 a môžeme vidieť, že aj napriek relatívne krátkemu času dostupnosti tejto metódy je táto metóda pomerne rozšírená, pretože ju používa takmer polovica navštívených webových stránok.

Medzi relatívne často používanými novými metódami starších API môžeme nájsť aj metódu `PerformanceObserver.observe`, ktorá je súčasťou skupiny API implementujúcich model Observer, o ktorej sme písali v časti 5.2.1. Metóda `PerformanceObserver.observe` slúži na začatie sledovania rôznych atribútov danej webovej stránky, ktoré vypovedajú o aktuálnom zaťažení na úrovni webového prehliadača ako napríklad čas načítania elementov príslušnej stránky alebo informácie o úlohách, ktoré dlho trvajú. Z tabuliek 5.8 a 5.6 môžeme vidieť, že množstvo stránok, ktoré využíva `PerformanceObserver.observe` je približne po-

dobné množstvu stránok využívajúcich `ResizeObserver.observe`. Z analýzy volaní týchto dvoch metód však vyplýva, že neexistuje príliš veľká závislosť, ktorá by potvrdzovala, že sú tieto dve metódy väčšinou volané na rovnakých webových stránkach. Na obrázku 5.8 uvádzame formou Vennovho diagramu počet webových stránok, ktoré používajú metódy `ResizeObserver.observe` a `PerformanceObserver.observe`, pričom môžeme vidieť, že použitie oboch týchto metód súčasne bolo identifikované na 1327 webových stránkach.



Obr. 5.8: Vennov diagram vyjadrujúci počty webových stránok, kde sme identifikovali volanie metód `ResizeObserver.observe` a `PerformanceObserver.observe`

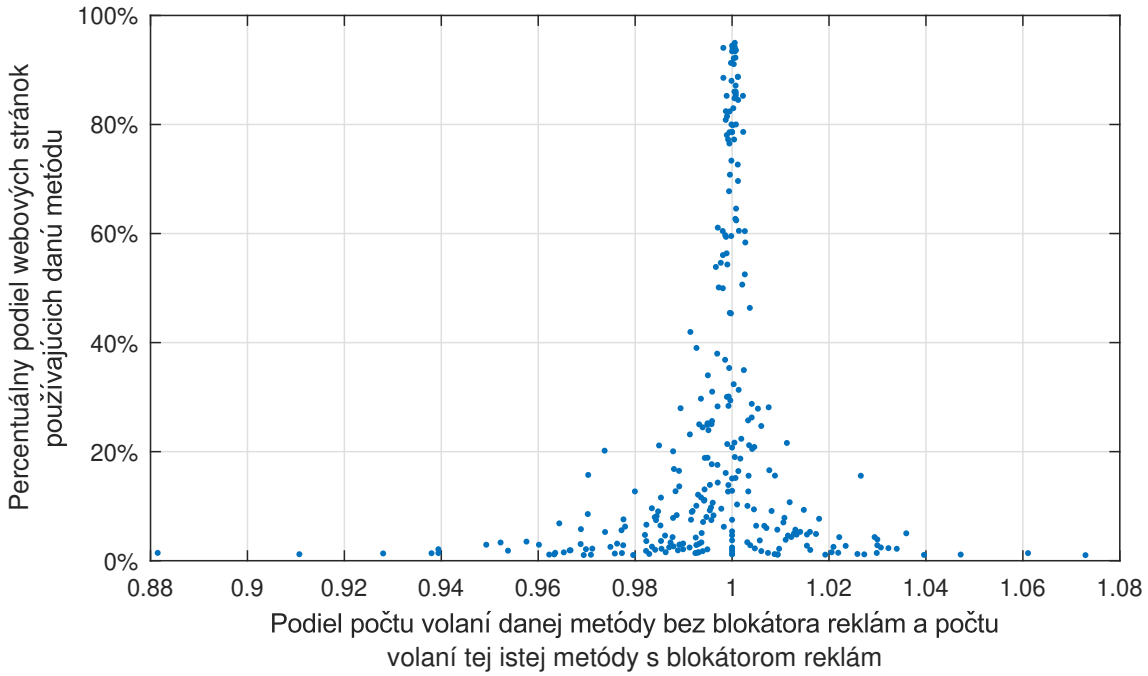
5.3 Analýza blokovania API doplnkom Ghostery

Pri vykonávaní meraní sme každú návštevu webovej stránky realizovali v dvoch režimoch. Prvým z týchto režimov bol režim s jediným aktívnym rozšírením, konkrétne s rozšírením Web API Manager, ktoré nám umožňovalo zaznamenávať volania JavaScriptových metód a ktoré bolo pre uskutočnenie našich meraní nevyhnutné. V rámci druhého režimu bolo okrem rozšírenia Web API Manager aktívne aj rozšírenie Ghostery. V tejto časti sa venujeme porovnaniu výsledkov získaných týmito dvoma spôsobmi.

Vo všeobecnosti môžeme povedať, že rozdiely v nameraných výsledkoch v rámci dvoch režimov meraní nie sú veľké. Konkrétne, pri meraniach s aktívnym rozšírením Ghostery sme na žiadnej webovej stránke neidentifikovali zablokovanie použitia celého API. Pri detailnejšej analýze volaní konkrétnych metód pri návšteve s aktívnym rozšírením Ghostery a bez neho sme zaznamenali drobné rozdiely, ktoré však nie sú významné a vyskytujú sa najmä pri volaniach takých metód, ktorých použitie sme identifikovali na pomerne malej časti webových stránok.

Na obrázku 5.9 uvádzame graf závislosti medzi blokovaním volaní metód, ktoré sme identifikovali na viac, než 1% webových stránok a množstvom webových stránok, na ktorých boli volania príslušných metód identifikované. Môžeme vidieť, že s rastúcim počtom webových stránok, na ktorých boli volania príslušnej metódy identifikované sa zmenšuje rozdiel medzi tým, koľko volaní príslušnej metódy sme identifikovali pri aktívnom rozšírení Ghostery a koľko volaní metód sme identifikovali bez jeho použitia.

Uvedené výsledky meraní nie sú uspokojivé a pri použití rozšírenia Ghostery zrejme nastala chyba, kvôli ktorej rozšírenie Ghostery neblokovalo volania metód tak, ako by sme očakávali. Príčinu zjavnej nefunkčnosti rozšírenia Ghostery pri našich meraniach sa nám nepodarilo identifikovať.



Obr. 5.9: Graf závislosti blokovania volaní metód, ktoré sme identifikovali na viac, než 1% webových stránok rozšírením Ghostery a množstvom webových stránok, na ktorých boli volania príslušných metód identifikované

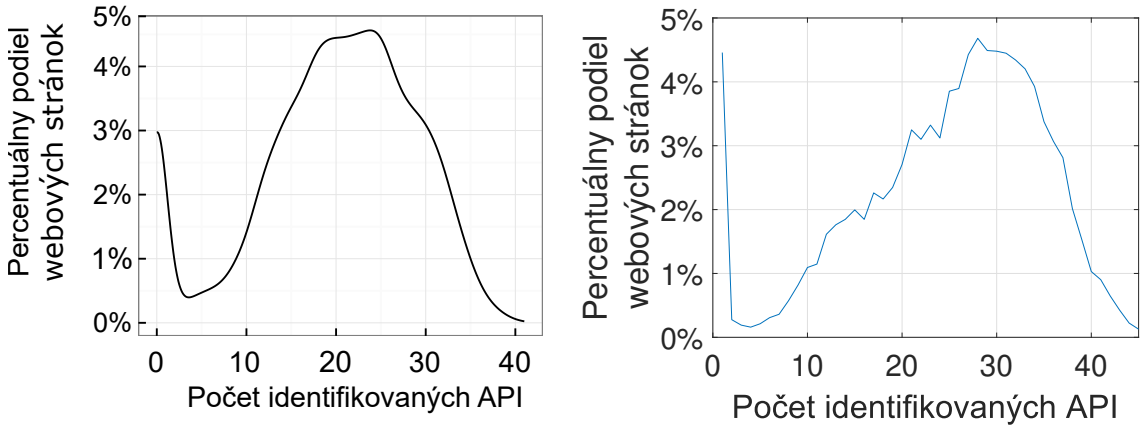
5.4 Porovnanie výsledkov s predošlým výskumom

Metodológia meraní, ktoré sme v rámci tejto práce vykonali je založená na metodológii navrhnutej Petrom Snyderom et al., ktorá bola publikovaná spolu s výsledkami meraní [41]. Vďaka tomu môžeme porovnať výsledky našich meraní s meraniami vykonanými v roku 2016.

Na obrázku 5.10 môžeme vidieť porovnanie počtu identifikovaných API, ktoré sa autorom uvedeného výskumu podarilo na webových stránkach identifikovať v roku 2016 (5.10a) a počtu identifikovaných API, ktoré sme na webových stránkach identifikovali v rámci našej práce (5.10b). Pri pohľade na toto porovnanie môžeme sledovať nárast počtu API identifikovaných na najväčšej množine webových stránok reprezentovaný posunutím vrcholu krivky na uvedených obrázkoch.

Vo všeobecnosti však môžeme povedať, že najpoužívanejšími API boli už v roku 2016 tie API, ktoré definujú možnosti pre prácu s HTML dokumentom a DOM. Hoci na úrovni konkrétnych jednotlivých API pre prácu s HTML a DOM môžeme nájsť rozdiely, vo všeobecnosti platí, že aj v roku 2016 boli najviac používané API definované štandardom HTML a DOM (napríklad DOM Level 1, DOM Level 2: Events alebo Selectors API).

V prípade, že sa zameriame na rozdiely medzi výsledkami meraní z roku 2016 a výsledkami našich meraní, môžeme sledovať zmeny napríklad v oblasti API určených pre prácu s požiadavkami posielanými na server asynchrónnym spôsobom. Konkrétne, pri XMLHttpRequest, Fetch a Beacon API môžeme pozorovať nárast v ich používaní, pričom pri používaní Fetch API môžeme pozorovať najväčší nárast počtu webových stránok využívajucich toto API. Vysvetlením takéhoto nárastu môže byť napríklad fakt, že Fetch API je najm-



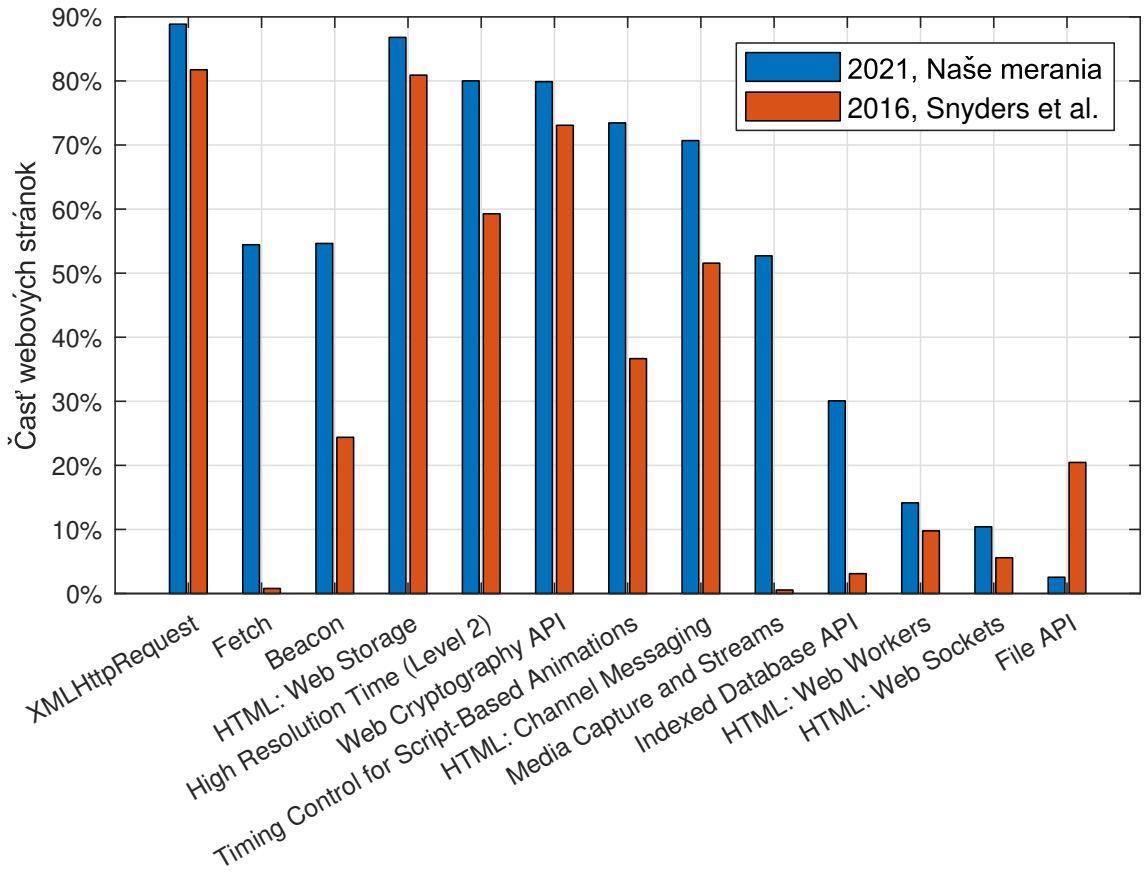
(a) Snyder et al.

(b) Naše merania

Obr. 5.10: Porovnanie počtov identifikovaných API voči relatívnej časti webových stránok, kde bol príslušný počet API identifikovaný

ladším API z uvedenej trojice a v prehliadači Mozilla Firefox je implementované od verzie 39, ktorá bola vydaná až v polovici roka 2015.

Porovnanie vybraných výsledkov meraní realizovaných v roku 2016 a výsledkov našich meraní môžeme vidieť na obrázku 5.11. Môžeme si všimnúť, že v rámci našich meraní sme s jednou výnimkou zaznamenali použitie všetkých uvedených API na väčšom množstve webových stránok, pričom sme tento trend sledovali aj pri analýze ďalších API, ktoré v tejto práci neuvádzame. Výnimkou v inak rastúcom trende používania rôznych API webovými stránkami je File API, ktorého použitie sme v rámci našich meraní zaznamenali na $2,54\%$ webových stránok, čo predstavuje pokles v používaní tohto API o necelých 20 percentuálnych bodov.



Obr. 5.11: Histogram vyjadrujúci percentuálnu časť webových stránok, ktoré používali API pre manipuláciu a posielanie asynchrónnych požiadavok na server v roku 2016 a 2021

Kapitola 6

Záver

Svet webových technológií sa postupom času stáva viac a viac komplikovaným. Užívatelia navštevujú webové stránky zo širokej škály zariadení, pričom tvorcovia webových stránok majú stále viac a viac možností, ako využiť potenciál týchto zariadení. V tejto práci sme sa zaoberali zmapovaním používania JavaScriptových API v bežnom webovom prehliadači.

V tejto práci sme sa venovali návrhu a implementácii prostredia pre vykonanie meraní, v rámci ktorých je možné navštíviť väčší počet webových stránok a zmerať používanie rôznych JavaScriptových API poskytnutých vývojárom webových stránok cez webový prehliadač. Implementované prostredie sa skladá najmä z nástroja OpenWPM, ktorý slúži na inštrumentáciu webového prehliadača. Ďalším dôležitým prvkom implementovanej meracej platformy je rozšírenie do webového prehliadača ovládaného platformou OpenWPM, ktoré nám umožňuje zaznamenávať volania metód, ktoré vystavujú funkcionalitu rôznych JavaScriptových API. Pri tvorbe tohto rozšírenia do webového prehliadača sme vychádzali z existujúceho rozšírenia s názvom Web API Manager, ktoré bolo použité pri predošlom výskume realizovanom Petrom Snyderom [41, 42]. Pre vykonanie meraní sme potrebovali získať aktuálny zoznam metód, ktoré použitý webový prehliadač používa. V rámci toho sme v prehliadači Mozilla Firefox identifikovali 211 nových metód, ktoré sme zaradili do 38 nových API. Ako zoznam webových stránok, ktoré sme v rámci našich meraní navštívili sme využili zoznam Tranco obohatený o podstránky.

Uvedenú platformu pre meranie používania JavaScriptových API na webových stránkach sme použili na návštevu 10000 webových stránok. Z interpretácie dát, ktorú uvádzame v kapitole 5 vyplýva, že najpoužívanejšími API sú staršie API, ktoré umožňujú pracovať s HTML a DOM. API špecifikované v štandarde HTML sme identifikovali takmer na každej webovej stránke. Istým prekvapením v obľúbenosti naprieč analyzovanými webovými stránkami môže byť pomerne časté používanie High Resolution Time API a Web Cryptography API, ktorých použitie sme zaznamenali na 80,01%, resp. 79,89% webových stránok. Zaoberali sme sa tiež analýzou používania nových API. Z uskutočnených meraní vyplýva, že najpoužívanejšími z nových API sú Intersection Observer API, Background Tasks API a Resize observer API, ktorých použitie sme identifikovali na $52,7\%$, $41,97\%$ a $29,75\%$ webových stránok. Pri porovnaní výsledkov našich meraní s výsledkami meraní uskutočnenými v roku 2016 môžeme vidieť, že počet Web API používaných jednotlivými webovými stránkami vzrástol.

Literatúra

[1] Adobe. *JavaScript for Acrobat* [online]. Adobe [cit. 2020-10-06]. Dostupné z: https://www.adobe.com/devnet/acrobat/javascript.html.

[2] Apple. *JavaScriptCore – WebKit* [online]. 2020 [cit. 2020-12-20]. Dostupné z: https://trac.webkit.org/wiki/JavaScriptCore.

[3] Aqeel, W., Chandrasekaran, B., Feldmann, A. a Maggs, B. M. On Landing and Internal Web Pages: The Strange Case of Jekyll and Hyde in Web Performance Measurement. In: *Proceedings of the ACM Internet Measurement Conference*. New York, NY, USA: Association for Computing Machinery, 2020, s. 680–695. IMC '20. DOI: 10.1145/3419394.3423626. ISBN 9781450381383. Dostupné z: https://doi.org/10.1145/3419394.3423626.

[4] Aycock, J. A Brief History of Just-In-Time. *ACM Comput. Surv.* Jún 2003, zv. 35, s. 97–113. DOI: 10.1145/857076.857077.

[5] Beaufort, F. a Lamouri, M. *Picture-in-Picture*. Editor's Draft. W3C, október 2020. Https://w3c.github.io/picture-in-picture/.

[6] Byers, R., Lauke, P., Zolghadr, N. a Brubeck, M. *Pointer Events*. W3C Working Draft. W3C, október 2020. Https://www.w3.org/TR/2020/WD-pointerevents3-20201001/.

[7] Das, A., Acar, G., Borisov, N. a Pradeep, A. The Web's Sixth Sense: A Study of Scripts Accessing Smartphone Sensors. In:. Október 2018, s. 1515–1532. DOI: 10.1145/3243734.3243860.

[8] Deveria, A. *Can I use... Support tables for HTML5, CSS3, etc* [online]. caniuse.com [cit. 2021-01-05]. Dostupné z: https://caniuse.com/.

[9] Dirk Balfanz and Arnar Birgisson and Juan Lang. *FIDO U2F JavaScript API* [online]. FIDO Alliance [cit. 2021-01-05]. Dostupné z: https://fidoalliance.org/specs/fido-u2f-v1.0-nfc-bt-amendment-20150514/fido-u2f-javascript-api.html.

[10] Ecma International. *TC39-Royalty Free Task Group members* [online]. ecma-international.org [cit. 2020-10-06]. Dostupné z: https://www.ecma-international.org/memento/tc39-rf-taskgroup-members.htm.

[11] Englehardt, S. *Automated discovery of privacy violations on the web*. Chicago, Illinois, 2018. Dizertačná práca. Princeton University.

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

[13] Expósito Ventura, M., Ruipérez Valiente, J. A. a Forné, J. Analyzing and Testing Viewability Methods in an Advertising Network. *IEEE Access*. Júl 2020, zv. 8, s. 118751–118761. DOI: 10.1109/ACCESS.2020.3005478.

[14] Flanagan, D. *JavaScirpt: The Definitive Guide*. 7. vyd. O'Reilly Media, Inc., 2020. ISBN 978-1-491-95202-3.

[15] Frisbie, M. *Professional JavaScript® for Web Developers*. 4. vyd. John Wiley & Sons, Ltd, 2019. ISBN 978-1-119-36644-7.

[16] Grigorik, I., Cáceres, M. a Moreno, F. J. *Network Information API*. Draft Community Group Report. WICG, máj 2020. Https://wicg.github.io/netinfo/.

[17] Iqbal, U., Englehardt, S. a Shafiq, Z. *Fingerprinting the Fingerprinters: Learning to Detect Browser Fingerprinting Behaviors*. 2020.

[18] Isaacson, S., Wilson, C., Wood, L., Champion, M., Byrne, S. B. et al. *Document Object Model (DOM) Level 1*. W3C Recommendation. W3C, október 1998. Https://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/.

[19] Jueckstock, J. a Kapravelos, A. VisibleV8: In-browser Monitoring of JavaScript in the Wild. In: *Proceedings of the ACM Internet Measurement Conference (IMC)* [/projects/vv8/]. Október 2019.

[20] Kukar Kinney, M. a Close Scheinbaum, A. The determinants of consumers' shopping cart abandonment. *Journal of the Academy of Marketing Science*. Apríl 2009, zv. 38, s. 240–250. DOI: 10.1007/s11747-009-0141-5.

[21] Laperdrix, P., Bielova, N., Baudry, B. a Avoine, G. Browser Fingerprinting: A survey. *CoRR*. 2019, abs/1905.01051. Dostupné z: http://arxiv.org/abs/1905.01051.

[22] Le Pochat, V., Van Goethem, T., Tajalizadehkhoob, S., Korczyński, M. a Joosen, W. Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation. In: *Proceedings of the 26th Annual Network and Distributed System Security Symposium*. Február 2019. NDSS 2019. DOI: 10.14722/ndss.2019.23386.

[23] Le Pochat, V., Van Goethem, T., Tajalizadehkhoob, S., Korczyński, M. a Joosen, W. *A research-oriented top sites ranking hardened against manipulation - Tranco* [online]. 2020 [cit. 2020-12-17]. Dostupné z: https://tranco-list.eu/.

[24] Majestic. *Majestic Million - Majestic* [online]. 2020 [cit. 2020-12-17]. Dostupné z: https://majestic.com/reports/majestic-million.

[25] McCormack, C. *WebIDL Level 1*. W3C Recommendation. W3C, december 2016. Https://www.w3.org/TR/2016/REC-WebIDL-1-20161215/.

[26] MDN. *Information contained in a WebIDL file - The MDN project | MDN* [online]. 2020 [cit. 2020-12-20]. Dostupné z: https://developer.mozilla.org/en-US/docs/MDN/Contribute/Howto/Write_an_API_reference/Information_contained_in_a_WebIDL_file.

[27] MDN. *Intersection Observer API - Web APIs | MDN* [online]. 2021 [cit. 2021-05-14]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Intersection_Observer_API.

[28] MDN CONTRIBUTORS. *Battery Status API - Web APIs | MDN* [online]. Mozilla Developer Network [cit. 2021-01-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Battery_Status_API.

[29] MDN CONTRIBUTORS. *Canvas API - Web APIs | MDN* [online]. Mozilla Developer Network [cit. 2021-01-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API.

[30] MDN CONTRIBUTORS. *Permissions API - Web APIs | MDN* [online]. Mozilla Developer Network [cit. 2021-01-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Permissions_API.

[31] MDN CONTRIBUTORS. *SpiderMonkey: The Mozilla JavaScript runtime* [online]. Mozilla Developer Network [cit. 2020-10-06]. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey.

[32] MDN CONTRIBUTORS. *Web Audio API - Web APIs | MDN* [online]. Mozilla Developer Network [cit. 2021-01-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API.

[33] MONGODB, INC. *JavaScript Changes in MongoDB 3.2* [online]. MongoDB, Inc [cit. 2020-10-06]. Dostupné z: https://docs.mongodb.com/manual/release-notes/3.2-javascript/.

[34] MOWERY, K. a SHACHAM, H. Pixel Perfect: Fingerprinting Canvas in HTML5. In: FREDRIKSON, M., ed. *Proceedings of W2SP 2012*. Máj 2012.

[35] OLEJNIK, L., ENGLEHARDT, S. a NARAYANAN, A. Battery status not included: Assessing privacy in web standards. *CEUR Workshop Proceedings*. CEUR-WS. január 2017, zv. 1873, s. 17–24. ISSN 1613-0073. 3rd International Workshop on Privacy Engineering, IWPE 2017 ; Conference date: 25-05-2017.

[36] RAUSCHMAYER, A. *Speaking JavaScript*. 1. vyd. O'Reilly Media, Inc., 2014. ISBN 978-1-449-36503-5.

[37] SARKER, S., JUECKSTOCK, J. a KAPRAVELOS, A. Hiding in Plain Site: Detecting JavaScript Obfuscation through Concealed Browser API Usage. In: *Proceedings of the ACM Internet Measurement Conference (IMC)*. Október 2020.

[38] SCHEITLE, Q., HOHLFELD, O., GAMBA, J., JELTEN, J., ZIMMERMANN, T. et al. A Long Way to the Top: Significance, Structure, and Stability of Internet Top Lists. In:. Október 2018, s. 478–493. DOI: 10.1145/3278532.3278574.

[39] SELENIUM. *The Selenium project and tools :: Documentation for Selenium* [online]. 2020 [cit. 2020-12-18]. Dostupné z: https://www.selenium.dev/documentation/en/introduction/the_selenium_project_and_tools/.

[40] SNYDER, P. *Improving Web Privacy And Security with a Cost-Benefit Analysis of the Web API*. Chicago, Illinois, 2018. Dizertačná práca. University of Illinois at Chicago.

[41] SNYDER, P., ANSARI, L., TAYLOR, C. a KANICH, C. Browser Feature Usage on the Modern Web. In:. November 2016, s. 97–110. DOI: 10.1145/2987443.2987466.

[42] Snyder, P., Taylor, C. a Kanich, C. Most Websites Don't Need to Vibrate: A Cost-Benefit Approach to Improving Browser Security. In:. Október 2017, s. 179–194. DOI: 10.1145/3133956.3133966. ISBN 978-1-4503-4946-8.

[43] Solomakhin, R., Wang, D., Caceres, M. a Jacobs, I. *Payment Request API*. Candidate Recommendation. W3C, december 2020. Https://www.w3.org/TR/2020/CR-payment-request-20201203/.

[44] Sørensen, J. a Kosta, S. Before and After GDPR: The Changes in Third Party Presence at Public and Private European Websites. *The World Wide Web Conference*. Máj 2019. DOI: 10.1145/3308558.3313524.

[45] Tabarés, R. HTML5 and the evolution of HTML; tracing the origins of digital platforms. *Technology in Society*. 2021, zv. 65, s. 101529. DOI: https://doi.org/10.1016/j.techsoc.2021.101529. ISSN 0160-791X. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0160791X2100004X.

[46] Umbrella, C. *Cisco Popularity List* [online]. 2016 [cit. 2020-12-17]. Dostupné z: http://s3-us-west-1.amazonaws.com/umbrella-static/index.html.

[47] Web Hypertext Application Technology Working Group. *DOM Living Standard* [online]. WHATWG [cit. 2020-12-08]. Dostupné z: https://dom.spec.whatwg.org/#interface-document.

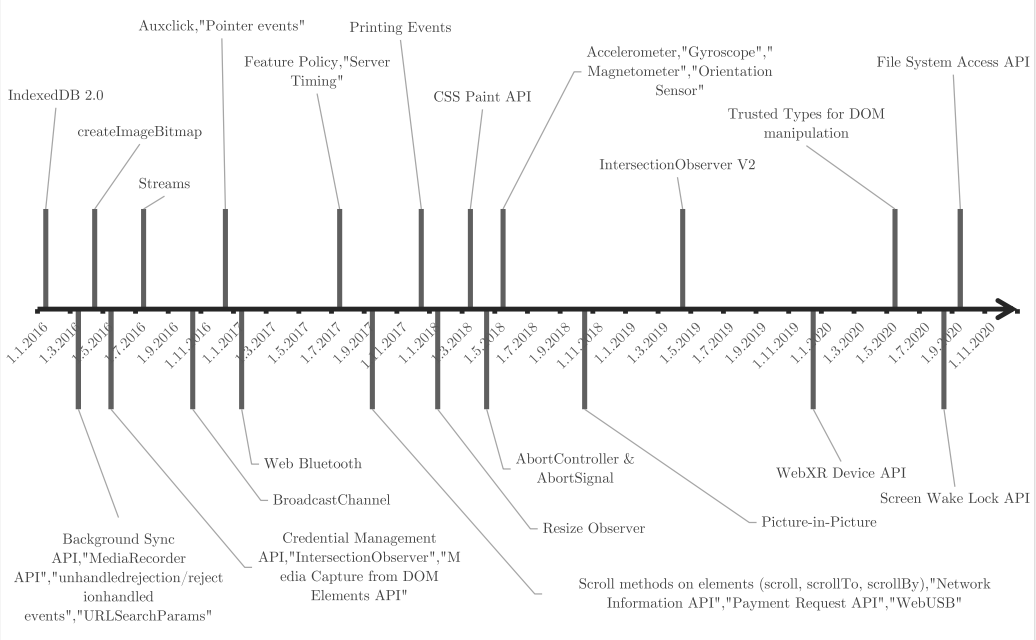
[48] Web Hypertext Application Technology Working Group. *Streams Living Standard* [online]. WHATWG [cit. 2021-01-11]. Dostupné z: https://dom.spec.whatwg.org/#interface-document.

[49] WHATWG. *XMLHttpRequest Living Standard* [online]. WHATWG [cit. 2020-12-08]. Dostupné z: https://xhr.spec.whatwg.org/.

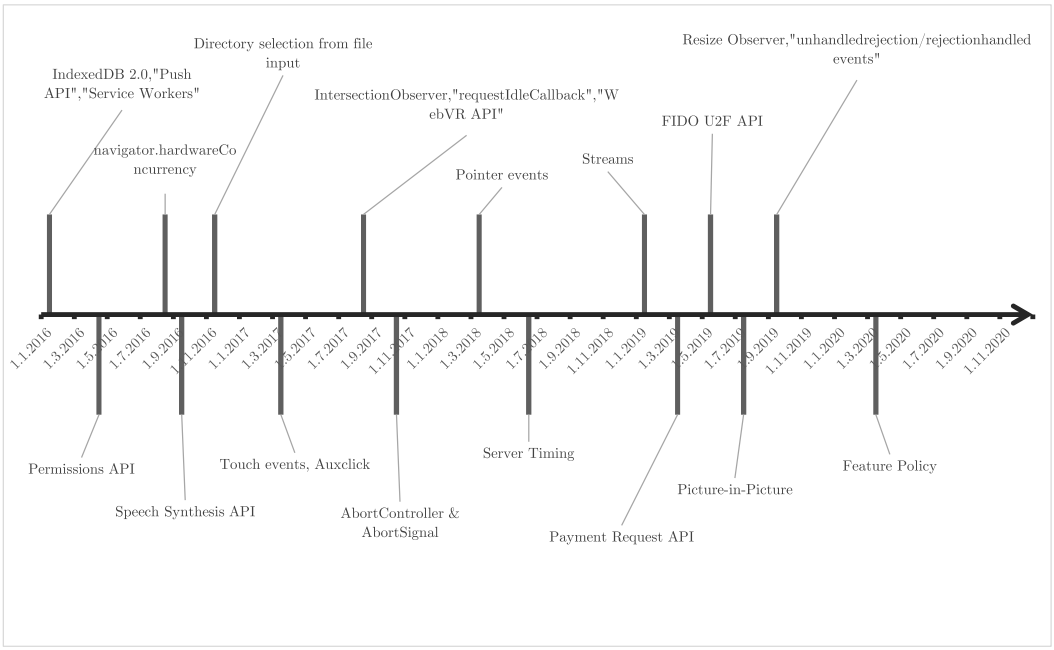
[50] Zager, S., Álvarez, E. C. a Blain, M. *Intersection Observer*. W3C Working Draft. W3C, november 2020. Https://www.w3.org/TR/2020/WD-intersection-observer-20201126/.

Príloha A

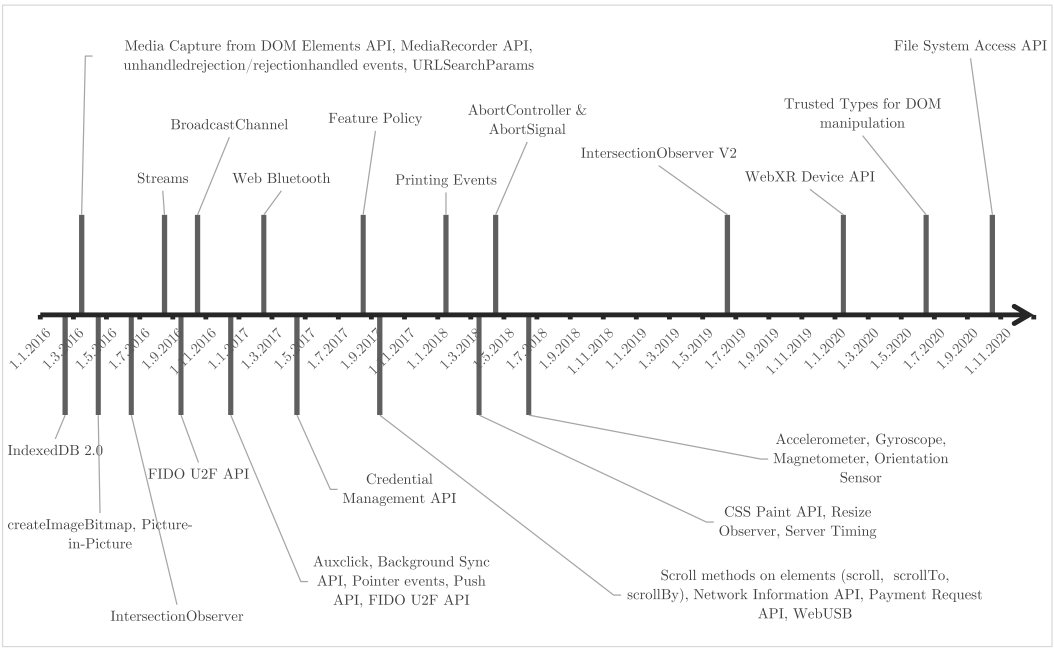
Časové osi implementovania API vo vybraných prehliadačoch



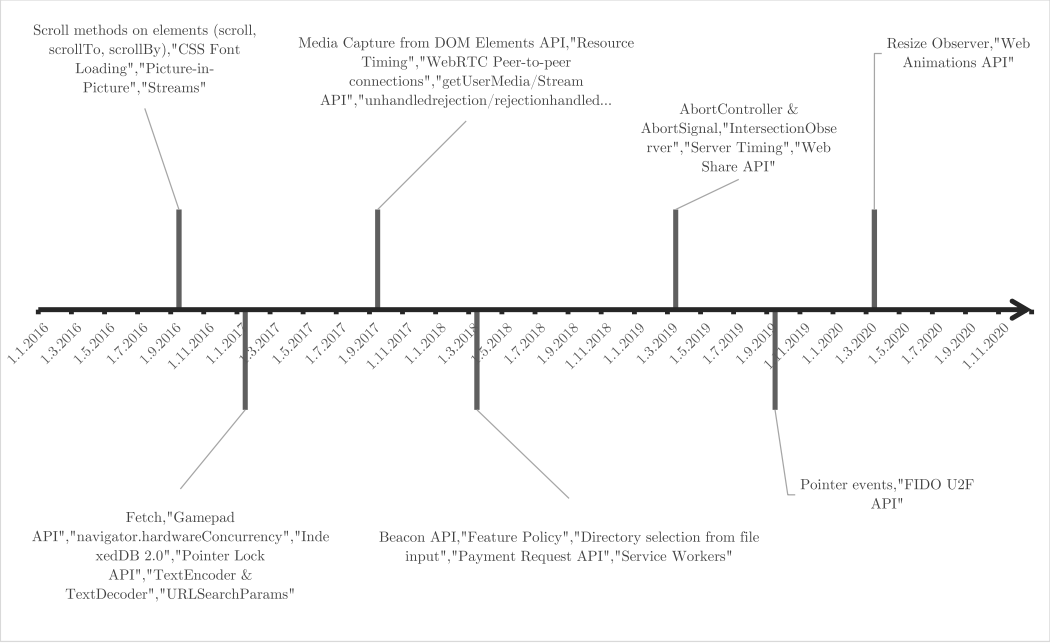
Obr. A.1: Časová os implementovania rôznych API prehliadačom Google Chrome od roku 2016, dáta pochádzajú z portálu caniuse.com



Obr. A.2: Časová os implementovania rôznych API prehliadačom Firefox od roku 2016, dáta pochádzajú z portálu caniuse.com



Obr. A.3: Časová os implementovania rôznych API prehliadačom Opera od roku 2016, dáta pochádzajú z portálu caniuse.com



Obr. A.4: Časová os implementovania rôznych API prehliadačom Safari od roku 2016, dáta pochádzajú z portálu caniuse.com

Príloha B

Používanie JavaScriptových API

V nasledujúcej tabuľke uvádzame zoznam API, ktoré sme v rámci vykonaných meraní identifikovali na viac, než 1% webových stránok. Stĺpec U značí percentuálnu časť webových stránok, kde bolo použitie daného API identifikované.

Názov API	**U**
HTML	99,86%
DOM Level 1	95,24%
DOM	95,11%
DOM Level 2: Events	94,98%
Selectors API (Level 1)	93,85%
DOM Level 2: Core	93,66%
DOM Level 2: Style	89,48%
XMLHttpRequest	88,86%
DOM Level 3: Core	87,12%
HTML: Web Storage	86,80%
CSS Object Model (CSSOM)	84,39%
High Resolution Time (Level 2)	80,01%
Web Cryptography API	79,89%
HTML: DOM	78,58%
DOM Level 4	77,26%
HTML 5	74,17%
Timing Control for Script-Based Animations	73,45%
HTML: Channel Messaging	70,69%
Performance Timeline	64,11%
CSSOM View Module	62,76%
HTML: Canvas Element	59,97%
URL	55,41%
Beacon	54,64%
Fetch	54,44%
Intersection Observer API	52,70%
User Timing (Level 2)	50,23%
Background Tasks API	41,97%
Performance Timeline (Level 2)	38,55%
HTML: Dynamic Markup Insertion	36,91%

DOM Parsing and Serialization	36,18%
Indexed Database API	30,07%
Resize Observer API	29,75%
CSS Font Loading (Level 3)	28,75%
DOM Level 2: Traversal and Range	28,72%
HTML: History Interface	27,10%
Navigation Timing Level 2	16,09%
HTML: Custom Elements	15,19%
Media Session API	15,17%
Service Workers	14,97%
Selection API	14,39%
HTML: Web Workers	14,15%
Resource Timing	10,90%
HTML: Web Sockets	10,42%
Permissions API	9,62%
UI Events Specification	8,73%
Encoding	7,76%
Navigation Timing	7,71%
Web Speech API	7,60%
DOM Level 2: HTML	6,46%
Not specified yet	6,00%
Web Audio API	5,37%
Non-Standard	4,48%
Media Source Extensions	3,97%
Gamepad	3,84%
uncategorized	3,74%
execCommand	3,37%
Scalable Vector Graphics (SVG) 1.1 (Second Edition)	2,99%
WebRTC 1.0: Real-time Communication Between Browser	2,96%
HTML: Broadcasting	2,89%
File API	2,54%
Media Capture and Streams	2,36%
DOM Level 3: XPath	2,18%
HTML: Plugins	1,87%
Geolocation API	1,53%
Push API	1,48%
Encrypted Media Extensions	1,40%
Resource Timing Level 2	1,40%
Web Animations API	1,25%
Storage API	1,01%

Tabuľka B.1: API, ktorých použitie sme identifikovali na viac, než 1% webových stránok, U značí percentuálnu časť webových stránok, na ktorých bolo identifikované použitie daného API

Príloha C

API identifikované ako nové

Nasledovná tabuľka obsahuje API, ktoré sme v rámci manuálne vykonanej klasifikácie identifikovali ako nové. Pod pojmom nové API rozumieme také API, ktoré nebolo kategorizované v rámci výskumu uskutočneného skupinou Snyder et al., ktorý bol uskutočnený v roku 2016. Spravidla teda ide o také API, ktoré boli v prehliadači Mozilla Firefox implementované v roku 2016 a neskôr.

Názov API	**Počet metód**
Web Animations API	14
HTML: AudioTrackList Interface	2
Service Workers	20
Clipboard API	5
Credential Management Level 1	4
CSS Animations Level 1	3
HTML: Custom Elements	4
HTML Drag and Drop API	8
Touch Events	4
Geometry Interfaces Module Level 1	19
Permissions Policy API	4
File and Directory Entries API	6
Background Tasks API	1
Canvas API	4
MediaStream Image Capture API	1
Input Events	1
Intersection Observer API	4
Media Capabilities API	2
Media Session API	2
Web MIDI API	5
Web Share API	1
CSS Painting API Level 1	1
Event Timing API	1
Navigation Timing Level 2	1
Resource Timing Level 2	1
Server Timing	1
Permissions API	2

Pointer Events	2
Reporting API	3
Resize Observer API	3
HTML Sanitizer API	2
Storage API	3
Scalable Vector Graphics (SVG) 2	4
Web Authentication API Level 1	3
WebXR Device API	11
WebGPU	55
Cooperative Scheduling of Background Tasks	2
HTML: Worklets	2

Tabuľka C.1: API, ktoré sme pri manuálnej kategorizácií identifikovali ako nové

Príloha D

Používanie metód vybraných API používaných pre prácu s HTML a DOM

V nasledovnej tabuľke uvádzame používanie metód vybraných API pre prácu s HTML dokumentami a DOM štruktúrou na webových stránkach. Stĺpec U značí percentuálnu časť webových stránok, kde bolo identifikované použitie príslušnej metódy a P značí priemerný počet volaní na každej stránke, kde bola invokácia príslušnej metódy identifikovaná. Uvádzame len tie metódy, ktoré boli identifikované na viac, ako 1% webových stránok.

Názov metódy	**API**	**U**	**P**
Document.createElement	DOM	95,03%	2873
EventTarget.addEventListener	DOM LVL 2: Events	94,98%	3127
Node.appendChild	DOM LVL 1	94,45%	3036
Document.getElementsByTagName	DOM LVL 1	94,18%	711
window.open	HTML	94,06%	8
Document.getElementById	DOM LVL 1	93,92%	1404
Element.setAttribute	DOM LVL 1	93,83%	5191
window.setTimeout	HTML	93,62%	3524
Node.removeChild	DOM LVL 1	93,42%	351
Element.getAttribute	DOM LVL 1	93,41%	10999
Document.querySelectorAll	Selectors API (LVL 1)	92,26%	1044
Node.insertBefore	DOM LVL 1	92,14%	417
EventTarget.removeEventListener	DOM LVL 2: Events	91,30%	429
Element.querySelectorAll	Selectors API (LVL 1)	91,07%	939
Element.getElementsByTagName	DOM LVL 1	88,56%	1170
Document.querySelector	Selectors API (LVL 1)	87,14%	387
Document.getElementsByClassName	DOM	86,03%	571
Document.createDocumentFragment	DOM LVL 2: Core	85,70%	893
window.clearTimeout	DOM	85,33%	881
Node.cloneNode	DOM LVL 1	85,25%	209
Document.createComment	DOM LVL 2: Core	84,81%	137
Document.createTextNode	DOM LVL 2: Core	84,50%	508
Element.removeAttribute	DOM LVL 1	82,44%	601

window.setInterval	HTML	82,40%	61
Element.getElementsByClassName	DOM	78,64%	317
Element.matches	DOM	78,58%	2408
Document.getElementsByName	HTML: DOM	78,58%	23
Node.contains	DOM LVL 3: Core	78,06%	1254
Node.compareDocumentPosition	DOM LVL 3: Core	77,25%	1058
MutationObserver.observe	DOM LVL 4	77,25%	83
DOMTokenList.add	DOM	76,52%	322
Element.hasAttribute	DOM LVL 2: Core	72,65%	1432
window.clearInterval	DOM	69,64%	57
Element.querySelector	Selectors API (LVL 1)	67,76%	476
DOMTokenList.remove	DOM	64,58%	604
EventTarget.dispatchEvent	DOM LVL 2: Events	62,70%	447
window.atob	HTML 5	61,08%	275
Document.createElementNS	DOM LVL 2: Core	60,47%	401
Document.createEvent	DOM LVL 2: Events	60,45%	135
HTMLCanvasElement.getContext	HTML: Canvas	59,71%	63
DOMTokenList.contains	DOM	59,54%	518
DOMImplementation.createHTMLDocument	DOM	56,37%	12
window.btoa	HTML 5	53,87%	159
DocumentFragment.querySelectorAll	Selectors API (LVL 1)	50,64%	72
DOMTokenList.supports	DOM	49,97%	94
Node.replaceChild	DOM LVL 1	46,37%	91
HTMLMediaElement.canPlayType	HTML 5	39,02%	66
Node.getRootNode	DOM	37,97%	385
Document.hasFocus	HTML	36,85%	121
HTMLElement.focus	DOM LVL 1	32,37%	10
HTMLCanvasElement.toDataURL	HTML: Canvas	29,39%	14
DOMImplementation.hasFeature	DOM LVL 2: Core	28,77%	11
CustomEvent.initCustomEvent	DOM	28,40%	211
CanvasRenderingContext2D.fillText	HTML: Canvas	28,30%	106
Element.setAttributeNS	DOM LVL 2: Core	28,13%	84
Event.initEvent	DOM LVL 2: Events	26,26%	72
Element.mozMatchesSelector	DOM	25,04%	158
HTMLMediaElement.load	HTML 5	22,36%	7
MutationObserver.disconnect	DOM LVL 4	21,66%	149
CanvasRenderingContext2D.fillRect	HTML: Canvas	21,59%	740
Element.closest	DOM	21,18%	571
Node.hasChildNodes	DOM LVL 1	20,18%	1246
CanvasRenderingContext2D.clearRect	HTML: Canvas	19,01%	230
CanvasRenderingContext2D.beginPath	HTML: Canvas	18,89%	1425
CanvasRenderingContext2D.fill	HTML: Canvas	18,85%	997
Element.remove	DOM	17,57%	18
DOMTokenList.toggle	DOM	16,59%	107
Event.stopPropagation	DOM LVL 2: Events	16,48%	26

Element.webkitMatchesSelector	DOM	16,43%	265
NodeList.item	DOM LVL 1	15,61%	925
CanvasRenderingContext2D.arc	HTML: Canvas	15,58%	504
HTMLInputElement.checkValidity	HTML 5	14,33%	12
CanvasRenderingContext2D.stroke	HTML: Canvas	12,82%	465
Event.preventDefault	DOM LVL 2: Events	12,71%	11
HTMLMediaElement.play	HTML 5	12,71%	10
CanvasRenderingContext2D.closePath	HTML: Canvas	12,69%	1788
CanvasRenderingContext2D.moveTo	HTML: Canvas	12,10%	1758
Element.insertAdjacentElement	DOM	11,60%	72
DocumentFragment.querySelector	Selectors API (LVL 1)	10,72%	284
CanvasRenderingContext2D.measureText	HTML: Canvas	10,32%	249
Node.isEqualNode	DOM LVL 3: Core	10,05%	1836
CanvasRenderingContext2D.drawImage	HTML: Canvas	9,78%	1223
CanvasRenderingContext2D.getImageData	HTML: Canvas	9,32%	52
CanvasRenderingContext2D.rect	HTML: Canvas	9,12%	385
HTMLCollection.item	DOM LVL 1	9,04%	345
CanvasRenderingContext2D.quadraticC...	HTML: Canvas	8,56%	8748
CanvasGradient.addColorStop	HTML: Canvas	8,01%	446
Element.append	DOM	7,52%	64
NamedNodeMap.getNamedItem	DOM LVL 2: Core	7,51%	190
HTMLFormElement.submit	DOM LVL 1	7,44%	6
CanvasRenderingContext2D.lineTo	HTML: Canvas	7,10%	14711
CanvasRenderingContext2D.isPointInPath	HTML: Canvas	7,02%	4
HTMLElement.click	DOM LVL 1	6,85%	4
CanvasRenderingContext2D.restore	HTML: Canvas	6,41%	4905
CanvasRenderingContext2D.save	HTML: Canvas	6,39%	4918
HTMLImageElement.decode	HTML	6,25%	27
CanvasRenderingContext2D.createRadial...	HTML: Canvas	6,22%	30
CanvasRenderingContext2D.bezierCurveTo	HTML: Canvas	5,69%	561
CanvasRenderingContext2D.strokeText	HTML: Canvas	5,38%	5
Element.attachShadow	DOM	5,27%	43
Element.getAttributeNode	DOM LVL 1	5,07%	310
DOMStringList.contains	DOM LVL 3: Core	4,72%	10
CanvasRenderingContext2D.ellipse	HTML: Canvas	4,61%	60
HTMLMediaElement.pause	HTML 5	4,32%	12
CanvasRenderingContext2D.setTransform	HTML: Canvas	4,25%	1057
CanvasRenderingContext2D.transform	HTML: Canvas	3,74%	7510
HTMLTableRowElement.insertCell	DOM LVL 1	3,35%	2140
Document.importNode	DOM LVL 2: Core	3,34%	901
AbortController.abort	DOM	3,15%	13
Element.hasAttributes	DOM LVL 2: Core	3,11%	2036
CanvasRenderingContext2D.clip	HTML: Canvas	2,93%	297
CanvasRenderingContext2D.translate	HTML: Canvas	2,83%	1092
Element.prepend	DOM	2,72%	15

Element.animate	DOM	2,43%	13
HTMLTableElement.insertRow	DOM LVL 1	2,42%	29
NamedNodeMap.item	DOM LVL 2: Core	2,30%	1721
CanvasRenderingContext2D.createLinear...	HTML: Canvas	2,21%	671
CanvasRenderingContext2D.scale	HTML: Canvas	2,19%	441
Navigator.registerProtocolHandler	HTML	2,01%	2
HTMLFormElement.reset	DOM LVL 1	1,76%	2
TimeRanges.end	HTML	1,70%	1092
CanvasRenderingContext2D.rotate	HTML: Canvas	1,55%	1079
Event.composedPath	DOM	1,51%	29
HTMLTableSectionElement.insertRow	DOM LVL 1	1,44%	969
HTMLInputElement.setCustomValidity	HTML 5	1,43%	38
window.focus	HTML	1,40%	7
TimeRanges.start	HTML	1,39%	590
Document.createAttribute	DOM	1,32%	42
CanvasRenderingContext2D.setLineDash	HTML: Canvas	1,24%	206
MutationObserver.takeRecords	DOM LVL 4	1,23%	145
Node.isSameNode	DOM	1,19%	506
Element.getAttributeNS	DOM LVL 2: Core	1,07%	1178
Element.setAttributeNode	DOM LVL 1	1,04%	47
CharacterData.appendData	DOM	1,02%	49

Tabuľka D.1: Používanie metód vybraných API pre prácu s HTML dokumentami a DOM štruktúrou na webových stránkach, stĺpec U značí percentuálnu časť webových stránok, kde bolo identifikované použitie príslušnej metódy a P značí priemerný počet volaní na každej stránke, kde bola invokácia príslušnej metódy identifikovaná, uvedené sú len tie metódy, ktoré boli identifikované na viac, ako 1% webových stránok

Príloha E

Používanie nových metód starších API

V nasledovnej tabuľke uvádzame informácie o používaní nových metód, ktoré boli pridané do už existujúcich starších API. Stĺpec U značí percentuálnu časť webových stránok, kde bolo identifikované použitie príslušnej metódy a P značí priemerný počet volaní na každej stránke, kde bola invokácia príslušnej metódy identifikovaná. Uvádzame len tie metódy, ktoré boli identifikované na viac, ako 1% webových stránok.

Názov metódy	**API**	**U**	**P**
DOMTokenList.supports	DOM	49,97%	93,66
Node.getRootNode	DOM	37,97%	384,91
Document.write	HTML: Dyn. Mark. Ins.	34,96%	51,08
PerformanceObserver.observe	Performance Timeline 2	27,89%	16,53
PerformanceObserverEntryList.getEntries	Performance Timeline 2	24,70%	51,45
Document.close	HTML: Dyn. Mark. Ins.	13,08%	7,49
Document.open	HTML: Dyn. Mark. Ins.	12,74%	7,67
Element.insertAdjacentElement	DOM	11,60%	71,60
PerformanceObserver.disconnect	Performance Timeline 2	7,87%	13,54
PerformanceTiming.toJSON	Navigation Timing	7,69%	4,66
SpeechSynthesis.getVoices	Web Speech API	7,58%	5,92
Element.append	DOM	7,52%	64,26
HTMLImageElement.decode	HTML	6,25%	27,44
Element.attachShadow	DOM	5,27%	43,24
AudioScheduledSourceNode.start	Web Audio API	4,99%	4,64
BaseAudioContext.createDyn...	Web Audio API	4,81%	3,97
BaseAudioContext.createOscillator	Web Audio API	4,79%	4,71
CanvasRenderingContext2D.ellipse	HTML: Canvas Element	4,61%	59,56
PerformanceObserver.takeRecords	Performance Timeline 2	3,62%	2,99
AbortController.abort	DOM	3,15%	13,41
Element.prepend	DOM	2,72%	14,58
Document.execCommand	execCommand	2,57%	5,41
Element.animate	DOM	2,43%	12,79
IDBObjectStore.getAll	Indexed Database API	1,91%	81,99
URLSearchParams.sort	URL	1,90%	11,11

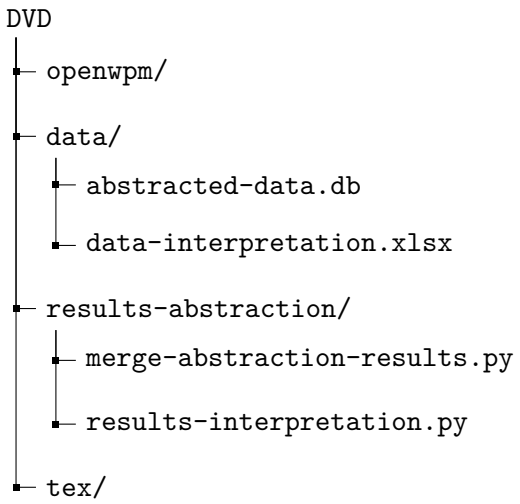
Document.writeln	HTML: Dyn. Mark. Ins.	1,72%	104,88
Event.composedPath	DOM	1,51%	29,12
Geolocation.getCurrentPosition	Geolocation API	1,45%	3,29
Node.isSameNode	DOM	1,19%	505,52
PerformanceEntry.toJSON	Performance Timeline 2	1,04%	39,88

Tabuľka E.1: Používanie nových metód pridaných do starších API, stĺpec U značí percentuálnu časť webových stránok, kde bolo identifikované použitie príslušnej metódy a P značí priemerný počet volaní na každej stránke, kde bola invokácia príslušnej metódy identifikovaná, uvedené sú len tie metódy, ktoré boli identifikované na viac, ako 1% webových stránok

Príloha F

Obsah priloženého pamäťového média

Priložené pamäťové médium má nasledovnú štruktúru:



Adresár `openwpm/` obsahuje meraciu platformu, ktorú sme použili na meranie používania JavaScriptových API na webe.

Ďalej, adresár `data/` obsahuje abstrahovanú SQLite databázu, v ktorej sú uložené dáta po abstrakcií dát popísanej v časti 4.4. Adresár `data/` tiež obsahuje súbor určený na otvorenie v programe Microsoft Excel, `data-interpretation.xlsx`, v ktorom je možné nájsť informácie o najpoužívanejších API, o volaní metód a podobne. Dáta uvedené v súbore `data-interpretation.xlsx` pochádzajú z analýzy abstrahovaných dát.

V adresári `results-abstraction/` je možné nájsť programy jazyka Python, ktoré boli použité pri abstrakcií dát nameraných platformou OpenWPM. Vzhľadom k tomu, že abstrakcia dát programom `results-interpretation.py` prebiehala na viacerých serveroch, výsledkom bolo niekoľko databáz, ktoré bolo nutné zlúčiť do jednej. Pre tento účel sme použili aplikáciu `merge-abstraction-results.py`, ktorej výstupom je jedna SQLite databáza.

Adresár `tex/` obsahuje súbory systému LaTeXpre vygenerovanie tejto technickej správy.