VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

ROZŠÍŘENÍ WEBOVÝCH PROHLÍŽEČŮ PRO ANALÝZU STRÁNEK

WEB BROWSER EXTENSION FOR WEB PAGE ANALYSIS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JAKUB NOVOTNÝ**
AUTHOR

VEDOUCÍ PRÁCE **doc. Ing. RADEK BURGET, Ph.D.**
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

156789

Ústav: Ústav informačních systémů (UIFS)
Student: **Novotný Jakub**
Program: Informační technologie
Název: **Rozšíření webových prohlížečů pro analýzu stránek**
Kategorie: Web
Akademický rok: 2023/24

Zadání:

1. Prostudujte existující technologie pro tvorbu rozšíření webových prohlížečů. Zaměřte se na technologii WebExtensions.
2. Seznamte se s detaily reprezentace zobrazené webové stránky v prohlížeči a souvisejícím aplikačním rozhraním webových prohlížečů.
3. Na základě konzultací s vedoucím navrhněte rozšiřující modul prohlížeče, který umožní uživateli odeslat detaily o aktuálně zobrazené stránce a jejím obsahu do serverové aplikace provádějící další analýzu a zobrazit výsledky analýzy v obsahu stránky.
4. Implementujte navrženou aplikaci pomocí vhodných technologií a integrujte s existující serverovou aplikací dodanou vedoucím práce.
5. Proveďte testování vytvořeného rozšíření na reálných webových stránkách různého zaměření.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Žára, O.: JavaScript - Programátorské techniky a webové technologie, Computer Press, 2015
- Web Extensions, Mozilla Developer Network, dostupné z https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions [cit. 2023-10-10]
- Michálek, M.: Vzhůru do CSS3, e-kniha, 2015, https://www.vzhurudolu.cz/ebook-css3

Při obhajobě semestrální části projektu je požadováno:
Bez požadavků.

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

Vedoucí práce: **Burget Radek, doc. Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1.11.2023
Termín pro odevzdání: 9.5.2024
Datum schválení: 30.10.2023

Abstrakt

Cílem tohoto projektu je vytvořit rozšíření prohlížeče, které využije technoligii WebExtensions API pro analýzu a zpracování dat z dynamicky se měnících webových stránek. Rozšíření shromažďuje data, která odesílá ve formátu JSON na server, kde proběhne jejich zpracování. Následně jsou výsledky vráceny do rozšíření a prezentovány uživateli buď jako textový výstup nebo vizuálně jako překryvná vrstva na stránce. Díky použití WebExtensions API je rozšíření kompatibilní s širokým spektrem webových prohlížečů.

Abstract

The goal of this project is to create a browser extension that will use the WebExtensions API technology to analyze and process data from dynamically changing web pages. The extension collects data, which it sends in JSON format to the server, where it will be processed. Subsequently, the results are returned to the extension and presented to the user either as text output or visually as an overlay on the page. Thanks to using the WebExtensions API, the extension is compatible with a wide range of web browsers.

Klíčová slova

rozšíření prohlížeče, analýza webových stránek, překryvná vrstva, JavaScript, HTML, WebExtensions API, FitLayout

Keywords

browser extension, web page analysis, overlay layer, JavaScript, HTML, WebExtensions API, FitLayout

Citace

NOVOTNÝ, Jakub. *ROZŠÍŘENÍ WEBOVÝCH PROHLÍŽEČŮ PRO ANALÝZU STRÁNEK*. Brno, 2024. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Radek Burget, Ph.D.

ROZŠÍŘENÍ WEBOVÝCH PROHLÍŽEČŮ PRO ANALÝZU STRÁNEK

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. Ing. Radka Burgeta. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Jakub Novotný
9. května 2024

Poděkování

Chtěl bych především poděkovat mému vedoucímu práce doc. Ing. Radku Burgetovi za podnětné připomínky a kvalitní vedení při tvorbě práce. Dále bych chtěl poděkovat své rodině za dlouhodobou podporu studia na FIT VUT Brno.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Webové prohlížeče v dnešní době poskytují jeho uživatelům moderní, rychlou a intuitivní cestou k získávání informacím, zprostředkovaní workflow i k zábavě, za dodržení standartů soukromí a bezpečnosti v kybernetickém světě. Už dávno nejde jen o zobrazení webového obsahu. Rozšíření na straně klienta dokáží provádět komplexní funkce se stránkou a to v reálném čase bez nutnosti delegovat tento výpočetní výkon na server. Webová rozšíření značně usnadňují práci a umožňují webovým aplikacím fungovat podobně jako nativní aplikace.

Webové prohlížeče fungují tak, že odesílají požadavky na webové servery pomocí protokolu `HTTPS` nebo `HTTP`, přijímají odpovědi ve formě webových stránek (napsaných v HTML) a pak tyto stránky renderují pro uživatele. To zahrnuje zpracování HTML, CSS pro stylizaci a JavaScript pro interagování.

V bakalářské práci je podrobně rozebráno konkrétní webové rozšíření pro analýzu stránek. Rozšíření by mělo variantně doplnit možnosti aktuálně využívaného řešení, kde současné řešení generuje při renderování na serveru poměrně významnou zátěž a tento systém se poměrně obtížně ladí. Předchozí řešení má architekturu čistě serverové aplikace, provádí veškerou činnost se stránkou na straně serveru. Proto cílem je vytvoření klientské části aplikace formou rozšíření prohlížeče za dodržené všech webových standardů napříč platformami. Samozřejmě toto řešeni samo o sobě komplikuje funkčnost systému svou vlastní existencí, ale na druhé straně se přesto domnívám, že pomocí webového rozšíření prohlížeče je vhodným doplňkem tohoto řešení z hlediska zátěže serveru a snadnosti ladění. Toto rozšíření je tedy kompatibilní s nejrozšířenějšími webovými prohlížeči vytvořenými jak nad jádrem `Blink` (např. Chrome, Edge), tak nad jádrem `Gecko` (např. Firefox). Uživatel si toto rozšíření jednoduše nainstaluje a v intuitivním jednoduchém grafickém rozhraní může extrahovat data z webové stránky. Data jsou následně ve specifikované datové struktuře odesílána na server. Reakce serveru může být variabilní. Tok dat ze serveru je pak následně zpětně zpracován rozšířením a buď v textové podobě nebo v překryvné vrstvě zobrazen uživateli na webové stránce.

Práce je složena celkem ze 7 kapitol, každá se zaměřuje na odlišné stránky projektu předmětného webového rozšíření. Kapitola 2 popisuje souhrn použitých technologií pro tvorbu rozšíření, včetně potřebných programovacích jazyků, frameworků a API, s důrazem na potřebnou kompatibilitu s různými webovými prohlížeči. Kapitola 3 se věnuje vykreslovaní a interakci s API, Kapitola 4 se zabývá detailní popisem návrhu webového rozšíření, včetně designu, způsobu určené komunikace s serverem a úplné zpracování dat. Kapitola 5 se zaměřuje na technické detaily celkové implementace rozšíření. Popisuje použité metody vývoje rozšíření. Kapitola 6 pojednává o testovacích scénářích, odhalování problémů a výsledcích

testů. Kapitola 7 shrnuje výsledky tohoto projektu a navrhuje hlavní směry pro budoucí rozvoj, včetně nových zásadních funkcí a možností dalšího rozšíření.

Kapitola 2

Technologie pro tvorbu rozšíření webových prohlížečů

Technologie pro tvorbu rozšíření webových prohlížečů umožňuje a usnadňuje vývojářům vytvářet vlastní funkce a nástroje, které mohou výrazně zlepšit nebo upravit uživatelskou zkušenost z prohlížení webu. Tyto základní technologie se bohužel liší v závislosti na konkrétním prohlížeči, ale proti tomu existují trvalá snaha o společné zásady, komponenty a principy.

Průkopníkem této snahy jsou vývojáři webového prohlížeče Firefox. Rozšíření nebo doplňky, které mohou upravovat a vylepšovat možnosti prohlížeče, jsou vytvořena pomocí technologie WebExtensions API, která je v základu kompatibilní s různými prohlížeči [30].

2.1 Přehled technologií pro tvorbu rozšíření

Rozvoj rozšíření pro webové prohlížeče kombinuje oblast standardní webové technologie a specifické API poskytované různými prohlížeči. Zde je shrnutí klíčových technologií a postupů pro vývoj rozšíření:

HTML, CSS a JavaScript

Úplným základem většiny rozšíření webových prohlížečů jsou standardní webové technologie: HTML (HyperText markup language) pro jednoduché strukturování obsahu, CSS (Cascading style sheets) pro pokročilou vizuální stylizaci a JavaScript pro interaktivitu a logiku aplikace. Tyto technologie umožňují snadno a korektně vytvářet rozsáhlé uživatelské rozhraní a funkcionalitu, která může interagovat s jednotlivými webovými stránkami nebo konkrétním prohlížečem. [44, 13, 31, 39].

WebExtensions API

V současné době se vývoj posunul k nejpoužívanější a nejrozšířenější technologií pro vývoj rozšíření prohlížečů WebExtensions API. Toto základní rozhraní API bylo zavedeno jako poměrně úspěšný pokus o standardizaci rozvoje rozšíření mezi různými populárními prohlížeči, včetně Google Chrome, Mozilla Firefox, Microsoft Edge a Opera. WebExtensions API poskytuje sadu JavaScriptových API pro interakci s funkcemi konkrétního prohlížeče, jako jsou především karty, historie, oznámení a mnoho dalších. Díky tomuto přístupu mohou být rozšíření snadněji přenositelná mezi různými prohlížeči. [31, 43, 6, 30]

Manifest soubor

Rozšíření pro webové prohlížeče nutně zahrnuje i soubor manifestu, který je psán ve formátu JSON. Tento soubor definuje základní informace o rozšíření, jako je jeho název, verze, ikony, popis, práva a definice různých komponent rozšíření (např. background skripty, content skripty, uživatelská rozhraní jako popupy nebo možnosti webové stránky). [22, 31, 43]

Polyfill pro WebExtensions API

Pro usnadnění vývoje cross-browser rozšíření a zajištění kompatibility mezi různými prohlížeči lze využít WebExtension browser API Polyfill. Tento polyfill [1] řeší rozdíly v `namespace` API a způsobu zpracování asynchronních událostí, umožňuje použití Promise (příslibu) pro asynchronní API ve všech hlavních prohlížečích. [31, 37]

Vývoj a ladění

Významný přínos představuje možnost vývoje, testování a ladění rozšíření lokálně v prohlížečích, což zahrnuje nahrání rozšíření do prohlížeče a použití vývojářských nástrojů pro identifikaci a opravu chyb [37].

2.2 Úskalí a výhody WebExtensions

WebExtensions mají své nesporné výhody, ale i úskalí, které mohou uživatele i vývojáře potkat. V zásadě ale výhody této technologie výrazně převažují.

Výhody:

- **Jednoduché rozhraní:** API WebExtensions je koncipováno tak, aby bylo maximálně přístupné a intuitivní při tvorbě, což umožňuje efektivně vytvářet rozšíření bez hlubokých znalostí nižších vrstev interních mechanizmů webových prohlížeče. [44, 30]
- **Univerzální přenositelnost:** Jedná se o zásadní výhodu. Díky jednotnému standardu lze rozšíření snadno adaptovat pro použití v různých webových prohlížečích, což zásadně eliminuje potřebu specifických úprav pro každou platformu zvlášť. [5, 44, 30]
- **Kompatibilita:** Správně napsaná rozšíření jsou navržena zásadně tak, aby byla v souladu s aktuálními webovými standardy, což zajišťuje jejich dlouhodobou udržitelnost a použitelnost.[5, 44, 30]
- **Zvýšená bezpečnost:** Rozšíření se spouští v izolovaném prostředí (`sandboxu`) a tím se brání přímému přístupu instrukcí k systémovým zdrojům a uživatelským datům, což výrazně zvyšuje bezpečnost používání internetu uživateli. [5, 44, 30]
- **Optimalizace výkonu:** Přestože rozšíření běží v sandboxu, jsou navržena tak, aby byla co nejvíce výkonná a efektivní, minimalizují tedy dopad na rychlost a výkon prohlížeče. [5, 44, 30]

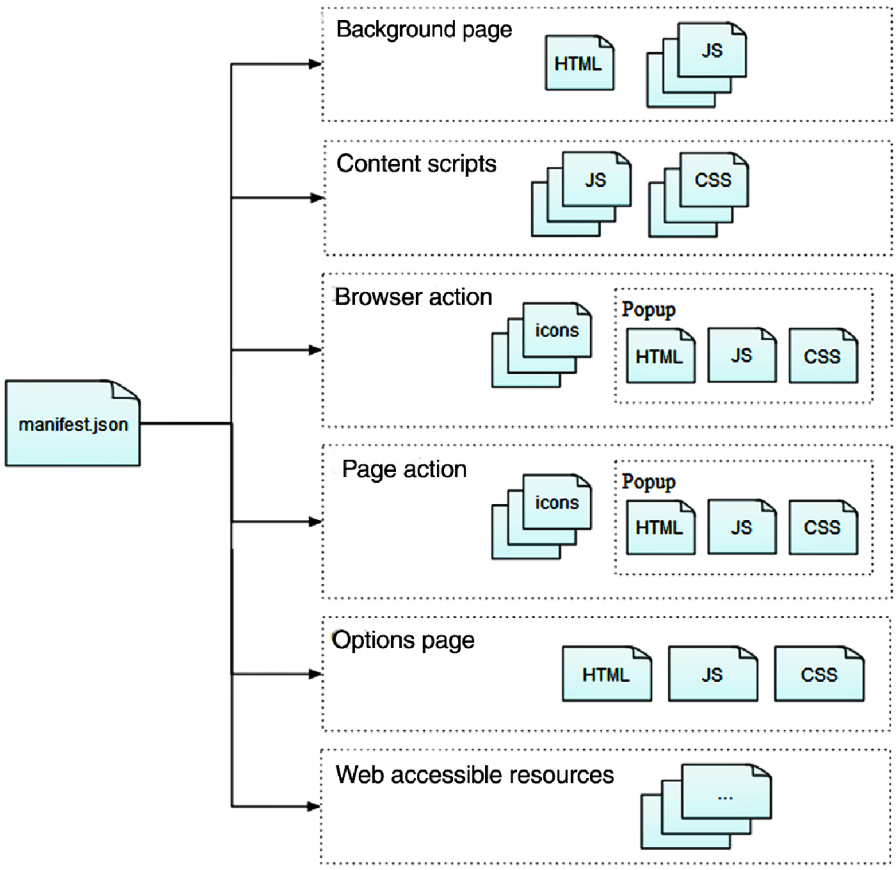
[1] Slovo "polyfill" se v kontextu webového vývoje používá a odkazuje na kód (obvykle JavaScript), který slouží k implementaci funkcionality napříč platformami a kterou například starší prohlížeče nepodporují [24].

Úskalí:

- **Často omezený přístup k některým funkcím:** Pokročilé funkce webového prohlížeče mohou být pro WebExtensions často nedostupné, což zásadně omezuje možnosti vývoje funkcionalit daného rozšíření. [44, 30]
- **Problematika kompatibility:** I přes snahu o standardizaci může docházet k problémům s kompatibilitou rozšíření mezi různými prohlížeči a to hlavně ve formě zvolených oprávnění. [5, 44, 30]

2.3 Základní komponenty a architektura WebExtensions

Struktura rozšíření pro webové prohlížeče (webextensions) je unikátní kombinací několika různých technologických prvků a především standardů a společně tvoří funkční, interaktivní a použitelný doplněk.

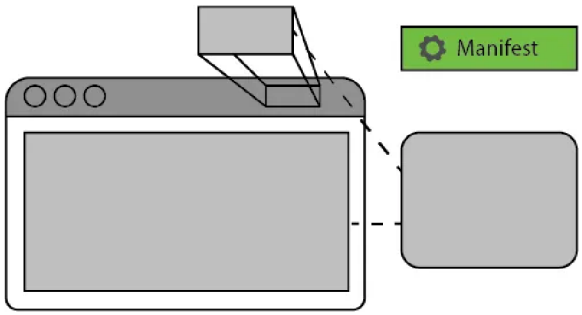


Obrázek 2.1: **Architektura WebExtensions** [14].

Manifest

Manifest soubor je základním kamenem každého rozšíření prohlížeče. Jedná se o JSON soubor, obvykle pojmenovaný `manifest.json`, který obsahuje důležité metadata a nastavení

rozšíření. Tento jedinečný soubor informuje prohlížeč o přesném názvu daného rozšíření, jeho verzi, ikonách, přidělených oprávněních a jiných důležitých vlastnostech, které jsou zásadní pro jeho správnou funkci. Díky manifestu se prohlížeč dozví, jak se má rozšíření chovat a co může dělat. [56, 22]



Obrázek 2.2: **Anatomie prvku Manifest** [56].

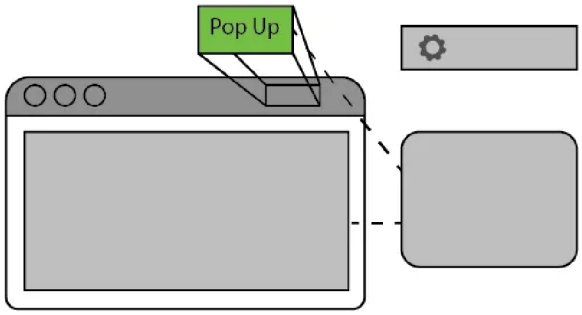
Manifest soubor se skládá z klíčů a jejich hodnot. Zde je uveden popis vybraných nejdůležitějších klíčů.

- **manifest version:** Udává verzi manifestu, kterou rozšíření používá. Aktuálně se doporučuje používat již verze 3. [22]
- **name:** Jméno rozšíření, jak se zobrazí uživatelům. [22]
- **version:** Verze rozšíření, která pomáhá identifikovat aktualizace a revize. [22]
- **description:** Krátký popis funkcionality rozšíření. [22]
- **icons:** Objekt definující ikony rozšíření pro různé velikosti. [22]
- **permissions:** Seznam oprávnění, které rozšíření vyžaduje pro přístup k funkcím prohlížeče nebo datům na webových stránkách. [22]
- **background:** Definuje skripty nebo stránky, které běží na pozadí a udržují rozšíření funkční i když není aktivně používáno. [22]
- **content scripts:** Určuje skripty, které se mají načíst do webových stránek a jaké stránky mají být ovlivněny. [22]
- **browser action nebo page action:** Umožňuje rozšíření přidat ikonu do panelu nástrojů prohlížeče, s možností otevřít vyskakovací okno nebo zobrazit nějakou akci, když je na ikonu kliknuto. [22]
- **options page nebo options ui:** Definuje stránku nebo uživatelské rozhraní pro nastavení rozšíření, na kterém je umožněno uživatelům upravovat chování rozšíření. [22]

Každý klíč a hodnota v manifestu mají často zásadní vliv na fungování webového rozšíření. Od zajištění, že je webové rozšíření správně identifikováno a klasifikováno prohlížečem, až po definování, jak rozšíření interaguje s webovými stránkami a samotným prohlížečem.

Pop Up

Soubor Popup je interaktivní částí, která umožňuje webovým rozšířením zobrazovat uživatelské rozhraní přímo v použitém prohlížeči. Často při kliku na příslušnou ikonu webového rozšíření umístěnou v některé části panelu nástrojů. Tento soubor bývá obvykle vytvořený jako HTML soubor a spolu s příslušnými CSS a JavaScript soubory slouží jako výchozí brána ke všem funkcím a nastavením webového rozšíření, nabízí uživatelům webového prohlížeče variabilní interakce a poskytuje nejrychlejší přístup k hlavním vlastnostem rozšíření. [25]



Obrázek 2.3: **Anatomie prvku Pop Up** [56].

Soubor Popup se v zásadě skládá z částí HTML, CSS a JavaScriptu. HTML definuje strukturu rozhraní, CSS definuje jeho vzhled a JavaScript umožňuje interaktivitu.

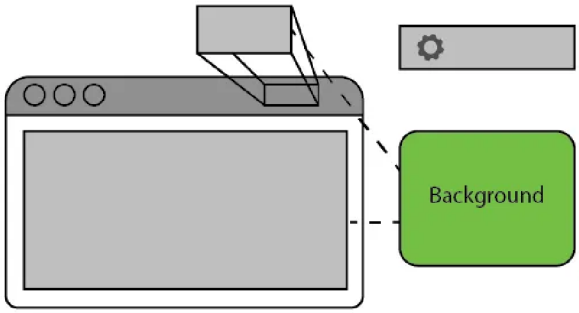
- **HTML:** Základní struktura popupu, která definuje layout a obsahové prvky. [25]
- **CSS:** Stylování popupu pro zajištění konzistentního vzhledu a rozlišení. [25]
- **JavaScript:** Skripty, které řídí interaktivitu popupu, včetně reakcí na uživatelské akce, komunikace s pozadím rozšíření pro načítání dat nebo uložení uživatelských preferencí. [25]

Vytváření efektivního a uživatelsky přívětivého popupu je zásadní, jelikož je to často první bod interakce mezi uživatelem a funkcionalitou rozšíření. Dobře navržený popup může výrazně přitáhnout zájem uživatele webového prohlížeče, usnadni navigaci a zlepšit celkový dojem a iteraktivnost z daného rozšíření.

Background

Soubor Background, jak název napovídá, funguje v zásadě na pozadí a je pro uživatele skrytým a nepřetržitě fungujícím motorem webového rozšíření. Tento soubor, obvykle realizovaný jako JavaScript nebo HTML soubor, je zodpovědný za řízení hlavních operací rozšíření, jako je naslouchání událostem, interakce s webovými stránkami, ukládání dat, provádění periodických úloh, a správa komunikace mezi různými částmi rozšíření, včetně popup okna, content skriptů, a dalších. [15]

Jednou z klíčových vlastností background souboru je jeho schopnost být aktivní i když jsou ostatní části rozšíření neaktivní. To umožňuje rozšířením provádět akce v reálném čase, reagovat na události prohlížeče, aktualizace z externích serverů, nebo plánovat úlohy, které se mají provést v určeném čase. [15]



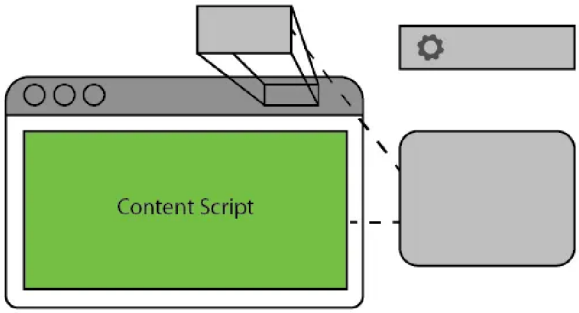
Obrázek 2.4: **Anatomie prvku Background** [56].

- **Reagování na události prohlížeče**: Například otevření nové karty, změny URL, aktualizace záložek a další. [15]
- **Asynchronní operace:** Umožnění provádět operace na pozadí, aniž by to blokovalo uživatelské rozhraní. [15]
- **Správa stálých dat:** Ukládání uživatelských preferencí, historie akcí nebo jiných dat mezi relacemi prohlížeče. [15]
- **Komunikace s externími API:** Získávání dat z webových služeb, synchronizace s cloudovými službami, odesílání notifikací atd. [15]
- **Interakce s content skripty:** Odesílání zpráv a řízení chování skriptů, které běží na webových stránkách. [15]
- **Plánování úloh:** Provedení akcí v pravidelných intervalech nebo v specifický čas. [15]

Pro správné fungování na pozadí je nezbytné, aby byl background soubor definován v manifest souboru rozšíření. Tím se zajišťuje, že prohlížeč ví, jaký soubor použít jako základ pro pozadí rozšíření a umožňuje automatické spuštění při startu prohlížeče nebo aktivaci rozšíření. [15]

Content Script

Soubor Content script představuje zásadní most mezi vzájemně izolovaným prostředím rozšíření a webovými stránkami, na které se uživatelé podívají prostřednictvím webového prohlížeče. Tento soubor bývá často napsaný v JavaScriptu, je vložen (injection) přímo do dané webové stránky. To mu umožňuje číst ze stránky hodnoty a na základě toho manipulovat s jejím obsahem. Na základě tohoto mechanismu mohou webová rozšíření přidávat do webových stránek nové funkce jako: možnost měnit vzhled nebo chování stránek, nebo extrahovat z nich specifické informace, a to vše při zachování bezpečnosti a izolace od samotného kódu rozšíření. [17]



Obrázek 2.5: **Anatomie prvku Content Script** [56].

Content scripty mají přístup k široké škále API webového prohlížeče, ale jsou omezeny v tom, jak mohou interagovat s jinými částmi rozšíření, jako jsou background skripty a musí komunikovat prostřednictvím zpráv. Tento design chrání uživatele tím, že dochází k blokaci přímého přístupu webového rozšíření k chráněným datům na webových stránkách a také umožňuje webovému rozšíření nabídnout užitečné a upravené funkce. [17]

Využití content scriptů v praxi představuje opravdu širokou škálu možností:

- **Změny UI:** Přizpůsobení vzhledu webových stránek, nejčastěji změnou stylů nebo přidáním nových prvků do stránky. [17]
- **Interaktivní funkce:** Vkládání objektů tlačítek, dalších ovládacích prvků nebo mnoha variant interaktivních elementů, které nabízí mnohé další funkce, jako je například stahování videí, formátování textu nebo integrace s některými externími službami. [17]
- **Automatizace a makra:** Vykonávání předem definované akcí na webových stránkách, jako je například automatické vyplňování formulářů nebo klik na tlačítko. Uživatelům mohou výrazně šetřit čas nebo usnadnit opakující se úkoly. [17]
- **Shromažďování dat:** Extrahování informací z webových stránek, jako jsou kontaktní informace nebo textový obsah, pro analýzu nebo archivaci. [17]

Aby byl content script úspěšně vložen (injection) do webové stránky, musí být správně definován v manifestu rozšíření, včetně určení, které skripty se mají načíst, a na jakých URL se mají spouštět. Toto umožňuje rozšíření mít přesnou kontrolu nad tím, kde a kdy se skripty spustí. To může výrazně zvyšit bezpečnost a efektivitní ochranu webového rozšíření. [17]

2.4 Bezpečnostní aspekty a omezení webových rozšíření

Jedná se především v současné době o klíčovou oblast. Ochranu uživatelů a zachování integritního a bezpečného prostředí při prohlížení webu klade vysoké nároky na konfiguraci

omezení ve webextensions. Zároveň webová rozšíření prohlížeče často v dnešní době představují zásadní nástroj pro úroveň uživatelské interakce na internetu. Proto je tak důležité znát hlavní bezpečnostní aspekty a omezení, která jsou implementována pro minimalizaci rizik spojených s používáním webextensions.

Přístup k datům a omezení oprávnění

Webextensions v zásadě mohou mít přístup i k citlivým datům uživatelů a k funkcionalitám webového prohlížeče. Aby bylo zajištěno, že rozšíření nezneužije tato oprávnění, vývojáři musí v manifestovém souboru specifikovat, které konkrétní oprávnění jejich rozšíření vyžaduje. Prohlížeče jako Chrome, Firefox a Edge pak tato oprávnění zobrazují uživatelům při instalaci rozšíření, což jim umožňuje učinit informované rozhodnutí o tom, zda oprávnění udělit či nikoli. [5]

Izolace kódu a SandBox

Neopominutelným způsobem zajištění bezpečnosti rozšíření a uživatelů, webového prohlížeče je izolace kódu mechanismem sandboxingu. Tyto techniky zásadně omezují možnost webového rozšíření interagovat s jinými aplikacemi nebo dokonce s operačním systémem Tím se výrazně snižuje riziko průniku útočníka nebo viru a zabraňuje se následným škodlivým akcím. Nepřekročitelnou hranicí je to, že skripty spuštěné v rámci webového rozšíření nemají přístup k lokálnímu souborovému systému uživatele ani nemohou spouštět libovolné binární kódy. [5]

Zásady aktualizace a kontrola zabezpečení

Webové prohlížeče velmi často obsahují speciální systémy pro automatickou aktualizaci webových rozšíření. To umožňuje vývojářům rychlou distribuci bezpečnostních záplat a aktualizací. Naštěstí webové prohlížeče ve standartu provádějí určité předběžné kontroly webových rozšíření před jejich publikací v oficiálních webových obchodech To zahrnuje kontrolu na přítomnost v té době známého škodlivého softwaru a zabezpečení kódu. Spolehlivost této ochrany je na poměrně vysoké úrovni. [4]

Komunikace mezi webovými rozšířeními a webovými stránkami

Komunikace mezi rozšířeními a webovými stránkami je v zásadě přísně regulována. Rozšíření musí explicitně vyžádat oprávnění k interakci s určitými webovými stránkami a webové stránky mohou omezit, která rozšíření s nimi mohou interagovat. Toto důležité omezení pomáhá chránit uživatele před útoky prostřednictvím cross-site scripting (XSS) a dalšími zákeřnými webovými zranitelnostmi. [4]

2.5 Vývoj a distribuce

Kromě distribuce prostřednictvím oficiálních obchodů s prohlížeči mohou vývojáři také zvolit samostatnou distribuci. Samostatná distribuce webových rozšíření představuje alternativní metodu, jak dosáhnout konečných uživatelů mimo tradiční kanály jako jsou oficiální obchody s prohlížeči (např. Chrome Web Store nebo Firefox Add-ons). Tento přístup umožňuje větší kontrolu nad distribucí a aktualizacemi svých rozšíření, ale na druhou stranu vyžaduje velkou profesionalitu ve zvládání možných rizik v kyberprostoru. [11]

```
<div id="example-option-1" class="install-ok">
  <a href="https://example.com/path/to/extension.xpi">
    Install my add-on
  </a>
</div>
```

Obrázek 2.6: **Možnost distribuce webového rozšíření Firefoxu pomocí přímého odkazu na XPI soubor (přejmenovaný zip se soubory rozšíření)** [11].

Lokální vývoj

V Mozilla Firefox existují dvě speciální interní adresy, které umožňují nahrání rozšíření do webového prohlížeče: `about:debugging` a `about:addons`. Každá z nich má specifické využití, a to zejména v kontextu vývoje, testování a správy rozšíření. Podepisování rozšíření hraje zásadní roli v obou případech. Důsledky jsou ovšem odlišné pro vývojáře a koncové uživatele. [8, 7]

about:debugging

Je vítaný nástroj pro vývojáře, kterým umožňuje dočasné načtení rozšíření a výrazně snazší testování a ladění. Umožňuje vývojářům načíst rozšíření, které ještě nebylo podepsáno. Tato funkce je užitečná pro vývoj a testování rozšíření před jeho odesláním pro oficiální podepisování a distribuci přes AMO [2]. Rozšíření zde fungují pouze do restartování Firefoxu, což znamená, že nejsou trvale instalována do prohlížeče. [8, 7]

about:addons

Je správce doplňků Firefoxu, kde uživatelé mohou spravovat všechna svá nainstalovaná rozšíření, témata a další doplňky. Zde můžete povolovat, zakazovat, odstraňovat, aktualizovat rozšíření nebo měnit jejich nastavení. Pro normální používání Firefoxu musí být všechna rozšíření nainstalovaná přes `about:addons` podepsána Mozillou. Firefox ve svém standardním nastavení nepovolí instalaci nepodepsaných rozšíření kvůli bezpečnostním obavám. [8, 7]

Podepisování rozšíření

Nejprve musíme mít nainstalovaný `web-ext`. Tento nástroj můžeme nainstalovat globálně pomocí npm (Node package manager): `npm install --global web-ext`. Pro podepisování rozšíření pomocí web-ext musíme mít účet na AMO a musíme si vygenerovat API klíče. Podepisování se provádí příkazem: [29, 10]

```
web-ext sign --api-key=API_KEY --api-secret=API_SECRET
```

Tento příkaz dokáže vytvořit podepsaný `.xpi` soubor a připravit ho k distribuci. Klíče `api-key` a `api-secret` získáme z našeho účtu na AMO. [29, 10]

[2] addons.mozilla.org

Kapitola 3

Reprezentace zobrazené webové stránky a aplikační rozhraní webových prohlížečů

V této části se budu podrobněji věnovat způsobu, jakým prohlížeče zobrazují webové stránky a možnostem, které nabízejí aplikační programovací rozhraní (API) poskytovaná prostřednictvím WebExtensions. Tato API umožňují rozšířením interagovat s konkrétním obsahem webových stránek a využívat různé funkce webového prohlížeče.

Při startu webové stránky webový prohlížeč převádí především HTML, CSS a JavaScriptový kód na vizuální a často i interaktivní formát, se kterým může uživatel do jisté míry pracovat. Tento proces se skládá z několika komponent, které jsou vzájemně provázané:

3.1 Reprezentace webové stránky DOM

Reprezentace webové stránky pomocí DOM slouží k dynamické manipulaci s HTML. DOM umožňuje přistupovat k dokumentům a interagovat s nimi jako s hierarchickou stromovou strukturou. WebExtensions využívají tento model aby mohla manipulovat s obsahem stránky. Takový přístup umožňuje rozšíření aby přidávala (vykreslení překryvné vrstvy), upravovala a nebo odstraňovala specifické prvky stránky.

Dokumentový model objektů (DOM)

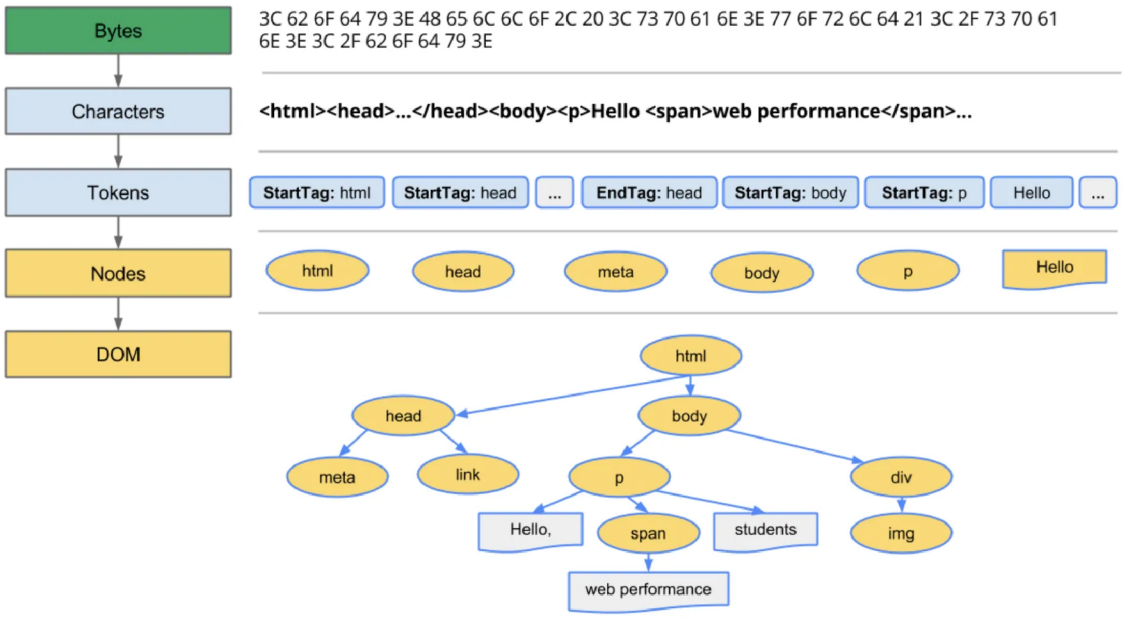
Dokumentový model objektů (DOM) je základní programátorské rozhraní pro HTML a XML dokumenty. Poskytuje přesně strukturovaný způsob, jak je dokument přístupný a jak se sním dá manipulovat. DOM zobrazuje dokument z hlediska stromové struktury. Každý uzel tohoto stromu odpovídá určité části dokumentu, jako je element, atribut nebo textový obsah. [20, 36]

- **Uzly (Nodes):** Uzly jsou základní stavební kameny DOM. Každý kus dokumentu je reprezentován jako uzel, včetně elementů, textových bloků a komentářů. DOM definuje různé typy uzlů, například `ElementNode` pro HTML tagy, `TextNode` pro textový obsah a `CommentNode` pro komentáře. Uzly jsou často spojeny do rozsáhlé stromové struktury, kde každý uzel může mít rodiče (parents), potomky (children) a sourozence (siblings), což umožňuje snadnou navigaci a manipulaci s dokumentem. [23]

- **Elementy (Elements):** Elementy jsou specifickým typem uzlu, který přímo odpovídá tagům v HTML nebo XML dokumentech. Každý element může dále obsahovat další elementy, text, nebo kombinaci obojího, což vytváří hierarchickou strukturu stromu. Elementy jsou v podstatě základními jednotkami pro strukturování dat a jsou přímo přístupné prostřednictvím rozhraní DOM za účelem, jako je například tvorba nových elementů, odstraňování stávajících elementů, nebo i často změna v jejich hierarchii. [38]

- **Atributy (Attributes):** Atributy jsou speciální typ uzlu, který poskytuje doplňující informace o elementech. Jsou to páry klíč-hodnota, které definují vlastnosti elementu, jako jsou `id, class, style` a další. Pomocí DOM je možné číst, přidávat, upravovat a mazat atributy elementů, což umožňuje dynamické změny ve vizuální prezentaci a funkčnosti elementů. [57]

  Vlastnosti (Properties): V momentě, kdy je DOM struktura přímo přístupná pomocí vhodných programovacích jazyků jako je JavaScript, elementy a uzly jsou reprezentovány jako objekty s vlastnostmi, které přímo odrážejí jejich strukturu a stav. Tyto vlastnosti tedy umožňují programový a objektový přístup k charakteristikám uzlů, jako jsou `textContent` pro získání nebo nastavení konkrétního textového obsahu uzlu, `innerHTML` pro manipulaci s HTML obsahem v rámci elementu, nebo `parentNode` pro přístup k rodičovskému uzlu. [57]



Obrázek 3.1: **Vizualizace procesu zpracování webového dokumentu: od bytů k DOM** [51].

Objektový model kaskádových stylů (CSSOM)

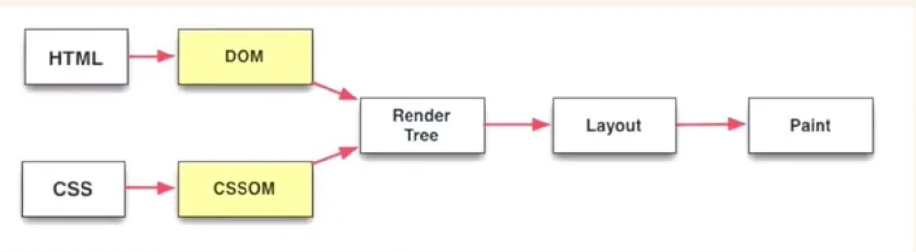
Model objektů kaskádových stylů (CSSOM) je velmi rozsáhlý a dynamický API, který umožňuje skriptům na straně klienta jednoduše manipulovat s kaskádovými styly, které jsou

použity na webové dokumenty. CSSOM rozšiřuje možnosti základního DOM o interakce se styly, což umožňuje programům číst a měnit styly v reálném čase. [34, 9]

- **Selektory:** Selektory v CSSOM identifikují HTML elementy, které mají být stylizovány. Mohou to být jednoduché selektory (např. název tagu, třída, id) nebo složitější kombinace selektorů. V CSSOM jsou selektory reprezentovány jako část objektů CSSRule, kde definují, na které elementy se dané styly aplikují. [35]
- **Vlastnosti (CSSStyleDeclaration):** Vlastnosti v CSSOM jsou specifické charakteristiky stylů, které lze aplikovat na elementy, například barva, velikost písma, okraje a další. Tyto vlastnosti jsou definovány v rámci `CSSStyleDeclaration`, což je objekt přidružený k jednotlivým pravidlům. Skripty mohou číst a upravovat hodnoty těchto vlastností, což umožňuje dynamické změny ve vzhledu stránky. [19]
- **CSSRule:** Pravidla v CSSOM jsou konkrétní direktivy, které kombinují selektory a vlastnosti, aby určily, jak by měl být konkrétní HTML element stylizován. V CSSOM jsou reprezentována jako objekty CSSRule, které mohou být součástí `CSSStyleSheet`. Tato pravidla jsou většinou přidávána, odebírána nebo modifikována za použití JavaScriptu. [18]
- **Dědičnost:** Dědičnost v CSSOM umožňuje velmi důležitou vlastnost. Některé styly se mohou "dědit"z rodičovských elementů na potomky. To znamená, že pokud je styl aplikován na rodiče, může být automaticky aplikován i na jeho děti, pokud není explicitně přepsán. CSSOM zpracovává tuto dědičnost automaticky a umožňuje skriptům zjišťovat výsledné styly, které jsou aplikovány na elementy. [32]
- **Kaskádové řazení:** Kaskádové řazení je základní metoda, kterou CSSOM používá k rozhodnutí, která pravidla se přednostně aplikují v případě, že existují více pravidel které kolidují u stejné vlastnosti na stejném elementu. Tento proces zahrnuje prioritizaci pravidel na základě specifičnosti selektorů, důležitosti (`!important`) a pořadí zdrojů. CSSOM umožňuje programové rozhraní pro zjišťování a manipulaci s výsledným stylem, což zahrnuje zjištění, které pravidlo má nakonec přednost. [33, 54]

Vykreslovací strom a layout

Spojením DOM a CSSOM vytvoří prohlížeč vykreslovací strom, který zahrnuje všechny vizuální elementy stránky a jejich styly. Na základě tohoto stromu prohlížeč vypočítá layout, tedy rozmístění a velikost každého prvku, a následně vykreslí finální vizuální reprezentaci stránky na obrazovku uživatele. [51]



Obrázek 3.2: **DOM render** [62].

3.2 Interakce s API s obsahem stránky

WebExtensions také umožňují rozšíření interagovat s obsahem webové stránky prostřednictvím `ContentScript`. Tyto skripty sice běží ve stejném prostředí jako cílová webová stránka, ale jsou bezpečně oddělené od skriptů, které patří samotné stránce. Tato jejich důležitá vlastnost jim umožňuje webové rozšíření číst nebo měnit informace na webových stránkách, reagovat na události v dokumentu, nebo manipulovat s DOM stránky. [30, 21]

Asynchronní zpracování a sliby (promises)

Nejmodernější WebExtensions API používají pro zpracování asynchronních operací tzv. sliby (promises), což zásadně zjednodušuje jakýkoliv vývoj webového rozšíření, které umí provádějí úlohy na pozadí, zadávat síťové požadavky a další složitější operace u kterých je nutné čekání na dokončení určené operace. Jedná se tedy o výraznou inovaci oproti dřívějším metodám založeným na zpětných voláních (`callbacks`) a je naštěstí plně podporována ve Firefoxu, Chrome a dalších hlavních prohlížečích. [21, 31]

Kompatibilita napříč prohlížeči

Jednou z hlavních výzev při kvalitním vývoji WebExtensions je zajištění kompatibility napříč různými platformami a webovými prohlížeči. Zatímco naštěstí základní funkce API jsou v zásadě kompatibilní napříč webovými prohlížeči, existují významné rozdíly v implementaci a dostupných nadstavbových funkcích. Často se používají tzv. polyfilly k nutnému řízení těchto rozdílů, což umožňuje udržet jednotný základ kódu, který funguje v několika prostředích a zároveň pro některé webové prohlížeče nabídnout více možností. [31]

API jsou přístupná prostřednictvím jmenných prostorů `browser.*` nebo `chrome.*`, přičemž specifický jmenný prostor, který se používá, závisí na prohlížeči. Jmenný prostor `browser.*` je standardem, který se používá ve Firefoxu a Safari, zatímco `chrome.*` se používá v Chrome, Opera a Edge. Tento standardizovaný přístup usnadňuje tvorbu rozšíření, která fungují napříč různými prohlížeči s minimálními změnami. [31]

API pro interakci s prohlížečem

WebExtensions API nabízí několik speciálních APIs, které také umožňují webovým rozšířením interagovat s různými funkcemi webových prohlížeče. Podrobný popis některých klíčových API:

- **Tabs API:** umožňuje jednoduše manipulovat s kartami prohlížeče. Vývojáři mohou pomocí tohoto API otevírat nové karty, získávat informace o aktuálně otevřených kartách, zavírat karty, přesouvat je mezi okny a měnit jejich vlastnosti, jako je URL, které zobrazují. Toto API také umožňuje skvěle reagovat na události související s kartami, jako je jejich tvorba, aktualizace, přesun nebo smazání. [27]
- **Bookmarks API:** poskytuje funkce pro správu záložek v prohlížeči. Vývojáři mohou pomocí tohoto API vytvářet, měnit, mazat a organizovat záložky. Můžete také načíst stromovou strukturu záložek, aby rozšíření mohla zobrazovat seznam záložek nebo synchronizovat záložky s externím serverem. [16]
- **Notifications API:** umožňuje webovým rozšířením ukazovat upozornění a notifikace. Tyto notifikace mohou jednoduše informovat uživatele o určených důležitých

událostech nebo změnách a to i v momentě, když rozšíření není aktivně používáno. API podporuje základní textové notifikace, stejně jako rozšířené možnosti, včetně tlačítek a ikon. [40]

- **WebRequestAPI:** umožňuje rozšířením sledovat, analyzovat a manipulovat s HTTP požadavky, které prohlížeč posílá a přijímá. Toto API je užitečné pro účely jako je blokování nebo upravování požadavků, filtrování obsahu nebo analýza bezpečnostních hlaviček. Lze zasahovat do různých fází životního cyklu požadavku, včetně jeho inicializace, přípravy k odeslání a zpracování odpovědi. [42]
- **Storage API:** poskytuje rozšířením možnost ukládat a načítat data lokálně nebo synchronizovaně (přes různá zařízení uživatele). Toto API je ideální pro ukládání uživatelských nastavení, dat z formulářů nebo jakýchkoli jiných dat, která rozšíření potřebuje pro svůj provoz. Data jsou uložena v klič:hodnota (key:value) formátu a lze je snadno a rychle získat za pomocí speciálních asynchronních API volání. To umožňuje opravdu efektivní správu dat bez nějakého blokování uživatelského rozhraní. [26]

3.3 Ladění a testování

Ladění a testování jsou zásadní v procesu vývoje webového rozšíření. Využívaní integrovaných nástrojů pro vývojáře ve webových prohlížečích (jako jsou DevTools v Chrome nebo Firefox) má zásadní dopad na efektivitu práce vývojáře a testera. Důležitá je především kompatibilita s různými webovými prohlížeči a bezpečnost daného webového rozšíření. Řízený proces nejenže umožní detekovat a opravit zjevné i skryté chyby, ale také ověřit, že uživatelské rozhraní a interakce rozšíření fungují podle očekávání zákazníka a uživatele. Lokální testování a nahrávání rozšíření do prohlížeče se může na první pohled zdát složitější oproti stažení z obchodů rozšíření, avšak je to relativně přímočarý proces. [12, 41, 1]

Krok 1: Příprava webového rozšíření

Podmínkou zahájení testování je existence zdrojového kódu, který je připravený a uložený lokálně na konkretním počítači. To zahrnuje všechny potřebné soubory, jako je manifestový soubor (`manifest.json`), HTML soubory pro uživatelské rozhraní, JavaScriptové skripty pro logiku rozšíření a CSS soubory pro styly. [12]

Krok 2: Nahrání do Prohlížeče

Pro Google Chrome:

1. Otevřete Chrome a přejdete na stránku `chrome://extensions/`. [12]
2. Zapnete "Režim vývojáře" v pravém horním rohu. [12]
3. Kliknete na "Načíst rozbalené" a vyberete kořenový adresář rozšíření. [12]

Pro Mozilla Firefox:

1. Otevřete Firefox a přejdete na `about:debugging`. [28]
2. Kliknete na "Toto zařízení" a poté na "Načíst dočasný doplněk". [28]
3. Vyberete soubor manifestu v projektovém adresáři. [28]

Krok 3: Testování a Ladění

Po nahrání rozšíření do webového prohlížeče se může okamžitě začít s testováním. Oba webové prohlížeče v základu nabízejí velni dobře použitelné vývojářské nástroje, které poslouží při ladění webového rozšíření, sledovaní chyb webové stránky, monitorování i výkonu a procházení různých uživatelských scénářů. Ve Firefoxu lze pro ladění použít nástroj "Prozkoumat", zatímco Chrome obsahuje podobné možnosti prostřednictvím svých DevTools přístupných přes stránku rozšíření nebo kliknutím pravým tlačítkem a výběrem "Prozkoumat" na uživatelském rozhraní rozšíření. [41, 1]

Krok 4: Vylepšení

Na základě výsledků testování je dalším logickým krokem provédení úpravy v kódu. Po každé úpravě je nutné rozšíření v prohlížeči aktualizovat, což v zásadě znamená jeho znovu načtení prostřednictvím uživatelského rozhraní pro vývojáře prohlížeče. [12]

Cyklus vývoje, testování, ladění je základem pro vytvoření všech funkčních, uživatelsky přivětivých a bezpečných webových rozšíření. Přístup k testování rozšíření lokálně a nahrání do prohlížeče je velmi důležitou částí vývoje.

Kapitola 4

Návrh rozšíření webového prohlížeče

Při návrhu webového rozšíření pro webový prohlížeč je základem vize a definice očekávání a cílů daného rozšíření. V této kapitole se podrobně zaměříme na proces přípravy návrhu a na strategii rozvržení tohoto webového rozšíření. Rozbor zahrne klíčové funkcionality, které rozšíření musí obsahovat a přístupy k jejich implementaci.

Vstupní požadavky

Výchozím bodem mého projektu byla již existující serverová aplikace FitLayout [1] (rozebráno v podkapitole 4.1), která obsahuje backendovou službu pro renderování stránek pomocí Puppeteeru. Tento nástroj pro analýzu stránek, používá webový prohlížeč serveru a dálkově ho řídí. Zadáním práce bylo vytvoření vhodné a efektivní metody renderování přímo v prohlížeči uživatele pomocí webového rozšíření. Předpokládaným výsledkem je snížení zátěže serveru, zjednodušení ladění a zajištění vyšší úrovně kompatibility. Různé verze prohlížečů, odlišné fonty, velikost okna a další faktory mohou na serveru způsobit problémy. Uživatelé se také mohou do svých prohlížečů přihlásit a vykonávat další interaktivní kroky před samotnou analýzou, což je na serveru obtížně realizovatelné.

4.1 FitLayout

Návrh rešeni navazuje na již existují analýzu stránek pana doc. Ing. Radka Burgeta a to konkrétně na FitLayout - Puppeteer pro FitLayout [48].

Framework analýzy webových stránek FitLayout

FitLayout je framework určený pro modelování, vykreslovaní a analýzu webových dokumentů. FitLayout nabízí rozhraní Java API, spolu s rozhraním příkazové řádky (`CLI`) a Web API (`REST`), což umožňuje nasadit do provozu kompletní proces zpracování webových stránek zahrnující následující funkce. [50, 59]

[1] https://layout.fit.vutbr.cz/, https://github.com/FitLayout/FitLayout

Jednotný model pro vykreslené webové stránky

Je model v Javě, který popisuje stránky na úrovni boxů generovaných renderovacím enginem, nezávisle na původním formátu stránky. Tento model je vhodný pro další analýzu, například pomocí algoritmů segmentace. [50, 59]

Možnosti vykreslování:

- **Playwright a Puppeteer:** Tyto moduly, založené na vzdáleně ovládaném prohlížeči Chrome, umožňují vykreslování dynamických webových stránek. Playwright je integrován přímo do FitLayout, zatímco Puppeteer vyžaduje samostatný backend na `Node.js`. [50]
- **CSSBox:** Jednoduchý Java renderer pro rychlé vykreslování statických stránek HTML+CSS a PDF dokumentů, který nevyžaduje externí prohlížeč. [50]
- **PDF renderer:** Založený na Apache PDFbox, umožňuje vykreslování PDF dokumentů s možností výběru rozsahu stránek a faktoru přiblížení. [50]

Algoritmy segmentace stránek:

- **VIPS:** Segmentace založená na vizuálním vnímání stránky. [50]
- **BCS:** Segmentace pomocí shlukování bloků. [50]
- **Seskupování vizuálních oblastí:** Konfigurovatelná metoda segmentace odspodu. [50]

Artefakty

Artefakty jsou výstupy z analýzy stránek, jako vykreslené stránky nebo segmentace, což je produkt analýzy. Jedná se například o vykreslenou nebo segmentovanou část stránky. Tyto artefakty mohou být vytvořeny nově nebo mohou být odvozeny od již existujících artefaktů. Tímto procesem vzniká strom artefaktů. [49]

Typy artefaktů

Artefakty mohou být různých typů a to:

- **Page (strom boxu):** Zobrazuje stránku jako strom boxů vytvořených z DOM stránky. [49, 59]
- **Area tree:** Staví na vykreslené stránce a ukazuje rozdělení stránky na vizuální oblasti. [49, 59]
- **LogicalAreaTree a Text Chunk Set:** Poskytují další analytické pohledy na data získaná z vykreslených nebo segmentovaných stránek. [49, 59]

Služby artefaktů

Jedná se o moduly, které vytvářejí artefakty daného typu. Každá služba má svůj jedinečný identifikátor, který ji umožňuje její spuštěni. [49]

Serializace artefaktů

Artefakty lze ukládat v různých formátech, jako je RDF/XML, Turtle, XML, HTML nebo PNG. [49]

Úložiště artefaktů

Slouží k ukládání artefaktů pro další zpracování. Artefakty mohou být uloženy dočasně v paměti nebo trvale v RDF úložišti, které umožňuje integraci a dotazování artefaktů. [49, 59]

Identifikace artefaktů

Každý artefakt ve FitLayout má unikátní identifikátor zvaný IRI (Internationalized resource identifier), který vypadá jako URL adresa, například `http://fitlayout.github.io/resource/art261`. Tento identifikátor je používán jen interně a není určen pro vyhledávání na internetu. [49]

FitLayout - Puppeteer

FitLayout Puppeteer je backendová komponenta určená pro platformu FitLayout, která je speciálně navržena pro vykreslování webových stránek. Využívá Puppeteer, knihovnu Node, která nabízí API na vysoké úrovni pro ovládání Chrome nebo Chromium přes protokol DevTools a díky tomu umožňuje FitLayout Puppeteer řídit integrovaný webový prohlížeč Chromium pro přesné vykreslení. Hlavní funkcí tohoto backendu je získání boxové reprezentace webových stránek po jejich vykreslení. To znamená, že dokáže přesně zachytit, jak jsou prvky na webové stránce rozloženy a zobrazeny v prostředí prohlížeče, včetně rozměrů, pozic objektů a dalších aspektů CSS stylů, které ovlivní vizuální stránku webového obsahu. [48, 61, 52, 47, 59, 46]

Integrací Puppeteeru je renderer FitLayout Puppeteer schopen automatizovat úkoly prohlížeče, jako je navigace po stránkách, pořizování snímků obrazovky a analýza obsahu stránek a to vše v prostředí `headless browser` nebo s viditelným uživatelským rozhraním, v závislosti na vstupních požadavcích. Tato schopnost je klíčová pro úkoly jako web scraping, automatizované testování webových stránek a aplikací a výzkum analýzy webového rozložení. [48, 61, 52]

Pro shrnutí, backend FitLayout Puppeteer poskytuje nástroj pro platformu FitLayout, který umožňuje automatizované, přesné vykreslení a analýzu rozložení webových stránek pomocí instance prohlížeče Chrome ovládaného Puppeteerem. To umožňuje podrobné zkoumání a manipulaci s boxovým modelem webové stránky, což přispívá k širokému spektru aplikací ve vývoji a analýze. [48, 47, 59]

Popis vlastností ve výstupním formátu JSON


```
{
  "id": 14,
  "xpath": "//body[1]/*[1]/*[1]/*[1]/*[1]/*[1]/*[3]/*[1]/*[1]/*[1]/*[1]",
  "tagName": "SPAN",
  "x": 123.015625,
  "y": 18.15625,
  "width": 65.125,
  "height": 16,
  "hasBgImage": false,
  "parent": 13,
  "domParent": 13,
  "css": "font:16px / 16px -apple-system, BlinkMacSystemFont, \"Segoe UI\", Roboto,
    Helvetica, Arial, sans-serif, \"Apple Color Emoji\", \"Segoe UI Emoji\", \"Segoe UI
    Symbol\";display:block;position:static;color:rgb(73, 80, 87);background-color:rgba(0, 0,
    0, 0);border-top:0px none rgb(73, 80, 87);border-right:0px none rgb(73, 80, 87);border
    -bottom:0px none rgb(73, 80, 87);border-left:0px none rgb(73, 80, 87);overflow:visible
    ;transform:none;visibility:visible;opacity:1;",
  "attrs": [
    {
      "name": "class",
      "value": "p-menuitem-text"
    },
    {
      "name": "data-pc-section",
      "value": "label"
    }
  ],
  "istart": 0,
  "iend": 1
},
```

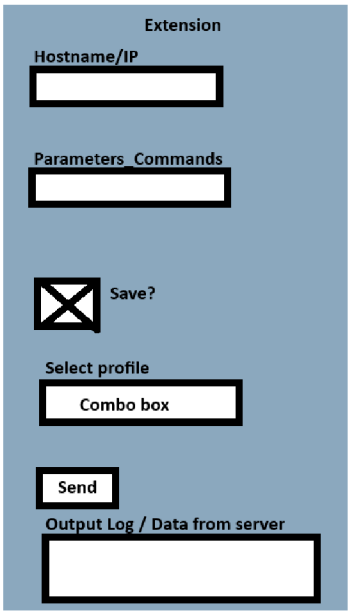

Obrázek 4.1: **Výstupní JSON, který popisuje element (box) webové stránky**.

- **id:** Unikátní identifikátor elementu, v tomto případě 14. Tento identifikátor se používá pro jednoznačné odkazování na element v rámci DOM.
- **xpath:** XPath výraz, který udává umístění elementu ve struktuře DOM dokumentu. V tomto příkladu
  `//body[1]/*[1]/*[1]/*[1]/*[1]/*[1]/*[3]/*[1]/*[1]/*[1]/*[1]`
  ukazuje na hlubokou vnořenost elementu od kořenového elementu `<body>`.
- **tagName:** Název tagu elementu, zde `"SPAN"`, což naznačuje, že se jedná o inline element obvykle používaný pro obalení textu nebo jiných malých částí HTML dokumentu.
- **x a y:** Souřadnice elementu na stránce, udávající jeho pozici od levého horního rohu okna prohlížeče. V tomto případě je element umístěn na souřadnicích x: 123.015625 a y: 18.15625.
- **width a height:** Rozměry elementu v pixelech, které určují jeho šířku (65.125 px) a výšku (16 px).
- **hasBgImage:** Boolean hodnota, která stanoví, zda element obsahuje obrázek na pozadí. V tomto příkladu je hodnota false. To znamená, že element nemá na pozadí žádný obrázek.
- **parent a domParent:** Identifikátory rodičovského elementu v DOM. V tomto případě oba ukazují na identifikátor 13, což znamená, že element je přímým potomkem elementu s ID 13.

- **css:** Řetězec CSS vlastností aplikovaných na element, který definuje jeho vizuální prezentaci, včetně fontu, barvy, pozadí, ohraničení, viditelnosti a dalších atributů.
- **attrs:** Pole objektů, každý reprezentující HTML atribut elementu. V tomto případě element obsahuje dva atributy: class s hodnotou `p-menuitem-text`, což je CSS třída používaná pro stylování, a `data-pc-section` s hodnotou "label", což je vlastní datový atribut.
- **istart a iend:** Indexy, které mohou být použity pro určení pozice elementu v rámci určité sekvenční struktury nebo pro analýzu obsahu. V tomto případě jsou hodnoty 0 a 1.

4.2 Wireframe

Prvním krokem pro vizualizaci obsahu rozšíření je nutné navrhnout wireframe a to po analýze dosavadní aplikace a konzultaci s vedoucím práce. Wireframe je strukturální náčrt webové stránky, který slouží jako základ nového návrhu, preferující rozložení obsahu, funkčních prvků a hierarchie informací, aniž by se přímo zaměřoval na grafické detaily, jako jsou barvy nebo typografie v rozšíření. Minimalisticky představuje zjednodušený vizuální průvodce, který zobrazuje, kde se budou nacházet různé prvky webové stránky, jako jsou texty, obrázky a navigační prvky. Až v dalších krocích bude dotvořen detailní design a začne vývoj vlastního webového rozšíření. [55, 2]



Obrázek 4.2: **Wireframe grafického rozhraní rozšíření**.

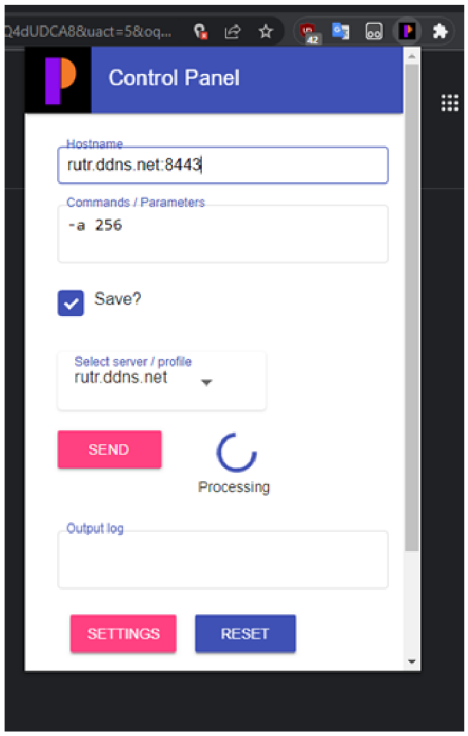
Na rozdíl od mockupu, který přidává vizuální detaily a interaktivitu, wireframe se soustředí na otázku co a kde má být [55, 60].

4.3 Architektura a design rozšíření

Po uvážení obsahu je vytvořen mockup rozšíření za využití technologii HTML a CSS. Pro dokonalejší design bylo využito hnihovny Material Design Lite.

Mockup

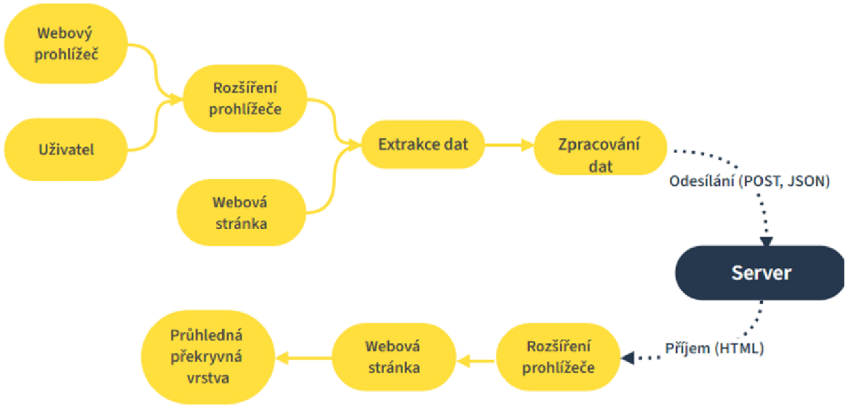
Pro detailnější vizualizaci rozšíření a ujasnění základního layoutu a požadavků, je vytvořen mockup. Mockup v grafickém uživatelském rozhraní (GUI) je vizuální návrh nebo prototyp, který demonstruje, jak bude finální webová stránka vypadat a fungovat, aniž by bylo nutné psát skutečný kód řešení nebo implementovat funkčnosti. Tato detailnější prezentace vzhledu umožňuje designérům a vývojářům vizualizovat rozložení prvků, barevné schéma, typografii a celkový estetický dojem ještě před jeho skutečným vývojem. [55, 45]



Obrázek 4.3: **Prvotní mockup navrženého GUI rozšíření.**

4.4 Interakce mezi rozšířením, webovou stránkou a serverovou aplikací

Uživatel si zobrazí webové stránky přes webový prohlížeč, posléze interaguje s rozšířením, které zprostředkuje informace o webové stránce. Následně jsou data zpracována a odeslána na server pomocí REST API s použitím metody POST ve formátu JSON. Server tato data zpracuje a vrátí odpověď, která může nabývat různých forem, od textové zprávy po fragment HTML kódu. Tato odpověď je dále zpracována rozšířením a v případě HTML kódu zobrazena v překryvné vrstvě nad aktuálně zobrazenou stránkou.



Obrázek 4.4: **Diagram aktivit architektury rozšíření**.

Kapitola 5

Implementace

U této kapitoly se detailně podíváme na implementaci vytvořeného rozšíření pro prohlížeče. Implementované rozšíření je postaveno na kombinaci hned několika technologií a struktura je organizována do více souborů. Aby byla zajištěna správná implementace bylo nezbytné provést důkladnou analýzu stávajících metod analýzy webových stránek od pana doc. Ing. Radka Burgeta, která posléze byla převzata a upravena pro prostředí rozšíření.

5.1 Technologie a nástroje

Při tvorbě rozšíření bylo využito hned několik technologii a nástrojů pro úspěšný vývoj. Zde je uvedený jejich seznam:

HTML5 a CSS3

Základní vizuální struktura rozšíření je definována pomocí značkovacího jazyka HTML5. Díky tomu lze vytvořit strukturovaný obsah webové stránky a tím vstupní a výstupní rozhraní rozšíření. CSS3 je použito pro stylování a další externí styly z knihovny Material design lite (MDL) a vlastní styly (`style.css`) zajišťují finální vizuální prezentaci.

Material design lite (MDL)

MDL je knihovna od vývojářů společnosti Google, která poskytuje sadu komponent ve stylu Material design, které lze použít v webových aplikacích a rozšířeních. Je použita pro zlepšení vizuálního vzhledu webového rozšíření. To zahrnuje tlačítka, textová pole, zaškrtávací políčka a indikátory načítání. [3]

JavaScript a Web APIs

Dynamické chování rozšíření, jako je odesílání formulářů, zpracování uživatelských vstupů a interakce s prohlížečem, je řízeno pomocí JavaScriptu. Externí skript `material.min.js` je součástí MDL a zajišťuje komponentům MDL správně fungovat (např. animace načítání, animace tlačítek). `Popup.js` je vlastní skript, který zprostředkovává logiku GUI pro rozšíření, jako je obsluha uživatelského rozhraní.

Web Extensions API

Manifest (manifest verze 3) specifikuje metadata rozšíření, požadovaná oprávnění ("permissions"), ikony, obsahové skripty (`content_script.js`), a další nezbytné aspekty rozšíření. Service worker (`background.js`) slouží jako background proces pro zachycení snímků.

Visual studio code

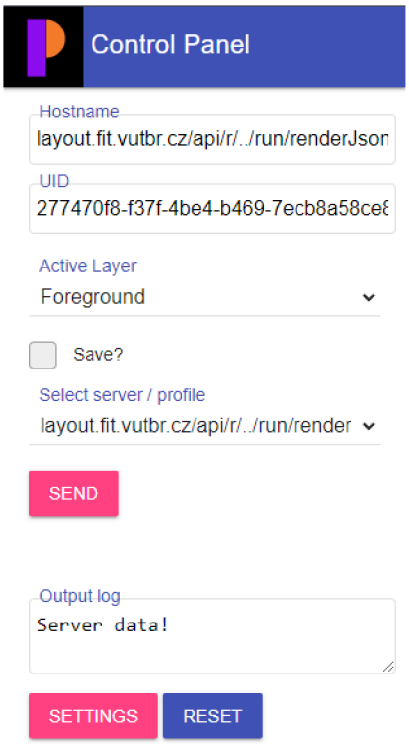
Visual studio code (VSCode) je vývojářský editor kódu, který poskytuje možnosti pro vývoj webových aplikací a rozšíření prohlížečů. Nabízí mnoho funkcí, jako je zvýrazňování syntaxe, debugger. VSCode má rozšíření (moduly) pro další podporu specifických jazyků a nebo dalších frameworků, díky tomu je ideálním nástrojem pro vývoj webových rozšíření.

5.2 Implementace

Implementace je složena z více vzájemně propojených komponent a dalších funkcí a díky tomu společně extrahují detailní informace z DOM (document object model) webové stránky. Níže je uveden podrobný přehled jeho unikátní struktury a funkcionalit:

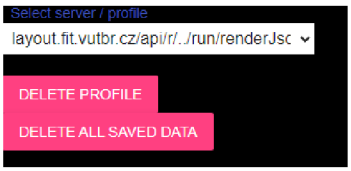
Implementované rozšíření pro prohlížeče je navrhnuto tak, že poskytne uživatelům interaktivní ovládací panel s moderním uživatelským rozhraním založeným na technologii material design lite (MDL). Taktéž dovoluje uživatelům zadávat a spravovat různorodá nastavení a vstupní parametry.

Rozhraní se otevírá v rámci aktivní části panelu prohlížeče ikonou s logem rozšíření. Tělo stránky je rozděleno do mřížky, kde hlavní formulář umožňuje uživatelům zadávat informace jako `hostname`, `unikátní identifikátor (UID)` a aktuální aktivní vrstvu. Možnost `Save?` se zaškrtávacím políčkem určuje, že uživatelé mohou své konfigurace uložit pro budoucí použití. Volitelné výběrové pole nabízí možnost vybrat z předdefinovaných serverů nebo profilů, zatímco interaktivní tlačítka umožňují odesílat data nebo přistupovat k dalším nastavením. Prvek `Active Layer` umožňuje uživatelům přepínat mezi dvěma základními vrstvami interakce - překryvnou vrstvou (foreground) a zobrazenou stránkou (background).



Obrázek 5.1: **Finální podoba GUI rozšíření (layout.html).**

Rozšíření rovněž nabízí funkci resetování změn ve vstupních polích, což uživatelům umožňuje snadno vrátit původní hodnoty. Dále obsahuje nastavení, které při aktivaci otevře novou kartu s konfigurací rozšíření. Uživatelé mají zde možnost smazat všechna uložená data profilů nebo se mohou rozhodnout pro vymazání pouze konkrétního profilu.



Obrázek 5.2: **Dostupná nastavení pro rozšíření (options.html).**

5.2.1 ContentScript.js

Inicializace a detekce písma (`"checkfont"`)

Tato funkce prověřuje, jestli je určité písmo aplikováno na elementy webové stránky a to srovnáním dvou pomocných elementů span, které jsou do stránky dočasně vloženy. Jednomu nastaví jako rodinu písma testované písmo s fallbackem na monospace, druhému pouze monospace. Pokud se rozměry obou elementů liší, je zřejmé, že prohlížeč testované písmo rozpoznal a použil. Po testu se pomocné elementy odstraní, aby neovlivňovaly vzhled ani funkčnost stránky. Originální zdroj funkce `checkfont`: [58].

Export boxů a analýza rozložení (`"fitlayoutExportBoxes"`)

Funkce `fitlayoutExportBoxes` prochází celým DOM stromem stránky, identifikuje všechny viditelné elementy a pro každý z nich vytváří "box", což je datová struktura obsahující informace o umístění, rozměrech a stylu elementu. Důležité jsou zejména vlastnosti jako pozice, barva, obrázky na pozadí, typ zobrazení, transformace a viditelnost, které určují jak vizuální, tak strukturální charakteristiku stránky. Funkce také pečlivě zachycuje informace o textových dekoracích a identifikuje nahrazené elementy, jako jsou obrázky a iframy.

Vytváření boxů

Proces začíná extrakcí geometrických informací (rozměry a pozice) a stylizačních atributů (barva, pozadí, fonty) z každého viditelného elementu stránky. Funkce `createBoxes` projde DOM stromem a pro každý element vytvoří "box", který uchovává klíčové vizuální informace.

Geometrické a stylové atributy

Pro každý element jsou shromažďovány následující informace:

- Rozměry a pozice: výška, šířka, X a Y pozice, které určují umístění elementu v rámci stránky.
- Styly: vlastnosti jako display, position, color, background-color a další, které definují vzhled elementu.
- Dekorace textu: zjišťuje se, zda je text podtržený či přeškrtnutý.
- Přítomnost obrázků: identifikuje, zda má element obrázek na pozadí nebo jestli jde o `<img>` nebo `<svg>` element.

Každý box tak reprezentuje jednu "vizuální jednotku" na stránce s vlastní sadou atributů a stylů, která ho definuje.

Výpočet super-obdélníků

Pro elementy skládající se z více obdélníků (například text rozdělený do více řádků) se vypočítá "super-obdélník", což je obdélník obalující všechny jednotlivé obdélníky elementu. Tento přístup umožňuje efektivnější analýzu a manipulaci s komplexními strukturami.

Vizuální hierarchie

Vytvořené boxy jsou také propojeny s jejich rodičovskými a potomkovskými elementy, což umožňuje zachovat vizuální hierarchii původní struktury DOM. Informace o rodičovských boxech pomáhají pochopit vnoření a kontext elementů.

Zpracování písem a obrázků

Detekuje se a zaznamenává použitá písma na stránce, což pomáhá sestavit seznam všech typografických stylů používaných v obsahu. Kromě toho identifikuje obrázky a elementy s obrázky na pozadí, kterým přiřazuje unikátní identifikátory. To umožňuje snadno identifikovat a analyzovat vizuální prvky stránky, což je zásadní pro plné pochopení jejího vizuálního designu.

Modifikace DOM pro detekci řádků (`"fitlayoutDetectLines"`)

Tento proces zahrnuje dočasné obalení textových uzlů vlastními elementy a následné zpracování těchto elementů k identifikaci a označení jednotlivých řádků textu. Tato technika umožňuje podrobněji analyzovat, jak je text na stránce rozložen a identifikovat, jak různé stylování a formátování ovlivňuje jeho prezentaci.

Extrakce metadat (`"extractJsonLd"`)

Extrakce JSON-LD metadat je rozhodující pro získání strukturovaných dat z webové stránky, jako jsou informace o organizaci, osobě nebo události, které jsou zabalené v rámci stránky v tomto specifickém formátu.

Vypnutí písem definovaných v CSS (`"disableCSSFonts"`)

Tato funkce prochází všechny CSS styly načtené na stránce a pokud najde odkazy na externí písma (např. z Google fonts), tyto styly deaktivuje. Cílem je zajistit, že při detekci použitého fontu nedojde k záměně s fontem, který není lokálně dostupný na počítači uživatele. To je užitečné u situací, kde je vždy nezbytné zaručit konzistenci vizuální prezentace webové stránky a to bez závislosti na externích zdrojích.

Následné zpracování detekce řádků

Po detekci a označení textových řádků skript dále analyzuje textové elementy, aby získal co nejpřesnější představu o rozložení textu na stránce. U tohoto kroku jde vidět, jak se text rozděluje do řádků v rámci různých elementů a jak toto rozdělení se jeví v celkovém vizuálním vzhledu webové stránky.

1. Vytvoření vizuálních kontejnerů

Pro detekci řádků je nejprve nutné mít text umístěný v elementech, které lze vizuálně analyzovat. Kód zahrnuje mechanismus pro obalení textových uzlů speciálními kontejnery (`<XX>` pro textové uzly a `<XL>` pro detekované řádky), které umožňují další zpracování.

2. Získání geometrických informací

Každý obalený textový uzel je analyzován z pohledu jeho vizuálního zobrazení na stránce. Pomocí metody `getClientRects()` je získán seznam obdélníků (rects), vyjadřující geometrickou pozici každé části textu (většinou jednotlivé řádky).

3. Rozpoznání konců řádků

Pro každý obdélník se porovnává jeho vertikální pozice (`rect.y`) s pozicí předchozího, aby se zjistilo, zda došlo ke změně řádku. Pokud se aktuální `y` pozice liší od pozice předchozího obdélníku, je to interpretováno jako konec řádku. Tato metoda efektivně rozpoznává, kdy text "skáče" na nový řádek.

4. Vytváření boxů pro řádky

Po detekci konce řádku se vytvoří nový vizuální kontejner pro každý řádek. Tyto kontejnery (`<XL>`) slouží jako vizuální reprezentace jednotlivých řádků textu. Proces se opakuje pro

každý řádek v textovém uzlu, čímž se zajišťuje, že celý text je správně rozdělen podle vizuální struktury stránky.

Asynchronní vyhodnoceni a výstup

V poslední části analýzy je definována asynchronní funkce `evaluatePage()`, která slouží k spuštění celého procesu analýzy (přes použitá písma, layoutové boxy, obrázky, až po metadata), sběru výsledků do jednoho JSON objektu a jeho odeslání na server nebo výpis do konzole. Tato funkce také zahrnuje mechanismus timeoutu, který zajistí ukončení procesu v případě, že by analýza trvala příliš dlouho. V rámci této funkce se také získává screenshot stránky prostřednictvím `browser.runtime.sendMessage`, kde `action` je nastaveno na `captureTab`. Odpověď z této metody je zpracována a v případě úspěchu je obrázek přidán do proměnné `pg.screenshot`.

Odesílání dat na cílový server a implementace překryvné vrstvy

Funkce `sendDataToServer(data)` ve zdrojovém kódu je asynchronní funkce pro odeslání dat na server prostřednictvím HTTP POST požadavku. Funkce upravuje URL a přidává HTTPS a UID, pokud ve vstupním poli je je tato URL a UID zkrácena kvůli přehlednosti. Následně se pomocí funkce `fetch` odesílají data s nastavenými HTTP hlavičkami pro obsah typu JSON.

```
method: 'POST',
headers: {
'Content-Type': 'application/json',
},
body: data
```

Pokud server vrátí odpověď bez chyb (`response.ok`), data byla úspěšně odeslána a funkce zpracovává odpověď serveru. Tato odpověď vytvoří nový HTML element, který se přidá do stránky jako překryvná vrstva:

- **Vytvoření prvku:**

  Nejprve se vytvoří nový HTML element `div` s identifikátorem `overlayDiv`. Tento element je nastaven jako absolutně pozicovaný, což znamená, že bude překrývat ostatní prvky na stránce bez toho, aby narušil jejich layout.

- **Stylování prvku:**
  - Styly jsou nastaveny tak, aby `div` pokryl celou obrazovku (width a height).
  - Pozadí je nastaveno na průhlednou barvu `rgba(0,0,0,0.0`, což znamená, že původní obsah stránky zůstane viditelný, ale překryvná vrstva je stále rozpoznatelná.
  - Index `z-index` je nastaven na hodnotu 1000, což zajišťuje, že překryvná vrstva bude zobrazena nad ostatními prvky stránky.
  - CSS vlastnost `pointerEvents` je nastavena podle hodnoty `request.layer`, což ovlivňuje, jak prvky reagují na ukazatel myši (například může být nastaveno, aby prvky pod překryvnou vrstvou byly neklikatelné).

- **Obsah prvku**

  Do `div` je vložena odpověď serveru ve formě HTML nebo textu. Tento krok umožňuje uživateli vidět odpověď serveru.

- **Přidání do stránky**

  Nakonec je `overlayDiv` přidán do `document.body` (v tomto případě na konci těla stránky), což způsobí, že se vrstva zobrazí na stránce.

V poslední části kódu je zavedena podmínka pro automatické spuštění funkce `evaluatePage()` závislé na stavu načtení dokumentu. Pokud je dokument již načtený (complete nebo interactive), funkce se spustí okamžitě. Pokud dokument ještě není plně načtený, registrace události `DOMContentLoaded` zajišťuje, že `evaluatePage` bude spuštěna, jakmile bude dokument připraven.

5.2.2 Manifest.json

Nastavení `permissions` definuje soubor oprávnění, která rozšíření potřebuje k tomu, aby mohlo správně fungovat a interagovat s prohlížečem a webovými stránkami. Tato oprávnění umožňují rozšíření vykonávat určité operace, které by jinak byly omezené. Pro realizaci funkce pořízení snímku obrazovky, rozšíření vyžaduje přístup k oprávněním jako `tabs` a `activeTab`, která umožňují manipulaci s kartami prohlížeče a získávání informací o aktivní kartě. K tomu jsou přidána oprávnění `storage` pro ukládání dat a `scripting` pro spouštění skriptů.

```
"permissions": [
    "tabs",
    "activeTab",
    "storage",
    "scripting"
]
```

Pro umožnění rozšíření interagovat s různými webovými stránkami je v konfiguraci v sekci `host_permissions` specifikován přístup k libovolným adresám HTTP a HTTPS. To rozšířenímu umožňuje pracovat s obsahem na těchto URL adresách. Uživatelské rozhraní pro nastavení tohoto rozšíření je definováno v souboru `options.html`, který se automaticky otevírá v nové kartě prohlížeče.

```
"host_permissions": [
    "http://*/*",
    "https://*/*"
]
```

V sekci `"content_scripts"` jsou zahrnuty skripty `browser-polyfill.min.js` a `contentScript.js`, které jsou injektovány do všech načítaných webových stránek, jak je specifikováno parametrem `"matches": ["<all_urls>"]`. Skript `browser-polyfill.min.js` zajišťuje kompatibilitu API rozšíření mezi různými prohlížeči, což dovoluje rozšíření prohlížeče fungovat mezi různými platformami prohlížečů.

```
"content_scripts": [
{
    "matches": ["<all_urls>"],
    "js": [
    "js/browser-polyfill.min.js",
    "contentScript.js"]
}
```

Kromě standardních hodnot typických pro manifest verze 3 byla implementována také specifická nastavení pro prohlížeče založené na jádru `Gecko` (Firefox). Pro toto rozšíření je specifikováno unikátní ID, které zajistí jeho jedinečnou identifikaci v rámci tohoto prohlížeče. [7]

```
"browser_specific_settings": {
    "gecko": {
    "id": "<email>"
}

"background": {
    "scripts": ["background.js"],
}
```

5.2.3 Popup.js

Po načtení obsahu konkrétní webové stránky, zprostředkuje daný skript interakci s uživatelským rozhraním webového rozšíření s načítající animací a zobrazení existujících uživatelských profilů v rozbalovacím seznamu. Toto umožňuje uživatelům vytvářet nové nebo upravovat stávající profily. Rozšíření k tomu využívá posluchače událostí, které reagují na uživatelské vstupy a to jako jsou změny v hodnotách formulářových prvků a kliknutí na tlačítka.

Tento skript dále obsahuje taktéž pokročilé funkce pro práci se vstupními daty, jako je validace formátu URL a UID pomocí regulárních výrazů. Díky tomu je zabráněno chybám při zadávání a zajišťuje správnou funkci odeslání dat na vzdálený server. Toto ověření je důležité pro správné propojení a funkčnost různých závislých komponent rozšíření.

Pro jednoduchost použití rozšíření je implementováno dynamická aktualizace uživatelského rozhraní, reaguje na změny v nastaveních a zobrazuje nebo skrývá grafické prvky (například indikátor načítání, UID), v závislosti na stavu daných operací analýzy a zpracování.

5.2.4 Background.js

Skript definuje funkci `screenshot()`, která asynchronně zpracovává požadavek na vytvoření screenshotu stránky a manipuluje s výsledkem nebo chybami pomocí javascriptových promises. Nejprve se získá seznam aktivních karet v aktuálním okně prohlížeče. Pokud dojde k chybě při získávání karty nebo pokud nejsou žádné aktivní karty, promise je zamítnuta s příslušnou chybovou zprávou. Pokud je karta dostupná, provede se `captureVisibleTab`, která snímá viditelnou část aktuální záložky ve formátu PNG. Po úspěšném získání snímku se výsledek předává volajícímu. Skript naslouchá příchozím zprávám od jiných částí rozšíření.

Když přijme zprávu s akcí `captureTab`, spustí funkci `screenshot`. Po dokončení screenshotu se výsledek nebo chyba vrátí aktuálnímu odesílateli zprávy pomocí `sendResponse`, což umožňuje komunikaci mezi specifikovanou částí rozšíření.

Pro zpracování z dat snímku aktivní karty je využito služeb regulárního výrazu a to pro odstranění metadat spojených s formátem obrázku, konkrétně prefixu `data:image/png;base64`, který je pro data obrázků zakódovaných ve formátu `Base64`. Po odstranění tohoto prefixu zůstávají čistá data obrázku ve formátu `Base64`, která jsou připravena k dalšímu zpracování v části `content_script.js`. Čistá data ve formátu `Base64` jsou pak vložena do JSON objektu s klíčem `"screenshot"`.

5.3 Integrace s existující serverovou aplikací

Před nasazením rozšíření prohlížeče byl implementován a používán lokální server vytvořený s použitím technologie `Node.js`. Tento server byl hostován na domácím dedikovaném serveru s operačním systémem Fedora, kde naslouchal na specifické lokální adrese a portu. Server přijímal zprávy a generoval výsledky ve formě JSON, které obsahovaly analýzu prohlížených webových stránek. Odpovědi od lokálního serveru zahrnovaly taky fragmenty HTML kódu a to umožňovalo další jejich zpracování.

Integrace webového rozšíření vyžadovala nejprve zajistit jeho spolehlivé propojení s cílovým vzdáleným serverem a zřízení cílové webové adresy a ID relace, které mě umožnily přímý přístup k serverové aplikaci. V dalším kroku byl určen název vstupního bodu (entry point) a požadovaný formát výstupních dat. Toto nastavení zajistilo, že rozšíření může komunikovat se vzdáleným serverem. Tato komunikace se odehrávala přes HTTPS protokol, zajišťující bezpečný přenos dat mezi rozšířením uživatele a vzdáleným serverem.

Kapitola 6

Testování

Testování bylo rozděleno do několika etap. Prvně bylo klíčové stanovit jasnou strategii a přesný plán testování. To především zahrnovalo jasnou definici jednotlivých testovacích scénářů a stanovení kritérií pro ověření funkčnosti. Testování muselo započít již v rané fázi projektu ještě před vlastním nasazením, kdy byly testy prováděny pouze lokálně. Prvním krokem bylo ověření funkčnosti základní komunikace mezi komponentami rozšíření, což zahrnovalo testování správnosti výměny dat a odpovědí mezi serverem a klientem. Tento přístup umožnil identifikovat a vyřešit potenciální problémy. V dalších fázích bylo provedeno porovnání výsledků při použití serverové aplikace a webového rozšíření, současně byla testována správnost vykreslení překryvné vrstvy v obou řešeních.

6.1 Strategie a plán testování

Testování bylo zahájeno důkladnou analýzou a ověřením funkčnosti po nasazení rozšíření na server. Testování se skládalo z několika na sebe navazujících fází, které postupně prověřily všechny aspekty webového rozšíření:

1. **Lokální testování:** První fáze projektu zahrnovala lokální testy. Ty byly zaměřeny na základní funkčnost a komunikaci mezi novým webovým rozšířením a cílovým serverem. Toto zahrnovalo ověření, že všechny požadavky na server a odpovědi ze serveru jsou korektně zpracovány a že neexistují žádné zjevné chyby v základních operacích aplikace.

2. **Integrace:** Po úspěšném lokálním testování bylo rozšíření upraveno pro komunikaci se vzdáleným serverem, kde probíhalo rozšířené testování v rámci stávajícího systému pro analýzu webových stránek FitLayout. Tento krok umožnil ověření funkčnosti rozšíření v reálných provozních podmínkách a nabídl podrobnější pohled na jeho vzájemnou interakci s dalšími systémovými komponentami.

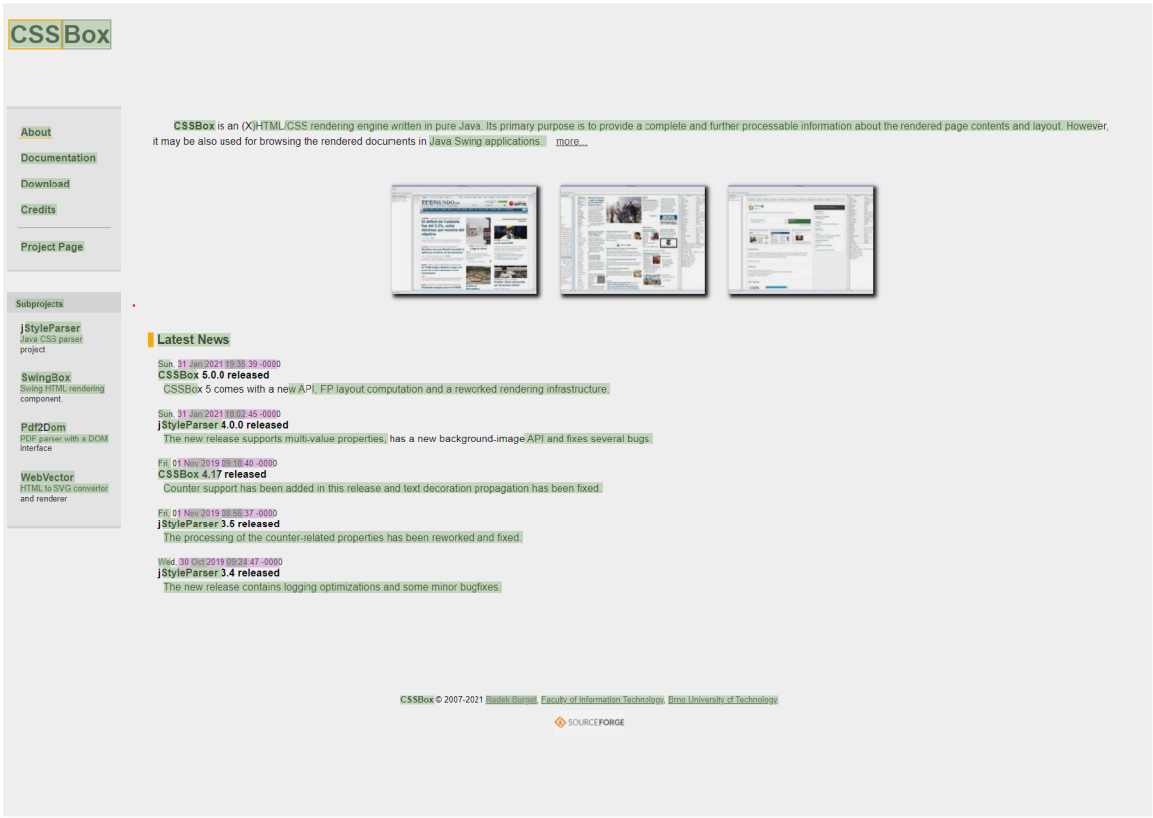
3. **Testování:** Další fáze se zaměřila renderováni webových stránek a v jiné fázi projekt zahrnoval výrazně komplexnější testovací scénáře, jako je sledování zatížení systému, ověření zabezpečení dat a kvality uživatelská interakce.

4. **Iterativní zlepšování a úpravy webového rozšíření:** Na základě výsledků opakovaných testů byly postupně identifikovány a implementovány potřebné úpravy. Tento přístup byl klíčový při zvýšení kvality finálního produktu před vlastním nasazením webového rozšíření do rutinního provozu.

5. **Závěrečné ověření funkčnosti:** Po úspěšném dokončení všech testovacích fází a provedení nezbytných úprav v několika testovacích cyklech bylo konečně nové webové rozšíření připraveno k finální nasazení. Tato fáze zahrnovala znovu i komplexní kontrolu všech aspektů webového rozšíření a dopadla s pozitivním výsledkem.

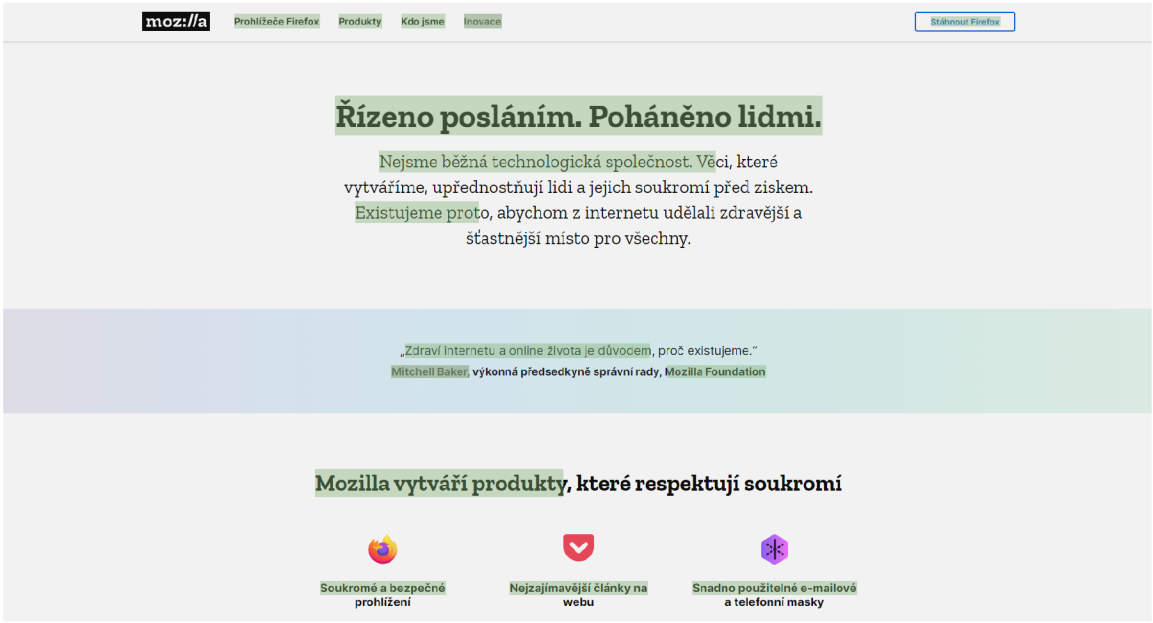
6.2 Testování funkcionality

Rozšíření bylo testováno na sadě náhodně vybraných webových stránek. Tyto vzorky sloužily pro základní ověření funkcí renderování. Bylo ověřeno správné renderování pozic barevných obdélníků v překryvné vrstvě nad stránkou s pozicemi textů na stránce.



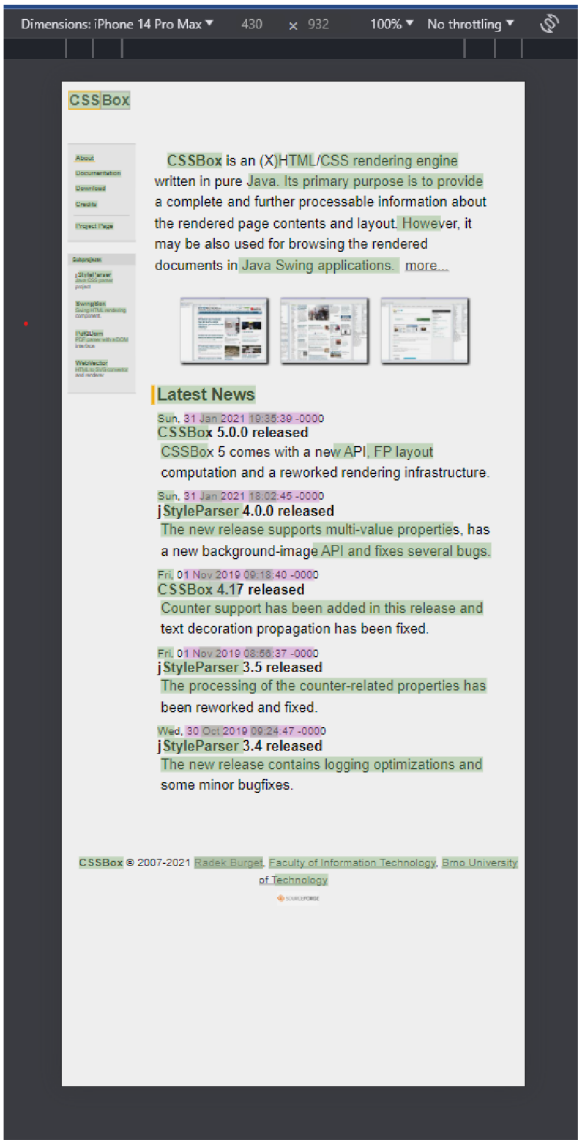
Obrázek 6.1: **Překryvná vrstva z testovacího endpointu `tagJson`.**

Příchozí část HTML z endpointu do rozšíření, jak vidíme na obrázku, je vykreslena odpovídajícím způsobem a kryje se s pozicí textu.



Obrázek 6.2: **Překryvná vrstva jiné stránky z testovacího endpointu `tagJson`.**

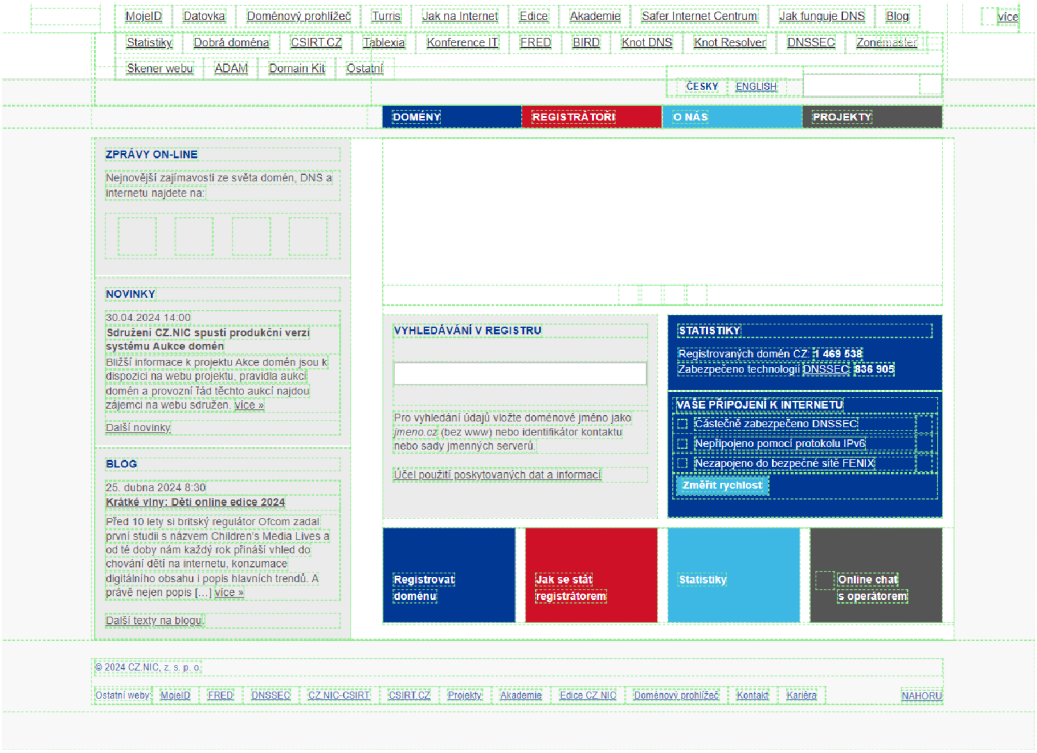
Testování následovalo při změně zobrazovací oblasti (`viewport`) na velikost odpovídající mobilnímu telefonu. Ověřila se tak přizpůsobivost rozhraní.



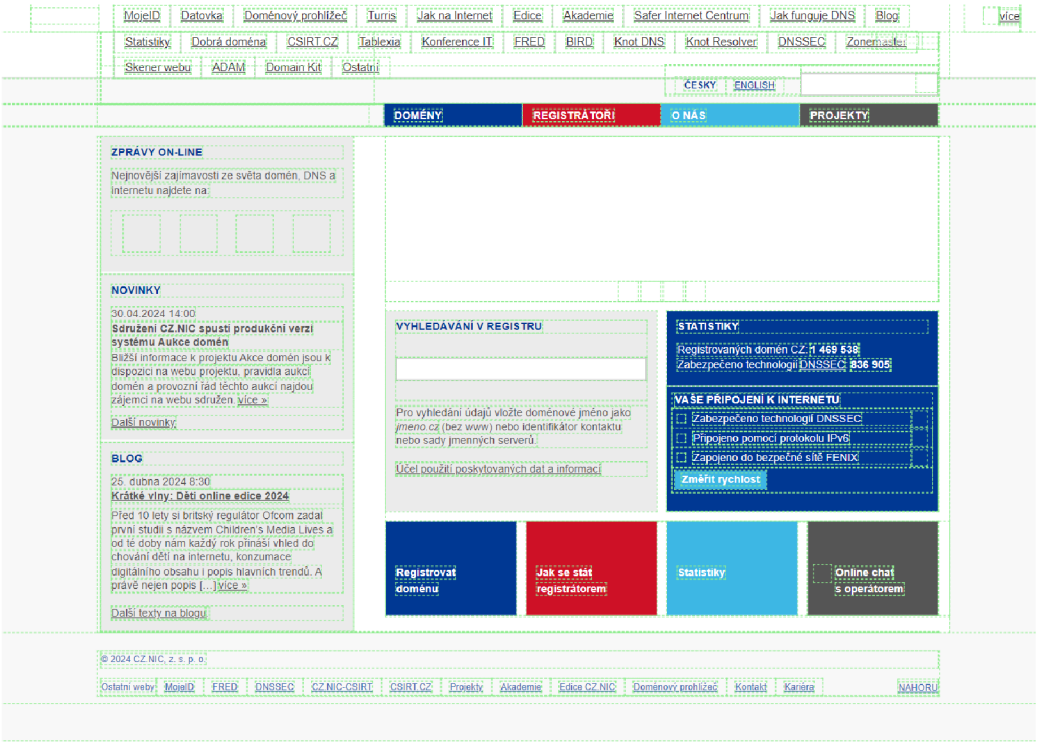
Obrázek 6.3: **Překryvná vrstva z testovacího endpointu `tagJson` odpovídající mobilnímu telefonu iPhone 14 Pro Max.**

Nové webové rozšíření bylo podrobně testováno ve skupině nejrozšířenějších prohlížečů, aby byla zaručena jeho co nejširší kompatibilita. Testy zahrnovaly prohlížeče Firefox, Edge, Chrome a Opera, což jsou v současnosti hlavní platformy používané širokým spektrem uživatelů k přístupu k internetu. U prohlížeče Firefox bylo rozšíření speciálně podepsáno, což je krok vyžadovaný Mozillou. Kromě toho rozšíření úspěšně prošlo sérií automatických testů prováděných Mozillou, což potvrzuje jeho bezpečnost a spolehlivost.

Na dalších testovacích sadách stránek bylo ověřeno správné vytvoření artefaktu na FitLayout. Tento testovací artefakt byl porovnán s nativní serverovou aplikací, kde byl zjištěn identický výsledek.

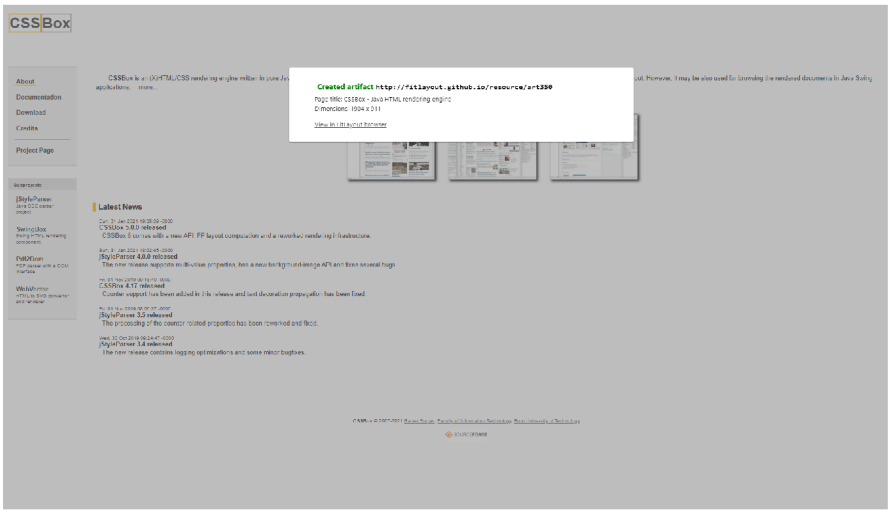


Obrázek 6.4: **Výsledný renderer přes FitLayout prostřednictvím serverové aplikace.**

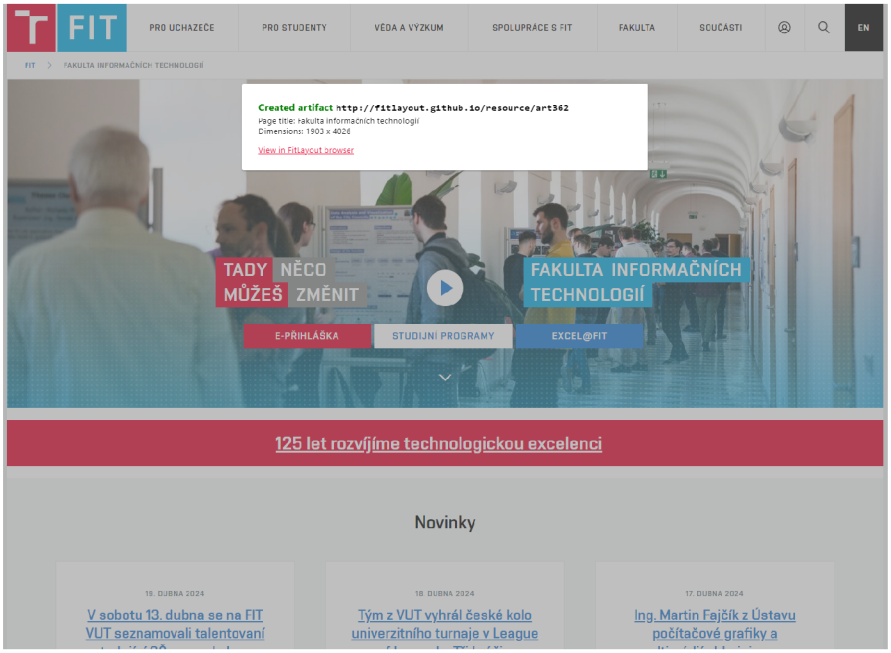


Obrázek 6.5: **Výsledný renderer přes FitLayout prostřednictvím nového webového rozšíření.**

Příchozí úryvek HTML ze serveru do rozšíření, jak vidíme na obrázcích, kde je vykreslen jako překryvná vrstva stránky, zobrazuje IRI adresu artefaktu.



Obrázek 6.6: **Překryvná vrstva z testovacího endpointu `renderJson`.**



Obrázek 6.7: **Překryvná vrstva jiné stránky z testovacího endpointu `renderJson`.**

6.3 Vyhodnocení

Všechny testovací scénáře potvrdily teoretické předpoklady funkčnosti nového webového rozšíření a jeho rovnocenné funkce s původním serverovým řešením. Proto lze konstatovat, že toto webové rozšíření je pro tento účel plně funkční.

Potencionální rozšíření

Potenciální rozšíření pro budoucí vývoj vidím v možnostech vylepšení funkcionality pro pořízení snímků obrazovky, které by umožnilo zachytit celou webovou stránku, včetně částí momentálně neviditelných v prohlížeči. Aktuálně WebExtensions API nedovoluje snímat celou stránku najednou, pouze její viditelnou část. Možným řešením by bylo postupné snímání jednotlivých segmentů stránky s následným posunem a spojováním těchto snímků do jednoho celku. Tento proces však komplikuje limitace API, specificky omezení `MAX_CAPTURE_VISIBLE_TAB_CALLS_PER_SECOND`, které omezuje frekvenci snímkování. Kromě toho by bylo třeba vyřešit i problémy s dynamickým obsahem, jako jsou posouvající se menu, která by mohla ovlivnit konzistenci snímků. Totéž by mohlo platit pro zlepšení schopností analýzy a efektivnější správu více otevřených karet v prohlížeči současně a její případné propojení s relační databází.

Mezi dalšími uvažovanými rozšířeními funkčnosti tohoto nově vytvořeného nástroje by mohla být především integrace s `html2canvas`, která dokáže získat snímky celých webových stránek a to i dokonce včetně těch částí, které nejsou v momentu použití přímo zobrazeny ve webovém prohlížeči. Tento nástroj by mohl překonat stávající omezení WebExtensions API, které umožňuje zachytávat pouze viditelné části stránek. Pomocí `html2canvas` by bylo možné pořídit snímek celé délky stránky tím, že se stránka "vykreslí" do plátna (canvas) a následně se toto plátno konvertuje na obrázek. Tato metoda by zároveň vyřešila problém s omezením počtu snímků za sekundu. `Html2canvas` ale nemusí správně zachytit všechny CSS styly, zejména ty, které jsou složitější nebo méně běžné. Příklady zahrnují některé transformace, filtry a pokročilé stínování. [53]

Kapitola 7

Závěr

Tato má bakalářská práce podrobně zkoumá vývoj klientského rozšíření webového prohlížeče a v praktické části na konkrétním projektu přináší vetší variabilitu ve způsobu analýzy webových stránek oproti již tradičnímu serverovému řešení. Toto webové rozšíření dokáže analyzovat a zpracovávat data přímo na straně klienta, což výrazně snižuje celkovou zátěž cílového serveru.

Opravdu důležitým aspektem této práce je popis vývoje a implementace pří plném respektu k webovým standardům a kompatibilitě s nejrozšířenějšími prohlížeči napříč platformami. Integrace webového rozšíření do prohlížečů jako Chrome, Edge, Firefox a Opera názorně demonstruje univerzálnost a variabilnost tohoto řešení při striktním dodržení všech standardů bezpečnosti. Grafické uživatelské rozhraní rozšíření je navrženo tak, aby bylo maximálně uživatelsky přívětivé, což následně umožňuje velmi efektivní extrakci a analýzu dat webových stránek.

V budoucnu by mohlo na tento úspěšný projekt navázat další výzkum v oblasti možné integrace s dalšími moderními technologiemi a inovativní rozšíření funkcí tohoto nového nástroje. V první řadě se nabízí možnost integrace modulu umožňujícího pokročilé snímání webových stránek.

Výsledky testování plně potvrdili teoretické předpoklady, s kterými jsem zahájil práci na tomto webovém rozšíření. Bylo prokázáno, že překryvná vrstva v testování je v akceptovatelném rozmezí kvality a je totožná při použití serverového verze aplikace i prostřednictvím webového rozšíření.

S odstupem času jsem rád, že jsem si vybral toto téma práce, protože jsem měl možnost seznámit s velmi zajímavou technologií a naučil jsem se tvorbě webových rozšíření pro prohlížeče. Také jsem mohl navázat na zajímavou problematiku analýzy stránek, ve které je významným odborníkem můj vedoucí práce.

Literatura

[1] *Debug extensions* online. Google Developers, 2012. Dostupné z: https://developer.chrome.com/docs/extensions/get-started/tutorial/debug. [cit. 2024-03-12].

[2] *What is Wireframing?* online. Interaction Design Foundation - IxDF, 2016. Dostupné z: https://www.interaction-design.org/literature/topics/wireframing#what_is_wireframing?-0. [cit. 2024-04-10].

[3] *Material Design Lite* online. Google Design, 2017. Dostupné z: https://getmdl.io/. [cit. 2024-03-12].

[4] *Protect user privacy* online. Google Developers, 2018. Dostupné z: https://developer.chrome.com/docs/extensions/develop/security-privacy/user-privacy. [cit. 2024-04-09].

[5] *Stay secure* online. Google Developers, 2018. Dostupné z: https://developer.chrome.com/docs/extensions/develop/security-privacy/stay-secure. [cit. 2024-04-09].

[6] *About the WebExtensions API* online. Mozilla, 2019. Dostupné z: https://extensionworkshop.com/documentation/develop/about-the-webextensions-api/. [cit. 2024-04-08].

[7] *Testing persistent and restart features* online. Mozilla, 2019. Dostupné z: https://extensionworkshop.com/documentation/develop/testing-persistent-and-restart-features/. [cit. 2024-04-22].

[8] *Debugging* online. Mozilla, 2020. Dostupné z: https://extensionworkshop.com/documentation/develop/debugging/. [cit. 2024-04-22].

[9] *CSS Object Model (CSSOM)* online. The World Wide Web, 2021. Dostupné z: https://www.w3.org/TR/cssom-1/. [cit. 2024-03-15].

[10] *Getting started with web-ext* online. Mozilla, 2021. Dostupné z: https://extensionworkshop.com/documentation/develop/getting-started-with-web-ext/. [cit. 2024-04-22].

[11] *Distributing an add-on yourself* online. Mozilla, 2022. Dostupné z: https://extensionworkshop.com/documentation/publish/self-distribution/. [cit. 2024-04-09].

[12] *Hello World extension* online. Google Developers, 2022. Dostupné z: https://developer.chrome.com/docs/extensions/get-started/tutorial/hello-world. [cit. 2024-03-12].

[13] *Add-ons* online. MDN Web Docs, 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons. [cit. 2024-02-02].

[14] *Anatomy of an extension* online. MDN Web Docs, 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Anatomy_of_a_WebExtension. [cit. 2024-03-02].

[15] *Background scripts* online. MDN Web Docs, 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Background_scripts. [cit. 2024-04-08].

[16] *Bookmarks* online. MDN Web Docs, 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/bookmarks. [cit. 2024-04-14].

[17] *Content scripts* online. MDN Web Docs, 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Content_scripts. [cit. 2024-04-09].

[18] *CSSRule* online. MDN Web Docs, 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/CSSRule. [cit. 2024-03-15].

[19] *CSSStyleDeclaration* online. MDN Web Docs, 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/CSSStyleDeclaration#css_properties. [cit. 2024-03-15].

[20] *Introduction to the DOM* online. MDN Web Docs, 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction. [cit. 2024-03-12].

[21] *JavaScript APIs* online. MDN Web Docs, 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API. [cit. 2024-04-12].

[22] *Manifest.json* online. MDN Web Docs, 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/manifest.json. [cit. 2024-04-08].

[23] *Node* online. MDN Web Docs, 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Node. [cit. 2024-03-15].

[24] *Polyfill* online. MDN Web Docs, 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/Polyfill. [cit. 2024-05-02].

[25] *Popups* online. MDN Web Docs, 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/user_interface/Popups. [cit. 2024-04-08].

[26] *Storage* online. MDN Web Docs, 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/storage. [cit. 2024-04-14].

[27] *Tabs* online. MDN Web Docs, 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/tabs. [cit. 2024-04-14].

[28] *Your first extension* online. MDN Web Docs, 2023. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Your_first_WebExtension. [cit. 2024-03-12].

[29] *ADD-ONS* online. Mozilla, 2024. Dostupné z: https://addons.mozilla.org/cs/firefox/. [cit. 2024-04-22].

[30] *Browser extensions* online. MDN Web Docs, 2024. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions. [cit. 2024-02-02].

[31] *Build a cross-browser extension* online. MDN Web Docs, 2024. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Build_a_cross_browser_extension. [cit. 2024-02-02].

[32] *Cascade, specificity, and inheritance* online. MDN Web Docs, 2024. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Cascade_and_inheritance. [cit. 2024-03-15].

[33] *Cascade, specificity, and inheritance* online. MDN Web Docs, 2024. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Cascade_and_inheritance. [cit. 2024-03-15].

[34] *CSS Object Model (CSSOM)* online. MDN Web Docs, 2024. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/CSS_Object_Model. [cit. 2024-03-15].

[35] *CSS selectors* online. MDN Web Docs, 2024. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Selectors. [cit. 2024-03-15].

[36] *DOM* online. WHATWG, 2024. Dostupné z: https://dom.spec.whatwg.org/. [cit. 2024-04-6].

[37] *An Easy Guide to Build a Cross-Browser Extension* online. GrayCell Technologies, 2024. Dostupné z: https://www.graycelltech.com/an-easy-guide-to-build-a-cross-browser-extension/. [cit. 2024-02-02].

[38] *Element* online. MDN Web Docs, 2024. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Element. [cit. 2024-03-15].

[39] *Get started* online. Google Developers, 2024. Dostupné z: https://developer.chrome.com/docs/extensions/get-started/. [cit. 2024-04-08].

[40] *Notifications* online. MDN Web Docs, 2024. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/notifications. [cit. 2024-04-14].

[41] *Open the Inspector* online. Mozilla, 2024. Dostupné z: https://firefox-source-docs.mozilla.org/devtools-user/page_inspector/how_to/open_the_inspector/index.html. [cit. 2024-03-12].

[42] *WebRequest* online. MDN Web Docs, 2024. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/webRequest. [cit. 2024-04-14].

[43] *Welcome to Extensions!* online. Google Developers, 2024. Dostupné z: https://developer.chrome.com/docs/extensions. [cit. 2024-02-02].

[44] *What are extensions?* online. MDN Web Docs, 2024. Dostupné z: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/What_are_WebExtensions. [cit. 2024-02-02].

[45] *What Is a Mockup — The Final Layer of UI Design* online. UXPin, 2024. Dostupné z: https://www.uxpin.com/studio/blog/what-is-a-mockup-the-final-layer-of-ui-design/. [cit. 2024-04-10].

[46] *What is Puppeteer?* online. Google, 2024. Dostupné z: https://pptr.dev/guides/what-is-puppeteer. [cit. 2024-04-10].

[47] Burget, R. *FitLayout/2 - Web Page Analysis Framework* online. 2021. Dostupné z: https://github.com/FitLayout/FitLayout/wiki. [cit. 2024-04-08].

[48] Burget, R. *FitLayout - Puppeteer* online. 2022. Dostupné z: https://github.com/FitLayout/fitlayout-puppeteer. [cit. 2024-04-08].

[49] Burget, R. *FitLayout/2 Basic Concepts* online. 2022. Dostupné z: https://github.com/FitLayout/FitLayout/wiki/Basic-Concepts. [cit. 2024-05-03].

[50] Burget, R. *FitLayout/2 - Web Page Analysis Framework* online. 2024. Dostupné z: https://github.com/FitLayout/FitLayout. [cit. 2024-05-03].

[51] Carrizo, J. *Understanding DOM, CSSOM, Render Tree, Layout, and Painting.* online. 2020. Dostupné z: https://medium.com/weekly-webtips/understand-dom-cssom-render-tree-layout-and-painting-9f002f43d1aa. [cit. 2024-03-10].

[52] Gruenbaum, B. *Puppeteer, Selenium, Playwright, Cypress – how to choose?* online. 2020. Dostupné z: https://www.testim.io/blog/puppeteer-selenium-playwright-cypress-how-to-choose/. [cit. 2024-04-08].

[53] Hertzen, N. von. *Html2canvas* online. 2022. Dostupné z: https://html2canvas.hertzen.com/documentation.html. [cit. 2024-04-25].

[54] Johnson, B. *How CSS works: Understanding the cascade* online. 2018. Dostupné z: https://blog.logrocket.com/how-css-works-understanding-the-cascade-d181cd89a4d8/. [cit. 2024-03-15].

[55] Jůn Šimon. *Co je to wireframe, prototyp a mockup?* online. 2024. Dostupné z: https://www.simonjun.cz/blog/co-je-to-wireframe-prototyp-mockup. [cit. 2024-04-10].

[56] Kalambay, J. *Anatomy of a Chrome Extension* online. 2017. Dostupné z: https://medium.com/@jonnykalambay/anatomy-of-a-chrome-extension-54b9dd019825. [cit. 2024-03-03].

[57] Kantor, I. *Attributes and properties* online. 2022. Dostupné z: https://javascript.info/dom-attributes-and-properties. [cit. 2024-03-15].

[58] Leung, D. *JFont Checker* online. 2017. Dostupné z: https://github.com/derek1906/jFont-Checker. [cit. 2024-05-02].

[59] Milička, M. a Burget, R. Information Extraction from Web Sources based on Multi-aspect Content Analysis. *In: Semantic Web Evaluation Challenges, SemWebEval 2015 at ESWC 2015. Communications in Computer and Information Science*. Portorož: Springer International Publishing, 2015, sv. 2015, s. 81–92.

[60] Perera, R. *Wireframe vs Mockup: What's the Difference? (And When to Use Each)* online. 2023. Dostupné z: https://designshack.net/articles/graphics/wireframe-vs-mockup/. [cit. 2024-04-10].

[61] Tayar, G. *Comparing JavaScript Browser Automation Frameworks: Selenium Versus Webdriver.io Versus Puppeteer* online. 2018. Dostupné z: https://applitools.com/blog/comparing-javascript-browser-automation-frameworks/. [cit. 2024-04-08].

[62] Zhou, Z. *CSS Object Model (CSSOM)* online. 2022. Dostupné z: https://medium.com/@zhouzy/css-object-model-cssom-29d0a1951b5f. [cit. 2024-03-12].

Příloha A

Obsah přiloženého paměťového média

- **README.md** – Návod na přidání rozšíření do prohlížečů.
- **/Extension_BP/** – Zdrojové kódy rozšíření a soubory pro přidání do prohlížečů Chrome, Edge a Opera.
- **/Firefox/** – Složka se souborem rozšíření `12c1eb142ef547c6a029-0.1.xpi` pro prohlížeč Firefox.
- **/Latex/** – Zdrojové soubory LaTeX.
- **xnovot2a.pdf** – Soubor PDF bakalářské práce.