VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

ROZŠÍŘENÍ WEBOVÉHO PROHLÍŽEČE PRO ANALÝZU STRÁNEK

WEB BROWSER EXTENSION FOR PAGE ANALYSIS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **ROSTISLAV NAVRÁTIL**
AUTHOR

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

BRNO 2018

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Navrátil Rostislav**
Program: Informační technologie
Název: **Rozšíření webového prohlížeče pro analýzu stránek**
Web Browser Extension for Page Analysis
Kategorie: Web

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 jednotlivých prohlížečů.
3. 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 této analýzy.
4. Po dohodě s vedoucím zvolte vhodnou aplikační doménu, implementujte navržený rozšiřující modul a vzorové řešení serverové aplikace.
5. Proveďte testování vytvořeného řešení.
6. Zhodnoťte dosažené výsledky.

Literatura:

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

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3

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

Vedoucí práce: **Burget Radek, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 30. října 2018

Abstrakt

Cílem téhle práce je vytvořit rozšiřující modul webové prohlížeče se zaměřením na technologií WebExtensions. Rozšiřující modul umožní uživateli odeslat detaily o aktuálně zobrazené webové stránce a jejím textovém obsahu do serverové aplikace provádějící další analýzu a zobrazení výsledku. Rozšiřují modul je realizován pomocí technologie Extension APIs, ale je také podporován ve webových prohlížečích založených na WebExtensions. Komunikace mezi rozšiřujícím modulem a serverem je realizována pomocí XMLHttpRequest. Serverová aplikace je realizována v jazyce PHP.

Abstract

The purpose of this thesis is to create a WebExtensions oriented module for a web browser. An extension module allows the user to submit details about the currently displayed web page and text content to a server application for thorough analysis and displaying the result. It is built on ExtensionAPIs, but is also supported in WebExtensions based web browsers. The communication between the extension module and the server is realized by XMLHttpRequest and the server application itself is implemented in PHP.

Klíčová slova

WebExtensions, Extensions APIs, webové technologie, rozšíření, javascript, analýza webových stránek, textová analýza, webový prohlížeč

Keywords

WebExtensions, Extensions APIs, web technology, extension, javascript, website analysis, Web browser

Citace

NAVRÁTIL, Rostislav. *Rozšíření webového prohlížeče pro analýzu stránek*. Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Burget, Ph.D.

Rozšíření webového prohlížeče pro analýzu stránek

Prohlášení

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

.....
Rostislav Navrátil
28. dubna 2019

Poděkování

Rád bych poděkoval panu Ing. Radkovi Burgetovi, Ph.D za odborné vedení, cenné rady a svůj čas při tvorbě této práce.

Obsah

Kapitola 1

Úvod

Můžeme sami sobě, nebo i lidem v našem okolí položit jednoduchou otázku. Co se nám vybaví pod pojmem Webový prohlížeč? S největší pravděpodobností uslyšíme odpovědi typu „Chrome“, „FireFox“, „Safari“, „Opera“ a tak podobně. Uslyšíme názvy programů, které dennodenně používáme pro přístup k Word Wide Web (WWW), neboli k internetu. Cíle kvůli kterým lidé používají internet jsou různé. Může to být kvůli vyhledávání zábavy, nebo kvůli důležitým informacím. Návštěvnost internetu bezesporu stoupá. Troufám říct, že o internetu můžeme mluvit jako o základní potřebě 21. století. Je to něco co lidé jen tak nepřestanou používat a jak už je v lidské povaze dáno, lidé chtějí čím dál víc a víc. To platí jak u internetu, tak i u webových prohlížečů. Webové prohlížeče jsou jakou si vstupní branou do internetu. Prostředníkem mezi počítačem a internetem. S postupujícím časem se vyvíjejí a přizpůsobují potřebám, které na ně lidé kladou. Nejen že dokážou zobrazit požadovanou webovou stránku, uložit ji do záložek, prohlédnout si její zdrojový kód, dokonce je možné je obohatit o další programy, aplikace, či rozšíření. Tahle práce je zaměřena právě na tyhle rozšíření. Jsou to malé programy, které dokážou webový prohlížeč obohatit o další funkce. Některé rozšíření se dokonce stali už jakým si standardem, bez kterého si mnoho z nás nedokáže představit, používání webového prohlížeče. Například rozšíření, které skryje otravné reklamy, kterými je zahlcena každá druhá webová stránka. Rozšíření se pomalu stávají nedílnou součástí webových prohlížečů a je v nich veliký potenciál. To je jeden z hlavních důvodů proč jsem si vybral tohle zadání pro svou bakalářskou práci. V dnešní době existují dva hlavní systémy, díky kterým se dají vytvářet rozšíření. Jedním je Extension APIs a jeho hlavním propagátorem je prohlížeč Google Chrome. Druhým systémem je WebExtensions, který propaguje prohlížeč Mozilla FireFox.

Cílem téhle bakalářské práce je navrhnout a implementovat rozšíření prohlížeče, podporované jak v prohlížeči Google Chrome, tak i v Mozilla Firefox. Tohle rozšíření umožní uživateli odeslat detaily o aktuálně zobrazené webové stránce a jejím obsahu do serverové aplikace k provedení další analýzy a zobrazení výsledku na nové kartě webového prohlížeče.

Práce je rozdělena do několika kapitol, které čtenáře nejprve seznámí s webovými technologiemi a technologiemi s nimi spojenými tak, aby čtenář poté neměl problém s pochopením samotné implementace téhle bakalářské práce.

Jelikož hlavním tématem téhle bakalářské práce jsou webové technologie, je nezbytné pro snadné pochopení, aby se čtenář seznámil s programovacím jazykem, ve kterém se webové technologie programují. Proto se kapitola 2 zabývá právě JavaScriptem. Kapitola 3 se zabývá reprezentací webové stránky. Následně je čtenář zcela připravený na kapitolu 4, která je zaměřena na samotné rozšíření webových prohlížečů. Jsou zde podrobně vysvětleny systémy

WebExtensions a Extension APIs. Kapitola 5 je věnována návrhu a implementaci bakalářské práce. Jsou zde popsaný a vysvětleny způsoby jakými je rozšíření a server implementován.

Kapitola 2

javaScript

Tahle kapitola bakalářské práce si klade za cíl okrajově seznámit čtenáře s programovacím jazykem JavaScript. Jedná se o jazyk, ve kterém se programují rozšíření pro aplikační rozhraní WebExtensions a Extension APIs. Tudíž je vhodné, aby čtenář téhle práce měl přehled o programovacím jazyce webových technologií, které se probírají v dalších kapitolách.

Tahle kapitola je rozdělena na čtyři podkapitoly. Kapitola 2.1 se zabývá nezbytnými základy jazyka. V podkapitole 2.2 jsou vysvětleny zajímavosti, které mohou JavaScript odlišit od jiných jazyků. Podkapitola 2.3 je zaměřena na způsoby, jakými lze programovací jazyk JavaScript obohatit o externí moduly. A v poslední podkapitole 2.4 jsou vysvětleny způsoby, jakými lze JavaScript integrovat do HTML souboru.

2.1 Definice

JavaScript je interpretovaný, dynamický, programovací jazyk, který obohacuje funkčnost webové stránky. Stará se o dynamické zobrazení obsahu. V praxi se to může projevit například vložením interaktivní mapy, měnícím se obsahem bez aktualizování webové stránky, 2D a 3D grafikou. Pomocí JavaScriptu lze také získat přístup k různým aplikačním rozhraním (dále jen „API“).[59]

- **Document Object Model** - Umožňuje snadněji manipulovat a upravovat HTML a CSS webových stránek. Představuje dokument s logickým stromem, který se skládá z uzlů. Každý uzel reprezentuje HTML objekt. K uzlům se dá přistupovat a upravovat je. Tím lze změnit celou strukturu, styl a nebo obsah dokumentu. [59][58]
- **Google maps API** - Aplikační rozhraní, vytvořené společností Google, díky kterému lze snadno vkládat vybrané mapy do webových stránek[59].
- **Canvas, WebGL** - Umožňuje vytvářet 2D a 3D grafiku[59].

Standardem pro jazyk JavaScript je ECMAScript. ECMAScript je spravován organizací ECMA International. Od roku 2012 je plně podporovanou verzí ECMAScript 5.1. Novější verze jsou ve formě draftu. Tou nejnovější je desátá verze, které nese název ECMAScript 2020.[2][7]

2.2 Zajímavosti

Proměnné

Existují tři druhy proměnných. Mají odlišné vlastnosti a při deklaraci se liší i jejich klíčové slova.[57][56]

- **var** - Přístup k proměnné je možný z jakéhokoliv místa v kódu.
- **let** - K proměnné je možné přistupovat pouze v bloku kódu, kde proběhla její deklarace.
- **const** - Přístup k proměnné bude stejný jako u `let`. Při deklaraci se musí uvést i hodnota, která se dále nedá měnit.

Porovnávání hodnot

Existují dva druhy porovnání rovnosti hodnot. Klasické při kterém se používá znak `==`. A striktní u kterého se používá znak `===`. Výsledkem obou druhů porovnání je vždy hodnota typu boolean. Rozdíl je ve způsobu porovnání. U klasického porovnávání se porovnávají jen hodnoty a u striktního i typy.[8]

```
22 == '22' // true
22 === '22' // false
```

Výpis 2.1: Rozdíl mezi striktním a klasickým porovnání rovnosti. (Zdroj: [39])

Při klasickém porovnání rovnosti mezi číslem a textovým řetězcem se nejprve hodnota textového řetězce převede na číselnou hodnotu a poté dojde k porovnání[8].

Datový typ boolean sestává z hodnot `true` a `false`. Hodnota `false` může být v jazyce JavaScript reprezentována šesti způsoby: `false`, `null`, `undefined`, `NaN`, nula a prázdný textový řetězec. V případě porovnání rovnosti klasickou metodou pomocí znaménka `==`, lze hodnoty značící nepravdu rozdělit do dvou skupin. Následující dvě odrážky značí dané dvě skupiny:

- **false, 0, ""** - Booleovský zápor, číslo nula, prázdný textový řetězec.
- **null, undefined** - nepřítomnost jakékoliv hodnoty, nedefinovaná hodnota

Pokud dojde k porovnání rovnosti hodnot ze stejné skupiny, výsledek bude vždy `true`. Při porovnání hodnot z odlišných skupin bude výsledkem `false`. Jedinou výjimkou je hodnota `NaN` (Not-A-Number), což v překladu znamená „Není číslo“. Kdykoliv se objeví v porovnávání rovnosti, výsledkem bude vždy hodnota `false`. Kladná i záporná nula jsou si rovny[8].[39]

```
false == 0 // true
false == "" // true
0 == "" // true
null == undefined // true
Nan == NaN // false
Nan == false // false
Nan == null // false
```

Výpis 2.2: Příklady porovnání rovnosti hodnot značící nepravdu. (Zdroj: [39])

V případě, kdy je při klasickém porovnání rovnosti jednou z hodnot objekt, dojde nejprve k jeho převodu na primitivní hodnotu a poté až k porovnání rovnosti. Primitivní hodnoty jsou: `undefined`, `null`, booleovské hodnoty, číselné hodnoty a textové hodnoty.[8]

Funkce

Funkce je datový typ a díky tomu je možné ji přiřadit do proměnné a předávat jako parametr [61]. Funkce definovaná v přiřazení do proměnné se nazývá anonymní funkce[60].

```
let abc = function() {
    console.log("VUT");
}

setTimeout(abc,3000);
```

Výpis 2.3: Do proměnné abc je přidána funkce, která vypíše do konzole text. Proměnná abc je následně předána jako parametr do metody setTimeout, která zpozdí vykonání funkce o tři vteřiny. (Zdroj: [61])

Ve funkci je možné vytvořit další funkci a tímhle způsobem do sebe funkce zanořovat. Proměnné z vnější funkce, jsou k dispozici i ve vnitřní funkci.

```
function first(a){
    function second(b){
        return a+b;
    }
    return second;
}

var f = first(2);
f(5);
```

Výpis 2.4: Parametr a má číselnou hodnotu dva a parametr druhé funkce b má číselnou hodotu pět. (Zdroj: [61])

Primitivní hodnoty jsou: `undefined`, `null`, booleovské hodnoty, číselné hodnoty a textové hodnoty. Pokud je použijeme jako parametr funkce, bude jejich hodnota zkopírována a uvnitř funkce se bude pracovat s její kopií. Tudíž jsou předány hodnotou. Složitější datové typy jsou předány odkazem. To znamená, že pokud dojde ke změně parametru uvnitř funkce, změna se projeví i mimo danou funkci. Složitějšími datovými typy rozumíme pole, funkce a objekty.[61]

2.3 Moduly

Vanilla JavaScript, neboli čistý JavaScript je typ JavaScriptu, který není závislí na externích modulech. Jedinou vyjímkou může být použití polyfillů. Způsobů jak obohatit JavaScript o další funkce, je hned několik. [61]

- **knihovna** - Sbírka znovupoužitelných částí zdrojového kódu. Knihovny mají většinou jedno konkrétní zaměření. Například knihovna řešící konkrétní fyzikální simulace. Velmi populární knihovnou je jQuery, která mění způsob javascirptového zápisu.

- **polyfilly** - Jedná se o moduly zpravidla menší velikosti, které mají za cíl obohatit webový prohlížeč o standardizovanou funkci, která v něm chybí.
- **framework** - Framework, neboli aplikační rámec. Modul převážně větší velikosti, který si programátor upravuje a nastavuje k řešení konkrétních problémů. Většinou se skládá z velkého mnnožství souborů a obsahuje složitější rozhraní. Rozdíl mezi frameworkem a knihovnou není zcela jednoznačný.
- **preprocesor** - Nástroj který před spuštěním transformuje nepodporovanou verzi JavaScriptu, nebo úplně jiný programovací jazyk, do podporované verze JavaScriptu. Díky tomu si může programátor pro implementaci vybrat programovací jazyk, který mu více vyhovuje.

2.4 Použití

Existuje několik způsobů jak začlenit JavaScript do webové stránky. Zvolení toho správného způsobu závisí na rozsáhlosti projektu a osobních preferencích každého programátora. Formou externího souboru je to v mnoha případech nejpřehlednější, protože je kód webové stránky oddělen souborem od kódu JavaScriptu. Ovšem v menších projektech může být tohle řešení zbytečné a je vhodnější použít prostší způsob začlenění JavaScriptu do webové stránky.

- **HTML tag <script>** - Vymezuje prostor pro JavaScriptový kód. V případě že je ve webovém prohlížeči vypnut JavaScript, nebo z nějakých jiných důvodů nejede, tak HTML tag `<noscript>` vypíše oznámení uživateli o nefunkčnosti JavaScriptu.[55]

```
<!DOCTYPE html>
<html>
<body>
    <script>
        document.getElementById("el").innerHTML = "VUT";
    </script>
    <noscript>JavaScript Error</noscript>
</body>
</html>
```

Výpis 2.5: Vložení JavaScriptu do HTML kódu pomocí speciálního tagu. (Zdroj: [55])

- **Inline** - Zapisuje se jako atribut do již existujícího HTML tagu. Jedná se o prosté řešení. Při složitějším projektu může být značně nepřehledné.

```
<a href="#" onmouseover="alert('FIT')">click here</a>
```

Výpis 2.6: Inline zápis. (Zdroj: [25])

- **Externí soubor** - Kód JavaScriptu se nachází v jiném souboru než je kód HTML webové stránky.

```
<script src="j.js"></script>
```

Výpis 2.7: Importování souboru s JavaScriptovým kódem. (Zdroj: [61])

Kapitola 3

Reprezentace webové stránky

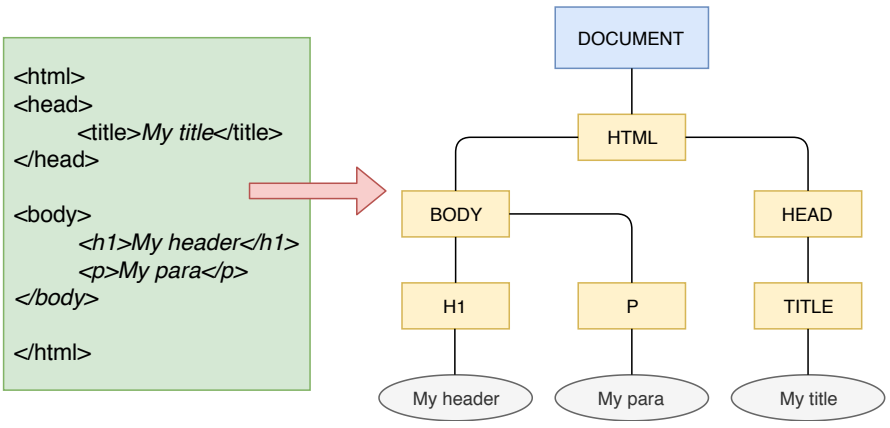
Kapitola pojednává o detailech reprezentace zobrazené webové stránky v prohlížeči. Dále se zaměřuje na aplikačním rozhraní, díky kterému lze s těmito detaily manipulovat. Kapitola je rozdělena na dvě podkapitoly. V podkapitola 3.1 se zabývá teorií a podkapitola 3.2 praktickým využitím.

3.1 DOM (Document Object Model)

Existuje několik možností zobrazení webové stránky. Zobrazení pomocí webového prohlížeče, který interpretuje HTML kód, nebo samotným HTML kódem. A poslední zde zmíněnou a pro tuhle kapitolu nejdůležitější možností je reprezentace pomocí objektového modelu dokumentu (dále jen „DOM“). [43]

Struktura

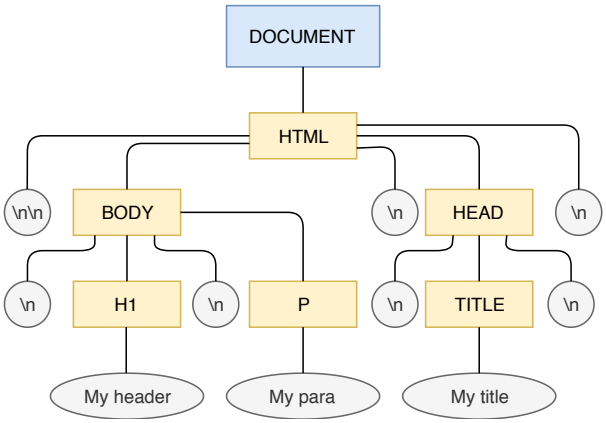
DOM je reprezentace dat objektů, které obsahují strukturu a obsah dokumentů. Webová stránka je v DOM reprezentována stromovou strukturou. Skládá se z uzlů, které reprezentují HTML objekty webové stránky. [40]



Obrázek 3.1: Zobrazuje reprezentaci webové stránky. Na levé straně v jazyce HTML a na pravé straně pomocí DOM. Lze vidět, že jednotlivé HTML elementy jsou v DOM zobrazené ve stromové struktuře, kde uzly stromu znázorňují HTML elementy.

Bílé znaky

Jeden z problémů, který se v DOM vyskytuje jsou bílé znaky. Například znak pro zalomení řádku `\n`. Tenhle problém může ztížit manipulaci se strukturou DOM.[5]



Obrázek 3.2: Zobrazení výskytu bílých znaků ve struktuře DOM.

3.2 DOM a JavaScript

K manipulaci s DOM za účelem upravení obsahu, nebo vizuální podoby webové stránky, je potřeba jazyk, který je podporovaný webovým prohlížečem. Jedním s takových jazyků je JavaScript, nicméně DOM je navržen tak, aby nebyl závislí na žádným jazyku. Díky JavaScriptu je možné snadno upravovat vzhled i obsah. [43]

Základní příklady jak je možné použít DOM pomocí JavaScriptu. Jak lze vidět k dokumentu webové stránky se přistupuje pomocí objektu `document`.

- **document.body** - vrátí HTML element `body`.
- **document.body.childones** - vrací všechny přímé potomky HTML elementu `body`.
- **document.body.style** - vrací všechny CSS vlastnosti HTML elementu `body`.

Kapitola 4

Rozšíření webových prohlížečů

Kapitola se zaměřuje na aplikační rozhraní (dále jen „API“), které jsou určeny k vývoji rozšíření pro webové prohlížeče. Pojem rozšíření je vysvětlen v podkapitole 4.1. V současné době existují dvě hlavní API pro rozšíření webových prohlížečů. Prvním je WebExtensions, které podporují prohlížeče Mozilla Firefox (dále jen „FireFox“) a Microsoft Edge (dále jen „Edge“). Druhým je Extension APIs, které podporují prohlížeče Google Chrome (dále jen „Chrome“) a Opera. Rozšíření psané pro tyhle API jsou si velice podobné a liší se jen minimálně. Rozdíly jsou podrobně popsané v podkapitole 4.3. Struktura rozšíření je podrobně rozebrána v podkapitole 4.2, která je rozdělena do několika sekcí, které se zaměřují na jednotlivé části rozšíření, kde je podrobně vysvětleno k čemu slouží a jaké mají možnosti. Podkapitola 4.4 se zabývá distribucí již hotových rozšíření. Podkapitola 4.5 je zaměřena na komunikaci skriptů v rámci jednoho rozšíření. Oproti tomu podkapitola 4.6 se zabývá komunikací mezi rozšířením a serverem, tudíž komunikací po síti. V podkapitole 4.7 je rozebrán úložný prostor webového prohlížeče, který může rozšíření využívat. V poslední podkapitole 4.8 je vysvětlena bezpečnost CSP.

4.1 Pojem rozšíření

Rozšíření jsou malé programy v prohlížeči, které podle potřeb uživatelů přizpůsobují a obohacují chování a funkce webového prohlížeče. Jsou postaveny na známých webových technologiích jako jsou HTML, CSS a JavaScript. Rozšíření lze využít k jednoduchých úkonům, jako je například indikátor příchozí emailové pošty, až po složitější úkony, které umožňují měnit, nebo i nahrazovat obsah webové stránky.[22]

4.2 Anatomie rozšíření

I přes to, že jsou mezi rozšířeními psané pro Extension APIs a WebExtensions rozdíly, jejich anatomie je stejná, až na malé drobnosti. Anatomií zde rozumíme seskupení vzájemně komunikujících souborů, které jsou nedílnou součástí pro správnou funkčnost a podobnu rozšíření.

manifest.json

Nejdůležitějším souborem celého rozšíření je soubor s názvem manifest, který obsahuje základní metadata o rozšíření. Je napsán v upraveném formátu JSON. Oproti klasickému

JSON formátu může obsahovat i komentáře. Komentář začíná znakem „//“ a končí koncem řádku. Je to jediný povinný soubor u rozšíření.[30]

Manifest soubor se skládá z klíče a hodnot. Klíče lze rozdělit do následujících skupin:

- **Povinné klíče** - musí být součástí každého manifest souboru. Klíč „name“ obsahující název rozšíření. Dalším je klíč „version“, který obsahuje verzi daného rozšíření a posledním klíčem je „manifest_version“, který určuje verzi manifest souboru. Aktuální verze manifest souboru nese číslo dva. Druhá verze přinesla několik změn, mezi které patří změny názvů klíčů a změna výchozího nastavení u Content Security Policy (CSP) 4.8.[18]
- **Doporučené klíče** - nemusí být součástí manifest souboru, ale doporučuje se je použít pro zkvalitnění rozšíření. Prvním je klíč „description“, který obsahuje základní informace o daném rozšíření. Druhým je klíč „icons“, jehož hodnota obsahuje relativní cestu k ikoně, která se zobrazuje na stránce zabývající se správou rozšíření. U prohlížeče FireFox má tahle stránka URL „about:debugging“ a u Chrome je to „chrome://extensions“. Ideální rozměr ikony je 48x48 px, avšak ikonu lze vložit ve více rozměrech a webový prohlížeč si sám určí, která z velikostí je vhodná pro zobrazení.[24][17]
- **Volitelné klíče** - používáme podle požadavků kladených na rozšíření, protože ovlivňují funkčnost webového prohlížeče a rozšíření. Volitelných klíčů je mnoho a mohou se lišit podle druhu webového prohlížeče a jeho verze. Nejnezbytnějším volitelným klíčem je „permissions“. Je podporovány v aktuálních verzí prohlížečů Chrome, Firefox, Opera, Edge a FireFox for Android. Jak už anglický název napovídá, jedná se o klíč, který zpřístupňuje určitou funkčnost webového prohlížeče, tak aby ji mohlo rozšíření používat. Zanořené klíče v „permissions“ se liší podle druhu prohlížeče a jeho verze.[15][32]

```
"permissions": [
  "notifications" // Dovoluje pouzit notifikace.
],
"content_scripts": [{ // Skript spusteny s~webovou strankou.
    "matches": ["<all_urls>"],
    "js": ["content.js"]
  }],
"background":{ // Skript bezici v~pozadi.
  "scripts": ["background/background.js"]
},
"browser_action": { // Popup v~panelu nastroju.
  "default_icon": "icons/logo.png",
  "default_title": "My extension",
  "default_popup": "popup.html"
},
"options_ui": { // Sprava rozsireni.
  "open_in_tab": true,
  "page": "options.html"
}
```

Výpis 4.1: Ukázka často používaných volitelných klíčů. (Zdroj:[30])

Background

Background nejčastěji obsahuje pouze JavaScriptové soubory, které jsou v manifest souboru přiřazeny pod klíčem „background“. Tyhle soubory jsou v rozšíření výjimečné tím, že běží v pozadí prohlížeče. Jsou nezávislé na době běhu konkrétních webových stránek, nebo okna prohlížeče. Spouští se při načtení rozšíření a běží do té doby, dokud nedojde k jejich zakázání, nebo odinstalování daného rozšíření. Jsou omezené pomocí CSP 4.8, vůči potencionálnímu nebezpečí. Background skripty nemají přístup k obsahu webových stránek, ale mají možnost plně využít rozhraní, které jim WebExtensions, či Extension APIs nabízí.

Background může kromě skriptů obsahovat i HTML stránku. Jejím účelem může být například propojení více background skriptů dohromady. [11][26][16]

Content

Content v rámci rozšíření obsahuje JavaScriptový soubor (dále jen „Content skript“). Oproti ostatním souborům v rozšíření se liší tím, že má přímý přístup ke kontextu webové stránky. Tudíž má možnost upravovat její obsah, nebo přistupovat k objektovému modelu dokumentu (dále jen „DOM“). Content skript má omezený přístup k rozhraní WebExtensions, či Extension APIs, proto je v některých případech závislí na Background skriptech. Existují dva způsoby jak Content skript spustit.[14][27]

- *Deklarativně* - V manifest souboru je u klíče „conten_script“ vložen podklíč „matches“, jehož hodnota je URL vzor. Ke spuštění Content skriptu dojde v případě, kdy se URL adresa nově načítané stránky shoduje s URL vzorem klíče „matches“.
- *Programově* - Užitím příkazu `tabs.executeScript()`. Při spouštění Content skriptu tímto způsobem je nutné mít povolenou hodnotu „ActiveTab“ v manifest souboru u klíče „permissions“.

Jak už bylo zmíněno Content skript má přístup k DOM. Může provádět změny jako každý jiný skript spuštěný danou webovou stránku až na dvě výjimky.[14][27]

- Content skript nevidí JavaScriptové proměnné definované danou webovou stránkou.
- Pokud jiný skript spuštěný danou webovou stránkou redefinuje DOM, tak Content skript bude mít pořád přístup jen k původní verzi DOM.

Popup

Popup, neboli dialogové okno se skládá z HTML a CSS souborů, které určují jeho vizuální podobu. Dále z JavaScriptového souboru, který se stará o funkčnost dialogového okna. Dialogové okno se vyvolá stisknutím tlačítka rozšíření, které se nachází v panelu nástrojů (toolbar), nebo v adresovém řádku (adress bar). Druhou možností je pomocí klávesových zkratek uvedených v manifest souboru. Zavření dialogového okna lze uskutečnit kliknutím na libovolné místo mimo dialogové okno, nebo pomocí příkazu „window.close()“ v JavaScriptovém souboru dialogového okna. Dialogové okno v rozšíření většinou slouží jako prostředník mezi uživatelem a rozšířením. Uživatel si zde nastavuje, spouští, nebo jinak obsluhuje rozšíření. JavaScriptový soubor zde může využívat všech možností, které mu nabízí rozhraní WebExtensions, nebo Extension APIs. Jediným omezením je CSP 4.8. [12]

Options page

Skládá se z HTML, CSS a JavaScriptových souborů. Používá se pro správu rozšíření. Uživatel si zde může nastavit funkčnost daného rozšíření. Existují dva způsoby jak otevřít Options page. První způsob je pomocí příkazu `runtime.openOptionsPage()`. A druhý způsob je přes správu rozšíření ve webovém prohlížeči. Existují i dva způsoby zobrazení Options page. Buď přímo ve správě rozšíření ve webovém prohlížeči, nebo v nové kartě prohlížeče. Pro druhý způsob zobrazení je nutné mít v manifest souboru u klíče „options_ui"další podklíč „open_in_tab"s hodnotou „true". Některé prohlížeče klíč „options_ui", nepodporují a místo něj používají jeho předchůdce klíč „options_page". Mimo jiné Options page podléhá omezením CSP 4.8. [49][44][50]

	FireFox	**Chrome**	**Edge**	**Opera**	**FireFox Android**
options_page	NE	ANO	14	15	NE
options_ui	48	40	NE	15	57
open_in_tab	48	40	NE	NE	57

Tabulka 4.1: Znázorňuje webové prohlížeče a jejich podporu pro dané klíče. Pokud webový prohlížeč klíč podporuje, tak je v tabulce označen slovem „ANO", nebo verzí od jaké je klíč podporován. V opačném případě je označen slovem „NE". (Zdroj: [44][49])

Zdroje dostupné z webu

Jedná se o externí zdroje, u kterých se očekává, že budou použitelné v kontextu webové stránky. Můžou to být obrázky, HTML, CSS, nebo JavaScriptové soubory, které lze vložit do webové stránky. Pro použití externího zdroje je nutné v manifest souboru u klíče „web_accessible_resources“ zadat hodnotu, která je relativní cestou k externímu zdroji.[33][19]

Stránka rozšíření

Stránkou rozšíření se myslí stránka, která je plně vytvořena rozšířením. Pomocí rozšíření lze vytvořit novou kartu v prohlížeči, nebo i celé nové okno prohlížeče.[51][11]

```
var createData = {
  type: "detached_panel",
  url: "panel.html",
  width: 250,
  height: 100
};
```

Výpis 4.2: Vytvoření nového okna webového prohlížeče již s připtaveným HTML obsahem. Okno bude mít velikost 250x100 px. (Zdroj:[51])

4.3 Rozdíly mezi Extension APIs a WebExtensions

Rozšíření psané pro rozhraní Extension APIs a WebExtensions jsou si na pár drobností velice podobné, ale jsou mezi nimi i významné rozdíly. Prohlížeče které jsou založené na technologií WebExtensions do značné míry podporují i rozšíření psané pro Extension APIs. A prohlížeče založené na technologií Extension APIs už bohužel nejsou tak kompatibilní

s rozšíření psané pro WebExtensions. Jednoduše řečeno, pokud programátor chce, aby jeho rozšíření bylo kompatibilní v co nejvíce prohlížečích je výhodnější ho psát pro Extension APIs, protože s minimálními změnami pojede i na prohlížečích založených na WebExtensions.

	Chrome	**Firefox**	**Opera**	**Edge**
Svět	71.58%	8.72%	2.41%	4.34%
Česká Republika	61.75%	15.65%	4.99%	5.02%
Slovenská Republika	65.45%	15.96%	5.93%	3.29%

Tabulka 4.2: Nejpoužívanější webové prohlížeče 1.2.2018 - 1.2.2019. Webový prohlížeč Google Chrome vede na plné čáře jak v České i Slovenské republice, tak i v celém světě. Po té následuje webový prohlížeč Mozilla FireFox, který je ve světě skoro o polovinu méně používanější než v Česku a na Slovensku. Prohlížeče Opera a Microsoft Edge tvoří nepatrnou část. (Zdroj: [45])

Při vývoji rozšíření, které má být kompatibilní s vícero webovými prohlížeči, je nutné vzít v potaz následující odrážky. Odrážky značí významné rozdíly mezi webovými technologiemi a webovými prohlížeči. [34]

- **JavaScript** - Podpora JavaScriptu se v různých webových prohlížečích může lišit. Tím pádem některé JavaScriptové implementace v rozšíření nemusí být kompatibilní v jiném prohlížeči.
- **Manifest klíče** - Struktura manifest souboru je totožná v prohlížečích podporující WebExtensions i Extension APIs. Ovšem o klíčích v manifest souboru to říci nelze. Každý prohlížeč může podporovat různé klíče.
- **Jmenný prostor** - Webové prohlížeče založené na technologií WebExtensions používají jako základní jmenný prostor slovo „browser“. Oproti tomu webové prohlížeče založené na technologií Extension APIs používají jako základní jmenný prostor slovo „chrome“.
- **Promises/callbacks** - Webové prohlížeče založené na technologií WebExtensions používají „promises“ pro asynchronní API. Kdežto webové prohlížeče založené na technlogií Extension APIs používají „callbacks“.

Vzájemná kompatibilita

Rozšíření psané pro Extension APIs bude s minimálními změnami podporováno i v prohlížečích založených na technologií WebEtensions. WebExtensions si poradí i s odlišným jmenným prostorem, který u Extension APIs má označení „chrome“. Lze to i opačně s použitím speciální polyfilly, která se jmenuje „WebExtension browser API Polyfill“. Tohle řešení umožní, aby webové prohlížeče založené na technologií Extension APIs podporovaly slovo „browser“ jako svůj jmenný prostor.

V každém JavaScriptovém souboru, kde je zmíněn jmenný prostor „browser“, musí dojít k jeho definování. Aby to bylo možné, je nutné na každém takovém místě načíst samotné polyfilly před daným JavaScriptovým souborem.

Nekompatibilní implementace, jako jsou promises, polyfilly neřeší a je nutné je přepsat do jiné formy.[23]

4.4 Distribuce

Tahle podkapitola se zabývá distribucí rozšíření v obchodu Google a v Mozilla Add-ons (dále už jen „AMO“). Obě dvě zmíněné služby jsou internetové obchody, kde je možné stahovat i publikovat rozšíření. Podmínky pro stahovaní jsou prosté. Stačí mít požadovaný webový prohlížeč. Pro distribuci rozšíření je to o něco složitější.[29]

Mozilla FireFox

Předtím než je možné rozšíření zveřejnit, je nutné získat tzv. podpis. Rozšíření které získá podpis je podepsané a může být stahováno koncovými uživateli. Nepodepsané rozšíření mohou instalovat jen uživatelé s vývojářskou verzí webového prohlížeče. Na klasických verzích webových prohlížečů lze rozšíření dočasně spustit také na speciální stránce, která je určená pro správu rozšíření. Nachází se na URL adrese „about:config“. Životnost dočasně načtených rozšíření trvá do zavření okna prohlížeče. Mimo jiné rozšíření nahrané v AMO lze zveřejnit pro všechny uživatele a nebo jen pro určitou privátní skupinu. [29][35]

Postup:

- **Nahrání rozšíření** - Prvním krokem je přihlášení se k AMO. Poté je možné rozšíření nahrát na server.
- **Žádost o podpis** - Získáním podpisu je možné rozšíření distribuovat v obchodě AMO. Ovšem není to podmínkou a vývojář si může distribuovat své rozšíření i jinou cestou.

Rozšíření musí splňovat stanovené podmínky. Mozilla může kdykoliv otestovat, nebo monitorovat jakékoliv rozšíření. Zahrnutí rozšíření do AMO, nebo vypsání certifikátu záleží čistě na úsudku společnosti Mozilla. U každého rozšíření dojde k automatické kontrole a případně může dojít i ke kontrole manuální.. [29][35]

Google Chrome

Obdobný postup distribuce rozšíření je u webových prohlížečů Google Chrome. Je zde také možnost spustit rozšíření dočasně a to přes speciální stránku pro správu rozšíření na URL adrese „chrome://extensions/“. Životnost dočasně načtených rozšíření přetrvává i po znovu otevření okna prohlížeče. [21]

- **Nahrání rozšíření** - Pro nahrání rozšíření na Google server, je nutné mít založený vývojářský účet a rozšíření musí být archivováno ve formátu zip.
- **Publikace rozšíření** - Rozšíření je nyní nahrané na Google serveru. Je zde možnost přizpůsobit vzhled stránky v Google obchodu, kde bude rozšíření nabízeno.
- **Zaplacení poplatku** - Před publikováním prvního rozšíření, je nutné zaplatit jednorázové členský poplatek ve výši 5 USD.
- **Zveřejnění aplikace** - Po zveřejnění rozšíření obdrží jeho majitel tzv. app ID, pod kterém je rozšíření uloženo v Google obchodu.

4.5 Komunikace v rámci rozšíření

Existují dvě metody jak umožnit komunikaci mezi JavaScriptovými soubory v rámci jednoho rozšíření. Jednou možností je komunikace pomocí jednorázových zpráv s volitelnou odpovědí. Druhou možností je vytvoření delšího spojení mezi oběma stranami a využít ho ke vzájemné komunikaci. Zprávy jsou odesílané ve formátu JSON. [20][27]

Jednorázové zprávy

Jednorázové zprávy jsou nejprostším způsobem jakým lze komunikovat mezi skripty v rámci jednoho rozšíření. U webových prohlížečů založených na technologii Extension APIs se přistupuje k API pod názvem „chrome“. U WebExtensions pod názvem „browser“.

	Content skript	**Ostatní skripty**
Odesílání zpráv	chrome.runtime.sendMessage()	chrome.tabs.sendMessage()
Přijímání zpráv	chrome.runtime.onMessage	chrome.runtime.onMessage

Tabulka 4.3: Znázorňuje způsob jakým komunikuje Content script s ostatními skripty v rámci rozšíření. (Zdroj: [27])

Výpisy 4.3 a 4.4 znázorňují komunikaci pomocí jednorázových zpráv. Ve výpisu 4.3 Content script odešlě zprávu. Ve výpisu 4.4 je znázorněn příjemce zprávy. Může se jednat například o Background script, či Popup script. Příjemce zprávu zpracuje a odešle odesílateli odpověď.

```
chrome.runtime.sendMessage({greeting: "hello"}, function(response) {
  console.log(response.farewell);
});
```

Výpis 4.3: Odesílání jednorázové zprávy z Content scriptu. Obsahem zprávy je textový řetězec „hello“. Je zde připravena funkce, která v konzolovém logu zobrazí odpověď od příjemce zprávy. (Zdroj: [20])

```
chrome.runtime.onMessage.addListener(
  function(request, sender, sendResponse) {
    console.log(sender.tab ?
                "from a content script:" + sender.tab.url :
                "from the extension");
    if (request.greeting == "hello")
      sendResponse({farewell: "goodbye"});
  });
```

Výpis 4.4: Znázorňuje příjem jednorázové zprávy. Do konzolového logu se vypíše URL adresa karty webového prohlížeče, z které Content script odeslal zprávu. Následuje odeslání odpovědi odesílateli zprávy. (Zdroj: [20])

V opačné situaci, kde je Content script příjemcem, je potřeba změnit způsob odesílání zpráv. Musí se určit, které kartě webového prohlížeče má být zpráva odeslána. Příjem zprávy a následná odpověď odesílateli bude stejná jako ve v předchozím výpisu 4.4.

```
chrome.tabs.query({active: true, currentWindow: true}, function(tabs) {
    chrome.tabs.sendMessage(tabs[0].id,
        {greeting: "hello"},
        function(response) {
            console.log(response.farewell);
        }
    );
});
```

Výpis 4.5: Znázornění odeslání jednorázové zprávy, kde příjemcem je Content script spuštěný na aktuálně zobrazené kartě webového prohlížeče.(Zdroj: [20])

Dlouhodobé spojení

V situacích kdy se komunikace skládá z více jak jednoho požadavku a odpovědi, je moudré zvážit zda není vhodnějším způsobem pro komunikaci, použití dlouhodobého spojení. Při dlouhodobém spojení se otevře kanál během, kterého mají obě strany možnost spolu komunikovat. Kanál obsahuje název podle, kterého je možné rozlišovat jednotlivé připojení.[27][20]

Následující ukázka popisuje komunikaci mezi Content a Background skriptem. Na straně Content scriptu se postupuje následovně:

Ve výpisu 4.6 a 4.7 je znázorněna komunikace mezi Content a Background scriptem pomocí dlouhodobého spojení. Strana Content scriptu vypadá následovně. [27]

1. Vytvoření spojení s Background scriptem a uložení portu do proměnné `myPort`.
2. Čekání na odpověď od Background scriptu.
3. Odeslání Backgroundu scriptu zprávu, kdykoliv uživatel klikne na tlačítko.

```
var myPort = browser.runtime.connect({name:"port-from-cs"});
myPort.postMessage({greeting: "hello from content script"});

myPort.onMessage.addListener(function(m) {
  console.log("Received message from background script: ");
  console.log(m.greeting);
});

document.body.addEventListener("click", function() {
  myPort.postMessage({greeting: "they clicked the page!"});
});
```

Výpis 4.6: Dlouhodobé spojení - Content script (Zdroj: [27])

Strana Background scriptu má následující podobu. [27]

1. Čekání na spojení s Content scriptem.
2. Při pokusu o připojení se uloží port do proměnné. Následuje odeslání zprávy do Content scriptu o používání portu. Poté začne naslouchání zprávy na portu její zobrazení v konzolovém řádku.

3. Odeslání zprávy do Content scriptu, kdykoliv když uživatel klikne na tlačítko rozšíření.

```
var portFromCS;

function connected(p) {
  portFromCS = p;
  portFromCS.postMessage({greeting: "hi there content script!"});
  portFromCS.onMessage.addListener(function(m) {
    console.log("Received message from content script")
    console.log(m.greeting);
  });
}

browser.runtime.onConnect.addListener(connected);

browser.browserAction.onClicked.addListener(function() {
  portFromCS.postMessage({greeting: "they clicked the button!"});
});
```

Výpis 4.7: Dlouhodobé spojení - Background script (Zdroj: [27])

4.6 Komunikace v síti

Podkapitola zaměřená na způsoby jakými lze vytvořit komunikaci mezi rozšířením a serverem. Veškeré zde zmíněné způsoby komunikace jsou vytvořené pomocí objektu XMLHttpRequest (dále už jen „XHR“).[10]

Základní metody, které jsou součástí požadavku, který se odesílá na server. Všechny zde zmíněné metody jsou součástí protokolu HTTP 1.1. [1]

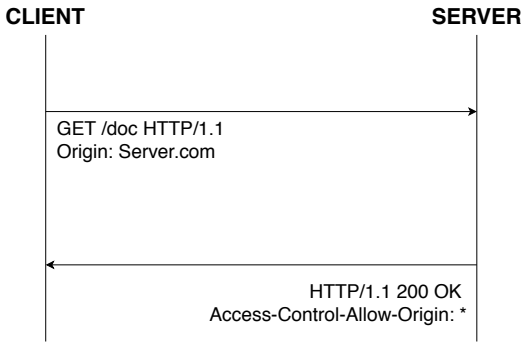
- **GET** - Získá požadované informace, které jsou identifikovány pomocí Requst-URI.
- **POST** - Tahle metoda se používá k tomu, aby server přijal nový subjekt indentifikovaný pomocí Request-URI.
- **HEAD** - Je identická metodě GET až na to že nevrací tělo zprávy. Metainformace obsažené v HTTP hlavičce v odpovědi mohou být identické jak u metody GET. Používá se k ověření zda soubor na serveru existuje.
- **PUT** - Vyžaduje, aby odeslaný subjekt byl uložený pod dodaným Requset-URI. V případě již existujícího cílového subjektu, provede nahrazení dat.
- **DELETE** - Vyžaduje aby server odstranil prostředek identifikovány pomocí Requst-URI.
- **CONNECT** - Využívá se pro tunelování. Například SSL tunelování.
- **OPTIONS** - Získá informace o komunikačních možnostech mezi klientem a serverem, které jsou identifikovány pomocí Requst-URI.
- **TRACE** - Využívá se při sledování cesty dotazu.
- **PATCH** - Téměř stejná jako metoda PUT. Rozdíl je v tom, že PATCH obsahuje seznam rozdílů mezi původní verzí subjektu identifikovanou pomocí Request-URI a verzí po provedení metody PATCH.[9]

Povolené hodnoty, které může obsahovat tělo zprávy. [3]

- **application/x-www-form-urlencoded** - Kódovaná forma URL.
- **multipart/form-data** - Používá se pro nahrání souborů.
- **text/plain** - Jedná se čistě o text.

Simple

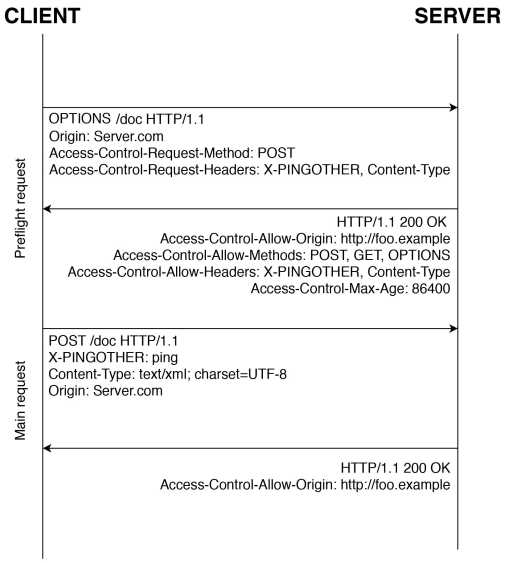
Nejprostším způsobem komunikace je tzv. Simple request. V překladu jednoduchý požadavek. Jak lze vidět na obrázku 4.1 požadavek obsahuje odeslání požadavku s metodou na server a přijmutí odpovědi. Povolené metody jsou: `GET`, `POST` a `HEAD`. Hlavička může obsahovat pouze hodnoty typu: `application/x-www-form-urlencoded`, `multipart/form-data` a `text/plain`.[10]



Obrázek 4.1: Simple - Klient odeslal serveru požadavek na dokument. Hlavička obsahuje kromě metody i doménu serveru. Server odpověď zpracoval a odpověděl. Klient obdrží odpověď, podle návratové hodnoty 200 zjistí, že všechno proběhlo v pořádku. (Zdroj: [10])

Preflight

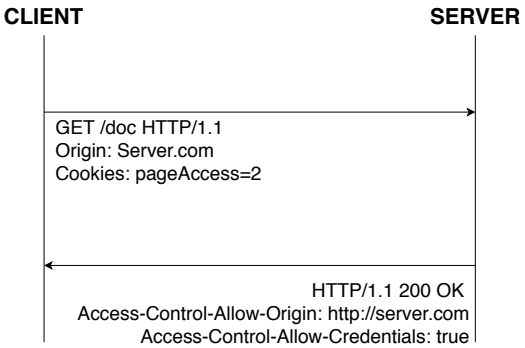
Rozdíl mezi Preflight a Simple požadavkem je v tom, že u Preflight před odesláním požadavku, proběhne odeslání požadavku s metodou OPTIONS. Důvodem je ověření zda je spojení bezpečné. Pokud vše proběhne bez problému, následuje odeslání hlavního požadavku. Povolené metody jsou: `PUT`, `DELETE`, `CONNECT`, `OPTIONS`, `TRACE` a `PATCH`. Tělo zprávy může obsahovat hodnoty typu: `application/x-www-form-urlencoded`, `multipart/form-data` a `text/plain`.[10]



Obrázek 4.2: Preflight - Klient odešle požadavek s metodou OPTIONS. Server odpoví a vypíše všechny metody, které podporuje. Následuje odeslání hlavního požadavku s metodou POST. Server následně odešle odpověď, že všechno proběhlo v pořádku. (Zdroj: [10])

Cookies

Tenhle typ umožňuje provádět ověření požadavků pomocí cookies. Simple požadavky a Cookies požadavky jsou stejné až na použití cookies pro ověření požadavků. [10]



Obrázek 4.3: Cookies - Klient odeslal serveru požadavek na dokument. Hlavička obsahuje kromě metody i doménu serveru a cookies. Server odpověď zpracoval a odpověděl. Klient obdrží odpověď, podle návratové hodnoty 200 zjistí, že všechno proběhlo v pořádku. (Zdroj: [10])

4.7 Úložný prostor prohlížeče

Tahle podkapitola se zaměřuje na způsoby, které může rozšíření využít k tomu, aby mohlo pracovat s pamětí webového prohlížeče. Rozšíření má přístup k dočasnému i k trvalému úložišti. Ukládání dat do paměti prohlížeče, je také efektivní způsob pro přesun informací mezi soubory v rámci rozšíření.

LocalStorage

Data uložené tímhle způsobem obsahují klíč pod, kterým lze k uloženým datům přistupovat. Klíč i hodnota jsou textové řetězce. Pokud dojde k zadání například číselné hodnoty, tak se převede na textový řetězec. Již uložená data nelze v paměti přepisovat, lze je pouze číst. Pro jejich změnu je nutné data načíst do proměnné. V proměnné data patřičně upravit. Vymazat z paměti původní data a uložit do paměti upravená data. Data nemají určenou dobu expirace jako u sessionStorage. v localStorage i v sessionStorage jsou data specifická pro protokol stránky. [41]

```
localStorage.setItem('myData', 'text');
var someData = localStorage.getItem('myData');
localStorage.removeItem('myData');
localStorage.clear();
```

Výpis 4.8: První řádek znázorňuje vytvoření nového objektu. Na druhém řádku se získá hodnota daného objektu. Na třetím se objekt odstraní z paměti a na čtvrtém se odstraní všechny objekty. (Zdroj: [41])

SessionStorage

LocalStorage a sessionStorage jsou téměř totožné. Jediný rozdíl je v času expirace uložených dat. U localStorage není stanoven čas expirace, ale u sessionStorage se data odstraní při ukončení relace webové stránky. Relace stránky trvá do té doby, dokud je prohlížeč otevřen. Trvá i přes znovu načtení webové stránky a i přes její obnovení. [4]

```
sessionStorage.setItem('myData', 'text');
let data = sessionStorage.getItem('myData');
sessionStorage.removeItem('myData');
sessionStorage.clear();
```

Výpis 4.9: Na prvním řádku dojde uložení dat. Na druhém k jejich načtení do proměnné. Na třetím řádku se klíč i data odstraní z paměti prohlížeče. Na posledním řádku se odstraní všechny data z paměti prohlížeče. (Zdroj: [4])

Storage.local

Pro použití téhle lokální paměti je nutné mít v manifest souboru povolení storage. Jedná se o lokální paměť, jejíž výchozí nastavení limituje maximální kapacitu přibližně na 5MB. To platí u prohlížečů Google Chrome i Mozilla FireFox. Výchozí nastavení, které limituje kapacitu úložiště lze změnit v manifest souboru a to použitím zvláštního povolení u klíče permissions. Povolení má název unlimitedstorage. Data uložená tímhle způsobem jsou zachová i při uzavření a znovuotevření webového prohlížeče. K jejich smazání dojde při odinstalování rozšíření. Druhou možností jak data odstranit je použitím speciální příkazu `storage.local.remove()`, nebo `storage.local.clear()`. Oproti sessionStorage a localStorage lze přepisovat hodnoty v paměti. Pokud u metody Storage.local použijeme již existující klíč, dojde k nahrazení staré hodnoty novou hodnotou. [37]

```
chrome.storage.local.set({myData: "text"});

chrome.storage.local.get(['myData'], function(result) {
    console.log(result.myData);
});

let removeInfo = chrome.storage.sync.remove("myData");
let clearInfo = chrome.storage.local.clear();

chrome.storage.local.getBytesInUse("myData");
```

Výpis 4.10: Na prvním řádku lze vidět nastavení nového klíče s hodnotou. Poté nahrání hodnoty klíče do proměnné. Dále odstranění konkrétního klíče. Na dalším řádku odstranění všech klíčů. Do daných proměnných s uloží u Firefoxu promise a u Chrome callback, obsahující informace o úspěšném či neúspěšném odstranění klíče. Posledním zmíněný příkaz vrací velikost zaplněné paměti v bajtech. (Zdroj: [37])

Storage.sync

Synchronizační paměť pro jejíž použití je nutné mít v manifest souboru povolení storage. Tenhle typ paměti se vyznačuje nízkou kapacitou, která je v prohlížečích Google Chrome a

Mozilla FireFox okolo 100kB. Využití nalézá v ukládání nastavení rozšíření, které může sdílet mezi různými profily prohlížeče. Například prostřednictvím synchronizace profilů Firefox, či Google účtů napříč zařízeními. [38]

Obsahuje stejné příkazy jako storage.local: `get`, `set`, `remove`, `clear`, `getBytesInUse`. Příkazy jsou ukázány ve výpisu 4.10. [38]

	FireFox	**Chrome**	**Edge**	**Opera**	**FireFox Android**
getBytesInUse()	NE	ANO	14	33	NE

Tabulka 4.4: Znázorňuje webové prohlížeče a jejich podporu vůči příkazu getBytesInUse(). Podpora je znázorněna slovy a nebo verzí od jaké je příkaz podporován. (Zdroj: [36])

4.8 CSP - Content Security Policy

Content Security Policy, neboli zásady zabezpečení obsahu (dále už jen „CSP“). Jsou zahrnuty ve výchozím nastavení rozšíření u Webextensions i Extension APIs. U webových prohlížečů založených na technologii WebExtensions lze výchozí stav upravit. CSP klade omezení na potencionálně nebezpečné postupy, které lze načíst pomocí HTML tagů `<script>` a `<objekt>`.[42]

Hlavním úkolem CSP je detekování, nebo alespoň zmírnění některých druhů útoku. Mezi takové útoky patří Cross Site Scripting a Data injection. Jsou to typy útoků, díky kterým se může útočník zmocnit dat, nebo šířit svůj škodlivý kód. [46]

CSP s těmito hrozbami bojuje především určováním legitimních zdrojů. Mezi takové zdroje můžou patřit vložené skripty, nebo pluginy. Například JavaScriptový kód obsluhovaný z jiných zdrojů než z webové stránky je zakázaný, stejně jako použití příkazu `eval()`.[42]

.

Změna výchozího nastavení

Výchozí nastavení CSP je ve tvaru:

"script-src 'self'; object-src 'self';"

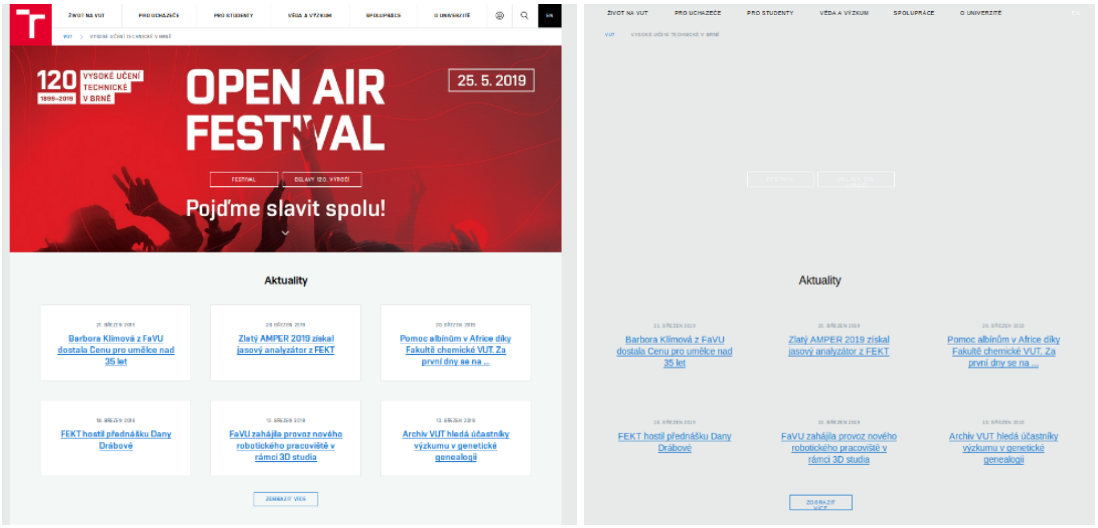
Změna výchozí stavu CSP se provádí v manifest souboru u klíče `content_security_policy`.[42] Nastavení lze změnit pomocí následujících klíčových slov. [28]

- **unsafe-eval** - Povoluje, aby rozšíření mohlo používat příkaz`eval()`.
- **script-src** - Musí být uveden s kombinací s klíčovým slovem `self`. A dovoluje, aby mohl být JavaScriptový kód načítán i z inline zápisu.
- **object-src** - Specifikuje zdroje pro HTML elementy: `<object>`, `<embed>`, a `<applet>`.

Kapitola 5

Rozšíření pro analýzu stránek

Tahle kapitola bakalářské práce je zaměřena na způsob jakým je dané téma navrhnuto a implementováno. Na následujícím obrázku 5.1 je názorná ukázka toho jak rozšíření funguje. Na levé straně je webová stránka, na které je rozšíření spuštěno. V rozšíření se zpracuje celá webová stránka a zpracované textové hodnoty se odešlou ve formátu JSON na server. Server vygeneruje HTML5 strukturu, do které vsadí zpracované textové hodnoty a výsledek odešle zpět. Rozšíření zpracuje odpověď serveru a zobrazí ji v nové kartě webového prohlížeče. Nová karta je zobrazena na pravé straně obrázku 5.1. Výsledkem je webová stránka, která se skládá pouze s textových elementů. CSS vlastnosti i pozice textových elementů jsou zachovány.



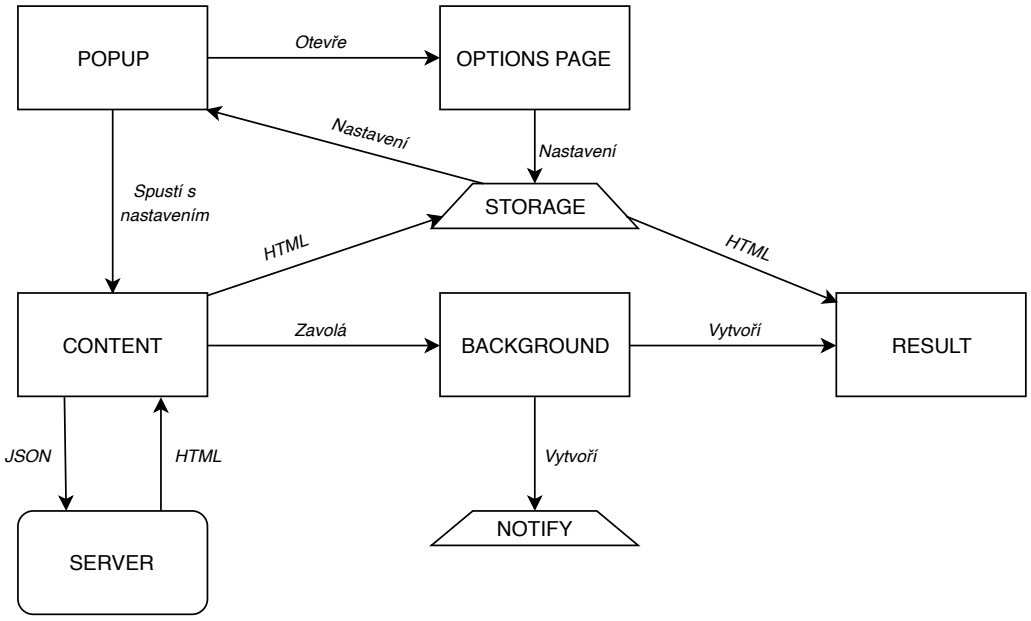
Obrázek 5.1: Na levé straně obrázku je webová stránka, na které bylo rozšíření spuštěno. Na pravé straně je výsledek. (www.vutbr.cz)

Tahle kapitola je rozdělena do několika podkapitol. V první podkapitole 5.1 je zobrazena a popsána základní podoba rozšíření. Podkapitola je rozdělena do několika sekcí, kde je velice dopodrobna vysvětleno jak mezi sebou jednotlivé komponenty rozšíření spolupracují a co všechno bylo nutné pro to udělat. Podkapitola 5.2 se zabývá zpracováním textových elementů z webové stránky. Podkapitola 5.3 se zabývá získáváním vypočteného stylu pro textový element. Podkapitola 5.4 se zabývá skrytými textovými elementy. Je zde

vysvětleno jakým způsobem je možné textové elementy skrýt a jak s nimi pracuje rozšíření. V podkapitole 5.5 je vysvětleno jakým způsobem se zjišťuje absolutní pozice textových elementů na webové stránce. V podkapitole 5.6 je popsáno jakým způsobem se tvoří a co všechno obsahuje JSON soubor, který je rozšířením odeslán serveru. V podkapitole 5.7 je podrobně vysvětlena komunikace mezi rozšířením a serverem. V poslední podkapitole 5.8 je vysvětleno, jakým způsobem server zpracuje požadavek od rozšíření. Následně jak probíhá vygenerování odpovědi a jak její odeslání zpět k rozšíření.

5.1 Rozšíření

Jako aplikační rozhraní pro rozšíření webového prohlížeče jsem si vybral Extension APIs místo WebExtensions. Důvodem byla kompatibilita s prohlížeči Mozilla FireFox a Google Chrome. Jak již bylo zmíněno v podkapitole 4.3, rozšíření napsané pro Extension APIs bez větších změn pojedou i ve webových prohlížečích založených na technologii WebExtensions. Proto jsem se v implementaci vyhl metodám, které by nebyli u WebExtensions podporované. Výsledkem je rozšíření, které je kompatibilní v prohlížeči Google Chrome a Mozilla FireFox bez použití externích knihoven.



Obrázek 5.2: Diagram znázorňující podobu rozšíření.

Permissions

Jak již bylo zmíněno v kapitole 4.2. Rozšíření musí mít povolení k tomu, aby mohlo využívat speciálních funkcí webového prohlížeče. Jednotlivé povolení se píší v manifest souboru u klíče s názvem „permissions“. Zde jsou vypsány povolení, které obsahuje rozšíření této bakalářská práce:

- **activeTab** - Dočasné povolení pro přístup k aktivní kartě webového prohlížeče v případě, kdy uživatel interaguje s rozšířením [32].
- **webRequest** - Rozšíření získá přístup k WebRequest API, které je potřebné pro komunikaci ze serverem [48].
- **<all_urls>** - Zastupuje všechny URL zápisy [47]. Tohle povolení je potřebné k tomu, aby šel Content script spustit na všech webových stránkách.
- **tabs** - Dovoluje, aby rozšíření mohlo vytvářet nové karty [13]. Na nové kartě webového prohlížeče se zobrazuje výsledek rozšíření.
- **unlimitedStorage** - Lokální úložiště webového prohlížeče bude mít neomezenou kapacitu [37]. Povolení je potřebné, protože úložiště je limitováno přibližně na 5 MB a potřeby rozšíření mohou limit přesáhnout.
- **notifications** - Rozšíření může vytvářet notifikace webového prohlížeče [31]. Notifikace zobrazující chybové hlášení.

Popup

Pro umístění dialogového okna jsem se rozhodl využít panel nástrojů místo adresního řádku. Jelikož logem rozšíření je lupa, která by v adresním řádku mohla připomínat spíše logo na změnu velikosti webové stránky.

Popup obsahuje HTML, CSS a JavaScriptový soubor. Jelikož je Popup komponentou, která je svým způsobem prostředník mezi uživatelem a hlavní funkcí rozšíření, snažil jsem se, aby na uživatele působila příjemně. Jednou z možností je přepnutí dialogového okna do tmavého režimu. K možnosti tmavého režimu se v poslední době uchyluje spousta známých aplikací. Jednou z nich je například Messenger od společnosti Facebook.

Při spuštění dialogové okna dojde k načtení informací z lokální paměti webového prohlížeče. Na diagramu 5.1 je tahle paměť znázorněna slovem „STORAGE“. Načtené informace obsahují URL adresu serveru, jméno aktuálního vzhledového schématu a informace o tom, zda se budou zpracovávat i skryté textové elementy. Podle jména aktuálního vzhledového schématu se vygeneruje předem definovaný vzhled dialogového okna a čeká se na další krok uživatele:

- **Zpracování textu** - Odešle se jednorázová zpráva, kterou přijme komponenta Content. Komponenta Content obsahuje JavaScriptový soubor spuštěný s aktuálně zobrazenou stránkou. Zpráva obsahuje URL adresu serveru a informace o tom zda se mají zpracovávat i skryté textové elementy. Po té dojde k uzavření dialogové okna pomocí příkazu `window.close()`.
- **Options page** - Pomocí příkazu `chrome.runtime.openOptionsPage()` se otevře webová stránka pro správu rozšíření v nové kartě webového prohlížeče a stane se aktuálně zobrazenou kartou. Po té dojde k uzavření dialogové okna pomocí příkazu `window.close()`.

Content

Content se skládá z JavaScriptového souboru (dále už jen „Content script“), který obsahuje hlavní algoritmus rozšíření. Pro každou nově otevřenou webovou stránkou se spustí Content

script, který čeká na příchozí zprávu od komponenty Popup. Po obdržení zprávy dojde ke kontrole, zda zpráva obsahuje URL adresu serveru. Zatím se nekontroluje, jestli je URL adresa validní.

V případě že zpráva neobsahuje URL adresu serveru, odešle se zpráva, kterou zachytí komponenta Background. Content script se uzavře a Background vytvoří notifikaci, která o chybějící URL adrese serveru upozorní uživatele. Důvod proč notifikaci vytváří Background je ten, že Content script má omezený přístup k Extension APIs.

Když Content script obdrží validní zprávu, tak provede zpracování textových elementů. Princip je popsán v podkapitole 5.2. Zpracované textové elementy se v JSON formátu odešlou na server, který je zpracuje a jako odpoví plnohodnotným HTML 5 kódem. Zpracování textových elementů je podrobněji popsané v podkapitole 5.7.

Při problému s navázáním spojení se serverem, nebo při neobdržení odpovědi od serveru, dojde opět k vytvoření upozorňující notifikaci. V téhle části se řeší, zda je URL adresa serveru validní.

Content script příchozí HTML kód uloží do lokální paměti prohlížeče `local.storage`. Typ úložiště je vysvětlen v podkapitole 4.7. Předávání dat mezi jednotlivými skripty pomocí úložiště, se mi osvědčilo jako nejefektivnější způsob předávání dat v rozšíření.

Options page

Options page se skládá z HTML, CSS a JavaScriptového souboru. V Options page se nastavují základní vlastnosti rozšíření. Nejdůležitějším je URL adresa serveru. Dalšími jsou nastavení vzhledu dialogového okna a zpracování skrytých textových elementů. Skryté textové elementy jsou podrobněji popsané v sekci Zpracování textu 5.2.

Nastavené hodnoty se ukládají do úložiště webového prohlížeče `storage.local`, kde budou uloženy i po znovu otevření webového prohlížeče. Úložiště `storage.local` je rozebráno v podkapitole 4.7.

Rozhodl jsem se pro zobrazení Options page v nové kartě, místo zobrazení ve správě rozšíření. Důvod je pouze estetický. Na nové kartě jsem mohl vzhled Options page udělat podle svých představ a nikoliv tak aby zapadl do vzhledu webového prohlížeče.

Background

Background obsahuje jeden JavaScriptový soubor, který přijímá zprávy od Content skriptu. Po přijetí zprávy zjistí jaký příkaz obsahuje a podle toho se vykoná jedna ze dvou činností, které background skript dělá.

- **Notifikace** - Vytvoření notifikace, která uživatele informuje o tom, že je něco v nepořádku ze serverem. Neplatná URL adresa serveru, nepodařilo se vytvořit spojení ze serverem, nedošla odpověď ze serveru.
- **Karta prohlížeče** - Vytvoření nové karty webového prohlížeče, který zobrazí HTML kód, který odeslal server. Jedná se o kartu, která zobrazí výsledek.

Výsledná karta

Obsahuje předem připravený HTML soubor a JavaScriptový soubor. JavaScriptový soubor načte z úložiště webové prohlížeče HTML kód, který byl uložen Content scriptem. HTML kód obsahuje výsledek rozšíření. Následně dojde u připraveného HTML souboru k nahrazení elementu `<body>`. Nahradí ho `<body>` element z HTML kódu, který byl načten z úložiště

webového prohlížeče. Tímto způsobem dojde k zobrazení výsledku rozšíření v nové kartě webového prohlížeče.

5.2 Zpracování textu

Jak již bylo zmíněno v podkapitole 4.2. Komponenta Content má jako jediná v rozšíření přímý přístup ke kontextu webové stránky. Proto zpracování textových elementů musí proběhnout právě tam.

Pro přístup ke kontextu webové stránky je použit příkaz `document`. Jelikož se textové elementy nachází v těle `<body>`, tak je použit příkaz `querySelector('body')`. Ten vrací první element, který se shoduje s hledaným textem. V tomhle případě je to „body“. A nakonec se vypíší jeho přímý následovníci. Celý příkaz bude vypadat následovně:

document.querySelector('body').childNodes

Získá se tím struktura obsahující všechny elementy webové stránky. Tahle struktura bude dále v textu označovaná jako „NodesList“.

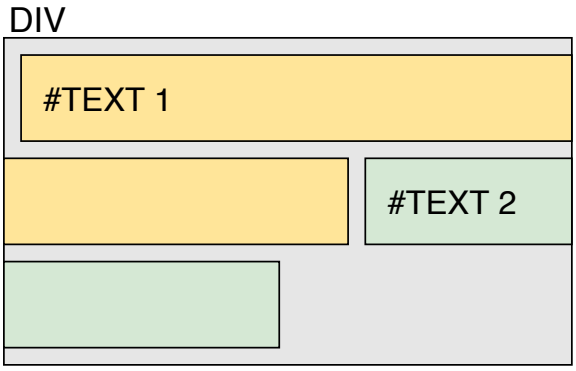
Zpracování textu je implementováno pomocí funkce, která postupně zpracovává jednotlivé uzly struktury NodesList. Uzly reprezentují jednotlivé HTML elementy dokumentu. V případě že narazí na uzly obsahující podřízené uzly, tak proběhne rekurzivní volání téže funkce, kde bude parametrem daný uzel. Tímhle způsobem dojde ke zpracování všech uzlů struktury NodesList.

Ne všechny HTML elementy jsou důležité pro zpracování. Elementy jako jsou: `script`, `noscript`, `comment`, `img` a `style` algoritmus přeskakuje, protože neobsahují textové elementy, které mají být zobrazeny ve výsledku. I když se zpracování textových elementů může zdát přímočaré, jsou zde věci, na které je potřeba se zaměřit.

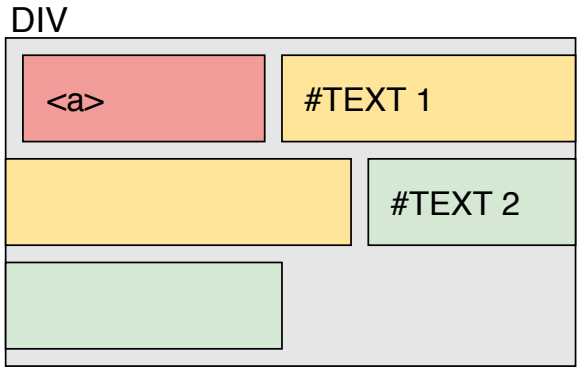
V téhle struktuře je text podřízeným uzlem nějakého nadřazeného uzlu, který obsahuje všechny jeho potřebné vlastnosti až na samotnou textovou hodnotu. Tudíž pokud uzel má více podřízených uzlů, které obsahují textové hodnoty. Všechny tyhle textové hodnoty mají stejné vlastnosti, včetně pozice textu. Tudíž je potřeba zjistit, kde se přesně text má nacházet, aby ve výsledku všechny texty uzlu, nebyly vypsané na jednom místě. Zjištění pozice textových elementů je podrobně vysvětleno v podkapitole 5.5.

První element

Na následujících obrázcích 5.3 a 5.4 je znázorněno jaký má vliv první element na zpracování textových elementů. Na obou obrázcích se nachází HTML element `<div>`, který obsahuje podřízené textové elementy (dále už jen „Text1“ a „Text2“) a v obrázku 5.4 je i odkaz. Odkaz je HTML realizován pomocí tagu `<a>`.



Obrázek 5.3: Oba texty mají společný nadřízený uzel. Tudíž mají stejnou absolutní pozici a stejné CSS vlastnosti. Proto dojde ke spojení obou textů dohromady. Výsledkem bude jeden textový element obsahující řetězce Textu1 a Textu2.



Obrázek 5.4: I když to na obrázku není zobrazeno, tak odkaz obsahuje taky textový element. Nejprve se zpracuje <a>. Poté se začne Text1, který pozici nebude zjišťovat z nadřazeného uzlu jak u minulého příkladu, ale zjistí pozici a rozměr posledního přidaného textového elementu. Posledním přidaný text je z odkazu. Vypočítá se nová pozice pro Text1. Text2 se připojí k Textu1 jako v minulém příkladu.

Obecně platí: Pokud je prvním elementem v uzlu textový element, zjišťuje se pozice a vlastnosti z nadřazeného uzlu. A následující textové elementy se spojují s předchozím textovým elementem.

V případě, kdy se mezi textovými elementy objeví element, který obsahuje podřízené textové elementy. Tak následující textové elementy si musí vypočítat z posledního přidaného elementu novou pozici. Jinak může dojít k překrytí textu jiným textem.

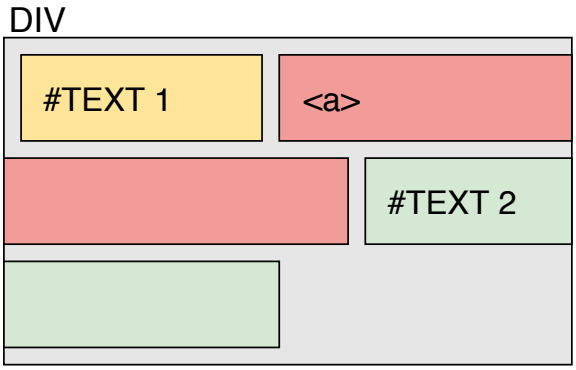
Zalomení řádku

Zalamování řádku u textových hodnot probíhá automaticky. Bohužel to neplatí u HTML elementů, které obsahují podřazené elementy obsahující textové elementy. Jedním z takových elementů je odkaz `<a>`. Na obrázku 5.5 lze vidět, že odkaz přesahuje rámec nadřazeného elementu `DIV` a je potřeba ho zalomit. Algoritmus nejprve podle následujícího vzorce zjistí, jestli element přesahuje rámec.

(aPosition.x + aSize.width - boxPosition.x) > boxSize.width

- *aPosition.x* - Pozice odkazu na webové stránce. Stačí souřadnice na ose x, protože počítáme šířku.
- *aSize.width* - Šířka odkazu.
- *boxPosition.x* - Pozice nadřazeného elementu. Na obrázku 5.5 se jedná o `DIV`.
- *boxSize.width* - Šířka nadřazeného elementu. Na obrázku 5.5 se jedná o `DIV`.

Pokud odkaz přesahuje šířku rámce, jeho textová hodnota se přiřadí rámci a dojde k automatickému zalomení. Na obrázku 5.5 to znamená, že na odkaz se bude nahlížet jako na textovou hodnotu, která je součástí Text1. Nevýhoda takového řešení je ta, že textové hodnotě odkazu budou nahrazeny její CSS vlastnosti. Bude mít CSS vlastnosti nadřazeného elementu. V tomhle případě elementu `DIV`.



Obrázek 5.5: Zalomení řádku

5.3 Vypočtený styl

Tahle sekce se zabývá způsobem jak zjistit vypočtený styl textových elementů. Vypočteným stylem se rozumí výsledný CSS styl. Protože textový element může mít definovaných více CSS stylu je potřeba zjistit, který je ten výsledný co se projeví při zobrazení.

Máme tři způsoby jak definovat CSS styl:

- **Inline** - CSS styl definován přímo v HTML tagu. Má nejvyšší prioritu. Například: <p style=”color:blue;” >.
- **HTML style tag** - Styl uvnitř tagů <style> ve stejném HTML dokumentu, kde se nachází text, pro který je styl definován. Má druhou největší prioritu.
- **Vzdálený stylesheet** - Styl, který je importován v hlavičce HTML souboru. Ve tvaru: <link rel=”stylesheet” type=”text/css” href=”mystyle.css”>. Má nejnižší prioritu.

Prioritu dále ovlivňují selektory. Selektor s nejvyšší prioritou je „id“. Ovšem nemá přednost před inline vložením CSS. Druhou nejvyšší prioritu má selektor „class“ a třetí je

selektor, který vybere určitý druh html tagů. Například selektor „a“, který vybere všechny odkazy.

O výsledném stylu rozhoduje i pořadí v CSS souboru. Pokud by existovala duplicitní definice, která používá stejný selektor, na výsledném zobrazení se projeví ta co se v dokumentu nachází později.

Pro určení nejvyšší priority a tím i pro určení vypočteného stylu jsem se rozhodl využít metodu `getComputedStyle()`. Jejím vstupem je HTML element, který obsahuje textovou hodnotu a výstupem je vypočtený styl ve formátu CSS2.

Určení vypočteného stylu probíhá v komponentě rozšíření, která se v téhle práci nazývá Content.

5.4 Skryté textové elementy

Výchozí nastavení rozšíření je nastaveno tak, aby zpracovávalo pouze viditelné textové elementy. Tohle nastavení však lze vypnout a rozšíření bude zpracovávat i ty skryté. Rozeznání skrytých textových elementů od těch viditelných probíhá na základě těchto CSS vlastností:

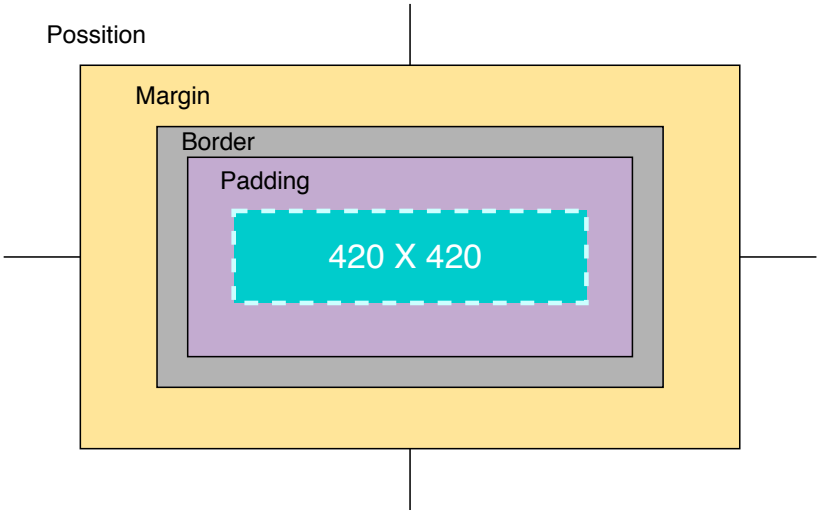
- **Opacity** - Neboli neprůhlednost. V případě, kdy má Opacity hodnotu „0“ stává se neviditelným a rozšíření ho vnímá jako skrytý element.
- **Visibility** - V případě, kdy je viditelnost nastavena na hodnotu „hidden“ stává se skrytým textovým elementem.
- **Display** - Poslední CSS vlastností, která dokáže skrýt textový element, je Display nastavený na hodnotu „none“.

Rozeznávání skrytých textových elementů od neskrytých není úplně dokonalé a to kvůli CSS vlastnosti Overflow. Vlastnost Overflow určuje co se s obsahem elementu stane poté co její obsah přeteče. Pokud je jeho hodnota „hidden“, tak se přetečený obsah skryje a rozšíření nerozezná, která část obsahu je skrytá a která ne.

5.5 Pozice textu

Na následujícím obrázku 5.6 je zobrazený diagram, který znázorňuje co všechno určuje pozici elementu na webové stránce. Jedná se o následující CSS vlastnosti:

- **Margin** - Odsazení od okolních HTML elementů tzv. vnější okraj.[53]
- **Border** - Ohraničení HTML elementu.[52]
- **Padding** - Vnitřní okraj určující prostor kolem obsahu prvku uvnitř definovaných hranic. [54]



Obrázek 5.6: Diagram pozice HTML elementu.

Webová stránka se většinou skládá s více vnořených HTML elementů. Každý element může mít libovolné ohraničení, či odsazení. Tudíž k zjištění absolutní pozice HTML elementu na stránce je potřeba sečíst všechny ohraničení a odsazení všech nadřazených HTML elementů. Posledním nadřazeným HTML elementem je `<body>`. V CSSOM View Module march 2019 jsou zmíněny nové vlastnosti, které tenhle problém usnadní. Jsou to `offsetLeft`, `offsetTop`, `ClientLeft` a `ClientTop`. Vlastnosti `Offset` nahrazují `margin`, `padding` a odsazení `left` a `right`. Vlastnosti `CLient` nahrazují vlastnost `border`. [6]

5.6 Vytvoření JSON formátu

Požadavek odeslaný rozšířením na server obsahuje všechny potřebné informace k tomu, aby byl server schopný vytvořit výslednou HTML stránku. Všechny potřebné informace jsou zahrnuty v JSON formátu. Příklad je zobrazený ve výpisu 5.1.

Prvním klíčem je `description`. Je to popis, který nemá vliv na žádnou funkci serveru. Klíč `url` značí URL adresu stránky, z které byly zpracovány textové elementy. Protože výsledná stránka bude mít jako titulek stránky URL adresu, z které byla vytvořena. Klíč `backgroundColor` znázorňuje barvu pozadí celé stránky. Klíč `text_elements` obsahuje jednotlivé textové hodnoty webové stránky. Textová hodnota obsahuje klíč `Xtext` značící

samotný textový řetězec, `Xposition` značící absolutní pozici na webové stránce, `Xsize` znázorňující nadřazeného HTML elementu, ve kterém se text nachází. Další klíče obsahují CSS vlastnosti.

```
{
"description":"Output from Page Analysis WebExtensions app.",
"url":"https://www.seznam.cz",
"backgroundColor":"rgb(255, 255, 255)",
"text_elements":[{
    "Xtext":"Seznam",
    "Xposition":{"x":1278,"y":884},
    "Xsize":{"height":33,"width":120},
    ...
```

Výpis 5.1: Příklad požadavku ve formátu JSON, který je odeslán rozšířením na server.

5.7 Komunikace se serverem

Po vytvoření souboru ve formátu JSON je na řadě samotná komunikace se serverem. Komunikace je implementována pomocí objektu XmlHttpRequest (dále už jen „XHR“). U XHR jsem zvolil jednoduchý způsob dotazovaní, který je podrobně vysvětlen v podkapitole 4.6 pod názvem „Simple“. Rozšíření odešle JSON na uvedenou adresu serveru a čeká na odpověď. Jestli všechno proběhne v pořádku, tak od serveru obdrží odpověď, která obsahuje výsledný HTML soubor.

5.8 Server

Strana serveru je implementována v jazyce PHP. Skládá se z jednoho PHP souboru, který využívá rozhraní DOMDocument(dále už jen „DOM“). Jeho účelem je přijmout zprávu odeslanou komponentou Content. Dále zprávu zpracovat. Vytvořit z ní HTML kód, který bude obsahovat textové elementy s příslušnými CSS vlastnostmi a absolutně určenou pozicí.

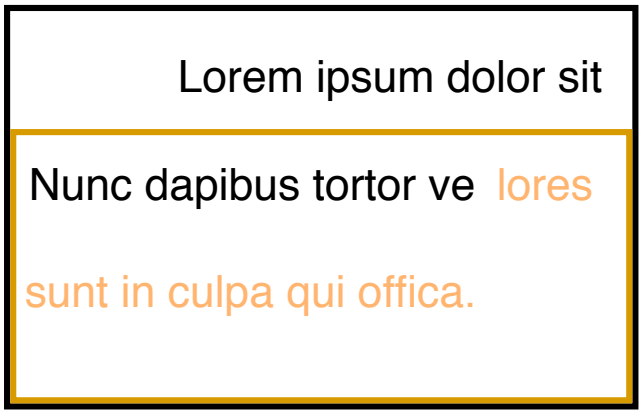
1. U přijaté zprávy dojde k dekódování HTML entit.
2. Vygenerování základní HTML struktury, do které se postupně budou přidávat zpracované informace z přijaté zprávy.
3. Přidání barvy pozadí pro celý HTML dokument.
4. Generování jednotlivých HTML elementů, které obsahují textové řetězce.
5. Odeslání odpovědi.

Generování textových elementů

Pro každý textový element je vygenerován pomocí DOMu samostatný DIV element, kterému jsou přiřazeny do inline stylu všechny CSS vlastnosti daného textu, pevné rozměry a absolutní pozice na stránce. Poté je přidána i samotná textová hodnota.

Jednotlivé elementy DIV se vzájemně překrývají. Nevýhodou takového řešení je to, že nemůže být použita, u jednotlivých elementů DIV, CSS vlastnost `background-color`, neboli

barva pozadí. Barva pozadí by překrývala text u jiných elementů. Může nastat situace, kdy na výsledné kartě nebude viditelný text a to když barva textového elementu bude shodná s barvou pozadí celé stránky.



Obrázek 5.7: Znázorňuje dva HTML elementy DIV, které obsahují textové řetězce tvořící jednu větu. Jednotlivé textové řetězce jsou označeny rozdílnou barvou, která je totožná s elementem DIV, ke kterému patří. Jak lze vidět spojení textových hodnot je řešeno pomocí odsazení. Text druhého elementu je odsazen, tak aby navazoval přímo na text předešlého elementu.

Kapitola 6

Testování

Tato kapitola popisuje způsoby manuálního testování funkčnosti rozšíření. Rozšíření bylo otestováno ve webových prohlížečích Mozilla FireFox a Google Chrome. Během testování byl použit server, umístěný na localhostu i veřejný server podporující PHP. K manuálnímu testování byla využita zdrojová webová stránka a výsledná stránka obsahující pouze textový obsah. Zdrojovou stránkou se rozumí webová stránka, na které bylo spuštěno rozšíření a vznikla z ní výsledná stránka. V podkapitole 6.1 je vysvětlen způsob testování pomocí překrývání obrázků. V podkapitole 6.2 je rozebrán způsob testování pomocí vyhledávání.

6.1 Překrývání

Jedná se o manuální testování, kde se ze zdrojové a výsledné stránky vytvoří obrázky, které se v grafickém editoru umístí na sebe a u výsledné stránky se změní propustnost na 50%. Jak lze vidět na obrázku 6.1 vzhled i pozice textových elementů se může u zdrojové a výsledné stránky lišit. Textové elementy můžou být posunuté, nebo rozmístěné v jiném poměru než u zdrojové stránky. Tudíž tahle metoda není vhodná ve všech případech.

Obrázek 6.1: Obsahuje část obrázku, kde je umístěna zdrojová stránka, přes kterou je umístěna výsledná stránka z propustností 50%. (www.vutbr.cz)

6.2 Vyhledávání

Druhou použitou manuální metodou, kterou jsem si vybral je metoda založená na vyhledávání jednotlivých textů. Oproti předchozí metodě je časově náročnější, ale je vhodná i v případech, kdy jsou pozice textových elementů na jiných místech, než u zdrojové stránky. Z výsledné stránky jsou postupně vybrány všechny ucelené textové řetězce, které se vyhledají na zdrojové stránce. Vyhledávání probíhá ve webovém prohlížeči promocí funkce pro vyhledávání textu. U vyhledaného textu se zkontroluje vzhled a pozice textového elementu vůči ostatním textovým elementům.

Kapitola 7

Závěr

Cílem téhle práce bylo seznámit se s webovými technologiemi pro tvorbu rozšíření webových prohlížečů se zaměřením na technologií WebExtensions a s detaily reprezentace zobrazené webové stránky v prohlížeči a souvisejícím aplikačním rozhraním jednotlivých prohlížečů. Na základě těchto získaných znalostí navrhnout a implementovat rozšiřující modul prohlížeče, který umožní uživateli odeslat detaily o aktuálně zobrazené webové stránce a jejím textovém obsahu do serverové aplikace provádějící další analýzu a zobrazit výsledky této analýzy.

Po prostudování možností webových technologií, způsobů efektivního zpracování velkého množství textových elementů webové stránky a způsobu komunikace se serverem byl navržen a poté implementován rozšiřující modul webového prohlížeče, který je podporovaný v prohlížečích založených na technologii WebExtensions a Extension APIs. Na těchto webových technologiích jsou založeny nejpoužívanější webové prohlížeče. Nejznámějšími jsou Google Chrome založený na Extension APIs a Mozilla FireFox založený na WebExtensions. Rozšiřující modul, neboli rozšíření, zpracuje veškerý textový obsah webové stránky a odešle ho na server. Od serveru obdrží stránku složenou pouze z textového obsahu, kterou zobrazí v nové kartě prohlížeče. Server je implementován v jazyce PHP.

Práce s webovými technologiemi pro tvorbu rozšíření a způsobu získávání a zpracování textového obsahu webových stránek, pro mě bylo zajímavou zkušeností. Obzvlášť webové technologie, kterým bych se chtěl věnovat i v budoucnu.

V budoucnu by se tahle bakalářská práce mohla obohatit o zpracování i jiného než textového obsahu, například o obrázky. Strana serveru by mohla být obohacena o nové funkce. Server by mohl být propojený s databází, kde by po přihlášení uživatele mohlo docházet k ukládání vyhledaných údajů a jejich následné zpracování pro další účely.

Literatura

[1] Adams, C.; Gilchrist, J.: *The CAST-256 Encryption Algorithm*. RFC 2612, RFC Editor, June 1999.

[2] Almeroth, S.; aj.: *JavaScript*. Mozilla Developer Network, Duben 2019, [Online; navštíveno 11.04.2019].
URL https://developer.mozilla.org/en-US/docs/Web/JavaScript

[3] Bajali, D.: *Understanding HTML Form Encoding: URL Encoded and Multipart Forms*. [Online; navštíveno 16.04.2019].
URL https://dev.to/sidthesloth92/understanding-html-form-encoding-url-encoded-and-multipart-forms-3lpa

[4] Chao, A.; aj.: *Window.sessionStorage*. Mozilla Developer Network, Březen 2019, [Online; navštíveno 05.04.2019].
URL
https://developer.mozilla.org/en-US/docs/Web/API/Window/sessionStorage

[5] Chao, B.; aj.: *Whitespace in the DOM*. Mozilla Developer Network, Březen 2019, [Online; navštíveno 26.04.2019].
URL https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Whitespace_in_the_DOM

[6] Chinnathambi, K.: *Get an Element's Position Using JavaScript*. Březen 2016, [Online; navštíveno 08.03.2019].
URL
https://www.kirupa.com/html5/get_element_position_using_javascript.htm

[7] Ecma: *ECMAScript® 2020 Language Specification*. Ecma International, Duben 2019, [Online; navštíveno 11.04.2019].
URL https://tc39.github.io/ecma262/

[8] Ecma: *Testing and Comparison Operations*. Ecma International, Duben 2019, [Online; navštíveno 13.04.2019].
URL
https://tc39.github.io/ecma262/#sec-testing-and-comparison-operations

[9] Fielding, R.; Gettys, J.; Mogul, J.; aj.: *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2068, RFC Editor, January 1997.

[10] Fujimoto, M.; aj.: *Cross-Origin Resource Sharing (CORS)*. Mozilla Developer Network, Březen 2019, [Online; navštíveno 21.03.2019].
URL https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

[11] Garland, T.; aj.: *Anatomy of an extension*. Mozilla Developer Network, Únor 2019, [Online; navštíveno 08.03.2019].
URL https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Anatomy_of_a_WebExtension

[12] Giger, M.; aj.: *Popups*. Mozilla Developer Network, Červenec 2018, [Online; navštíveno 12.03.2019].
URL https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/user_interface/Popups

[13] Google: *chrome.tabs*. [Online; navštíveno 16.03.2019].
URL https://developer.chrome.com/extensions/tabs

[14] Google: *Content Scripts*. [Online; navštíveno 08.03.2019].
URL https://developer.chrome.com/extensions/content_scripts

[15] Google: *Declare Permissions*. [Online; navštíveno 08.03.2019].
URL https://developer.chrome.com/extensions/declare_permissions

[16] Google: *Manage Events with Background Scripts*. [Online; navštíveno 08.03.2019].
URL https://developer.chrome.com/extensions/background_pages

[17] Google: *Manifest - Icons*. [Online; navštíveno 08.03.2019].
URL https://developer.chrome.com/extensions/manifest/icons

[18] Google: *Manifest Version*. [Online; navštíveno 08.03.2019].
URL https://developer.chrome.com/extensions/manifestVersion

[19] Google: *Manifest - Web Accessible Resources*. [Online; navštíveno 13.03.2019].
URL https://developer.chrome.com/extensions/manifest/web_accessible_resources

[20] Google: *Message Passing*. [Online; navštíveno 10.03.2019].
URL https://developer.chrome.com/extensions/messaging

[21] Google: Publish in the Chrome Web Store. [Online; navštíveno 26.03.2019].
URL https://developers.chrome.com/webstore/publish

[22] Google: *What are extensions?* [Online; navštíveno 05.03.2019].
URL https://developer.chrome.com/extensions

[23] Greco, L.; aj.: *WebExtension browser API Polyfill*. [Online; navštíveno 14.04.2019].
URL https://github.com/mozilla/webextension-polyfill

[24] Hofmann, J.; aj.: *icons*. Mozilla Developer Network, Leden 2019, [Online; navštíveno 08.03.2019].
URL https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/manifest.json/icons

[25] Janovský, D.: *Začlenění skriptu do stránky*. [Online; navštíveno 11.04.2019].
URL https://www.jakpsatweb.cz/javascript/zacleneni.html

[26] Kaindl, C.; aj.: *background*. Mozilla Developer Network, Srpen 2018, [Online; navštíveno 08.03.2019].
URL https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/manifest.json/background

[27] Kingston, J.; aj.: *Content scripts*. Mozilla Developer Network, Leden 2019, [Online; navštíveno 08.03.2019].
URL https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Content_scripts

[28] Maglione, K.; aj.: *content_security_policy*. Mozilla Developer Network, Březen 2019, [Online; navštíveno 16.04.2019].
URL https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/manifest.json/content_security_policy

[29] Maglione, K.; aj.: *Firefox Add-on Distribution Agreement*. Mozilla Developer Network, Březen 2019, [Online; navštíveno 26.03.2019].
URL https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/AMO/Policy/Agreement

[30] Maglione, K.; aj.: *manifest.json*. Mozilla Developer Network, Leden 2019, [Online; navštíveno 08.03.2019].
URL https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/manifest.json

[31] Maglione, K.; aj.: *notifications.create()*. Mozilla Developer Network, Leden 2019, [Online; navštíveno 16.03.2019].
URL https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/notifications/create

[32] Maglione, K.; aj.: *permissions*. Mozilla Developer Network, Únor 2019, [Online; navštíveno 08.03.2019].
URL https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/manifest.json/permissions

[33] McKay, A.; aj.: *web_ accessible_resources*. Mozilla Developer Network, Leden 2019, [Online; navštíveno 13.03.2019].
URL https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/manifest.json/web_accessible_resources

[34] McMillan, K.; aj.: *Porting a Google Chrome extension*. Mozilla Developer Network, Listopad 2018, [Online; navštíveno 05.03.2019].
URL https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Porting_a_Google_Chrome_extension

[35] Mills, C.; aj.: *Signing and distributing your add-on*. Mozilla Developer Network, Květen 2019, [Online; navštíveno 26.03.2019].
URL
https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/Distribution

[36] Mills, C.; aj.: *StorageArea.getBytesInUse()*. Mozilla Developer Network, Březen 2019, [Online; navštíveno 15.04.2019].

URL https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/storage/StorageArea/getBytesInUse

[37] Mills, C.; aj.: *storage.local*. Mozilla Developer Network, Březen 2019, [Online; navštíveno 15.04.2019].
URL https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/storage/local

[38] Mills, C.; aj.: *storage.sync*. Mozilla Developer Network, Březen 2019, [Online; navštíveno 15.04.2019].
URL https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/storage/sync

[39] Morelli, B.: *JavaScript — Double Equals vs. Triple Equals*. Listopad 2017, [Online; navštíveno 13.04.2019].
URL https://codeburst.io/javascript-double-equals-vs-triple-equals-61d4ce5a121a

[40] Ponomarev, N.; aj.: *Using the W3C DOM Level 1 Core*. Mozilla Developer Network, Březen 2019, [Online; navštíveno 26.04.2019].
URL https://developer.mozilla.org/en-US/docs/Web/API/Document_object_model/Using_the_W3C_DOM_Level_1_Core

[41] Scarfone, K.; aj.: *Window.localStorage*. Mozilla Developer Network, Březen 2019, [Online; navštíveno 04.04.2019].
URL https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage

[42] Scholz, F.; aj.: *Content Security Policy*. Mozilla Developer Network, Březen 2019, [Online; navštíveno 16.04.2019].
URL https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Content_Security_Policy

[43] Shepherd, E.; aj.: *Introduction to the DOM*. Mozilla Developer Network, Březen 2019, [Online; navštíveno 26.04.2019].
URL https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

[44] Shepherd, E.; aj.: *options_ui*. Mozilla Developer Network, Leden 2019, [Online; navštíveno 13.03.2019].
URL https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/manifest.json/options_ui

[45] StatCounter: *Desktop Browser Market Share*. StatCounter, [Online; navštíveno 25.03.2019].
URL http://gs.statcounter.com/browser-market-share/desktop/

[46] Swisher, J.; aj.: *Content Security Policy (CSP)*. Mozilla Developer Network, Duben 2019, [Online; navštíveno 16.04.2019].
URL https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP

[47] Truong, A.; aj.: *Match patterns in extension manifests*. Mozilla Developer Network, Březen 2019, [Online; navštíveno 17.04.2019].
URL https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Match_patterns

[48] Truong, A.; aj.: *webRequest*. Mozilla Developer Network, Leden 2019, [Online; navštíveno 16.03.2019].
URL https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/webRequest

[49] Villeneuve, C.; aj.: *options_page*. Mozilla Developer Network, Říjen 2018, [Online; navštíveno 13.03.2019].
URL https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/manifest.json/options_page

[50] Villeneuve, C.; aj.: *Options page*. Mozilla Developer Network, Září 2018, [Online; navštíveno 13.03.2019].
URL https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/user_interface/Options_pages

[51] Villeneuve, C.; aj.: *Extension pages*. Mozilla Developer Network, Únor 2019, [Online; navštíveno 13.03.2019].
URL https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/user_interface/Extension_pages

[52] W3Schools: *CSS Border*. [Online; navštíveno 20.03.2019].
URL https://www.w3schools.com/css/css_border.asp

[53] W3Schools: *CSS Margin*. [Online; navštíveno 20.03.2019].
URL https://www.w3schools.com/css/css_margin.asp

[54] W3Schools: *CSS Pading*. [Online; navštíveno 20.03.2019].
URL https://www.w3schools.com/css/css_padding.asp

[55] W3Schools: *HTML script Tag*. [Online; navštíveno 02.04.2019].
URL https://www.w3schools.com/tags/tag_script.asp

[56] W3Schools: *JavaScript Const*. [Online; navštíveno 13.04.2019].
URL https://www.w3schools.com/css/css_padding.asp

[57] W3Schools: *JavaScript Let*. [Online; navštíveno 13.04.2019].
URL https://www.w3schools.com/css/css_padding.asp

[58] Wuest, A.; aj.: *Document Object Model (DOM)*. Mozilla Developer Network, Březen 2019, [Online; navštíveno 11.04.2019].
URL
https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

[59] Zartner, S.; aj.: *What is JavaScript?* Mozilla Developer Network, Březen 2018, [Online; navštíveno 05.04.2019].
URL https://developer.mozilla.org/cs/docs/Learn/JavaScript/First_steps/Co_je_JavaScript

[60] Čápka, D.: *Lekce 7 - Funkce v JavaScriptu*. [Online; navštíveno 13.04.2019]. URL https://www.w3schools.com/css/css_padding.asp

[61] Žára, O.: *JavaScript: programátorské techniky a webové technologie*. Brno: Computer Press, 2015, ISBN 978-80-251-4573-9.