České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra počítačů

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Eiselt Zbyněk

Studijní program: Otevřená informatika
Obor: Softwarové inženýrství

Název tématu: WebExtensions, bezpečnostní model a kompatibilita s XPCOM

Pokyny pro vypracování:

Seznamte se s technologií WebExtensions, určenou pro tvorbu tzv. "cross-browser" rozšíření webových prohlížečů. Nastudujte dostupná rozhraní WebExtensions a porovnejte jejich možnosti se starší množinou technologií XUL+XPCOM. V tomto srovnání akcentujte zejména otázku bezpečnostních modelů obou přístupů.

Dále aplikujte nabyté znalosti na existující vzorové rozšíření "Seznam Lištička" pro prohlížeč Firefox. Naprogramujte funkčně shodné řešení postavené na platformě WebExtensions a popište řešení situací, kdy vzájemná nekompatibilita obou rozhraní brání přímočarému technologickému přepisu. Toto rozšíření publikujte buď v "Chrome Web Store", nebo na webu "addons.mozilla.org". Zhodnoťte, došlo-li přechodem na WebExtensions ke kvalitativnímu a bezpečnostnímu vylepšení tohoto rozšíření. Nedílnou součástí nově vzniklého rozšíření bude technická dokumentace, uživatelská i programátorská (včetně dokumentace automaticky generované).

Posledním výstupem této práce nechť je 'Migration guide', obecný návod pro přechod z technologií XUL+XPCOM na WebExtensions. V návodu navrhněte postupy pro ty komponenty XPCOM, které nemají ve WebExtensions žádné korespondující API.

Seznam odborné literatury:

[1] https://developer.mozilla.org/en-US/Add-ons/WebExtensions
[2] https://developer.chrome.com/extensions
[3] https://developer.mozilla.org/en-US/docs/Mozilla/Tech/XPCOM

Vedoucí: RNDr. Ondřej Žára

Platnost zadání do konce letního semestru 2017/2018

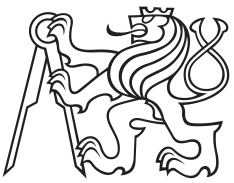
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE Fakulta elektrotechnická L.S.

prof. Dr. Michal Pěchouček, MSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 24.1.2017

Diplomová práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

WebExtensions, bezpečnostní model a kompatibilita s XPCOM

Bc. Zbyněk Eiselt

Program: Otevřená informatika
Obor: Softwarové inženýrství

Květen 2017
Vedoucí práce: RNDr. Ondřej Žára

Poděkování / Prohlášení

Rád bych vřele poděkoval vedoucímu práce RNDr. Ondřeji Žárovi za poskytnutou záštitu nad tématem práce a času věnovanému konzultacím.

Dále bych chtěl poděkovat mé rodině a Adéle Hrudové, kteří mi byli silnou oporou po celou dobu studia a nedovolili, aby všechno úsilí věnovanému studiu přišlo vniveč.

V neposlední řadě patří velký dík Jirkovi Bachelovi, Filipovi Mösnerovi, Miloslavu Mrvíkovi, Lukáši Kovačovi a zbylým členům týmu S-Browser ze Seznam.cz za jejich odborné rady v oblasti webových technologií a porozumění důležitosti, jakou pro mě diplomová práce přestavuje.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 25. 5. 2017

.....

Abstrakt / Abstract

Cílem práce je nastudovat rozhraní WebExtensions pro tvorbu tzv. „cross-browser“ rozšíření do webových prohlížečů a porovnat jejich možnosti se starší množinou technologií XUL/XPCOM. Během srovnávání bude kladen důraz na otázku bezpečnostních modelů obou přístupů.

Praktická část práce je zaměřena na vytvoření funkčně shodného řešení v platformě WebExtensions k již existujícímu rozšíření „Seznam Lištička“ postaveného na množině technologií XUL/XPCOM. Práce dále popisuje, jak postupovat v situacích, kdy vzájemná nekompatibilita obou rozhraní brání přímočarému technologickému přepisu. Součástí je i zhodnocení, došlo-li přechodem na WebExtensions ke kvalitativnímu a bezpečnostnímu vylepšení. Nedílnou součástí práce je „Migration guide“, což je obecný návod pro přechod z technologií XUL/XPCOM na WebExtensions.

Klíčová slova: diplomová závěrečná práce; rozšíření; doplňky; prohlížeč; XUL; XPCOM; WebExtensions; technologická migrace; Chrome; Firefox.

The goal of this thesis is to study the WebExtensions interface, designed for the creation of so-called „cross-browser“ extensions and compare their options with the older set of technologies XUL/XPCOM. During the comparison we will focus on the question of security modules of both approaches.

The practical part of thesis focus on creating a functionally identical solution in WebExtensions base on existing extension „Seznam Lištička“ built in XUL/XPCOM and describing how to proceed in situations where mutual incompatibility of both interfaces prevents straightforward transcription. It will include an evaluation of whether there were qualitative and security improvements by going to WebExtensions. An inseparable part of the thesis consists of „Migration guide“ - general guidance for the transition from XUL/XPCOM to WebExtensions technology.

Keywords: master thesis; extensions; add-on; browser; XUL; XPCOM; WebExtensions; technology migration; Chrome; Firefox.

Obsah /

/ Obrázky

Kapitola 1
Úvod

Doplňky, častěji překládané jako rozšíření, rozšiřují webový prohlížeč o další funkce, upravují webové stránky a integrují do prohlížeče další služby, ke kterým přistupujeme přes webové rozhraní. Rozšíření jsou psána za pomoci standardních webových technologií – HTML, CSS a především JavaScriptu, který je obohacený o dedikované JavaScriptové API poskytované prohlížečem.

Proč bychom ale měli používat rozšíření v prohlížeči? Můžeme je chtít použít z několika různých důvodů:

- Integrace s dalšími službami, které používáme. Například *Evernote* [1] nabízí rozšíření, které umožňuje snadno připínat webové stránky a ukládat je do vašeho účtu Evernote.
- Přidání dalších funkcí do prohlížeče. Například rozšíření *JoinTabs* [2] pro Chrome vám dává tlačítko, na které můžete kliknout, a kombinovat všechny karty Chrome z více oken do jednoho okna.
- Chcete-li upravovat webové stránky tak, jak se zobrazují na vašem počítači - přidávání, odebírání nebo úpravy obsahu. Například rozšíření *Rozšíření Pricescout for Google Chrome* [3] přidává informace na online e-shopy a informuje o tom, zda je v konkurenčním obchodním řetězci k dispozici nižší cena.

Rozšíření mohou sloužit i k mnoha dalším věcem. Jsou jako každý jiný software, přestože prohlížeče do nich vkládají určité limity na to, co všechno smí dělat.

Každé rozšíření je distribuováno přes oficiální webové obchody, které spravují přímo poskytovatelé prohlížeče. Mezi největší se bezkonkurenčně řadí Chrome Web Store [4] a Add-ons for Mozilla [5], jenž společně obsahují v současnosti desítky tisíc unikátních rozšíření. Díky velké rozmanitosti není těžké najít již existující doplněk dle našich požadavků. Naprosto drtivá většina z nich je zdarma, avšak můžeme najít i takové, které za stažení vyžadují poplatek v řádech desítek korun.

V polovině roku 2011 proběhl průzkum v rámci Firefoxu [6], který měl za úkol zjistit, kolik jeho uživatelů má nainstalované alespoň jedno rozšíření. Výsledkem bylo číslo dosahující hranice 85 %. Z průzkumu zároveň vyplynulo, že průměrný uživatel Firefoxu má nainstalovaných méně než 5 doplňků. Z průzkumu vyplynuly i další zajímavé údaje: v letech 2007 až 2011 bylo staženo více než 2,5 miliardy doplňků a každý den je jich používáno v průměru na 580 miliónů.

V tom samém roce uveřejnil Google na svém blogu [7], že rozšíření v jeho prohlížeči využívá jen 33 % uživatelů. V té době bylo v Chromu dostupných zhruba 70 miliónů rozšíření, což se v porovnání v Firefoxem může zdát jako velmi málo. Stojí však za zmínku, že tyto statistiky jsou již 6 let staré a prohlížeč Chrome byl v té době při porovnání s Firefoxem relativně mladý. Chrome se navíc aktualizuje automaticky a jeho rozšíření netrpí neduhy s kompatibilitou mezi jednotlivými verzemi prohlížeče. Lze tedy předpokládat, že Chrome je v současnosti co do počtu uživatelů s alespoň jedním rozšířením leaderem na trhu.

Téma této diplomové práce vzniklo v souvislosti s poptávkou na aktualizaci stávajícího rozšíření *Seznam Lištička*. První verze tohoto rozšíření přišla v roce 2005 a to pro Internet Explorer, v té době nejpoužívanější prohlížeč na světě. O rok později vyšla i verze pro Firefox. Webové technologie v té době byly na zcela jiné úrovni, než jsou dnes a proto jejich vývoj byl náročný a značně komplikovaný. Byť byly první ohlasy na *Lištičku* převážně kladné, čas k nim byl neúprosný a jejich technologie velice rychle zastarala. Zároveň s tím bylo v listopadu 2016 ohlášeno, že Firefox v následujícím roce upustí od podpory všech starších technologií uzpůsobených k tvorbě rozšíření [8]. Tudíž již nebylo možná dále odkládat přesun na novou technologii. Bylo zapotřebí prozkoumat možnosti migrace na WebExtensions, jenž jsou nástupcem starších technologií a měly by zaručovat dlouhotrvající podporou ze strany prohlížečů.

Kapitola 2
Technologie pro vývoj rozšíření

2.1 Historie

Původní rozhraní pro přístup k interním procesům a funkcím prohlížeče bylo NPAPI. Poprvé bylo vyvinuto pro prohlížeče Netscape a počínaje rokem 1995 pro aplikaci Netscape Navigator 2.0. Prohlížeč Internet Explorer začal podporovat rozšíření od verze 5 vydané v roce 1999 [9]. Vývoj a podpora tehdejších rozšíření se značně lišily od dnešních standardů, jelikož v té době ještě nebyly webové technologie, především JavaScript, dostatečně rozvinuté. Společnost Microsoft nepřijala rozhraní NPAPI pro svůj prohlížeč a namísto toho zvolila ActiveX pro pluginy s úpravou obsahu, označované jako Browser Helper Objects. Firefox podporuje rozšíření od svého uvedení v roce 2004. Webový prohlížeč Opera Desktop podporoval rozšíření od verze 10, která spatřila světlo světa v roce 2009. Google Chrome a Safari začaly s nativní podporou rozšíření v roce 2010 a společnost Microsoft začala v březnu 2016[10] podporovat rozšíření ve značně omezeném měřítku i ve svém nejnovějším prohlížeči Edge.

Když vývojáři v minulosti vyvíjeli první rozšíření prohlížeče, byli často nuceni používat neobvyklé nebo složité programovací jazyky jako C++. To vyžadovalo mnoho práce, času a zkušeností. Vzrůstající složitost zdrojového kódu prohlížeče zároveň přinesla i vyšší riziko jeho zneužití. Kód byl někdy natolik složitý, že rozšíření často způsobovala selhání prohlížeče.

2.2 Rozdíly mezi prohlížeči: Chrome, Firefox, Internet Explorer

Syntaxe rozšíření se může lišit zcela nebo natolik, aby rozšíření fungující v jednom prohlížeči nefungovalo v jiném. Obecně vzato platí, že Firefox má základnu rozšíření a jejich technologií nejsilnější. Mnoho lidí používá tento prohlížeč právě kvůli tomu, že umožňuje vytvářet mnoho pokročilých rozšíření, která by nebyla možná v jiných. V minulosti šlo v rámci Firefoxu vytvořit například i vlastní minifikovaný prohlížeč. Firefox propůjčoval programátorovi obsáhlé množství API, pomocí nichž mohl manipulovat od obsahu webových stránek až po soubory na disku uživatele. To samozřejmě šlo ruku v ruce s velkou měrou zneužitelnosti systému, a proto se i díky tomu v posledních měsících od těchto technologií pomalu ustupuje.

Chrome má stejně jako Firefox prosperující ekosystém rozšíření. Chrome však umisťuje do svých rozšíření více limitů, aby nemohly být stejně obsáhlé (a zneužitelné) jako v prohlížeči Firefox. Zároveň s tím Chrome přináší systém oprávnění (obdobně jako tomu je u mobilních aplikacích pro Andriod instalovaných z Google Play) a umožňuje tak ještě více omezovat (či povolovat) přístupy k jednotlivým API prohlížeče.

Aplikace Internet Explorer má na rozdíl od Chromu a Firefoxu velmi malý ekosystém doplňků. Jen pramálo z nich je k dispozici a většina je v reálném prostředí zcela

nepraktická. Příkladem můžou býti ony lišty nástrojů, které byly uživateli nejčastěji instalovány přibalením k instalaci jiného softwaru. Důvodem takto malé obliby doplňků v Internet Exploreru jsou i technologie, kterými lze rozšíření do prohlížeče naimplementovat. Jeden ze způsobů může být vytvoření lokálního serveru na pozadí systému, který následně servíruje prohlížeči webové stránky. Zároveň s tím vzniká nutnost vytvářet většinu úprav vzhledu a funkcí samotného prohlížeče v programovacím jazyce C++, je ve světě moderních webových technologií značně nepraktická.

Rozšíření pro Edge, Safari a Operu existují taktéž, ovšem jejich ekosystémy jsou nesrovnatelně menší, než jaké jsou u Firefoxu nebo Chromu.

2.3 Nejrozšířenější technologie

V současné době[1] existuje celá řada technologií, které jsou uzpůsobené k tvorbě rozšíření do prohlížečů, ale pouze jedna z nich je podporována napříč předními prohlížeči - WebExtensions. Mezi ostatními technologiemi vynikají především XUL/XPCOM a Add-on SDK, které byly vyvinuty firmou Mozilla a využívají je výhradně její produkty.

Pokud tedy budeme v textu hovořit o technologiích XUL/XPCOM nebo Add-on SDK, budeme brát v potaz výhradně rozhraní Firefoxu.

2.3.1 XUL/XPCOM

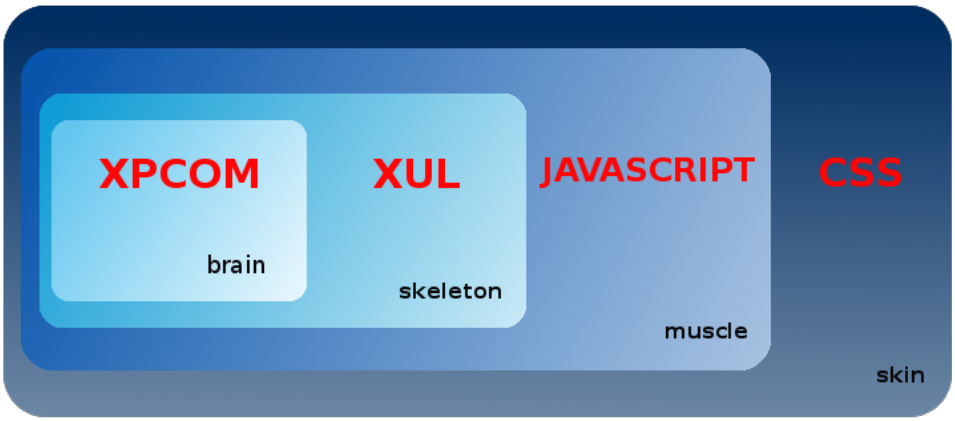
Firefox a další aplikace od Mozilly lze považovat za složené ze dvou různých částí: z vrstvy uživatelského rozhraní, která je pro každý projekt odlišná, a ze společné platformy, nad níž je vytvořena vrstva s rozhraním. Uživatelské rozhraní je postaveno na technologii známé jako XUL. Platforma je pak známá jako XULRunner.

XUL je jednou z několika technologií používaných pro tvorbu produktů a rozšíření založených na platformě Mozilla. Je to jen jedna z částí vývoje, ale vzhledem k tomu, že je prakticky exkluzivní pro Mozillu, má/měla tendenci být používána k vývoji veškerých produktů souvisejících s Mozillou. Někdy se můžeme setkat s termíny jako „aplikace XUL“ a „XUL rozšíření/doplněk“, ale zřídka se budou vztahovat na projekty, které jsou výhradně postaveny pomocí XUL. Obvykle to znamená, že projekty byly postaveny pomocí technologií Mozilla a můžeme se v daném projektu setkat i s JavaScriptem nebo CSS.

XPCOM je multiplatformní komponentový model používaný v aplikacích založených na platformě Mozilla. Je podobný architektuře CORBA či COM od Microsoftu. Zahrnuje například práci se soubory, správu paměti či komponent produktu.

Vztah mezi XUL, XPCOM a dalšími technologiemi používanými v rozšířeních tohoto typu podrobněji popíši v kapitole 3.1. Prozatím si vystačíme s jednoduchým nákresem.

[1] první polovina roku 2017



Obrázek 2.1. Vztah mezi technologiemi XUL, XPCOM, JS a CSS. Zdroj: Mozilla Developer Network

2.3.2 Jetpack

Jetpack je projekt, který vyvíjí nástroje a frameworky pro usnadnění tvorby doplňků pro Firefox. Projekt vytvořil Add-on SDK, sadu poskytující **API rozhraní jazyka JavaScript**, které zjednodušuje mnoho běžných úloh při vývoji, což značně urychluje proces vývoje, testování a sestavování rozšíření. Během vývoje se využívá standardních webových technologií, jako jsou HTML nebo CSS.

Výhody Add-on SDK:

- Pokud nejsou doplňky pečlivě navrženy, mohou posléze prohlížeč otevřít a infikovat jej škodlivým obsahem, který pak může odesílat soukromá data uživatele bez jeho vědomí. Přestože je možné pomocí SDK vytvořit takovéto doplňky, není to již tak snadné jako v případě technologie XUL/XPCOM a škody, které může napáchat, jsou obvykle značně omezené.
- Doplňky vytvořené pomocí sady SDK lze nainstalovat **bez nutnosti restartovat prohlížeč**.
- Rozhraní API bylo navrženo tak, aby bylo **kompatibilní s novou architekturou podporující více procesů** (kódové označení Electrolysis) naplánované pro nejnovější Firefox.
- Komponenty uživatelského rozhraní, které jsou k dispozici v sadě SDK, jsou navrženy tak, aby vyhovovaly směrnicím použitelnosti pro prohlížeč Firefox a umožnily uživatelům lepší a konzistentnější zážitek.

Výhody doplňků založených na XUL:

- XUL overlay nabízejí spoustu možností pro vytváření uživatelského rozhraní a integraci do prohlížeče. API pro rozhraní SDK, má svoje uživatelské rozhraní mnohem omezenější.
- Tradiční doplňky mají přístup k ohromnému množství funkcí Firefoxu přes XPCOM. Podporované API rozhraní SDK vystavují poměrně malou sadu této funkce za cenu zvýšené bezpečnosti a omezeným možnostem zneužití.

2.3.3 WebExtensions

WebExtension je zatím nejmodernější dostupná technologie pro vývoj doplňků (rozšíření). Jsou napsány pomocí standardních webových technologií - JavaScript, HTML a CSS - a některé specializované API jazyka JavaScript. Kromě jiného mohou doplňky přidat do prohlížeče nové funkce nebo změnit vzhled.

Byla vytvořena v dílnách Googlu začátkem tohoto desetiletí jako nástupce stávajících platforem, které byly implementovány pouze v produktech od Mozilly a umožňovaly až příliš velkou míru možností k zneužití. WebExtensions jsou v současné době podporovány všemi předními prohlížeči, jako jsou Google Chrome, Mozilla Firefox, Microsoft Edge nebo Opera.

Chrome a Opera mají největší počet přístupných API, jelikož oba prohlížeče jsou postaveny na stejném renderovacím jádru Blink a zároveň vývoj pro Chrome probíhá, na rozdíl od Firefoxu a Edge, již několik let. Doplňky psané pro Chrome ve většině případů běží i ve Firefoxu a Edgi jen s minimem provedených úprav. WebExtensions jsou zároveň plně kompatibilní s nejnovějším Firefoxem podporující multiprocessing.

Kompletní porovnání platforem XUL/XPCOM a WebExtensions uvádím v kapitole č. 3.

Kapitola 3
Anatomie XUL/XPCOM a WebExtensions

3.1 XUL/XPCOM

Sekce 3.1 se bude vztahovat pouze na doplňky vytvořené pro prohlížeč Firefox, jelikož Chrome, Opera, Edge a další prohlížeče nepodporují v dnešní době již zastaralý přístup XUL/XPCOM.

3.1.1 Soubor install.rdf

Formát souboru `install.rdf` vychází z technologie XML. RDF bývalo centrálním mechanismem pro ukládání dat ve Firefoxu, nyní je ale nahrazen jednodušším databázovým systémem SQLite. Nyní se podívejme na důležité části souboru.

```
<em:id>{ea614400-e918-4741-9a97-7a972ff7c30b}</em:id>
```

Jedná se o jedinečný identifikátor rozšíření. Firefox potřebuje rozlišovat jednotlivá rozšíření od těch ostatních, takže je třeba mít pro každé rozšíření jedinečné ID. Existují dva přijaté standardy pro ID rozšíření. Jeden formát je podobný e-mailu, který by měl být ve formátu `<nazev-projektu>@<domena>`. Další standardní praxe je použít generovaný řetězec UUID, u kterého je velmi nepravděpodobné, že bude duplikován. Důležité je dodržet jedinečnost ID napříč všemi existujícími rozšířeními. Jakou variantu ID nakonec použijeme je jen na nás.

```
<em:name>Seznam Lištička</em:name>
<em:description>Seznam lištička pro Firefox a další prohlížeče založené
    na jádru Mozilla Gecko.</em:description>
<em:version>3.3.3</em:version>
<em:creator>Seznam.cz, a.s.</em:creator>
<em:homepageURL>https://software.seznam.cz/listicka?
    browser=geck3</em:homepageURL>
```

Toto jsou údaje zobrazené před a po instalaci rozšíření, které můžete vidět ve Správci doplňků. Existuje ještě mnoho dalších značek, které lze přidat pro přispěvatele [11]. Úplná specifikace souboru `install.rdf` a všechny její podrobnosti můžeme nalézt na stránkách MDN[1].

[1] `https://developer.mozilla.org/en-US/Add-ons/Install_Manifests`

Vzhledem k tomu, že rozšíření lze přeložit do několika jazyků, je často nutné překládat popis rozšíření nebo dokonce jeho název. Lokalizovaný popis a název lze přidat pomocí následujícího kódu:

```
<em:localized>
  <Description>
    <em:locale>en-GB</em:locale>
    <em:name>Seznam Lištička</em:name>
    <em:description>Seznam lištička for Firefox and other browsers
        running on Mozilla Gecko core.</em:description>
  </Description>
</em:localized>
```

Řetězec en-GB označuje, že se jedná o anglickou lokalizaci pro Velkou Británii. Můžeme přidat tolik lokalizovaných sekcí, kolik je potřeba.

```
<em:type>2</em:type>
```

To určuje, že nainstalovaný doplněk je rozšířením. Dalšími možnostmi jsou:

- skiny (4),
- kontrola pravopisu (64),
- experimenty s telemetrií (128)
- a další

```
<em:targetApplication>
  <Description>
    <em:id>{ec8030f7-c20a-464f-9b0e-13a3a9e97384}</em:id>
    <em:minVersion>24.0</em:minVersion>
    <em:maxVersion>55.*</em:maxVersion>
  </Description>
</em:targetApplication>
```

Tento uzel určuje cílovou aplikaci a verzi rozšíření. Zde konkrétně se jedná o aplikaci Firefox, od verze 24 až po verzi 55. Jiné aplikace společnosti Mozilla, jako jsou Thunderbird a SeaMonkey, mají své vlastní UUID označení. Můžeme mít rozšíření, které funguje ve více aplikacích a verzích. Například pokud vytvoříme rozšíření do Firefoxu, nemuselo by správně fungovat třeba v SeaMonkey, byť má velmi podobné funkce a uživatelské rozhraní.

3.1.2 Soubor chrome.manifest

`chrome.manifest` je sada prvků uživatelského rozhraní okna aplikace, které se nacházejí mimo oblast obsahu okna[1]. Panely nástrojů, pruhy nabídek a panely s nadpisy oken jsou příklady prvků, které jsou obvykle součástí chromu [12].

Nutno podotknout, že když mluvím o „chromu“ ve Firefoxu, nemám tím na mysli prohlížeč Chrome od Googlu ani nic s ním spojeného. Nevím, jak někdo přišel na pojem „chrome“, ale byl pravděpodobně vizuální analogií k použití kovového chromu na velkých amerických automobilech během padesátých let. Karoserie auta byla v té době obklopena lesklými chromovanými nárazníky, žebrování chladičů a podobně.

[1] Webview, prostor pro vykreslování webových stránek

Podobně tomu je tak i ve většině moderních grafických uživatelských rozhraní, kde „chrome“[1] žije po okrajích obrazovky, obklopující prostřední oblast, která je věnována uživatelským datům. Všechna okna prohlížeče Firefox mohou být tedy chápána jako dvě separátní části:

- chrome
- obsah, který zobrazují webové stránky v panelu Firefoxu

Chrome se skládá ze tří částí:

- **content**
- **locale**
- **skin**

Všechny tři jsou nutné pro většinu rozšíření. Otevřeme-li soubor chrome.manifest, můžeme vidět:

```
content   foxcub              chrome/content/
skin      foxcub  classic/1.0 chrome/skin/
locale    foxcub  cs-CZ       chrome/locale/cs-CZ/
```

Soubor `chrome.manifest` informuje Firefox o tom, kde hledat chrome soubory. Text je odsazen tak, aby vypadal jako tabulka, ale to není nutné. Analyzátor ignoruje opakované mezery.

První slovo v řádku říká Firefoxu, o čem je právě to, co se deklaruje (content, skin, locale nebo jiné, které jsou uvedeny později). Druhý sloupec je název balíčku.

Balíčky pro skin a locale mají třetí hodnotu, která určuje, jaká lokalizace nebo o jaký skin se rozšiřují. Může existovat více položek lokalizací a skinů, které se vztahují k různým motivům nebo místům. Nejběžnějším případem je, že má jeden globální skin, `classic/1.0`. Můžeme mít položky, které jsou specifické pro OS. To je důležité, protože vzhled prohlížeče je pro každý operační systém velmi odlišný.

Existují některé další možnosti, které mohou být zahrnuty do položek souboru `chrome.manifest`, jako je například položka `resource` či `override`[2].

3.1.3 Chrome

Jak již bylo zmíněno v sekci 3.1.2, `chrome` se skládá ze tří částí. **Content** je z všech nejdůležitější, obsahuje uživatelské rozhraní (XUL) a soubory skriptů (JS). Sekce **skin** má soubory, které definují většinu vzhledu uživatelského rozhraní (pomocí CSS a obrázků, stejně tak jako webové stránky). A v sekci **locale** nastavujeme jazyk, kde se ukládá veškerý text použitý v rozšíření (soubor ve formátu DTD). Toto dělení umožňuje dalším vývojářům vytvářet náhradní skiny a překlady, a vytvářet tak lokalizace v různých jazycích, a to vše bez nutnosti měnit původní rozšíření nebo jeho kód. To dává rozšířením Firefoxu velkou flexibilitu.

Soubory Chrome jsou přístupné prostřednictvím protokolu `chrome`. To vypadá takto:

```
chrome://package/section/path/to/file
```

[1] Reference na *chrome* Firefoxu. V jiných aplikacích okraje uživatelského rozhraní nemusí být označovány jako chrome.

[2] Kompletní dokumentace se na stránce
`https://developer.mozilla.org/en-US/docs/Chrome_Registration`

Čili pokud chci získat například přístup k souboru `browserOverlay.xul`, chrome URI bude *chrome://foxcub/content/settings/browserOverlay.xul*. Máme-li v obsahu příliš mnoho souborů a chcete je uspořádat v podadresářích, nemusíte v chrome.manifest nic změnit, vše, co potřebujete, je přidání správné cesty k obsahu v URI.

Soubory *skin* a *locale* fungují stejným způsobem a nemusíte specifikovat názvy skinů nebo názvy národních prostředí.

3.1.4 Content

V adresáři *content* jsou 2 hlavní typy souborů:

- GUI soubory: XUL/HTML a CSS
- JS soubory

Podíváme se nejprve na GUI soubory, konkrétně typu XUL.

Soubory XUL vycházejí z technologie XML, které definují prvky uživatelského rozhraní v rozšířeních pro Firefox. XUL byl inspirován HTML, můžeme si tedy mezi nimi všimnout mnoha podobností. Nicméně, XUL je jakási nadstavba nad HTML, protože se poučil z mnoha chyb, které se vyskytly během vývoje HTML. XUL umožňuje vytvářet bohatší a interaktivnější rozhraní než ty, které můžete vytvářet pomocí HTML.

Soubory XUL obvykle definují jednu ze dvou věcí: okna nebo překryvy.

```
overlay   chrome://browser/content/browser.xul
          chrome://foxcub/content/foxcubOverlay.xul
```

Tímto řádkem Firefox ví, že potřebuje převzít obsah `foxcubOverlay.xul` a překrýt ho v hlavním okně prohlížeče, `browser.xul`. Můžete deklarovat překryvy pro všechna okna nebo dialogové okno ve Firefoxu, ale překrývání hlavního okna prohlížeče je nejběžnějším případem.

Nyní se podíváme na obsah souboru XUL.

```
<?xml version="1.0" encoding="UTF-8"?>
<overlay id="foxcubOverlay"
    xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
    xmlns:html="http://www.w3.org/1999/xhtml">
```

Jiné dokumenty XUL používají okno nebo dialogový tag. Prvek má jedinečné ID, které by měla mít většina prvků XULu. Druhým atributem je jmenný prostor, který je definován v kořenovém elementu XUL. Tento uzel a všechny uzly jemu podřízené jsou taktéž typu XUL. Stačí pak pouze změnit definici názvů uzlů, když smícháme různé typy obsahu ve stejném dokumentu, například XUL s HTML nebo SVG.

```
<script type="application/x-javascript"
        src="chrome://foxcub/content/main.js" />
```

Stejně jako v jazyce HTML obsahuje i soubor XUL JavaScript. V dokumentu XUL můžeme mít tolik skriptových souborů, kolik potřebujeme.

```
<toolbarpalette id="BrowserToolbarPalette">
    <toolbarbutton
        id="foxcub_translator_status"
        tooltiptext="Přejít na Seznam.cz"
        class="toolbarbutton-1"
```

```
        image="chrome://foxcub/skin/seznam-s.png"
        label="Seznam"
        onclick="FoxcubChrome.FoxcubOverlay.prototype.butoon(event);">
    </toolbarbutton>
</toolbarpalette>
```

Jedním z možných funkcí obsažených v těle souboru XUL je přidání tlačítka do panelu nástrojů prohlížeče. Každý GUI prvek ve Firefoxu má svou vlastní paletu označení, v tomto případě `toolbarpalette`, která má další prvky jako jsou `toolbar`, `toolbargrippy`, `toolbarseparator` a další[1]. Každý takový prvek musí mít opět jedinečné ID. V případě tlačítka specifikujeme i popisek při najetí myší, obrázek ikonky nebo co se stane při kliknutí. Každý `event listener` definuje speciální objekt s názvem `event`, která je obvykle předávaná jako argument funkce.

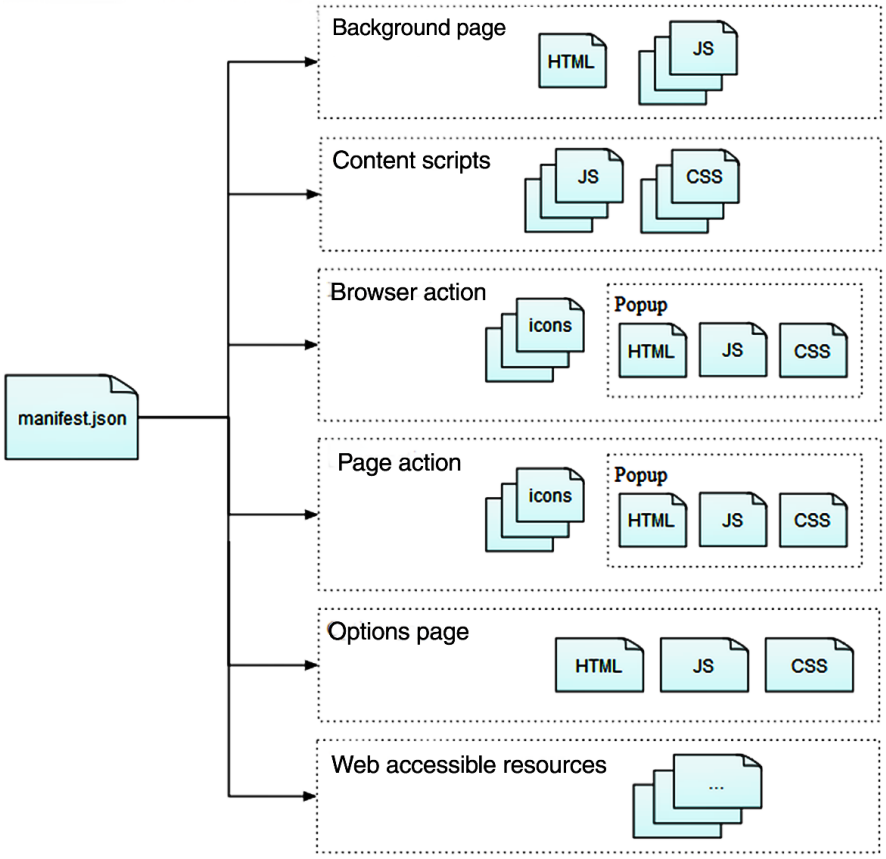
Každý XUL soubor může obsahovat více definic naráz, všechny ale musí ovládat stejnou logiku v rámci prohlížeče (překryvy nebo práce s okny).

3.2 WebExtensions

WebExtension se skládá z kolekce souborů, které jsou zabaleny pro distribuci a instalaci. Každý WebExtension musí obsahovat soubor s názvem `manifest.json` [13]. Obsahuje základní metadata o rozšíření, jako je název, verze a oprávnění, která vyžaduje. Dále může manifest obsahovat odkazy na několik dalších typů souborů:

- **Background page**[2]: Implementuje dlouhodobou logiku na pozadí.
- **Content scripts**[3]: Interakce s obsahem webových stránek (není ekvivalentem JavaScriptového prvku `<script>` v HTML stránky).
- **Browser action file**: Přidání tlačítek na panel nástrojů.
- **Page action file**: Přidání tlačítka do adresního řádku.
- **Options page**: Definuje uživatelské rozhraní pro zobrazení a změnu nastavení doplňku.
- **Web-accessible resources**: Zpřístupní přibalený obsah doplňku webovým stránkám a content scripts.

[1] `https://developer.mozilla.org/en-US/docs/Mozilla/Tech/XUL/`
[2] Stránka na pozadí
[3] Obsahové skripty



Obrázek 3.1. Členění komponent WebExtensions. Zdroj: Mozilla Developer Network

WebExtension jsou na rozdíl od doplňků z rodiny XUL/XPCOM striktně rozdělené na soubory, které můžou manipulovat s obsahem webových stránek a na soubory běžící na pozadí. Pokud bychom tedy chtěli získat text z webové stránky, následně ho zpracovat a odeslat na náš server, měli bychom jej správně získat v content script souborech, odtud jej poslat do background script, kde bychom provedli odeslání na náš server. Tato logika nám zajistí, že okno s webovou stránkou, na které se zrovna uživatel pohybuje, nezamrzne z důvodu, že by naše rozšíření komplikovaně zpracovávalo získaný text z HTML a posílalo jej na server.

3.2.1 Manifest

Soubor manifest.json je soubor ve formátu JSON a je jediným souborem, který musí obsahovat každé rozšíření WebExtension.

Pomocí příkazu manifest.json se zadávají základní metadata týkající se rozšíření, například název a verze, a také může specifikovat aspekty funkcí rozšíření, jako jsou skripty na pozadí (`background scripts`), obsahové skripty (`content scripts`) a akce prohlížeče (`browser actions`).

Příklady podporovaných klíčů manifestu:

- name
- version
- background
- permissions
- a další

Mezi povinné klíče se řadí `manifest_version`, `version` a `name`. Pokud je adresář `locales` v rozšíření přítomen, musí být přítomen i `default_locale`. `Applications` nejsou v prohlížeči Google Chrome podporovány a ve Firefoxu jsou povinné před Firefoxem verze 48.

Jedním z nejzajímavějších klíčů WebExtensions v porovnání s XUL/XPCOM je klíč **permissions**. Tento klíč nám umožňuje přistupovat k výhradním API prohlížeče. Více k oprávnění uvedu později v kapitole 4.1.

3.2.2 Background scripts

WebExtensions často potřebují udržovat dlouhodobý stav nebo provádět dlouhodobé operace nezávisle na životnosti určité webové stránky nebo okna prohlížeče. To je to, k čemu jsou background scripts určené.

Skripty na pozadí jsou načteny, jakmile je rozšíření nahráno a zůstávají načteny, dokud není rozšíření zakázáno nebo odinstalováno. Ve skriptu můžeme použít libovolné rozhraní API WebExtension, pokud jsme je v manifestu zpřístupnili.

Přidání background scriptu v `manifest.json` vypadá následovně:

```
"background": {
  "scripts": ["background-script.js"]
}
```

Můžeme přidat i více skriptů najednou. Všechny skripty se pak spustí ve stejném kontextu, stejně tak jako několik skriptů, které jsou načteny do jedné webové stránky.

3.2.3 Content scripts

Content scripts (nebo-li obsahové skripty) jsou součástí rozšíření a běží v kontextu konkrétní webové stránky (na rozdíl od skriptů na pozadí nebo skriptů, které jsou součástí samotné webové stránky a jsou načteny pomocí elementu `<script>`).

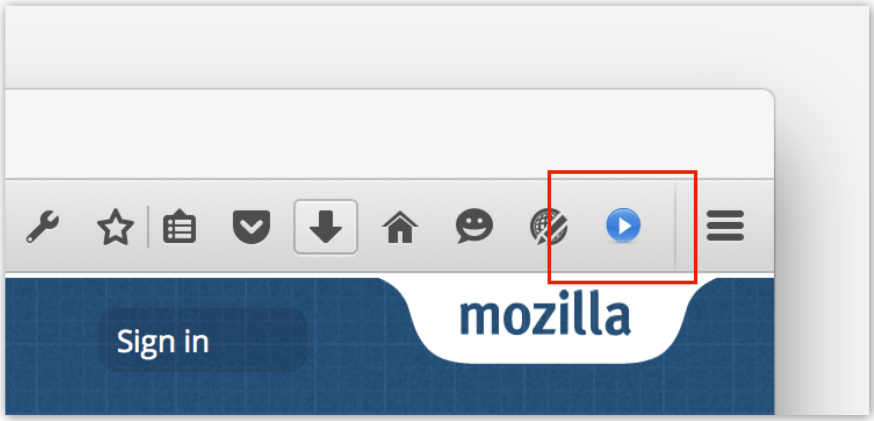
- **Skripty na pozadí** mohou přistupovat ke všem rozhraním WebExtension API, ale nemohou přímo přistupovat k obsahu webových stránek.
- **Obsahové skripty** mohou přistupovat pouze k malé podmnožině WebExtension API, ale mohou komunikovat se skriptem v pozadí pomocí systému zasílání zpráv a tím nepřímo přistupovat ke všem rozhraním WebExtension API.

Obsahový skript se může načíst do webové stránky jedním ze dvou způsobů [14]:

- *Deklarativně*: pomocí klíče `content_scripts` v manifestu můžeme požádat prohlížeč o načtení obsahového skriptu vždy, když prohlížeč načte stránku, jejíž URL odpovídá danému vzoru.
- *Programově*: pomocí rozhraní API `tabs.executeScript()` můžeme vkládat obsahový skript do konkrétního panelu, kdykoli budeme chtít: například v reakci na akci uživatele.

3.2.4 Browser actions

Browser actions [15] je označení pro tlačítko, které můžeme implementovat do panelu nástrojů prohlížeče. Uživatelé můžou na tlačítko kliknout a interagovat tak s naším doplňkem.



Obrázek 3.2. Demonstrace Browser action. Zdroj: Mozilla Developer Network

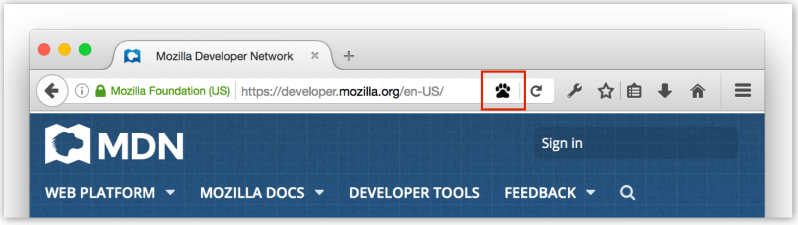
Máme možnost nastavit volitelné vyskakovací okénko (dále jen popup), které je stylizováno pomocí HTML, CSS a JavaScriptu. Pokud popup nadefinujeme, událost není předána dále do rozšíření, ale namísto toho je odchycena a je vykreslen nový popup. Uživatel následně může interagovat s popupem. Ten se automaticky zavře, jakmile uživatel přesune focus mimo něj. Popup lze vyvolat pouze událostí vyvolanou přímo uživatelem, programově toho docílit nemůže.

3.2.5 Page actions

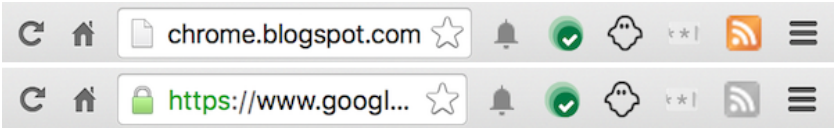
Page action jsou velice podobné browser actions, liší se pouze v:

- **Browser actions** jsou zobrazeny po celou dobu běhu prohlížeče a všechny jejich akce jsou vždy přístupné.
- **Page actions** jsou takové akce, které dávají smysl pouze v určitých okamžicích, tzn. když jsou načtené předem specifikované webové stránky. Zároveň jsou zobrazeny jen v momentě, když je daná karta (tab) otevřená.

Pro lepší pochopení, uživateli jsou page actions pozičně/graficky svázány s vybranou webovou stránkou, ať už v případě Firefoxu 3.3 za pomoci umístění ikonky do adresního řádku prohlížeče nebo v případě Chromu 3.4 zašednutím v případě, že page action není specifikovaný pro aktivní tab.



Obrázek 3.3. Demonstrace Browser action ve Firefoxu. Zdroj: Mozilla Developer Network



Obrázek 3.4. Demonstrace Browser action v Chromu. Zdroj: Chrome Developers Site

3.2.6 Options page

Options page (stránka s nastavením) umožňuje definovat předvolby pro WebExtension, které mohou uživatelé měnit. Způsob, jakým uživatelé přistupují k této stránce, a způsob, jakým je integrován do uživatelského rozhraní prohlížeče, se liší prohlížeč od prohlížeče.

3.2.7 Web accessible resources

Zdroje dostupné na webu jsou takové zdroje, jako například obrázky, HTML, CSS a JavaScript soubory, které využíváme v rámci rozšíření, ale nechceme je přibalovat do rozšíření. Na zdroje, které jsou zpřístupněny na webu, se lze pak odkazovat pomocí speciálního URI schématu.

Například pokud obsahový skript chce vložit některé obrázky do webových stránek, můžeme je zpřístupnit na webu a pak obsahovému skriptu stačí vytvořit img tag, který odkazuje na obrázky uložené mimo zdrojový adresář rozšíření.

Kapitola 4
Bezpečnost ve WebExtensions

V roce 2012 bezpečnostní odborník Zoltan Balazs *„vyvinul malware se vzdáleným ovládáním, který funguje jako rozšíření prohlížeče a je schopen modifikovat webové stránky, stahovat a spouštět soubory na disku, odcizovat účty, obcházet dvoufaktorové bezpečnostní prvky vynucené některými webovými stránkami a mnohem více.“* [16] V květnu 2013 společnost Microsoft odhalila v Brazílii rozšíření pro prohlížeče Chrome a Firefox, které se pokoušelo zneužívat profily Facebooku [17].

V roce 2015 skupina odborníků ze společnosti Google publikovala práci na téma boje proti škodlivým rozšířením. V této práci odhalují míru úsilí zločinců o zneužití webového obchodu Chrome jako platformy pro šíření škodlivých rozšíření. Hodnocení pokrývá zhruba 100 000 unikátních rozšíření odeslaných do Chrome Store po dobu tří let od ledna 2012 do začátku roku 2015. Jako výsledek analýzy vzešlo, že téměř jedno z deseti rozšíření bylo označeno jako škodlivé. Autoři též uvádí, že takto vysoká pravděpodobnost infikování počítače může být způsobena přesunem škodlivého softwaru do oficiálních obchodů poskytovaných prohlížeči Chrome, Firefox, iOS a Android s vidinou toho, že cílový uživatel bude slepě věřit neškodnosti softwaru poskytovaným na stránkách velkých společností [18].

Rozšíření prohlížeče totiž mají přístup k většině činností, které prohlížeč provádí, a mohou dělat věci, jako je například vkládání reklam do webových stránek nebo vytváření HTTP požadavků na pozadí na servery třetích stran. Zatímco webové stránky jsou omezeny bezpečnostním modelem webového prohlížeče (zejména *same-origin policy*), rozšíření nejsou. V důsledku toho může škodlivé rozšíření prohlížeče podniknout kroky proti zájmům uživatele, který jej nainstaloval. Taková rozšíření prohlížeče jsou formou malwaru. Některé stahování softwaru přicházejí s nežádoucími balíčkovými programy, které instalují rozšíření prohlížeče bez znalosti uživatele, přičemž je pro uživatele těžké odinstalovat rozšíření.

4.1 Oprávnění v manifestu

V kapitole 3.2.1 jsem uvedl, že soubor manifest.json specifikuje základní metadata rozšíření, jako jsou například název a popis nebo může specifikovat i tzv. druhotné údaje jako jsou background a content scripts nebo právě oprávnění[1].

Použitím klíče **permissions** dostává rozšíření přístup ke specifikovaným API prohlížeče, jako jsou například persistentní úložiště, *notifikace*, *webNavigation*, *webRequest* a další. O těchto oprávněních je uživatel informován při instalaci, které u starších doplňků XUL/XPCOM nebylo zobrazováno. Běžný uživatel tak neměl žádné ponětí o tom, k čemu všemu má instalovaný doplněk přístup.

[1] V `manifest.json` a přecházející kapitole uváděno jako permissions

Klíč může obsahovat tři druhy oprávnění:

- Oprávnění hostitele
- Oprávnění rozhraní API
- Oprávnění activeTab

4.1.1 Host permissions

Povolení hostitele (host permissions) jsou určena jako vzory pro shodu adres a každý vzorec identifikuje skupinu adres URL, pro které rozšíření požaduje další oprávnění. Zvláštní výhody zahrnují:

- XHR přístup k těmto zdrojům.
- Schopnost programové vkládání skriptů (pomocí `tabs.executeScript()`) na stránky, které jsou z těchto zdrojů dodávány.
- Možnost přístupu k souborům cookie pro daného hostitele pomocí Cookie API, pokud je také zahrnuto oprávnění API cookies.
- Možnost přijímat události z rozhraní API webRequest pro specifikované hostitele.

4.1.2 Funkce executeScript()

`tabs.executeScript()` vloží JavaScript kód do webové stránky.

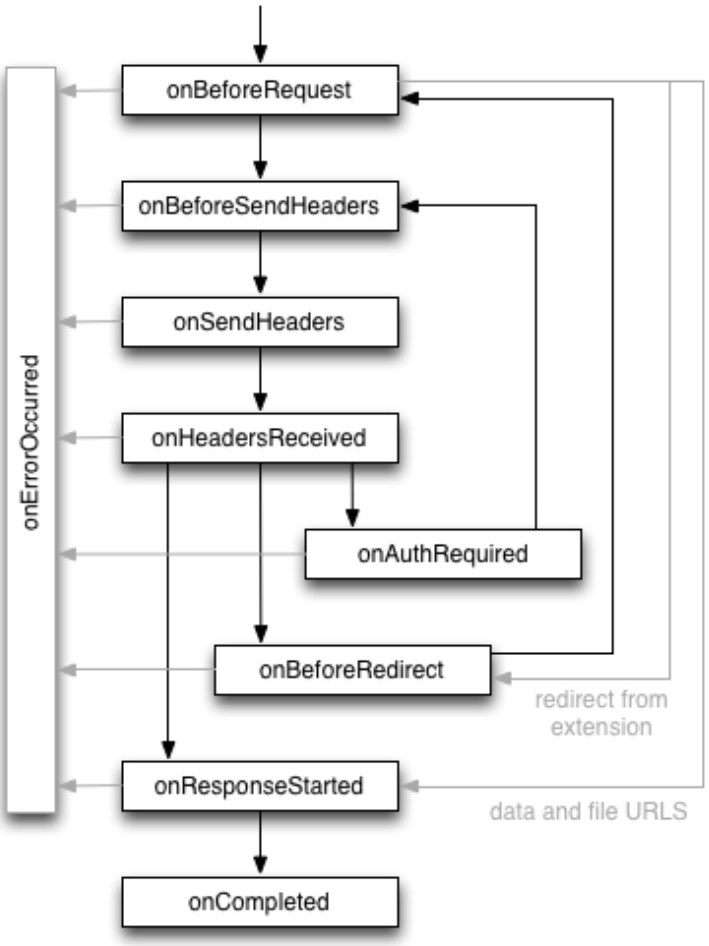
Kód můžeme odesílat pouze na stránky, jejichž adresu URL lze vyjádřit jako shodnou nebo podmnožinou cesty z definované kolekce povolených adres. To znamená, že jeho schéma musí být `http`, `https`, `file` nebo `ftp`. Pokud kód nelze vložit do žádných stránek prohlížeče, například: debugging, about:addons nebo na stránce, která se otevře při otevření nové prázdné karty.

Je vhodné kontrolovat všechny URL, ke kterým má rozšíření přístup, abychom tak zamezili případnému úniku osobních dat, případně se takovým stránkám vyhnuli.

4.1.3 WebRequest API

Zářným příkladem může být oprávnění *webRequest* [19], které má za úkol sledovat a analyzovat provoz a zachycovat, blokovat nebo upravovat požadavky za běhu. Pro plnou kontrolu nad požadavky potřebuje ještě další oprávnění - *webRequestBlocking* a seznam povolených domén, u kterých však lze povolit všechny typy URI.

WebRequest API definuje sadu událostí, které se řídí životním cyklem webové žádosti a některé synchronní události umožňuje zachytit, zablokovat nebo upravit. Životní cyklus požadavku pro úspěšné vykonání, viz obrázek 4.1.



Obrázek 4.1. Životní cyklus webového požadavku. Zdroj: Chrome Developers Site

Bezpochyby nejzajímavější událostí je *onBeforeSendHeaders*. Účelem této události je umožnit rozšířením přidávat, upravovat a odebírat hlavičky požadavků. Událost *onBeforeSendHeaders* je předána všem účastníkům komunikace, takže se mohou pokoušet upravit různé požadavky. Tuto událost lze použít i ke zrušení požadavku.

Můžeme si povšimnout, že API žádosti představuje abstrakci zásobník sítě pro rozšíření. Vnitřně lze tedy jednu žádost rozdělit na několik separátních HTTP požadavků (například pro načtení jednotlivých rozsahů bajtů z velkého souboru) nebo může být zpracováno síťovým zásobníkem bez komunikace se sítí. Z tohoto důvodu API neposkytuje konečné hlavičky HTTP, které jsou odesílány do sítě. Proto jsou například všechny hlavičky, které se vztahují k ukládání do mezipaměti nebo k autorizaci pro rozšíření nepřístupné:

- `Authorization`
- `Cache-Control`
- `Connection`
- `Proxy-Authorization`
- `Transfer-Encoding`
- a další

4.2 Content Security Policy

Za účelem zmírnění velké škály případných problémů s vytvářením skriptů mezi počítači, Webextensions obsahují koncept zásad zabezpečení obsahu (CSP) [20]. To zavádí některé poměrně přísné zásady, které zajistí, že rozšíření budou ve výchozím nastavení bezpečnější a budou poskytovat možnost vytvářet a prosazovat pravidla, která upravují typ obsahu, který lze načíst a spouštět pomocí rozšíření.

CSP zpravidla funguje jako mechanismus černých/bílých entit pro zdroje načítané nebo provedené rozšířením. Tyto zásady pak poskytují zabezpečení nad rámec oprávnění hostitele, které rozšíření požaduje. Je to však pouze dodatečná vrstva ochrany, nikoliv její komplexní náhrada.

Na webových stránkách je takové pravidlo definováno pomocí hlavičky nebo meta elementu HTTP, což ale není vhodné pro rozšíření. Místo toho existuje pravidlo zadefinované přímo v manifestu s který pak ve výchozím formátu vypadá následovně:

```
"content_security_policy": "script-src 'self'; object-src 'self'"
```

Tento příznak v manifestu zapříčiní, že:

- Můžeme načíst pouze zdroje `<script>` a `<object>`, které jsou distribuovány spolu s doplňkem.
- Doplněk nemůže spustit řetězce jako JavaScript (funkce `eval()` je zakázána).
- Řádkový JavaScript není spouštěn.

4.2.1 Umístění skriptů a objektů

Ve výchozím nastavení CSP můžete pouze načíst zdroje `<script>` a `<object>`, které jsou pro doplněk lokální. Pokud uvážíme například tento řádek kódu v WebExtension:

```
<script src="https://code.jquery.com/jquery-2.2.4.js"></script>
```

Rozšíření nebude schopno zdroj načíst a požadovaný skript bude chybět. Existují dvě hlavní řešení:

- Stáhnout kódy a vložit je mezi soubory doplňku.
- Použít klíč `content_security_policy` v manifestu pro umožnění vzdáleného přístupu, který je zde zapotřebí.

4.2.2 Funkce eval()

Ve výchozím nastavení CSP WebExtensions nemohou být řetězce vykonány[1] jako JavaScript. To znamená, že následující kód se opět neprovede:

```
eval("console.log('Some output');");
```

[1] Stringy

4.2.3 Inline JavaScript

Ve výchozím nastavení CSP není spuštěn vnořený JavaScript. To zakáže provádění JavaScriptu umístěného přímo ve značkách `<script>` v rámci HTML dokumentu a řádkových handlerů událostí, jako jsou například:

```
<script> console.log("foo"); </script>
<div onclick="console.log('click')"> Click on me!</div>
```

Pokud budeme dodržovat zásady správného HTML kódu, neměli bychom se s vnořeným JavaScriptem vůbec setkat ani ve světě webových stránek. Přesto se i v dnešní době setkáváme s nově vytvořenými stránkami, které těchto nešvarů stále využívají.

Kapitola 5
Migration guide

5.1 Výběr technologií pro rozšíření ve WebExtensions

Na začátku musíme vymezit, jakých technologií se budeme během vývoje držet, tedy kromě již zmíněných WebExtensions. Od finálního rozšíření chceme, aby bylo akceptováno jak prostředím Chromu tak i Firefoxu a byť se oba dva prohlížeče drží jednotných specifikací WebExtension, stále mezi rozšířeními pro Chrome a Firefox existují drobné rozdíly, na které musíme dopředu myslet. Zpravidla se bude jednat o specifické klíče manifestu nebo volání a implementace JavaScriptového API prohlížeče.

5.1.1 HTML5 a CSS3

HTML je značkovací jazyk používaný pro strukturování a prezentaci obsahu na webových stránkách.

Pátá řada HTML byla publikována v říjnu 2014 a přispěla ke zlepšení jazyka s podporou nejnovějších multimediálních aplikací, přičemž je stále snadno čitelná jak lidmi tak počítači a zařízeními, jako je například webový prohlížeč, různé typy parserů atd. HTML5 je zpětně kompatibilní tak, aby dokázal zpracovat a zobrazit nejen HTML 4, ale také XHTML 1 a DOM Level 2 [21].

K dispozici je mnoho nových syntaktických funkcí. Pro zpracování multimediálního a grafického obsahu byly přidány nové prvky `<video>`, `<audio>` a `<canvas>`, zároveň s tím i podpora pro škálovatelnou vektorovou grafiku (SVG) a MathML pro matematické vzorce. Pro sémantické obohacení obsahu dokumentu byly přidány nové prvky, jako jsou `<main>`, `<section>`, `<article>`, `<header>`, `<footer>`, a další. S novými prvky musely být některé zastaralé prvky odstraněny, a prvky jako `<a>`, `<cite>` a `<menu>` byly změněny, odebrány nebo plně standardizovány.

CSS je určen především k oddělení vzhledu a obsahu, včetně aspektů, jako je rozvržení uspořádání prvků na stránce, barev a písma. Toto rozdělení může zlepšit přístupnost obsahu, poskytnout větší flexibilitu a kontrolu v prezentaci obsahu, umožnit více stránkám sdílet formátování specifikováním příslušného CSS v samostatném souboru *.css* nebo snižovat složitost a opakování v strukturálním obsahu.

Třetí verze kaskádových stylů přináší

- zaoblené rohy
- stíny u textu
- transformace
- a další

V dnešní době jsou HTML5 a CSS3 základním stavebním kamenem webových stránek a jelikož rozšíření do prohlížečů vychází ze stejných technologií, použiji právě je.

5.1.2 JavaScript ES6

ECMAScript (známější pod zkratkou ES) je specifikace jazyka pro skriptovací jazyky [22], která byla vytvořena pro standardizaci jazyka JavaScript. JavaScript je nejznámější implementací ECMAScriptu od vydání prvního standardu. Mezi další známé implementace patří JScript nebo ActionScript [23].

Šesté vydání (odtud pochází zkratka ES6), oficiálně známé taky jako ECMAScript 2015, bylo dokončeno v červnu 2015 [24]. Tato aktualizace přidává novou syntaxi pro psaní složitých aplikací, včetně tříd a modulů, definuje je ale shodně jako striktní režim ECMAScript 5. Další nové funkce zahrnují iterátory a `for` cykly ve stylu Pythonu, `arrow` funkce, binární data, kolekce (`maps`, `sets` a `weak maps`), promisy, vylepšení aritmetiky a mnoho dalšího.

Na rozdíl od HTML a CSS měl JavaScript odjakživa těžký život ve starších prohlížečích kvůli jejich kompatibilitě. To v našem případě neplatí, protože doplňky typu WebExtension podporují jenom nejmodernější prohlížeče, ve kterých je podpora ES6 samozřejmostí.

Někdo by mohl namítnout, že novinky představené v ES6 jsou v porovnání s jejich ekvivalentem v ES5 znatelně pomalejší a v některých případech by měl i pravdu. Příkladem nám může být právě tolik proklamované ES6 třídy. V jejich základní podobě jsou téměř totožně rychlé jako jejich ekvivalent v ES5 (`prototype`). Pokud s nimi začneme dělat pokročilejší úkony, řekněme například dědění, zůstává ES6 výkonově oproti ES5 pozadu. Jako příklad si vezmeme následující kus kódu:

- JavaScript ES5:

```
function Foo () {
   Base.call(this);
}

Foo.prototype = Object.create(Base.prototype);
Foo.prototype.foo = function () {
    return Base.prototype.foo.call(this) + "bar";
}
check(new Foo(), Foo);
```

- JavaScript ES6:

```
class Foo extends Base {
    constructor () {
        super();
    }

    foo () {
        return super.foo() + "bar";
    }
}
check(new Foo(), Foo);
```

Identický kus kódu zapsaný v ES6 je zhruba o 40 % až 50 % pomalejší jak v nejnovějším Chromu 58 tak Firefoxu 54, viz [25]. Tento handicap ES6 nás však moc tížit nemusí. Rozšíření by měla pouze doplňovat či obohacovat pohyb uživatele po internetu, tudíž se od nich nečeká nikterak složitá logika k provádění, na rozdíl od server-side aplikací.

5.1.3 Node.js

Node.js je open-source prostředí pro spuštění JavaScriptového kódu na straně serveru. Historicky byl JavaScript používán především pro skriptování na straně klienta, ve kterém byly skripty vloženy do HTML stránky. Node.js umožňuje spouštění JS skriptů na straně serveru k vytvoření obsahu dynamické webové stránky před odesláním stránky do webového prohlížeče uživatele. V důsledku toho se Node.js stal jedním ze základních prvků paradigmatu „JavaScript everywhere“ [26], umožňující sjednocení vývoje webových aplikací okolo jednoho programovacího jazyka, než spoléhat na jiný programovací jazyk běžící na straně serveru a jiný na straně klienta.

V našem případě však Node.js využiji jinak. Poslouží mi totiž jako nástroj pro sestavování rozšíření pro specifikovaný prohlížeč, jelikož jeho prostředí můžeme využít jenom na straně serveru a v rozšířeních by v rámci prohlížečů nefungoval (prohlížeče ho nemají jak zkompilovat a spustit). Nejvíce mě bude zajímat jeho ekosystém balíčků - **npm**, který je největším systémem pro distribuci open-source knihoven na světě. Právě díky *npm* budu moci využít knihovny Gulp.

5.1.4 Gulp.js

Gulp.js[1] je nástroj pro automatizaci časově náročných úkolů vznikajících během vývoje softwaru. Mnoho úkolů může být zjednodušeno chytrou automatizací a my tak můžeme věnovat více času do psaní kódu.

Gulp je „JavaScriptový správce úloh“ (JavaScript Task Runner), který umožňuje automatizovat úkoly, jako jsou:

- přibalování a zmenšování knihoven a stylů
- obnovování prohlížeče při uložení souboru
- spouštění analýzy kódu
- automatická Less/Sass do kompilace čistého CSS
- kopírování souborů do adresářů

Gulp je svým způsobem podobný svému většímu a mnohem známějšímu bratříčkovi z rodiny *task runnerů* - **Gruntu**[2]. Gulp má však tendenci dělat věci jednodušeji a přehledněji za cenu nemožnosti vytvoření přehledného konfiguračního souboru a nižšího výkonu při kompilaci. V našem případě však Gulp soubory nebudou nijak rozsáhlé a jejich logika nebude nikterak náročná, tudíž si s Gulpem vystačíme.

5.1.5 AngularJS

AngularJS je open-source frontendový JavaScriptový framework[3], jehož cílem je rozšířit webové aplikace pomocí architektury Model-View-Controller a snížit tak množství JavaScriptu potřebného k dosažení efektu dynamiky stránek. Tyto typy aplikací jsou známé taktéž pod označením *Single-Page Applications*[4].

[1] `http://gulpjs.com/`
[2] `https://gruntjs.com/`
[3] Framework je softwarová struktura, která slouží jako podpora při programování a vývoji a organizaci jiných softwarových projektů. Může obsahovat podpůrné programy, knihovny API, podporu pro návrhové vzory nebo doporučené postupy při vývoji. [27]
[4] Příslušné zdroje a obsah ve stránce je dynamicky načten a měněn dle potřeby, obvykle jako odezva na akce uživatele.

Mezi hlavní přednosti Angularu patří zejména:

- **REST**, který se rychle stává standardem pro komunikaci klienta se serverem. JavaScript tak můžete rychle komunikovat se serverem a získat data, která potřebujete k interakci s webovými stránkami. AngularJS to změní na jednoduchý objekt JavaScriptu, jako modely, podle vzoru MVVM (Model View View-Model). Všechno co se v rámci Angularu děje je po vzoru **MVVM** a je automaticky komunikováno s uživatelským rozhraním vždy, když se něco změní. To eliminuje potřebu obalů, geterů/seterů a dalších
- **Rozšiřování HTML**. Většina webových stránek je v dnešní době jen obrovská nudle *divů* s jen malou sémantickou jasností. Je posléze zapotřebí vytvořit rozsáhlé a vyčerpávající CSS třídy, které vyjadřují záměr každého objektu v doméně DOM. Pomocí Angularu můžeme ovládat HTML jako XML, což nám nabízí nekonečné možnosti pro tagy a atributy. Angular toho dosáhne prostřednictvím kompilátoru HTML a za použití směrnic k vyvolání chování na základě nově vytvořené syntaxe, kterou píšete. Můžeme díky tomu vytvářet i zcela nové HTML značky jako například *my-modal*.
- **Vytvoření HTML šablony**. Angular prochází DOM pro tyto šablony, které jsou pak předány do kompilátoru Angularu jako DOM prvky, které mohou být následně upravovány.

Díky výše uvedeným vlastnostem mi v kódu odpadne jedna z nejvíce nepříjemných činností, a to manipulace DOMu JavaScriptem. Pro lepší porozumění těchto předností opět uvádím dva bloky kódu, které mohou být součástí libovolného *index.html* a mají shodnou funkcionalitu:

- Čistý JavaScript:

```
<h1>Write your name in textbox</h1>
<input id="hello-input" type="text">
<p id="hello-output">Hello </p>

<script>
  var inputField = document.getElementById('hello-input');
  var label = document.getElementById('hello-output');

  var handleKeyup = function() {
    var value = inputField.value;
    label.innerHTML = 'Hello ' + value;
  }

  if (document.addEventListener) {
    document.addEventListener('keyup', handleKeyup);
  } else if (document.attachEvent) {
    document.attachEvent('keyup', handleKeyup);
  }
</script>
```

- Angular:

```
<h1>Write your name in textbox</h1>
<input type="text" ng-model="name" >
<p>Hello {{name}}</p>
<script src="/angular.js"></script>
```

Při otevření souboru v prohlížeči uvidíme, že oba dva kódy dělají to samé, avšak v druhém případě je zdrojový kód znatelně kratší a mnohem přehlednější. Zároveň s tím i jeho vývoj a údržba bude rychlejší.

5.2 Přepis „Seznam Lištičky“

Na začátku jsem začal s vývojem rozšíření pouze pro prohlížeč Chrome. Vývoj a ladění v něm bylo velmi pohodlné a intuitivní. Načítání nedokončených doplňků je jednoduché a Chrome kolem celé problematiky neověřených doplňků nedělá příliš cavyků, na rozdíl od Firefoxu, který nám dovolí načíst doplněk jen dočasně, přes speciální prostředí *about:debugging*.

To s sebou nese komplikace v podobě nekonzistentnosti ID doplňků a tudíž nemůžeme jednoduše odladit například komunikaci mezi dvěma doplňky. Zároveň s tím i testování migrace dat mezi verzemi doplňků ve Firefoxu je značně nekonzistentní, protože pokaždé načítáme nový doplněk. Navíc ve Firefoxu při načtení doplňku označeného jako *dočasný* nejsou provolány některé události, jako například `onInstall` nebo `onStart`, ve kterých můžeme mít implementované inicializační metody pro zavedení dat.

V neposlední řadě přináší Chrome s každým doplňkem jeho vlastní konzoli pro debuggování. Máme tak krásný přehled nad tím, odkud data do konzole přicházejí, jelikož se v rámci konzole příslušící danému doplňku zobrazují data vztahující se pouze k doplňku samotnému a ničemu jinému. Ve Firefoxu máme jednu společnou konzoli pro všechny úkony na pozadí.

A proto je pohodlnější a rychlejší vyvíjet rozšíření prvotně pro Chrome a až posléze jej upravovat pro prostředí Firefoxu.

Základ všech rozšíření prohlížeče je stejný: jde o aplikace skládající se z HTML/CSS/JavaScript souborů a manifestu. Díky tomu byl můj první nápad, že pokaždé zkopíruji repozitář s rozšířením pro Chrome a upravím ho tak, aby kód fungoval i ve Firefoxu.

Během vývoje jsem měl pocit viny za to, že dělám zbytečně mnohokrát *copy-paste*. Je jasné, že 99 % kódu je naprosto shodných pro obě rozšíření a v budoucnu by tento způsob přenosu mezi prohlížeči mohl přinést problémy s podporou aplikace, neboť stále více a více času by zabralo kopírování a opakované přepisování rozdílných částí.

Díky tomu jsem se začal zabývat, jak zautomatizovat proces sestavování doplňku pro specifický prohlížeč, aniž bych musel cokoliv repetitivně kopírovat. S tímto problémem mi nakonec pomohl Gulp.js, který jsme si podrobněji uvedli v sekci 5.1.4.

5.2.1 Struktura nového rozšíření

Důležité je si uvědomit, které soubory se budou týkat logiky běžící na pozadí a které budou zobrazovat obsah uživateli. Zároveň s tím je vhodné vytyčit si jednu třídu, která se bude start o komunikaci mezi obsahem a pozadím rozšíření.

Uzpůsobil jsem k tomu i strukturu aplikace, o které se domnívám, že je vhodná pro jakékoliv rozšíření:

```
WebExtension/
    build/
        chrome/
        firefox/
    css/
    fonts/
    dist/
        chrome/
        firefox/
    img/
    js/
        libs/
    node_modules/
    tools/
    vendor_specific/
    gulpfile.js
```

- Adresáře `build` a `dist` jsou generické složky, které obsahují sestavené zdrojové kódy a ubalené aplikace připravené k distribuci.
- `img` a `fonts` obsahují statické soubory.
- `html`, `css` a `js` obsahují zdrojové soubory.
- `tools` obsahuje nástroje pro sestavování rozšíření apod.
- `vendor_specific` obsahuje platformově závislé soubory, jako je například `manifest.json`, v oddělených adresářích pro každý prohlížeč.

5.2.2 Automatizace sestavování rozšíření

Účel automatizovaného sestavovacího procesu je zkopírovat jádro aplikace spolu s platformě závislými částmi kódu a statickými soubory do složek určených dle cílového prohlížeče.

Ukázka buildu pro Chrome by mohla vypadat zhruba jako:

```
gulp.task('chrome', () => {
   return es.merge(
      pipe('./fonts/**/*', './build/chrome/fonts'),
      pipe('./img/**/*', './build/chrome/img'),
      pipe('./js/**/*', './build/chrome/js'),
      pipe('./css/**/*', './build/chrome/css'),
      pipe('./html/**/*', './build/chrome'),
      pipe('./vendor_specific/chrome/api.js', './build/chrome/js/libs/'),
      pipe('./vendor_specific/chrome/manifest.json', './build/chrome/')
   );
});
```

Po zavolání příkazu `gulp chrome` v terminálu z kořenového adresáře rozšíření se nakopírují všechny vyjmenované soubory ve složkách do struktury, ve které je bude rozšíření po sestavení očekávat.

Podobně by vypadat i kód pro zabalení rozšíření. Jelikož se všechny rozšíření již distribuují v souboru *.zip*, stačí nám jen zabalit složku s hotovým buildem. V případě Chromu kód vypadá následovně:

```
gulp.task('dist-chrome', () => {
   gulp.src('./build/chrome/**/*')
      .pipe(zip('chrome-listicka-' + chrome.version + '.zip'))
      .pipe(gulp.dest('./dist/chrome'));
});
```

Byť kódy výše uvedené nejsou kompletní, můžeme jasně vidět, že kód není nikterak složitý. Přitom můžeme tímto ušetřit mnoho času sestavováním a kopírováním souborů jen proto, abychom otestovali jeden změněný řádek kódu.

5.2.3 Soubory specifické pro každý prohlížeč

Podpora dedikovaných JavaScriptových API v prohlížeči je mezi Chromem a Firefoxem na dobré úrovni a byť Chrome co do počtu podporovaných API vede, všechny nejpoužívanější API jsou implementovány v obou prohlížečích. Samotná implementace se taktéž od sebe moc neliší, jelikož API Firefoxu je derivací API Chromu[1]. Přesto stále existují drobné rozdíly, které musíme ošetřit.

Prvním rozdílem, který nás hned napadne, jsou odlišnosti v manifestu. Byť se jedná jen o pár pozměněných řádků, nevyplatí se data souboru vytvářet genericky. Krom toho, že každý doplněk může mít odlišný název či popis, tak můžou mít odlišné i verze doplňku díky tomu, že můžeme opravit například dedikované JS API, které daný prohlížeč nedávno pozměnil. V tom případě by ale nedávalo smysl vydávat novou verzi i pro prohlížeč, ve kterým jsme úpravy dělat nemuseli.

Navíc některé klíče v manifestu jsou definovány odlišně nebo nejsou v prohlížeči vůbec podporovány. Jako příklad uvedu rozdíl v klíči manifestu určující výchozí soubory pro zobrazení stránky s nastavením rozšíření.

- Chrome:

```
"options_page": "options.html",
```

- Firefox:

```
"options_ui": {
    "page": "options.html",
    "open_in_tab": true
},
```

Díky těmto drobným rozdílům, se vyplatí držet v repozitáři oddělené manifesty pro každý prohlížeč.

To samé platí i pro soubor určený k volání API prohlížeče. Nejjednodušší a esteticky nejlepší řešení bude, když si vytvoříme vlastní soubor, ve kterém definujeme metody, jenž můžeme posléze volat napříč celou aplikací ať už z prostředí Chromu nebo Firefoxu. Abychom zabránili kolizím názvů funkcí, bude vhodné, když si vlastní metody zadefinujeme do vlastního namespacu - řekněme třeba `Foxcub.API`.

[1] `https://chromium.googlesource.com/chromium/src/+/master/extensions/common/api/web_request.json`

Volání API v Chromu by pak namísto

```
chrome.storage.local.get();
```

a ve Firefoxu namísto

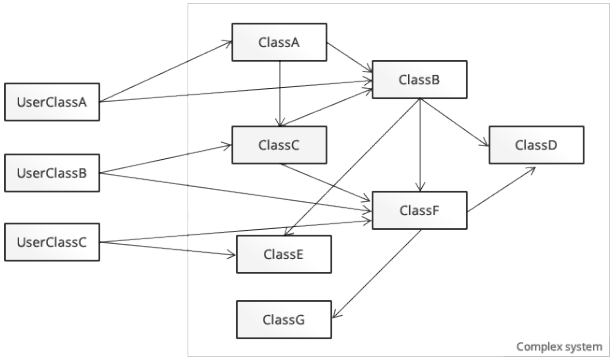
```
browser.storage.local.get();
```

mohlo vypadat následovně:

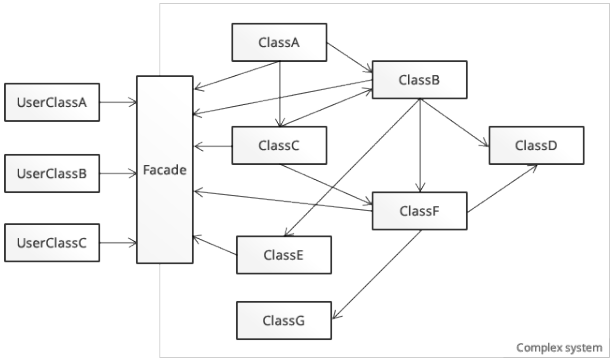
```
Foxcub.API.getFromStorage();
```

Tím docílíme, že kód může zůstat konzistentní a jediné, co se bude měnit, bude soubor s vlastním definovaným API. Může ovšem nastat situace, že implementace API nebude shodná mezi prohlížeči. V tom případě bychom museli vytvořit abstraktní wrapper, který by zajišťoval rozdíly při volání funkcí nebo ošetřoval jejich odlišné návratové hodnoty.

Při využívání komplexního systému pro relativně jednoduché úlohy je výhodné vytvořit vrstvu mezi samotným systémem a vlastními třídami, což má za následek zjednodušení používání systému a dále jeho „schování“ za fasádu[1]. Vedlejším efektem fasády je poté možnost změnit systém, které komunikují pouze s prostředníkem, tedy fasádou. Ukázka jednoduchého diagramu bez použití fasády je na obrázku č. 5.1 a s použitím na obrázku č. 5.2.



Obrázek 5.1. Systém bez použití fasády. Autor: Aleš Menzel



Obrázek 5.2. Systém s použitím fasády. Autor: Aleš Menzel

[1] z anglického sousloví Facade pattern

Účel fasády lze shrnout jako: „Účelem fasády je umožnit uživatelům s jednoduššími požadavky pracovat se složitým systémem, jako by byl jednoduchý.“ [28]

Ukázkový kus kódu, jak by takový soubor mohl vypadat:

```
let fox = window.Foxcub = window.Foxcub || {};
fox.API = {
    saveToStorage: chrome.storage.local.set,
    getFromStorage: chrome.storage.local.get,
    removeFromStorage: chrome.storage.local.remove
}
```

Největší zvláštností se může jevit první řádek, který nám definuje jakýsi namespace `Foxcub` v rámci globálního `window` objektu. Jinak řečeno, vytvoříme objekt `Foxcub` v rámci `window` s tím, že mu přiřadíme již existující objekt `Foxcub` za předpokladu, že již byl vytvořen, jinak mu přiřadíme objekt prázdný. Tento objekt `Foxcub` posléze přiřadíme do vlastní lokální proměnný `fox`, do kterého zadefinujeme objekt `API`. Kouzlo první řádku je především v tom, že všechny změny, které provede v lokální proměnné `fox` se zároveň promítnou i do globální `Foxcub`.

Další raritou se může zdát samotné volání metod API, ke kterým pouze předáváme reference. Pokud bychom za `chrome.storage.local.set` přidali závorky, změnila by se reference na volání funkce a museli bychom tím pádem naimplementovat celou logiku volání funkce a jejich návratových hodnot. Ty navíc dostáváme buďto pomocí callbacku nebo Promis, tudíž bychom museli ošetřovat další stupeň implementace. Vytvářeli bychom tak zbytečný mezistupeň volání funkce, který by zbytečně způsoboval nabalování kódu.

5.2.4 Práce s třídami

JavaScriptové soubory v rozšíření se dělí na skripty na pozadí a obsahové skripty. Všechny skripty jsou načítány v rámci HTML stránky, ať už se jedná o webovou stránku s nastavením nebo stránku běžící na pozadí. Všechny JavaScripty se totiž musí načíst v nějakém jmenném prostoru a proto (pokud neurčíme HTML stránku sami, například `background.html`) ji prohlížeč při zavádění automaticky vytvoří a skripty do ní nakopíruje.

V novém doplňku jsem se rozhodl vytvořil vlastní stránku, která vytváří prostředí pro běh na pozadí. V tomto případě jsem mohl skripty zadefinovat přímo v manifestu, protože jich je jen pár, ovšem kdyby doplněk obsahoval skriptů víc, manifest bych mohl zbytečně přeplnit pouhým vyjmenováváním JS souborů. Navíc ve vlastní `background.html` mám lepší kontrolu nad pořadím načítání skriptů, což je v tomto případě nezbytné. Potřebujeme totiž zajistit, že naše vlastní `Foxcub.API` bude načtené ještě předtím, než začneme inicializovat jednotlivé třídy.

Hlavním JavaScriptovým souborem je `main.js`. Jeho obsah je krátký avšak velmi důležitý.

```
if (window.Foxcub && window.Foxcub.API) {
    Foxcub.History = new History();
    Foxcub.RSS = new RSS();
    ...
} else {
    if (!window.Foxcub)     console.error('Foxcub');
    if (!window.Foxcub.API) console.error('Foxcub.API');
}
```

Soubor `main.js` zajistí načtení všech potřebných souborů, ale jen a pouze v případě, že je již zadefinované `Foxcub.API`. Pokud by totiž objekt Foxcub.API neexistoval, nemohli bychom přistupovat k API prohlížeče a tím pádem by nefungovalo celé rozšíření.

Po inicializaci všech tříd pak k nim a jejím funkcím můžeme přistupovat přes `Foxcub.<className>`.

Nutno podotknout, že se nemusíme striktně držet namespacu Foxcub. Můžeme si vymyslet svůj vlastní nebo jej vůbec nepoužívat, protože každý doplněk běží ve svém vlastním jmenném prostoru.

5.2.5 Ukládání dat

Při výběru úložiště dat ve WebExtensions máme na výběr z několika možností. Já se ale zaměřím na dva hlavní způsoby a to:

- `chrome.storage.local` nebo `browser.storage.local` (záleží na prohlížeči)
- `window.localStorage`

Při výběru záleží zcela na tom, co od rozšíření očekáváme. `Window.localStorage` je úložiště HTML5. Pokud není úložiště nepoužíváno na pozadí, umožňuje získávat a ukládat data pouze pro konkrétní doménu. To samé platí pro kód vložený do kontextu DOM, protože by na webové stránce používalo `localStorage`.

Jinými slovy, v případě použití `localStorage` nebudeme moci sdílet data mezi různými webovými stránkami, pokud tento typ úložiště nepoužijeme na stránce v pozadí, která pracuje nezávisle na webových stránkách, neboť její doména obsahuje řetězec `chrome://URI`.

Na druhou stranu `chrome.storage.local` je od základu navržen pro rozšíření k ukládání dat na centrálnějším místě. Vzhledem k tomu, že není přístupné běžným webovým stránkám, každé rozšíření získá vlastní úložiště. Jednou z možností je, aby stránka na pozadí zpracovávala logiku týkající se třeba nastavení nebo získávání dat, zatímco obsahové skripty se zabývají úpravou a interakcí s webovou stránkou. Nicméně rozhraní storage API může být voláno jak z obsahových tak ze skriptů běžících na pozadí.

Další výhodou `chrome.storage.local` oproti `localStorage` je typ ukládání dat. Zatím co `localStorage` ukládá data pouze jako řetězec, `chrome.storage.local` ukládá data jako objekt a díky tomu dokáže v alespoň omezené míře udržet typovost dat. Pokud chceme uložit data ve formátu objekt, nemusíme je převádět na řetězec a zase zpátky při získávání dat z úložiště. Prohlížeče navíc poskytují k API úložiště doplňující funkce a posluchače, tudíž můžeme být například informováni pokaždé, když se změní část dat. Díky jeho neblokujícímu chování se nemůže stát, že doplněk zamrzne při komunikaci s úložištěm.

Navíc, tím že jsou rozšíření na tzv. *whitelistu* prohlížeče, protože se je uživatel rozhodl nainstalovat, mají obvykle více volného prostoru než normální webové stránky s `localStorage`. Dokonce můžeme zadáním oprávněním `unlimitedStorage` v manifestu ukládat data mnohem větší než limit 5MB omezující HTML5 `localStorage`.

Na druhou stranu má i `chrome.storage.local` svá omezení. Na rozdíl od `localStorage` je totiž asynchronní. Musíme tedy při každém dotazu na data v úložišti navěsit callback a teprve až při jeho získání pokračovat v práci s daty. Jelikož jsou data ukládána lokálně v prohlížeči, asynchronní komunikace s uložištěm je svižná a standardně proběhne během několika malá milisekund. Bohužel to ale přispívá horší čitelnosti kódu.

Přes všechny nevýhody `localStorage` jsem v modelovém doplňku použil právě jej, protože všechna komunikace probíhá synchronně, kód je tedy úhlednější, a díky jeho používání výhradně na pozadí rozšíření, jsou data přístupná vždy když potřebujeme. Zároveň s tím, data, se kterými doplněk pracuje, jsou velice jednoduchá, není tedy zapotřebí je strukturalizovat do složitých objektů.

5.2.6 Šablonovací systém Angularu

Jak už jsem uvedl v kapitole 5.1.5, Angular mi usnadní velikou část práce při manipulaci s DOMem. V Angularu jsou šablony zakomponovány do HTML kódu, který obsahuje elementy a atributy specifické čistě pro Angular. Ten navíc kombinuje šablony s informacemi, které získává z Modelu a Controllerů, aby mohl dynamicky vykreslit data, která uživatel právě vidí [29].

V Angularu existují 4 hlavní prvky a atributy, které můžou být použity:

- Směrnice - *directive* - je atribut nebo prvek, který rozšiřuje existující prvek DOMu nebo definuje jeho opakovaně použitelnou část.
- Značka - *markup* - jsou dvojité složené závorky `{{ }}` pro určení pozice v DOMu, na které jsou navázány proměnné Angularu.
- Filtr - *filter* - Formátuje data pro zobrazení.
- Ovládací prvky formuláře - *form controls* - Ověří vstup uživatele.

Následující fragment kódu z mého rozšíření (speedDial.html) zobrazuje šablonu se směrnicemi a vazbami Angularu:

```
<!DOCTYPE html>
<html ng-app="speedDial">
<head></head>
<body ng-controller="RootCtrl">
  <div id="header" ng-class="bigResolution ? 'big' : 'small'">
    <h2 id="headline">Rychlé volby</h2>
    <div class="item" ng-repeat="tile in tiles">
      <a href="{{tile.url}}" rel="{{$index}}"
         ng-click="goto($event, $index)">
      </a>
      <p>{{tile.title}}</p>
    </div>
  </div>
</body>
</html>
```

Ve skriptu (speedDial.js) kód k obsluze výše uvedené šablony ve zkrácené formě vypadá takto:

```
var sd = angular.module('speedDial', []);
sd.controller('RootCtrl', function($scope) {
  let viewWidth = window.innerWidth;
  let viewHeight = window.innerHeight;
  $scope.bigResolution = !(viewWidth < 1250 || viewHeight < 825);

  $scope.tiles = JSON.parse(localStorage.getItem("tiles"))
                 || DEFAULT_SPEEDDIAL;

  $scope.goto = function (e, index) {
    Foxcub.API.updateTab(null, {url: targetTile.url});
  };
});
```

Díky direktivám můžeme mít v jednom souborů více šablon najednou, protože každá HTML šablona musí mít ID. Pomocí něho přiřadíme ke každé šabloně obsluhující controller. Všechny data, která se změní buďto v šabloně nebo controlleru ihned změní díky obousměrné komunikaci, které mezi nimi neustále probíhá.

5.3 Migrace dat

Může se stát, že bychom chtěli během přechodu na novější technologii zachovat data uživatele, které má již ve stávajícím doplňku. Jelikož se Chromu tato problematika netýká (Chrome podporuje pouze rozšíření typu WebExtension), zaměřím se v této sekci výhradně na problematiku Firefoxu.

Od Firefoxu verze 51 můžeme vložit rozšíření WebExtension do klasického bootstrapped nebo Add-on SDK doplňku.

Vložené soubory WebExtension jsou zabaleny uvnitř původního doplňku. Vestavěná aplikace WebExtension přímo nesdílí svůj obsah s doplňky starších verzí, ale mohou si vyměňovat zprávy pomocí funkcí pro zasílání zpráv definovaných v runtime API prohlížeče. To znamená, že můžeme migrovat uložená data z původního doplňku do nového WebExtension, a to napsáním hybridního doplňku, který čte data pomocí starších rozhraní API (například *prefs* nebo služby předvoleb Firefoxu) a zapisuje pomocí rozhraní API WebExtension.

Uložená data uživatele ve starém doplňku jsou velkým problémem při přechodu na novější technologie, protože starší doplňky nemohou používat API pro ukládání dat WebExtension, zatímco WebExtensions nemohou používat starší rozhraní API pro úložiště. Například pokud doplněk Add-on SDK používá pro ukládání předvoleb simple-prefs, verze WebExtension nebude mít k těmto datům přístup. To samé platí i pro doplňky typu XUL/XPCOM, které využívají úložiště předvoleb Firefoxu[1]. Jelikož se tímto dostáváme do interních částí Firefoxu, je pochopitelné, že v nových WebExtensions je tendence tyto části nezpřístupňovat, právě i díky možnému zneužití, protože doplňku pak nic nebrání ve zneužití předvoleb.

[1] Do předvoleb se dostaneme přes adresní řádek Firefoxu, kam zadáme adresu *about:config* a potvrdíme, že jsme si vědomi rizik, které s sebou přináší možnost změny konfiguračních předvoleb samotného Firefoxu.

Standardní proces migrace by byl následující:

1. Výchozí verze rozšíření, ze kterého chceme data přenést a zapisuje předvolby do preferencí prohlížeče.
2. Vytvoření hybridního rozšíření, které bude obsahovat vnořené a silně omezené API WebExtension. To bude schopné si vyžádat od starší části načtení uložených dat z preferencí a jejich předání části s WebExtension, které je posléze uloží do svého vlastní uložiště.
3. Nakonec se vytvoří nová verze, která bude založená pouze na WebExtension a bude již mít přístup k uložišti s daty ze staré verze.

Bohužel tento postup nemůžeme použít v našem rozšíření „Seznam Lištička“, protože nezapadá ani do jedné z podporovaných technologií pro implementaci vestavěných WebExtension. Náš doplněk se totiž řadí do ještě starší rodiny rozšíření, a tudíž budeme muset najít jinou cestu, jak dostat data uživatele do novějších WebExtensions.

Jelikož nám odpadá využití nativních API jak XUL/XPCOM tak WebExtensions, musíme nějaké jiné médium pro přenos dat takové, aby k nim měly přístup obě dvě verze rozšíření. Máme na výběr z:

- Uložení dat na externí server
- HTML5 localStorage
- Downloads API
- Bookmarks API
- Cookies API

Externí servery v našem případě nejsou zcela bezpečné řešení. Pokud bychom totiž chtěl migrovat nějaká osobní data, jako například přihlašovací údaje do emailu uživatele, museli bychom zaručit, že se k přihlašovacím údajům nedostane někdo jiný. Museli bychom tedy aplikovat jistou formu autorizace, která by znamenala dodatečnou práci a potřebný čas. Navíc bychom museli připravit servery, na které by se data odesílala, což by mohla být mimo jiné i finanční zátěž, pokud bychom již neměli k dispozici žádné vlastní servery.

HTML5 localStorage: Lokální úložiště má využití v případě, kdy je potřeba návštěvníkovi uložit data v rámci webové stránky, která není potřeba přenášet na server. To je třeba případ průběžného ukládání obsahu formulářů, které se díky lokálnímu úložišti může provádět velmi často. Ve WebExtensions máme absolutní přístup k localStoragi, jelikož se tento typ rozšíření tváří v rámci prohlížeče jako speciální webovka. Mohli bychom tedy využít jej. V XUL/XPCOM je ale situace zcela jiná. Pokud bychom chtěli z něj přistoupit k localStorage, prohlížeč nám vrátí následující chybu:

```
Exception... "Component is not available"
nsresult: "0x80040111 (NS_ERROR_NOT_AVAILABLE)"
location: "JS frame :: debugger eval code :: <TOP_LEVEL> :: line 1"
data: no
```

Downloads API nám zpřístupňuje historii stahování prohlížeče, ovšem pomocí tohoto API bychom nebyli schopni trvale uložit žádná data, tudíž nebudeme moci použít ani tento způsob migrace dat.

Bookmarks API, alias záložky nebo oblíbené prohlížeče, už vypadají slibněji. Můžeme definovat název záložky, který by byl shodný s názvem proměnné, a url záložky, která by místo webového odkazu obsahovala data. Ani toto řešení není zrovna nejšťastnější, protože by i běžný uživatel mohl lehce na data narazit a navíc se snažíme ukládat data v různých formátech (JSON, řetězce, čísla) do pole určené výhradně pro formát URL. Prohlížeč by posléze mohl považovat takové záložky za poškozené nebo i nedůvěryhodné a automaticky je smazat, čímž bychom o data v novém rozšíření přišli.

Cookies API je tedy poslední možností, jak data přemigrovat. Soubory cookie jsou obvykle malé textové soubory, které obsahují vlastní ID a jsou obvykle uloženy v adresáři prohlížeče počítače. V rámci souboru může sledovat a uchovávat informace o pohybech uživatele v rámci webu a veškeré informace, které jste mohli dobrovolně poskytnout při návštěvě webových stránek. Každá cookie je tedy pevně spojená s nějakou doménou. Obsahuje název, obsah dat, čas expirace, úroveň zabezpečení a další. Má tedy všechny potřebné parametry k naší migraci dat a zároveň jen pramalá část uživatelů prohlížeče přistupuje a kontroluje obsah cookies. Stačí nám už jenom vymyslet vlastní URL s ideálně neexistující doménou prvního řádu tak, aby nemohlo dojít k přepsání cookie nějakou webovou stránkou. Jediná nevýhoda je, že uživatel může automaticky mazat cookies při zavírání prohlížeče. V tom případě bychom o data určené k migraci přišli, stále je to ale nejlepší volba, kterou můžeme použít k migraci dat z rozšíření typu XUL/XPCOM.

Kapitola 6
Zhodnocení a výsledky práce

V této kapitole se budu zabývat celkovým zhodnocením své práce, především části praktické. Vytyčím odlišnosti mezi starým a novým rozšířením jak po stránce technické, tak z pohledu uživatele. Vyhodnotím přínos technologií použitých v novém rozšíření a nastíním jejich budoucnost v Chromu a Firefoxu, včetně plánů na implementaci technologie WebExtensions v ostatních prohlížečích.

6.1 Přínos použitých technologií

Nesporná výhoda migrace do technologie WebExtensions je její téměř identická podpora napříč nejrozšířenějšími prohlížeči. To bezesporu přispívá ke zrychlení vývoje nového rozšíření a jeho následné údržbě.

Další výhodou se může jevit fakt, že tento typ rozšíření vyžaduje pro správný běh zpravidla nejnovější verze prohlížečů, tudíž není nutné myslet na zpětnou kompatibilitu používaných technologií. Samozřejmě to zmenšuje potenciální uživatelskou základnu, avšak v době automatických aktualizací prohlížečů, kterých si uživatel kolikrát ani nevšimne, se toto nejeví jako signifikantní překážka.

Mně osobně se na WebExtensions líbí, že využívají technologie, se kterými se můžu setkat při běžném vývoji webových stránek. Každý, kdo alespoň trochu ovládá HTML, CSS a JavaScript může bez větších problémů vytvořit vlastní rozšíření. Co se týče dedikovaných JS API prohlížečů, nejsou nikterak složitá a jejich dokumentace je jak ze strany Chromu [30], tak i Firefoxu [31], na velmi vysoké úrovni a díky tomu programátor nemusí zbytečně ztrácet čas hledáním.

Značný podíl na zrychlení vývoje měl v mém případě i Node.js v kombinaci s knihovnou Gulp. Automatizace procesů sestavování a balení rozšíření zmírnila mentální náročnost těchto úkonů. Zároveň s tím umožnila pokračovat v práci, zatímco se rozšíření potichu sestavovalo na pozadí.

Pokud se podíváme na obsáhlost kódu, můžeme vidět, že ke zlepšení zde nejvýrazněji přispěl šablonovací systém Angular. Jeho použití ve srovnání s čistým HTML a JavaScriptem jsem již uvedl v sekci 5.1.5. Navíc kombinací Angularu s novou verzí JavaScriptu (ES6) jsem dosáhl, při dodržení shodné funkcionality, značného zkrácení délky kódu.

Pro ověření svého tvrzení, jsem nechal spočítat délku kódu obou dvou aplikací (XUL/XPCOM a WebExtensions) knihovnou *sloc*[1], která podporuje více jak 50 programovacích jazyků, včetně HTML, CSS a JS.

[1] `https://www.npmjs.com/package/sloc`

Sloc rekurzivně projde vstupní adresář a spočítá počet řádků ve zdrojových souborech. Výsledkem je seznam s:

- počtem fyzických řádků
- počtem všech řádků se zdrojovým kódem
- počtem všech řádků s komentáři
- počtem všech prázdných řádků
- počtem řádků s TODO
- počtem zpracovaných souborů

V případě rozšíření postavené na množině technologií XUL/XPCOM jsem nechal spočítat všechny soubory v kořenovém adresáři. Výstupem bylo:

```
             Physical :  31510
               Source :  22080
              Comment :  5544
 Single-line comment :  849
       Block comment :  4695
 Mixed src + comment :  431
                Empty :  4415
                To Do :  2
------------------------------
Number of files read :  97
```

U WebExtensions jsem provedl výpočet v sestaveném rozšíření pro Chrome[1] (počet řádků ve Firefoxu je shodný, tudíž není nutné provádět výpočet pro každý z nich odděleně). Výstupem bylo:

```
             Physical :  4485
               Source :  3590
              Comment :  507
 Single-line comment :  14
       Block comment :  493
 Mixed src + comment :  9
                Empty :  423
                To Do :  0
------------------------------
Number of files read :  21
```

Důvodem, proč jsem nepoužil kořenový adresář tak jako v případě XUL/XPCOM, bylo, že v případě WebExtensions jsem použil Node.js, jenž si stahuje do adresáře `node_modules`, který je součástí kořenového adresáře, vlastní balíčky a knihovny potřebné pro svůj běh. Navíc by byl výsledek včetně adresáře se sestavenými rozšířeními zároveň pro Chrome i Firefox a došlo by tedy k multiplikaci počtu duplicitního kódu.

Jak můžeme vidět, přepisem došlo **sedminásobnému snížení** počtu všech řádků a zároveň skoro **pětinásobnému snížení** počtu zdrojových souborů. To bude mít v budoucnu za následek zvýšení udržovatelnosti kódu.

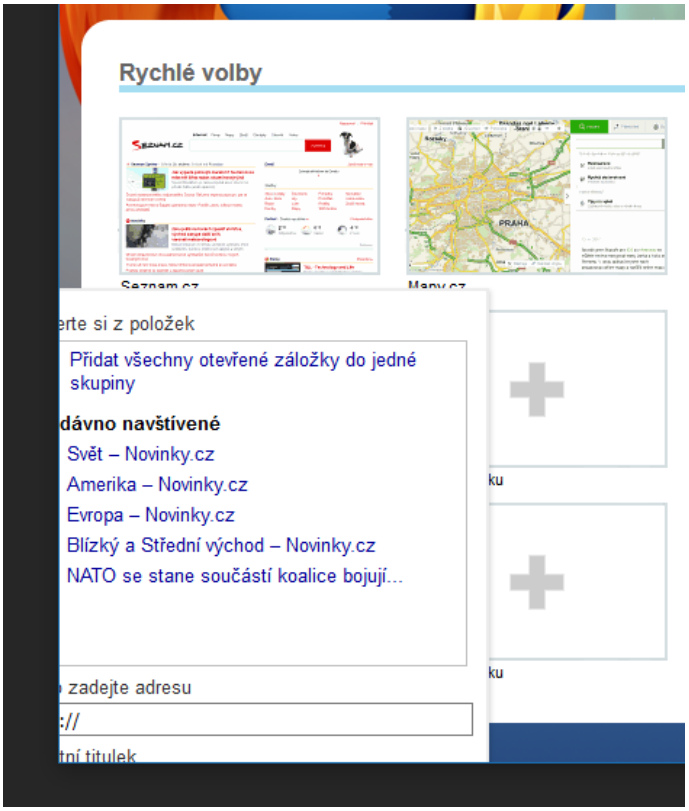
[1] Nejprve se musí rozšíření sestavit a teprve až posléze se může provést výpočet. Cesta k adresáři pro výpočet je: /app/WebExtension/build/chrome

6.2 Rozdíly z pohledu uživatele

Byť jedním z úkolů bylo naprogramování shodného řešení v technologii WebExtensions, nebylo vždy možné úkolu dostát na sto procent. Zároveň se vyskytly dva případy, kdy z pohledu UX bylo stávající řešení krajně nepraktické (řeč je o vyskakovacím okně pro přidávání či úpravu dlaždice a pak přiblížení dlaždic po najetí myší).

6.2.1 Vyskakovacího okno

Řešení, jakým jsou ve starém doplňku řešena vyskakovací okna při přidávání nové rychlé volby nebo změně současné, vskutku není optimální. Pokud se pohybujeme v prostředí s vysokým rozlišením, vyskakovací okno (dále jen popup) funguje standardně a nikterak nebrání funkčnosti aplikace. V případě, že menšíme okno pod určitou hranici, popup se nedokáže vykreslit do omezeného prostoru okna a místo toho se vykreslí jen částečně tak, jak je uvedeno na obrázku č. 6.1, nebo se nevykreslí vůbec[1]. Navíc popup se vykresluje v relativním poměru vzhledem k aktuální pozici myši.



Obrázek 6.1. Chybový stav vyskakovacího okna v XUL/XPCOM

K problému jsem přistoupil ze záměrem jej vyřešit jednoduchý trikem, a to vytvořením absolutně pozicovaného popupu zarovnaného na střed okna, jehož podklad by překrýval celou stránku, viz obrázek D.11. Uživatel by tak jasně viděl, že je v jiné vrstvě grafického rozložení a měl by se vypořádat s nastalým stavem.

[1] Ve skutečnosti se vykreslí zcela mimo aktivní okno, uživatel jej ale nemá šanci spatřit.

Pro lepší ovladatelnost aplikace má několik cest, jak se s popupem vypořádat:

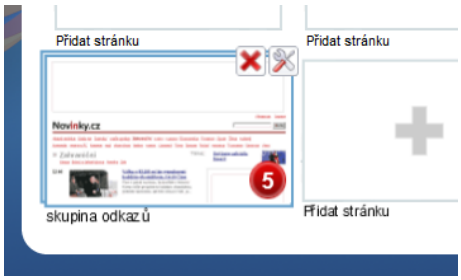
- Uložit vložená data
- Navrácení se přes tlačítko „Zrušit“
- Zavření popupu a neuložení dat pomocí křížku v pravém horním rohu
- Zavření popupu a neuložení dat klikem do ztmaveného podkladu

Moje řešení se navíc vypořádá i s velmi malým rozlišením obrazovky, byť tomu byla obětována část přístupnosti. Neočekávám však, že by bylo rozšíření dlouhodobě používáno v takto extrémních podmínkách.

6.2.2 Násobné dlaždice

Starší verze podporuje funkci přidání více karet do jedné dlaždice, viz obrázek 6.2. Násobné dlaždice se posléze chovají stejně jako ty jednoduché, jen s tím rozdílem, že se otevře více karet najednou. V případě modelového obrázku by to bylo karet pět. Použití násobných karet se může zkomplikovat v případě, že si přesně nepamatujeme, jaké karty byly součástí které násobné dlaždice. Na první pohled totiž nikde neinformují o jejich přesném obsahu.

Po prozkoumání jejich použitelnosti jsem upustil od implementace v novém rozšíření. Výsledkem analýzy bylo, že výběr karet, které by měly být součástí jedné dlaždice, byl velice nepraktický a pro pohodlné používání by musel být prvek podroben UX analýze a vytvoření návrhu možného řešení. Jelikož toto ale nebylo součástí původního zadání diplomové práce, nevěnoval jsem tomu pozornost a přenechal případným zájemcům o další rozpracování této problematiky.



Obrázek 6.2. Násobné dlaždice

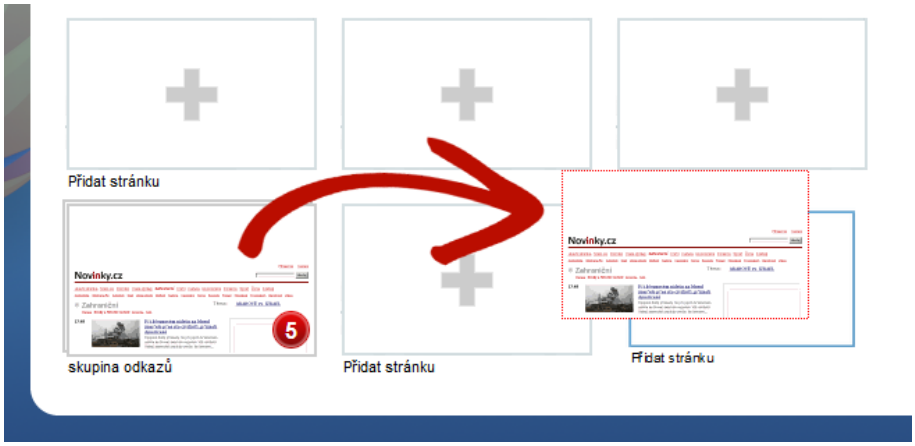
6.2.3 Drag and Drop

Pokud bych měl vysvětlit, co přesně znamená drag and drop, tak asi nejpřesnější definice pojmu bude: „*Operace v informatice, která se používaná v grafickém uživatelském rozhraní, kdy uživatel v počítači „uchopí“ pomocí ukazovacího zařízení (např. myši) virtuální objekt (např. soubor nebo ikonu) a přesune ho „přetažením“ na jiné místo (nebo na jiný objekt).*“ [32]

Ve starém doplňku je tato funkce využívána k přemisťování jednotlivých dlaždic mezi sebou tak, jak můžeme vidět na obrázku č. 6.3. Je to velice šikovná pomůcka, která usnadní manipulaci s pořadím dlaždic, aniž bychom museli dlaždice mazat a zase přidávat na jiné pozici. Původně jsem chtěl tuto funkci implementovat i v rozšíření novém, bohužel však již nezbyl dostatek času na dokončení a proto jsem ji musel zcela odebrat.

Idea byla, že k implementaci využiji knihovny *angular-drag-and-drop-lists*[1] jejíž autorem je Marcel Juenemann. Narazil jsem však na komplikace při jejím začleňování do svého projektu jejichž řešení by zabralo mnohem více času, než jsem původně vyhradil.

[1] `https://marceljuenemann.github.io/angular-drag-and-drop-lists/demo/#/nested`



Obrázek 6.3. Drag and Drop dlaždic

Je to jeden z dalších návrhů na pozdější dopracování.

6.2.4 Zoom dlaždic

Zoom, nebo-li přiblížení, dlaždic je další ukázkou nepraktického návrhu chování aplikace, i když v tomto případě nikterak nebrání v její plné funkčnosti. Zoom v rozšíření funguje tak, že při najetí myší na dlaždici se automaticky „přiblíží“. Problém spočívá v tom, že tohoto efektu bylo docíleno použitím CSS vlastnosti `transition`.

`Transition` se projevuje tak, že vezme element v původním rozlišení a změní jeho rozměry v závislosti na předaném koeficientu[1]. Koeficient předávaný v našem rozšíření zapříčiní zvětšení („přiblížení“) dlaždice za cenu rozmazání jejího obsahu za předpokladu, že neobsahuje vektorovou grafiku (té se problematika rozmazávání netýká). V tomto případě to s sebou nese i druhý negativní účinek - vysunutí okrajů dlaždice mimo předdefinovanou mřížku, jak můžeme vidět na obrázku 6.4.

Ve WebExtensions jsem efekt nahradil pouhým podbarvením okrajů dlaždice. Toto řešení považuji za elegantní kompromis mezi uvědoměním uživatele na změnu stavu aplikace (najel na živý objekt) a zachováním původního vzhledu rozšíření.

[1] Pokud je koeficient větší než 1, element se vizuálně zvětší, a pokud je koeficient menší než 1, tak jej naopak zmenší.

Obrázek 6.4. Nevhodné použití přiblížení dlaždic

6.3 Publikace rozšíření

Po dokončení vývoje a otestování rozšíření je pravděpodobné, že budeme chtít publikovat svoje dílo ostatním lidem. Jak Google, tak Mozilla provozují své vlastní distribuční stránky: Google Web Store[1] a Add-ons for Firefox[2] (častěji označováno jako AMO), kde mohou vývojáři publikovat a uživatelé prozměnu vyhledávat doplňky.

Nikdo nás nenutí svá rozšíření veřejně publikovat, ať už v obchodě Google Web Store či AMO.

Nicméně, pokud se rozhodneme nepublikovat, nebudeme schopni do Chromu distribuovat své rozšíření. Jediná oficiální cesta je totiž přes jejich obchod[3]. Nahráním rozšíření na distribuční portál Chromu se provede kontrola interních procesů a v přídě nálezů zranitelných částí, je rozšíření vyřazeno ze schvalovacího procesu, jenž musí být úspěšně absolvováno, aby rozšíření obdrželo digitální podpis. Teprve až s podpisem je může být vystaveno v obchodě.

V případě Firefoxu jsou podmínky poněkud volnější, ovšem i v tomto případě je nutné rozšíření nechat prověřit, schválit a podepsat dřív, než jej budeme moci distribuovat vlastní či oficiální cestou.

[1] `https://chrome.google.com/webstore/category/extensions`
[2] `https://addons.mozilla.org/cs/firefox/`
[3] Existují i jiné cesty, jak rozšíření do Chromu nainstalovat, všechny ale potřebují přídavný podpůrný program, který instalaci provede na pozadí bez nutnosti asistence uživatele.

Postup pro zveřejnění rozšíření vypadá zhruba takto:

- Zabalit rozšíření do *.zip* souboru.
- Vytvořit účet na Google Web Store nebo AMO.
- Nahrát *.zip* soubor do obchodu kvůli verifikaci a digitálnímu podpisu.
- Opravit případné chyby.
- Rozhodnout se, zda bude rozšíření publikováno v obchodě.

Avšak v mém případě nastává problém už při registraci v obchodě jako vývojář. Google vyžaduje od nově se registrujících členů, aby zaplatili vstupní poplatek, který má za úkol snížit počet vývojářů, kteří by chtěli opakovaně nahrávat infikovaná rozšíření pod falešnými účty. Při odhalení rozšíření úmyslně využitého k infikaci počítače jak Google, tak Mozilla trvale zablokují účet, jenž byl jeho autorem. Zároveň s tím je poplatek i jakási obrana před tzv. boty[1], kteří by jinak zahlcovali servery registracemi účtů.

Zároveň s tím se vyskytly komplikace s licencemi, jejichž výhradním majitelem je autor původního doplňku „Seznam Lištička“. Navíc, publikováním rozšíření v oficiálních obchodech vstupuje do hry ještě další hráč - vlastník portálů Google Web Store a AMO.

Právě z těchto dvou důvodů jsem se rozhodl pro nepublikování rozšíření na oficiálních obchodech Chromu a Firefoxu. Výsledné práci to nijak neuškodilo a díky tomu, že byla navrhnuta jako výzkumný akademický projekt, nebylo nikdy v plánu její výsledek jakkoliv zpeněžit.

6.4 Budoucnost WebExtensions

WebExtensions je ve světě rozšíření a doplňků relativně nová technologie, přesto si již stihla získat na svou stranu zástup příznivců vítající především možnost přenositelnosti mezi prohlížeči, a zároveň s tím i nemalou skupinu odpůrců kritizující značně omezenou svobodu. Pokud bychom chtěli v dobách dávno minulých (dobře, ne tak dávných, ale technologie se dnes mění neuvěřitelnou rychlostí) vyvinout rozšíření do více prohlížečů zároveň, museli bychom pro každý prohlížeč vytvořit vlastní - pro Chrome, Firefox, Operu, Safari a Internet Explorer. Asi si dokážeme spočítat, že to byla nemalá překážka pro všechny, kteří chtěli do prohlížeče přidat vlastní funkci.

Není to tak dávno, co po vzoru Chromu a Opery oznámila Mozilla totální upuštění od zastaralých technologií a přechod čistě na WebExtensions. Netrvalo dlouho, a Microsoft oznámil podporu WebExtensions ve svém novém prohlížeči, nástupci Internetu Exploreru, Edgi. Nutno podotknout, že Microsoft již zcela upustil od veškerého vývoje vylepšení pro IE a nelze tedy v budoucnu očekávat podpora WebExtensions.

To všechno výrazně přispělo ke snížení úsilí potřebného k vytvoření rozšíření přenositelného mezi více prohlížeči.

Je nutné si uvědomit, že Apple zůstává, tak jako vždy, daleko od všech těch příjemných věcí, které se ve světě technologií odehrávají a nadále zůstává ve svém izolovaném světě integrace. Vývoj rozšíření pro Safari vyžaduje speciální registraci do programu vývojářů Applu a naučit se jazyky Objective-C nebo Swift. Na druhou stranu, díky tomu nejsme omezeni jen na jedinou platformu od Applu, ale můžeme vyvíjet i pro ostatní platformy. Na podporu WebExtensions však Apple stále nepřistoupil.

[1] Internetový robot , který pro svého majitele opakovaně vykonává nějakou rutinní činnost na internetu - obvykle sbírá data, zpracovává a odesílá požadavky na služby vzdálených serverů.

Dalším problémem se může jevit i rychlost implementace dedikovaných JavaScriptových API v prohlížeči Edge, jenž je nejmladším přírůstkem do rodiny podporující WebExtensions. Microsoft aktivně přispívá k vývoji chybějících API, bohužel se ale rozchází v aplikaci některých klíčových funkcí v porovnání s ostatními prohlížeči. Navíc v kombinaci s nedostatečnou dokumentací vlastních úprav API a malou oblibou tohoto prohlížeče se jeví úsilí věnovaného úpravám kódu pro Edge zbytečné.

Kapitola 7
Závěr

Cílem této diplomové práce bylo analyzovat možnost přechodu mezi technologiemi pro tvorbu rozšíření ve Firefoxu a aplikovat nabyté znalosti na již existujícím doplňku. Jedním z hlavních požadavků bylo, aby nově vznikající aplikace splňovala vlastnosti tak zvaného „cross-browser“ rozšíření, tedy rozšíření postaveného na množině technologií podporovaných ve více prohlížečích zároveň. Práce se měla věnovat i otázce zabezpečení před zneužitelností rozšíření a zhodnocení, zda došlo během přepisu ke kvalitativnímu zlepšení.

Už během absolvování předmětu *Softwarový nebo výzkumný projekt* jsem prostudoval dostupné technologie a vyhodnotil možnosti jejich použití během tvorby diplomové práce. Zároveň s tím jsem poznal základy technologie WebExtensions, kterou byla posléze využita při tvorbě rozšíření nového. Popis dostupných technologií, jejich historie, výhody i nevýhody jsou shrnuty v kapitole č. 2.

Posléze jsem porovnal dvě vybrané technologie, množinu XUL/XPCOM, jenž je použita v zastaralém rozšíření „Seznam Lištička“, které bylo určeno k přepisu, a WebExtensions, jenž je považována za moderního nástupce ostatních technologií a zároveň jako jediná splňuje podmínku přenositelnosti mezi prohlížeči. Zároveň s tím jsem provedl analýzu zabezpečení ve WebExtensions v porovnání s XUL/XPCOM, ze kterého vyplynulo, že přechodem na modernější technologii došlo k výraznému zlepšení zabezpečení dat uživatele za cenu omezení přístupu rozšíření k funkcím prohlížeče.

V praktické části, kapitola č. 5, jsem se zaměřil na výběr technologií, frameworků a knihoven usnadňující vývoj rozšíření postavené na WebExtension a zároveň demonstroval jejich využití na vlastním portu již existujícího doplňku pro prohlížeč Firefox. Důraz jsem kladl především na zjednodušení a zrychlení vývoje i jejich následné údržby.

Nakonec jsem v kapitole č. 6 provedl zhodnocení technologického posunu, včetně návrhu na možná zlepšení a dopracování.

Během práce byly dodrženy všechny hlavní body, které měla práce splňovat, až na úkol, kde měla být provedena publikace nových rozšíření na oficiálních distribučních portálech *Chrome Web Store* a *Add-ons for Firefox*. K rozhodnutí nepublikovat rozšíření jsem dospěl po konzultacích týkajících se licenčních podmínek již existujících rozšíření.

Literatura

[1] Evernote Corporation. *Evernote Web Clipper*. 2017.
https://evernote.com/webclipper/.

[2] Rodrigo Siqueira. *JoinTabs*. 2017.
https://tinyurl.com/ke6u8zv.

[3] appbeat.com. *Pricescout for Google Chrome*. 2017.
https://tinyurl.com/mmft53v.

[4] Google. *Chrome Web Store*. 2017.
https://chrome.google.com/webstore/category/extensions.

[5] Mozilla. *Firefox addons*. 2017.
https://addons.mozilla.org/cs/firefox/extensions/.

[6] Mozilla. *How many Firefox users have add-ons installed?* 2017.
https://tinyurl.com/khj6hxg/.

[7] Google. *Chromium Blog*. 2015.
https://tinyurl.com/m8n9cz3/.

[8] Mozilla. *Mozilla Add-ons Blog*. 2017.
https://blog.mozilla.org/addons/2016/11/23/add-ons-in-2017/.

[9] appbeat.com. *Browser Extensions*. 2017.
https://msdn.microsoft.com/en-us/library/aa753587(VS.85).aspx.

[10] Mary Jo Foley. *Microsoft releases first Edge extensions preview in newest Windows 10 test build*. 2017.
http://www.zdnet.com/article/microsoft-releases-first-edge-extensions-preview-in-newest-windows-10-test-build/.

[11] Mozilla Developer Network. *Install Manifests*. 2017.
https://developer.mozilla.org/en-US/Add-ons/Install_Manifests.

[12] Mozilla Developer Network. *Chrome registration*. 2017.
https://developer.mozilla.org/en-US/docs/Chrome_Registration.

[13] Mozilla Developer Network. *WebExtension*. 2017.
https://tinyurl.com/jzofes7.

[14] Mozilla Developer Network. *Loading content scripts*. 2017.
https://tinyurl.com/lat52sx.

[15] Mozilla Developer Network. *Toolbar button*. 2017.
https://developer.mozilla.org/en-US/Add-ons/WebExtensions/Browser_action.

[16] Lucian Constantin. *Researcher to demonstrate feature-rich malware that works as a browser extension*. 2017.
http://www.computerworld.com/article/2492866/desktop-apps/researcher-to-demonstrate-feature-rich-malware-that-works-as-a-browser-extension.html.

[17] msft-mmpc. *Browser extension hijacks Facebook profiles*. 2017.
https://blogs.technet.microsoft.com/mmpc/2013/05/10/browser-extension-hijacks-facebook-profiles/.

[18] msft-mmpc. *Browser extension hijacks Facebook profiles*. 2017.
https://blogs.technet.microsoft.com/mmpc/2013/05/10/browser-extension-hijacks-facebook-profiles/.

[19] Google. *chrome.webRequest*. 2017.
https://developer.chrome.com/extensions/webRequest.

[20] Mozilla Developer Network. *Content Security Policy*. 2017.
https://tinyurl.com/jjgsbf7.

[21] World Wide Web Consortium. *HTML5 differences from HTML4*. 2017.
https://www.w3.org/TR/2011/WD-html5-diff-20110405/.

[22] Stoyan Stefanov. *JavaScript Patterns*. 1. vydání. O'Reilly Media, Inc., 2010. ISBN 9781449396947.

[23] World Wide Web Consortium. *A Short History of JavaScript*. 2017.
https://www.w3.org/community/webed/wiki/A_Short_History_of_JavaScript.

[24] Paul Krill. *It's official: ECMAScript 6 is approved*. 2017.
https://tinyurl.com/mxqjfa3.

[25] Miloslav Mrvík. *es6 class vs prototype*. 2017.
https://jsperf.com/es6-class-vs-prototype.

[26] IBM. *Javascript everywhere and the three amigos*. 2017.
https://tinyurl.com/l59uy2r.

[27] Wikipedia. *Framework*. 2017.
https://cs.wikipedia.org/wiki/Framework.

[28] Rudolf PECINOVSKÝ. *Návrhové vzory: 33 vzorových postupů pro objektové programování*. 1. vydání. Computer Press, 2007. ISBN 9788025115824.

[29] AngularJS. *Templates*. 2017.
https://docs.angularjs.org/guide/templates.

[30] Google. *Google Developers Site*. 2017.
https://developer.chrome.com/extensions/api_index.

[31] Mozilla. *Mozilla Developer Network*. 2017.
https://developer.mozilla.org/en-US/Add-ons/WebExtensions.

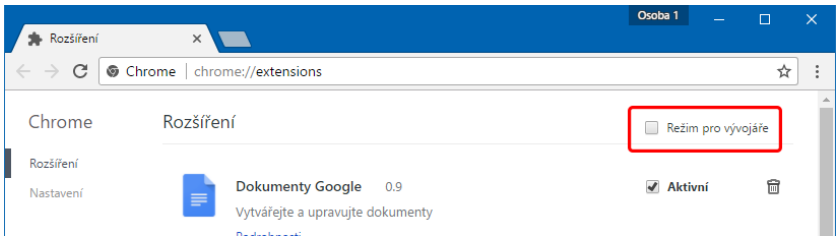
[32] Wikipedia. *Drag and Drop*. 2017.
https://cs.wikipedia.org/wiki/Drag_and_drop.

Příloha A
Zkratky a symboly

HTML	**HyperText Markup Language** je značkovací jazyk používaný pro tvorbu webových stránek
CSS	**Cascading Style Sheets** je jazyk pro popis způsobu zobrazení elementů na stránkách napsaných v jazycích HTML, XHTML nebo XML.
JS	**JavaScript** je multiplatformní, objektově orientovaný skriptovací jazyk.
NPAPI	**Netscape Plugin Application Programming Interface** je multiplatformní architektura modulů používána řadou webových prohlížečů.
BHO	**Browser Helper Object** je DLL modul navržený jako doplněk pro IE.
IE	**Internet Explorer**
XUL	**XML User Interface Language**
XML	**Extensible Markup Language** je obecný značkovací jazyk.
XPCOM	**Cross Platform Component Object Model**
CORBA	**Common Object Request Broker Architecture**
API	**Application Programming Interface** označuje rozhraní pro programování aplikací nebo rozhraní aplikací vystavené.
SDK	**Software Development Kit** je sada vývojových nástrojů umožňující vytváření aplikací pro softwarové balíčky.
JSON	**JavaScript Object Notation** je datový formát, určený pro přenos dat.
URI	**Uniform Resource Identifier** je textový řetězec s definovanou strukturou, který slouží k přesné specifikaci zdroje informací.
CSP	**Content Security Policy**, nebo-li politika zabezpečení obsahu.
RDF	**Resource Description Framework** je rodina specifikací W3C původně navržená jako datový model metadat.
W3C	**World Wide Web Consortium** je hlavní mezinárodní organizace stanovující standardy pro World Wide Web.
UUID	**Universally Unique Identifier** je 128-bitové číslo použité k identifikaci informací v počítačových systémech.
AMO	**addons.mozilla.org**
MDN	**Mozilla Developer Network**
DTD	**Document Type Definition**
GUI	**Graphical User Interface** je uživatelské rozhraní, které umožňuje ovládat počítač pomocí interaktivních grafických ovládacích prvků.
SVG	**Scalable Vector Graphics** je značkovací jazyk a formát souboru, který popisuje dvojrozměrnou vektorovou grafiku pomocí XML.
MVVM	**Model View View-Model** usnadňuje oddělení vývoje grafického uživatelského rozhraní - ať už prostřednictvím značkovacího jazyka nebo GUI kódu - od vývoje obchodní logiky nebo back-end logiky.
UX	**User Experience** je sada zásad, metod a technik pro návrh čehokoli – od designu boty až po sofistikovaná uživatelská rozhraní počítačového softwaru.

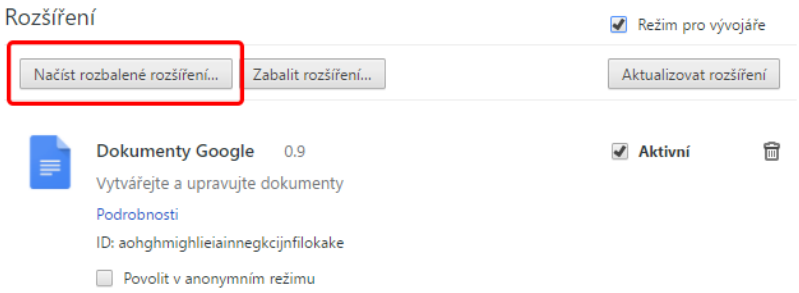
Příloha B
Instalační příručka pro Chrome

1. Prvně musíme stáhnout soubor s příponou *.zip* nebo *.crx* a extrahovat jeho obsah.
2. Jakmile máme rozšíření rozbalené, přejdeme na adresu **chrome://extensions** nebo pomocí **Menu - Další nástroje - Rozšíření** a následně aktivovat vývojářský režim:



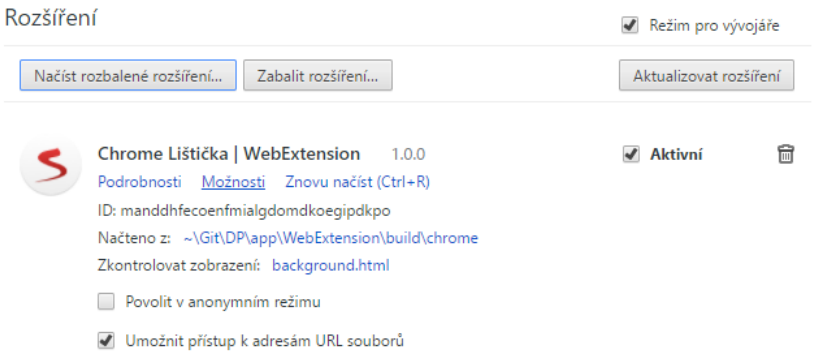
Obrázek B.1. Instalace WebExtensions v Chromu, 01

3. Potom musíme specifikovat, kde v počítači se soubory nacházejí:



Obrázek B.2. Instalace WebExtensions v Chromu, 02

4. Doplněk máme nainstalovaný a můžeme jej používat.

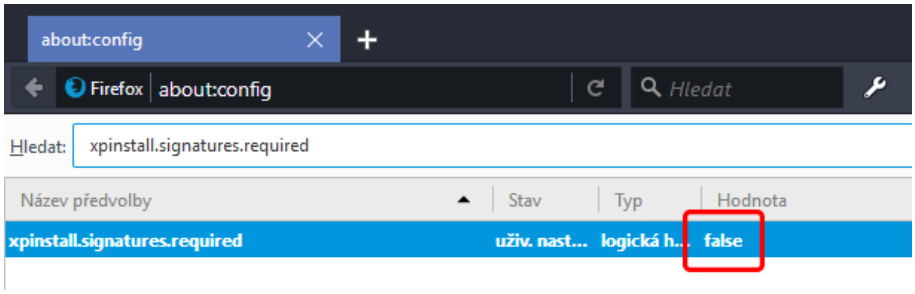


Obrázek B.3. Instalace WebExtensions v Chromu, 03

Příloha C
Instalační příručka pro Firefox

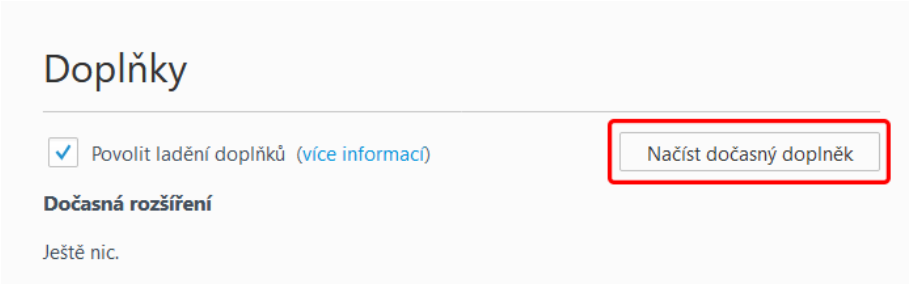
Instalace WebExtensionss

1. Pro instalování nepodepsaných doplňků musíme nainstalovat **Firefox Developer Edition v50+**. Viz `https://www.mozilla.org/cs/firefox/developer/`.
2. Jakmile máme Firefox nainstalovaný, musíme opět rozbalit rozšíření.
3. Následně přejdeme na adresu **about:config** a vyhledáme `xpinstall.signatures.required`, u kterého nastavíme logickou hodnotu na **false**:



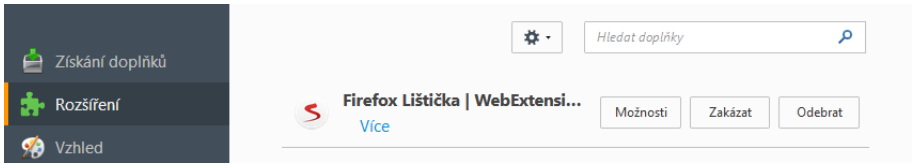
Obrázek C.4. Instalace WebExtensions ve Firefoxu, 01

4. Po tom, co povolíme instalce nepodepsaných doplňků, přejdeme na adresu **about:debugging**, kde načteme dočasný doplněk:



Obrázek C.5. Instalace WebExtensions ve Firefoxu, 02

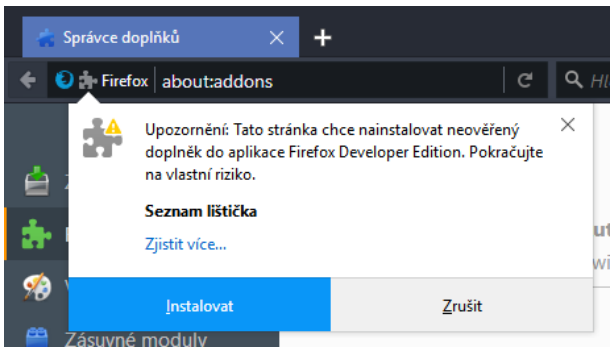
5. Označíme soubor `manifest.json` a klikneme otevřít.
6. Nainstalovaný doplněk můžeme najít na adrese **about:addons**.



Obrázek C.6. Instalace WebExtensions ve Firefoxu, 03

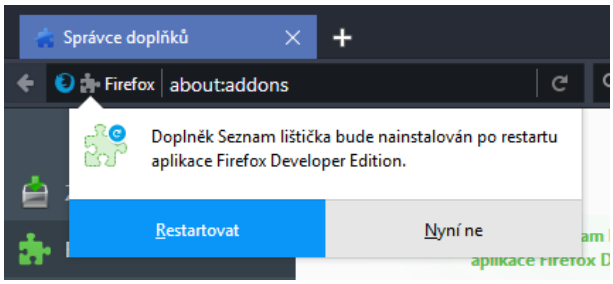
Instalace XUL/XPCOM

1. Pro instalování nepodepsaných doplňků musíme nainstalovat **Firefox Developer Edition v50+**. Viz `https://www.mozilla.org/cs/firefox/developer/`.
2. Následně přejdeme na adresu **about:config** a do vyhledávacího pole zadáme `xpinstall.signatures.required`, kde nastavíme logickou hodnotu na **false**.
3. Následně přejdeme na adresu **about:addons**, kam přetáhneme soubor s doplňkem, který končí na *.xpi*. Nebo můžeme *Nainstalovat doplněk ze souboru* přes ozubené kolečko ve stránce s rozšířeními.
4. Povrdíme instalaci doplňku:



Obrázek C.7. Instalace XUL/XPCOM ve Firefoxu, 01

5. A restartujeme prohlížeč:

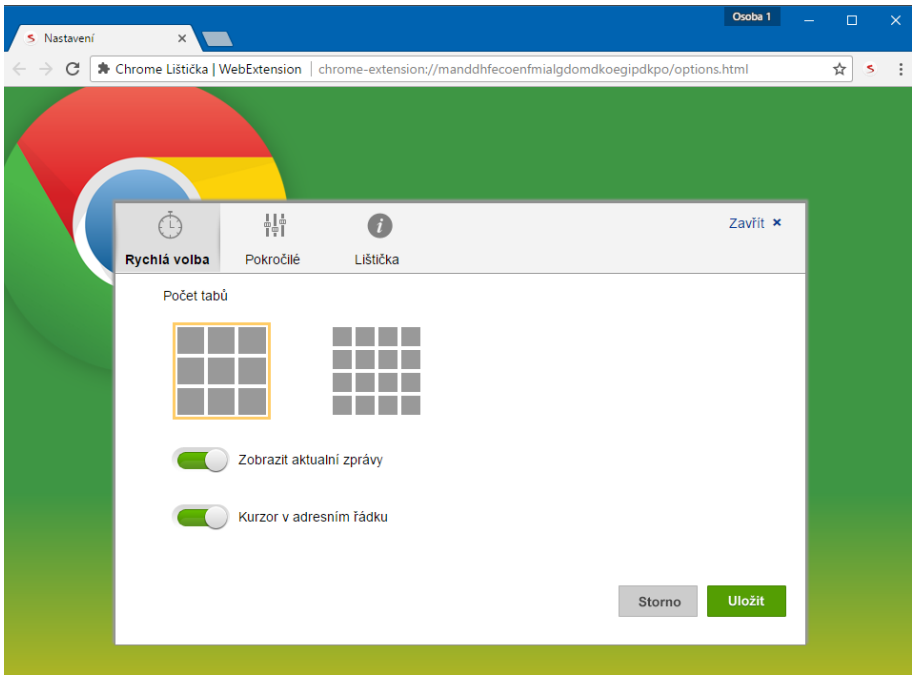


Obrázek C.8. Instalace XUL/XPCOM ve Firefoxu, 02

Příloha D
Uživatelská příručka

Vytvořené rozšíření je značně jednoduché a v zásadě obsahuje pouze stránku s nastavením rozšíření a stránku, která se zobrazí při otevření nového panelu. Pro jeho správnou funkčnost musí být povolený JavaScript v prohlížeči.

Nastavení rozšíření



Obrázek D.9. Instalace XUL/XPCOM ve Firefoxu, 02

Do nastavení rozšíření se můžeme dostat přes:

- link umístěný vpravo nahoře v novém panelu,
- pravým klikem na ikonu rozšíření a přejití na **Možnosti**
- nebo přes stránku s rozšířeními prohlížeče opět přes link **Možnosti** pod nazvem rozšíření.

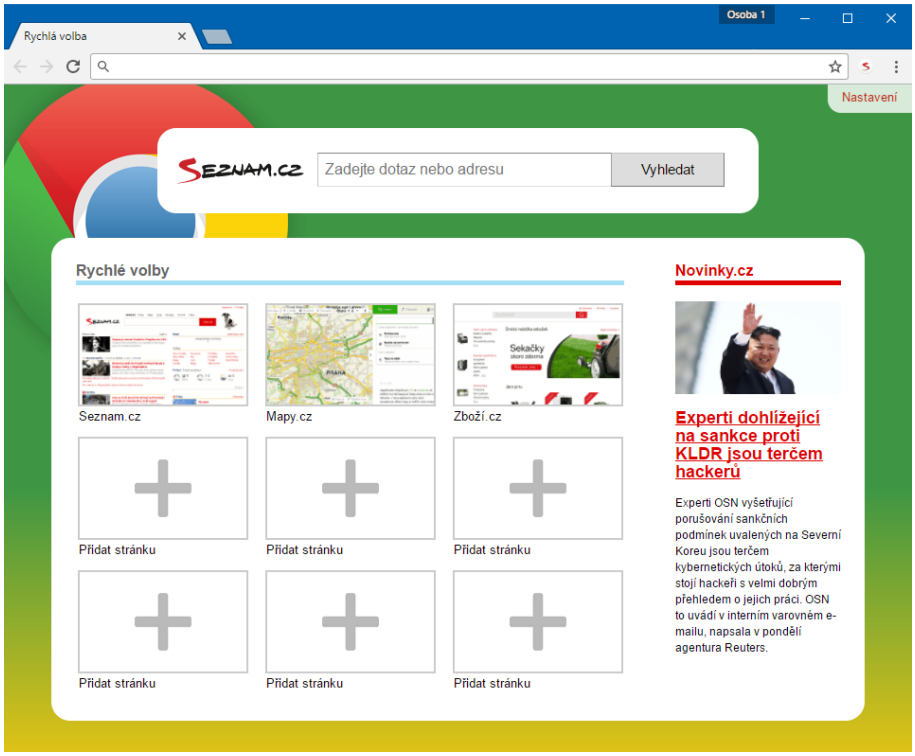
V nastavení jsou 3 záložky.

1. Rychlá volba mění vzhled a rozložení šachovnice odkazů v rychlé volbě. Zde máme na výběr z 9 nebo 16 panelů. Dále můžeme vypnout/zapnout zprávy z Novinky.cz nebo nastavit focus[1] do vyhledávacího pole, které je součástí stránky, při otevření nového panelu. Bohužel je tato možnost v Chromu trvale vypnutá (v doplňku typu XUL tato volba byla prohlížečem povolena).

[1] Zaměření

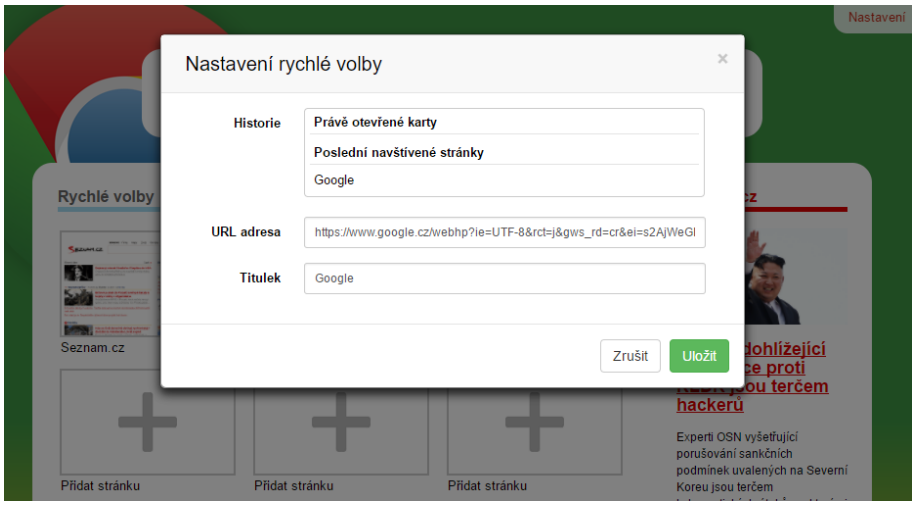
2. Pokročilé obsahují opět nastavení focusu do stránky, které je taktéž zakázaná, a možnost otevření nové stránky Seznam.cz při otevření prohlížeče.
3. Lištička obsahuje autora, licence a detaily týkající se lištičky.

Nový panel



Obrázek D.10. Nový panel

Hlavní část obsahuje Rychlé volby a zpravodajský panel. Ve výchozím stavu má rychlá volba 9 dlaždic, které můžou být rozšířeny až na 16. Celý obsah se přizpůsobuje rozlišení obrazovky, dle předem definovaných rozměrů. Při kliknutí na již vytvořený odkaz nás doplněk přesměruje na danou URL adresu. Pokud ještě odkaz zadefinován nebyl, zobrazí se okno s výzvou pro zadání nového odkazu:



Obrázek D.11. Nový panel, popup

Můžeme si vybrat z předpřipravených rychlých voleb skládajících se z právě otevřených panelů a nedávné historie nebo můžeme zadat vlastní URL adresu a popisek. Posléze odkaz uložíme.

Obrázek D.12. Nový panel, detail

Všechny rychlé volby můžeme buďto smazat nebo upravit, přičemž se otevře okno jako při zadávání nové rychlé volby, avšak již s předvyplněnými daty.

Příloha E
Obsah přiloženého CD

app	Vytvořená aplikace
app/WebExtension	Rozšíření ve WebExtension
app/WebExtension/.jshintrc	Konfigurační soubor Gulp.js
app/WebExtension/gulpfile.js	Konfigurační soubor Gulp.js
app/WebExtension/package.json	Konfigurační soubor Node.js
app/WebExtension/build	Adresář sestavených rozšíření
app/WebExtension/build/chrome	
app/WebExtension/build/firefox	
app/WebExtension/css	CSS soubory používané v HTML
app/WebExtension/dist	Adresář s ubalenými rozšířeními připravených k distribuci
app/WebExtension/dist/chrome	
app/WebExtension/dist/firefox	
app/WebExtension/fonts	Adresář s písmy
app/WebExtension/html	Adresář s HTML soubory
app/WebExtension/img	Adresář s obrázky
app/WebExtension/js	Adresář se skripty pro běh aplikace
app/WebExtension/js/libs	Knihovny třetích stran
app/WebExtension/node_modules	Importované knihovny Node.js použité pro sestavování
app/WebExtension/tools	Podporné nástroje pro distribuci aj.
app/WebExtension/vendor_specific	Soubory a skripty specifické pro prohlížeč
app/WebExtension/vendor_specific/chrome	
app/WebExtension/vendor_specific/firefox	
app/XUL	Rozšíření v XUL/XPCOM
app/XUL/XUL.xpi	Ubalené rozšíření
app/XUL/chrome	
app/XUL/chrome/content	Soubory s uživatelským rozhraním (XUL) a soubory skriptů (JS)
app/XUL/chrome/locale	Definuje jazykové sady rozšíření
app/XUL/chrome/skin	Definuje vzhled uživatelského rozhraní
app/XUL/modules	Moduly se skripty
app/XUL/modules/classes	Vlastní třídy nutné pro běh
app/XUL/modules/components	Podpůrné funkce a třídy, doplňující funkcionalitu
app/XUL/modules/JAK	JavaScriptový framework od Seznam.cz
text	Text diplomové práce
text/main.pdf	Hlavní PDF soubor s DP
text/img	Obrázky použité v DP