VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

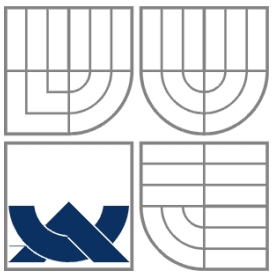
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

WEBOVÁ APLIKACE V DESKTOPOVÉM PROSTŘEDÍ

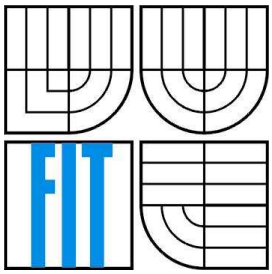
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE MAREK BUGÁŇ
AUTHOR

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

WEBOVÁ APLIKACE V DESKTOPOVÉM PROSTŘEDÍ

WEB APPLICATION IN DESKTOP ENVIRONMENT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MAREK BUGÁŇ

VEDOUCÍ PRÁCE
SUPERVISOR

MGR. ROMAN TRCHALÍK PH.D.

BRNO 2014

Abstrakt

Tato bakalářská práce by měla ukázat, jak se dá pracovat s webovými aplikacemi v destkopovém prostředí. Jsou zde popsány technologie, které se dají požít pro vývoj těchto aplikací. Jádrem práce je popsat a demonstrovat vývoj aplikací v prostředí Google Chrome API.

Abstract

This bachelor thesis shoul show us, how to work with web application in desktop environment. It show us, wthch technologies we can use for development of this kind of application. The aim of thesis is to describe and demonstrate the development of applications in Google Chrome API.

Klíčová slova

Webová apilikace, Google Chrome, Chromiu, Cappuccino prostředí, APE-projekt, TideSDK, package apliakce

Keywords

Web application, Choogle Chrome, Chromum, Cappuccino framework, APE-project, TideSDK, package application

Citace

Marek Bugáň: Webová aplikace v desktopovém prostředí, bakalářská práce, Brno, FIT VUT v Brně, 2014

Webová aplikace v desktopovém prostředí

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením mgr. Romana Trchalíka Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Marek Bugáň
21.5.2014

Poděkování

Chtěl bych poděkovat svému vedoucímu mgr. Romanu Trchalíku Ph.D. za odborné vedení mé práce.

Obsah

1 Úvod

V dnešní době se začaly rozšiřovat webové aplikace. Setkáme se s nimi s nimi téměř na každém kroku, kupříkladu jsou velmi rozšířené na sociálních sítích. Webové aplikace mají jednu velkou nevýhodu. Bez internetového připojení nefungují. Proto se začínají vyvíjet aplikace v dektopových prostředích a z webových aplikací se dělají aplikace nativní. Ty ke svému fungování internetové připojení nezbytně nepotřebují. Využívají ho, pokud existuje ta možnost, ale i při jeho absenci se zachována ta funkčnost, která připojení nepotřebuje.

V následujícím textu si ukážeme některé technologie, které umožňují vývoj webových aplikací v dektopovém prostředí. Podíváme se na projekt APE (Ajax Push Engine), dále na Cappuccino Web Framework, TideSDK platformu a na Google Chrome API. Stručně si probereme jejich silné a slabé stránky a s jednou z těchto aplikací se seznámíme podrobněji. Technologii, kterou jsem si vybral pro další práci, je Google Chrome API.

V práci se seznámíme s tím, co vše musíme udělat, abychom mohli pracovat s touto technologií. Uvedeme si, jak vypadá práce v prostředí Google Chrome a popíšeme zabezpečení této technologie. Se zabezpečením se váží i jistá omezení, kterým se budeme věnovat a zmíníme se o tom, co díky omezením nelze v tomto prostředí vytvořit. Ukážeme si vzorové aplikace, které nastíní, jak lze s webovými aplikacemi v desktopovém prostředí pracovat a jaké zdroje k tomu můžeme využít. I ve vzorových aplikacích narazíme na omezení. Na závěr se pokusím nastínit, jaká by mohla vypadat budoucnost těchto technologií.

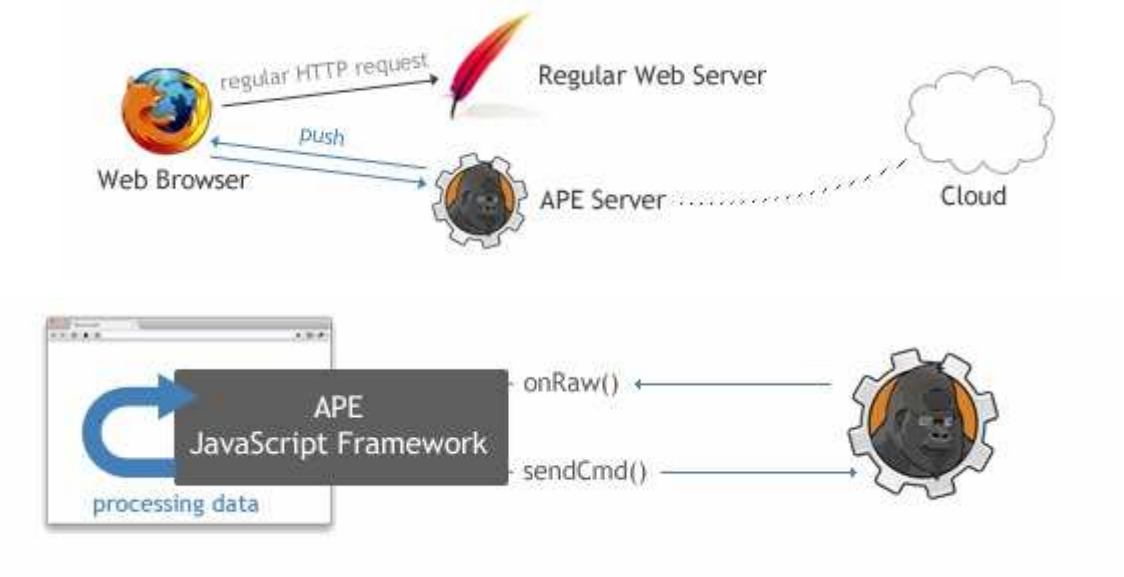
2 Výběr Technologie

2.1 Ajax Push Engine Project (APE)

Ajax Push Engine je open source technologe umožňující tvorbu webových aplikací. Tato technologie je rozdělena na dvě základní části. První část tvoří APE Server, druhou částí je APE JavaSkript Framework. Komunikace mezi těmito entitami probíhá formou „klient-server“. Zdrojové kódy pro APE Server jsou napsány v programovacím jazyce C. APE Server implementuje funkce GET a POST, které známe z klasických webových serverů, nicméně webové servery nenahrazuje. APE server lze spustit pouze pod operačními systémy Linux, MacOS nebo FreeBSD. Pokud chceme technologii APE využívat v prostředí systému Windows, je zapotřebí zprovoznit

virtuální stroj, na kterém běží jiný operační sytém (Linux nebo MacOS). V prostředí APE pracujeme se základními jazyky pro tvorbu webu (JavaScript, HTML a CSS). [1]

Schéma práce APE:



obr. 2.1 schéma páce APE technologie [1]

Výhody a nevýhody APE technologie:

Výhody:

- Možnost použití JavaScript, HTML, CSS
- Open source technologie
- Spustitelnou v internetových prohlížečích. Mozilla Firefox, Google Chrome, Opera, Internet Exploer
- Plně asynchronní režim

Nevýhody:

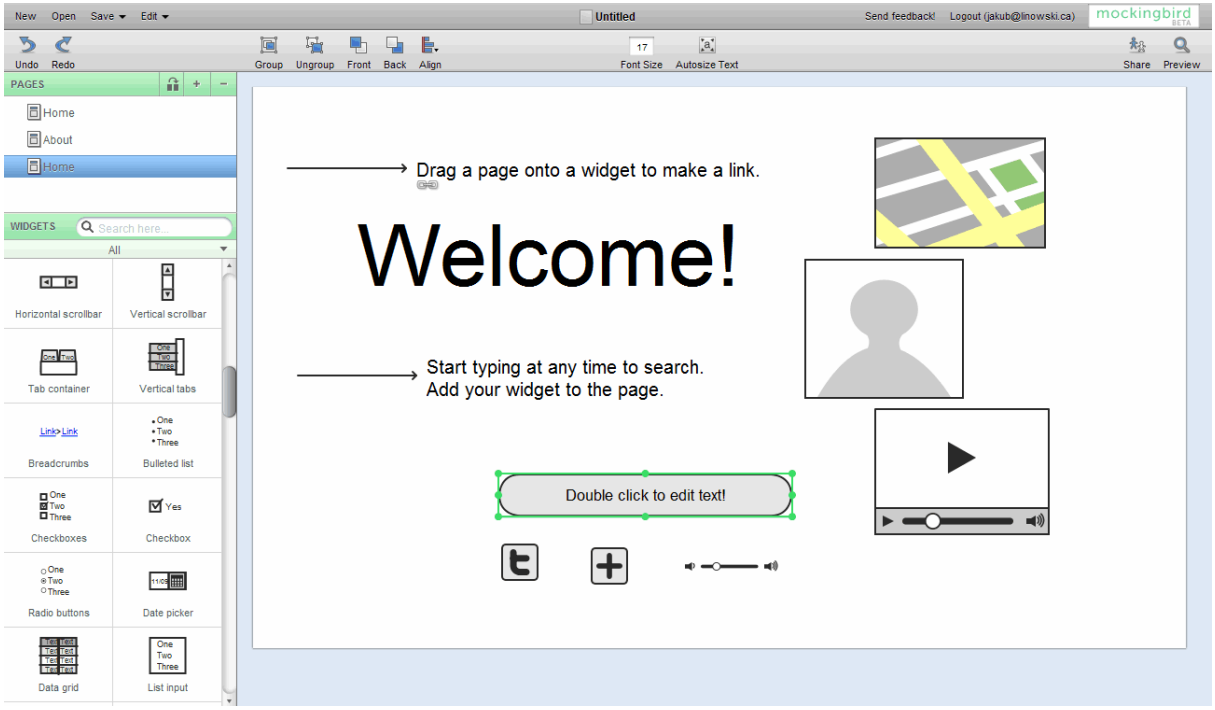
- Nemožnost práce v operačním systému Windows
- Nutnost konfigurace APE Serveru
- Nutnost přístupu k DNS serveru

2.2 Cappuccino Web Framework

Cappuccino Web Framework se řadí mezi plně Open source systémy pro tvorbu webových aplikací v desktopovém prostředí. Cappuccino je samostatný program s vlastním editorem, který uživateli zjednodušuje práci v daném prostředí. Cappuccino je vytvořeno v programovacím jazyce Objective-J a rovněž tento jazyk používá pro tvorbu aplikací. Cappuccino bylo navrženo tak, aby

[1] APE project. *APE project* [online]. [cit. 2014-05-09]. Dostupné z: http://ape-project.org/

uživateli umožňovalo jednoduchou práci při tvorbě webových aplikací. Prostředí cappuccina se zakládá na technologii Cocoa, ve kterém můžeme vyvíjet aplikace pod operačním systému MacOS a IOs. Cappuccino není závislé na webovém prohlížeči. Aplikace vytvořené v tomto prostředí jsou spustitelné ve všech prohlížečích, které podporují Objectiv-J. (Internet Explorer 8 a výše, Mozilla Firefox 2 a výše, Safari 3 a výše. Google Chrome, Opera 9 a výše). Technologie Cappuccino je vhodná pro vytváření různých klientů (např. emailových) a lze ji použít při tvorbě real-time aplikací, které vyžadují interakci s uživatelem. [2]



obr.2.2 Vhled Cappuccina

Výhody a nevýhody Cappuccina

Výhody:

- OpenSource technologie
- Spustitelnost ve většině webových prohlížečů
- Zaměřen především na vývoj webových aplikací

Nevýhody:

- Nemožnost využití jiného programovacího jazyka kromě Objectiv-J

2.3 TideSDK

TideSDK je vývojový prostředek pro vytváření multiplatformních desktopových aplikací za použití webových technologií. Pro vývoj aplikace užíváme programovací jazyky HTML5, CSS3 a

[2] Cappuccino. *Cappuccino-project* [online]. [cit. 2014-05-09]. Dostupné z: http://www.cappuccino-project.org/

JavaScript. Pro práci však lze využít i jiné jazyky jako např. Phyton, PHP, či Ruby. Aplikaci můžeme bez větších obtíží spustit v operačních systémech Windows, MacOS, či různých distribucích Linuxu. Abychom mohli pracovat s TideSDK, vývojové prostředí TideSDK musí být v operačním systému uloženo na předem specifikovaném místě.

Pro MacOS je to:

```
~/Library/Application Support/TideSDK
```

V Linuxu:

```
~/.tidesdk
```

Ve Windows XP:

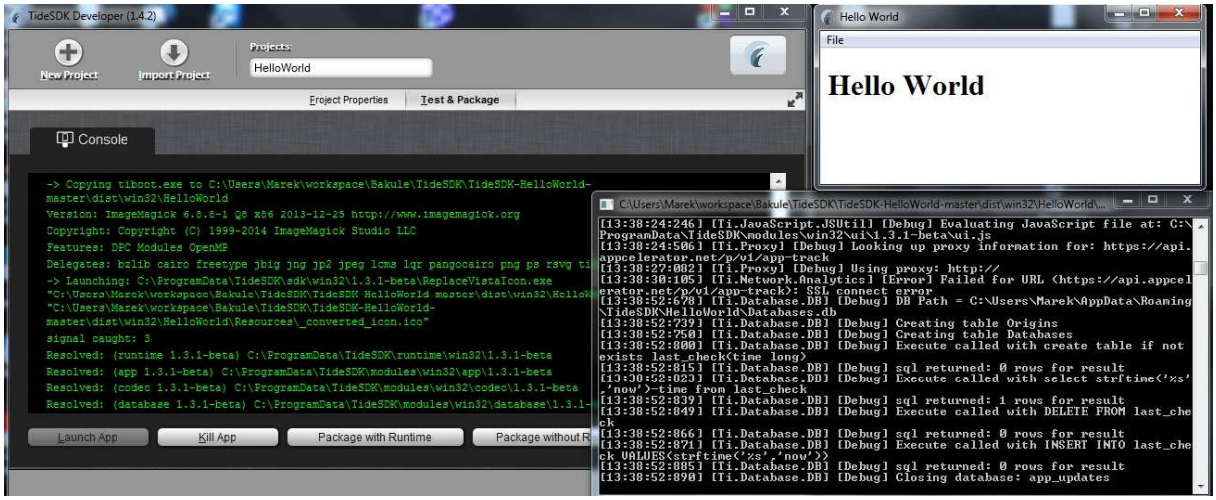
```
C:\Documents and Settings\All Users\Application Data\TideSDK
```

Ve Windows 7:

```
C:\ProgramData\TideSDK
```

V operačním systému Windows je rovněž potřeba nainstalovat aplikace Imagemagick a Wix 3.0

Pokud by aplikace Imagemagic a Wix 3.0 nainstalovány nebyly, TideSDK by nefungoval. [3]



obr.2.3 Vzhled prostředí TideSDK

Výhody a nevýhody TideSDK:

Výhody:

- OpenSource technologie
- Podpora mnoha programovacích jazyků (Phyton, Ruby, PHP,…)
- Multiplatformní technologie.
- Vlastní vývojová aplikace(editor)

Nevýhody:

- Potřeba program nainstalovat na předem určené místo

[3] *TideSDK* [online]. 2012 [cit. 2014-05-09]. Dostupné z: http://www.tidesdk.org/

2.4 Google Chrome API

Google Chrome je prostředí, kde pomocí webových prostředků vytváříme nativní aplikace, které fungují v offline i online režimu. Veškeré aplikace jsou spuštěny a pracují primárně v offline režimu. Pokud se lze připojit k síti, můžeme se při práci přepnout do režimu online. Důležité je však nezapomenout na fakt, že všechny zdrojové kódy musí být uložené lokálně. Google chrome je multiplatformí. Vše, co potřebujeme k vývoji a provozu aplikace, je prohlížeč Google Chrome. Google Chrome bohužel není projektem s open source licencí. Je to výsledek práce společnosti Google, která si chrání část zdrojových kódů. Nicméně Google vypustil open source projekt s názvem Chromium, což je prohlížeč s plně open source licencí. Prohlížeč Chromium je založen na stejných zdrojových kódech jako Chrome. Oba prohlížeče poskytují stejnou základnu pro aplikace a funkcionalitou jsou si velice podobné. [4]

Výhody a nevýhody Google Chrome API:

Výhody:

- Uživatelská přívětivost
- Práce s webové technologie (HTML, CSS, JavaScript)
- Mnoho výukových materiálů
- Multiplatformní technologie

Nevýhody:

- Není open source

2.5 Shrnutí a výběr jedné technologie

V předchozí části bakalářské práce byly představeny vybrané technologií pro tvorbu webových aplikací v desktopovém prostředí. Nyní je potřeba učinit rozhodnutí, ve které ze zmíněných technologií budeme demonstrovat funkcionalitu aplikací. Pro posouzení vhodnosti jednotlivých technologií je potřeba porovnat silné a slabé stránky těchto technologií. Na základě vyhodnocení silných a slabých stránek byla jako první vyloučena technologie Ajax Push Engine. Hlavním důvodem k vyloučení této technologie bylo omezení APE serveru na operační systémy Linux a MacOS. Další nevýhodou technologie APE je nutnost mít připojení k DNS serveru. Druhou vyloučenou technologií bylo Cappuccino Web Framework. Důvodem k vyloučení bylo, že v prostředí lze pracovat pouze v jazyce Objectiv-J. Poslední dva kandidáti, mezi kterými je potřeba zvolit, jsou Google Chrome API a TideSDK. Obě platformy jsou uživatelsky přívětivé, velmi propracované a založené na stejných webových technologiích (HTML, CSS, JavaScript). TideSDK je velmi dobrá

[4] *Chrome APP* [online]. [cit. 2014-05-09]. Dostupné z: https://developer.chrome.com/apps

technologie. TideSDK umožňuje vyvíjet aplikace v uživatelsky přívětivém prostředí. Přestože TideSDK nemá žádné zásadní nevýhody, rozhodl jsem se pro další práci využít technologii Google Chrome API. Jedním z hlavních důvodů mého rozhodnutí bylo, že tato technologie je zastřešená jednou z největších programátorských komunit. Mé rozhodnutí podpořil také fakt, že o Chrome API existuje velké množství výukových materiálů v podobě textů i v podobě názorných videí, ze kterých lze čerpat.

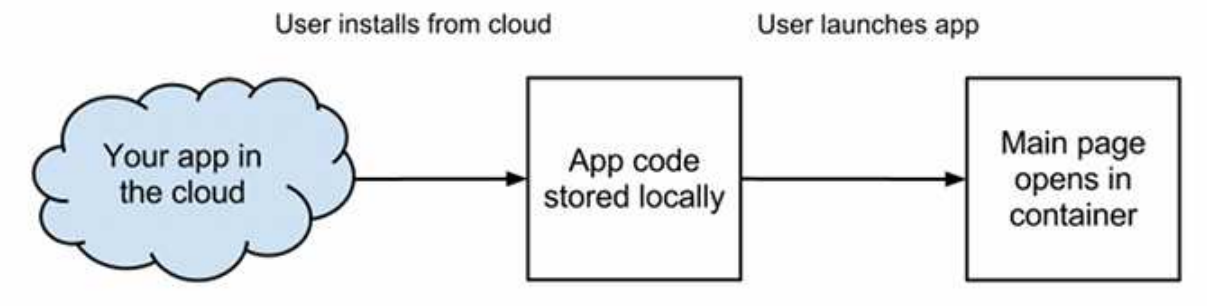
3 Chrome API

Chrome API je velmi propracovaná technologie pro tvorbu webových aplikací v desktopovém prostředí. Společnost Google věnovala velké úsilí vývoji a zveřejnila precizně propracovanou a zabezpečenou technologii. Díky této technologii lze vytvářet aplikace, které se chovají jako webové aplikace, vypadají jako webové aplikace, ale přitom se jedná o aplikace nativní. Ve srovnání s webovými aplikacemi je zde však značně rozšířená funkcionalita. U webových aplikací je nemyslitelné, aby přistupovaly např. k filesystému počítače. V aplikacích Chromu však tato možnost existuje. [5]

3.1 Architektura

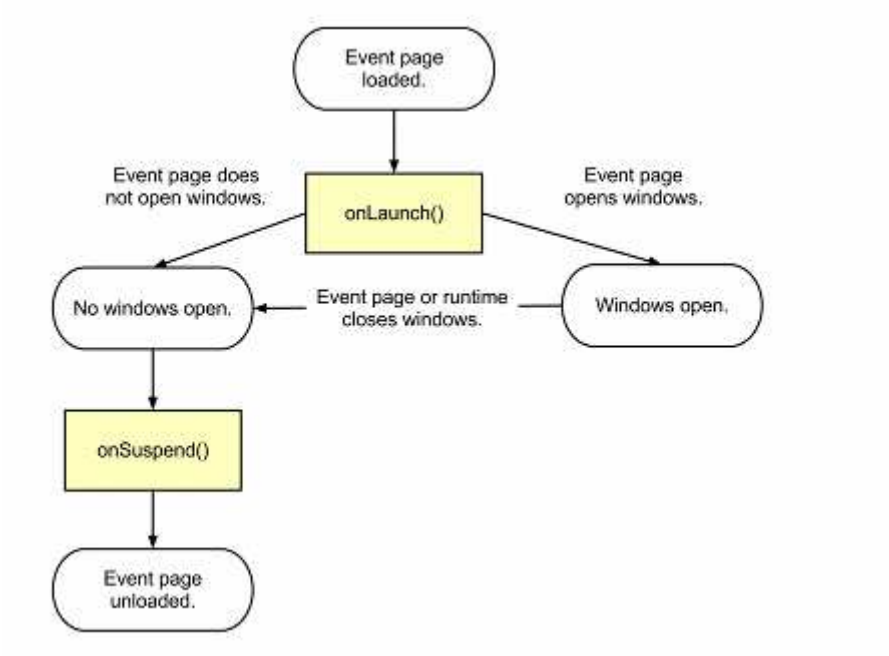
Aplikace Chromu jsou nevrženy tak, aby byly co nejméně závislé na prohlížeči a internetovém připojení. Rovněž velmi úzce spolupracují s operačním systémem. Pokud je srovnáme s webovými aplikacemi, najdeme mnoho rozdílů. Jedním z rozdílů mezi aplikacemi chromu a webovými aplikacemi je způsob, jakým se načítají. Oba typy aplikací načítají stejný obsah - HTML dokument rozšířený o CSS a JavaScript. Rozdíl je v tom, že webová aplikace obsah načte do prohlížeče, ale Chrome aplikace obsah načte do kontejneru. To znamená, že se důležité zdrojové kódy musí načíst z lokálního úložiště. Díky tomu aplikace vždy poskytuje alespoň minimální funkčnost i bez připojení k internetu.[6]

[5] *Chrome Apps Architecture* [online]. [cit. 2014-05-09]. Dostupné z: https://developer.chrome.com/apps/app_architecture



obr. 3.1 načítaní aplikace v Crhome

V následujícím diagramu je zobrazený životní cyklus aplikace.

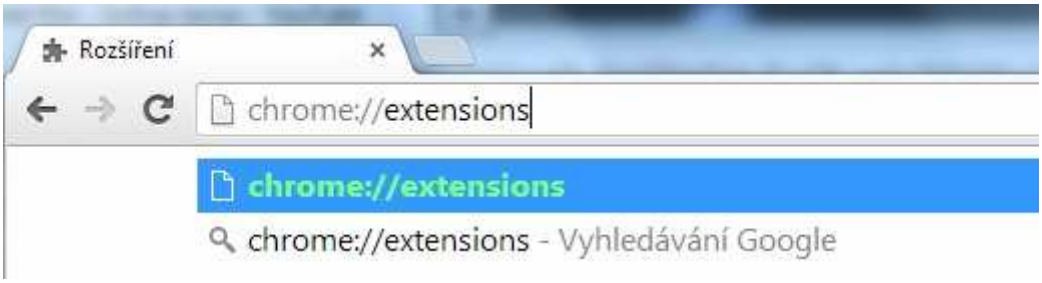


obr 3.2 - životní cyklus aplikace [6]

3.2 Seznámení s prací v Chrome API

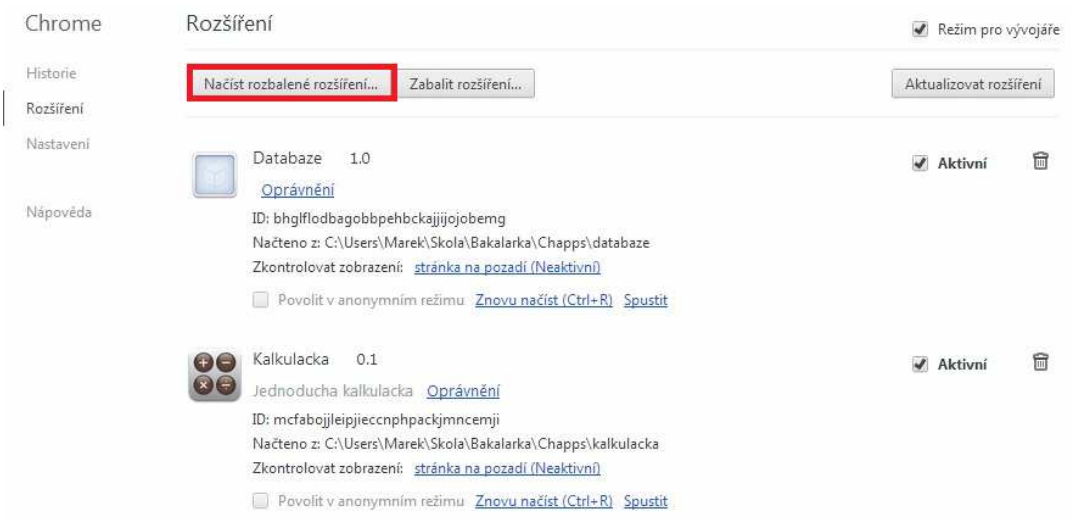
Jediným předpokladem pro práci v Chrome API je instalace prohlížeče Google Chrome. Vzhledem k tomu, že Chrome API pracuje v základu v offline režimu, pro práci není nutné internetového připojení. Zdrojové kódy se ukládají lokálně a výpočetní síla je převedena na samotného uživatele. Aby bylo možné libovolné rozšíření do prohlížeče načíst, do vyhledávacího panelu zadáme slovní spojení: `chrome://extensions`

[6] *Chrome App Lifecycle* [online]. [cit. 2014-05-12]. Dostupné z: https://developer.chrome.com/apps/app_lifecycle



obr 3.3 Začátky s Chrome API

Tento příkaz nás pustí do základní obrazovky pro rozšíření. Zde můžeme načítat a spouštět naše rozšíření pomocí tlačítka s názvem: `Načíst rozbalené rozšíření.`



obr. 3.4 Začátky s Chrome API

3.3 Struktura aplikace

Každá aplikace začíná manifestem. Manifest popisuje základní informace o aplikaci. Mezi tyto informace patří jméno aplikace, verze aplikace, popis aplikace, jaké oprávnění aplikace obsahuje, kam se může podívat, kde získávat data a jiné. Důležitou součástí manifestu je určení jména a lokace background skriptu.

```
{
  "name": "Filesystem",
  "description": "Ukazka prace s filesystememem",
  "version": "1",
  "app": {
    "background": {
      "scripts": ["background.js"]
    }
  },
  "permissions": [
    {"fileSystem": ["write", "retainEntries", "directory"]},
    "storage"
  ],
  "file_handlers": {
    "text": {
      "types": [
          "text/*"
      ],
      "title": "Text editor"
    }
  }
}
```

zdrojový kód 3.1: ukázka manifest.json

Background script určuje, jak se bude aplikace chovat. V rámci background scriptu se určují základní akce aplikace, jako:

- Co se stane při spuštění aplikace.
- Zda se provede něco po ukončení aplikace atd.

Ve zdrojovém kódu 3.2 je velmi důležitý první řádek, který určuje akci po spuštění. Akce po spuštění je definována následovně: aplikace se spustí až po kliknutí na spouštěcí ikonu.

```
// JavaScript Document

chrome.app.runtime.onLaunched.addListener(function() {
  chrome.app.window.create('window.html', {
    'bounds': {
      'width': 500,
      'height': 500
    }
  });
});
```

zdrojový kód 3.2: ukázka backround.js

Poslední složkou aplikace je HTML soubor. V tomto souboru se pomocí HTML s rozšířením o CSS a JavaScriptem nadefinuje celkový vzhled a funkcionalita vyvíjené aplikace.

```
<html>
  <head>
    <style>
      body
      {
      background-color:#b0c4de;
      }
    </style>
  </head>

  <body>
    Moje první aplikace
  </body>
  <script>
    document.write(" v google chrome API ");
  </script>
</html>
```

zdrojový kód 3.3: ukázka window.html

3.4 Bezpečnost

Před začátkem práce s aplikacemi v Chromu je v hodné se seznámit se zabezpečením a bezpečnostními riziky.

Bezpečnou práci v aplikacích zajišťují tři základní prvky.

1) Izolace procesů a úložiště (Process & Storage Isolation)

2) Contet Security Policy (CSP),

3) Model oprávnění (Permissions model).

3.4.1 Process & Storage Isolation

Model Process & Storage Isolation zajišťuje, aby akce jedné aplikace nemohly nijak ovlivňovat data a procesy v jiné aplikaci. To znamená, že každá aplikace spouští svůj vlastní proces a pokud se něco pokazí, tak to přímo neovlivní jiné běžící aplikace. Data uložené v každé aplikaci jsou také izolovány od ostatních aplikací. To znamená, že např. soubor uložený v aplikaci bude viditelný pouze pro tuto aplikaci a uživatele, který ji vytvořil. [7]

3.4.2 Content Security Policy (CSP)

Content Security Policy je bezpečnostní model, který zabraňuje spuštění nežádoucích skriptů (Cross-site scripting). Tato politika přináší do aplikací několik omezení. Prvním z omezení je znemožnění použití vyskakovacích oken, tzv. alertů. Dalším omezením je, že jsou zakázány rutiny obslužných událostí (event handlers), což se projevuje např. tím, že nelze provést kód `<button onclick="…">`. Dále nelze odkazovat na externí zdroje s výjimkou audio a video soborů. Také některé z JavaScriptových metod jako například `eval()` a `new Fiction ()` jsou zakázány. Aplikace v chromu nám umožní odkazovat pouze na skripty a objekty přímo v aplikaci s výjimkou dříve zmíněných audio a video souborů.[8]

3.4.3 Model oprávnění.

Model povolení/oprávnění slouží k definování, k jakým datům bude mít aplikace přístup. Díky využití tohoto modelu je zabráněno neoprávněným přístupům aplikace např. k filesytému. Pokud aplikace nemá přímo v manifestu definovano, že může přistoupit k filesistému nebo webové

[7] *Chrome Apps Architecture* [online]. [cit. 2014-05-09]. Dostupné z: https://developer.chrome.com/apps/app_architecture

[8] *Content Security Policy* [online]. [cit. 2014-05-12]. Dostupné z: https://developer.chrome.com/apps/contentSecurityPolicy

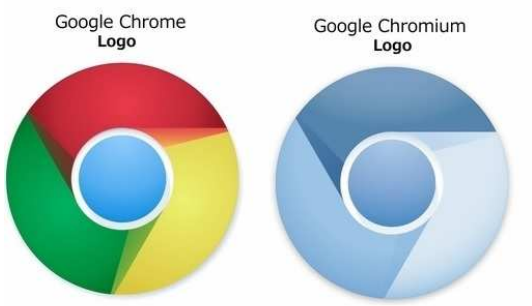
kameře, tak se k datům nedostane. V základě nemá aplikace práva žádné. Povolení definuje uživatel ručně v manifestu každé aplikace, kde je potřeba použít klíčového slova `permissions`. Následně uživatel definuje vybraná specifická povolení, čímž lze přidělit přístup např. k souborům filesystému, historii prohlížeče, k určitým webovým stránkám, informacím o procesoru atd.

```
"permissions": [
    {"fileSystem": ["write", "retainEntries", "directory"]},
    "storage"
  ]
```

zdrojový kód 3.4: přístup k zapisování do složek filesystému

3.5 Chrome vs. Chromium

Google Chrome lze v současnosti považovat za jeden z nejrychleji se rozvíjejících internetových prohlížečů. Jeho hlavní nevýhodou je, že jeho zdrojové kódy ve většině nejsou volně dostupné (open source). Společnost Google zveřejnila pouze část zdrojových kódů pro prohlížeč Chrome. Hlavní části, které dělají Chrome jedinečným, si Google nechal pro sebe. Mezi nezveřejněnou část zdrojových kódů patří např. ty, které definují, jak Chrome pracuje s flash přehrávačem.



obr. 3.5 loga prohlížečů

V roce 2009 Google uvedl na trh prohlížeč Chromium. Chromium je založen na prohlížeči Chrome a z velké části jsou si oba prohlížeče podobné (nejen v logu). Hlavní rozdíl mezi oběma spočívá v tom, že zdrojové kódy Chromia jsou veřejné. Společnost Google si přesto stále udržuje velký vliv na vývoj Chromia. Díky tomu, že je chromium open-source, se na vývoji začala podílet i velká programátorská komunita. Chromiu sice chybí některé funkce, které Chrome zvládá, to ale neznamená, že je Chomium horší. Oba prohlížeče mají např. přístup k marketu s webovými aplikacemi a u obou prohlížečů tyto aplikace fungují. Uživatele se taktéž mohou připojit ke Google účtu jak z Chromu, tak z prohlížeče Chromium. Pro uživatele jsou zde tedy přístupné veškeré služby a data na těchto účtech (např. dokumenty). Výhodou Chromia je, že je bezpečnější. Chromium lze považovat za bezpečnější ve srovnání s Chromem hlavně proto, že prohlížeč Chrome je mnohem složitější a tudíž i více náchylnější k potenciální chybě. Menší bezpečnost Chromu je dána také tím, že některé kódy nejsou k dispozici veřejnosti a komunita tak neví, zda právě nezveřejněné pasáže

nejsou bezpečnostním rizikem. V následující tabulce jsou zdůrazněny hlavní rozdíly mezi oběma prohlížeči.

	Google Chrome	Chromium
Logo	barevné	modré
Hlášení pádů	Ano, pokud je zapnuté	není
Metriky uživateke	Ano, pokud jsou zapnuté	nejsou
Sandbox	Vždy zapnut	Zaleží na distribuci, jestli ano nebo ne
Adobe Flash	Zásuvný modul Sandbox PPAPI (neveřejný) je vždy zahrnut v aktualizaci. Pro větší bezpečnost	Podporuje NPAPI (bez sandboxingu) zásuvný modul, zahrnující
Kód	Otestován vývojáři chrome	Může být změněn koncovými uživateli
Zajištění kvality	Nové aktualizace jsou vždy otestovány před vypuštěním pro veřejnost	Ne vždy testováno. Záleží na distribuci.
Automatické aktualizace	Ano	Ne

Tabulka 3.1 rozdíly chrome a chromia[9]

3.6 Omezení aplikací v Chrome

Každý programovací jazyk, stejně jako každá technologie, má své hranice a nelze s ní vytvořit libovolný program. V rámci aktuální kapitoly se zaměříme na tato omezení. Prvním z řady příkladů je alert. Vyskakovací okno neboli alert v package aplikacích nefunguje. Místo alertu lze použít tzv. lightboxy[10] . Další z webových vlastností, která je nepřístupná v aplikacích Chrome, je otevírání a zavírání dokumentů `document.close` a `document.open` a rovněž nefunguje příkaz `document.write`. Ten však lze nahradit pomocí příkazu `document.createElement`. Za zmínku také stojí příkaz `XMLHttpRequest` určený pro získávání zdrojů z externích webových

[9] *ChromiumBrowserVsGoogleChrome* [online]. [cit. 2014-05-12]. Dostupné z: https://code.google.com/p/chromium/wiki/ChromiumBrowserVsGoogleChrome

[10] Lightbox je javascriptová technika pro zobrazování obrázků či jiného webového obsahu

stránek. Tento příkaz funguje pouze částečně. V package aplikacích je povolen pouze v asynchronní podobě. Pokud jej použijeme synchronně, nefunguje. Všechna tato omezení jsou v Chrome zabudována převážně z bezpečnostních důvodů.[11]

4 Zdrojové aplikace

Cílem mé bakalářské práce je nastínit možnosti, jak pracovat s aplikacemi v Google Chrome prostředí. Pro tento účel byly vytvořeny následující aplikace: Audio přehrávač, kalkulačka, dále aplikace pro demonstrace práce s okny, aplikace, která pracuje s filesystémem a aplikace pro práci s databázemi.

4.1 Audio přehrávač

První program, u kterého jsem se rozhodl předvést funkčnost package aplikací v Chromu, je jednoduchý přehrávač hudby. Program demonstruje přehrávání hudby ve formátu mp3. Audio soubor, který si uživatel přeje poslechnout, musí být uložen na lokálním disku ve stejné složce, kde se nachází zdrojové soubory aplikace.

Aplikace „Audio přehrávač“ se skládá z manifestu, background skriptu, souboru ve formátu HTML a z JavaScriptového souboru. Všechny tyto části si postupně popíšeme.

V prvním kroku si ukážeme, jak vypadá manifest. Manifest slouží k definování základních informací o aplikaci. Více informací o manifestu již bylo uvedeno v kapitole 3.2 s názvem „Struktura aplikace“.

[11] *Disabled Web Features* [online]. [cit. 2014-05-12]. Dostupné z: https://developer.chrome.com/apps/app_deprecated

```
{
  "name": "music_player",
  "description": "program demonstruje jenoduche prehravani hudby
ve formatu mp3",
  "version": "0.1",
  "app": {
    "background": {
      "scripts": ["background.js"]
    }
  },
  "permissions": [
    {"fileSystem": ["directory"]},
    "storage"
  ]
}
```

zdrojový kód 4.1: manifest.json

V manifestu je určeno jméno aplikace `music_player`. Za klíčovým slovem `"description"` se nachází krátký popis aplikace. Dalšími informacemi v manifestu jsou: verze aplikace `0.1`, lokalizace background skriptu a také oprávnění aplikace. V ukázce zdrojového kódu 4.1 je za klíčovým slovem `permissions` definováno, že aplikace může číst soubory a složky, které jsou v aplikaci otevřené.

V druhém kroku si popíšeme background skript.

```
// JavaScript Document

  chrome.app.runtime.onLaunched.addListener(function() {
  chrome.app.window.create('window.html', {
    'bounds': {
      'width': 500,
      'height': 500
    }
  });
});
```

zdrojový kód 4.2: background.js

Background skript je velice jednoduchý. Na začátku kódem `chrome.app.runtime.onLaunched.addListener(function()` definujeme, že aplikace čeká na spuštění do okamžiku stisku tlačítka určeného ke spuštění. Po spuštění aplikace se vytvoří nové HTML okno s názvem `'window.html'` o velikosti 500x500 pixelů, což zajišťuje úsek kódu 4.3.

```
'bounds': {
      'width': 500,
      'height': 500
 }
```

zdrojový kód 4.3:definování rozměrů nového okna

Background skript spustí soubor `window.html`. Soubor `Window.html` je základem aplikace, definuje její vzhled (s případnou pomocí CSS) a volá javaScript, který určuje funkčnost aplikace.

```
<html>
  <head>
    <style>
      input
      {
        width: 100%;
      }
    </style>
  </head>
  <body>
    <input type="text" id="cesta" readonly>  <br>
    <button id="id_soubor">Nacti mp3</button>  <br>
    <audio id="player" controls>
      <source src = "" type="audio/mpeg">
      Your browser does not support the audio element.
    </audio>
    <pom_text></pom_text>
  </body>
  <script src="m_player.js"></script>
</html>
```

zdrojový kód 4.4: window.html

V třetím kroku si popíšeme soubor window.html. Zdrojový kód 4.4 je velmi jednoduchý a skládá se ze tří prvků. Prvním z nich je input, ve kterém se zobrazuje název přehrávaného audio soboru. Druhým prvkem je tlačítko. Při zmáčknutí tlačítka vyskočí dialogové okno, ve kterém lze vyhledat a zvolit mp3 soubor. Dalším prvkem zdrojového kódu 4.4 je HTML5 audio. Audio je vestavěná funkce HTML5, která přehrává hudbu ze zadaného zdroje. Pro přehrávání mp3 souboru stačí do funkce zadat zdroj (např. `src="hudba.mp3")`. Poslední akcí, kterou zde musíme provést, je volání JavaScriptového souboru. `<skript src="m_player.js"></script>`. Tento JavaScriptový soubor dává přehrávači funkčnost.

```
var tlSoubor = document.querySelector('#id_soubor');
var accepts =
[{
  mimeTypes: ['text/*'],
  extensions: ['mp3']
}];
tlSoubor.addEventListener('click', function(e)
{
  chrome.fileSystem.chooseEntry({type: 'openFile', accepts:
accepts}, function(vstup)
  {
    if (!vstup)
    {
      pom_text.textContent = 'Nezvolil si soubor';
      return;
    }
    chrome.fileSystem.getDisplayPath(vstup, function(cesta)
    {          //ziskavam cestu k souboru a ulozim si ji
      var regular = /(\\)/g;
      var pozice;
      while (pozice != -1)
      {
        pozice = cesta.search(regular);
        cesta = cesta.substring(pozice+1);
      };
      document.querySelector('#cesta').value = cesta;
      document.getElementById('player').src = cesta;
    });
  });
});
```

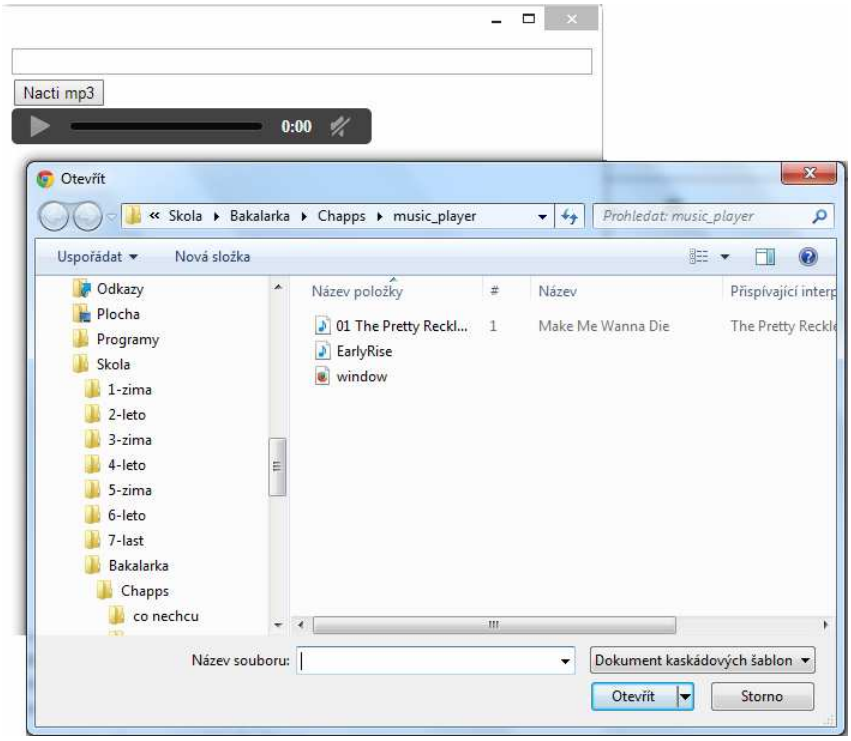
zdrojový kód 4.5: m_player.js

V posledním kroku se zaměříme na JavaScript aplikace. Na začátku zdrojového kódu 4.5 jsou deklarovány proměnné. V proměnné `accepts` je pomocí zdrojového kódu 4.6 určeno, že aplikace přijímá (přehrává) pouze soubory ve formátu mp3.

```
var accepts =
[{
  mimeTypes: ['text/*'],
  extensions: ['mp3']
}];
```

zdrojový kód 4.6: omezení na formát mp3

Následné akce se dějí až po stlačení tlačítka. V první řadě je potřeba zvolit soubor, který chceme přehrávat. Tento soubor zvolíme z dialogového okna, které se objeví po stisku tlačítka.



obr 4.1: dialogové okno pro výběr mp3

Výběr souboru provádí zdrojový kód 4.7.

```
chrome.fileSystem.chooseEntry({type: 'openFile', accepts:
accepts}, function(vstup)
```

zdrojový kód 4.7: výběr audio souboru

Do proměnné `vstup` se přiřadí informace o vybrané položce. Z těchto informací nás zajímá především název audio soboru, který získáme z přístupové cesty

`chrome.fileSystem.getDisplayPath(vstup, function(cesta)`

Následně je potřeba z cesty získat pouze jméno vybraného souboru. To provedeme pomocí jednoduchého regulárního výrazu, který z řetězce odstraní vše, co je napsáno před znakem „\“. Toto

opakujeme, dokud nezůstane pouze jméno soboru a to pak přiřadíme jako zdroj, ze kterého má HTML5 čerpat audio soubor k přehrávání.

```
var regular = /(\\)/g;
var pozice;

while (pozice != -1)
{
  pozice = cesta.search(regular);
  cesta = cesta.substring(pozice+1);
};

document.querySelector('#cesta').value = cesta;
document.getElementById('player').src = cesta;
```

zdrojový kód 4.7: osamostatnění jména audio soboru

Tímto je přehrávač hudby hotov.

4.2 Kalkulačka

Druhým programem pro demonstraci schopností package aplikací v Chrome je kalkulačka. Tato jednoduchá kalkulačka provádí základní početní operace – sčítaní, odčítání, násobení a dělení. Předpokladem pro správnou funkčnost kalkulačky je, že uživatel zadává pouze kladná čísla. Vzhledem ke zmíněným omezením nelze předpokládat, že se jedná o komplexní kalkulačku, která by uživateli umožňovala provádět výpočty všech matematických funkcí. Jedná se pouze o ukázku toho, že v prostředí Chrome lze vytvořit kalkulačku. Pokud by na tom uživatel trval, bylo by možné vytvořit aplikaci, která by umožňovala práci se složitějšími matematickými funkcemi.

V prvním roku si opět popíšeme manifestem aplikace.

```
{
  "name": "Kalkulacka",
  "description": "Jednoducha kalkulacka",
  "version": "0.1",
  "app": {
    "background": {
      "scripts": ["background.js"]
    }
  },
  "icons": { "16": "calculator-16.png", "128": "calculator-
128.png" }
}
```

zdrojový kód 4.8: manifest.json

Manifest je velmi podobný tomu, který byl popsán u aplikace audio přehrávač. Za zmínku stojí především dvě věci. Kalkulačka nepotřebuje žádné speciální oprávnění, tudíž je vynechána část kódu s oprávněními (`permissions`). Dalším rozdílem oproti manifestu audio přehrávače je část definující ikony.

```
"icons": { "16": "calculator-16.png", "128": "calculator-128.png"
}
```

zdrojový kód 4.9: definice ikon

Tato část přiřadí obrázky `calculator-16.png` o velikosti 16x16 pixelů a obrázek `calculator-128.png` o velikosti 128 pixelů jako ikony, které lze vidět před spouštěním aplikace.

Další části manifestu jsou obdobné jako v předchozí ukázce manifestu audio přehrávače. V dalším kroku se zaměřme na background skript.

obr. 4.2: ikona

```
// JavaScript Document
  chrome.app.runtime.onLaunched.addListener(function() {
  chrome.app.window.create('window.html', {
    minWidth: 200,
    minHeight: 300,
    'bounds': {
      'width': 200,
      'height': 300
    }
  });
});
```

zdrojový kód 4.10: background.js

Background script je opět obdobný background scriptu audio přehrávače. Pozastavit se můžeme nad částí `minWidth: 220,` a `minHeight: 290`. Tento úsek kódu určuje, že minimální hranice okna, pod kterou nelze okno zmenšit, je 220x290 pixelů.

Další část programu je HTML okno. Zde definujeme především tlačítka kalkulačky. Tlačítka vkládáme do tabulky o velikosti 4x4, aby byly uspořádány v uživatelsky přívětivém pořadí.

```
<body id = "kalulacka">
<form name="kalulacka" >
 <input type="text" id="vystup" value="0" readonly>
  <input type="text" id="vysledek" value="vysledek" readonly>
   <table>
    <tr>
     <td><p><input type="button" value="7" id="cislo_7"></p>
</td>
      ...
     ...
    </tr>
   </table>
       <td><p><input type="button" value="," id="carka" ></p>
</td>
</form>
  </body>
<script src="kalkulacka.js"></script>
```

zdrojový kód 4.11: ukázka window.html

Závěrečnou částí aplikace kalkulačka je JavaScriptový soubor `kalkulacka.js`. V tomto souboru určujeme, jak se bude program chovat.

```
document.querySelector('#kalulacka').onkeydown = function(e)
{
  switch(e.keyCode) //zpracovani stisku klaves
  {
    case 13:
      vypocet();
    break;
    case 109:
     zpracuj_znamenko('-');
      ...
    break;
    case 97:
      zpracuj_cislo('1');
    break;
  }
}
```

zdrojový kód 4.12: ukázka zpracování stisku kláves

V prvním kroku je potřeba ošetřit stisky kláves klávesnice. Způsob provedení nalezneme ve zdrojovém kódu 4.12. Každá klávesa počítače je reprezentována určitou specifickou číslicí. Tato číslice je zachytávána a podle toho, která číslice je detekována, se provede určitá akce. Pokud je zachycen číselný kód odpovídající číslici na klávesnici, program přejde do funkce, která zpracovává číslice. Pokud je zachycen číselný kód odpovídající operátoru, provede se funkce na zpracování znaménka. Jestliže dojde ke stisku klávesy enter, dojde k přechodu do funkce na výpočet rovnice.

V dalším kroku je nutné ošetřit stisky tlačítek kalkulačky, což je vidět ve zdrojovém kódu 4.13.

```
znamenkoDeleno.addEventListener('click', function(e)
{
  zpracuj_znamenko('/');
});
document.querySelector('#cislo_1').onclick = function(e) {
  zpracuj_cislo('1');
};
```

zdrojový kód 4.13: ukázka zpracování tlačítek kalkulačky

Jednotlivé zachycené znaky se přidávají do řetězce, který může obsahovat libovolný počet číslic a jeden operátor. Tento řetězec je následně vyhodnocen. Nejjednodušším způsobem vyhodnocení by bylo využití vestavěné JavaScriptové funkce `eval()`. Tato funkce dokáže vrátit výsledek řetězce, který obsahuje čísla a operátory. Jak už ale bylo zmíněno v kapitole 3.3 Bezpečnost, tato JavaScriptová funkce je v package aplikacích z bezpečnostních důvodů zakázaná. Proto je zapotřebí vyčíslit řetězec jiným způsobem.

Vyčíslení provedeme následovně. Řetězec si rozdělíme na tři části pomocí regulárního výrazu, který hledá v textu operátory plus(„+“), mínus(„-“), krát(„*“) a děleno(„/“). Vše, co se nachází před operátorem, je považováno za první číslo a vše, co nalezneme za operátorem, je považováno za druhé číslo (viz. ukázka zdrojového kódu č. 4.14).

```
var regular = /(-|\/|\+|\*)/g;
var input = document.querySelector('#vystup').value;
var pozice = input.search(regular);

prvniCislo=input.substring(0,pozice);
znamekno = input.substring(pozice,pozice+1);
druheCislo = input.substring(pozice+1);
```

zdrojový kód 4.14: rozdělení textu

Po rozdělení rovnice na jednotlivé části se zaměříme na její vyhodnocení a získání výsledku. Podle porovnání operátoru zjistíme, zda máme sčítat, odčítat, dělit či násobit. U dělení je potřeba si uvědomit, že nikdy nelze dělit nulou a tuto možnost patřičně ošetřit. Následně je výsledek vepsán na výstup.

```
if (znamekno=="/")
  if (druheCislo=="0")
    document.querySelector('#vysledek').value = "ERR";
  else
    document.querySelector('#vysledek').value =
parseFloat(prvniCislo) / parseFloat(druheCislo);
```

zdrojový kód 4.15: ukázka výpočtu

4.3 Práce s okny

Třetí program znázorňuje, jak se pracuje s okny. V programu „Práce s okny“ je demonstrováno, jak lze v prostředí package aplikací využít oken. Nejedná se však o okna prohlížeče Chrome, jak by se na první pohled mohlo zdát, ale o okna v rámci aplikace. K prácí s okny se využívá

vestavěné API Chromu `chrome.app.window`. Pomocí tohoto API lze nastavovat hranice okna, pozice okna na obrazovce atd. Toto API bylo využito již v předchozích programech. U kalkulačky i u audio přehrávače bylo v background skriptu pomocí `chrome.app.window.create` vytvořeno první a základní okno aplikace. Prvním krokem je jako vždy vytvoření manifestu.

```
{
  "manifest_version": 2,
  "name": "window_work",
  "description": "example of work with windows",
  "version": "4",
  "app": {
    "background": {
      "scripts": ["background.js"]
    }
  }
}
```

zdrojový kód 4.16: manifest.json

Manifest aplikace „Práce s okny“ je obdobný jako u předešlých aplikací. Stejně jako aplikace „kalkulačka“ nepotřebuje ani aplikace „ Práce s okny“ žádná povolení. Proto v manifestu nenalezeme část s `permissions`.

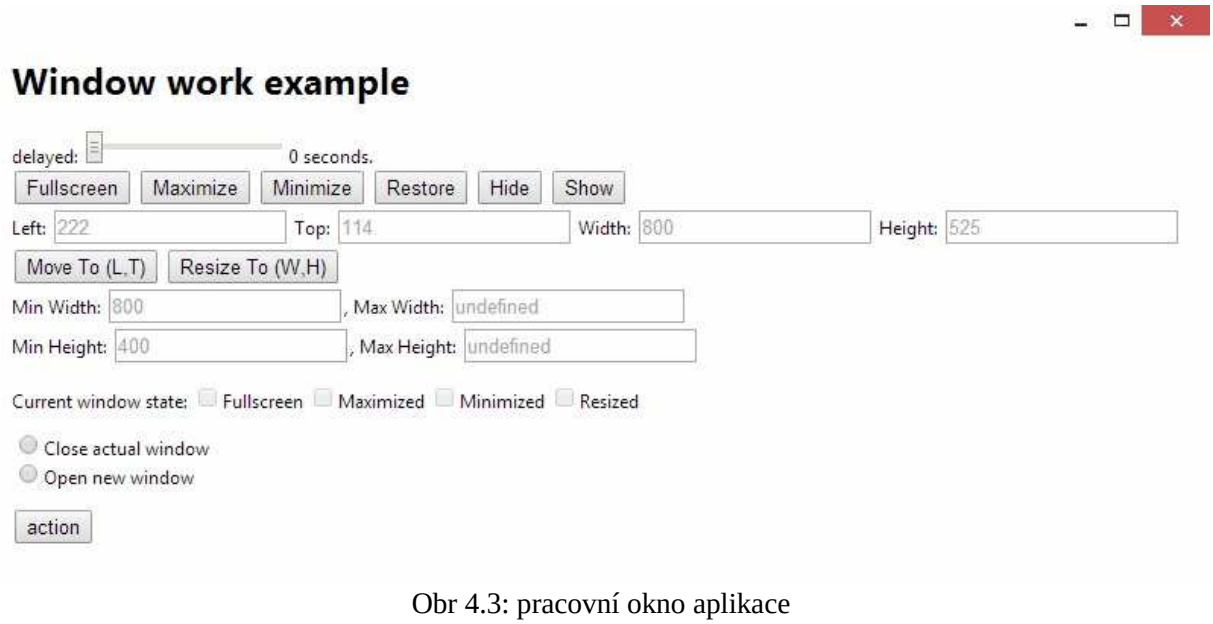
Přejděme na další krok a to na popis background scriptu. Pro vytvoření úvodního okna aplikace použijeme kód `chrome.app.window.create`.

```
// JavaScript Document
chrome.app.runtime.onLaunched.addListener(function() {
   chrome.app.window.create('window.html', {
     id: "mainwin",
      minWidth: 800,
      minHeight: 400,
     'bounds': {
     'width': 800,
     'height': 400
     }
   });
});
```

zdrojový kód 4.17: background.json

První řádek kódu zajišťuje čekání na spuštění aplikace. Jakmile je aplikace spuštěna, vytvoří se okno `window.html` s následujícími vlastnostmi: ID nového okna `mainwin`, minimální šířka `minWidth` 800 pixelů, minimální výška `minHeight` 400 pixelů, velikost okna při otevření 800x400 pixelů.

Dalším krokem je práce ve `window.html`. Zdrojový kód `window.html` si v následující části popíšeme podrobněji. Výsledný vzhled aplikace je vidět na obrázku 4.3.



Obr 4.3: pracovní okno aplikace

HMTL kód obsahuje značky `<html> </html> <body> </body> <head> <body>`, kterými se dál podrobněji nebudeme zabývat.

```
<h1>Window work example</h1>
  <p>delayed:
    <input id="delay-slider" type ="range" min ="0" max="20000"
step="1000" value="0"/>
    <span id="delay-label"></span>  <br>
```

zdrojový kód 4.18: posuvník zpoždění

obr 4.4 posuvník zpoždění

Posuvník zpoždění slouží k zabudování zpoždění ke každé akci. Akci můžeme provést ihned nebo až po uplynutí určitého času. Záleží na tom, jakou hodnotu na posuvníku nastavíme. Zpoždění můžeme nastavit v rozsahu od nuly do dvaceti sekund. Vedle posuvníku se vždy zobrazuje hodnota aktuálního zpoždění. Na obrázku 4.4 je vidět zpoždění o jednu vteřinu.

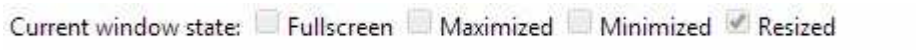
Další funkčností aplikace je možnost manipulovat s okny.

```
<button id="fullscreen">Fullscreen</button>
<button id="maximize">Maximize</button>
<button id="minimize">Minimize</button>
<button id="restore">Restore</button>
<button id="hide">Hide</button>
<button id="show">Show</button>
```

zdrojový kód 4.19: tlačítka akcí

Ve zdrojovém kódu 4.19 jsou nadefinovány tlačítka pro změnu stavů oken. Prostřednictvím těchto tlačítek lze okno maximalizovat, minimalizovat nebo převést do stavu fullscreen, kdy okno zabírá celou plochu obrazovky. Pomocí tlačítka `Restore` lze okno převést do původní velikosti. Pomocí tlačítka `hide` je možné okno schovat a po stisku tlačítka `show` se okno opětovně zobrazí. Problém nastává v situaci, kdy uživatel okno schová, neboť tak zmizí i tlačítko `Show`, na které je nutné kliknout pro opětovné zobrazení okna. Takže je zcela irelevantní, že funkčnost tlačítka je naimplementována, když díky zneviditelnění tlačítka chybí možnost kód spustit.

S těmito tlačítky silně souvisí část, která zobrazuje aktuální stav okna.



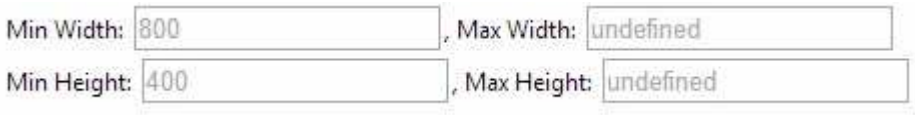
obr 4.5 stav okna

```
    <p>Current window state:
      <input type="checkbox" id="isFullscreen"
disabled>Fullscreen
      <input type="checkbox" id="isMaximized" disabled>Maximized
      <input type="checkbox" id="isMinimized" disabled>Minimized
      <input type="checkbox" id="isResized" disabled>Resized
    </p>
```

zdrojový kód 4.20: aktuální stav okna

Aktuální stav okna je zobrazen v checkboxech viz. obr. 4.5. Pokud se okno nachází ve stavu, který checkbox reprezentuje, je daný checkbox zatržen. Např. na obrázku 4.5 je vidět, že okno v minulosti změnilo svou velikost, proto je checkbox s názvem `Resized` zatržen.

Další částí aplikace je zobrazování velikosti okna. K zobrazování velikosti okna slouží sada tzv. inputboxů. Od spuštění aplikace je nastavená minimální velikost okna, maximální velikost není definovaná.



obrázek 4.6: minimální a maximální velikost okna

Další funkčnost aplikace spočívá v možnosti nastavování pozice levého horního bodu okna a výšky a šířky okna. Veškeré souřadnice se nastavují do iputboxů přímým vepsáním číslice nebo je k tomuto účelu možné využít malé šipky, která vepsanou hodnotu zvýší či sníží o hodnotu jedna. Pokud v inputboxu nic vepsané není, začíná se na čísle nula. Zdrojové kódy jsou k nahlédnutí v ukázce zdrojového kódu 4.21.

```
Left: <input type="number" class="size" id="moveWindowLeft"
step=1 />
Top: <input type="number" class="size" id="moveWindowTop" step=1
/>
Width: <input type="number" class="size" id="resizeWindowWidth"
step=1 />
Height: <input type="number" class="size" id="resizeWindowHeight"
step=1 />
<br>
<button id="move">Move To (L,T)</button>
<button id="resize">Resize To (W,H)</button>
```

zdrojový kód 4.21: zobrazování velikosti okna

Poslední částí aplikace je možnost uzavření stávajícího či otevření nového okna. Výběr mezi těmito dvěma možnostmi se provádí pomocí radiobuttonu. Akce se provede až po stisku tlačítka `action`. Nově otevřené okno má totožné vlastnosti jako okno základní. Jediný rozdíl mezi okny spočívá v nastavení maximální a minimální velikosti okna. U základního okna je nastavena minimální velikost okna, u nově otevřeného okna je naopak nastavena maximální velikost okna.

```
    <form action="">
    <input type="radio" id= "close" name="action"
value="Close">Close actual window<br>
    <input type="radio" id="open" name="action" value="Open">Open
new window
    </form>
   <button id="action">action</button>   <br>
```

zdrojový kód 4.21: zavření/otevření okna

V další části se zaměříme na JavsSript aplikace. Jednou ze zajímavější částí zdrojového kódu je ošetření úvodních tlačítek na změnu stavu okna. Vybraný zdrojový kód 4.22 ukazuje ošetření tlačítka `restore`.

```
document.querySelector('#restore').onclick = function(e) {
  setTimeout(chrome.app.window.current().restore,
document.querySelector('#delay-slider').value);
};
```

zdrojový kód 4.22: funkčnost tlačítka restore

Na začátku zdrojového kódu 4.22 je definováno, že při kliknutí na tlačítko s ID rovnajícímu se slovu `restore`, se provede akce `SetTimeout`. `SetTimeout` je vestavěná JavaScriptová funkce, která má dva parametry. První parametr definuje, jaká akce se provede. Druhý parametr určuje, s jakým zpožděním se akce provede. V našem případě je požadovaná akce definována tak, že se aktuální okno `chrome.app.window.current()` převede do původního nastavení `chrome.app.window.current().restore`.

Dále je možné např. změnit pozici okna (viz. zdrojový kód 4.23). Tato funkce se spustí při stisku tlačítka s názvem `Move To (L,T)`.

```
document.querySelector('#move').onclick = function(e) {
  var x =
parseInt(document.querySelector('#moveWindowLeft').value);
  var y =
parseInt(document.querySelector('#moveWindowTop').value);
  setTimeout(
    function() {
      chrome.app.window.current().moveTo(x, y);
    },
    document.querySelector('#delay-slider').value);
};
```

zdrojový kód 4.23: změna pozice okna

Obsahem zdrojového kódu 4.23 je získání souřadnic pro posunutí okna. Souřadnice pro posunutí okna se nalézají v inputboxech označených jmény `Left` a `Top`. Pomocí příkazu `parseInt(document.querySelector('#moveWindowLeft').value);` získáme hodnotu inputboxu a přiřadíme jí do proměnné. Stejným způsobem postupujeme při získávání druhé souřadnice. Následně pomocí příkazu `chrome.app.window.current().moveTo(x, y);`

posuneme okno na zadané souřadnice. Vzhledem k zabudovanému zpoždění je celé toto posunutí ve funkci pro nastavení zpoždění `setTimeout`. Pokud bychom chtěli provést změnu velikosti okna, použijeme téměř totožný kód. Jediný rozdíl spočívá v tom, že místo `chrome.app.window.current().moveTo(x, y);` použijeme kód pro změnu velikosti `chrome.app.window.current().resizeTo(w, h);`

Všechny akce lze provést se zpožděním. Abychom měli stále přehled o tom, jak velké zpoždění bylo nastaveno, aktualizujeme si jeho hodnotu v hlavním okně.

```
var updateDelaySiderText = function updateDelaySiderText() {
  document.querySelector('#delay-label').innerText =
document.querySelector('#delay-slider').value / 1000 + "
seconds.";
}
document.querySelector('#delay-slider').onchange =
updateDelaySiderText;
updateDelaySiderText();
```

zdrojový kód 4.24: aktualizace hodnoty zpoždění

Detekce změny zpoždění je prováděna prostřednictvím kódu `document.querySelector('#delay-slider').onchange`. Při každé změně zpoždění se provede aktualizace hodnoty zpoždění `updateDelaySiderText()`. Samotné zobrazení se provádí tak, že vezmeme hodnotu posuvníku `document.querySelector('#delay-slider'). innerText`, vydělíme ji tisícem, abychom z milisekund dostali sekundy, a výslednou hodnotu vložíme do políčka `delay-label. document.querySelector('#delay-label').innerText`.

Změna zpoždění ale není to jediné, co musíme upravovat. Dalšími položkami, které je nutné upravit v závislosti na aktuálním stavu okna, jsou checkboxy.

```
chrome.app.window.current().onBoundsChanged.addListener(updateSta
te);
setInterval(updateState, 1000);
```

zdrojový kód 4.25: intervaly aktualizace

Checkboxy aktualizujeme pomocí funkce `updateState`. Tuto funkci voláme vždy při změně hranic okna, což znázorňuje první řádek zdrojového kódu 4.25. Pro jistotu se aktualizace provádí i každou vteřinu příkazem `setInterval`. Funkce aktualizace checkboxů je k nahlédnutí v ukázce zdrojového kódu 4.26.

```
function updateState() {
  document.querySelector('#isFullscreen').checked =
chrome.app.window.current().isFullscreen();
  document.querySelector('#isMaximized' ).checked =
chrome.app.window.current().isMaximized();
  document.querySelector('#isMinimized' ).checked =
chrome.app.window.current().isMinimized();

  document.querySelector('#moveWindowLeft').placeholder =
chrome.app.window.current().getBounds().left;
  document.querySelector('#moveWindowTop').placeholder =
chrome.app.window.current().getBounds().top;
  document.querySelector('#resizeWindowWidth').placeholder =
chrome.app.window.current().getBounds().width;
  document.querySelector('#resizeWindowHeight').placeholder =
chrome.app.window.current().getBounds().height;

  document.querySelector('#windowMinWidth').placeholder =
chrome.app.window.current().getMinWidth();
  document.querySelector('#windowMaxWidth').placeholder =
chrome.app.window.current().getMaxWidth();
  document.querySelector('#windowMinHeight').placeholder =
chrome.app.window.current().getMinHeight();
  document.querySelector('#windowMaxHeight').placeholder =
chrome.app.window.current().getMaxHeight();

  window.onresize = function() {
    document.querySelector('#isResized' ).checked = true;
  }
}
```

zdrojový kód 4.26: aktualizace checkboxů

Nyní zaměříme naší pozornost na vybrané příklady a popišme si je podrobněji. Kód `document.querySelector('#isFullscreen').checked = chrome.app.window.current().isFullscreen();` definuje, že checkbox s názvem `isFullscreen` bude zatrhnut, pokud se aktuální okno nachází ve stavu `isFullscreen()`, což

je stav okna přes celou obrazovku monitoru (není-li nastavená maximální velikost okna). Dalším příkladem je:

```
window.onresize = function() {
    document.querySelector('#isResized' ).checked = true;
  }
```

Tento kód definuje, že checkbox s názvem `isResized` je zatrhnut, pokud se oknu jakkoliv změnila velikost `window.onresize`. Tímto způsobem postupně aktualizujeme všechny checkboxy. Další úsek kódu, o kterém se zmíníme, je otevření nového, případně zavření aktuálního okna.

```
document.querySelector('#action').onclick = function(e) {
  if (document.getElementById('open').checked) {
    chrome.app.window.create('window.html',{
      id: "new_window",
      maxWidth: 1200,
       maxHeight: 1000
    });
  }
  if (document.getElementById('close').checked){
    JavaScript:window.open('', '_self', '');window.close();
  }
};
```

zdrojový kód 4.27: otevření/zavření okna

Otevření nebo zavření okna se provede vždy, když je stlačeno tlačítko s názvem `action`. Existují pouze tři varianty průběhu budoucí akce. První možností je, že je zatrhnut radiobutton pro otevření nového okna `document.getElementById('open').checked`. V tomto případě se otevře nové okno s ID `new_window` s maximální velikostí 1200x1000 pixelů. Druhou možností je zatržení radiobuttonu pro zavření okna. V tomto případě se tedy aktuální okno zavře pomocí kódu `JavaScript:window.open('', '_self', '');window.close();` Poslední možností je, že není zatržen ani jeden radiobutton. V tomto případě se neprovede nic.

4.4 Práce s filesystémem

Další kapitolou je práce s filesystémem. Pozornost zaměříme na to, jak Chrome k filesystému přistupuje a jak s ním dokáže pracovat. V čistém javascriptu je přístup k filesystému velmi obtížný, ne-li úplně nemožný. V prostředí Chrome je pro přístup k lokálnímu disku vytvořeno API `chrome.fileSystem`. Toto API slouží k vytváření, čtení a zapisování na lokální disk. Vše v tomto

prostředí je v sandboxu, což znamená, že je aplikace spuštěna v samostatném procesu a oddělena od všech ostatních aplikaci.

```
{
  "name": "Filesystem",
  "description": "Ukazka prace s filesystemem",
  "version": "1",
  "app": {
    "background": {
      "scripts": ["background.js"]
    }
  },
  "permissions": [
    {"fileSystem": ["write", "retainEntries", "directory"]},
    "storage"
  ],
  "file_handlers": {
    "text": {
      "types": [
          "text/*"
      ],
      "title": "Text editor"
    }
  }
}
```

zdrojový kód 4.28: manifest.json

Aby bylo možné přistupovat k disku, je k tomuto účelu potřeba definovat povolení v manifestu aplikace. Přístup ke čtení a zápisu do složek, které v aplikaci otevřeme, uděluje následující ukázka zdrojového kódu 4.29.

```
  "permissions": [
    {"fileSystem": ["write", "retainEntries", "directory"]},
    "storage"
  ],
```

zdrojový kód 4.29: oprávnění aplikace

Další částí manifestu jsou takzvané „File Handlers“, které určují, s jakými typy souborů může aplikace pracovat.

```
  "file_handlers": {
    "text": {
      "types": [
          "text/*"
      ],
      "title": "Text editor"
    }
  }
```

zdrojový kód 4.30: přístup k textovým soborům

Zdrojový kód 4.30, který je využit i v manifestu, povoluje pracovat s textovými soubory jakéhokoliv formátu. Typ formátu definujeme za klíčovým slovem `types`. Zde je pomocí regulárního výrazu určeno, že na typu textového soboru nezáleží. Další potencionální možností práce se soubory je práce s obrázky. V případě, kdybychom chtěli pracovat s obrázky, museli bychom zavést povolení pomocí zdrojového kódu 4.31. Tento kód by povolil použití obrázků s příponou .jpg nebo .png.

```
"file_handlers": {
  "image": {
    "types": [
      "image/png",
      "image/jpeg"
    ]
  }
```

zdrojový kód 4.31: přístup k obrázkům

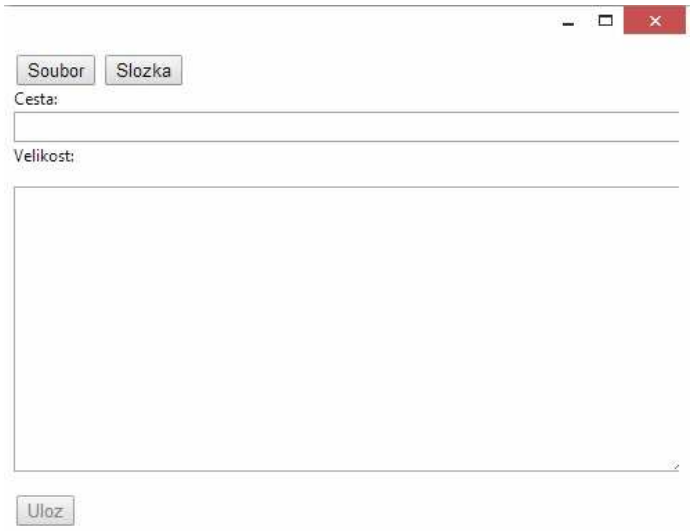
Background skript aplikace pro práci s filesystémem je koncipován obdobně jako předchozí ukázky backgroundscriptů.

```
// JavaScript Document

chrome.app.runtime.onLaunched.addListener(function(launchData) {
  chrome.app.window.create('window.html', {
    id:"fileWin",
    bounds: {
      width: 500,
      height: 500
    }
  },
  function(win) {
    win.contentWindow.launchData = launchData;
  });
});
```

zdrojový kód 4.32: Background.js

Pomocí kódu `chrome.app.runtime.onLaunched.addListener` čekáme na spuštění aplikace. Pak po spuštění vytvoříme nové okno o velikosti 500x500 pixelů a do okna zobrátíme obsah pomocí `win.contentWindow.launchData = launchData`.

Dalším zdrojovým souborem je `window.html` určující design stránky.



obr 4.7: vzhled aplikace pro práci s filesystémem

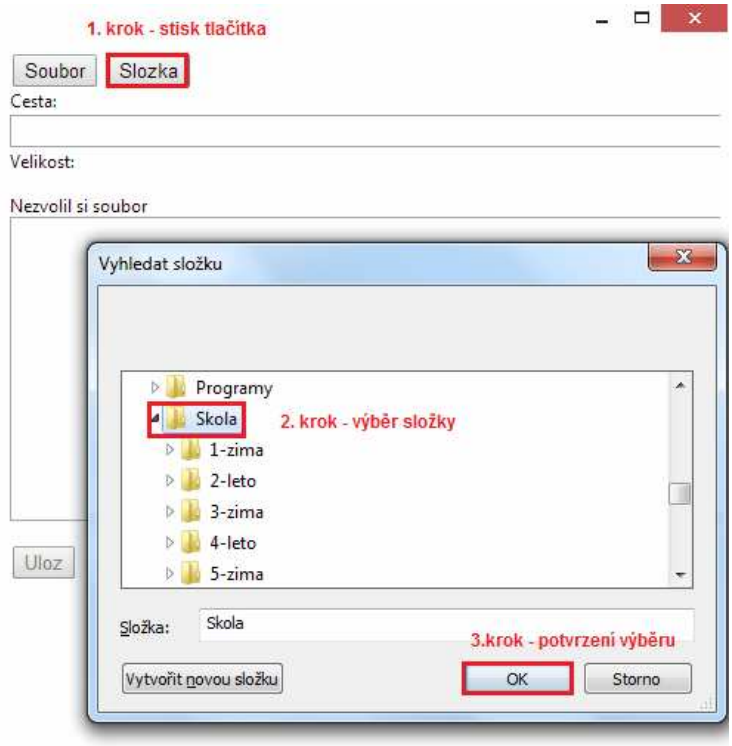
```
<!DOCTYPE html>
<html>
  <head>
    <style>
      textarea
      {
        width: 100%;
        height: 200px;
      }
      input
      {
        width: 100%;
      }
    </style>
  </head>
  <body>
    <button id="id_soubor">Soubor</button>
    <button id="id_slozka">Slozka</button> <br>
    Cesta: <input type="text" id="cesta" readonly>
    <span>Velikost: <span id="velikost"></span> </span> <br><br>

    <pom_text></pom_text>
    <textarea id="textarea"></textarea>
    <p><button id="id_uloz" disabled>Uloz</button></p>
    <script src="b_file_syst.js"></script>
  </body>
</html>
```

zdrojový kód 4.33: window.html

Jak je vidět z obrázku 4.7 aplikace se skládá z:

- dvou tlačítek určených pro výběr složky či souboru, který chceme zobrazit (v případě souboru i editovat)
- inputu, do kterého vkládáme cestu k danému souboru či složce. Tento input je protažen po celé šířce okna díky CSS kódu `input{width: 100%;}`.
- místa, kam můžeme zobrazit velikosti otevřeného souboru.
- prostoru pro případné hlášky programu `<pom_text></pom_text>`.

- prostoru pro zobrazení přečtených dat s názvem `<textarea id="textarea"></textarea>`.
- tlačítka ve spodní části okna pro uložení případných změn v souboru.



obr.4.8: práce v aplikaci

Obrázek 4.8 názorně ukazuje, jak lze s aplikací pracovat. V prvním kroku si představíme prací se složkami. Nejdříve stiskneme tlačítko s názvem `slozka`, následně v dialogovém oknu vybereme složku, kterou chceme zobrazit, a výběr potvrdíme stiskem tlačítka `OK`. Po stisku tlačítka `OK` se do textového pole (`textarea`) vepíšou veškeré soubory a složky, které se nachází ve zvoleném adresáři. Výše popsaný postup provede zdrojový kód 4.34.

```
tlSlozka.addEventListener('click', function(e)
{
  chrome.fileSystem.chooseEntry({type: 'openDirectory'},
function(vstup)
  {
    if (!vstup)
    {
      pom_text.textContent = 'Nezvolil si slozku.';
      return;
    }
    pom_text.textContent = 'zvolil si slozku';
    nactiSlozku(vstup);
  });
});
```

zdrojový kód 4.34: otevření adresáře

Při stisku tlačítka pro načítání složky `tlSlozka.addEventListener('click', function(e)`se otevře adresář `chrome.fileSystem.chooseEntry({type: 'openDirectory'` a spustí funkce `nactiSlozku(vstup);` se získanými vstupními údaji. Velmi podobný je i kód pro otevření souboru.

```
tlSoubor.addEventListener('click', function(e)
{
  var accepts =
  [{
    mimeTypes: ['text/*'],
    extensions: ['js', 'css', 'txt', 'html', 'xml', 'c']
  }];
  chrome.fileSystem.chooseEntry({type: 'openFile', accepts:
accepts}, function(vstup)
  {
    if (!vstup)
    {
      pom_text.textContent = 'Nezvolil si soubor';
      return;
    }
    nactiSoubor(vstup);
  });
});
```

zdrojový kód 4.35: otevření souboru

Rozdíl oproti předchozímu kódu pro otevření adresáře je v tom, že u soborů nyní musíme určit, jaký typ souborů chceme otevírat. Nejprve tedy definujeme, jaké typy souborů povolíme aplikaci zobrazovat.

```
var accepts =
  [{
    mimeTypes: ['text/*'],
    extensions: ['js', 'css', 'txt', 'html', 'xml', 'c']
  }];
```

zdrojový kód 4.36: definice typu soborů

Následně toto omezení aplikujeme a díky tomu se budou zobrazovat pouze soubory s požadovanými příponami.

```
chrome.fileSystem.chooseEntry({type: 'openFile', accepts:
accepts}, function(vstup)
```

zdrojový kód 4.37: aplikace omezení typu

My tedy zobrazujeme soubory s libovolným jménem, ale pouze s příponami .js, .css, .txt, .html, .xml a .c. Ostatní formáty souborů jsou aplikací ignorovány.

V další části se budeme věnovat jednotlivým funkcím v aplikaci. V prvním kroku si budeme chtít zobrazit složky a soubory v námi zvoleném kořenovém adresáři. První funkci, která se provede, bude funkce `nactiSlozku()`.

```
function nactiSlozku(_mujVstup)  //slozku
{
  mujVstup = _mujVstup;
  if (mujVstup.isDirectory)     //pokud je vstup slozka
  {
    var ctiSlozku = mujVstup.createReader(); //vztvorim reader
    var prichoziData = [];
    var prectiVstup = function()
    {
       ctiSlozku.readEntries (function(vysledek)    //prectu
vstup
       {
        if (vysledek.length)    //pokud neco najdu dam si to do
prichozi data
        {                       //a pokuracuju v nacitani
          vysledek.forEach(function(item)
          {
            prichoziData = prichoziData.concat(item.fullPath);
          });
          prectiVstup();
        }
        else
        {              //vlozim data do textarea
          textarea.value = prichoziData.join("\n");
          tlUloz.disabled = true; // u slozky nechci nic ukladat
          vstupniInfo(mujVstup);
        }
      }, chyba);
    };
    prectiVstup(); // cti slozky.
  }
}
```

zdrojový kód 4.37: načtení složky

Na začátku zdrojového kódu 4.37 si potvrdíme, že na vstupu byl opravdu adresář, což zajistíme pomocí kódu `mujVstup.isDirectory`. Pokud byl zvolen, vytvoříme si prostřednictvím kódu `mujVstup.createReader()` reader, který adresář přečte. Do proměnné

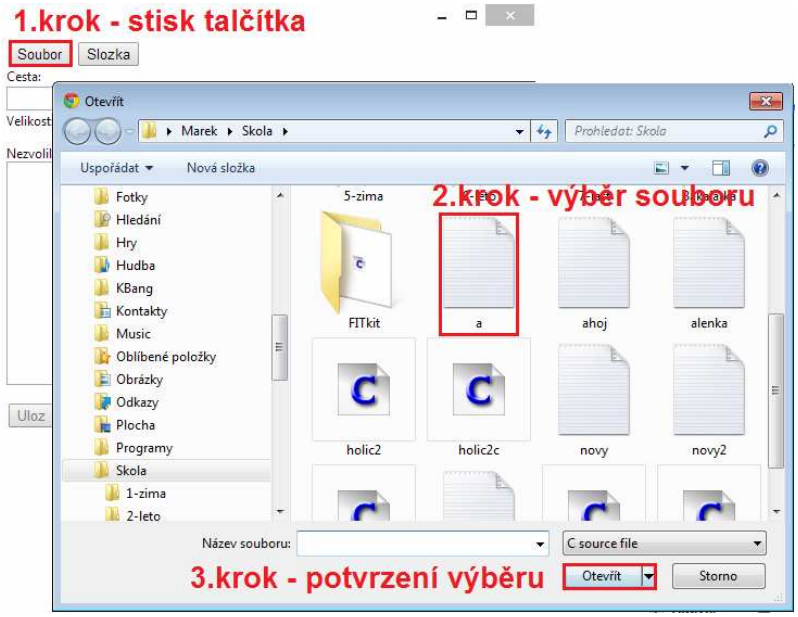
`prichoziData` vkládáme adresáře a složky, které se nacházejí v kořenovém adresáři, a následně je vepisujeme do `textarea`. Na samostatný řádek se vždy vepíše jedna složka/soubor `textarea.value = prichoziData.join("\n")`. Takto rekurzivně postupujeme, dokud je co číst. Pro získání informací o kořenovém adresáři zavoláme funkci `vstupniInfo()`, která je znázorněna v následujícím zdrojovém kódu 4.38.

```
function vstupniInfo(vstup)
{
  if (vstup.isFile)   //zjituju zda li je vstup soubor nebo
slozka
  {                   //pokud je to soubor
    chrome.fileSystem.getDisplayPath(vstup, function(cesta)
    {                   //ziskavam cestu k souboru a ulozim si ji
      document.querySelector('#cesta').value = cesta;
    });
    vstup.getMetadata(function(data)
    {               //ziskavam data a ulozim si jejich velikost
      document.querySelector('#velikost').textContent =
data.size;
    });
  }
  else      //pokud zvolim soubor
  {
    document.querySelector('#cesta').value = vstup.fullPath;
//ulozim si cestu
    document.querySelector('#velikost').textContent = "Neni";
  }                                 //velikost nezjistuju
}
```

zdrojový kód 4.38: vstupní informace

Na začátku zdrojového kódu 4.38 musíme rozhodnout, zda se na vstupu nachází soubor či složka, což zajistíme pomocí kódu `vstup.isFile`. Pokud se jedná o soubor, chceme znát cestu k souboru a jeho velikost. Cestu získáme pomocí funkce `chrome.fileSystem.getDisplayPath()`. Tato funkce potřebuje jako vstupní parametr soubor. Funkce nám vrátí výslednou cestu, kterou uložíme do inputboxu s ID `cesta` (`document.querySelector('#cesta').value = cesta`). Druhou informaci, kterou chceme znát, je velikost souboru. Abychom byli schopni velikost souboru určit, potřebujeme znát o

vstupu více informací než jeho název. Tyto informace získáme pomocí funkce `getMetadata`. Veškeré informace o metadatech si uložíme do proměnné `data`. Následně v této proměnné přistoupíme k informaci o velikosti vstupního soboru pomocí `data.size` a tuto informaci vepíšeme do HTML okna. V případě, že je na vstupu složka, stačí vepsat cestu k složce.



obr. 4.9: výběr souboru

Nyní se zaměříme na práci se soubory. Pokud se na vstupu nachází soubor, budeme jej chtít přečíst. K tomu slouží funkce `nactiSoubor()`. Tato funkce se zavolá při stisku tlačítka s názvem `Soubor`.

```
function nactiSoubor(_mujVstup)  //precteny text vlozim do
textarea
{
  mujVstup = _mujVstup;
  mujVstup.file(function(file)
  {
    prectiText(mujVstup, function(result)
    {
      textarea.value = result;
    });
    tlUloz.disabled = false; // povolim ulozit text
    vstupniInfo(mujVstup);
  });
}
```

zdrojový kód 4.39: načtení souboru

Funkce načtení souboru je velmi jednoduchá. Jejím jádrem je zavolání další funkce, která přečte obsah zvoleného souboru a uloží jej do proměnné. Tuto proměnou vepíšeme do textového pole.

```
prectiText(mujVstup, function(result){
      textarea.value = result;
});
```

zdrojový kód 4.40: volání funkce pro načtení obsahu souboru a výpis dat

Dále funkce zpřístupní tlačítko pro uložení soboru `tlUloz.disabled = false;` a zavolá funkci `vstupniInfo()`, která dodá základní informace o souboru. Pozornost nyní zaměřme na funkci `prectiText()`, která čte data souboru.

```
function prectiText(vstupiData, callback)  //funkce na precteni
dat
{
  vstupiData.file(function(file)
  {
    var nacitani = new FileReader();//inicializuju si FileReader
    nacitani.onerror = chyba;    //pokud se spusti s chybou
zpracuju chybu
    nacitani.onload = function(e) //nactu a vracim pozadovane
vracim data
    {
      callback(e.target.result);
    };
    nacitani.readAsText(file);
  });
}
```

zdrojový kód 4.41: přečtení obsahu souboru

Nejdříve přistoupíme ke vstupnímu souboru prostřednictvím kódu `vstupiData.file(function(file)`. Poté si inicializujeme FileReader. FileReader je JavaScriptový objekt, který umožňuje číst obsah souborů. Pokud se podaří FileReader spustit, FileReader přečte soubor a vrátí jeho obsah.

Poslední části aplikace pro přístup k filesystému je uložení souboru. Tlačítko uložení se zpřístupní až po načtení souboru.

```
tlUloz.addEventListener('click', function(e)
{
  var ulozit = {type: 'saveFile', suggestedName: mujVstup.name};
  chrome.fileSystem.chooseEntry(ulozit, function(zapisovanyText)
  {
    ulozSoubor(zapisovanyText, function(e)
    {
      pom_text.textContent = 'ulozeno';
    });
  });
});
```

zdrojový kód 4.42: tlačítko uložení soboru

Nejdůležitější částí zdrojového textu 4.42 je část `type: 'saveFile'`, která definuje, že tlačítko slouží k ukládání. Dále uživateli navrhneme, že jméno ukládaného soboru bude stejné jako jméno soboru otevřeného `suggestedName: mujVstup.name`. Funkce `ulozSoubor()` pak přečte textarea a obsah uloží.

```
function ulozSoubor(souborNaVstupu, callback)
{
  if (!souborNaVstupu) //je neco na vstupu
  {
    pom_text.textContent = 'neni nic na vstupu';
    return;
  }
  souborNaVstupu.createWriter(function(zapisuj)  //vytovrim
writer
  {
    zapisuj.onerror = chyba;         //pokud se vytvori chyba tak
se zpracuje
    zapisuj.onwriteend = callback;
    zapisuj.onwrite = function(evt)
    {
        console.log("write success");
    };
    var blob = new Blob([textarea.value], {type: 'text/plain'});
    zapisuj.write(blob);          //zapis
  }, chyba);
}
```

zdrojový kód 4.43: ukládání souboru

Nejdříve se ve zdrojovém kódu 4.34 zkontroluje, zda je něco na vstupu. Pokud na vstupu není nic, vypíše se hlášení, že vstup neobsahuje žádná data. Poté si inicializujeme writer, což je JavaScriptový objekt sloužící k zápisu do souborů. Dále vytvoříme tzv. Blob. Blob je objekt reprezentující sekvenci dat podobných souboru, který poskytuje možnost s těmito daty pracovat jako se souborem. Do blobu vložíme to, co se nachází v textovém poli aplikace `var blob = new Blob([textarea.value], {type: 'text/plain'});` a data zapíšeme do souboru `zapisuj.write(blob)`.

4.5 Práce s databází

V rozšířeních Chromu je možnost pracovat i s databázemi. Velkým potenciálem je, že pokud by se dalo databázi spustit jako package aplikaci v Chrome, výrazná část výpočetní síly by byla na straně klienta. Servery by tak nebyly tolik zatížené a mohlo by k nim např. přistupovat více uživatelů. Aby bylo možné pracovat s databázemi ve webovém prostředí, musíme podporovat HTML5. HTML5

totiž umožňuje pracovat s webovými databázemi přímo uvnitř prohlížeče. Díky tomu máme možnost s databázemi pracovat v offline i v online režimu. Bohužel ne všechny prohlížeče podporují práci s webovými databázemi. Chrome je jeden z prohlížečů, který tuto možnost umožňuje. Nejdříve je potřeba vytvořit webovou stránku, ve které následně zprovozníme databázi.

```
<body>
    <div id="doleva">
      Data databaze: <br>
      <ul id="data_databaze">
      </ul>
    </div>
    <div id=doprava>
      <div id="vkladani"> Vloz do databaze:<br>
        <form type="post">
          <input type="text" id="vkladani_textu"
name="vkladani_textu" />
          <input type="button" id="Vloz" value="Vloz do
Databaze"/>
        </form>
      </div>
      <div id="smazani"><br>
      <br>Smaz z databaze: <br>
        <form type="post">
            <select id="select_del">
            </select>
            <input type="button" id="Smaz" name="Smaz"
value="Smaz">
        </form>
      </div>
    </div>
</body>
```

zdrojový kód 4.44: window.html

HTML zdrojový soubor je velmi jednoduchý a skládá se z tří prvků. Prvním z nich je prostor pro data databáze `<ul id="data_databaze"></ul>`. Zde budeme za odrážky vkládat jednotlivé řádky databáze. Druhou části je input, přes který se do databáze vkládají nová data. `<input type="text" id="vkladani_textu" name="vkladani_textu" />`. Poslední část tvoří Select, prostřednictvím kterého budeme vybírat data určená ke smazání z databáze.

Následně musíme spustit javascriptovou část zdrojového kódu.

```
<skript src="database.js"></script>
```

Nejdříve inicializujeme databázi. Určíme její jméno, verzi, přidělíme ji popis a velikost.

```
var velikost = 5 * 1024 * 1024; // 5MB
databaze.webdb.db = openDatabase("databaze_chrome", "1.0",
"Databaze v prohlizeci", velikost);
```

zdrojový kód 4.45: inicializace databáze

Poté vytvoříme tabulku databáze, která bude obsahovat jedinečný primární klíč a k němu přiřazená textová data.

```
var db = databaze.webdb.db;
db.transaction(function(tx)
{
  tx.executeSql("CREATE TABLE IF NOT EXISTS databaze_chrome(ID
INTEGER PRIMARY KEY ASC, data TEXT)", []);
});
```

zdrojový kód 4.46: vytvoření tabulky

Pokud už databáze data obsahuje, budeme je chtít zobrazit, což zajišťuje funkce `nactiDataDatabaze()`.

```
function nactiDataDatabaze(data)
{
  var db = databaze.webdb.db;
  db.transaction(function(tx) {
    tx.executeSql("SELECT * FROM databaze_chrome", [], data,
        err);
  });
}
```

zdrojový kód 4.47: načtení dat databáze

Data načteme pomocí SQL dotazu. Vykonáme tedy příkaz `tx.executeSql("SELECT * FROM databaze_chrome", [], data,databaze.webdb.onError);` Tím říkáme, že chceme načíst všechna data databáze. Data, která jsme vybrali z databáze, chceme i zobrazit.

```
function nactiData(tx, rs)
{
  var vystup = "";
  var select_option = "";
  var databaze = document.getElementById("data_databaze");
  var select_na_mazani = document.getElementById("select_del");

  for (var i=0; i < rs.rows.length; i++)
  {
    vystup += zobrazData(rs.rows.item(i));
```

zdrojový kód 4.48: zobrazení dat databáze

Nejdříve do proměnných `vystup` a `select_option` načteme jednotlivé prvky databáze, což provedeme pomocí for cyklu, který prochází řádky databáze. Odtud zavoláme funkce, které vrátí data v požadovaném formátu pro výstup a následně tyto data zobrazíme do prostoru pro data databáze `data_databaze.innerHTML = vystup`. Data zobrazíme také do selectu, prostřednictvím kterého provádíme výběr k mazání dat `select_na_mazani.innerHTML = select_option`. Data databáze chceme zobrazit ve formě odrážek, což zajistí první volaná funkce, která vrátí data následujícím způsobem: `return "<li>" + data.data + "</li>"` . Funkce `zobrazMazani()` vrací data v podobě optionů pro select takto: `return "<option value='" + data.ID + "'>" + data.data + " </option>";`

Zdrojový kód 4.49 provádí vložení dat do databáze. Funkce se zavolá vždy při stisku tlačítka s názvem `Vloz do databáze`.

```
function vlozDoDatabaze(data_text)
{
  var db = databaze.webdb.db;
  db.transaction(function(tx)
  {
    tx.executeSql("INSERT INTO databaze_chrome(data) VALUES (?)",
[data_text],
        editujDatabazi,
        err);
   });
}
```

zdrojový kód 4.49: vkládání dat do databáze

Funkce pouze vykoná SQL příkaz INSER INTO a to tak, že do databáze se jménem `databaze_chrome` vloží do položky s názvem `data` hodnotu, která je v proměnné `data_text.` Pokud se tato akce provede úspěšně, editujeme databázi `editujDatabazi`. Pokud je akce neúspěšná, zpracujeme chybu `err`.

Pro práci s databázemi je potřebné data do databáze nejen vkládat, ale i je z databáze odebírat. K tomuto slouží funkce, která je upravená pro mazání dat. Funkce `del()` je volána vždy při stisku tlačítka se jménem `Smaz`.

```
function del()
{
  var e = document.getElementById("select_del");
  var co_smazat = e.options[e.selectedIndex].value;
  var db = databaze.webdb.db;

  db.transaction(function(tx){
    tx.executeSql("DELETE FROM databaze_chrome WHERE ID=?",
[co_smazat],
        editujDatabazi,
        err);
    });
}
```

zdrojový kód 4.50: mazání dat do databáze

Prvním krokem v procesu mazání dat je získaní ID prvku, který chceme smazat `var co_smazat = e.options[e.selectedIndex].value;`. Tento prvek poté pomocí SQL příkazu DELETE FROM z databáze odebereme `tx.executeSql("DELETE FROM databaze_chrome WHERE ID=?", [co_smazat], editujDatabazi,err);`. Po úspěšném odebrání prvku databázi editujeme.

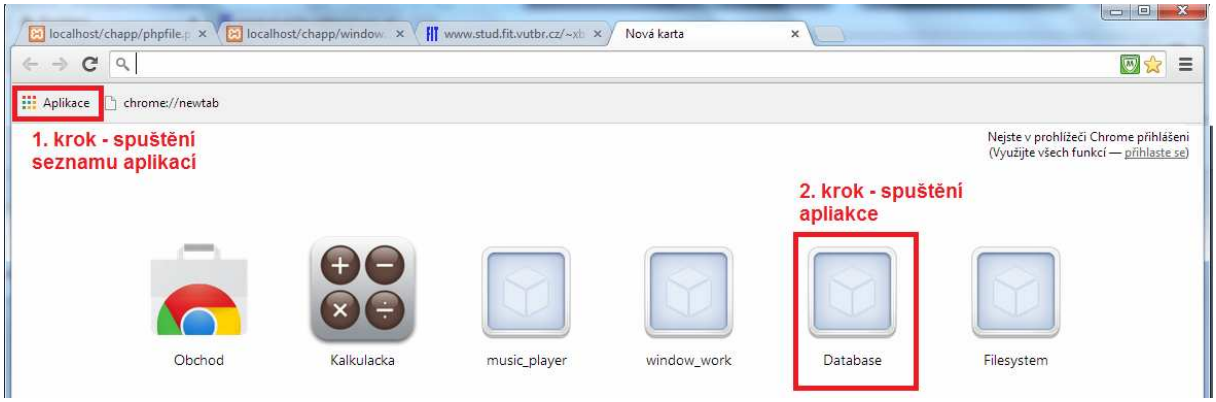
Takto tedy vypadá funkčnost velmi jednoduché webové databáze. Nyní k webové stránce přidáme manifest a vytvoříme z ní tzv. hosted aplikaci. Manifest hosted aplikace je podobný manifestu u package aplikací. Program však nespouštíme lokálně, nýbrž odkazem na webovou stránku, kde je aplikace uložená.

```
{
  "manifest_version": 2,
  "name": "Database",
  "version": "1.0",
  "description": "Program demonstruje jednoduchy pristup do
databaze",
  "app": {
    "urls": [
      "http://www.stud.fit.vutbr.cz/~xbugan00/window.html",
      "https://www.stud.fit.vutbr.cz/~xbugan00/window.html"
    ],
    "launch": {
      "web_url":
"http://www.stud.fit.vutbr.cz/~xbugan00/window.html"
    }
  },
     "permissions": [
      "http://www.stud.fit.vutbr.cz/~xbugan00/window.html",
      "https://www.stud.fit.vutbr.cz/~xbugan00/window.html"
    ]
}
```

zdrojový kód 4.51: manifest.json

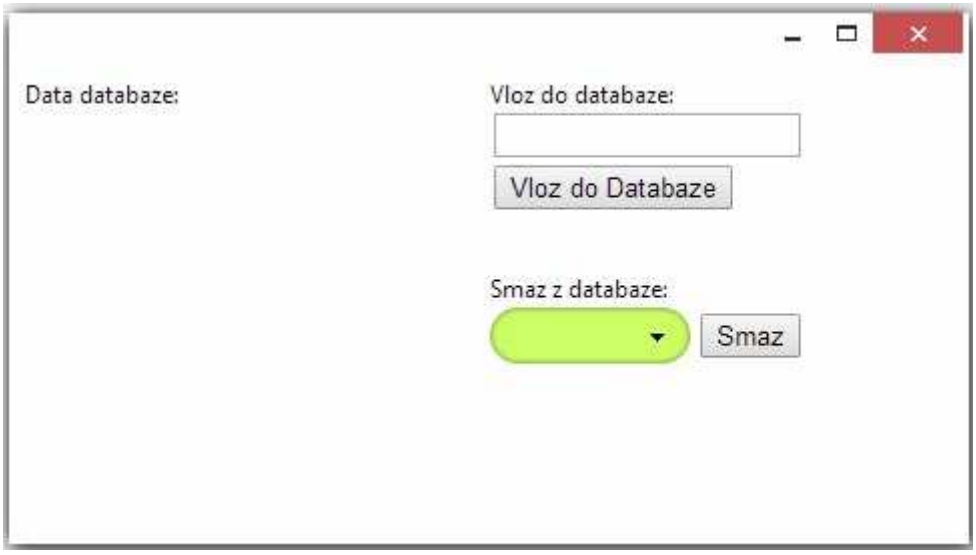
Manifest ukazuje, které odkazy aplikace využívá `"app": { "urls":"*" }`, jaký odkaz má být spuštěn `"launch":{"web_url":"*"}` a k jakým odkazům stránka může přistoupit z bezpečnostního hlediska `"permissions": ["*"]`.

Celou aplikaci pak spustíme způsobem znázorněným na obrázku 4.10.



obr 4.10: Spuštění hosted aplikace pro databáze

Tím, že jsme použili hosted aplikaci a nikoliv package aplikaci, jsme přišli o hlavní výhodu webových databází. V tomto případě je výpočetní síla soustředěna stále na straně serveru. Abychom výpočetní silu dostali na stranu klienta, museli bychom kód spustit v package aplikaci, což bohužel za současné situace není možné. Z bezpečnostních důvodů vývojáři Chrome aplikací tuto možnost zamezili. Kód by sice částečně fungoval, nebylo by však možné se k databázi dostat pomocí SQL příkazů. Výsledek stejného kódu převedeného do package aplikace je znázorněn na obrázku 4.11: databáze v package applikaci.



obr. 4.11: databáze v package applikaci

5 Budoucnost

V package aplikacích se skrývá velký potenciál. Už nyní se hojně využívají nejrůznější webové aplikace např. u her či jiných doplňků. Nevýhoda webových aplikací spočívá především v tom, že potřebují být vždy připojeny k internetu. Chrome tento nedostatek v package aplikacích odstraňuje. Pomocí stejných technologií můžeme vytvářet aplikace se stejnou funkčností a nemusíme být online. Package aplikace jsou vytvořeny tak, aby pracovaly primárně v offline režimu a pokud existuje možnost práce online, tak se package aplikace připojí a využívají internetové připojení pro práci, dokud je to možné. Další nespornou výhodou je jejich multiplatforemnost, díky které se nepotřebujeme se zaobírat tím, zda aplikaci vyvíjíme pro Windows, linux, macOS, Android či jiné prostředí. Pokud operační systém, ve kterém se právě nacházíme, podporuje Chrome framework, bude nám fungovat i aplikace. Všechny aplikace jsou vytvořeny v HTML5 a JavaScriptu, tudíž není potřeba se učit žádný další programovací jazyk. Jednoduše pro práci využijeme to, co známe z tvorby webu, a použijeme tyto znalosti při vytváření package aplikací. Další výhodou package aplikací je, že Google zde rozšířil funkčnost JavaScriptu a vytvořil tak podmínky pro práci s hardwarem počítače. Díky tomu máme např. přístup do filesytému počítače, což s webovými aplikacemi není možné,

protože by to bylo z bezpečnostního hlediska velmi nebezpečné. Package aplikace však k hardwaru přistupovat můžou a proto přes ně lze např. ovládat kameru počítače, upravovat nastavení jasu monitoru, ovládat připojení USB portu atd. Google také zajišťuje velmi jednoduchou distribuci aplikací. Pokud se rozhodneme vytvořit a zveřejnit aplikaci, stačí ji vložit na webstore všech aplikací Googlu. V rámci webstore aplikací námi vloženou aplikaci Google otestuje, zda neobsahuje škodlivý kód. Pokud aplikace projde přes bezpečnostní kontrolu, Google jí zveřejní a už se stará o vše, co je s aplikací spojené. Pokud upravíte kód vaší aplikace, na Google webstoru se objeví aktualizovaná verze. Veškeré aplikace dostanou informaci o tom, že je přístupná nová verze a že se mají aktualizovat.

Díky výhodám aplikací lze očekávat, že jejich funkčnost a obliba bude nadále růst. Do vývoje se zapojí větší komunita a díky tomu budou aplikace různorodější a sofistikovanější.

6 Závěr

Cílem této práce bylo zhodnotit prostředí pro vývoj webových aplikací v desktopovém prostředí. Popsali jsme základy způsobu práce, výhody a nevýhody technologií jako jsou Ajax Push Egine, Cappuccino Web Framework, TideSDK a Chrome aplikace. Z tohoto výčtu jsme následně vybrali technologii pro vývoj Chrome aplikací k demonstraci funkcionality.

Pomocí Chrome aplikací jsme si vytvořili vzorek aplikací, které částečně demonstrují možnosti práce s nativními aplikacemi. Technologie Chrome API umožňuje vytvářet základní aplikace jako např. kalkulačka a audio přehrávač. V práci bylo demonstrováno, že aplikace v desktopovém prostředí můžou bez větších obtíží přistupovat a pracovat s filesystémem. Rovněž byly demonstrovány základy toho, jak lze v aplikacích ovládat jednotlivá okna. Taktéž je zde nastíněn pokus, jak pomocí těchto aplikací tvořit databáze, což bohužel v této chvíli není zcela možné, neboť z bezpečnostního hlediska by to bylo stále riziko. Dle mého názoru v budoucnu tato funkcionalita bude zprovozněna. Je však předvedeno, že prostřednictvím aplikací je možné se připojit k webovým stránkám a tam s databázemi pracovat. Už nyní je tato technologie dobře uzpůsobená pro různorodou práci od práce s hardwarem počítače až k práci s webem. Pouze tyto technologie nejsou v povědomí programátorské komunity. Proto zastávám názor, že vytváření nativních aplikací pomocí webových technologií čeká v blízké budoucnosti rozvoj.

Literatura

[1] Rábová, Z., Hanáček, P., Peringer, P., Přikryl, P., Křena, B.: Užitečné rady pro psaní odborného textu [online]. Brno: c1993-2008, aktualizováno 2008-11-01 [cit. 2008-11-28]. Dostupné na URL: <http://www.fit.vutbr.cz/info/statnice/psani_textu.html>

[2] Kolektiv autorů: *Pravidla českého pravopisu*. Academia, Praha, 1993. ISBN 80-200-0475-0

[3] WEI-MENG, Lee. *Begining Android Application Development*. Indianapolis: Wiley Publishing, Inc., 2011. ISBN 978-1-118-01711-1.

[4] APE project. *APE project* [online]. [cit. 2014-05-09]. Dostupné z: http://ape-project.org/

[5] Cappuccino. *Cappuccino-project* [online]. [cit. 2014-05-09]. Dostupné z: http://www.cappuccino-project.org/

[6] *TideSDK* [online]. 2012 [cit. 2014-05-09]. Dostupné z: http://www.tidesdk.org/

[7] *Chrome Apps Architecture* [online]. [cit. 2014-05-09]. Dostupné z: https://developer.chrome.com/apps/app_architecture

[8] *Content Security Policy* [online]. [cit. 2014-05-12]. Dostupné z: https://developer.chrome.com/apps/contentSecurityPolicy

[9] *ChromiumBrowserVsGoogleChrome* [online]. [cit. 2014-05-12]. Dostupné z: https://code.google.com/p/chromium/wiki/ChromiumBrowserVsGoogleChrome

[10] *Disabled Web Features* [online]. [cit. 2014-05-12]. Dostupné z: https://developer.chrome.com/apps/app_deprecated

[11] KINLAN, Paul. *A Simple TODO list using HTML5 WebDatabases* [online]. 2010 [cit. 2014-05-12]. Dostupné z: http://www.html5rocks.com/en/tutorials/webdatabase/todo/

[12] BIDELMAN, Eric. Filesystem-access. In: *Chrome-app-samples* [online]. 2012 [cit. 2014-05-12]. Dostupné z: https://developer.chrome.com/apps/samples

[13] MANGINI, Renato. Window-state. In: *Chrome-app-samples* [online]. [cit. 2014-05-12]. Dostupné z: https://developer.chrome.com/apps/samples

Seznam příloh