Univerzita Hradec Králové
Fakulta informatiky a managementu

BAKALÁŘSKÁ PRÁCE

Electron.js

O. S. Mikeska

Autor: *Ondřej Silvestr Mikeska*

Studijní obor: *Aplikovaná Informatika*

Vedoucí práce: *Mgr. Daniela Ponce Ph.D.*

Hradec Králové červen 2020

Prohlášení

Prohlašuji, že jsem bakalářskou/diplomovou práci zpracoval/zpracovala samostatně a, s použitím uvedených zdrojů.

V Hradci Králové dne 2020 Ondřej Silvestr Mikeska

Poděkování

Poděkování patří vedoucí bakalářské práce doktorce Daniele Ponce za výtečné vedení práce.

Anotace

V současné době JavaScript ve všech jeho podobách patří mezi nejvyužívanější programovací jazyky, převážně díky rostoucí popularitě webových aplikací. Nicméně stále je zde poptávka po desktopových aplikacích.

Raději než vyvíjet desktopovou aplikaci pro každou platformu zvlášť je nyní možné vyvíjet multiplatformní aplikace prostřednictvím frameworků využívajících tradičně webové technologie ve desktopovém prostředí. Toto se zdá jako dobrá alternativa, kdy je možné implementovat tyto webové technologie, které jsou již hojně využívány při vývoji webových aplikací a zkombinovat je s funkcionalitami, které jinak umožňují spíše desktopové aplikace.

Electron, dříve známý jako atom-shell, je jeden z těchto frameworků, který spojuje prohlížeč Chromium a Node.js.

Táto práce se zaměřuje na vyhodnocení tohoto frameworku za použití několika kritérií a srovnává jej s podobnými frameworky a tradičními webovými aplikacemi.

Annotation

Title: Electron.js

In recent years, JavaScript, in all its forms, is one of the most used programming languages, which can be attributed primarily to the rising popularity of web applications. However, there is still a demand for desktop apps.

Rather than developing a desktop app for each platform, it is now possible to develop cross-platform apps using frameworks that allow the use of JS, HTML and CSS in a desktop environment. This seems like a good alternative, making it is possible to implement web technologies that are already used in development of web apps and combine them with functionality that otherwise only desktop applications could offer.

Electron, originally called atom-shell, is one of these frameworks, combining Chromium rendering engine and Node.js. This thesis will focus on evaluating said framework using several criteria and compare it to other similar frameworks and traditional web applications.

Obsah

Seznam obrázků

1 Úvod

1.1 Důvod výběru tématu práce

V současné době JavaScript ve všech jeho podobách patří mezi nejvyužívanější programovací jazyky spolu se aplikacemi realizovanými jako webové aplikace. Nicméně stále je zde poptávka po desktopových aplikacích.

Electron.js či ostatní frameworky, které umožňují využívat JavaScriptu HTML a CSS pro tvorbu multiplatformních desktopových aplikací, se zdají být dobrou alternativou, kde je možné využít těchto primárně webových technologií a spojit je s funkcemi, které by jinak nabízely pouze desktopové aplikace.

Kombinace těchto tří technologií nabízí mnoho výhod, zejména z pohledu tvorby uživatelských rozhraní. HTML, CSS, JS, či nějaké jejich pre-procesory, umožňují jednu z nejefektivnějších, nejjednodušších a nejméně omezujících metod tvorby designu GUI, které je důležitou součástí každé aplikace.

2 Cíl práce

Cílem práce je vyhodnotit framework Electron a demonstrovat jeho použití na vývoji ukázkové aplikace.

Poznatky zjištěné z práce poskytnou přehled o frameworku Electron.js a frameworcích jemu podobných, které mohou být nápomocny při rozhodování, jestli daný framework je stále vhodnou volbou při vývoji desktopových aplikací. Cílem je také zhodnocení možností použití frontend frameworku. Dalším obecným cílem je vytvořit přehled o některých jeho kvalitách v porovnání s alternativními frameworky.

Vyhodnocena bude JavaScriptová verze Electron.js, ačkoli existují i jiné alternativy, jako například Electron.NET [1] umožňující vývoj pomocí ASP.NET.

3 Metodika vyhodnocení

Vyhodnocení frameworku proběhne primárně na základě informací dodaných vývojáři – dokumentace [2], poznatky od uživatelů, porovnáním s alternativními frameworky, dále s prostřednictvím poznatků zjištěných při vývoji ukázkové testovací aplikace.

Nejdříve bude zhodnocen přímo framework sám o sobě, například pro jaké aplikace je nejčastěji využíván, dále bude prozkoumána nabídka možností, jakými je framework dostupný k získání, CLI, balíčky, dále jeho fungování společně s frontend frameworky a příklady funkcionalit, které nabízí.

Při vyhodnocování bude framework primárně porovnáván v kontrastu s architekturou s webových aplikací běžícími na serveru a v prohlížeči, jelikož jehož de facto principem je umožnění použití technologií užívaných při vývoji těchto webových aplikací, ale v desktopovém prostředí.

Další částí bude zjištěni alternativ z podobně fungujících frameworků a jejich srovnání ve vybraných kritériích.

Srovnání s tvorbou desktopových aplikací pomocí technologií založených na jiném principu či jazyce než JavaScript/Node.js není součástí této práce.

Nakonec bude vytvořena ukázková aplikace pro představení Electron.js frameworku a některých jeho možností. Důraz bude kladen na vyzkoušení jeho funkcionality spolu s frontend frameworkem, dále užití lokálního úložiště a následné zhodnocení znovupoužitelnosti kódu při případném znovupoužití ve webové aplikaci či naopak.

4 Electron.js

4.1 Co je to Electron

Electron je open-source knihovna umožňující vývoj multiplatformních aplikací s využitím HTML, CSS a JavaScriptu. Tato knihovna byla původně zvaná atom-shell a byla vytvořena primárně pro textový editor Atom vyvíjený společností GitHub. Nicméně se od jejího vydání v roce 2013 rozšířila a stala velmi populárním řešením pro multiplatformní aplikace.

Ke svému fungování ji využívají například: Textové editory Visual Studio Code či již zmíněný Atom, dále populární komunikační aplikace pro hráče Discord, desktopoví klienti pro WhatsApp, Skype či Slack, dále editor pro design a návrh uživatelských rozhraní a vektorovou grafiku Figma a další. [3]

4.2 Technologie

Ke svému fungování Electron využívá primárně dvou technologií, a to:

Chromium [4] - webový prohlížeč vyvíjený Googlem, jedná se o open-source verzi Google Chrome [5]. Electron z něj nicméně využívá pouze část zvanou Chromium Content Module, která mimo jiné zahrnuje vykreslovací jádro prohlížeče Blink [6], a jejíž úkolem je vykreslovat webové stránky (zpracování HTML, CSS, JS). Tato technologie umožňuje Electronu zobrazovat uživatelské rozhraní jako webový prohlížeč.

Druhou technologií, která naopak zajišťuje logiku na pozadí, je Node.js [7] běhové prostředí pro JavaScript založené na stroji V8 [8], umožňující vývoj logiky na straně serveru pomocí JavaScriptu. Node.js také přidává API, které například umožňují přístup k lokálnímu souborovému systému. [9, 10]

4.3 Základní princip

Základní princip fungování Electronu je rozdělení na dva typy procesů main a render.

Proces main zajišťuje logiku aplikace na pozadí, systémové funkce, spuštění/ukončení aplikace, vytváření oken a podob. Mimo jiné také zajišťuje spouštění procesů render.

Proces render zajišťuje frontend část aplikace, zobrazování uživatelských rozhraní a komunikaci s uživatelem.

Zatímco proces main je pouze jeden, procesů render může být více (většinou každé okno má svůj vlastní).

Každý z procesů má omezení, k jakým modulům může přistupovat, například přístup k systémovým API má pouze proces main, nikoli proces render. Data mezi procesy je pak možno předávat pomocí meziprocesních zpráv (IPC). [9, 10]

5 Vyhodnocení Frameworku

5.1 Využití frameworku

Electron umožňuje vývoj aplikací pro Windows, macOS a Linux.

Electron na svém webu uvádí oficiální portfolio aplikací, jež tento framework využívají ke své funkci, resp. alespoň pro funkci jejich desktopového klienta. K březnu 2020 to je 878 (865 k 3.1.) registrovaných aplikací. Aplikaci lze přidat do tohoto seznamu pomocí GitHub repositáře, podaná aplikace však musí být schválena.

Nejobsáhlejší kategorie jsou Productivity, Developer tools a Utilities, následovány kategorií Music. Nejčastěji se tedy jedná o pomocné/doplňkové aplikace, ačkoli to není pravidlem.

Mezi nejpopulárnější z těchto aplikací lze považovat například již zmíněné Visual Studio Code, Slack, WhatsApp, Discord, Skype, Figma, dále pak desktopové klienty jako GitHub desktop, MongoDB Compass a další.

Z podstaty mnoha těchto aplikací lze usuzovat, že jejich typ lze rozdělit na plnohodnotné aplikace zcela fungující za pomocí Electronu a aplikace, kde je Electron pouze desktopovým lehkým klientem a aplikační logika je realizována mimo Electron aplikaci (nejspíše na serveru). Do první kategorie lze zařadit například Visual Studio Code či editor Atom a pravděpodobně většinu z jednoduchých/pomocných aplikací, do druhé lze zahrnout například komunikační aplikace viz WhatsApp, Skype, Discord apod., které mají mnoho dalších klientů. Dále také například aplikaci Figma, jež je primárně webovou aplikací.

V mnoha případech se lze také domnívat, že aplikace využívají vzdálené databáze, API a podobně, přestože Electron umožňuje ukládat data i lokálně.

Jedná se jak o open source, tak closed source projekty [11].

5.2 Možnosti vývoje – Dostupnost

V této sekci bude vyhodnocena dostupnost frameworku.

Jelikož Electron využívá node.js a je založen primárně na JavaScriptu, využívá správce balíčků Node Package Manager (npm) [12]. Tato veřejná databáze umožňuje sdílení balíčků a obsahuje velké množství, balíčků s kódem, primárně javascriptových. Nejčastěji je s ní interagováno pomocí CLI, které nabízí řadu příkazů pro její efektivní využívání.

Důležitým prvkem, který je také třeba brát v potaz, je zdali bude při vývoji použit správce modulů, například Webpack [13] či jiný. Tento správce by umožnil efektivnější vývoj díky mapování závislostí modulů.

5.2.1 Samotný balíček

První, a nejpřímější možností, jak získat Electron je instalace přímo z oficiálního balíčku z npm pomocí příkazu:

```
npm install electron
```

Tento příkaz nainstaluje potřebné balíčky pro fungování Electronu do složky *node_modules* a vytvoří soubor *package-lock.json*, který slouží ke správě těchto balíčků.

Vše zbylé je nutno vytvořit s vlastním úsilím včetně nezbythných *package.json, main.js, index.html* a podobně.

Tato možnost je “nejčistší”, nicméně je nutné naprogramovat mnoho věcí ručně, a to i včetně věcí, které se často opakují v každé aplikaci Electronu. Proto byly vytvořeny různé šablony (Boilerplate) a CLI, které už mají v sobě vytvořené tyto opakující se prvky a není nutné vše programovat pokaždé znovu.

5.2.2 Boilerplate

Druhou možností je tedy užití šablony (Boilerplate), z nichž několik vybraných bude následně zhodnoceno.

Ukazuje se, že šablony Electronu jsou distribuovány nejčastěji pomocí git repositářů.

electron-quick-start

Tento repositář, doporučený i v oficiální dokumentaci, obsahuje pouze nezbytný základ pro běh aplikace; mimo jiné již zmíněné *package.json*, *main.js, index.html* a dále také *renderer.js* pro základ procesu render. Jedná se o velmi minimální strukturu.

electron-boilerplate

Podobně jako předchozí se i tato šablona snaží doručit potřebné základy pro aplikaci, nicméně oproti předchozímu je mnohem rozsáhlejší, obsahuje například základní rozdělení do adresářů, základ pro unit a e2e testy, optimalizace startu aplikace a další. Kormě toho obsahuje již předinstalované některé nástroje Electronu pro funkci tohoto obsahu, například Webpack.

electron-react-boilerplate / electron-vue / angular-electron

Příklady repositářů s předpřipravenou funkcionalitou s frontend frameworky, jako například uvedené React.js, Vue.js, Angular.js. jejichž problematice se bude věnovat následující sekce. Tyto repositáře obsahují jak základ nutný pro běh Electron aplikace a další nástroje jako předešlý, tak součásti nutné pro běh daného frontend frameworku a jejich propojení.

electron-api-demos

Zajímavý repositář obsahující jednoduchou aplikaci, sloužící jako návod, představení či dokumentace funkcí, které Electron nabízí. Spíše tedy vhodný jen jako informativní.

Mimo vyjmenovaných existuje mnoho dalších šablon, je možné je najít například na webu Electronu v sekci *community* nebo v repozitáři *awesome-electron* [11], který odkazuje na mnoho užitečných materiálů k Electronu.

Dále například stojí za zmínku nástroj *electron-builder* [14] sloužící k sestavování aplikací Electronu, který také nabízí několik šablon i v kombinaci s React či Vue a je také součástí některých z již jmenovaných.

5.2.3 Shrnutí

Výběr z možností instalace závisí na dané situací a zamýšlené aplikaci. V každém případě je pravděpodobně výhodnější zvolit některou ze šablon, což ušetří velkou část práce s přípravou aplikace. Pokud je potřeba co nejzákladnější sestava, například pro experimentální aplikaci, je vhodný electron-quick-start. Pokud se vytváří složitější aplikace, je vhodný electron-boilerplate či nějaký s již implementovaným frontend frameworkem využívající Webpack nebo jiný module bundler, případně některý z dalších podle požadovaných technologií.

5.3 Možnosti vývoje – Použití s frontend frameworky

Přestože lze vyvíjet aplikaci v Electronu, stejně jako webovou aplikaci, pouze s užitím Vanilla JavaScriptu, užizí nějakého frontend frameworku by mohlo výrazně usnadnit a zefektivnit vývoj uživatelského rozhraní aplikace. V této sekci bude zhodnoceno a vyzkoušeno použití těchto frameworků s Electronem. Konkrétně pro účely zhodnocení bude zvolen zástupce, a to framework ***Vue.js,*** ačkoli díky podobnému principu těchto frameworků lze usuzovat, že podobně se bude moci jednat i s ostatními frameworky (React, Angular, ...).

První hypotézou je, že použití těchto frameworku bude přímočaré – Electron bude sloužit jako schránka zastupující prohlížeč a daný framework bude běžet v ní, stejně jako by tomu bylo v případě webové aplikace.

Uvažováno ale bude, že optimálním výsledkem pro efektivní vývoj je bezproblémová funkce obou frameworků, pokud možno s pomocí module bundleru (pro tuto práci bude jako zástupce uvažován Webpack).

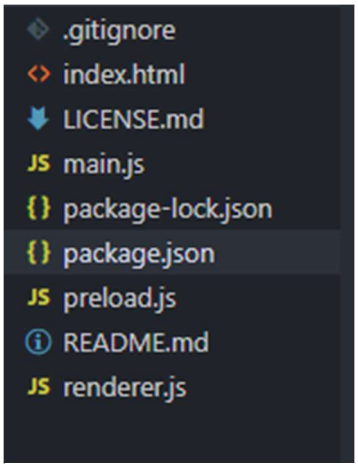
Pokud je ale toto a také fakt, že oba frameworky mají vlastní nastavení Webpacku, bráno v potaz,je pravděpodobné, že jejich kombinace nebude až tak jednoduchá, jelikož konfigurace Webpacku může být poměrně složitá

5.3.1 Electron + Vue (bez Webpacku)

Nejprve bude zhodnocena možnost bez využití Webpacku.

Jako výchozí bod bude využito šablony *electron-quick-start* (zmíněné v předchozí sekci), který poslouží jako dostatečný a přehledný základ.

Obsah pak vypadá přibližně takto:



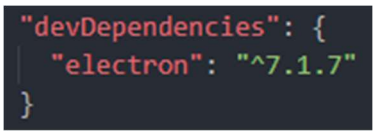
Obr. 1 – Obsah složky electron-quick-start

V package.json je pak možné vidět, že jedinou závislostí je právě Electron. Main.js obsahuje základní kód pro běh aplikace, zejména hlavní okno. *Index.html* základní hello world stránku a odkaz na renderer.js, který je prázdný.

Vue pak bude nainstalováno za pomocí npm příkazu:

```
npm install vue
```

Tento příkaz nainstaluje potřebné moduly pro běh Vue.js a přidá ho do závislostí. Kód Vue pak bude umisťován do souboru renderer.js, a jelikož není užíván webpack, je nutno použít verzi Vue umožňující vývoj bez použití webpacku, jež se nazývá *runtime-only*.



Obr. 2 – Electron v package.json

Dalším krokem je vytvoření jednoduché "Hello World" aplikace. V *index.html* v body tagu bude smazán současný obsah; ponechán bude pouze script odkaz na renderer.js. Zbývající kód bude nahrazen div tagem s id app. Bude tedy vypadat takto:

```
<body>
  <div id="app"></div>
  <script src="./renderer.js"></script>
</body>
```

Dále do v *renderer.js* bude přidán následující Vue kód:

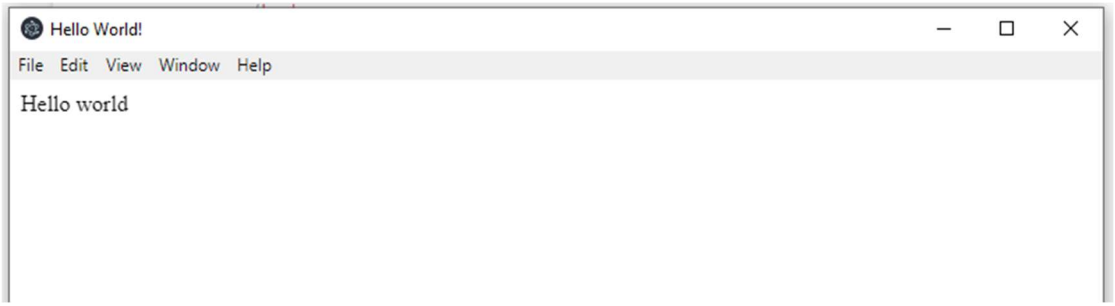
```
const Vue = require('vue')

const txt = 'Hello world'
new Vue({
    render: h => h('div', txt)
}).$mount(`#app`)
```

Je také nutné v *main.js* nastavit při tvoření okna parametr *nodeIntegration* na *true*, aby bylo možné v render procesu načítat node.js moduly viz. Vue.

```
mainWindow = new BrowserWindow({
      width: 800,
      height: 600,
      webPreferences: {
      preload: path.join(__dirname, 'preload.js'),
      nodeIntegration: true
      }
})
```

Nyní je možné zadáním příkazu *electron .* nebo *npm run start* spustit aplikaci. Po spuštění lze vidět, že kombinace těchto dvou funguje, jelikož text *„Hello world“* se zobrazil.



Obr. 3 – Ukázka electron aplikace

Nicméně tento přístup funguje jen díky užití verze Vue, která nepotřebuje ke svému fungování kompilaci kódu. Toto není však podporováno všemi frontend frameworky a často obnáší ztrátu některých funkcionalit a optimalizace kterou Webpack nabízí. Dále v případě Vue dochází ke ztrátě efektivity a jednoduchosti vývoje, které umožňuje Vue s kompilací a *„.vue“* soubory.

5.3.2 Electron + Vue (Webpack)

Další zkoumanou možností bude užití Webpacku k propojení Electronu a Vue.

V tomto případě bude prozkoumán *Vue CLI Plugin Electron Builder* [15], který umožňuje získat základ pro vývoj ve Electronu a Vue jednoduše, a bude prozkoumáno, jak funguje.

Postup bude odlišný než v předešlém případě, jelikož se jedná o plugin Vue.js. Nejdříve bude nainstalován Vue pomocí Vue CLI 3, a poté daný plugin pro Electron.

Vue.js bude nainstalován pomocí příkazu: (předpokladem je již globálně nainstalovaný Vue CLI 3)

```
vue create appname
```

Plugin bude poté přidán příkazem:

```
vue add electron-builder
```

Výsledný adresář je již mnohem rozsáhlejší než v předchozím případě, obsahuje rozdělení na podadresáře jako *public, src* a podobně...

Proces main je schován v souboru *background.js* a render proces je daná Vue.js aplikace iniciovaná v *main.js*

Jak daný stack funguje je patrné z *package.json*, kde jsou již připravené příkazy na práci s frameworkem, z nichž jsou nejdůležitější příkazy serve a build, viz dokumentace.

Script **serve**, který slouží ke spuštění pro účely vývoje, funguje tak, že nejprve je spuštěn Webpack dev server pro Vue, který je upraven pro fungování spolu s Electronem, dále je sestaven main process v režimu vývoje, a nakonec je spuštěn samotný Electron a v něm otevřeno url daného serveru.

Script ***build***, sloužící ke kompilaci a sestavení aplikace pro účely distribuce, funguje tak, že nejprve je sestaven render proces neboli Vue aplikace, dále je sestaven main proces neboli samotný Electron a nakonec je využit *electron-builder* pro kompilaci do desktopové aplikace, i s možností generace instalačního souboru. Build se může lišit na základě platformy, pro kterou je daná aplikace sestavovaná.

Po zadání příkazu:

```
npm run electron:serve
```

se tedy spustí předpřipravená ukázková aplikace.

5.3.3 Závěr

Možnost užití frontend frameworků má jistě mnoho výhod, ale jejich implementace a konfigurace, jak je patrné, může být složitější, než by se mohlo na první pohled zdát, a právě proto také nejspíše existuje mnoho šablon, které nabízejí již připravený základ pro usnadnění vývoje.

Velkou výhodou může být, že frontend vytvořený za pomocí těchto frameworků může být podobný, ne-li totožný s takovým vytvořeným pro webové prostředí, tudíž je pravděpodobně možné snadno využít stejný frontend pro obě platformy.

Dále, jak již bylo zmíněno, tvorba uživatelského rozhraní pomocí těchto technologií je velmi přívětivá pro vývojáře a nabízí velké volnosti.

Nicméně v této práci byly prozkoumány pouze dvě možnosti kombinace a pouze jeden frontend framework, proto je dosti možné, že jiné kombinace mohou být jednodušší či složitější či zcela nemožné implementovat. Ale alespoň co se týče frameworku React.js a Angular.js lze spekulovat, i na základě dostupnosti šablon s nimi (viz. předchozí sekce), že jejich implementace bude možná.

5.4 Možnosti vývoje – Dostupné funkcionality

Electron nabízí řadu funkcionalit, které tento framework odlišují od klasických webových aplikací. Mnohé z nich jsou právě funkcionality typické pro nativní desktopové aplikace.

Následně budou představeny ty nejzajímavější z nich.

5.4.1 Okna

Asi nejzásadnější funkcionalitou, která Electron (a ostatní podobné frameworky) odlišuje od webových aplikací, je tvorba vlastních oken. Mimo hlavní okno, kde pravděpodobně aplikace poběží, je možné tvořit libovolný počet oken.

Okno je možné upravit pomocí nastavení při jeho tvorbě nebo i v průběhu, je možné nastavit všechny klasické parametry jako rozměry, pozice, typ okna atd. a i některé více specifické možnosti. Některá tato nastavení jsou pak závislá na konečné platformě, zejména typ oken a podobně. Okna mohou být řazena do hierarchické struktury rodič – potomek. Tato možnost nachází uplatnění zejména při tvoření dialogových oken a podobně.

Každé okno má svůj vlastní proces render, který lze přirovnat k stránce webové aplikace. To může komplikovat situaci při použití některých frontend frameworků, jelikož mnoho z nich funguje na bázi single-page aplikace, kde by v případě Electronu byla pro každé okno potřeba samostatná aplikace, avšak některé frameworky nabízí řešení i pro multi-page aplikace.

Komunikace mezi okny pak probíhá prostřednictvím procesu main, tudíž například data z jednoho okna musí být pomocí IPC poslána do procesu main, a poté z něj následně poslána do požadovaného okna. Toto opět může způsobit za určitých situací komplikace.

5.4.2 Menu

Další funkcionalitou Electronu je tvorba menu, a to jak aplikačního menu, nejčastěji umístěného v levém horním rohu okna, tak kontextuálních menu vyvolaných obvykle pomocí pravého tlačítka myši.

Každé okno má v základu již zabudované menu, které je pak v procesu main možno měnit. Pro menu je možné nastavit klávesové zkratky.

5.4.3 Autoupdate

Electron v základu nabízí možnost aktualizovat výslednou aplikaci pomocí update serveru; tuto funkcionalitu většinou řeší balíčky zajišťující sestavování aplikace, například již zmíněný electron builder.

5.4.4 Další funkcionality

- ***Notifikace***

Electron umožňuje využití systémových notifikací. Jejich konečná podoba se liší dle platformy a podle toho, který proces je vyvolává. Tuto funckionalitu však také v poslední době umožňují webové prohlížeče pomocí HTML5 API.

- ***Systémové dialogy***

Electron umožňuje vyvolávání systémových dialogu jako upozornění, otevíraní/ukládání souboru a podobně.

5.5 Hledání dostupných alternativ

Hledanými kandidáty jsou JS frameworky podobné principem Electronu, a to takové, které umožňují tvorbu desktopových aplikací. Alternativami budou tedy frameworky, které plní stejnou funkci jako Electron, a to možnost tvorby aplikacích oken, interakce se systémem a jiné. Tudíž se jedná o frameworky, které neslouží přímo jako část běžné webové aplikace, ale spíše zastupují prohlížeč, resp. vytvářejí ho.

Je tedy možno vyřadit frontend frameworky, jako React.js či Vue.js, jelikož tyto frameworky plní funkci zobrazování obsahu a obecně zabezpečují uživatelské rozhraní. Dále také je možno vyřadit čistě backend Node.js frameworky, jako například (Express.js),

jelikož jedná se o web frameworky sloužící účelu vývoje serverové logiky a také full-stack frameworky, zajišťující obě výše zmíněné části.

Nejlepšími nalezenými kandidáty jsou NW.js, AppJS, Proton Native a případně Vuido. [16]

- ***NW.js***

Dříve nazýván node-webkit, podobně jako Electron je knihovna využívající Chromium a Node.js. Oproti Electronu se liší v několika ohledech, jako spuštění nebo stavba aplikace. [17]

- ***AppJS***

Nejstarší z frameworků založených na Chromiu. Podpora frameworku se nicméně zdá být ukončena, jak je uvedeno na webu frameworku [18].

- ***Proton Native***

JS framework umožňující vytvářet nativní desktopové aplikace. Je založen na frameworku React.js a je inspirován frameworkem React Native, který slouží k podobnému účelu pro mobilní zařízení. Ke svému chodu nevyužívá Chromium, ale nativní knihovny dané platformy pomocí libui, konkrétně libui-node. [19] Jedná se o C GUI knihovnu dostupnou pro několik jazyků, mimo jiné libui-node pro propojení s Node.js. [20]

- ***Vuido***

Framework je inspirován předchozím frameworkem Proton Native. Jediný rozdíl je, že je založen na Vue.js frameworku. [21]

Všechny uvedené frameworky běží na Node.js backendu a využívají npm správce balíčků.

Jelikož se jedná o framework s ukončenou podporou, je vhodné vyřadit AppJS.

Porovnávání bude rozděleno na několik fází. Nejdříve mezi sebou budou porovnány frameworky na bázi Chromia, tzn. Electron.js a NW.js, následně bude porovnán Electron.js s “nativními” frameworky využívající Libui tzn. Vuido a Proton Native. Díky své velké podobnosti budou porovnávány dohromady, jelikož jejich rozdíly plynou především z rozdílů mezi Vue.js a React.js.

Při porovnání bude kladen důraz na popularitu frameworku, zejména na GitHub, jelikož z ní lze do určité míry odvodit spolehlivost frameworku.

5.6 Porovnání Electron.js a NW.js

5.6.1 Obecné pojetí

NW funguje více jako prohlížeč zobrazující webovou aplikaci, zatímco podle tvrzení autorů Electron má naopak více API pro Node.js pro low-level integraci se systémem a má simulovat nativní desktopové aplikace.

5.6.2 Obtížnost vývoje

NW.js má přímější vývoj, kdy postupujeme podobně jako při vývoji webové aplikace pomocí Node.js. Electron využívá architektury více kontextů, kdy využívá IPC zprávy ke komunikaci mezi procesy main a render. NW.js tuto komplexitu nevyžaduje, místo toho je na vývojáři, jestli si zvolí separaci kontextů či nikoli. [22]

5.6.3 Dostupnost informací

Dokumentace Electronu je lépe organizovaná než NW.js, nicméně obě jsou velmi rozsáhlé a detailní. Co se týče komunit, komunita Electronu je o něco větší a aktivnější, avšak oba frameworky mají silné komunity (více viz popularita).

5.6.4 Počáteční velikost a nadbytečný kód (bloatware)

Protože oba frameworky běží s pomocí V8 Chromium enginu, které je samo o sobě složitá aplikace, i prázdná aplikace má velikost kolem 100 MB. Každá aplikace je de facto prohlížečem (Chromium). Obecně oba frameworky často obsahují mnoho nadbytečného kódu.

5.6.5 Integrace se systémem

NW.js se snaží imitovat klasické prostředí webových aplikací a vlastní platforma je pouze schránkou na existující webové API. Electron má k dispozici víc vlastních API a API Node.js pro mnohem větší integraci s danou platformou (např. využití upozornění ve Windows 10, ukládání do lokálních adresářů atd.).

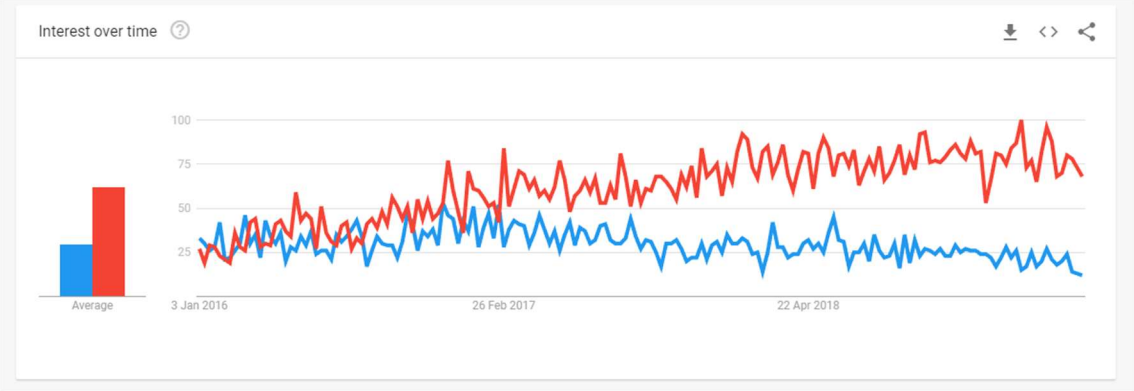
5.6.6 Zabezpečení zdrojového kódu

Electron využívá asar [23] pro vytváření balíčků, který obsahuje nechráněný kód aplikace.

NW.js využívá nwjc [24], které chrání zdrojový kód prostřednictvím kompilace do nativního kódu za cenu snížené rychlosti aplikace.

5.6.7 Popularita

Jak je patrné z grafu na Obr. 4, Electron vede v popularitě, a to od roku 2017. Popularita NW.js mírně klesá v posledních dvou letech. To samé lze vyvodit i ze statistik na GitHub. *(Obr. 5, 6)*

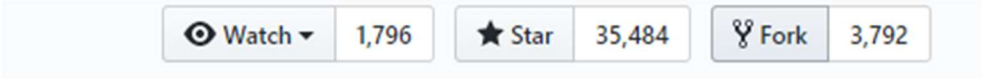


Obr. 4 – Porovnání popularity Electronu (červená) a NW.js (modrá)

Zdroj: [25]

Obr. 5 – Popularita frameworku Electron na platformě GitHub

Zdroj: [26]



Obr. 6 – Popularita frameworku NW.js na platformě GitHub

Zdroj: [17]

Co se týče známých aplikací běžících s pomocí Electronu, jsou jimi již zmíněné Textové editory Visual Studio Code či Atom, dále populární komunikační aplikace WhatsApp desktop nebo na hráče zaměřená Discord, či editor pro design ui a vektorovou grafiku Figma a další.

NW.js neuvádí oficiální portfolio, nicméně mezi známé aplikace běžící na NW.js jsou například hra Game Dev Tycoon či další hry, mnoho developerských pomůcek jako Scout App 2, jež umožňuje snadnou konverzi SASS do CSS, či Firework umožňující rychlejší spouštění webových aplikací a mnoho dalších. [27]

5.6.8 Shrnutí

Obecně z výsledků vyplývá, že Electron vede jakožto JS framework pro vývoj desktopových aplikací, a to jak v integraci se systémem, tak v dalších funkcionalitách. Také jeho větší popularita a užívání "velkými" aplikacemi jako Slack či VS Code přidává frameworku na důvěryhodnosti.

Co se týče jednoduchosti vývoje, obtížnost odpovídá obtížnosti vývoje v Node.js, a u obou frameworků je podobná, v případě Electronu může na složitosti přidávat separace procesů.

NW.js vede v zabezpečení a zdá se, že nwjc v tomto ohledu vede oproti asar.

Oba frameworky mají poměrně dost nadbytečného kódu, který výrazně zvětšuje velikost aplikace. V tomto ohledu lze očekávat, že nativní frameworky, které budou prozkoumány následovně, si povedou lépe.

5.7 Porovnání frameworku Electron.js a frameworků založených na Libui

5.7.1 Obecné pojetí

Zatímco Electron připomíná spíše webový prohlížeč simulovaný pomocí Chromia, Proton Native využívá libui k tvorbě nativních komponentů dané platformy (GTK3, Cocoa, Windows API). Všechny frameworky mají přístup k Node.js API.

5.7.2 Obtížnost vývoje

Vývoj pomocí libui se zdá být jednodušší díky využívání již předpřipravených komponent. Electron je z pohledu náročnosti vývoje mnohem složitější (viz. 5.6.2).

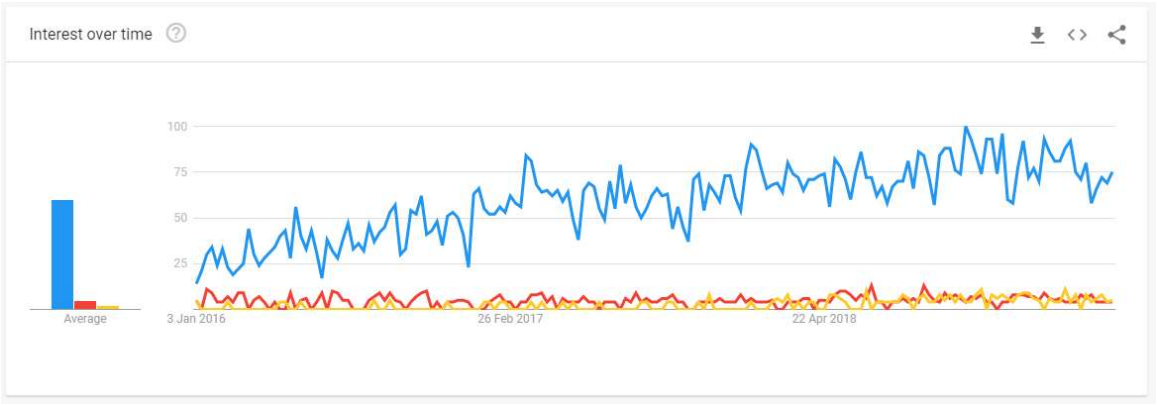
5.7.3 Dostupnost informací

Dokumentace Electronu je několikanásobně rozsáhlejší a podrobnější, než Proton Native či Vuido, nicméně z těchto dvou Proton Native vede co se týče rozsahu.

5.7.4 Počáteční velikost

Díky absenci kódu Chromia je počáteční velikost jak Vuido, tak Proton Native (cca. 20 MB), menší než Electronu (100 MB).

5.7.5 Popularita



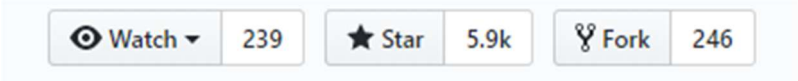
Obr. 7 – provnání popularity Electronu (modrá), Vuido (červená) a Proton Native (žlutá)

Zdroj [28]

Graf z Google Trends (Obr. 7) v tomto případě nemá příliš velkou vypovídací hodnotu, nicméně je patrné, že Electron vede ve velké míře oproti Vuido i Proton Native.

Obr. 8 – Popularita frameworku Proton Native na platformě GitHub

Zdroj [29]



Obr. 9 – Popularita frameworku Vuido na platformě GitHub

Zdroj [30]

I ze statistik z GitHub (Obr. 8, 9) je patrné, že Proton Native mírně vede co se týče popularity (alespoň na GitHub), nicméně oba se ani zdaleka nerovnají Electronu se skoro 74 tisíci stars.

Co se týče portfolia aplikací v případě Proton Native i Vuido, nebylo možné nalézt nějaký známější příklad užití. Většina aplikací vytvořených s použitím těchto frameworků je víceméně malého rozměru, bez využití pro větší projekty. Electron se může chlubit velkým portfoliem aplikací (viz 5.6.7).

Co se týče aktivity, oba libui frameworky jsou udržovány vždy jedním člověkem či malou skupinkou. U Vuido je poslední commit starý skoro rok (z ledna 2019); Proton se zdá být aktivnější s posledním commitem v lednu 2020.

5.7.6 Shrnutí

Ze zkoumání je patrné, že Proton Native i Vuido se zdaleka netěší takové popularitě a rozšířenosti jako Electron, což je nejspíše z velké části zapříčiněno jejich poměrně krátkou historií (Proton 2017, Vuido 2018, Electron 2013) [26, 29, 30].

Co se týče technických parametrů, v porovnání s Electronem jsou libui frameworky méně náročnější na paměť a jejich počáteční velikost je menší. [30] Nevýhodou ale může být závislost na nativním designu komponentů, kde není taková možnost vlastního přizpůsobení, dále jsou pak frameworky vázány na daný frontend framework. Electron je v tomto ohledu mnohem flexibilnější.

Zůstává tedy otázkou, zdali tyto frameworky dosáhnou většího využití a do jaké míry je možné spolehnout se na jejich budoucí podporu.

6 Ukázková Aplikace

Tato kapitola se zabývá vývojem ukázkové aplikace, na které budou ukázány některé dříve zmíněné vlastnosti či výhody/nevýhody frameworku.

6.1 Základní myšlenka

6.1.1 Téma aplikace

Za použití frameworku Electron byla vyvinuta desktopová aplikace. Účelem aplikace je jednoduchá evidence osobních financí, resp. transakcí, které reprezentují určité výdaje a příjmy. Tyto transakce je možné rozdělit do kategorií. Každá transakce je též spojena s účtem, který reprezentuje bankovní účet nebo hotovost. Aplikace uchovává data lokálně na disku uživatele.

6.1.2 Přístup

K vývoji aplikace bylo přistupováno tak, aby bylo možné ukázat možnost: ukládat data lokálně do uživatelova systému, proto bylo zvoleno NeDB [31] jako úložiště dat a dále také ukázat použití frontend frameworku spolu s Electronem.

6.2 Technologie

Architektura aplikace bude velmi podobná architektuře node.js webových aplikací, často skládajících se z backendu běžícím na serveru, frontendu zobrazovaným prohlížečem uživateli a často také z úložiště dat v podobě databáze.

6.2.1 Backend – Electron (main process)

Logiku na pozadí a spojení s daty zajišťuje právě zkoumaný Electron.js. Electron tedy zajišťuje tvorbu a správu okna či oken aplikace a dále tvorbu nativních menu aplikace či dalších propojení s danou platformou. Kromě toho zajišťuje spojení s databází a dotazy na ni. Data jsou pak posílána do frontendu (vykreslovacího procesu render) pomocí IPC zpráv, jak bylo již zmíněno v předešlých kapitolách.

6.2.2 Data – NeDB

Jako úložiště dat bude využito lokální databáze NeDB, která je častou volbou pro Electron aplikace. Jedná se o jednoduchou čistě JavaScriptovou NoSQL databázi založenou na systému MongoDB.

Tato technologie bude dostačující pro tuto ukázkovou aplikaci, ačkoli se jedná o technologii vhodnou spíše pro aplikace malého rozměru. V případě nárůstu obsahu potřeb aplikace by bylo vhodné pokročit na rozsáhlejší MongoDB, což se sebou nese své výhody a nevýhody, jako například nutnost databázového serveru. SQL alternativou by mohlo být například SQLite [32], které je také rozsáhlejší, ale náročnější na implementaci.

6.2.3 Frontend - Vue.js (render process)

Pro logiku zobrazení a ukázání funkce s frontendovými webovými frameworky byl zvolen framework Vue.js, který zajišťuje zobrazení uživatelského rozhraní. Jeho úkolem je tvorba komponent a zobrazení dat přijatých z backendu.

6.3 Implementace

6.3.1 Aplikace

Jako základ aplikace bylo využito již zmíněné šablony *Vue CLI Plugin Electron Builder*, která nabízí základ pro Electron aplikace s užitím Vue CLI 3.

Main process je obsažen v souboru *background*, kde je vytvořeno hlavní okno a iniciována aplikace, dále pak využívá vlastní modul *dataAccess*, který obsahuje logiku dotazování se na data.

Aplikace běží pouze v jednom hlavním okně. Pro tuto aplikaci nebylo nutné tvořit více oken a v mnoha případech bylo elegantnější a jednodušší využití frontend frameworku pro zpracování případů užití, kde by bylo vedlejší okna možné využít.

Render process zahrnuje Vue.js aplikaci, včetně všech jejích součástí jako state manager vuex, vue-router atd.

Pro mnoho UI komponent bylo využito UI knihovny *Buefy* [33] pro zrychlení a zjednodušení vývoje.

6.3.2 Data

Data jsou rozdělena do čtyř NeDB kolekcí (datastore), a to *Transactions, Categories, Accounts* a poslední *Settings* pro ukládání případných nastavení aplikace.

*Transactions o*bsahuje záznamy o transakcích, které jsou propojeny s kategoriemi, a účty pomocí id, které je automaticky generované NeDB a transakce jsou tedy na nich závislé. Každá transakce má povinně přidělen účet i kategorii, ačkoli při vymazání kategorie nebudou transakce s ní spojené smazány a jejich kategorie bude nastavena na

“undefined”. V případě vymazání účtu budou (kvůli logice vyplývající z jeho podstaty) smazány i jeho transakce.

6.3.3 Dodatečné balíčky

Mimo základu dodaného v šabloně a NeDB byly při vývoji aplikace použity tyto npm balíčky:

Momentjs [34] -	formátování času
electron-better-ipc [35] -	intuitivnější využití IPC zpráv
nedb-promise [36] -	asynchronní dotazy pro NeDB pomocí JavaScript promise

6.4 Shrnutí poznatků z vývoje

Nevyužité funkcionality

Některé funkcionality Electronu nebyly využity, a to z důvodu, že pro ně buď nebylo nalezeno využití v rámci aplikace či bylo lepší řešení pomocí Vue nebo bylo propojení s frontendem komplikované, či úplně nemožné.

Pro aplikaci byla využita dvě okna. Jedno jako hlavní okno aplikace a druhé jako „Splash window“ pro načítání aplikace. Jelikož Vue funguje jako SPA, pro každé okno by bylo nutno tvořit novou Vue instanci či je tvořit bez použití frameworku. Nicméně pro více oken nebyly nalezeny ani žádné případy užití v kontextu aplikace.

Co se týče aplikačních a kontextuálních menu, v aplikaci bylo využito pouze jednoduchého kontextuálního menu pro otevření či ukončení aplikace pomocí ikony na hlavním panelu, jiné smysluplné využití pro tuto funkcionalitu nebylo v kontextu aplikace nalezeno.

Systémová dialogová okna bylo lepší řešit přes frontend, zaprvé díky možnosti větší přizpůsobitelnosti a zadruhé odpadla nutnost posílat IPC zprávy procesu main.

Co se týče funkcionality umožňující ukládání a otevírání souborů, nabízí se zde možnost exportu a importu transakcí z/do aplikace, např. z bankovního účtu, nicméně v rozsahu ukázkové aplikace tato funkcionalita implementována nebyla.

Při vývoji bylo pozorováno, že při použití frontend frameworku se stávají některé funkcionality Electronu složitější a komplikovanější na implementaci, a pro mnoho z nich může existovat jednodušší řešení pomocí frontendu.

Další poznatky

Co se týče IPC zpráv, forma poskytovaná v základu Electronu se zdá občas být neintuitivní, a to kvůli chybějícím metododám pro přímé odpovídání na poslané zprávy. Tento nedostatek lze vyřešit dodatečným balíčkem (viz. *electron-better-ipc* či další).

Dokumentace se zdá být částmi nejasná. Například často obsahuje již vyřazené metody/funkcionalitu, ale jejichž vyřazení není vždy jasně uvedeno. To je příklad metod IPC zpráv (*invoke, handle, ...*).

Velikost sestavené aplikace je přibližně 153 MB. Zjednodušeně lze uvažovat, že cca. 100 MB je samotný Electron, zbylých cca. 53 MB je samotná logika a další součásti aplikace. Tudíž samotná logika Electronu se v tomto případě rovná přibližně 2/3 celkové velikosti. Toto potvrzuje nedostatek Electronu ve spojení s nadbytečným kódem.

Data se ukládají nešifrovaná do adresáře aplikace, tudíž nejsou zabezpečena, nicméně toto již záleží na implementaci případného zabezpečení dat, ale je vhodné tuto problematiku brát v potaz při vývoji. Jelikož se jedná o offline aplikaci, je v tomto případě odpovědnost za svá data ponechána na uživateli.

Z pohledu designu uživatelského rozhraní, tedy při využívání webového frontend frameworku či knihovny, je potřeba brát v potaz rozdílný přístup k responzivitě, která je často implementována do těchto knihoven. Může vést k nepřehlednosti výsledného rozhraní na některých zařízeních, a je tedy nutné jiného přístupu k tomuto problému než u webových aplikací.

6.5 Znovupoužitelnost

Kód aplikace je částečně znovupoužitelný. Frontendovou část aplikace by bylo také možné použít jako frontend webové aplikace. Je však nutné nahradit IPC zprávy HTTP dotazy, nahradit Electron (main process) jiným backend řešením (například Express.js) a pravděpodobně použít jiný databázový systém.

Toto je ukázkou, proč mnoho aplikací Electronu využívá vzdálené společné API, kde i desktopová verze aplikace se může přímo dotazovat na toto API přes http dotazy, a tím zaručí ještě větší znovupoužitelnost, zde se také nabízí možnost ukládat přijatá data lokálně, a umožnit tímto offline fungování aplikace i v případě napojení na toto vzdálené API.

7 Shrnutí výsledků

Ukázková aplikace měla ukázat, potvrdit či vyvrátit poznatky zjištěné při vyhodnocování frameworku, tudíž pro její vypracování byla využita šablona, užito bylo frontend frameworku Vue pro ukázání funkcionality s těmito frameworky a implementovány byly některé funkce, které Electron nabízí, a naopak nebyly využity ty, které v daném případě nebyly nejlepším řešením či by byly nadbytečné.

Použitá šablona byla velmi nápomocna při vývoji a usnadnila vývoj již předprogramovaným Webpackem. Propojením Vue a Electronu umožnila vyhnout se mnoha případným problémům a ušetřila mnoho času, alespoň v tomto případě, kde šablona má dobrou dokumentaci.

Užití frontend frameworku přináší své výhody a nevýhody. Výhodami určitě bude tvorba uživatelského rozhraní, jak ji tyto frameworky umožňují, možnost jednoduše vytvářet dynamická rozhraní a mnoho aplikační logiky může být zpracováno ve frontendu bez nutnosti přístupu na backend, který z části či úplně zastupuje samotný Electron. Tato separace také umožnuje snadné znovupoužití kódu, zejména při užití vzdáleného API. Dále možnost užití frontend UI knihovny bylo také velkým usnadněním.

Nicméně některé funkce Electronu se hůře implementují při využití těchto frameworků, například v případě více oken či menu. Dále bez využití šablony, implementace těchto frameworků může být poměrně náročná, pokud je předpokládáno zachování optimálních vývojových náležitostí, jako možnost rychlé odezvy a podobně.

Pří vývoji aplikace, jejíž primární podoba bude pouze jako desktopová aplikace, například bez online funkcionalit, je třeba zvážit do jaké míry je přínosné užití těchto frameworků a v některých případech je pravděpodobně vhodnější se jim vyhnout a využít jiných technologií.

Oproti jeho konkurenci Electronu určitě přidává vestavěná možnost aktualizací, velikost komunity, která se kolem něj vytvořila i jeho široké využití mnoha společnostmi. NW.js je jistě možnou alternativou, která nicméně postrádá některé tyto vlastnosti, které dělají Electron lepší volbou pro mnoho vývojářů. Frameworky Proton Native či Vuido nabízí řešení některých nedostatků Electronu, nicméně na druhou stranu postrádají mnoho funkcionalit a také většího rozšíření mezi vývojáři.

8 Závěry a doporučení

Cílem práce bylo zhodnotit framework Electron a zároveň zjistit jaké je jeho využití, možnosti a jaké jsou jeho přednosti a kde naopak postrádá.

Electron umožňuje využít vyspělých technologií webových aplikací, a to zejména v problematice implementace uživatelských rozhraní a také dostupnosti velkého množství knihoven, nástrojů a jiných zdrojů. Poskytuje nenáročný vývoj vyplývající z JavaScriptu a dává možnost primárně webovým vývojářům jednoduše vytvořit desktopovou aplikaci pomocí již naučených postupů a technologií.

Electron nicméně nenahrazuje veškeré možnosti desktopové aplikace. Jeho největší slabinou je samotná technologie, na níž je založen. Díky využívaní stroje V8 sdílí většinu nedostatků s webovými aplikacemi, a to zejména v ohledu výkonu a využívání systémových zdrojů a dalších limitací způsobených těmito technologiemi. Také samotný JavaScript přináší své nevýhody. Díky tomu lze usuzovat, že není vhodný pro výkonově náročné aplikace jako hry, grafické aplikace a podobně.

Jako nejvhodnější použití Electronu, jak lze vyvodit ze zkoumání i podoby mnoha aplikací na něm založených, se zdá být využití jako desktopový klient pro multiplatformní aplikace, které mají společné backend API či logiku aplikace zpracovávanou na serveru. Toto také umocňuje fakt, že je často snadné použít stejný kód pro frontendovou část aplikace v její případné webové či mobilní verzi a dále také nachází využití jako pomocný nástroj či jednoduchá aplikace; zde ale může být překážkou nemalá velikost aplikace.

Důvod pro existenci desktopové verze aplikace může být takový, že možnost mít přístup k webové aplikaci mimo webový prohlížeč přináší mnoho výhod, zejména z UX hlediska. Například mnohdy je pro uživatele snazší mít možnost spustit aplikaci přímo z prostředí systému než v prohlížeči. Dále se také aplikace Electronu vyhýbá případným zpomalením či jiným problémům vyplývajícím z nastavení uživatelova prohlížeče, např. problémům způsobených rozšířeními. Také zobrazovací logika může být vykonávána lokálně, což sníží velikost dat posílaných po síti, a tím i případnému zlepšení výkonu celé aplikace a ušetření přijatých dat.

Tudíž Electron jistě nachází využití v současném světě aplikací, což i potvrzuje jeho rozsáhlé portfolio aplikací, kde multiplatformní webové aplikace jsou na vzrůstu a mnoho z nich benefituje z existence desktopového klienta.

Seznam použitých zdrojů

1. GitHub - ElectronNET/Electron.NET: Build cross platform desktop apps with ASP.NET Core (Razor Pages, MVC, Blazor)... *The world's leading software development platform · GitHub* [online]. Copyright © 2020 GitHub, Inc. [cit. 08.06.2020]. Dostupné z: https://github.com/ElectronNET/Electron.NET
2. Documentation | Electron. *Electron | Build cross-platform desktop apps with JavaScript, HTML, and CSS.* [online]. Dostupné z: https://www.electronjs.org/docs
3. Electron Apps | Electron. Electron | *Build cross-platform desktop apps with JavaScript, HTML, and CSS.* [online]. Dostupné z: https://www.electronjs.org/apps
4. Chromium – The Chromium Projects. *The Chromium Projects* [online]. Dostupné z: https://www.chromium.org/Home
5. Google Chrome – Download the Fast, Secure Browser from Google. *Google* [online]. Dostupné z: https://www.google.com/chrome/index.html
6. Blink – The Chromium Projects. *The Chromium Projects* [online]. Dostupné z: https://www.chromium.org/blink
7. Run JavaScript Everywhere.. *Run JavaScript Everywhere.* [online]. Copyright © OpenJS Foundation [cit. 07.06.2020]. Dostupné z: https://nodejs.dev/
8. V8 JavaScript engine. *V8 JavaScript engine* [online]. Dostupné z: https://v8.dev/
9. JENSEN, Paul B. *Cross-platform desktop applications: using Electron and NW.js*. Shelter Island, NY: Manning Publcations Co., [2017]. ISBN 1617292842.
10. GRIFFITH, Chris; WELLS, Leif. *Electron: From Beginner to Pro: Learn to Build Cross Platform Desktop Applications using Github's Electron*. New York: Springer Science+Business Media New York, [2017]. ISBN 978-1-4842-2826-5.
11. GitHub - sindresorhus/awesome-electron: Useful resources for creating apps with Electron. *The world's leading software development platform · GitHub* [online]. Copyright © 2020 GitHub, Inc. [cit. 07.06.2020]. Dostupné z: https://github.com/sindresorhus/awesome-electron
12. npm | build amazing things. *npm | build amazing things* [online]. Dostupné z: https://www.npmjs.com/
13. webpack. *webpack* [online]. Dostupné z: https://webpack.js.org/
14. GitHub - electron-userland/electron-builder: A complete solution to package and build a ready for distribution Electron app with "auto update" support out of the box. *The world's leading software development platform · GitHub* [online]. Copyright © 2020 GitHub, Inc. [cit. 08.06.2020]. Dostupné z: https://github.com/electron-userland/electron-builder
15. Vue CLI Plugin Electron Builder. *A Vue Cli 3 plugin for Electron with no required configuration· GitHub Pages* [online]. Copyright © 2018 [cit. 12.01.2020]. Dostupné z: https://nklayman.github.io/vue-cli-plugin-electron-builder/

16. GitHub - sudhakar3697/electron-alternatives: Few Cross platform desktop GUI App development options are listed here. *The world's leading software development platform · GitHub* [online]. Copyright © 2020 GitHub, Inc. [cit. 08.06.2020]. Dostupné z: https://github.com/sudhakar3697/electron-alternatives

17. GitHub - nwjs/nw.js: Call all Node.js modules directly from DOM/WebWorker and enable a new way of writing applications with all Web technologies.. *The world's leading software development platform · GitHub* [online]. Copyright © 2019 [cit. 31.05.2019]. Dostupné z: https://github.com/nwjs/nw.js

18. AppJS. *AppJS* [online]. Copyright © Morteza Milani and Brandon Benvie 2012 [cit. 31.05.2019]. Dostupné z: http://appjs.com/

19. GitHub - andlabs/libui: Simple and portable (but not inflexible) GUI library in C that uses the native GUI technologies of each platform it supports.. *The world's leading software development platform · GitHub* [online]. Copyright © 2020 GitHub, Inc. [cit. 08.06.2020]. Dostupné z: https://github.com/andlabs/libui

20. Proton Native - React Native for the desktop, cross compatible. *Proton Native - React Native for the desktop, cross compatible*[online]. Dostupné z: https://proton-native.js.org/#/about

21. Introduction – Vuido. About Vuido – Vuido [online]. Dostupné z: https://vuido.mimec.org/introduction

22. Why I prefer NW.js over Electron?. *Hackernoon* [online]. Dostupné z: https://hackernoon.com/why-i-prefer-nw-js-over-electron-2018-comparison-e60b7289752

23. asar – npm. npm | *the ❤ of the modern development community* [online]. Dostupné z: https://www.npmjs.com/package/asar

24. Protect JavaScript Source Code. *NW.js Documentation* [online]. Dostupné z: http://docs.nwjs.io/en/latest/For%20Users/Advanced/Protect%20JavaScript%20Source%20Code/

25. Google *Google Trends* [online]. Dostupné z: https://trends.google.com/trends/explore?date=2016-01-01%202019-05-12&q=nwjs,electron%20js

26. GitHub - electron/electron: Build cross-platform desktop apps with JavaScript, HTML, and CSS. *The world's leading software development platform · GitHub* [online]. Copyright © 2019 [cit. 31.05.2019]. Dostupné z: https://github.com/electron/electron

27. List of apps and companies using nw.js · nwjs/nw.js Wiki · GitHub. *The world's leading software development platform · GitHub*[online]. Copyright © 2019 [cit. 12.05.2019]. Dostupné z: https://github.com/nwjs/nw.js/wiki/List-of-apps-and-companies-using-nw.js

28. Google *Google Trends* [online]. Dostupné z: https://trends.google.com/trends/explore?date=2016-01-01%202019-05-12&q=electron%20js,Vuido,Proton%20Native

29. GitHub - kusti8/proton-native: A React environment for cross platform desktop apps. *The world's leading software development platform · GitHub* [online]. Copyright © 2020 GitHub, Inc. [cit. 21.03.2020]. Dostupné z: https://github.com/kusti8/proton-native GitHub - mimecorg/vuido:

30. Native desktop applications using Vue.js.. *The world's leading software development platform · GitHub* [online]. Copyright © 2020 GitHub, Inc. [cit. 21.03.2020]. Dostupné z: https://github.com/mimecorg/vuido
31. GitHub - louischatriot/nedb: The JavaScript Database, for Node.js, nw.js, electron and the browser. *The world's leading software development platform · GitHub* [online]. Copyright © 2020 GitHub, Inc. [cit. 21.03.2020]. Dostupné z: https://github.com/louischatriot/nedb/
32. SQLite Home Page. [online]. Dostupné z: https://sqlite.org/index.html
33. Buefy: lightweight UI components for Vue.js based on Bulma. *Buefy: lightweight UI components for Vue.js based on Bulma* [online]. Dostupné z: https://buefy.org/
34. GitHub - moment/moment: Parse, validate, manipulate, and display dates in javascript.. *The world's leading software development platform · GitHub* [online]. Copyright © 2020 GitHub, Inc. [cit. 21.03.2020]. Dostupné z: https://github.com/moment/moment/
35. GitHub - sindresorhus/electron-better-ipc: Simplified IPC communication for Electron apps. *The world's leading software development platform · GitHub* [online]. Copyright © 2020 GitHub, Inc. [cit. 21.03.2020]. Dostupné z: https://github.com/sindresorhus/electron-better-ipc
36. GitHub - jrop/nedb-promise: Promisified nedb wrapper. *The world's leading software development platform · GitHub* [online]. Copyright © 2020 GitHub, Inc. [cit. 08.06.2020]. Dostupné z: https://github.com/jrop/nedb-promise

Přílohy

A. Zdrojový kód aplikace

Zdrojový kód aplikace je dostupný na https://github.com/SilvestrM/Moneynote.

UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Akademický rok: 2018/2019

Studijní program: Aplikovaná informatika
Forma studia: Prezenční
Obor/kombinace: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

Jméno a příjmení:	**Ondřej Silvestr Mikeska**
Osobní číslo:	**I1600579**
Adresa:	SNP 850, Hradec Králové – Slezské Předměstí, 50003 Hradec Králové 3, Česká republika
Téma práce:	Electron framework
Téma práce anglicky:	Electron framework
Vedoucí práce:	Mgr. Daniela Ponce, Ph.D. Katedra informačních technologií

Zásady pro vypracování:

Cílem práce je vyhodnotit framework Electron a demonstrovat jeho použití na vývoji ukázkové aplikace.

Osnova
1. Úvod
2. Metodika vyhodnocení frameworku
3. Představení frameworku
4. Vyhodnocení frameworku
5. Vývoj ukázkové aplikace
6. Shrnutí výsledků
7. Závěr
8. Seznam literatury
9. Přílohy

Seznam doporučené literatury:

- JENSEN, Paul B. *Cross-platform desktop applications: using Electron and NW.js*. Shelter Island, NY: Manning Publcations Co., [2017]. ISBN 1617292842.
- GRIFFITH, Chris; WELLS, Leif. *Electron: From Beginner to Pro: Learn to Build Cross Platform Desktop Applications using Github's Electron*. New York: Springer Science+Business Media New York, [2017]. ISBN 978-1-4842-2826-5.
- Documentation | Electron. *Electron | Build cross platform desktop apps with JavaScript, HTML, and CSS*. [online]. Dostupné z: https://electronjs.org/docs

Podpis studenta: Datum:

Podpis vedoucího práce: Datum:

© IS/STAG, Portál – Podklad kvalifikační práce , mikeson1, 10. srpna 2020 23:49

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor:	**Ondřej Silvestr Mikeska**
Studium:	I1600579
Studijní program:	B1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název bakalářské práce:	**Electron framework**
Název bakalářské práce AJ:	Electron framework

Cíl, metody, literatura, předpoklady:
Cílem práce je zhodnotit framework pro účely webové/desktopove aplikace a demonstrovat jeho použití na vývoji ukázkové aplikací.

Garantující pracoviště:	Katedra informačních technologií, Fakulta informatiky a managementu
Vedoucí práce:	Mgr. Daniela Ponce, Ph.D.
Oponent:	Ing. Martina Husáková, Ph.D.
Datum zadání závěrečné práce:	21.10.2014