VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

TECHNOLOGIE ELECTRON PRO VÝVOJ DESKTOPOVÝCH APLIKACÍ

ELECTRON TECHNOLOGY FOR DESKTOP APPLICATIONS DEVELOPMENT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **JÚLIA ŠČENSNÁ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. FILIP ORSÁG, Ph.D.**
SUPERVISOR

BRNO 2017

Zadání bakalářské práce/19424/2016/xscens00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Ščensná Júlia**
Obor: Informační technologie
Téma: **Technologie Electron pro vývoj desktopových aplikací**
Electron Technology for Desktop Applications Development
Kategorie: Web

Pokyny:

1. Seznamte se s programovacím jazykem JavaScript a návrhem vzhledu aplikace pomocí HTML a CSS pro různá multiplatformní prostředí určených pro běh desktopových aplikací založených na webových technologiích (například Electron, UWP - *Universal Windows Platform*, NW.js - *Node Webkit* a další) a vyberte některé z nich pro další práci.
2. Navrhněte aplikaci, která bude komunikovat se servery YSoft SafeQ prostřednictvím rozhraní REST.
3. Navrženou aplikaci implementujte ve vybraných prostředích a otestujte ji z hlediska funkčnosti.
4. Porovnejte implementovaná řešení především z hlediska možností internacionalizace, přístupu ke kameře, zabezpečení, výkonu a možnosti konverze HTML do PDF.

Literatura:

- JENSEN, Paul B. *Cross-Platform Desktop Applications: With Node, Electron, and NW.js*. 1. vydání. Manning Publications, 2016. ISBN 978-1617292842

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2 zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Orság Filip, Ing., Ph.D.**, UITS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstrakt

Práca sa zaoberá vývojom desktopových aplikácií založených na webových technológiách. Tieto aplikácie budú určené pre beh na majoritných operačných systémoch na trhu, konkrétne Windows, Linux a macOS. Cieľom práce je preskúmanie technológií, ktoré takýto vývoj ponúkajú a následný výber niekoľkých technológií pre ďalšiu prácu, čo zahŕňa aj implementáciu aplikácií. Následne je popísané porovnanie aplikácií a ich testovanie. Výsledkom práce sú implementované aplikácie a ich porovnanie na základe možností, ktoré poskytujú.

Abstract

The thesis is focused on development of desktop applications based on web technologies. These applications are determined to run on major operation systems on the market, namely Windows, Linux and MacOS. The aim of this thesis is to examine technologies that offer this development and to select some of the technologies for future work that includes implementation of applications into selected technologies. At the end, the thesis describes comparison of applications and their testing. The results of this work are implemented applications and their comparison on basis of possibilities they provide.

Kľúčové slová

JavaScript, HTML, CSS, multiplatformná aplikácia, desktopová aplikácia, Electron, NW.js, Haxe, Webkit, webové technológie, REST

Keywords

JavaScript, HTML, CSS, cross-platform application, desktop application, Electron, NW.js, Haxe, Webkit, web technology, REST

Citácia

ŠČENSNÁ, Júlia. *Technologie Electron pro vývoj desktopových aplikací.* Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Filip Orság, Ph.D.

Technologie Electron pro vývoj desktopových aplikací

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracovala samostatne pod vedením pána Ing. Filipa Orsága, Ph.D. Ďalšie informácie mi poskytla firma Y Soft Corporation prostredníctvom pána Mgr. Juraja Michálka. Uviedla som všetky literárne pramene a publikácie, z ktorých som čerpala.

.....
Júlia Ščensná
16. mája 2017

Poďakovanie

Ďakujem svojmu vedúcemu práce, Ing. Filipovi Orságovi, Ph.D., za všetky konzultácie a čas strávený nad vedením mojej práce. Taktiež by som rada poďakovala pánovi Mgr. Jurajovi Michálkovi z firmy Y Soft Corporation za odbornú konzultáciu pri vývoji aplikácií a rady týkajúce sa tejto bakalárskej práce.

Obsah

Kapitola 1

Úvod

Práca sa zaoberá vývojom multiplatformových desktopových aplikácií založených na webových technológiách, konkrétne sa jedná o jazyk JavaScript s návrhom vzhľadu pomocou značkovacieho jazyka HTML a kaskádových štýlov CSS. Z technológií, ktoré sú založené na webových technológiách, boli vybrané Electron, NW.js a Haxe. Všetky tieto technológie využívajú pre zobrazovanie aplikácie WebKit[1]. To umožňuje dosiahnuť rovnaký beh aplikácie na všetkých majoritných operačných systémoch na trhu, Linux, Windows a macOS. Podrobný popis technológií Electron, NW.js a Haxe vrátane histórie, vývoja a známych implementovaných aplikácií je tiež zahrnutý vrámci tejto práce.

Následne sú prostredníctvom vybraných technológií vyvinuté aplikácie, ktoré komunikujú prostredníctvom REST rozhrania so serverom YSoft SafeQ. Jednotlivé aplikácie sú porovnané z hľadiska prístupu ku kamere, výkonnosti a bezpečnosti.

REST (Representational state transfer) [7] definuje architektúru služby, pomocou ktorej môžu komunikovať počítačové systémy na internete prostredníctvom HTTP[2] dotazov. Odpoveď môže byť napríklad vo formáte XML alebo JSON. Termín *REST* prvý krát definoval vo svojej dizertačnej práci Roy Fielding v roku 2000.

1.1 Cieľ práce

Cieľom práce je preskúmať možnosti vývoja multiplatformovej aplikácie prostredníctvom nástrojov založených na webových technológiách. Niekoľko z preskúmaných technológií vybrať a implementovať pomocou nich aplikáciu, ktorá bude komunikovať prostredníctvom rozhrania REST so serverom YSoft SafeQ. Ďalším cieľom bolo porovnať navrhnuté a implementované aplikácie najmä z hľadiska internacionalizácie, prístupu ku kamere, zabezpečenia a výkonu.

1.2 Y Soft Corporation

Václav Muchna sa stal roku 2000 spoluzakladateľom a CEO spoločnosti Y Soft Corporation[3] s hlavným sídlom v Českej republike. Aktuálne pôsobí v 120 štátoch s vyše 14 000 zákazníkmi a zaoberá sa najmä správou, zabezpečením a optimalizáciou 2D tlače. V oblasti 3D tlače pôsobí v smere vzdelávania.

[1] https://webkit.org/
[2] https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol
[3] https://www.ysoft.com

Pre moju prácu je dôležitá oblasť 2D tlače, konkrétne produkt s názvom *YSoft SafeQ*. Je to systém pre centralizovanú tlač, kopírovanie a skenovanie prostredníctvom počítačovej siete. Pomáha používateľom redukovať cenu za manažovanie tlače a zároveň zvyšuje bezpečnosť dokumentov určených pre tlač. Verzia tohto produktu s kompletným balíčkom služieb, *YSoft SafeQ Enterprise Suite*, obsahuje modul pre tlač z mobilov, tlač z viacerých tlačiarní prostredníctvom jedného ovládača, možnosť účtovania za jednotlivé akcie tlačiarní alebo skenerov, overovanie totožnosti a mnohé ďalšie[4].

Aktuálne je umožnené zasielanie požiadavku pre tlač cez mobilnú a webovú aplikáciu. Pre komfort zákazníka chce firma Y Soft Corporation vyvinúť multiplatformovú aplikáciu pre desktopové počítače a základný prehľad o tejto možnosti prináša táto práca.

1.3 Štruktúra práce

V druhej a tretej kapitole je rozobraná potrebná teória k tejto práci. Druhá kapitola sa venuje základným znalostiam ohľadom multiplatformových desktopových aplikácií. Popisuje programovací jazyk JavaScript, značkovací jazyk HTML a kaskádové štýly CSS. Tretia kapitola sa venuje priamo technológiám Electron, NW.js a Haxe, ktoré sú založené na vývoji v jazyku JavaScript. Každá technológia je popísaná z hľadiska histórie, funkčnosti, vývoja a následnej distribúcii výslednej aplikácie užívateľom. V tejto časti práce sa tiež uvádzajú známe aplikácie, ktoré využívajú vybrané technológie.

Grafický návrh a vývoj implementovaných aplikácií je rozpracovaný vo štvrtej kapitole. Piata kapitola zahŕňa možnosti a porovnanie jednotlivých technológií a taktiež testovanie aplikácií. Záverečná, šiesta kapitola formuluje výsledky práce a plány pre ďalší vývoj aplikácie implementovanej technológiou Electron.

[4] https://www.ysoft.com/en/safeq-enterprise-suite

Kapitola 2

Multiplatformové desktopové aplikácie

Ako ukazujú štatistiky [6] z rokov 2008 až 2015, dospelá osoba strávi približne 2,5 hodiny denne na desktopových počítačoch. Aj keď sa zvyšuje čas strávený na mobilných telefónoch, čas strávený na desktopových počítačoch sa neznižuje. Na základe týchto štatistík sa dá povedať, že investícia času do desktopových aplikácií nie je zbytočná. Ak sa však rozhodneme vyvíjať aplikáciu pre rôzne platformy, cena vzrastá s vyššou špecializáciou vývojárov a časom stráveným nad jednotlivými aplikáciami [9]. Riešením tejto situácie je vytvoriť jednu multiplatformovú aplikáciu.

Multiplatformové aplikácie majú výhodu v spoločnom zdrojovom kóde napísanom pre rôzne operačné systémy. Pre spoločnosť, ktorá bude aplikáciu financovať, je výhodou, že má náklady len na jedného vývojára. Pre tvorbu takýchto aplikácií je možné využívať mnoho technológií, ktoré používajú rôzne programovacie jazyky ako napríklad JavaScript v kombinácii s HTML5[1]/CSS3[2] alebo C++.

Vývoj prostredníctvom technológie JavaScript je zaujímavý z toho dôvodu, že vývojár píše aplikáciu podobne ako webovú stránku. Pre vývojárov, ktorý už tvoria webové stránky môže byť preto jednoduchšie naučiť sa v tomto jazyku programovať. Ďalšou, veľmi dôležitou výhodou je fakt, že pre zobrazovanie aplikácie zväčša používame WebKit a tým máme istotu, že aplikácia pobeží na viacerých operačných systémoch podobne ako webový prehliadač. Preto sa ďalej budem zaoberať najmä technológiami, ktoré využívajú JavaScript. Sú to napríklad Electron [1], AppJS[3] a NW.js [4]. Tieto technológie zväčša pracujú s Node.js[4]. Aj keď to nebolo primárnym účelom, aj Node.js je možné použiť pre tvorbu desktopových aplikácií, ktoré pobežia na rôznych platformách. Dve majoritné technológie, ktoré využívajú node.js sú Electron a NW.js. Firmy Intel a GnorTech podporujú NW.js a podobne firma GitHub podporuje vývoj technológie Electron. Vytváranie desktopových aplikácií používaním JavaScriptu s HTML5/CSS3 mimo iné umožnilo vývojárom, ktorý sú zameraný na webové aplikácie, stať sa jednoducho vývojármi desktopových aplikácií. Táto možnosť priniesla veľký benefit v zdieľanom kóde ako webovým aplikáciám tak desktopovým aplikáciám. Vytváranie desktopových aplikácií namiesto webových má mnoho výhod. Medzi najdôležitejšie patria:

[1] https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5

[2] https://developer.mozilla.org/en/docs/Web/CSS/CSS3

[3] http://appjs.com/

[4] Node.js [12] je nástroj od Googlu pre vyvíjanie serverovej časti aplikácie použitím JavaScriptu. Je to cross-platform projekt s otvoreným softvérom, ktorý obsahuje vyše 250 000 knižníc.

- Prístup k internetu nie je nevyhnutný.
- Možnosť prístupu k operačnému systému počítača a hardwarovým zdrojom.
- Väčšia kontrola nad tým ako aplikácia zapôsobí na užívateľa. Nie je nutné zamýšľať sa nad tým aký webový prehliadač a jeho verziu bude užívateľ aktuálne využívať. Tým sa odstránila nepríjemnosť pri vyvíjaní webových aplikácií a to ladiť aplikáciu pre rôzne webové prehliadače.
- Keďže aplikácia je uložená lokálne, v počítači užívateľa, zaniká závislosť na vzdialenom servery.

2.1 JavaScript

Jazyk JavaScript [11] je vysoko úrovňový, netypovaný a interpretovaný jazyk, ktorý podporuje objektové programovanie. Je možné ním vybudovať značnú časť webovej aplikácie, ktorá sa bude odohrávať vo webovom prehliadači užívateľa. Skripty napísané v tomto jazyku spĺňajú štandardy normy ECMAScript[5]. Existuje množstvo frameworkov pre jazyk JavaScript, ktoré pridávajú ďalšie možnosti vývoja. K najznámejším patrí React[6], ktorý vyvinula firma Facebook, a Angular 2[7].

Keďže jazyk JavaScript v dnešnej dobe naberá na popularite [8], existuje mnoho nástrojov a balíčkov, ktoré uľahčujú vývoj aplikácií v tomto jazyku. Pre odstraňovanie chýb je možné využiť napríklad *Chrome Dev Tools* (nástroje pre vývojárov) vstavané do prehliadača Chrome. Samozrejme obdobné nástroje existujú aj pre iné prehliadače, avšak *Chrome Dev Tools* sa radí medzi najznámejšie. Poskytuje širokú škálu možností. Príkladom môže byť konzola, možnosť použiť body prerušenia v programe, pomocník pre optimalizáciu kódu a mnoho ďalších. Pre prácu s balíčkami napísanými v jazyku JavaSript môžeme využiť *npm*[8], nástroj pre správu týchto balíčkov.

2.2 HTML a CSS

HTML (*Hypertext Markup Language*) [11] je značkovací jazyk, ktorý sa využíva k tvorbe webových stránok od čias vzniku prvej webovej stránky na webe. HTML používa prostriedky pre formátovanie zvané značky (*tagy*). Existuje mnoho značiek pre rôzne použitie. Aktuálne je najnovšia verzia HTML HTML5, ktorá prináša mnoho výhod pre tvorbu webových stránok tak aby boli interaktívnejšie. Príkladom môže byť pridávanie zvukových nahrávok alebo videí.

CSS (*Cascading Style Sheets*) [11] je jazyk, ktorý dodatočne popisuje grafickú podobu prvkov stránky napísaných v jazyku HTML. Najnovšia verzia CSS je CSS3. Priniesla so sebou grafické výhody ako animácie, zaoblenie rohov a mnohé ďalšie.

[5] http://www.ecma-international.org/publications/standards/Ecma-327.htm
[6] https://facebook.github.io/react/
[7] https://angular.io/
[8] https://www.npmjs.com/

Kapitola 3

Technológie Electron, NW.js a Haxe

3.1 Historický úvod

Technológie Electron a NW.js majú svoje začiatky spoločné. Ako píše Paul Jensen vo svojej knihe [10], v roku 2011 sa Roger Wang snažil nájsť cestu ako skombinovať WebKit a Node.js. WebKit je základ webového prehliadača, ktorý renderuje obsah ako napríklad HTML, XML a formátovacie informácie zo súborov CSS, XSL a ďalšie. Dôvodom tejto kombinácie bolo pristupovanie k Node.js modulom z JavaScriptového kódu vnútri webovej stránky. Vznikol teda modul, ktorý Roger nazval *Node WebKit*. V roku 2012 sa k vývoju modulu pridal Cheng Zhao ako interný pracovník Intelu. Pomohol Rogerovi vylepšiť kombináciu Node.js a WebKitu čím prispel k tomu, aby sa z tohto modulu stal samostatný framework pre tvorbu desktopových aplikácií.

Prvou aplikáciou, ktorou sa dostal *Node WebKit* do povedomia a pomohla tak ďalšiemu rozvoju bol editor Light Table[1]. Ešte ten istý rok prešiel Cheng Zhao pracovať pre GitHub, kde sa podieľal na vývoji Atomu. Pre rôzne problémy skombinovať Atom a *Node WebKit*, vyvinuli vlastný framework, ktorý kombinuje Node.js a WebKit, *Atom Shell*. Prvá známa aplikácia v tomto frameworku bol textový editor Atom[2]. Postupom času Intel premenoval *Node WebKit* na NW.js a GitHub premenoval *Atom Shell* na Electron. Obe tieto technológie podporujú veľké firmy, obe majú veľkú komunitu vývojárov a používateľov a dajú sa v nich vyvinúť slušné multiplatformové aplikácie.

Projekt Haxe založil francúzsky vývojár Nicolas Cannasse v roku 2005 ako nástupcu populárneho kompilátora MTASC (*Motion-Twin Action Script Compiler*) s otvoreným softvérom. Tento kompilátor bol určený pre ActionScript 2.0 a bol napísaný v programovacom jazyku OCaml. Nikolasova záľuba k návrhu programovacích jazykov a vzostup nových príležitostí miešať rôzne technológie viedli k vývoju úplne nového jazyka. Beta verzia tohto jazyka vydaná vo februári 2006 bola nazvaná haXe a podporovala výstup v bytekóde, ktorý medzi spracovával aj virtuálny stroj Neko taktiež vyvinutý Nicolasom.

Prvá verzia Haxe, Haxe 1.0, vyšla v máji roku 2006. Bola to prvá významná zmena, ktorá so sebou priniesla podporu pre generovanie kódu pre jazyk JavaScript. Táto verzia obsahuje prvky, ktoré definujú Haxe ako ho poznáme dnes. Medzi tieto prvky sa radia napríklad automatické odvodenie dátového typu (*type inference*) a štrukturované sub-typy (*structural*

[1] http://lighttable.com/
[2] https://atom.io/

sub-typing). Haxe 1 v priebehu času zaznamenalo ešte niekoľko menej významných zmien vo forme menších verzií vydávaných pod verziou 1 (napr. Haxe 1.1). Medzi najdôležitejšie sa radí podpora ActionScript 3 a pridanie nástroja `haxelib`[3]. Niektoré boli vydané na základe opravy chýb. Vývoj v tomto období bol zameraný najmä na zlepšenie stability systému.

Ďalším významným míľnikom bolo vydanie Haxe verzie 2.0 v roku 2008. Postaral sa o to najmä Franco Ponticelli, ktorý zaviedol podporu programovacieho jazyka PHP, Hugh Sanderson s podporou jazyka C++ a Caue Waneck s podporou pre programovacie jazyky Java a C++. Haxe 2 prinieslo taktiež podporu pre makrá.

Zatiaľ poslednou zaznamenanou zmenou bola verzia Haxe 3.0 v roku 2013. Začiatky Haxe 3 započali po tom ako sa do tímu pripojil Simon Krajewski. Tím Haxe je dnes známy ako Haxe Foundation (pre viac informácii viz [2]).

3.2 Electron

Technológia Electron [1] je knižnica s otvoreným softvérom vyvinutá spoločnosťou GitHub pre vytváranie multiplatformových desktopových aplikácií prostredníctvom HTML, CSS a JavaScriptu. Využíva všetky výhody natívnej desktopovej aplikácie, ako napríklad kompletnú prácu so súborovým systémom či využívanie systémových notifikácií. Electron kombinuje Node.js a jadro Chrómia (presnejšie zdieľanú knižnicu `libchromiumcontent`[4] pre prístup ku aplikačnému rozhraniu Chrómia). Narozdiel od technológie NW.js, ktorá je postavená na rovnakom základe, Electron nezavádza nový kontext jazyka JavaScript ale využíva multi-kontext Node.js[5] (viac o kontextoch jazyka JavaScript v sekcii NW.js).

Hlavné platformy [1], na ktoré sa tento framework zameriava sú Linux, Windows a macOS. Pre operačný systém Linux je zaručený beh aplikácie pre platformy Ubuntu 12.04 a novšie verzie, Fedora 21 a Debian 8. Operačný systém Windows podporujú vývojári od verzie 7 a systém macOS od verzie 10.9.

Aplikácia napísaná v prostredí technológie Electron musí pre svoju funkčnosť obsahovať minimálne súbory `package.json`, `index.html` a `main.js` (viz obrázok číslo 3.1). Pre vytvorenie grafického užívateľského rozhrania aplikácie používa web stránky a zároveň grafické prvky natívneho operačného systému.

Aplikáciu tvoria dva druhy procesov [12], hlavný proces (*Main process*) a renderovací proces (*Renderer process*). Každý z týchto procesov zastáva rôzne role v aplikácii. Niektoré funkcie je možné vykonávať z oboch typov procesov, niektoré len z hlavného. Hlavný proces reaguje na rôzne stavy aplikácie ako zapnutie, pripravovanie na vypnutie, samotné vypnutie aplikácie, beh na pozadí a mnohé ďalšie. Tento proces môže tiež komunikovať s natívnym aplikačným rozhraním operačného systému. Ak chceme napríklad otvoriť dialóg pre uloženie súboru, musíme komunikovať z hlavného procesu. Súbor, ktorý Electron spustí ako hlavný proces, musí byť jednoznačne definovaný v súbore `package.json`. Hlavný proces môže vytvárať a ukončiť ľubovoľný počet renderovacích procesov použitím modulu `BrowserWindow`. Renderovací proces vykresľuje web stránku a tým zobrazuje grafické užívateľské prostredie. Každý z renderovacích procesov využíva výhodu multi-procesovej architektúry Chrómia a spúšťa vlastné vlákno.

[3] Haxelib je nástroj pre manažovanie knižníc Haxe [2]. Tento nástroj je aktuálne zahrnutý v Haxe Toolkite. Umožňuje vyhľadávanie, inštaláciu, upgradovanie a mazanie knižníc z haxelib odkladacieho priestoru (*repository*).

[4] https://github.com/electron/libchromiumcontent

[5] https://strongloop.com/strongblog/whats-new-node-js-v0-12-multiple-context-execution/

Narozdiel od klasickej webovej aplikácie, Electron umožňuje upravovanie grafického užívateľského rozhrania natívne podľa použitého operačného systému. Príkladom môže byť lišta okna aplikácie. Tieto úpravy však nemôže vykonávať priamo renderovací proces. Je nutné, aby všetky podobné operácie vykonával hlavný proces, teda renderovacie procesy ich musia delegovať na hlavný proces. V inom prípade by vznikla hrozba straty zdrojov.

Pre vývoj aplikácie je vhodné používať správcu balíkov pre Node.js *npm*. Pomocou tohto nástroja je možné zaznamenávanie potrebných modulov v priebehu vývoja do súboru `package.json`, spúšťať samotnú aplikáciu a mnohé ďalšie. Tiež je vhodné použiť automatické načítanie okna spustenej aplikácie v prípade, že sa sledovaný súbor zmení. Využijeme to najmä pri vývoji vzhľadu aplikácie. Modul `electron-connet`[6] spĺňa túto funkciu.

3.2.1 Komunikácia medzi procesmi

Ako bolo vysvetlené v predchádzajúcej podkapitole, v Electron aplikácii existujú dva druhy procesov, hlavný proces a renderovací proces a každý má svoju úlohu. Hlavný proces spúšťa renderovacie procesy a stará sa o komunikáciu s aplikačným rozhraním operačného systému. Každý renderovací proces vykresľuje jednú webovú stránku a beží izolovane od ostatných renderovacích procesov.

```
// Hlavný proces
global.sharedObject = {
        someProperty: 'default value'
}

// Prvá webová stránka
require('electron').remote.getGlobal('sharedObject').someProperty =
    ↪ 'new value'

// Druhá webová stránka
console.log(require('electron').remote.getGlobal('sharedObject').
    ↪ someProperty)
```

Kód 3.1: Základ IPC systému.

Komunikácia medzi procesmi[7] je v aplikácii vytvorenej prostredníctvom technológie Electron veľmi dôležitá. A to nie len pre upravovanie grafického užívateľského rozhrania natívneho operačného systému, ale aj z hľadiska zdieľania dát medzi renderovacími procesmi jednotlivých web stránok. Máme niekoľko spôsobov ako komunikovať medzi hlavným a renderovacím procesom. Najjednoduchšia cesta je použitie aplikačných rozhraní HTML5, ktoré sú aktuálne prístupné vo webových prehliadačoch. Sú to napríklad `localStorage`, `sessionStorage` a `Storage API`. Ďalším spôsobom je použitie systému špecifickému pre Electron, IPC systém. Základ IPC systému je ukladanie globálnych premenných v hlavnom procese a pristupovanie k ním pomocou `remote` vlastnosti v renderovacom procese. Jednoduchý príklad je možné vidieť v kóde 3.1.

[6] https://github.com/Quramy/electron-connect

[7] https://github.com/electron/electron/blob/master/docs/faq.md#how-to-share-data-between-web-pages

3.2.2 Odstraňovanie chýb aplikácie

V tejto podkapitole vychádzam z oficiálnych stránok technológie Electron zo sekcie návodov [1]. Bežný spôsob ako odstraňovať chyby vo webovej aplikácií, ktorý je prístupný aj pre Electron, je použitie rozšírenia *DevTools*. Avšak toto rozšírenie dokáže ladiť len kód JavaScript, ktorý je aktuálne spustený v okne webového prehliadača. Pre ladenie JavaScriptu, ktorý je spustený v hlavnom procese, je potrebné nainštalovať externý nástroj a spúšťať Electron s prepínačmi `--debug` alebo `--debug-brk`. Príkladom takýchto nástrojov sú *node-inspector* alebo doplnok editora Visual Studio Code.

Keďže *node-inspector*[8] vyžaduje niektoré moduly Node.js, je možné použiť *electron-inspector*, ktorý sa o tieto závislosti postará za nás. Samozrejme je tiež možná manuálna inštalácia nástroja *node-inspector*.

Pre ladenie pomocou editora Visual Studio Code[9] je nutné pridanie konfiguračného súboru v textovom formáte `json` do priečinka `.vscode`. Následne nastavíme zarážky do súboru `main.js` a ladíme prostredníctvom okna pre ladenie v editore Visual Studio Code.

3.2.3 Rozdistribuovanie aplikácie

Po vyvinutí aplikácie existujú tri možnosti pre jej distribúciu na vybrané platformy (viz sekcia návody v dokumentácii [1]). Prvou možnosťou je manuálne stiahnuť všetky najnovšie verzie Electronu pre potrebné platformy a nakopírovať aplikáciu do príslušného priečinku. Pre tento proces existuje nástroj z tretích strán *Electron-packager*, ktorý je spustiteľný z príkazového riadka. Druhou možnosťou je využitie nástrojov pre vytvorenie inštalačných balíčkov z tretích strán. Aktuálne môžeme využiť dva nástroje, *Electron-builder* a *Electron-windows-store*. V týchto prípadoch použitia môžeme zdrojové kódy distribuovať zabalené v balíku *asar*[10]. Poslednou možnosťou, ktorá zahŕňa distribuovanie čistého zdrojového kódu, je použitie nástroja *npm*.

Nástroj *Electron-packager*[11] stiahne vybranú verziu Electronu pre vybrané operačné systémy spolu so spustiteľnými binárnymi súbormi. Prostredníctvom príkazového riadka je možné upresniť o aké platformy máme záujem. Na príslušné miesta špecifické pre každý operačný systém nakopíruje tento nástroj zdrojové súbory vytvorenej aplikácie. *Electron-packager* sa používa výlučne prostredníctvom príkazového riadka a nevytvára súbory určené pre inštaláciu.

Electron-builder[12] je nástroj, ktorý využíva balíčky vytvorené prostredníctvom nástroja *Electron-packager* a vytvára súbor potrebný pre inštaláciu aplikácie. Formát inštalátoru, ktorý *Electron-builder* podporuje pre platformu Windows je `NSIS`. Z najpoužívanejších formátov balíčkov pre Linux sú to `deb`, `rpm`, `freebsd` a `apk`. Podobne pre macOS sú to `dmg`, `pkg` a `mas`. Tento nástroj naviac poskytuje podporu pre automatické aktualizácie. Kompilácia aplikácie prostredníctvom doplnku *Electron-builder* [3] závisí na závislostiach aplikácie. Ak existujú závislosti, ktoré sú natívne danej platforme, je nutná kompilácia na danej platforme. Ak takéto závislosti neexistujú, je možná kompilácia buď na niektorom zvolenom operačnom systéme alebo využitie serverov pre zostavenie aplikácie. Príklad takéhoto ser-

[8] http://electron.atom.io/docs/tutorial/debugging-main-process-node-inspector/

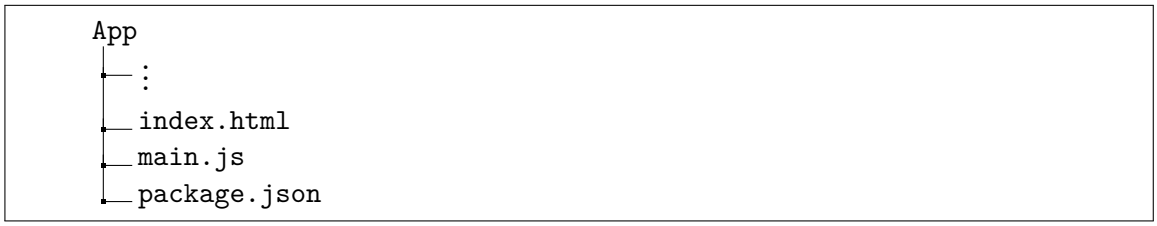
[9] http://electron.atom.io/docs/tutorial/debugging-main-process-vscode/

[10] https://electron.atom.io/docs/tutorial/application-distribution/#packaging-your-app-into-a-file

[11] https://github.com/electron-userland/electron-packager

[12] https://github.com/electron-userland/electron-builder

vera pre cieľovú platformu Windows je AppVeyor[13] a pre Linux a macOS Travis[14]. Na obrázku číslo 3.1 je znázornená štruktúra aplikácie bez použitia nástroja *Electron-builder* v porovnaní so štruktúrou aplikácie pri použití tohto nástroja na obrázku číslo 3.2. V druhom prípade je potrebné mať jeden súbor s názvom `package.json` určený pre aplikáciu, v ktorom sa nachádzajú minimálne informácie o verzii aplikácie, jej názov a popis. Do tohto súboru sa tiež ukladajú informácie o moduloch potrebných pre beh aplikácie. Ďalší súbor s názvom `package.json` sa nachádza v nadradenom adresári než zdrojové súbory aplikácie a slúži pre informácie a skripty týkajúce sa vývoja a rozdistribuovania aplikácie.



Obr. 3.1: Minimálna štruktúra dokumentu bez použitia nástroja *Electron-builder*.

Obr. 3.2: Minimálna štruktúra dokumentu s použitím nástroja *Electron-builder*.

Tretím nástrojom je *Electron-windows-store*[15], ktorý vyvinula firma Microsoft pre kompiláciu Electron aplikácie do balíka vo formáte aplikácie pre Windows store, teda vo formáte `.appx`. Nástroj sa využíva prostredníctvom Microsoft PowerShell[16]. Pred použitím nástroja *Electron-windows-store* je nutné vlastniť certifikát, ktorý slúži k podpísaniu danej aplikácie, taktiež Windows 10 SDK[17] a Node.js minimálne verziu 4. *Electron-windows-store* je možné využiť len na operačnom systéme Windows 10 s výročnou aktualizáciou (*Windows 10 Anniversary Update*). Výslednú aplikáciu vo formáte `.appx` vygeneruje z balíčka pre Windows vytvoreného prostredníctvom nástroja *Electron-packager*.

Použitie nástroja *npm* pre distribuovanie aplikácie zahŕňa nutnosť zaznamenania všetkých potrebných Node.js modulov v súbore `package.json`. Užívateľ si následne stiahne zdrojový kód aplikácie a spustí príkazy popísané v kóde 3.2, ktoré zabezpečia inštaláciu modulov Node.js a spustenie aplikácie.

Prostredníctvom nástroja *Electrify*[18] máme tiež možnosť vytvoriť z existujúcej webovej aplikácie funkčnú desktopovú aplikáciu v Electrone.

[13] https://www.appveyor.com/
[14] https://travis-ci.org/
[15] https://github.com/felixrieseberg/electron-windows-store
[16] https://msdn.microsoft.com/en-us/powershell/mt173057.aspx
[17] https://developer.microsoft.com/en-us/windows/downloads/windows-10-sdk
[18] https://github.com/jyapayne/Electrify

```
# inštalácia potrebného modulu Node.js a jeho zápis do súboru
   ↪ package.json ako závyslosť pri vývoji aplikácie
$npm install [meno_modulu_nodejs] --save
# inštalácia potrebného modulu Node.js a jeho zápis do súboru
   ↪ package.json ako závyslosť pre beh aplikácie
$npm install [meno_modulu_nodejs] --save-dev
# inštalácia všetkých modulov Node.js zapísaných v súbore package.
   ↪ json
$npm install
# spustenie aplikácie
$npm start
```

Kód 3.2: Súhrn príkazov *npm* manažéra balíčkov Node.js pre zápis modulov do súboru `package.json`, inštaláciu modulov a spustenie aplikácie.

3.2.4 Známe aplikácie a aktualizácie

Electron využívajú malé aj veľké spoločnosti [12] pre budovanie desktopových aplikácií. Prvou aplikáciou, ktorou sa Electron dostal do povedomia, bol textový editor Atom od spoločnosti GitHub. Následne vyvinula firma Facebook balíček `nuclide` do Atomu, ktorý prináša mnoho výhod vo vývoji pre web a mobily. Firma Microsoft tiež vyvinula svoj textový editor Visual Studio Code v prostredí Electronu, vďaka ktorému je táto aplikácia dostupná z operačného systému Windows, Linux aj macOS. Startup Jibo[19] stavia na technológii Electron, aby sprístupnil ľahší a pohodlnejší vývoj softwaru programátorom z tretích strán. K známym aplikáciám sa radí tiež Slack, aplikácia pre komunikáciu v tíme, alebo Brave, webový prehliadač. Viac vytvorených aplikácii je možné násť na oficiálnych stránkach [1] alebo v knihe od autora menom Steve Kinney [12].

Nové verzie technológie Electron vychádzajú z troch hlavných dôvodov. Prvým je vydanie novej verzie Chromia. Pri tejto zmene trvá zhruba dva týždne, kým vývojári zverejnia novú verziu Electronu. Druhým dôvodom je vydanie novej verzie Node.js a v tomto prípade je nová verzia Electronu oneskorená zhruba o mesiac. Posledným dôvodom je prípadné odstránenie chýb v aplikácii.

3.3 NW.js

Zakladateľ NW.js je spomínaný Roger Wang [10], ktorý produkt vyvinul ako zamestnanec centra otvoreného softvéru vo firme Intel v roku 2011. Podobne ako Electron aj NW.js je založený na kombinácii Node.js a WebKitu. Skombinovaním Node.js a Chromia sa dosiahlo interakcie aplikácií s operačným systémom skrz JavaScriptové aplikačné rozhranie. Túto možnosť webové aplikácie neumožňovali. Cieľové platformy NW.js sú Linux, macOS a Windows [10].

NW.js prináša mnoho výhod [5] ako využitie natívneho užívateľského rozhrania, čo umožní budovať aplikáciu s prvkami daného operačného systému. Umožňuje tiež ľahko pretvoriť webovú aplikáciu na desktopovú aplikáciu. Vďaka príkazovému riadku a modulov z tretích strán môžme relatívne ľahko [5] odstraňovať chyby, zabaliť a spustiť aplikáciu na platformách Windows, macOS a Linux.

[19] https://www.jibo.com/

K nevýhodám [5] sa radí veľkosť aplikácie na disku (70-90 MB v základe) v porovnaní s natívnou aplikáciou. Kompresia nám umožňuje znížiť túto veľkosť na zhruba polovicu. Ďalšou nevýhodou je zložitá distribúcia aplikácie na Mac App Store a chýbajúca podpora pre iOS a Android.

3.3.1 Vývoj aplikácie

Binárne súbory NW.js je možné stiahnuť z oficiálnych stránok. Druhou možnosťou ako sa dostať k spustiteľným binárnym súborom, je ich vytvorenie vlastnoručne podľa oficiálnych inštrukcií na stránke NW.js v sekcii návodov [4]. K dispozícii je viac typov balíkov, ktoré sa líšia najmä použitím a ich veľkosťou. Balík typu *NaCl* obsahuje *NaCl* modul, ktorý slúži pre podporu spustenia skompilovaného kódu napísaného v jazyku C a C++ nezávisle na operačnom systéme. Balík SDK obsahuje nástroje pre vývojárov (*DevTools*) a taktiež *NaCl* modul. Je určený pre vývoj aplikácie a zaberá viac miesta na disku než ostatné typy. Posledný typ balíka je *Normal*. Ten zaberá najmenej miesta na disku než spomínané typy balíkov a je vhodný pre distribúciu aplikácie užívateľom.

Základná aplikácia vyžaduje minimálne dva súbory a to súbor `index.html`, ktorý je označený ako hlavný v súbore `package.json` a samotný súbor `package.json`. Tieto súbory uložíme do priečinku so stiahnutým balíkom NW.js a spustíme binárny súbor s príponou podľa platformy, pre ktorú je určený. Narozdiel od technológie Electron, NW.js umožňuje spúšťať prvý súbor napísaný nie len v JavaScripte ale aj v jazyku HTML.

Všetky aplikačné rozhrania NW.js sa nachádzajú v jednom globálnom objekte `nw` a môžu byť použité priamo v JavaScriptových súboroch. Nevýhodou NW.js je, že ak chceme inštalovať natívne moduly Node použitím príkazu `npm install`, musíme znovu preložiť zdrojový kód NW.js s parametrom `nw-gyp`.

Po vyvinutí aplikácie je možné zdrojové kódy v JavaScripte zabezpečiť použitím nástroja `nwjc` [5]. Nástroj je možné nájsť len v balíčku typu SDK na oficiálnych stránkach NW.js [4] a používa sa z príkazového riadka. Potrebný kód v JavaScripte prostredníctvom `nwjc` skompilujeme do binárneho súboru, viz. kód číslo 3.3, ktorý budeme distribuovať namiesto originálneho zdrojového súboru. Tento binárny súbor vložíme na potrebné miesto prostredníctvom kódu číslo 3.4 v JavaScripte. Nevýhodou, ktorú poznamenávajú vývojári na oficiálnych stránkach, je zníženie výkonnosti pri spustení skompilovaného kódu o zhruba 30 percent. Platí to však len pri verzii NW.js 0.22 a nižšej. Binárne súbory tiež nie sú kompatibilné medzi verziami NW.js a nie sú multiplatformové čo znamená, že aplikácia bude vyžadovať kompiláciu s každou verziou NW.js pre každý operačný systém.

```
$nwjc source.js binary.bin
```

Kód 3.3: Použitie nástroja `nwjc`.

```
$nw.Window.get().evalNWBin(frame, 'binary.bin');
```

Kód 3.4: Vloženie binárneho súboru do JavaScriptového kódu.

Pri vývoji vzhľadu aplikácie značne urýchli prácu automatické načítanie okna [5] NW.js v prípade, že sa súbor zmení. V kóde 3.5 je znázornená najjednoduchšia aplikácia tejto funkcie. Modifikáciou kódu zabezpečíme rekurzívnu kontrolu podpriečinkov avšak len pre operačné systémy Windows a macOS. Pre robustnejšie riešenie [4] môžeme použiť Node.js balíčky `gaze`, `chokidar` alebo `gulp`.

```
<script>
        var path = './';
        var fs = require('fs');

        var reloadWatcher=fs.watch(path, function() {
                location.reload();
                reloadWatcher.close();
        });
</script>
```

Kód 3.5: Kód v JavaScripte, ktorý zabezpečí automatické načítanie okna aplikácie NW.js pri zmene určeného súboru.

3.3.2 Kontexty jazyka JavaScript

NW.js využíva architektúru aplikácií firmy Chrome, teda využíva kontext jazyka JavaScript pre prehliadač. Pri vytvorení okna sa vytvorí aj tento kontext. To znamená, že každé okno má vlastný globálny objekt a vlastný set globálnych konštruktorov. Tento koncept prináša výhody ako ochrana pred možnými chybami programátorov (ak programátor urobí chybu, ktorá ovplyvní okno globálne, neovplyvní tým ostatné okná). Okrem toho prináša kontext pre moduly Node.js. Technológia NW.js využíva dva módy kontextu [2], a to separovaný (*Separate Context Mode*) a zmiešaný (*Mixed Context Mode*).

V separovanom móde kontextu okrem kontextu, ktorý vytvorí prehliadač, beží v pozadí stránky separovane aj kontext pre moduly Node.js. Tento mód je v aplikácii využívaný predvolene. Od verzie NW.js 0.13 je možné využiť zmiešaný mód kontextu, v ktorom sa spolu s kontextom jazyka JavaScript pre prehliadač vytvorí aj kontext pre Node.js a beží v rovnakom kontexte ako pre prehliadač. Tento mód je možné docieliť prostredníctvom prepínača `--mixed-context` alebo pridaním položky `chromium-args` do súboru `package.json`.

3.3.3 Odstraňovanie chýb aplikácie

Pre vývoj a testovanie aplikácie [4] je odporúčaný typ balíčku SDK a to z dôvodu využitia nástrojov pre vývojarov (tzv. *DevTools*). Tie sa nachádzajú len v tomto type aplikácie. Nástroje pre vývojárov je možné otvoriť prostredníctvom klávesovej skratky F12 na operačnom systéme Linux a Windows. Alternatívou k týmto klávesovým skratkám je použiť aplikačné rozhranie NW.js `win.showDevTools()` pre DOM objekt `window`.

NW.js beží predvolene v móde separovaného kontextu. V tomto móde je možné detekovať chyby modulov Node.js v okne *DevTools* a to kliknutím pravého tlačítka myšky a následne zvolením skontrolovať stránku na pozadí. Porovnanie kontextov JavaScriptu viz podkapitola 3.3.2.

Vytvorenej aplikácii technológia NW.js umožňuje zapnúť mód vzdialeného odstraňovania chýb (*Remote debugging*) [5]. Pre aktiváciu tohto módu je nutné spustiť aplikáciu s prepínačom `--remote-debugging-port=<číslo portu>` pričom si môžme špecifikovať port, na ktorom bude *DevTools* počúvať. Po otvorení prehliadača s adresou `http://localhost:<číslo portu>` môžeme vzdialene sledovať ladiaci nástroj. Tento mód sa dá aplikovať aj v editore Sublime Text[20]. Pre ladiaci nástroj *DevTools* existuje ešte rozšírenie v prípade, že by bola potrebná detailnejšia detekcia chýb.

[20] https://www.sublimetext.com/

3.3.4 Rozdistribuovanie aplikácie

Pred samotným rozdistribuovaním aplikácie [5] je nutné zabalenie aplikácie pre rôzne platformy. Tento proces môžeme značne zjednodušiť použitím automatizovaných nástrojov `nwjs-builder` a `nw-builder`. Pred použitím týchto nástrojov musí súbor `package.json` obsahovať minimálne položky `main` a `name`, čiže určenie súboru, ktorý sa spustí ako prvý a meno aplikácie. Nástroje následne prečítajú `package.json`, stiahnu balíčky pre zvolené cieľové platformy z oficiálnych stránok NW.js a vytvoria zo zdrojových súborov potrebný spustiteľný súbor typický pre daný operačný systém. Prostredníctvom prepínačov týchto nástrojov si môžeme nastaviť vlastné ikony aplikácií, potrebné cieľové operačné systémy a mnoho ďalších možností.

Tieto nástroje však nevytvárajú inštalačné súbory, tie je treba vytvoriť dodatočne. Vývojári NW.js priamo na oficiálnych stránkach neodporúčajú žiadny nástroj pre vytvorenie inštalačných súborov, ako to bolo v prípade technológie Electron. Pre vytvorenie inštalačných súborov určených operačnému systému Windows je odporúčané použiť *Windows installer*, *NSIS* alebo *Inno Setup*. Podobne pre Linux a macOS je odporúčané postupovať podľa oficiálnych pokynov pre vytvorenie súborov potrebných pre inštaláciu a následné spustenie aplikácie (viac informácií na oficiálnej stránke [4] v sekcii návodov.).

Podobne ako v technológii Electron, aj NW.js umožňuje prostredníctvom nástroja z tretích strán, *Web2Executable*, transformovať webovú aplikáciu na funkčnú aplikáciu v NW.js [5]. Tieto dva nástroje v Pythone vytvoril Joey Paine z Kanady.

3.3.5 Známe aplikácie

Jednou z najznámejších aplikácií vytvorenou v prostredí NW.js je Intel XDK[21], prostredie pre vývoj mobilných a IoT aplikácií nezávislých na operačnom systéme. Na tejto technológii beží ešte séria hier (*Game Dev Tycoon*) a mnoho ďalších aplikácií[22].

3.4 Haxe

Haxe pozostáva z vysoko-úrovňového programovacieho jazyka s otvoreným softvérom a z kompilátora. Kombinácia jazyka Haxe a kompilátora umožňuje skompilovať programy, ktoré používajú syntax splňujúc štandard ECMAScript, do viacerých jazykov.

Programovací jazyk Haxe je silne typovaný, ale ak je to nevyhnutné typový systém môže byť pozmenený. Vďaka informácii o type môže typový systém Haxe detekovať chyby už pri kompilácii programu namiesto zachytenia chyby až za behu programu, ako by to bolo v cieľovom jazyku. Informácia o type premenných môže byť tiež využitá cieľovým generátorom ku generovaniu optimalizovaného a robustného kódu.

V tabuľke číslo 3.1 je deväť aktuálne podporovaných cieľových jazykov, ktoré majú rôzne využitie.

[21] https://software.intel.com/en-us/intel-xdk
[22] https://github.com/nwjs/nw.js/wiki/List-of-apps-and-companies-using-nw.js

Name	Output type	Main usages
JavaScript	Source code	Browser, Desktop, Mobile, Server
Neko	Bytecode	Desktop, Server
PHP	Source code	Server
Python	Source code	Desktop, Server
C++	Source code	Desktop, Mobile, Server
ActionScript3	Source code	Browser, Desktop, Mobile
Flash	Bytecode	Browser, Desktop, Mobile
Java	Source code	Desktop, Server
C#	Source code	Desktop, Mobile, Server

Tabuľka 3.1: Zoznam cieľových programovacích jazykov, ktoré Haxe podporuje [2].

3.4.1 Zameranie technológie Haxe

V tejto podkapitole vychádzam z dostupnej dokumentácie v sekcii použitie [2] na oficiálnych stránkach Haxe.

Jednou z možností ako využiť túto technológiu, je budovanie hier. Príkladom knižníc pre vývoj hier sú `OpenFL`, `Flambe`, `Kha`, `HaxeFlixel`, `HaxePunk` alebo `Nape Physics Engine`. Sú to rôzne frameworky, zväčša multiplatformové, určené pre webové prehliadače a tiež mobilné a desktopové zariadenia.

Ďalšou možnosťou vývoja sú webové stránky a aplikácie. Prostredníctvom Haxe môžeme napísať ako serverovú časť, tak aj klientskú časť. Príklady reálnych projektov sú Net Wars a LID Shirtshop. Medzi populárne knižnice patrí `Haxe JS Kit` a `Haxe React`.

Vývoj pre všetky mobilné zariadenia, ktoré majú majoritné zastúpenie na trhu, umožňuje cieľový jazyk C++ a knižnice `OpenFl`, `HaxeUI` a `StablexUI`. Pre tvorbu HTML5 mobilných aplikácií môžeme využiť technológiu špecifickú pre Haxe, Croxit, ktorá funguje rovnako ako známy PhoneGap[23].

Vyvíjať je možné aj aplikácie, ktoré sú určené pre príkazový riadok a tiež aplikácie alebo aplikačné rozhrania, ktoré môžeme zdieľať so všetkými cieľovými jazykmi (viz. tabuľka číslo 3.1), ktoré Haxe podporuje.

Poslednou oblasťou, pre ktorú je možné vyvíjať a ktorej sa budeme v tejto práci najviac venovať, sú desktopové aplikácie. Pre natívny vzhľad existuje knižnica `Waxe`, ktorá používa `WxWidgets` pre vytváranie natívneho rozhrania aplikácií všetkých majoritných platforiem. Ďalšia populárna knižnica pre vývoj aplikácií založených na HTML5 a JavaScripte je `Node-webkit-haxelib`, ktorá sa používa pre vykreslenie aplikácie spustiteľnej vo webovom prehliadači.

3.4.2 Vývoj v technológii Haxe

Prvému použitiu jazyka Haxe predchádza výber vhodného IDE a inštalácia Haxe Toolkitu. Užívateľmi Haxe je odporúčané IDE FlashDevelop[24] avšak toto IDE je určené len pre plat-

[23] http://phonegap.com/
[24] http://www.flashdevelop.org/

formu Windows. Pre iné platformy je Haxe užívateľmi odporúčané IDE Sublime Text[25]. Samozrejme existuje veľa ďalších možností pre výber s podporou Haxe.

Po spísaní jednoduchej aplikácie môžeme rovno spustiť Haxe a interpretovať kód, alebo spustením Haxe rozdistribuovať zdrojové kódy do cieľového jazyka. Napríklad do JavaScriptu, pri ktorom sa následne vygeneruje z Haxe kódu JavaScriptový kód, ktorý v súbore s príponou `.html` spustíme [2].

Pri vyvíjaní zložitejších aplikácií je vhodná inštalácia lime[26]. Lime je abstrakčná vrstva, ktorá umožňuje vyvíjať multiplatformovo. Tento nástroj výrazne uľahčuje inštaláciu ďalších kompilátorov a frameworkov.

3.4.3 Odstraňovanie chýb v aplikácii

Prostredníctvom Haxe je možné generovať tzv. zdrojové mapy, ktoré umožnia nástroju pre ladenie kódu namapovať z generovaného kódu späť originálny Haxe kód. Aplikáciu vytvorenú pre cieľový programovací jazyk je takto možné ladiť skompilovaním s prepínačom `--debug` a tým vytvoriť spolu s generovanými súbormi aj spomínaný `.map` súbor.

3.5 Zhrnutie

Pre implementáciu aplikácií som vybrala tri technológie, a to Electron, NW.js a Haxe. Tieto technológie majú spoločnú jednu dôležitú vlastnosť, umožňujú písať desktopové aplikácie prostredníctvom JavaScriptu, HTML a CSS. Grafické užívateľské rozhranie je pritom vykresľované podobne ako webová stránka.

[25] https://www.sublimetext.com/
[26] https://github.com/openfl/lime

Kapitola 4

Desktopová aplikácia YSoft SafeQ

Navrhovaná aplikácia komunikuje s YSoft SafeQ serverom prostredníctvom aplikačného rozhrania REST. Nakoľko je reálny server dostupný len zo súkromnej siete firmy Y Soft Corporation, vytvorila som pre ilustračné účely zo šablóny aplikačného rozhrania REST poskytnutého touto firmou ukážkové aplikačné rozhranie prostredníctvom webovej aplikácie Apiary[1]. Apiary je webová aplikácia, ktorú v januári roku 2017 odkúpila spoločnosť Oracle a slúži pre tvorbu dokumentácie aplikačného rozhrania.

Vytvorená aplikácia bude teda zasielať dotazy na webovú aplikáciu Apiary, ktorá následne odpovie pevne danými odpoveďami v textovom formáte JSON. To znamená, že niektoré funkcie aplikácie ako rušenie a samotná tlač dokumentov, budú spustiteľné len prostredníctvom reálnej aplikácie komunikujúcej so serverom YSoft SafeQ.

Ako bolo poznamenané, technológie Electron a NW.js vychádzajú zo spoločného historického jadra. Pri tvorbe aplikácií v týchto technológiách sú teda zdrojové kódy z 90 percent totožné. Rozdielna je hlavne ich distribúcia na rôzne operačné systémy a to najmä z hľadiska dostupnosti a jednoduchosti používania potrebných nástrojov. Aplikácia napísaná v technológii Haxe má s predchádzajúcimi aplikáciami spoločný vzhľad v podobe CSS súborov a tiež z väčšej časti HTML súbory.

Z hľadiska internacionalizácie, teda prispôsobenie použitia aplikácie vo viacerých jazykoch, je na tom NW.js a Electron rovnako. V oboch technológiách som implementovala možnosť dynamicky zmeniť jazyk za behu programu a to v rámci okna prihlásenie používateľa. Pre ukážku som zahrnula tri jazyky: slovenský, český a anglický. V technológii Haxe bola implementácia síce trochu odlišná ale taktiež celkom jednoduchá.

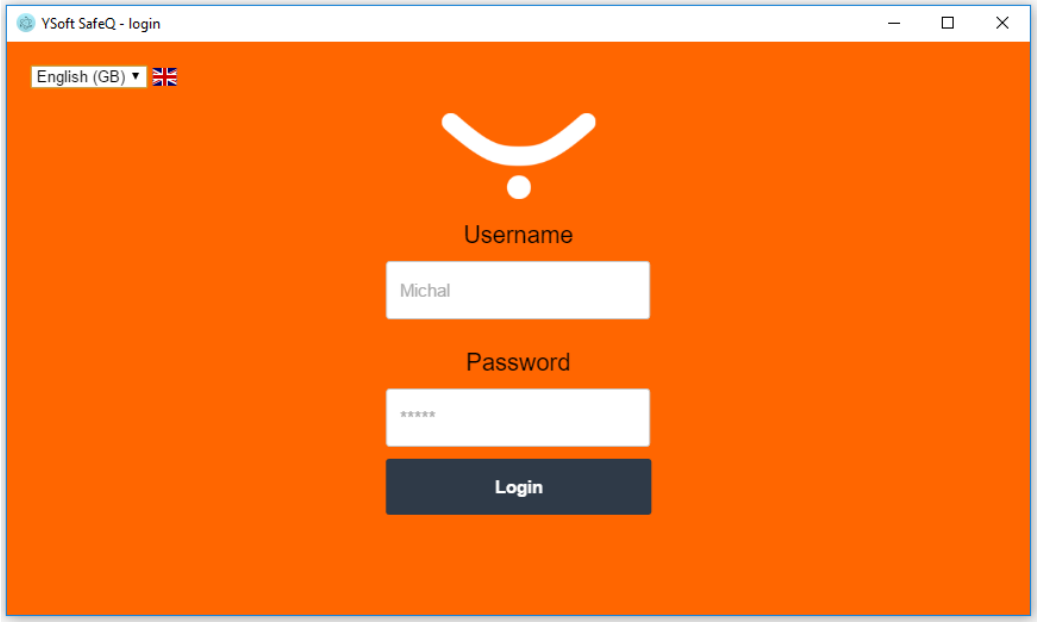
4.1 Grafický návrh

Návrh vizuálnej podoby aplikácie mi dodala firma Y Soft Corporation. Skladá sa z dvoch hlavných častí, prihlásenie a hlavná stránka so zoznamom dokumentov určených pre tlač. Výsledný grafický vzhľad aplikácií sa nachádza na obrázkoch číslo 4.1 až číslo 4.3. Pri implementácii grafickej časti aplikácií som sa striktne držala návrhu od spoločnosti Y Soft Corporation. Všetky popisované technológie sú založené na webových technológiách a pre ich grafickú časť som využila jazyk HTML v kombinácii so štýlmi CSS.

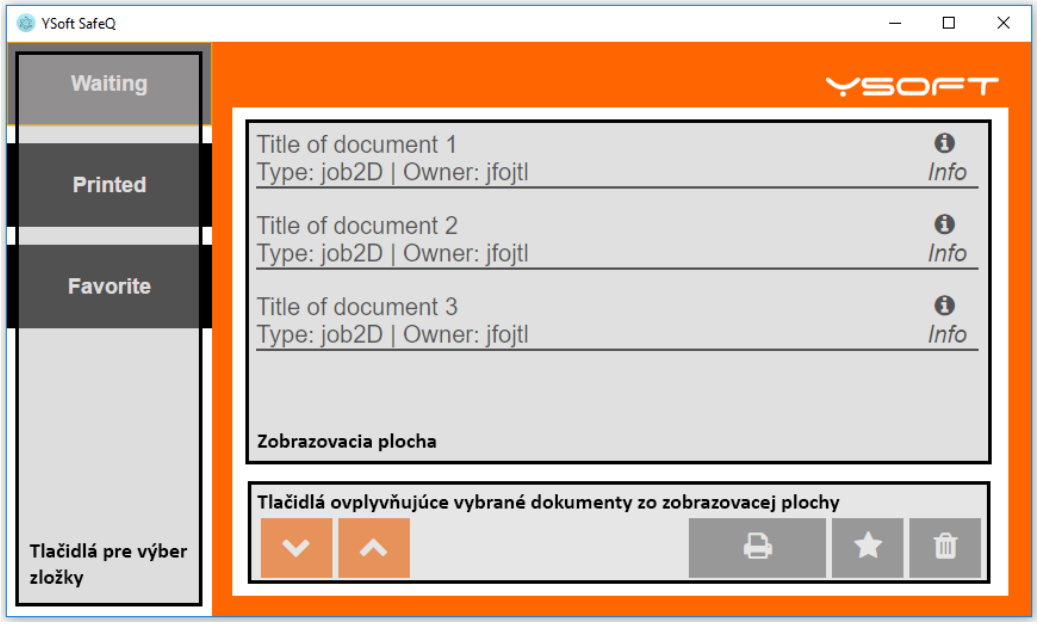
Vďaka WebKitu, ktorý používajú tieto technológie pre zobrazovanie jednotlivých stránok aplikácií, je grafický dizajn totožný na hlavných operačných systémoch na trhu, Windows, Linux a macOS. Jediné rozdiely sú v natívnych prvkoch operačných systémov ako

[1] https://apiary.io/

lišta aplikácie alebo zobrazovanie notifikácií. Pre všetky aplikácie som napísala rovnaký kód pre CSS štýly pre HTML kód, teda rozmiestnenie aplikácie. To mi umožnilo vytvoriť aplikácie s rovnakým grafickým užívateľským rozhraním nie len naprieč rôzne operačné systémy, ale aj naprieč všetky využité technológie.



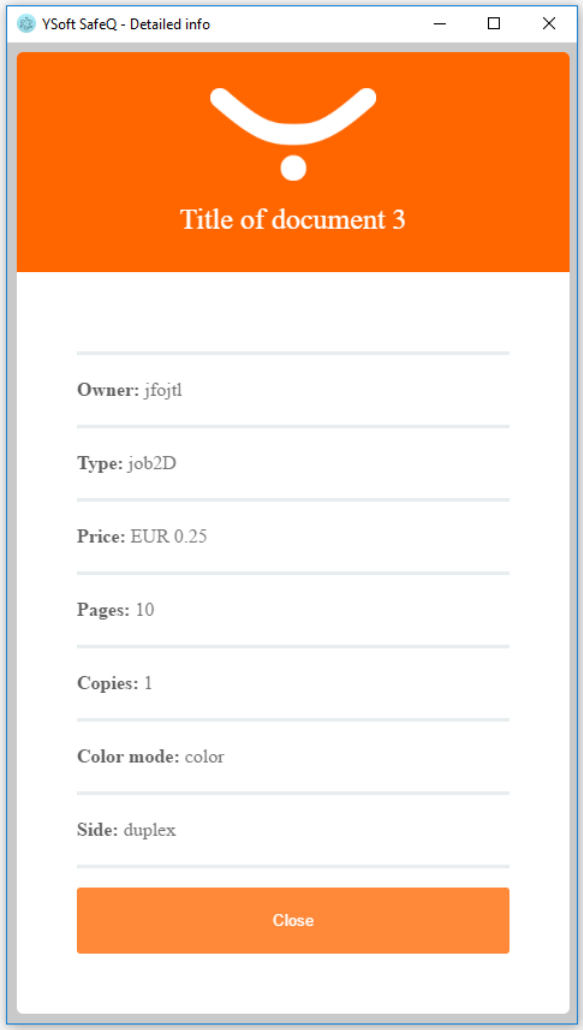
Obr. 4.1: Grafická podoba úvodnej stránky aplikácií - prihlásenie užívateľa.



Obr. 4.2: Grafická podoba aplikácií po prihlásení užívateľom (zoznam dokumentov určených pre tlač) a jej rozloženie.

Ako môžeme vidieť na obrázku číslo 4.2, kde sa nachádzajú dokumenty určené pre tlač, užívateľ si môže o danom dokumente získať detailnejšie informácie a to prostredníctvom

kliknutia na ikonu znázorňujúcu informácie. V tomto prípade sa mu zobrazí vyskakujúce okno s detailnejšími informáciami. Základný vzhľad tohto okna (viz obrázok číslo 4.3) som prevzala z webovej stránky *Designscrazed*[2] kde sa nachádzajú bezplatné šablóny tabuliek napísaných prostredníctvom jazyka HTML a štýlmi CSS.



Obr. 4.3: Grafická podoba okna s detailnými informáciami o dokumente určeného pre tlač. Užívateľ si ho zobrazí v prípade, že klikne myšou na ikonu znázorňujúcu informácie alebo na text info, ktorý je zobrazený pri každom dokumente v zozname.

Rozloženie

Hlavná stránka aplikácie, ktorá sa zobrazí po úspešnom prihlásení užívateľa, sa skladá z troch dôležitých častí (pre názornú ukážku viz obrázok číslo 4.2). Prvou sú tlačidlá pre výber zložky, vďaka ktorým si užívateľ môže prezerať dokumenty v jednotlivých zložkách (viac o funkčnosti zložiek bude popísané v podkapitole 4.2). Tieto dokumenty sa zobrazujú do druhej dôležitej časti aplikácie, zobrazovacej plochy. Dokumenty v tejto ploche je možné vyberať, zobrazovať o nich informácie a pracovať s nimi prostredníctvom tretej časti

[2] https://dcrazed.com/html-css-pricing-table-templates/

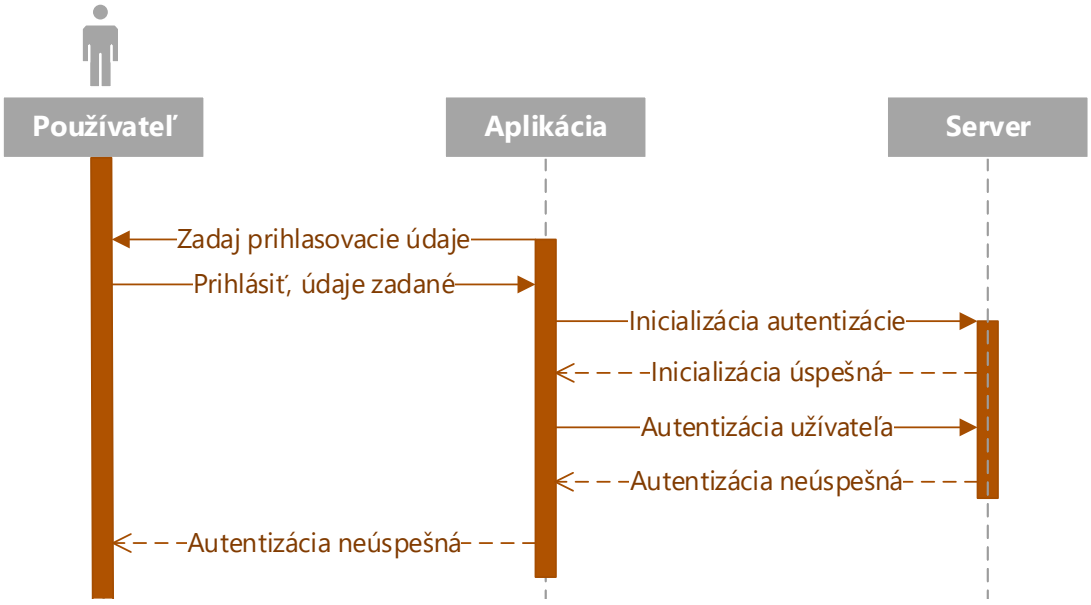
a to tlačidlami, ktoré ovplyvňujú tieto dokumenty. Nachádzajú sa tu tlačidlá pre postupný výber dokumentov, pre tlač, uloženie alebo odstránenie dokumentu zo zložky obľúbených dokumentov a odstránenie dokumentu. Na všetky tlačidlá v tretej časti je povolené klikať len za predpokladu, že je aktuálne možné s nimi vykonať nejakú akciu. Príkladom môže byť zložka, kde sa nenachádza žiaden dokument. V tomto prípade budú všetky tieto tlačidlá v stave zakázané a nebude možné na nich klikať.

4.2 Funkčnosť aplikácie

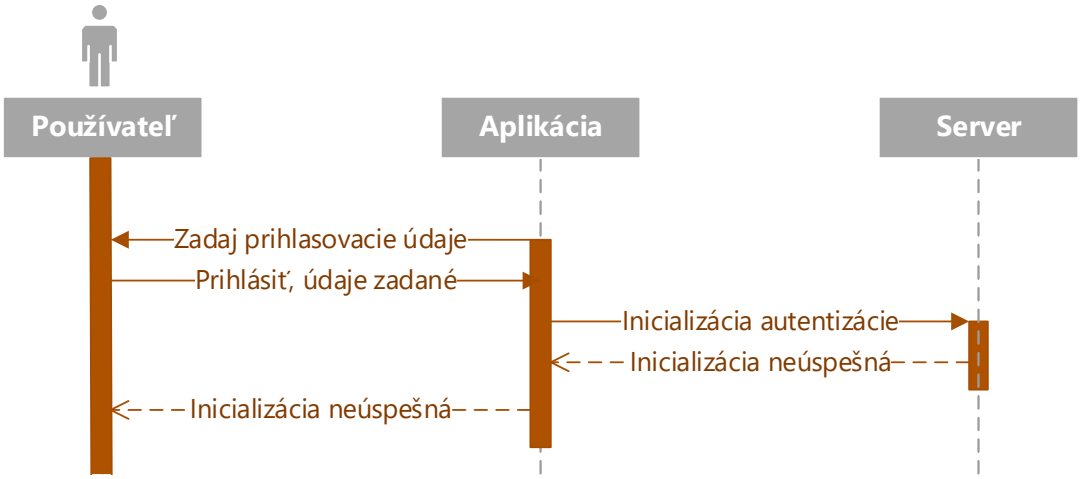
Prihlásenie užívateľa

Po zapnutí aplikácie je užívateľ vyzvaný k zadaniu prihlasovacích údajov. Aplikácia následne zaháji inicializáciu prihlásenia na server YSoft SafeQ. Tento krok sa stáva neúspešným len v prípade výpadku servera YSoft SafeQ, prípadne z dôvodu neúspešného pripojenia k sieti. Inicializácia prihlásenia slúži najmä k získaniu informácií o terminále, presnejšie konkrétnej tlačiarni. Tieto informácie sa využívajú k tomu, aby sa užívateľovi vykreslila patričná úvodná stránka k prihláseniu kartou, pinom alebo užívateľským menom a heslom. Ďalšie využitie spočíva v získaní verzie aplikačného rozhrania a tiež v testovaní spojenia aplikácie so serverom YSoft SafeQ. Všetky implementované aplikácie využívajú prihlásenie výlučne prostredníctvom zadaného mena a hesla. To znamená, že inicializáciu prihlásenia na server YSoft SafeQ využívam len pre zistenie stavu spojenia aplikácie so serverom.

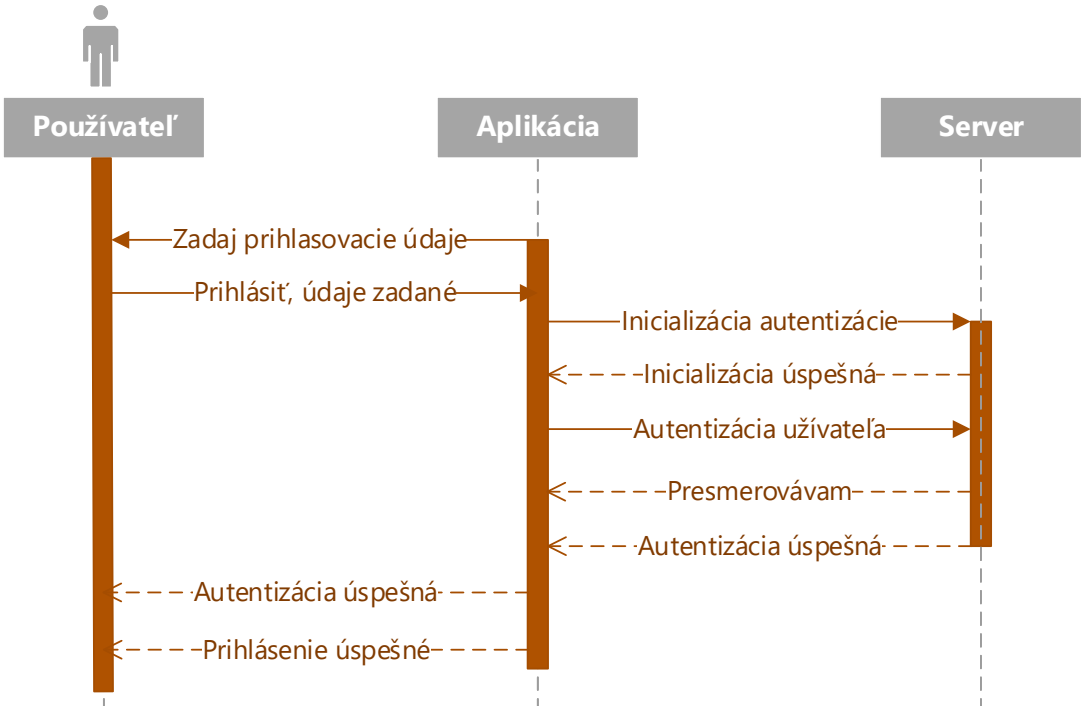
Po úspešnej inicializácii nastáva autentizácia užívateľa, teda overenie prihlasovacích údajov. Ak užívateľ zadal správne meno a heslo, autentizácia prebehne úspešne a užívateľovi aplikácia zobrazí zoznam dokumentov určených pre tlač. Následne je aplikácia pripravená na plné využívanie užívateľom.



Obr. 4.4: Časový diagram prihlásenia užívateľa s úspešnou inicializáciou ale so zlyhaním autentizácie. Po pokuse o prihlásenie s nesprávnymi údajmi YSoft SafeQ server odošle v textovom súbore informáciu o zlyhaní autentizácie v tele správy. Aplikácia detekuje túto chybu a zobrazí užívateľovi informáciu o neúspešnej autentizácii a opakovane ho vyzve o znovu zadanie mena a hesla.



Obr. 4.5: Časový diagram prihlásenia užívateľa so zlyhaním inicializácie. V prípade výpadku servera YSoft SafeQ, či chyby pripojenia k sieti, aplikácia po zahájení inicializácie získa kód odlišný od kódu 200. Užívateľovi je oznámené, že došlo k neúspešnej inicializácii komunikácie.



Obr. 4.6: Časový diagram užívateľa s úspešnou inicializáciou a autentizáciou čo zodpovedá stavu úspešného prihlásenia. Po úspešnej inicializácii komunikácie, návratovej hodnoty 200 a oznámení stavu komunikácie užívateľovi, sa aplikácia pokúša prihlásiť užívateľa k YSoft SafeQ serveru. Úspešná autentizácia prináša informáciu o tom, že užívateľ zadal správne meno a heslo. V tejto situácii zašle server návratový kód 302 značiac presmerovanie k podrobnejším informáciám. Po presmerovaní server vráti kód 200, po ktorom aplikácia zobrazí užívateľovi prostredie ku správe dokumentov určených pre tlač.

Zobrazenie dokumentov

Jednotlivé dokumenty určené pre tlač sa radia do troch zložiek: čakajúce na tlač (*Waiting*), vytlačené (*Printed*) a obľúbené (*Favorites*). Prostredníctvom tlačidiel pre výber zložky (viz obrázok 4.2) sa vypisujú dokumenty patriace pod danú zložku do zobrazovacej plochy. Po úspešnom prihlásení sa užívateľovi automaticky zobrazia dokumenty čakajúce na tlač v zložke *Waiting*. S dokumentmi zobrazenými v zobrazovacej ploche je možné pracovať prostredníctvom ovládacích tlačidiel pod zobrazovacou plochou. Je možné vykonávať akcie ako ukladanie dokumentov do zložky *Favorites*, alebo odstránenie dokumentov z tejto zložky. Ďalšími akciami sú tlač a odstránenie dokumentov. Vytlačenie dokumentu automaticky spustí akciu uloženia dokumentu do zložky *Printed*. Kompletný diagram prípadov použitia sa nachádza na obrázku číslo 4.7.

K vykonaniu týchto akcií musí aplikácia zasielať dotazy na YSoft SafeQ server. Jednoduchý dotaz v jazyku JavaScript pre získanie všetkých dokumentov v zložke *Waiting* sa nachádza v kóde 4.1. Po zaslaní dotazu aplikácia získa informácie v textovom súbore, ktorého formát sa nastavuje pri vytváraní dotazu. V mojom prípade ide o textový formát JSON. Tieto informácie musí aplikácia spracovať a vypísať užívateľovi len potrebné informácie o dokumentoch v zložke *Waiting*, teda čakajúcich na tlač.

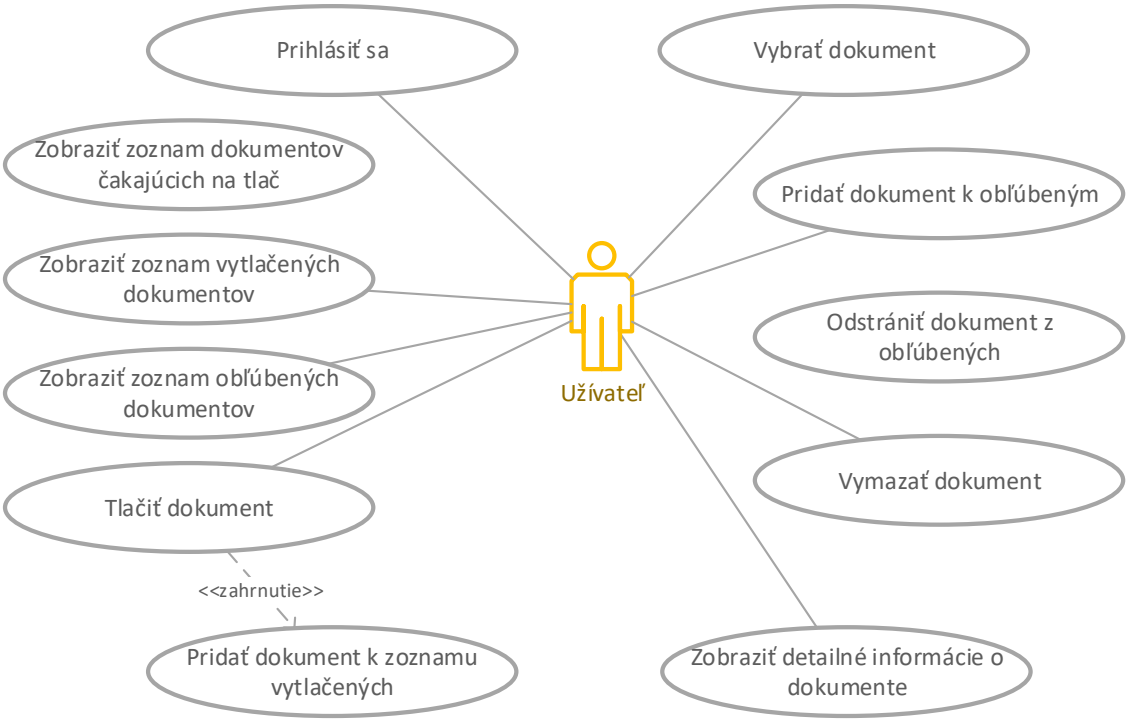
```
var request = new XMLHttpRequest();

request.open('GET', 'http://10.0.13.134:5021/et/v1/terminalId/jobs/?
    ↪ folder=Wainting/?pageSize=20');

request.onreadystatechange = function () {
        if (this.readyState === 4) {
                console.log('Status:', this.status);
                console.log('Headers:', this.getAllResponseHeaders()
                    ↪ );
                console.log('Body:', this.responseText);
        }
};

request.send();
```

Kód 4.1: Dotaz na server YSoft SafeQ, ktorý zabezpečí, že server pošle späť textový súbor s informáciami o všetkých dokumentoch, ktoré sa nachádzajú v zložke *Waiting* (prostredníctvom objektu `responseText`).

Tlač a rušenie tlače dokumentov

Pred zahájením tlače môže užívateľ vykonať posledné zmeny v nastaveniach. Dostupné nastavenia môžu byť špecifické pre každého užívateľa podľa oprávnení, ktoré dostal od zamestnávateľa, alebo podľa možností cieľovej tlačiarne. Príkladom môže byť voľba farebnej alebo čiernobielej tlače, obojstrannej tlače a ďalšie. Funkcia upravovania dokumentov pred tlačou je naplánovaná v ďalšom vývoji aplikácie.

Dokument, ktorý je naplánovaný pre tlač, môže užívateľ kedykoľvek zrušiť a vymazať. O každom dokumente si tiež môže zobraziť podrobnejšie informácie, ktoré sa mu zobrazia vo vyskakujúcom okne.



Obr. 4.7: Diagram prípadov užitia aplikáci.

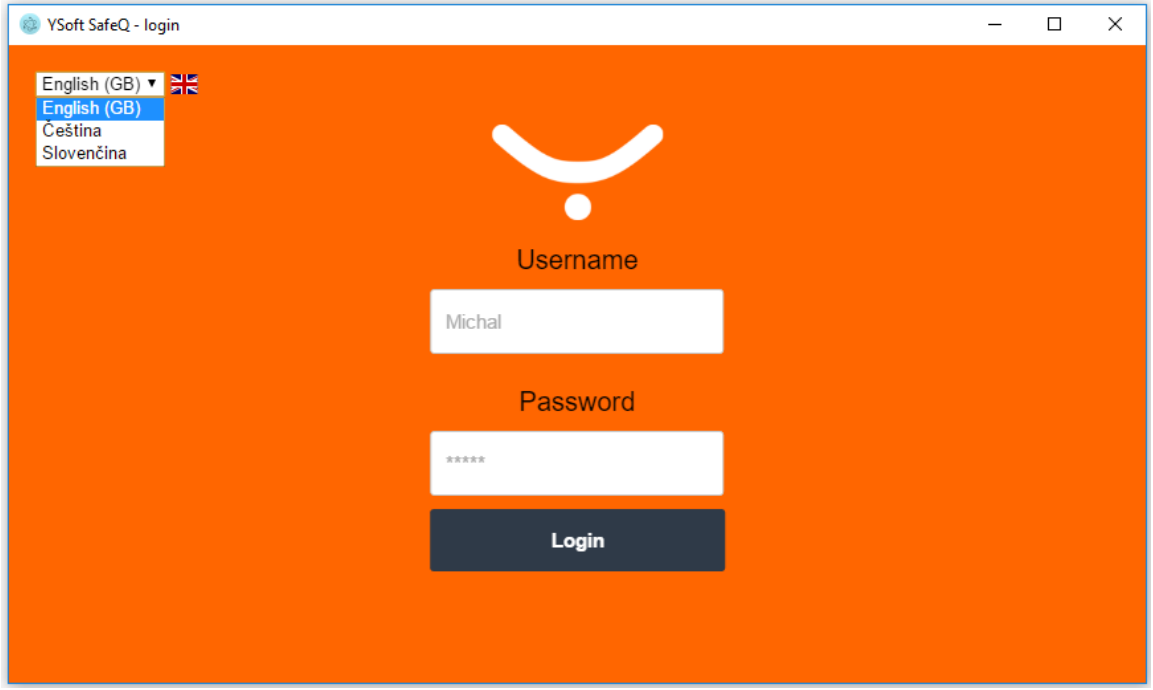
Platobný systém

Produkt YSoft SafeQ tiež umožňuje zaviesť platobný systém a uviesť tak prehľad o jednotlivých transakciách. Implementované aplikácie neumožňujú využívať platobný systém, zobrazujú len informáciu o cene jednotlivých dokumentov. Funkcia platobného terminálu je zaradená k ďalšiemu vývoju aplikácií.

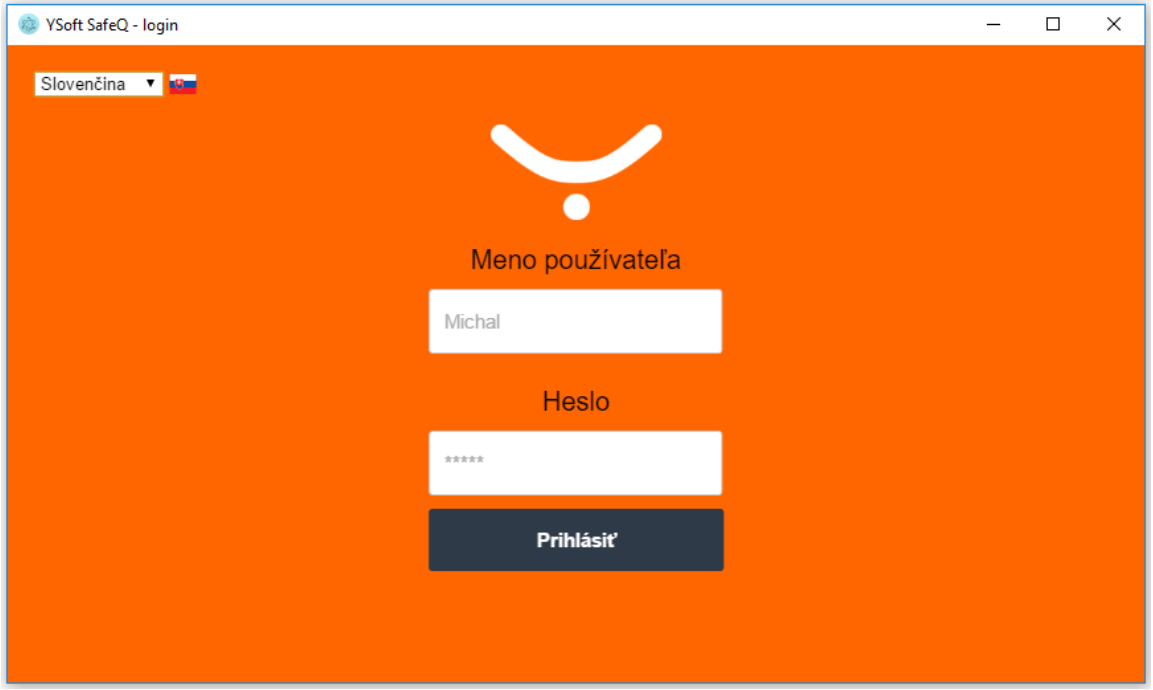
4.3 YSoft SafeQ aplikácia v technológii Electron a NW.js

Pre vývoj v technológiách Electron a NW.js som si vybrala textový editor Visual Studio Code, ktorý je zároveň napísaný v prostredí Electron. Editor Visual Studio Code ponúka možnosť kompletne upravovať svoje prostredie a to prostredníctvom súborov formátu JSON. Súbor, v ktorom sa nachádzajú východzie nastavenia, je určený len na čítanie. Samotná editácia prostredia sa vykonáva prostredníctvom podobného súboru pre užívateľa, do ktorého si môže užívateľ zapisovať svoje nastavenia, pričom sa môže inšpirovať východzím súborom. Pre tento editor existuje mnoho rozšírení, napríklad pre podporu syntaxe a odstraňovania chýb pre rôzne programovacie jazyky. Tiež je možné využiť *IntelliSense*[3], vstavané príkazy pre *Git*[4] a mnoho ďalších možností. Rozšírenia sú spustené v separátnom procese, čiže nespomaľujú editor. Výhodou tohto editoru je vstavaný terminál, vďaka čomu sa ľahko ladí aplikácia a keďže je tento editor napísaný prostredníctvom technológie Electron, existuje verzia pre všetky hlavné operačné systémy na trhu.

[3] https://code.visualstudio.com/docs/editor/intellisense
[4] https://git-scm.com/



(a) Predvolený stav aplikácie, je teda zvolený anglický jazyk.



(b) Voľba iného jazyka zo zoznamu.

Obr. 4.8: Ukážka použitia internacionalizácie v aplikácii založenej na technológii Electron.

Obe technológie NW.js a Electron zobrazujú aplikáciu v istej forme WebKitu, a preto sa zdrojové súbory vytvorených aplikácii líšia najmä hierarchiou súborov a samotným súborom, ktorý sa spustí ako prvý. Electron vyžaduje spustenie prvého súboru napísaného v JavaScripte. Naproti tomu technológia NW.js umožňuje spustiť aj súbor s príponou `.html`.

Spoločná časť zdrojových súborov teda predstavuje všetky súbory s kaskádovými štýlmi, obrázkami a písmami. Rozloženie prihlásenia užívateľa z obrázka číslo 4.1 predstavuje súbor `login.html` spolu s príslušným súborom kaskádových štýlov `login.css` v priečinku `css`. Podobne súbor index.html spolu so súborom `index.css` predstavuje rozloženie grafickej časti aplikácie po úspešnom prihlásení z obrázka číslo 4.2. Rovnako je to s vyskakujúcim oknom s detailnými informáciami o vybranom dokumente (obrázok číslo 4.3). Obrázky ikon pre informácie a pre tlačidlá pod zobrazovacou plochou sú vykreslené pomocou známeho písma *font-awesome*.

Dynamická časť aplikácie, teda kompletná funkčnosť, sa nachádza v súboroch napísaných v jazyku JavaScript v priečinku `js`. Nachádzajú sa tu funkcie pre tlač dokumentov, vytváranie zoznamu dokumentov a všetky ostatné. Akcie tlač, označenie, alebo od-značenie dokumentu ako obľúbeného a odstránenie dokumentu, musia dať užívateľovi vedieť, či sa vykonali úspešne alebo neúspešne. Tento stav užívateľovi oznamujú prostredníctvom notifikácií, ktoré sú kompatibilné so systémovými notifikáciami. Túto implementáciu prináša jazyk HTML verzie 5.

Informácia, na aký server sa má aplikácia pripojiť, je v súbore textového formátu JSON nazvanom `server_info.json`. Pre ukážkové účely je v tomto súbore adresa na webovú aplikáciu apiary, kde sa nachádza API kompatibilné s API reálneho YSoft SafeQ servera.

Prvý súbor, ktorý technológia NW.js spustí, je HTML súbor, ktorého URL je špecifikovaná v súbore `package.json`. Tým sa vo webovom okne, ako hlavné okno aplikácie, spustí rovno webová stránka. V tejto aplikácii je teda spúšťaný rovno súbor `login.html`.

Naproti tomu Electron využíva ako štartovací bod aplikácie skript napísaný v JavaScripte. Dosiahne sa tým to, že namiesto priamo definovanej URL adresy v súbore `package.json`, manuálne vytvoríme webové okno a načítame HTML súbor prostredníctvom príslušného aplikačného rozhrania. Zároveň musíme v tomto skripte riadiť chod aplikácie a komunikáciu s operačným systémom, čím dosiahneme vyššej kontroly nad samotnou aplikáciou. Všetky renderovacie procesy musia pre interakciu s operačným systémom komunikovať cez hlavný proces. V prípade implementovanej aplikácie je štartovací bod súbor `main.js`, ktorý vytvára inštanciu `BrowserWindow`. Vytvorená inštancia (renderovací proces) vykresľuje obsah webovej stránky (súbory `login.html` a `index.html`).

V aplikáciách som nakoniec implementovala možnosť zvoliť si pri prihlásení preferovaný jazyk. Potrebné súbory s názvami v jednotlivých jazykoch mi taktiež poskytla firma Y Soft Corporation. Pre ukážku je v aplikácii možnosť vybrať si z anglického, slovenského alebo českého jazyka. Na obrázku 4.8 je znázornená táto funkcia, jej umiestnenie a aplikovanie na úvodnú časť aplikácie, prihlásenie.

4.4 YSoft SafeQ aplikácia v technológii Haxe

Haxe je triedne založený jazyk a na rozdiel od jazyka JavaScript je silne typovaný. Kódy tohto jazyka sú prostredníctvom Haxe kompilátora preložené na kód v cieľovom jazyku. V mojom prípade teda na JavaScript. Z tohto dôvodu je vhodné mať editor, ktorý dokáže kompilovať a spúšťať aplikáciu, čo z časti ušetrí čas programátora. Aj keď je možné znova využiť editor Visual Studio Code, použiť rozšírenie a prispôsobiť si vývoj, zvolila som si prostredie Haxe develop, v ktorom je všetko potrebné pre vývoj hneď po inštalácii. Pri prvom spustení uľahčí začiatky s Haxe tým, že sám poskytne inštaláciu potrebných závislostí. Jediné, čo si užívateľ musí nastaviť po inštalácii závislostí a spustení editora Haxe develop je vlastný príkaz, ktorý zabezpečí, aby sa aplikácia spúšťala v technológii založenej na WebKite a nie ako okno predvoleného webového prehliadača.

Funkčná časť kódu je napísaná v jazyku Haxe a preložená do jazyka JavaScript. Tento vygenerovaný kód je následne vložený do súboru s príponou `.html` a zobrazený prostredníctvom technológie založenej na WebKite. V tvorbe desktopovej aplikácie prostredníctvom technológie Haxe založenej na WebKite je nevýhodou, že `Node-webkit-haxelib` vyvinutý v technológii NW.js od vývojárov Haxe nebol aktualizovaný od roku 2014. Nepodporuje napríklad HTML5 notifikácie, CSS selektory alebo WebGl verzie 2. Naproti tomu NW.js a Electron sú aktualizované v priemere každé dva týždne. Jednou z možností je zobrať vygenerované súbory a vykresliť ich v prostredí NW.js. Electron nie je v tomto prípade možné využiť, a to z toho dôvodu, že vyžaduje skript v jazyku JavaScript ako štartovací bod. Výhodou môže byť to, že aj keď je JavaScript netypovaný jazyk, Haxe je naproti tomu silne typový. V kombinácii s triedami v tomto jazyku pomáha programátorovi udržiavať svoj kód čitateľnejším a prehľadnejším. Aplikačné rozhrania Haxe pre JavaScript obsahujú podmnožinu webových aplikačných rozhraní popísaných na stránke *Mozzila developers network*[5]. Keďže implementovaná aplikácia nie je webová ale deskopová aplikácia, je potrebné pristupovať k súborovému systému užívateľa. Základné aplikačné rozhranie Haxe túto možnosť nepodporuje. Je nutné využiť externú knižnicu Haxe Node.js a prekladať aplikáciu s informáciou o tejto knižnici.

Rozloženie stránok má implementovaná aplikácia rovnako ako predošlé dve aplikácie, teda kaskádové štýly a z väčšej časti súbory HTML sú z nich prevzaté. Dynamická časť aplikácie je napísaná v jazyku Haxe, čo vyžadovalo implementovať potrebné triedy. Pre ukladanie dát a ich použitie v ktorejkoľvek triede som implementovala triedu podľa návrhového vzoru jedináčik (*singleton*). Uchovávať dáta je potrebné najmä pre aktuálne zvolený dokument určený pre tlač. Pre prístup k súborovému systému, ktorý bol potrebný napríklad pri implementácii internacionalizácie, som využila knižnicu `hxnodejs` [2]. `Node-webkit-haxelib` som sa rozhodla nepoužiť hlavne z dôvodu neaktuálnosti WebKitu. Taktiež nepodporuje HTML5 notifikácie, zobrazujúce úspešný alebo neúspešný stav akcií. Aplikácia sa po implementácii a vygenerovaní potrebných súborov v jazyku JavaScript zobrazuje v prostredí NW.js.

4.5 Zhrnutie

Implementovala som tri aplikácie postupne v technológiách Electron, NW.js a Haxe. Grafický dizajn, ktorého som sa striktne držala, mi poskytla firma Y Soft Corporation. Vo všetkých troch aplikáciách som dizajn písala v jazyku HTML v kombinácii s kaskádovými štýlmi CSS. Funkčná časť aplikácií prostredníctvom technológií Electron a NW.js je implementovaná v jazyku JavaScript. Každá z týchto aplikácií je zobrazovaná vo WebKite, ktorý poskytuje daná technológia. Posledná aplikácia je napísaná v jazyku Haxe a následne sú z týchto kódov generované JavaScriptové súbory. Pre zobrazenie aplikácie využívam WebKit technológie NW.js z dôvodu zastaralého produktu založenom na WebKite od vývojárov Haxe. Technológiu Electron nie je možné v tomto prípade použiť.

[5] https://developer.mozilla.org/en-US/docs/Web/API

Kapitola 5

Porovnanie technológií a testovanie aplikácií

V rámci praktickej časti tejto práce bola implementovaná aplikácia s rovnakou funkčnosťou v troch rôznych technológiách, a to Electron, NW.js a Haxe. Pre zobrazovanie grafickej časti používajú technológie istú formu WebKitu, pričom pre aplikáciu napísanú prostredníctvom Haxe sa používa WebKit technológie NW.js z dôvodu neaktuálnosti WebKitu technológie Haxe. V nasledujúcich podkapitolách je popísané porovnanie technológií z hľadiska pamäťovej náročnosti, bezpečnosti a základných možností technológií. Tiež je popísané testovanie aplikácií prostredníctvom automatických testov a testov aplikácie uskutočnených priamo vybranými užívateľmi.

5.1 Porovnanie výkonu a základných prostriedkov technológií

Jednotlivé aplikácie sa líšia najmä vo WebKite, ktorý používa technológia, na ktorej sú postavené. Tie sú testované prostredníctvom HTML5 testov popísaných viac v nasledujúcej podkapitole (číselné výsledky testov viz tabuľky 5.4 až 5.6).

Keďže špecifiká na použitie implementovanej aplikácie nevyžadujú prístup ku kamere a konverzie HTML do `pdf`, je v tomto smere aplikácia porovnaná z informačného hľadiska. Electron a NW.js umožňujú využiť metódu `getUserMedia()` webového aplikačného rozhrania `navigator`[1]. Toto aplikačné rozhranie napísané v JavaScripte umožňuje aplikáciám pristupovať ku kamere a mikrofónu. Na oficiálnych stránkach Haxe v sekcii dokumentácia aplikačných rozhraní [2] nie je táto metóda pri aplikačnom rozhraní `navigator` zahrnutá. Programátor teda nemá možnosť implementovať prístup ku kamere a mikrofónu použitím aplikačného rozhrania Haxe.

Konverziu stránky HTML na formát `pdf` je v technológii Electron možné implementovať prostredníctvom inštančnej metódy `printToPDFT()` [1]. NW.js a Haxe neposkytujú žiadne knižnice pre túto konverziu.

Z hľadiska výkonu som implementované aplikácie testovala porovnaním využitia pamäte RAM pri rôznom počte zobrazovaných dokumentov. Pre názornú ukážku som aplikáciu, implementovanú prostredníctvom technológie Haxe, zobrazovala vo WebKite najnovšej verzie, ktorú táto technológia ponúka. Binárne súbory NW.js a Electron sú testované taktiež vo

[1] https://developer.mozilla.org/en-US/docs/Web/API/Navigator

svojej najvyššej aktuálnej verzii. Výsledné využitie pamäte RAM je zobrazené v tabuľkách 5.1 až 5.3 rozdelených podľa testovaného operačného systému. Z testovania je zrejmé, že využitie pamäte RAM implementovanými aplikáciami je podobné na všetkých operačných systémoch. Najmenej pamäte zaberá produkt Haxe, ale na úkor možnosti implementovať funkcie, ktoré `node-webkit-haxelib` nepodporuje. So zvyšujúcim sa počtom zobrazujúcich sa dokumentov využitie pamäte mierne rastie. V najhoršom prípade som namerala využitie pamäte 125 MB aplikáciou implementovanou v technológii NW.js na operačnom systéme macOS. V tomto prípade sa zobrazovalo až 1000 dokumentov určených pre tlač. V dnešnej dobe je bežné mať na desktopových počítačoch 4 GB pamäte RAM, z čoho usudzujem, že na desktopových počítačoch by toto zistenie nemalo byť prekážkou v implementácii podobných aplikácií.

	Spustenie aplikácie	10 dokumentov	100 dokumentov	1000 dokumentov
Electron v1.6.2	47 MB	52 MB	61 MB	104 MB
NW.js v0.21.4	57 MB	63 MB	64 MB	95 MB
Node-webkit-haxelib v1.0.7	40 MB	48 MB	52 MB	66 MB

Tabuľka 5.1: Využitie pamäte RAM implementovaných a spustených aplikácií v aktuálne najnovšej verzii danej technológie. Využitie pamäte bolo merané pri spustení aplikácie, teda pri prihlásení užívateľa a následne pri zobrazení dokumentov v počte 10, 100 a 1000. Testovaný operačný systém je v tomto prípade Windows 10 Pro.

	Spustenie aplikácie	10 dokumentov	100 dokumentov	1000 dokumentov
Electron v1.6.2	45 MB	50 MB	68 MB	80 MB
NW.js v0.21.4	52 MB	69 MB	82 MB	95 MB
Node-webkit-haxelib v1.0.7	43 MB	52 MB	64 MB	72 MB

Tabuľka 5.2: Využitie pamäte RAM implementovaných aplikácií. Využité sú vždy aktuálne najnovšie verzie technológií Electron, NW.js a Haxe. Využitie pamäte bolo merané pri spustení aplikácie, teda pri prihlásení užívateľa a následne pri zobrazení dokumentov v počte 10, 100 a 1000. Tabuľka zobrazuje využitie pamäte na operačnom systéme Ubuntu 16.04.

	Spustenie aplikácie	10 dokumentov	100 dokumentov	1000 dokumentov
Electron v1.6.2	52 MB	55 MB	110 MB	120 MB
NW.js v0.21.4	50 MB	68 MB	120 MB	125 MB
Node-webkit-haxelib v1.0.7	45 MB	50 MB	80 MB	90 MB

Tabuľka 5.3: Tabuľka ukazuje využitie pamäte RAM implementovaných aplikácií v aktuálne najnovšej verzii WebKitu vybraných technológií na operačnom systéme macOS 10.12.1 (Darwin 16.1.0). Namerané výsledky zodpovedajú aplikácii po spustení, teda pri prihlásení užívateľa a následne pri zobrazení dokumentov v počte 10, 100 a 1000.

5.2 Porovnanie technológií z hľadiska bezpečnosti

V prípade Haxe používam pre zobrazovanie grafického užívateľského prostredia WebKit prostredníctvom technológie NW.js. Preto sa v tejto oblasti budem venovať iba porovnaniu technológií NW.js a Electron. Prínosom týchto technológií v oblasti zabezpečenia je, že obe technológie sa snažia čo najrýchlejšie prejsť na novú verziu WebKitu, ktorý používajú. Spolu s novou verziou môže priniesť WebKit nie len podporu nových funkcií, ale aj opravu chýb na úrovni zabezpečenia.

Aplikácia vytvorená prostredníctvom technológií založených na WebKite prináša so sebou prístup k súborovému systému, umožňuje pristúpiť k shell-u a mnoho ďalších funkcií. Preto so sebou prináša väčšie riziká oproti webovej aplikácii. Vývojári technológie Electron preto odporúčajú nahlásiť akúkoľvek bezpečnostnú chybu. Jednou z najbezpečnejších ciest, ktorú sledujú aj populárne aplikácie založené na tejto technológii ako Atom, Slack, atď, je zobrazovať primárne lokálny obsah alebo bezpečný vzdialený prístup bez integrácie Node.js. Ďalšie informácie spolu so zoznamom bezpečnostných postupov, ktorý zahŕňa zakázanie integrácie Node.js, sú dostupné na oficiálnych stránkach technológie Electron [1] v sekcii dokumentácie. Podobným spôsobom je možné postupovať aj pri vývoji aplikácie prostredníctvom NW.js.

Ďalšou oblasťou na úrovni zabezpečenia je ochrana zdrojových súborov. Technológia NW.js podporuje používanie tzv. *V8 Snapshot*. Čo znamená, že zdrojové kódy v jazyku JavaScript je možné ochrániť prostredníctvom distribuovania skompilovaním do natívneho kódu (*snapshot*), teda vytvorenia binárnych súborov. K vytvoreniu binárnych súborov slúži nástroj *nwjc* (viac popisovaný v podkapitole 3.3). Nevýhodou týchto súborov je, že je nutné ich vytvoriť pre každý operačný systém zvlášť a pri použití vo verzii NW.js 0.22 a nižšej, beží skompilovaný kód o cca 30 percent pomalšie [4]. Pre zabezpečenie zdrojových kódov ponúka technológia Electron ich zabalenie do balíkov `asar`. `Asar` pracuje podobne ako známy `tar`. Electron môže pristupovať k súborom bez nutnosti rozbalenia archívu. Je to však veľmi slabá ochrana vzhľadom na to, že z tohto archívu je celkom ľahké dostať sa k zdrojovým súborom. Vývojári technológie Electron plánujú podporu *V8 Snapshot*.

Poslednou oblasťou, ktorej sa budem venovať, je zabezpečenie dát pri komunikácii medzi aplikáciou, bežiacou na počítači užívateľa a serverom, na ktorý bude aplikácia posielať dotazy. Ide o zabezpečenie dát, ktoré sú zasielané sieťou, nie o dáta, ktoré sú uložené na počítači alebo na servery. Zabezpečenie zasielaných dát sa dosahuje prostredníctvom protokolu TLS (*Transport Layer Security*), ktorý nahrádza a doplňuje protokol SSL (*Secure Sockets Layer*). Tento spôsob zabezpečenia funguje na princípe šifrovania dát určitým šifrovacím algoritmom tak, aby prípadný útočník nemohol dáta prečítať. Protokol TLS pracuje nad TCP[2] komunikáciou. Po vytvorení TCP komunikácie prebehne komunikácia nutná k správnemu behu TLS[3]. Zabezpečí to komunikáciu s pravým serverom, nie s útočníkom a zároveň znemožnenie prečítania dát útočníkom (viac k TLS a certifikátom nájdete na stránke Chrómia v sekcii vzdelávanie[4]). Keďže Electron a NW.js používajú jadro Chrómia, podporujú zároveň použitie protokolu TLS pri `https` komunikácii. Electron naviac ponúka implementované aplikačné rozhranie `net`, ktoré implementuje dotazy `http/https` použitím knižnice Chromia.

[2] https://www.iplocation.net/tcp-ip

[3] Klient (aplikácia na strane užívateľa) a server si medzi sebou dohodnú verziu protokolu TLS, druh používaného šifrovania, atď. Server vyberie šifru z najvyššej podporovanej verzie protokolu na strane užívateľa. Po tomto základnom nastavení zašle server klientovi certifikát, ktorý si klient musí overiť.

[4] https://www.chromium.org

5.3 Testovanie implementovaných aplikácií

Prvá fáza testov prebehla pre každú technológiu zvlášť, a to prostredníctvom HTML5 testov[5]. Testovala som postupne 7 verzií technológií Electron a NW.js na operačných systémoch Ubuntu 16.4, macOS 10.12.1 a Windows 10 Pro so zostavou operačného systému 15063. Medzi testované verzie som zahrnula vždy aktuálne najnovšiu, najstaršiu verziu a 4 verzie podľa uváženia. Tento test spočíva v zobrazení webovej stránky v aplikácii a vyhodnotenia technologických možností jednotlivých kategórií. Pre praktickú ukážku sa v tabuľkách 5.7 až 5.11 nachádzajú vybrané prvky z HTML5 testov, testované na najnovšej verzii technológie Electron. V oblasti grafiky je v ukážke znázornená podpora pre JavaScriptové aplikačné rozhranie WebGL, ktoré umožňuje pracovať s 2D a 3D grafikou. Taktiež podporu pre virtuálnu realitu (WebVR), animácie a transformácie prostredníctvom kaskádových štýlov a použitie prvku `Canvas` pre vykresľovanie grafických elementov s využitím jazyka JavaScript. V oblasti úložiska je v ukážkach znázornená podpora pre webové úložisko (`Local Storage, Session Storage`), ktoré umožňuje mapovať hodnoty na základe kľúčových hodnôt. Taktiež využitie indexovania podobne ako v databázach (`IndexedDB`) čo prináša rýchlejšie vyhľadávanie. Z oblasti komunikácie sú v ukážkach vybrané prvky `XMLHttpRequest`, `WebSocket` a `Beacon`. Ďalej sú v tabuľkách znázornené ukážky prvkov v oblasti lokácie a orientácie. V tabuľke číslo 5.10 sa nachádzajú ukážky z možností použitia vstupov a výstupov ako notifikácie, možnosť pristúpiť ku kamere, používanie klávesových skratiek a ďalšie. Všetky ostatné testované prvky a ich popis je možné nájsť na stránke spomínaných HTML5 testov.

Pre skúmané technológie je táto forma testov ideálna vzhľadom na to, že každá používa pre zobrazovanie desktopovej aplikácie nejakú formu WebKitu skombinovanú s Node.js pre prístup k operačnému systému. Maximálne bodové ohodnotenie za všetky aspekty je aktuálne 555 bodov. V tabuľkách 5.4 až 5.6 sú jednotlivé bodové výsledky HTML5 testu v porovnaní s webovým prehliadačom Chrome od spoločnosti Google.

Verzia NW.js 0.8.7 vyžaduje zdieľanú knižnicu `libudev.so.0`, ktorá bola odstránená od verzie Ubuntu 14.4[6]. Táto závislosť bola odstránená vo verzii NW.js 0.12, z tohto dôvodu som okrem najstaršej verzie zvolila verzie pre technológiu NW.js vyššie ako 0.12.

Trochu zložitejšie je testovanie v prípade technológie Haxe. Vývojári pre túto technológiu vytvorili produkt vyvinutý v technológii NW.js *node-webkit-haxelib*, ktorý slúži pre vykreslenie grafickej časti aplikácie užívateľovi. Problém je, že najnovšiu verziu vydali 18. 6. 2014. V tom čase bol tento spôsob tvorby desktopových aplikácií v začiatkoch. Z tohto dôvodu som testovala len najnovšiu verziu WebKitu od vývojárov Haxe. V testoch dosiahla najnovšia verzia na operačnom systéme Windows 10 Pro len 401 z 555 bodov. Nepodporuje napríklad WebGL 2 alebo CSS selektory. V testoch síce zahrnutý je, ale pre reálnu aplikáciu je ideálnejšie zobrať vygenerované JavaScriptové súbory doplnené o súbory HTML a pre ich vykreslenie a samotnú distribúciu aplikácie použiť technológiu NW.js. Electron v tomto prípade nie je možné použiť a to z toho dôvodu, že vyžaduje spustenie prvého súboru napísaného jazyku JavaScript a nie v jazyku HTML.

[5] https://html5test.com/

[6] http://askubuntu.com/questions/288821/how-do-i-resolve-a-cannot-open-shared-object-file-libudev-so-0-error

Aplikácia - stabilná verzia	Dátum vydania	Windows 10 Pro	Ubuntu 16.04	macOS 10.12.1
Chrome version 57.0.2987	9. 3. 2017	519	519	519
Electron v1.6.2	1. 3. 2017	516	514	516
Electron v1.4.15	19. 1. 2017	496	494	496
Electron v1.3.14	14. 3. 2017	489	487	489
Electron v1.2.8	21. 6. 2016	489	487	489
Electron v1.1.3	25. 5. 2016	486	484	486
Electron v1.0.2	13. 5. 2016	486	484	486
Electron v0.30.7 (oldest)	16. 9. 2015	479	477	479

Tabuľka 5.4: Výsledky HTML5 testov technológie Electron v porovnaní s webovým prehliadačom Chrome.

Aplikácia - stabilná verzia	Dátum vydania	Windows 10 Pro	Ubuntu 16.04	macOS 10.12.1
Chrome version 57.0.2987	9. 3. 2017	519	519	519
NW.js v0.21.4	17. 3. 2017	516	514	516
NW.js v0.20.3	23. 2. 2017	516	514	516
NW.js v0.19.5	10. 1. 2017	504	502	504
NW.js v0.18.8	23. 11. 2016	501	499	501
NW.js v0.13.4	8. 4. 2016	486	484	486
NW.js v0.12.3	19. 8. 2015	460	460	460
NW.js v0.8.7 (oldest)	29. 10. 2014	397	—-	397

Tabuľka 5.5: Výsledky HTML5 testov technológie NW.js v porovnaní s webovým prehliadačom Chrome.

Aplikácia - stabilná verzia	Dátum vydania	Windows 10 Pro	Ubuntu 16.04	macOS 10.12.1
Chrome version 57.0.2987	9. 3. 2017	519	519	519
Node-webkit-haxelib v1.0.7	18. 6. 2014	401	401	401

Tabuľka 5.6: Výsledky HTML5 testov produktu *node-webkit-haxelib* v porovnaní s webovým prehliadačom Chrome.

	Windows 10 Pro	Ubuntu 16.04	macOS 10.12.1 (Darwin 16.1.0)	Chrome version 57
XMLHttpRequest	✓	✓	✓	✓
WebSocket	✓	✓	✓	✓
Beacon	✓	✓	✓	✓

Tabuľka 5.7: Vybrané technologické možnosti HTML5 testov v oblasti komunikácie. Testovaná je aplikácia napísaná prostredníctvom technológie Electron verzie 1.6.2 v porovnaní s funkčnosťou vo webovom prehliadači Chrome od spoločnosti Google.

	Windows 10 Pro	Ubuntu 16.04	macOS 10.12.1 (Darwin 16.1.0)	Chrome version 57
WebGL	✓	✓	✓	✓
WebGL 2	✓	✓	✓	✓
WebVR	✗	✗	✗	✗
CSS animation	✓	✓	✓	✓
CSS 3D Transforms	✓	✓	✓	✓
Web Animations API	✓	✓	✓	✓
Canvas 2D graphics	✓	✓	✓	✓

Tabuľka 5.8: Vybrané technologické možnosti HTML5 testov z oblasti grafiky. Testovaná je aplikácia napísaná prostredníctvom technológie Electron verzie 1.6.2 v porovnaní s funkčnosťou vo webovom prehliadači Chrome od spoločnosti Google.

	Windows 10 Pro	Ubuntu 16.04	macOS 10.12.1 (Darwin 16.1.0)	Chrome version 57
Session Storage	✓	✓	✓	✓
Local Storage	✓	✓	✓	✓
IndexedDB	✓	✓	✓	✓
Cookies	✓	✓	✓	✓

Tabuľka 5.9: Vybrané technologické možnosti HTML5 testov v oblasti úložiska. Testovaná je aplikácia napísaná prostredníctvom technológie Electron verzie 1.6.2 v porovnaní s funkčnosťou vo webovom prehliadači Chrome od spoločnosti Google.

	Windows 10 Pro	Ubuntu 16.04	macOS 10.12.1 (Darwin 16.1.0)	Chrome version 57
Notifications	✓	✓	✓	✓
Camera	✓	✓	✓	✓
Desktop capture	✓	✓	✓	✓
Local keyboard shortcuts	✓	✓	✓	✓
Global keyboard shortcuts	✓	✓	✓	✓
Gamepad control	✓	✓	✓	✓
Pointer event (mouse, pen, touch)	✓	✓	✓	✓

Tabuľka 5.10: Vybrané technologické možnosti HTML5 testov v oblasti užívateľského vstupu a výstupu. Testovaná je aplikácia napísaná prostredníctvom technológie Electron verzie 1.6.2 v porovnaní s funkčnosťou vo webovom prehliadači Chrome od spoločnosti Google.

	Windows 10 Pro	Ubuntu 16.04	macOS 10.12.1 (Darwin 16.1.0)	Chrome version 57
Geolocation	✓	✓	✓	✓
Device Orientation	✓	✓	✓	✓
Device Motion	✓	✓	✓	✓

Tabuľka 5.11: Vybrané technologické možnosti HTML5 testov v oblasti lokácie a orientácie. Testovaná je aplikácia napísaná prostredníctvom technológie Electron verzie 1.6.2 v porovnaní s funkčnosťou vo webovom prehliadači Chrome od spoločnosti Google.

Druhá fáza testovania spočívala v testovaní užívateľského prostredia navrhnutých aplikácií. Testovací protokol obsahuje sadu šiestich úloh, ktoré mal užívateľ vykonať a nakoniec ohodnotiť intuitívnosť aplikácie na stupnici od 1 po 10, kde 1 znamená neintuitívne a 10 intuitívne. Aplikáciu budú používať užívatelia rôzneho veku a vzdelania. Cieľová skupina teda tvorila široký záber. Vybraných užívateľov pre testovanie bolo 11 vo veku od 15 do 57 rokov. Pred samotným testovaním si každý užívateľ vybral preferovaný operačný systém. Zhodou okolností si všetci testovaný užívatelia zvolili systém Windows. Aplikácia je vďaka WebKitu rovnaká na operačných systémoch Windows, Linux a macOS. Užívatelia testovali aplikáciu vždy na jednom operačnom systéme a to aj z toho dôvodu, že pri testovaní na iných operačných systémoch by už boli oboznámení s prostredím aplikácie a výsledky by mohli byť zavádzajúce. Behom testovania som vybraných užívateľov pozorovala a hodnotila som užívateľské prostredia z hľadiska intuitívnosti a ľahkej orientácii v aplikácii.

Intuitívnosť jednotlivých častí aplikácie je znázornená v grafoch na obrázkoch 5.1 a 5.2. Testovací protokol sa nachádza v prílohe A a hodnotenie intuitívnosti aplikácie ako celok od užívateľov sa nachádza na obrázku 5.3. Pozorovaním užívateľov som zaznamenala náročnosť jednotlivých úloh znázornených v grafoch na obrázkoch Prílohy B. Všetky grafy majú prstencový tvar a sú na nich informácie o počte užívateľoch, ktorým zodpovedá dané bodové ohodnotenie zistené z testovania. Keďže funkčnosť a grafické prevedenie je v aplikáciách rovnaké, testovala som na aplikácii vytvorenej prostredníctvom technológie Electron.

Z testovania som zistila zaujímavú informáciu. Ak aplikáciu používali ľudia, ktorí už majú skúsenosť s podobným produktom, v tom prípade mali väčšie problémy s orientáciou v aplikácii. Na základe spätnej väzby som zistila, že je to spôsobené najmä malou odlišnosťou od produktu, ktorý používali. Títo užívatelia napríklad automaticky hľadali tlačidlá pre zobrazenie zoznamu dokumentov v určitom priečinku v hornej časti aplikácie namiesto ich aktuálneho umiestnenia, v pravo od zobrazovacej plochy. Naopak užívatelia, ktorí sa s podobnou aplikáciou nestretli, zvládli úlohy testovacieho protokolu bez problémov.

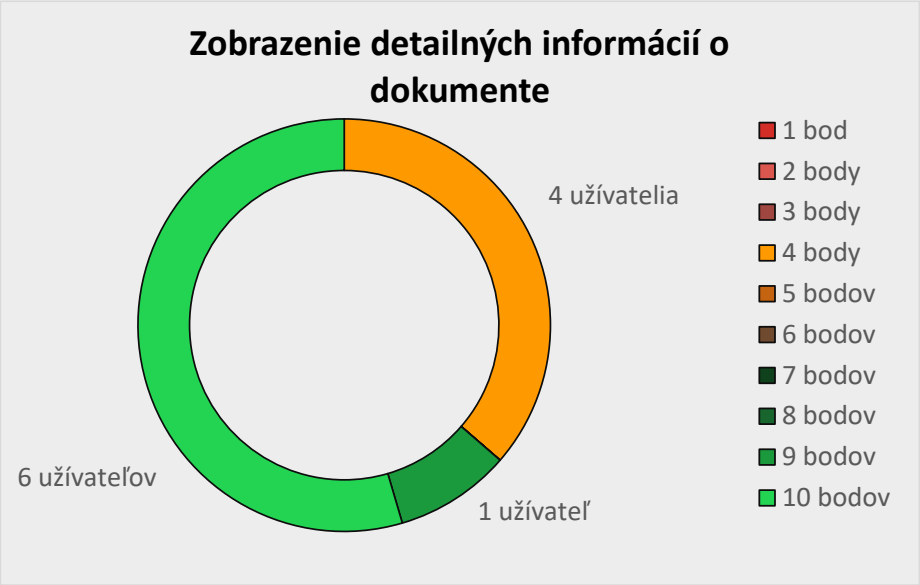
Ďalší prínos testovania bol v odstránení nedostatkov aplikácie, ktoré som priamym pozorovaním testovaných užívateľov zistila. Všetkým tlačidlám v zobrazovacej ploche som implementovala odlišný vzhľad v stave, keď na tlačidlo môžu užívatelia kliknúť a vykonať určitú funkciu a v stave, keď na neho kliknúť nemôžu. Štyria testovaní užívatelia pri úlohe so zobrazením detailných informácií o dokumente klikali namiesto ikonky na text `info`. Aktuálne je možné klikať na obe možnosti.

Na základe výsledkov z testovania je pre ďalší vývoj potreba zvážiť premiestnenie tlačidiel tak, ako sú na to zvyknutí užívatelia podobnej aplikácie. Tiež z testovania plynie, že by bolo vhodné implementovať grafickú funkciu k dokumentu, ktorá bude znázorňovať či dokument patrí do zoznamu obľúbených alebo nie a taktiež povolenie výberu viacerých dokumentov.

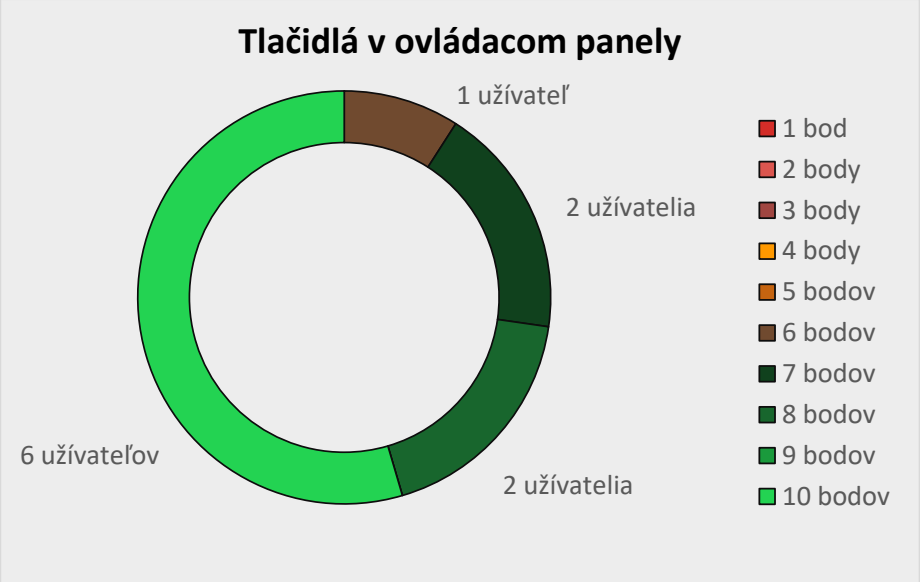
(a) Časť aplikácie, s ktorou sa užívateľ stretne ako s prvou, prihlásenie sa do aplikácie.

(b) Orientácia užívateľa v zobrazovaní zoznamu dokumentov v jednotlivých zložkách *Waiting*, *Printed* a *Favorite*.

Obr. 5.1: Na grafoch (a) až (b) je znázornená intuitívnosť jednotlivých častí aplikácie. Na oboch grafoch je možné vidieť bodové ohodnotenie užívateľov. Ak sa užívateľ orientoval v danej časti aplikácie veľmi intuitívne a nepotreboval žiadne inštrukcie, bol ohodnotený plným počtom bodov (10), na grafoch je táto hodnota znázornená svetlo zelenou farbou. Naopak, ak užívateľ potreboval inštrukcie a nevedel si pomôcť sám, bol ohodnotený 1 bodom, ktorý je na grafe znázornený ako svetlo červená farba.



(a) Intuitívnosť zobrazenia detailných informácií o vybranom dokumente.



(b) Intuitívnosť používania tlačidiel pre postupný výber dokumentov, tlač, odstránenie a pridanie alebo odstránenie dokumentu z obľúbených.

Obr. 5.2: Na grafoch (a) až (b) je znázornená intuitívnosť jednotlivých častí aplikácie podobne ako na predchádzajúcom obrázku. Na oboch grafoch je možné vidieť bodové ohodnotenie užívateľov. V grafe (a) je zobrazená funkcia aplikácie, a to zobrazenie detailných informácií o dokumente a v grafe (b) sú znázornené navigačné tlačidlá. Ak sa užívateľ orientoval v danej časti aplikácie veľmi intuitívne a nepotreboval žiadne inštrukcie, bol ohodnotený plným počtom bodov (10), na grafoch je táto hodnota znázornená svetlo zelenou farbou. Naopak, ak užívateľ potreboval inštrukcie a nevedel si pomôcť sám, bol ohodnotený 1 bodom, ktorý je na grafe znázornený ako svetlo červená farba.

Obr. 5.3: Graf zobrazujúci celkové hodnotenie intuitívnosti užívateľmi. Na grafe je znázornený počet užívateľov v závislosti na počte bodov, ktorý užívatelia zadali na konci testovania. Svetlo zelená farba znázorňuje najvyšší počet bodov, teda veľmi intuitívne prostredie aplikácie. Svetlo červená farba naopak, najnižší počet bodov a teda neintuitívne prostredie aplikácie.

Kapitola 6

Záver

Táto práca zahŕňa prieskum technológií Haxe, NW.js a Electron. Bola objasnená história, vývoj, distribúcia a existujúce aplikácie v týchto technológiách. Následne v nich bola implementovaná aplikácia, ktorá komunikuje so serverom firmy Y Soft Corporation prostredníctvom rozhrania REST. Implementované aplikácie majú vďaka WebKitu rovnaký vzhľad, ako medzi jednotlivými operačnými systémami, tak aj medzi implementovanými aplikáciami. Testovanie grafického užívateľského prostredia bolo teda uskutočnené len na jednej aplikácii, konkrétne aplikácii vyvinutej prostredníctvom technológie Electron. Z testovania som zistila niekoľko chýbajúcich funkcionalít, ktoré boli do aplikácií po testovaní implementované. Celkový názor na aplikáciu bol prívetivý, užívatelia nemali problém sa v aplikácii zorientovať, ako môžme vidieť z výsledkov celkovej intuitívnosti na obrázku 5.3.

Z prieskumu, porovnania a testovania aplikácií vyplýva, že technológia Haxe má síce výhodu v silnej typovej kontrole, avšak pre implementáciu desktopovej aplikácie založenej na WebKite to nie je dobrá voľba. Tento záver som urobila najmä preto, že po implementácii aplikácie musíme zobraziť aplikáciu prostredníctvom technológie NW.js, pretože WebKit od vývojárov technológie Haxe je značne zastaralý. Z toho vyplýva, že sme nie len ukrátení o možnosť používať technológiu Electron, ale sme ukrátení aj o možnosť využívať aplikačné rozhrania technológie NW.js. Electron a NW.js dosiahli vo výsledkoch testov HTML5 rovnaké bodové ohodnotenie. Obe technológie sú veľmi zaujímavé a obe majú populárne produkty vyvinuté úspešnými firmami ako Microsoft alebo Intel. Electron však prináša viac implementačných možností pre vývojárov. Ponúka väčší výber aplikačných rozhraní ako NW.js, kde je možné využívať rovnakú funkcionalitu, ale programátor si ju musí implementovať sám. Rovnako v distribuovaní aplikácie koncovým užívateľom vedie Electron. Táto technológia ponúka nástroje, vďaka ktorým vieme vytvoriť inštalačné balíčky. Naproti tomu technológia NW.js podobné balíčky neposkytuje a preto vývojári NW.js na oficiálnych stránkach odporúčajú postupovať podľa oficiálnych pokynov pre vytvorenie týchto balíčkov.

V ďalšom vývoji aplikácie vyvinutej prostredníctvom technológie Electron by som chcela zahrnúť implementáciu funkcií pre platobný systém a tiež dodať možnosť upravovania dokumentov pred samotnou tlačou. Ďalším krokom bude taktiež doplniť aplikáciu o možnosť automatických aktualizácií.

Na záver by som len chcela poznamenať, že technológie Electron a NW.js sú veľmi rýchlo rozvíjajúce sa technológie. V priebehu písania tejto práce vyšlo niekoľko nových verzií. Preto sú v testoch uvedené najnovšie verzie, ktoré boli aktuálne vydané.

Literatúra

[1] *Electron.atom.io: Build cross platform desktop apps with JavaScript, HTML, and CSS.* [Online; navštívená 6.1.2017].
URL http://electron.atom.io/

[2] *Haxe.org: The Cross-platform Toolkit.* [Online; navštívená 6.1.2017].
URL http://haxe.org/

[3] *Npmjs.com: electron-builder.* [Online; navštívená 30.1.2017].
URL https://www.npmjs.com/package/electron-builder

[4] *NWjs.io: NW.js.* [Online; navštívená 26.1.2017].
URL https://nwjs.io/

[5] Benoit, A.: *NW.js Essentials.* Community experience distilled, Packt Publishing, 2015, ISBN 9781785287008.

[6] Chaffey, D.: *Insights from KPCB US and global internet trends 2015 report.* 2015, [Online; navštívená 25.1.2017].
URL http://www.smartinsights.com/internet-marketing-statistics/insights-from-kpcb-us-and-global-internet-trends-2015-report/

[7] Fielding, R. T.: *Architectural Styles and the Design of Network-based Software Architectures.* Dizertační práce, 2000.

[8] Galib, A. S.: *Is JavaScript the most popular programming language in the world?* May 2016, [Online; navštívená 11.3.2017].
URL https://www.quora.com/Is-JavaScript-the-most-popular-programming-language-in-the-world

[9] Goldenberg, A.: *Pros and Cons of Developing Native vs. Cross-Platform Web-Based Mobile Application.* 2013, [Online; navštívená 25.1.2017].
URL https://www.dbbest.com/blog/pros-and-cons-of-developing-native-vs-cross-platform-web-based-mobile-application/

[10] Jensen, P.: *Cross-platform Desktop Applications: With Node, Electron, and Nw.js.* Manning Publications Company, 2017, ISBN 9781617292842.

[11] Johnson, G.: *Training Guide Programming in HTML5 with JavaScript and CSS3 (MCSD): 70-480.* Microsoft Press Training Guide, Pearson Education, 2013, ISBN 9780735674349.

[12] Kinney, S.: *Electron in Action.* Manning Publications Company, 2017, ISBN 9781617294143.

Prílohy

Príloha A

Testovací protokol

Úvodné informácie

Dobrý deň, vopred ďakujem za váš čas. Úlohy vám zaberú zhruba 10 minút. Ak si nebudete vedieť rady opýtajte sa na ďalší krok poverenej osoby, ktorá bude dohliadať nad testovaním. Chceme týmto zistiť ako intuitívna a prehľadná je aplikácia YSoftSafeQ. Prosím vás, ak prídete na akékoľvek funkcie, ktoré sú pre vás nezrozumiteľné, upozornite na to poverenú osobu. Zmyslom tohto testovania je testovať aplikáciu, nie vás. Na konci testovania zhodnoťte, či sa vám aplikácia zdala intuitívna a všetko bolo pre vás zrozumiteľné.

Úvod do prostredia aplikácie

Aplikácia slúži k bezpečnej a riadenej tlači dokumentov na základe mena a hesla. Táto aplikácia je určená firmám ako riešenie centrálnej tlače (napríklad knižnica, kde ľudia tlačia rôzne dokumenty na základe svojich údajov). V reálnej situácii by ste teda prihlasovacie údaje dostali od vedenia firmy, zamestnancov knižnice a podobne. Na začiatku testovania bude odkaz pre spustenie aplikácie pripravený na pracovnej ploche počítača.

Pre testovacie účely využite tieto prihlasovacie údaje:

- Meno: Janko
- Heslo: Hraško

Úlohy

1. Spustite aplikáciu *YSoft SafeQ* na počítači s preferovaným operačným systémom, ktorá sa nachádza na pracovnej ploche počítača. Prihláste sa do systému pomocou prihlasovacích údajov zo sekcie *Úvod do prostredia aplikácie*.
2. Pred sebou vidíte dokumenty určené k tlači. Vyberte si dokument a zobrazte o ňom detailné informácie.
3. Vyberte jeden z dokumentov a vymažte ho. Následne si vyberte ďalší dokument a vytlačte ho.
4. Zobrazte si zoznam vytlačených dokumentov.

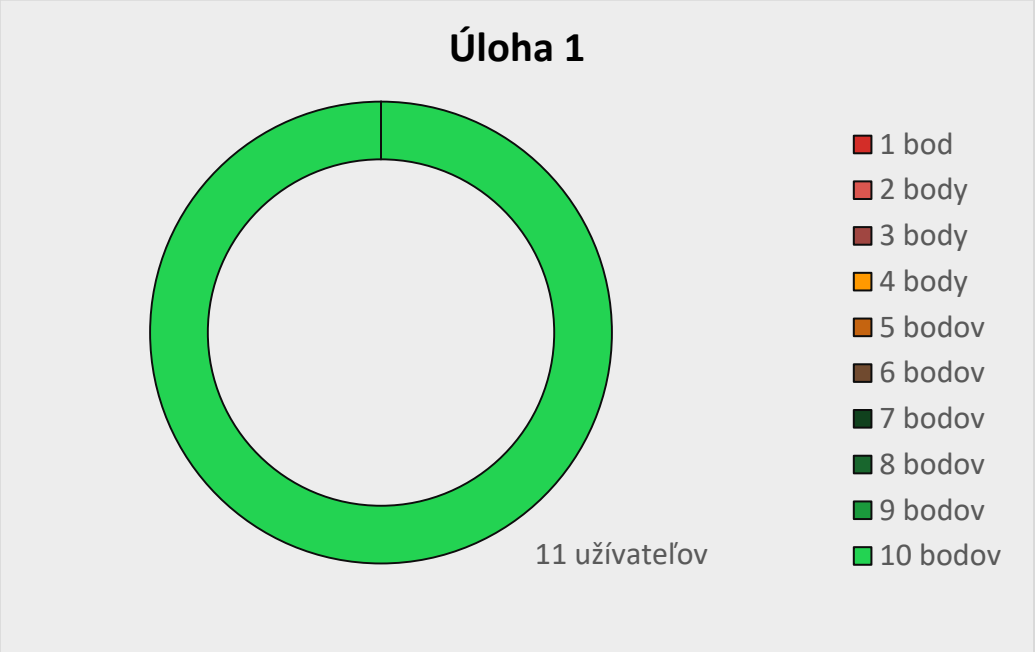
5. Jeden z vytlačených dokumentov pridajte do svojich obľúbených a následne si zobrazte zoznam obľúbených dokumentov.
6. Odstráňte všetky dokumenty zo zoznamu obľúbených (avšak nie v zmysle vymazania dokumentov).

Záver

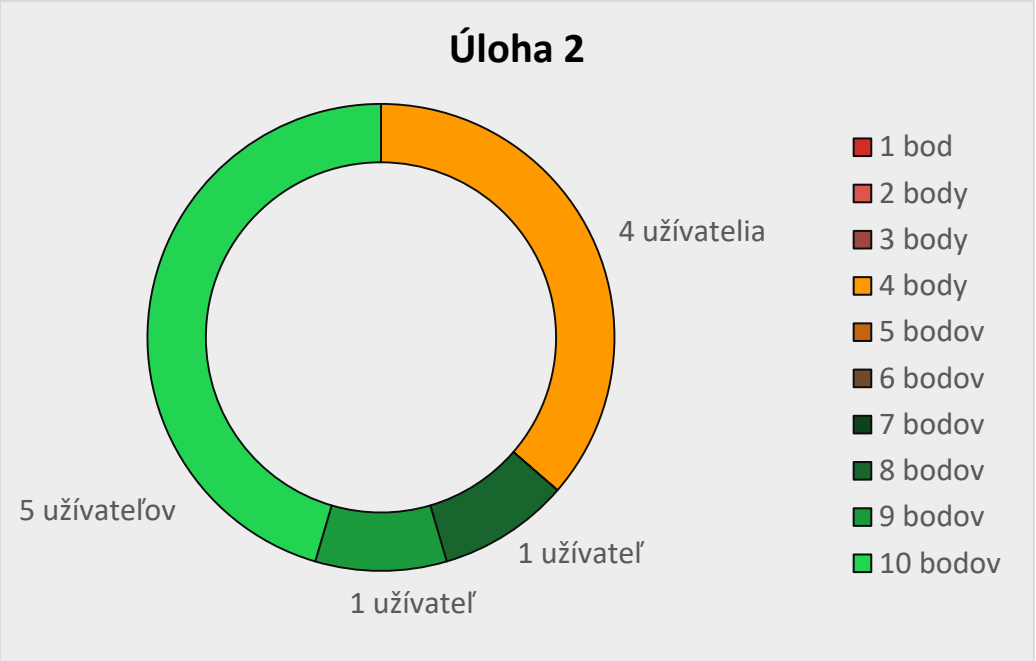
Ďakujem za poskytnutie vášho času. Posledná úloha spočíva v ohodnotení intuitívnosti aplikácie na základe vášho dojmu z testovania (1 - veľmi zložitá, 10 - veľmi intuitívna).

Príloha B

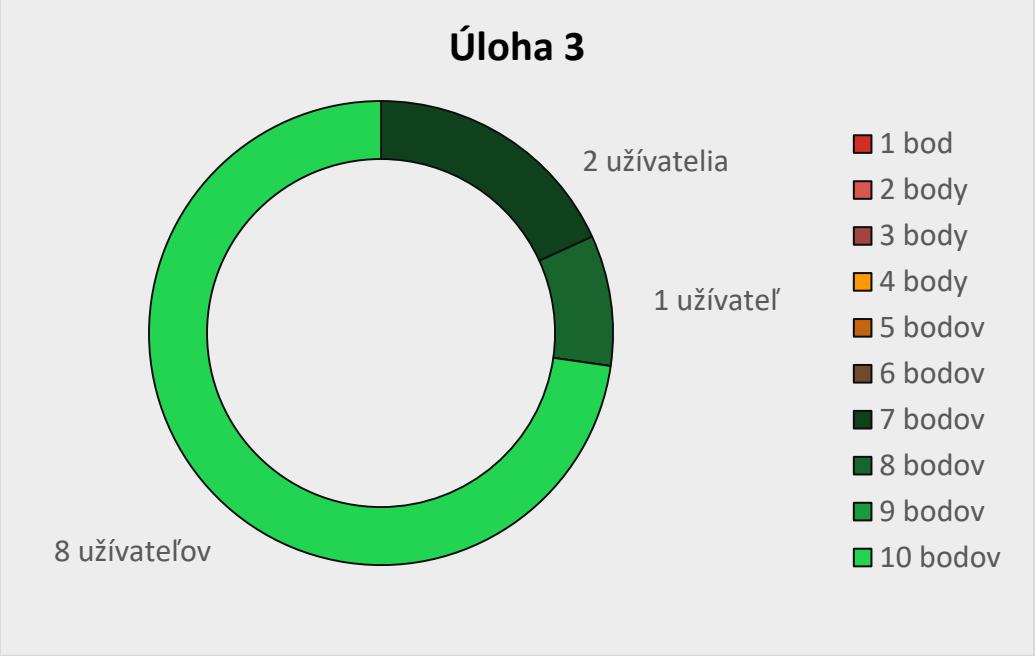
Grafy hodnotení úloh testovacieho protokolu



Obr. B.1: V grafe je znázornená intuitívnosť úlohy 1 testovacieho protokolu vo forme bodového ohodnotenia. Ak sa užívateľ orientoval v danej úlohe veľmi intuitívne a nepotreboval žiadne inštrukcie, bol ohodnotený plným počtom bodov (10), na grafoch je táto hodnota znázornená svetlo zelenou farbou. Naopak, ak užívateľ potreboval inštrukcie a nevedel si pomôcť sám, bol ohodnotený 1 bodom, ktorý je na grafe znázornený ako svetlo červená farba.

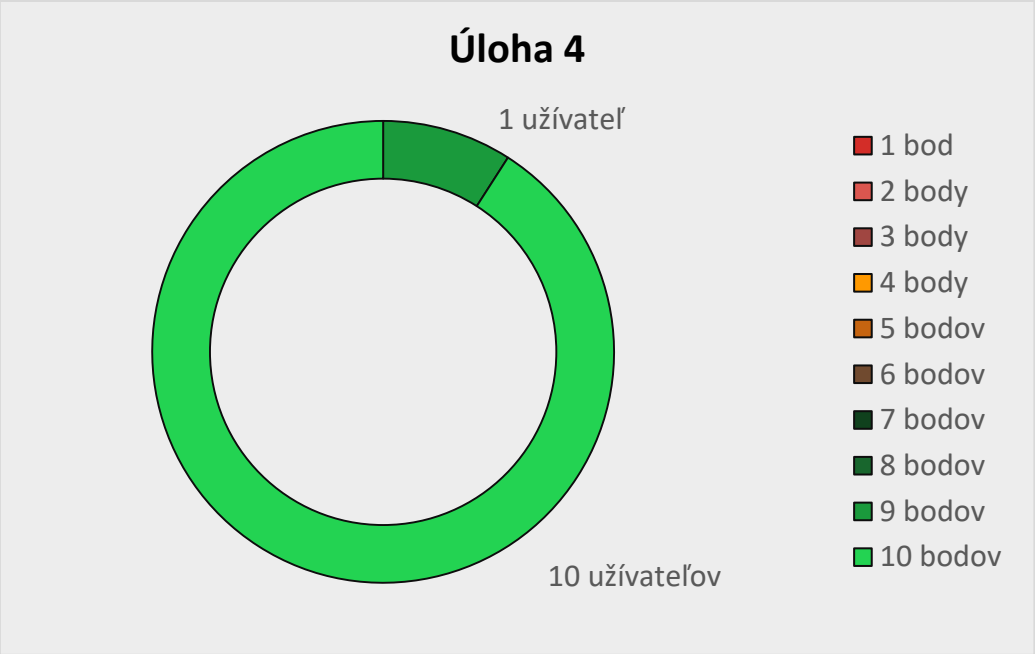


(a) Počet užívateľov v závislosti na bodovom ohodnotení druhej úlohy.

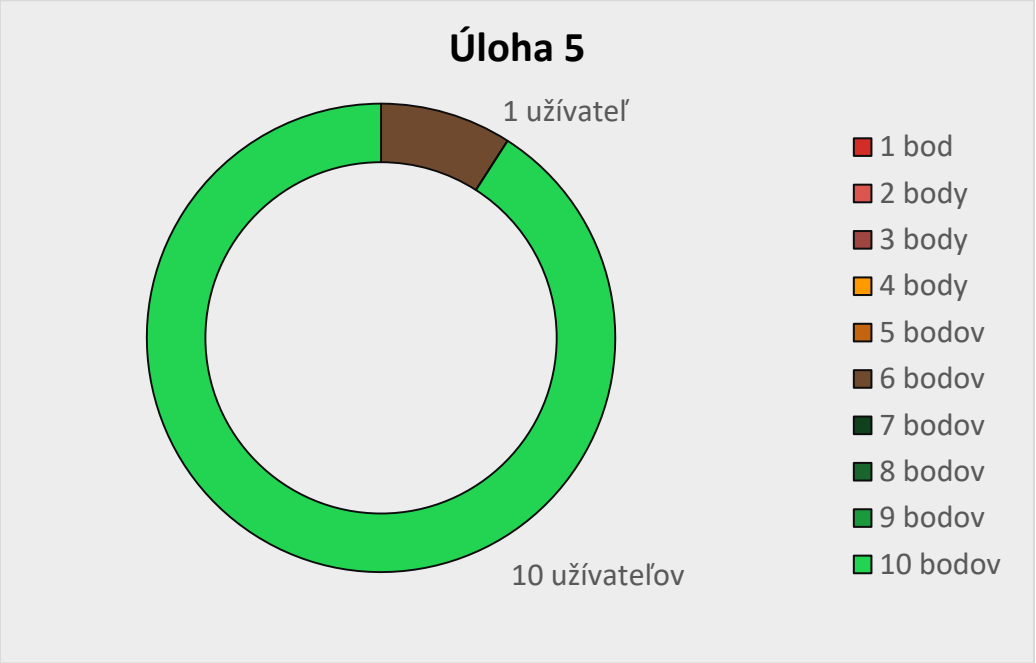


(b) Počet užívateľov v závislosti na bodovom ohodnotení tretej úlohy.

Obr. B.2: Na grafoch je znázornená intuitívnosť úloh 1 a 2 testovacieho protokolu. Na oboch grafoch je možné vidieť bodové ohodnotenie danej úlohy. Ak sa užívateľ orientoval v danej úlohe veľmi intuitívne a nepotreboval žiadne inštrukcie, bol ohodnotený plným počtom bodov (10), na grafoch je táto hodnota znázornená svetlo zelenou farbou. Naopak, ak užívateľ potreboval inštrukcie a nevedel si pomôcť sám, bol ohodnotený 1 bodom, ktorý je na grafe znázornený ako svetlo červená farba.

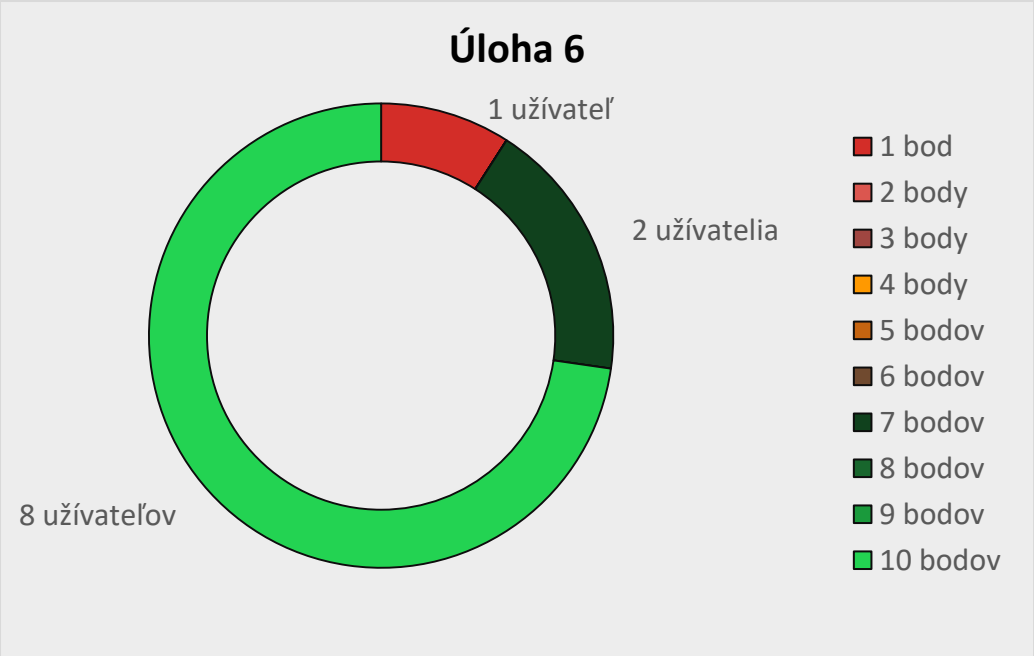


(a) Počet užívateľov v závislosti na bodovom ohodnotení štvrtej úlohy.



(b) Počet užívateľov v závislosti na bodovom ohodnotení piatej úlohy.

Obr. B.3: Na grafoch (a) a (b) je znázornená intuitívnosť úloh 1 a 2 testovacieho protokolu. Na oboch grafoch je možné vidieť bodové ohodnotenie danej úlohy. Ak sa užívateľ orientoval v danej úlohe veľmi intuitívne a nepotreboval žiadne inštrukcie, bol ohodnotený plným počtom bodov (10), na grafoch je táto hodnota znázornená svetlo zelenou farbou. Naopak, ak užívateľ potreboval inštrukcie a nevedel si pomôcť sám, bol ohodnotený 1 bodom, ktorý je na grafe znázornený ako svetlo červená farba.



Obr. B.4: V grafe je znázornená intuitívnosť úlohy 1 testovacieho protokolu vo forme bodového ohodnotenia. Ak sa užívateľ orientoval v danej úlohe veľmi intuitívne a nepotreboval žiadne inštrukcie, bol ohodnotený plným počtom bodov (10), na grafoch je táto hodnota znázornená svetlo zelenou farbou. Naopak, ak užívateľ potreboval inštrukcie a nevedel si pomôcť sám, bol ohodnotený 1 bodom, ktorý je na grafe znázornený ako svetlo červená farba.

Príloha C

Spustenie aplikácií

Electron

Podporované verzie operačných systémov sa nachádzajú v podkapitole 3.2. Pre spustenie aplikácie implementovanej prostredníctvom technológie Electron máme niekoľko možností:

1. Spustenie aplikácie bez nutnosti inštalácie. Na disku, ktorý je priložený k tejto práci sa nachádzajú pripravené aplikácie pre spustenie na vybranom operačnom systéme. Nachádzajú sa v adresári `Electron\spustiteľné_aplikácie\`.
   Upozornenie: Windows a macOS môžu odmietnuť spustiť aplikáciu, pretože nie je dôveryhodná. V tom prípade treba explicitne potvrdiť spustenie aplikácie.

2. Spustenie aplikácie pomocou správcu balíčkov pre Node.js `npm` na operačnom systéme Linux. Pomocou terminálu otvoríme zložku so zdrojovými súbormi na disku (umiestnenie: `Electron\zdrojové_súbory\`) a postupne spustíme dva príkazy:

   - `npm install`
   - `npm start`

3. Spustenie aplikácie až po jej nainštalovaní. V zložke s balíčkami pre inštaláciu (umiestnenie na disku: `Electron\inštalačné_balíčky\`) sa nachádzajú balíčky určené pre operačný systém Windows a Linux. Nachádza sa tu tiež inštalačný balíček vo formáte `.appx` určenom pre *Windows App Store*.

NW.js

Na priloženom disku sa nachádzajú pripravené aplikácie pre spustenie bez nutnosti inštalácie aplikácie (umiestnenie: `NWjs\spustiteľné_aplikácie\`). V priečinku so spustiteľnými aplikáciami si stačí vybrať operačný systém a spustiť binárny súbor s príponou v závislosti na vybranom operačnom systéme.

Upozornenie: Windows a macOS môžu odmietnuť spustiť aplikáciu, pretože nie je dôveryhodná. V tom prípade treba explicitne potvrdiť spustenie aplikácie. Systému CentOS Linux chýbajú knižnice potrebné pre spustenie aplikácie[1].

[1] https://github.com/drom/simple-nwjs-app/issues/4

Haxe

Podporované operačné systémy, na ktorých je možné aplikáciu spustiť sú rovnaké ako pri technológii NW.js. Aplikáciu, ktorej zdrojové kódy sú v jazyku Haxe, je možné spustiť dvoma spôsobmi. Prvým je využitie pripravených súborov pre spustenie aplikácie na vybranom operačnom systéme v zložke na disku `Haxe\spustiteľné_aplikácie\`. Druhým spôsobom je generácia potrebných JavaScriptových súborov a manuálne vytvorenie spustiteľnej aplikácie.

Pre druhý spôsob je nutné splniť tieto kroky:

1. Generácia JavaScriptových súborov zo zdrojových kódov napísaných v jazyku Haxe. Kroky pre generáciu súborov:
   (a) Inštalácia HaxeDevelop[2].
   (b) Inštalácia kompilátora Haxe. Tento krok je možné docieliť spustením HaxeDevelop, ktorý po prvom spustení vyzve užívateľa k inštalácii Haxe kompilátora a virtuálneho stroja Neko.
   (c) Spustenie súbora `Haxe\zdrojové_súbory\`HaxeApp.proj.
   (d) Potrebné nastavenie pre generáciu súborov. Po otvorení záložky `Project -> Properties` je nutné nasledujúce nastavenie:
   - V karte `Build` je pre možnosť písania viacerých príkazov potreba kliknúť na `Builder` a vpísať do okna dva príkazy, pričom je nutné časť príkazu `cesta\k\súboru` nahradiť správnou cestou k súboru `bin`. Potrebné príkazy:
     ```
     haxe -lib hxnodejs -cp src -js cesta\k\súboru\bin\login.js
     -main LogIn
     haxe -lib hxnodejs -cp src -js cesta\k\súboru\bin\jobInfo.js
     -main JobInfo
     ```
   - V karte `Compiler Options` v sekcii `General` je treba pridať do `Libraries` knižnicu `hxnodejs`.

   (e) Po stlačení klávesy `F8` sa vygenerujú potrebné súbory v jazyku JavaScript do zložky `bin`, kde sú pripravené súbory s grafickou časťou aplikácie.
2. Stiahnutie balíku NW.js s binárnym súborom z oficiálnej stránky[3] pre spustenie aplikácie. Pre prípadný vývoj je možnosť zvoliť balík typu SDK. V tomto prípade stačí balík typu *Normal*.
3. Po rozbalení balíka nakopírujeme do hlavnej zložky, kde sa nachádza aj spustiteľný binárny súbor, zložku `bin` z disku (umiestnenie: `Haxe\zdrojové_súbory\bin`) s celým jej obsahom. Následne nakopírujeme do hlavnej zložky súbor z disku `package.json` .
4. Spustenie binárneho súboru v stiahnutom balíčku.

Upozornenie: Windows a macOS môžu odmietnuť spustiť aplikáciu, pretože nie je dôveryhodná. V tom prípade treba explicitne potvrdiť spustenie aplikácie.

[2] http://haxedevelop.org/
[3] https://nwjs.io/

Príloha D

Obsah DVD

Na disku, ktorý je priložený k tejto práci, sa nachádzajú zdrojové súbory a spustiteľné súbory implementovaných aplikácií. Naviac sa na disku nachádzajú inštalačné balíčky aplikácie implementovanej prostredníctvom technológie Electron. Hierarchia súborov a adresárov na disku:

- `Electron\`
  - `inštalačné_balíčky\`
  - `spustiteľné_aplikácie\`
  - `zdrojové_súbory\`
- `Haxe\`
  - `spustiteľné_aplikácie\`
  - `zdrojové_súbory\`
- `NWjs\`
  - `spustiteľné_aplikácie\`
  - `zdrojové_súbory\`
- `tex\` obsahuje zdrojové súbory písomnej časti bakalárskej práce.
- `xscens00.pdf` obsahuje písomnú časť bakalárskej práce.
- `videá\` obsahuje video aplikácie na operačnom systéme Windows, Linux a macOS. Zachycuje grafické užívateľské rozhranie aplikácie implementovanej prostredníctvom technológie Electron.
- `readme.txt` obsahuje popis obsahu disku, stručný popis účelu vytvorenia disku a taktiež možnosti spustenia aplikácií.