Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Vývoj aplikace s využitím cross-platform frameworku

Erik Vojtěch

Diplomová práce

2018

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2017/2018**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Erik Vojtěch**
Osobní číslo:	**I16246**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Vývoj aplikace s využitím cross-platform frameworku**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

Cílem diplomové práce je vytvoření aplikace v cross-platform frameworku (např. Electron) a provedení srovnání existujících frameworků.
V teoretické části práce bude vypracována rešerše a porovnání existujících cross-platform frameworků (Haxe, Electron, NW.js, ...). Ve srovnání budou podrobně a přehledně uvedeny základní vlastnosti jednotlivých frameworků (programovací jazyk, použitá technologie a běhové prostředí, dostupné cílové platformy, prostředky dostupné pro uživatele a další). V práci bude poukázáno na odlišnosti a specifika využití těchto frameworků na rozdíl od klasických frameworků zaměřených na konkrétní technologii a platformu (např.: Java ? Swing, C# - WinForms, ...).
V praktické části bude ve vybraném frameworku realizována aplikace s grafickým uživatelským rozhraním. Aplikace bude disponovat perzistencí stavu a základní aplikační logikou (např. úkolovníček, nástroj pro kreslení diagramů, aj.).

Rozsah grafických prací: **10**

Rozsah pracovní zprávy: **60**

Forma zpracování diplomové práce: **tištěná**

Seznam odborné literatury:

JENSEN, Paul B. Cross-Platform Desktop Applications: With Node, Electron, and NW. Js. Manning Publications Company, 2017. ISBN 9781617292842.
ZAKAS, Nicholas C. Understanding ECMAScript 6: the definitive guide for JavaScript developers. San Francisco: No Starch Press, 2016. ISBN 9781593277987.
CROCKFORD, Douglas. JavaScript the Good Parts. Sebastopol: O'Reilly Media, 2008. ISBN 9780596554873.
Electron: Build cross platform desktop apps with JavaScript, HTML and CSS [online]. [cit. 2017-10-04]. Dostupné z: https://electron.atom.io/
NW.js [online]. c2015-2016 [cit. 2017-10-04]. Dostupné z: https://nwjs.io/
Haxe: The Cross-platform Toolkit [online]. Haxe Foundation, c2017 [cit. 2017-10-04]. Dostupné z: https://haxe.org/

Vedoucí diplomové práce: **Ing. Roman Diviš**
Katedra softwarových technologií

Datum zadání diplomové práce: **30. října 2017**

Termín odevzdání diplomové práce: **18. května 2018**

Ing. Zdeněk Němec, Ph.D.
děkan

L.S.

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 15. listopadu 2017

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 20. 8. 2018

Erik Vojtěch

Poděkování

Tímto bych rád poděkoval svému vedoucímu diplomové práce Ing. Romanu Divišovi za jeho cenné rady, odbornou pomoc a vedení při psaní této práce.

ANOTACE

Tato práce se zabývá vývojem aplikací pomocí cross-platformových frameworků. V úvodu práce jsou představeny klasické přístupy vývoje s využitím frameworků .NET a Cocoa. Další část uvádí multiplatformní přístupy. Mezi které se řadí frameworky Haxe, NW.js, Electron nebo jazyk Java. Tyto frameworky jsou následně porovnány. V poslední části práce je uskutečněna implementace jednoduché aplikace ve vybraném frameworku.

KLÍČOVÁ SLOVA

vývoj aplikace, multiplatformní, Electron, JavaScript, webové technologie, framework, JavaFX, .NET, Cocoa, Haxe, NW.js, Java

TITLE

Application development using cross-platform framework

ANNOTATION

This thesis deals with the development of applications using cross-platform frameworks. The beginning of the thesis introduces classical development approaches using .NET and Cocoa frameworks. The next section introduces multiplatform approaches. These include Haxe, NW.js, Electron or language Java. These frameworks are then compared. In the last part of the thesis is realized implementation of a simple application in selected framework.

KEYWORDS

application development, cross-platform, Electron, JavaScript, web technologies, framework, JavaFX, .NET, Cocoa, Haxe, NW.js, Java

Obsah

Seznam obrázků **9**

Seznam tabulek **10**

Seznam zkratek **11**

Úvod **12**

1 Klasické frameworky **13**
1.1 .NET framework . . . 13
1.2 Cocoa framework . . . 14

2 Cross-platform fameworky **16**
2.1 Haxe . . . 17
2.1.1 Jazyk, technologie a prostředí . . . 17
2.1.2 Architektura . . . 18
2.1.3 Cílové platformy . . . 19
2.2 NW.js . . . 20
2.2.1 Jazyk, technologie a prostředí . . . 21
2.2.2 Architektura . . . 22
2.2.3 Cílové platformy . . . 23
2.3 Electron . . . 23
2.3.1 Jazyk, technologie a prostředí . . . 24
2.3.2 Architektura . . . 26
2.3.3 Cílové platformy . . . 27
2.4 Java . . . 27
2.4.1 Jazyk, technologie a prostředí . . . 28
2.4.2 Architektura . . . 28
2.4.3 Cílové platformy . . . 30
2.5 Odlišnosti od klasických frameworků . . . 30
2.6 Shrnutí . . . 32

3 Vývoj vlastní aplikace **34**
3.1 Představení aplikace . . . 34
3.2 Požadavky . . . 34
3.2.1 Funkční požadavky . . . 34
3.2.2 Nefunkční požadavky . . . 35
3.3 Návrh . . . 35
3.3.1 Databázový model . . . 35
3.3.2 Struktura aplikace . . . 37
3.3.3 Grafické uživatelské rozhraní . . . 38

4 Implementace **41**
4.1 Vlastní ovládací prvky . . . 41
4.2 Mezi-procesní komunikace . . . 41
4.3 Otevření souborů pomocí výchozí aplikace systému . . . 42
4.4 Práce s DB . . . 43
4.5 Minimalizování aplikace . . . 44
4.6 Notifikace . . . 45
4.7 Hlavní proces aplikace po vzoru singleton . . . 45

4.8 Import a export 46
5 Aplikační manuál **48**
5.1 Plánování připomínky 49
5.2 Vložení souborů 49
5.3 Doplňkové funkce 50
Závěr **52**
Použitá literatura **54**

Seznam obrázků

1 Architektura frameworku .NET 13
2 Haxe logo 17
3 NW.js logo 20
4 NW.js schéma 23
5 Electron logo 24
6 Architektura zdrojového kódu Electron 26
7 Java logo 27
8 Architektura JavaFX 30
9 Databázový model 36
10 Přístup k databázi 36
11 Okno s ohraničením 39
12 Okno bez ohraničením 39
13 Výsledný vzhled aplikace 40
14 Import a export dat 46
15 Ovládání - základní funkce 48
16 Ovládání - mazání souboru 49
17 Ovládání - naplánování připomínky 49
18 Ovládání - drag & drop 50
19 Ovládání - vložení souboru 50
20 Ovládání - kontextové menu 51
21 Ovládání - export - dialog pro heslo 51

Seznam tabulek

1 Haxe - cílové platformy 20
2 Java - cílové platformy 30
3 Srovnání cross-platform frameworků 32
4 Požadavky - funkční 35
5 Požadavky - nefunkční 35

SEZNAM ZKRATEK

AES	Advanced Encryption Standard
API	Application Programming Interface
ARM	Advanced RISC Machine
AS	ActionScript
CIL	Common Intermediate Language
CLR	Common Language Runtime
CRUD	Create, Read, Update, Delete
CSS	Cascading Style Sheet
CTR	Counter Mode
FCL	Framework Class Library
GUI	Graphical User Interface
HTML	HyperText Markup Language
IDE	Integrated Development Environment
IPC	Inter-Process Communication
JDK	Java Development Kit
JRE	Java Runtime Environment
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
OS	Operační systém
PHP	Hypertext Preprocessor
RISC	Reduced Instruction Set Computer
SQL	Structured Query Language
URL	Uniform Resource Locators
XML	eXtensible Markup Language

ÚVOD

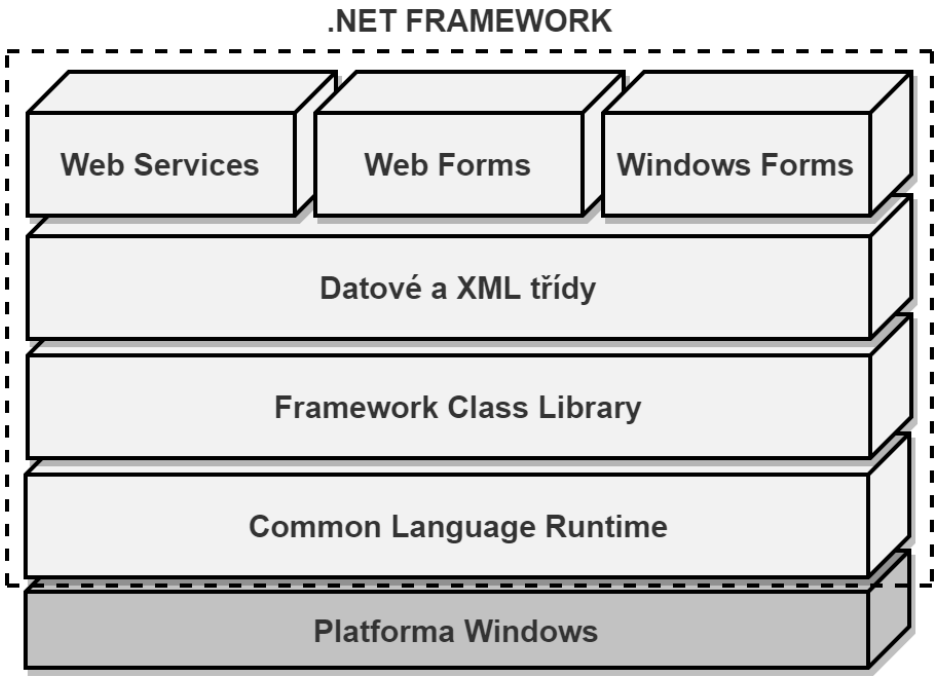
Vývoj softwaru byl v minulosti soustředěn především na aplikace běžící na desktopu. Poté se tento směr začal uchylovat k webovým aplikacím. A to hlavně díky rychlému rozvoji a rozšíření internetu a webových prohlížečů. V této době začalo vznikat mnoho nových technologií, které se specializovaly na vývoj robustních webových aplikací. Fakt, že nebylo nutné na cílových stanicích nic instalovat a k použití aplikace postačoval běžný prohlížeč také napomohl rozšíření webové platformy. Další změny ve vývoji software podnítil rozmach mobilních platforem, pro které začalo vznikat velké množství různorodých aplikací. Rovnoměrné zastoupení platforem mezi uživateli donutilo vývojáře vyvíjet aplikace pro každou z nich. Postupně se také začaly objevovat způsoby pro vývoj multiplatformních desktopových aplikací, které mají stále dostatek výhod oproti těm webovým. Výhody vycházejí z úzké spolupráce aplikace s operačním systémem, díky které má aplikace méně omezení a více privilegií. Pro její běh nemusí být vyžadováno žádné internetové připojení. To umožní využívat aplikaci v místech, kde je toto připojení nedostatečné nebo vůbec žádné např. na palubě letadla nebo v místech podzemních drah metra. Vysoká popularita jazyků jako je JavaScript a dalších webových technologií zapříčinila, že některé frameworky je využívají k vývoji klasických aplikací. Díky této skutečnosti je pro vývojáře snazší osvojit si tyto frameworky, a zároveň jsou sníženy náklady a čas vynaložený na vývoj.

Tato práce si dává za cíl představit a srovnat vybrané existující frameworky pro vývoj multiplatformní desktopové aplikace. Pro srovnávané frameworky budou zohledňovány jejich technologie, programovací jazyky, běhová prostředí a podporované cílové platformy. V práci bude také poukázáno na odlišnosti a specifika využití těchto frameworků na rozdíl od klasických, zaměřených na konkrétní platformu a technologii. Praktická část se bude věnovat realizaci aplikace ve vybraném frameworku.

1 KLASICKÉ FRAMEWORKY

1.1 .NET framework

.NET framework byl vyvinutý společností Microsoft. Poskytuje přístup k programátorským rozhraním služeb *Windows* a jeho API. Zajišťuje vývojová a běhová prostředí pro webové a desktopové aplikace systému *Windows*. Ten byl představen v roce 2000, ale první verze vyšla až roku 2003. Framework je součástí .NET platformy, která podporuje vývoj v jazycích C++, C#, F#, Visual Basic a dalších. Preferované vývojové prostředí je Visual Studio, případně existují alternativní vývojová prostředí. Schéma architektury frameworku je znázorněno na následujícím obrázku. [11]



Obrázek 1: Architektura frameworku .NET [11]

Podnětem, pro vývoj frameworku, bylo ulehčení vývoje služeb a aplikací na platformě *Windows*. Celý framework se skládá ze dvou částí: běhové prostředí *Common Language Runtime* (zkráceně CLR) a knihovny *Framework Class Library* (zkráceně FCL). První komponenta zajišťuje prostředí pro běh aplikací, které jsou napsané na platformě .NET. CLR podporuje každý jazyk, který je možné převést do jazyka *Common Intermediate Language* (zkráceně CIL). Díky tomu lze aplikace na platformě .NET psát v různých

programovacích jazycích. Knihovna FCL obsahuje několik tisíc definovaných datových typů, přičemž každý typ navíc zpřístupňuje specifickou funkcionalitu. Spojením těchto komponent získáme framework, který umožňuje vyvíjet: webové služby a aplikace, konzolové a formulářové aplikace pro windows, samostatné aplikační komponenty nebo služby operačního systému *Windows*. [10]

1.2 Cocoa framework

Framework *Cocoa* pochází z unixového prostředí. Původně se jednalo o sadu frameworků, které umožňovaly jednoduchou práci se správcem oken v systému NeXTSTEP (předchůdce *MacOS*). V roce 1993 se tato sada frameworků a nástrojů revidovala a přejmenovala na OpenStep. Tento název byl později změněn na název *Cocoa*. K vývoji aplikací, ve výše zmíněném frameworku, se doporučuje využít integrované vývojové prostředí Xcode. To je k dispozici ke stažení pouze pro systémy Mac, skrze aplikační obchod Mac App Store. Mezi podporované programovací jazyky frameworku patří C, Objective-C a jeho vylepšený a zároveň zjednodušený nástupce Swift. Podpora jazyka Swift je v IDE Xcode zahrnuta až od verze 6. [4] [6]

Frameworky v systému *MacOS* jsou v podstatě vysoce optimalizované knihovny, které poskytují přímé, intuitivní a konzistentní rozhraní k většině funkcí a služeb *MacOS*. Tyto frameworky jsou nedílnou součástí frameworku *Cocoa*. [9]

Všechny frameworky poskytované společností Apple, je možné na systému *MacOS* nalézt vestavěné v adresáři /System/Library/Frameworks. Frameworky třetích stran jsou umístěny ve složce /Library/Frameworks. *Cocoa* může využívat mnoho frameworků systému, nejčastěji jsou to však tyto: [4] [9]

Foundation – Poskytuje objektově orientovaným jazykům standardní datové typy, kolekce a užitečné třídy. Tím vytváří základní vrstvu funkcí pro vytvářené aplikace.

AppKit – AppKit je spojen s uživatelským rozhraním. Zajišťuje vytváření a práci s okny, tlačítky, textovými poli, událostmi a dalšími uživatelskými prvky.

CoreData – Framework, který má na starosti životní cyklus všech objektů a zajišťuje také jejich persistenci.

Pomocí IDE Xcode a frameworku *Cocoa*, lze tedy jednoduše vytvářet aplikace pro systémy *MacOS* a *iOS*. Xcode obsahuje kompilační nástroje, které dokáží vygenerovat aplikační balíček pro cílový systém. Na systému *MacOS* se ve skutečnosti jedná o složku, obsahující binární soubory, a případně další potřebné zdroje. Takto vytvořený balíček je přenositelný a je již možné ho spustit. [6]

2 CROSS-PLATFORM FAMEWORKY

Cross-platformové aplikace by měli být nezávislé na operačním systému (*MacOS*, *Linux*, *Windows*, *Android*, *iOS*), typu zařízení (mobilní telefon, počítač, tablet) a velikostí obrazovky. Specifika platforem se nejčastěji projevují u frontend částí, kde se platformy liší hlavně díky velikosti obrazovek a odlišným použitím. Přístupy, které je možné využít, pro vývoj cross-platformových aplikací:

1. webová aplikace;
2. nativní aplikace;
3. cross-platform framework.

Častou volbou, pro vývoj na více platforem, je webová aplikace. Pro většinu operačních systému je k dispozici webový prohlížeč, který je schopen podobným způsobem vykreslit obsah na každé platformě. Webové aplikace bývají provozovány na webovém serveru a je tedy nutné mít neustálé připojení k internetu. Díky tomu, je možné aplikace využívat bez nutnosti provedení jakékoliv instalace. Velké množství webových prohlížečů zvedá cenu jejich vývoje a následného udržování, nemluvě o podpoře mobilních verzí.

Nativní aplikace jsou vyvíjeny pod specifickým jazykem, určeným platformou. Například C, Objective-C a Swift pro platformu *iOS* a *MacOS*, nebo C++, C#, F#, Visual Basic a další pro platformu *Windows*. Pro každou aplikaci, je tedy nutné udržovat různé verze zdrojových kódů dle odpovídající platformy. Proto není tento přístup příliš vhodný pro menší firmy, které by musely vynaložit nemalé výdaje pro udržování vícero kódových základen aplikace. Výhodou těchto aplikací bývá jejich výkon a rychlost.

Cross-platform frameworky umožňují psát aplikaci pomocí jednoho programovacího jazyka, a tím snižují cenu vývoje. Tyto frameworky často využívají webové technologie. Na rozdíl od webových aplikací, ale mohou využívat nativních API operačních systémů. Využití webových technologií, nevyžaduje nutně internetové připojení. Proto je možné tímto způsobem vyvíjet online i offline aplikace. Následně budou představeny vybrané cross-platformové frameworky, s ohledem na vývoj desktopových aplikací, a platformě nezávislým přístupem jazyka *Java*.

2.1 Haxe

Haxe je open source sada nástrojů, složená z programovacího jazyka *Haxe*, kompilátoru a rozhraní příkazového řádku. Je ve veliké míře inspirován objektově orientovaným programovacím jazykem *ActionScript*, který byl vylepšen a obohacen o další funkcionalitu. Dále čerpá inspiraci z jazyka C#. Kód samotného jazyka *Haxe* může být snadno zaměněn se skriptovacím jazykem *ActionScript*, díky jejich značné podobnosti. Znalost jiného objektově orientovaného jazyka umožní snadné osvojení jazyka *Haxe*. Zdrojové kódy je pak možné pomocí kompilátoru zkompilovat na různé cílové platformy, což umožní udržovat poze jediný zdrojový kód, který bude fungovat jak na webu, tak na desktopových nebo mobilních platformách. [7]

Obrázek 2: Haxe logo [14]

2.1.1 Jazyk, technologie a prostředí

Jazyk byl navržen tak, aby byl jednoduchý a přesto výkonný. Syntaxe z veliké části odpovídá standardu *ECMAScript*, ale v případě potřeby se odchyluje. Úkázka kódu, který vytváří a následně vypisuje mapu, kde klíč značí jméno člověka a hodnota jeho zaměstnání: [14]

```
class Test {
    static function main() {
        var people = [
            "Elizabeth" => "Programming",
            "Joel" => "Design"
        ];

        for (name in people.keys()) {
            var job = people[name];
            trace('$name does $job for a living!');
        }
    }
}
```

S nainstalovaným *Haxe* a uloženým kódem v souboru "Test.hx", lze kód zkompilovat na cílové platformy skrze příkazový řádek:

```
haxe -main Test -js Test.js                    // -> JavaScript
haxe -main Test -java path/to/java/out         // -> Java
```

Nadace *Haxe* také vyvinula vývojové prostředí zvané HaxeDevelop. Toto prostředí je zdarma a nabízí plnou podporu pro jazyk *Haxe* společně s funkcemi jako generování kódu, doplňování příkazů, refaktorování, kompilace projektů, ladění a šablonami. Vyvíjet je možné i v dalších editorech kódu jako Visual Studio Code, Sublime Text, IntelliJ IDEA, Atom, Brackets a dalších. [12]

2.1.2 Architektura

Nástrojová sada *Haxe* se skládá z: [14]

Programovací jazyk HAXE – Vysokoúrovňový, silně typový programovací jazyk, používaný kompilátorem k produkci multiplatformního nativního kódu. Podobnost s jazyky *Java*, C++, PHP, AS3 a dalšími objektově orientovanými jazyky ho činí jednoduchým pro naučení. Jazyk *Haxe* byl speciálně navržen tak, aby se přizpůsobil chování jednotlivých nativních platforem a umožnil efektivní multiplatformní vývoj.

Cross-Kompilátor – Kompilátor *Haxe* je zodpovědný za překlad jazyka *Haxe* do nativního kódu cílové platformy nebo jeho binární podoby. Každá platforma je

nativně podporována, bez jakýchkoliv režijních nákladů z běhu uvnitř virtuálního stroje. Kompilátor je vysoce efektivní a dokáže kompilovat tisíce tříd během několika sekund.

Standardní knihovna – Standardní knihovna *Haxe* poskytuje společnou sadu rozhraní API, která umožní realizovat multiplatformní chování. Zahrnuje datové struktury, práci s matematikou a datem, serializaci, reflexe, bitové operace, kryptografii, souborový systém, databázový přístup, a další. Knihovna také obsahuje API pro konkrétní platformy, které umožní přístup k důležitým částem funkcionalit platformy, a které lze snadno rozšířit.

Další nástroje – Nástrojová sada je dodávána spolu s dalšími nástroji a funkcemi, které mohou být využity k vývoji a distribuci multiplatformních nástrojů. Sem se řadí rozšíření pro využití maker, balíkovací systém a další.

Frameworky a nástroje založené na HAXE – Na sadě *Haxe* bylo postaveno několik druhů frameworků, které je možné použít pro vývoj multiplatformních aplikací. Frameworky se liší na svém cílovém použití. Jsou rozděleny na vývoj multiplatformních her, API, nástrojů pro příkazový řádek,a nakonec webových, mobilní či desktopových aplikací.

2.1.3 Cílové platformy

Kód napsaný v jazyce *Haxe* lze snadno díky kompilátoru převést na zvolenou platformu. Není nutné udržovat různé verze stejného kódu pro různé platformy. Množství platforem, které kompilátor *Haxe* podporuje, je k nalezení níže v tabulce 1.

Tabulka 1: Haxe - cílové platformy [14]

Platforma	**Kód**	**Použití**
.NET Framework	C#	desktop, mobil, server
Adobe Flash	ActionScript3	desktop, web
Android, iOS, webOS	C++	mobil
HTML5, NodeJS, PhoneGap	JavaScript	desktop, mobil, server, web
Java	Java	desktop, server
Lua	Lua	CLI, desktop, mobil, web
NekoVM	Neko	CLI, desktop, Server
PHP	PHP	server
Python	Python	CLI, desktop, web
Windows, Linux, MacOS	C++	CLI, desktop, server

2.2 NW.js

NW.js bylo vytvořeno v roce 2011 Rogerem Wangem pracujícím v té době ve společnosti Intel v centru pro technologie s otevřeným kódem. Jedná se o framework pro budování desktopových aplikací pomocí webových technologií *HTML*, *CSS* a *JavaScript*. Projekt vznikl myšlenkou, že je možné spojit dvě existující technologie, v tomto případě *Node.js* a *WebKit*, k vytvoření plnohodnotných desktopových aplikací. Od spojení node-webkit, pak vznikl samotný název frameworku *NW.js*. [5]

Obrázek 3: NW.js logo [17]

2.2.1 Jazyk, technologie a prostředí

Aplikace je možné psát pouze ve značkovacím jazyce *HTML*, nebo společně se skriptovacím jazykem *JavaScript*. Pro minimální aplikaci jsou potřeba pouze dva soubory: [5]

Package.json – konfigurační soubor, obsahující základní informace o aplikaci, který je vyžadován *NW.js* pro funkční zavedení.

HTML soubor – soubor, který je zpravidla načten konfiguračním souborem package.json a následně vykreslen do okna aplikace. Nejčastěji je nazýván index.html, ale název může být libovolný např. app.html nebo main.html.

Pro vývoj je tedy možné použít libovolný textový editor či vývojové prostředí podporující webové technologie. Konfigurační soubor musí mít nutně pouze tři informační položky aplikace. Musí obsahovat alespoň název aplikace pro pole "name", zaváděcí soubor v poli "main" a nakonec verzi. Název se může skládat pouze z malých alfanumerických znaků, výsledný soubor může mít takovouto podobu:

```
{
  "name" : "hello-world",
  "main" : "index.html",
  "version" : "1.0.0"
}
```

Zaváděcí soubor lze zvolit *HTML* či *JavaScript*. Soubory *HTML* mají tendenci častěji zastávat tuto úlohu pro aplikace psané v *NW.js*. Následující kód demonstruje základní strukturu zaváděcího *HTML* souboru index.html:

```
<html>
  <head>
    <title>Hello World</title>
    <script>
      function sayHello () {
        alert('Hello World');
      }
    </script>
  </head>
  <body>
    <button onclick="sayHello()">Say Hello</button>
  </body>
</html>
```

Pro spuštění je nutné mít nainstalovaný modul *NW.js*, který je k dispozici skrze nástroj pro správu balíčků z *Node.js*. Po instalaci daného balíčku přes příkazový řádek, je možné spustit vytvořenou aplikaci z jejího adresáře:

```
npm install -g nw                    // -> instalace NW.js
nw                                   // -> spusteni vytvorene aplikace
```

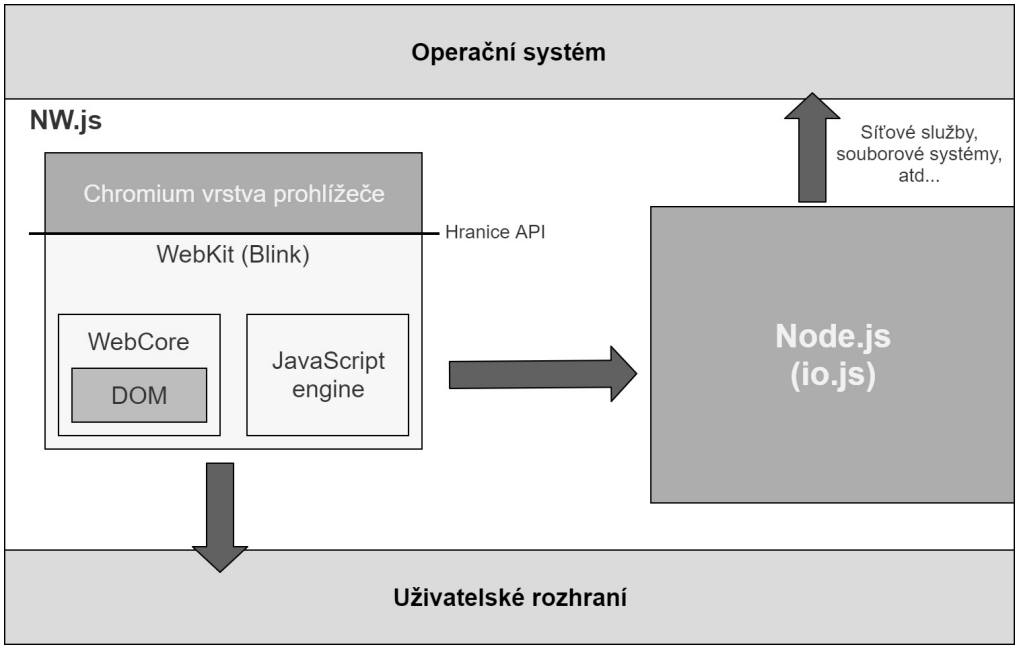
2.2.2 Architektura

NW.js je tedy webové běhové prostředí založené na Node.js a open source projektu prohlížeče *Chromium*. Skládá se tedy ze dvou hlavních komponent: [2]

Node.js – efektivní běhové prostředí pro *JavaScript* napsané v C++ a založené na *JavaScript* enginu V8 od společnosti Google. Ten se nachází v aplikační vrstvě operačního systému, *Node.js* má tedy přístup k hardwaru počítače, souborovému systému a síťovým funkcím což umožňuje jeho uplatnění v široké škále počítačových programů.

WebKit – layout engine, který má na starosti renderování webových stránek z objektového modelu dokumentu (zkráceně DOM). *NW.js* využívá *WebKit* verzi Blink, která pochází z oddělené větve klasického enginu *WebKit*. Blink byl vyvinut speciálně pro projekt prohlížeče *Chromium*.

Kombinací *Node.js* s enginem *WebKit*(Blink) vzniká *NW.js*, který už není omezen jako běžný prohlížeč. Má možnost přistupovat do operačního systému spolu s GUI aplikace. Schéma vnitřní struktury *NW.js* je vyobrazeno na následujícím obrázku.



Obrázek 4: NW.js schéma [2]

2.2.3 Cílové platformy

Díky užitým technologiím jsou aplikace v konečném důsledku webovými. Stejný kód psaný v jazyce *HTML* a *JavaScript* lze provozovat na vícero platformách. Pro správnou funkčnost postačuje podpora webového renderovacího jádra *WebKit*. Pomocí *NW.js* lze následně sestavit spustitelnou aplikaci na hlavní desktopové platformy *Windows*, *MacOS* a *Linux*.

2.3 Electron

Electron je framework pro vývoj desktopových aplikací, který je vytvořen a spravován společností GitHub. Původně byl postaven pro vývoj jejich vlastního textového editoru Atom a znám pod jménem Atom Shell. Framework umožňuje vývoj desktopových aplikací pomocí webových technologií *HTML*, *CSS* a *JavaScript*. Od svého vzniku v roce 2013, se stal velmi populární a je využíván ve velkém množství startupů a většími podniky pro vývoj vlastních aplikací. [5]

Obrázek 5: Electron logo [13]

Mezi populární aplikace napsané ve frameworku *Electron* patří:

Discord – komunikační software pro posílání textových zpráv a hlasové volání skrz počítačovou síť.

Atom – Textový editor obohacený o rozšiřující balíčky, pro vývoj aplikací obdobně jako v pokročilém IDE.

MS Visual Studio Code – Obdoba textového editoru Atom, podporuje velké množství programovacích jazyků a rozšíření od syntaxe, verzování, doplňování kódu až po ladění.

Slack – Populární týmový software pro podporu spolupráce a komunikace v pracovním prostředí.

Skype – Známá aplikace pro posílání textových zpráv a souborů, hlasovou komunikaci a video hovory.

2.3.1 Jazyk, technologie a prostředí

Vývoj *Electron* aplikací je možné uskutečnit v jednoduchém textovém editoru, nebo také přímo v editorech psaných v Electronu jako je Atom či Visual Studio Code. Obdobně jako u *NW.js* je nutné předepsat konfigurační soubor pro zavedení aplikace. Povinné položky jsou jméno, soubor pro zavedení a verze aplikace. Základní struktura souboru package.json:

```
{
  "name": "hello-world",
  "main": "main.js",
  "version": "1.0.0"
}
```

Vstupní soubor main.js psaný v jazyce *JavaScript*, slouží jako inicializační pro vytvoření hlavního okna aplikace. Obsahuje také reakce na běžné události jako je zavření okna, aktivace okna, připravenost okna a další. Jednoduchý zaváděcí soubor pro aplikace psané v Electronu může vypadat následovně:

```
const {app, BrowserWindow} = require('electron')
let mainWindow

function createWindow () {
  mainWindow = new BrowserWindow({width: 800, height: 600})
  mainWindow.loadFile('index.html')

  mainWindow.on('closed', function () {
    mainWindow = null
  })
}

app.on('ready', createWindow)
app.on('window-all-closed', function () {
  if (process.platform !== 'darwin') {
    app.quit()
  }
})

app.on('activate', function () {
  if (mainWindow === null) {
    createWindow()
  }
})
```

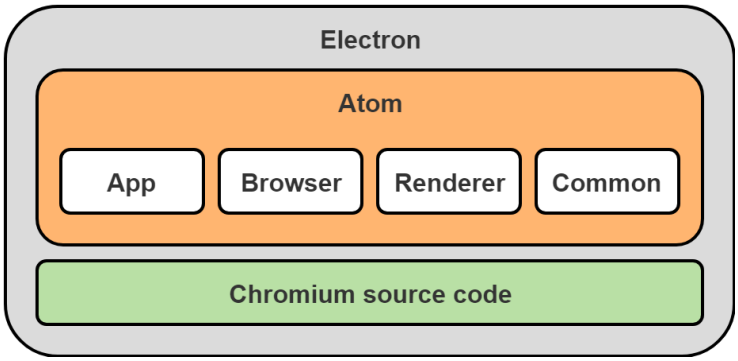
Index.html může mít stejný obsah jako v ukázce u frameworku *NW.js* v kapitole 2.2.1. Posledním krokem, je podobně jako u *NW.js* instalace frameworku pomocí *Node.js*.

Instalaci je možné provést přímo v adresáři aplikace, nebo odkudkoliv s použitím příznaku pro globální instalaci "-g". Lokální instalace Electronu a následné spuštění aplikace skrz příkazový řádek má následující podobu:

```
npm install electron            // -> instalace Electron
electron .                      // -> spusteni vytvorene aplikace
```

2.3.2 Architektura

Electron, podobně jako *NW.js*, využívá spolupráce *Node.js* a jádra *Chromium*. V *NW.js* je *Chromium* upraveno tak, aby společně s *Node.js* sdílelo stejný prostor vyhrazený pro *JavaScript*. V Electronu je využito obsahové API Chromia a pomocného modulu "node-bindings" z *Node.js*. Node.js má tedy k dispozici jeden sdílený context pro *JavaScript*, zatímco *Electron* obsluhuje minimálně dva oddělené. Jeden hlavní pro back-end procesy, které obsluhují aplikační okno. A další pro každé aplikační okno, pro tzv. renderovací proces. Jako vstupní bod využívá *Electron* obvykle JavaScriptový soubor, díky tomu následně deleguje odpovědnosti za načítání aplikačních oken. [5]



Obrázek 6: Architektura zdrojového kódu Electron [5]

Obrázek 6 znázorňuje zjednodušenou strukturu zdrojového kódu *Electron*. Díky oddělení zdrojového kódu jádra *Chromium* a samotné aplikace, je možné snadněji spravovat a také vylepšovat samotné komponenty Chromia stejně tak i prvky aplikace.

Komponenta Atom je napsána v jazyce C++ a tvoří shell. Zdrojový kód *Chromium* je používán shellem Atom ke zkombinování Chromia a *Node.js*. [5]

2.3.3 Cílové platformy

Electron podporuje následující platformy: [13]

macOS – Pro *MacOS* jsou k dispozici pouze 64 bitové binární soubory, a minimální podporovaná verze je *MacOS* 10.9.

Windows – Pro *Windows* jsou k dispozici binární soubory pro 32 a 64 bitovou architekturu. Podporované verze jsou *Windows* 7 a novější.

Linux – *Electron* garantuje podporu pro distribuci *Ubuntu* verze 12.04 a novější. Dále jsou ověřeny distribuce Fedora 21 a Debian 8.

2.4 Java

Java patří mezi nejpopulárnější programovací jazyky dnešní doby. Byla vytvořena Jamesem Goslingem ze společnosti Sun Microsystems, nyní Oracle Corporation, v první polovině roku 1995. Založení tohoto jazyka si pokládalo za cíl poskytnout platformě nezávislou alternativu k jazyku C++. Jde o jazyk se všeobecným využitím, který je vhodný na téměř jakýkoliv druh aplikací. Jazyk je open-source a objektově orientovaný. Platformní nezávislost vychází z faktu, že kód psaný v jazyce *Java* není uzpůsoben cílové platformě ani hardware na kterém poběží. Kód a aplikace jsou spouštěny v běhovém prostředí zvaném "*Java Runtime Environment*", zkráceně JRE, které je k dispozici pro platformy *Windows*, *MacOS*, *Linux* a *Solaris*. [3]

Obrázek 7: Java logo [17]

2.4.1 Jazyk, technologie a prostředí

Jazyk *Java* je tedy pro všechny platformy identický. Pro vývoj na odlišné platformy je nutné mít k dispozici vývojové nástroje označované jako "*Java Development Kit*", zkráceně JDK. V JDK je obsažen kompilátor, který je schopen převést zdrojový kód do platformě nezávislé podoby. Pro spuštění aplikace postačuje, již platformě závislé, běhové prostředí JRE. V JDK je obsaženo kompletní JRE spolu s kompilátorem a dalšími vývojovými nástroji. [3]

Program je možné napsat v jakémkoliv textovém editoru a poté zkompilovat v příkazovém řádku. Výsledný soubor má koncovku ".java". Ukázka kódu souboru HelloWorld.java:

```
public class HelloWorld {
        public static void main(String[] args){
                System.out.println("Hello World");
        }
}
```

Dalším krokem je zkompilování zdrojového kódu v jazyce *Java*, a následné spuštění výsledného běhového kódu virtuálního stroje *Java* :

```
javac HelloWorld.java          // -> kompilace HelloWorld.class
java HelloWorld                // -> spusteni vytvorene aplikace
```

Pro jazyk *Java* existuje mnoho populárních vývojových prostředí, například Eclipse, NetBeans, IntelliJ IDEA a další.

2.4.2 Architektura

Od roku 2008 je k dispozici od společnosti Sun Microsystems softwarová platforma pro vývoj desktopových aplikací s bohatými grafickými prvky zvaná *JavaFX*. Jedná se o sadu grafických a mediálních balíčků, které umožňují vývojáři navrhovat, vytvářet, testovat, ladit a nasazovat aplikace, bohaté na funkcionalitu i vzhled napříč různorodým platformám. *JavaFX* verze 8 a novější nabízí následující výčet klíčových funkcí: [15]

Java API – *JavaFX* je knihovna napsaná jako API pro jiné knihovny, a sama může využívat API dalších knihoven v jazyce *Java*.

FXML a Scene Builder – *FXML* je deklarativní značkovací jazyk založený na XML pro vytváření uživatelského rozhraní *JavaFX* aplikací. Alternativně je možné GUI interaktivně navrhovat přímo v aplikaci Scene Builder, která generuje *FXML* kód návrhu.

WebView – Webová komponenta, která umožňuje zanořit a zobrazit webové stránky uvnitř *JavaFX* aplikace.

Swing interoperabilita – Existující Swingové aplikace lze obohatit o prvky z knihovny *JavaFX*, nebo využít Swing komponenty v *JavaFX* aplikaci.

Vestavěné UI komponenty a CSS – Knihovna obsahuje běžné ovládací prvky, které je možné upravit pomocí stylů *CSS*.

Grafické téma Modena – Nové téma nahrazuje starší grafické téma Caspian.

3D prvky – API disponuje novými třídami pro práci s 3D objekty a možností nastavení vyhlazování hran scény.

Canvas API – Možnost kreslení přímo do zvolené oblasti grafického elementu scény.

Printing API – Balíček pro podporu tisku skrze *JavaFX* Printing API.

Podpora obohacených textů – Možnost rozšířených úprav a stylů pro textové prvky.

Podpora multitouch – Podpora pro multitouch operace.

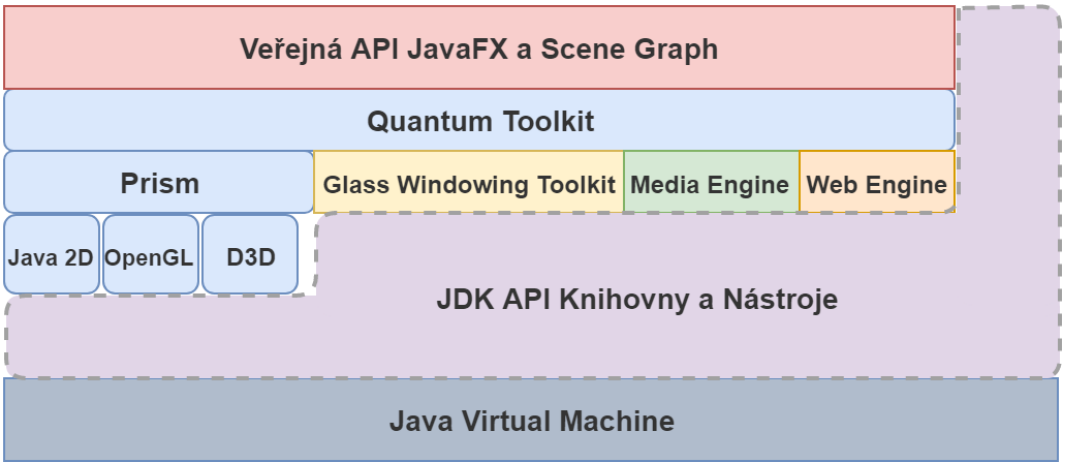
Podpora Hi-DPI – Podpora Hi-DPI displejů.

Hardwarově akcelerované vykreslování – API, které umožní podporovaným grafickým kartám a výpočetním jednotkám účastnit se vykreslovacího procesu.

Výkonný přehrávač médií – Umožní stabilní přehrávání webových multimédií společně s nízkou latencí.

Soběstačný model nasazení aplikace – Balíčky obsahují všechny zdroje aplikace společně s kopií s běhového prostředí *Java* a *JavaFX*.

Následující obrázek zobrazuje architekturu spolu s rozvrstvením *JavaFX*, API a komponentami, které využívá. Scene graph slouží jako výchozí bod při budování aplikací. Zpracovává uživatelské vstupy a usnadňuje práci s grafickými elementy. Quantum toolkit pomáhá zajistit spolupráci a dostupnost grafického modulu Prism spolu se souborem nástrojů pro práci s okny, který se nazývá Glass toolkit. Žádná z těchto komponent není veřejně přístupná, jsou však užívány pro běh *JavaFX* aplikací.



Obrázek 8: Architektura JavaFX [15]

2.4.3 Cílové platformy

Aplikace napsané v jazyce *Java* jsou následně zkompilovány do platformě nezávislé formy, kterou je možné spustit pomocí platformě závislého JRE. Desktopové platformy, které jsou podporovány je možné vidět v tabulce 2. Platformy využívající architektury procesorů typu ARM nejsou pro *JavaFX* podporovány. Od *JavaFX* verze 2.2 je podporována většina desktopových platforem, běžících na architekturách x86 a x64.

Tabulka 2: Java - cílové platformy [15]

Platforma	**CPU Architektura**	**Verze OS**	**Zavedeno ve verzi**
Windows Vista	x86 a x64	SP2	2.0
Windows 7	x86 a x64	SP1	2.0
Windows 8+	x86 a x64		2.2.4
MacOS X	x64	10.7.3+	2.2
Linux	x86 a x64	Ubuntu 10.4+, gtk2 2.18+	2.2

2.5 Odlišnosti od klasických frameworků

Při zpětném ohlédnutí ke kapitole 1, můžeme sledovat jisté podobnosti a rozdíly vývoje při použití klasických frameworků od těch cross-platformových. U obou je možné volit libovolnou strukturu architektury aplikace, a nejsme tedy striktně svazováni frameworkem k dodržení specifické konstrukce. Pro práci s operačním systémem jsou ve všech případech využívána systémová aplikační rozhraní, a jsme tedy limitováni pouze frameworkem nepodporovanými funkcemi knihoven systému. Framework vlastně zpřístupní funkcionalitu nativních API pomocí jednotného rozhraní. Například u frameworku *Electron* je možné

nastavit spuštění aplikace po zapnutí systému. Electron na pozadí vybere metodu dle cílové platformy. Takže na platformě *Windows* se toto provede pomocí zápisu do registrů skrze *Windows Registry API*, oproti tomu se na platformě *Linux* vytvoří konfigurační soubor "Desktop Entry". Dalším příkladem může být využití společné sady rozhraní skrze standardní knihovnu ve frameworku *Haxe*. Ta obsahuje společné funkcionality operačních systémů spolu se specializovanými knihovnami pro jejich odlišné funkce. Proto při použití pouze těch společných nemusí být aplikace tak bohatá na funkcionalitu. Na rozdíl od multiplatformního přístupu klasické frameworky přistupují ke všem funkcím, které API systému poskytuje bez nutnosti použití zpřístupňujících knihoven. Oba přístupy, klasické i cross-platform, umožňují vývoj s pomocí specializovaných vývojových prostředí.

Mezi viditelnější rozdíly můžeme zařadit volbu programovacího jazyka. U srovnávaných cross-platform frameworků se setkáváme častěji s užitím skriptovacího jazyka *JavaScript*, dále s jazykem *Java* a specializovaným jazykem *Haxe*. Kdežto pro klasické jsou využity jazyky C, Objective-C, C++, C# a Swift. Frameworky se od sebe také liší v použitých běhových prostředích.

Frameworky *Electron* a *NW.js* jsou postaveny na webových technologiích *HTML*, *CSS* a *JavaScript*. *Java* využívá bohatou softwarovou platformu *JavaFX*. A framework *Haxe* využívá vlastní grafickou knihovnu "HaxeUI", která podporuje nástroje knihovny "WxWidgets" obsahující prvky pro nativní vzhled aplikací. Návrh a implementace grafického uživatelského rozhraní se tedy pro frameworky liší. Hlavním důvodem jsou dostupné knihovny grafických prvků, které nejsou vždy kompatibilní na všech systémech. U klasických frameworků je uživatelské rozhraní vždy uzpůsobeno dle dostupných knihoven na cílové platformě. Zatímco např. pro cross-platform framework *Electron* jsou využity nativní pouze prvky pro okna a dialogy, zbytek GUI je možné dotvořit pouze pomocí webových technologií. GUI je pak interpretováno jádrem *Chromium*, což zajistí shodný uživatelský zážitek napříč platformami. GUI frameworku *Electron* lze implementovat kompletně webovými technologiemi, nebo je možné využít standardní ovládací prvky (např. aplikační okno), aby byl zachován jednotný styl operačního systému. Díky tomuto principu je snadné vytvářet vlastní (např. animované), nebo použít existující webové grafické prvky pro desktopové aplikace pouze se znalostí *HTML* a *CSS*.

2.6 Shrnutí

Zhodnocení a porovnání vybraných frameworků pro vývoj desktopové aplikace, bylo provedeno s ohledem na použitý jazyk, podporované platformy, licenční omezení, kvalitu dostupné dokumentace, dostupnost pomocných knihoven, možnost využít webové technologie, velikost komunity, rozšíření frameworků, možnosti tvorby grafického rozhraní aplikací a výstupu frameworku. Srovnání se nachází v následující tabulce.

Tabulka 3: Srovnání cross-platform frameworků

	Haxe	**NW.js**	**Electron**	**Java**
Jazyk	Haxe	JavaScript	JavaScript	Java
Podpora Windows	ano	ano	ano	ano
Podpora Linux	ano	ano	ano	ano
Podpora MacOS	ano	ano	ano	ano
Licence	GPL	MIT	MIT	Oracle Binary Code License[1]
Dokumentace	⋆ ⋆	⋆ ⋆ ⋆	⋆ ⋆ ⋆	⋆ ⋆
Dostupnost knihoven	⋆ ⋆ ⋆	⋆ ⋆ ⋆ ⋆	⋆ ⋆ ⋆ ⋆	⋆ ⋆ ⋆
Webové technologie	ano	ano	ano	ne
Velikost komunity	⋆	⋆	⋆ ⋆	⋆ ⋆ ⋆ ⋆
Reálné rozšíření	⋆	⋆	⋆ ⋆ ⋆	⋆ ⋆ ⋆ ⋆
Možnosti GUI	⋆ ⋆	⋆ ⋆ ⋆ ⋆	⋆ ⋆ ⋆ ⋆	⋆ ⋆ ⋆
Výsledná aplikace	nativní	nativní	nativní	JVM

Všechny srovnávané frameworky umožňují vývoj pro platformy *Windows*, *Linux* a *MacOS*. Každý z nich je šířen pod svobodnou licencí kromě platformy *Java*, která je pod licencí zabraňující její modifikace. Oficiální dokumentace frameworků byly hodnoceny především na základě přehlednosti, obsáhlosti, příkladů použití a snadného vyhledávání. Následně bylo zhodnoceno jak bohaté jsou jejich knihovny, a jak obtížné je nalezení a následná instalace balíčků a funkcí. Možnost využít webové technologie při vývoji desktopových aplikací bylo bráno jako výhoda. Velikost komunity ovlivňuje obtížnost nalezení řešení běžných problémů. Reálné rozšíření je dáno množstvím existujících aplikací. Jednoduchost přizpůsobení jednotlivých ovládacích prvků bylo klíčové při hodnocení možností grafického uživatelského rozhraní frameworků. Poslední hodnotící

[1] Java SE a JavaFX spadají pod licenci BCL: `http://www.oracle.com/technetwork/java/javafx/overview/faq-1446554.html#10`

bod závisel na výstupu frameworku, s ohledem na nutnost použít běhové prostředí na cílové platformě.

Za zmínku stojí využití frameworku *Haxe* pro tvorbu her, který je díky velkému množství knihoven, zaměřených na tuto oblast, více něž vhodný kandidát. Nevýhodou může být malá komunita nebo nutnost naučit se nový jazyk. Oproti tomu, u známější platformy *Java* je největší nevýhodou nutnost instalace běhového prostředí. Další nevýhoda je, že neumožňuje při vývoji desktopové aplikace využít webové technologie. *NW.js* je téměř shodný s frameworkem *Electron*, nicméně je celkově méně rozšířený.

Ná základě srovnání jednotlivých frameworků, byl tedy pro tvorbu vlastní desktopové aplikace zvolen framework *Electron*. Především kvůli bohatým knihovnám, jednoduché rozsáhlé dokumentaci, úspěchu a rozšíření reálných aplikací, podpoře webových technologií a díky možnosti nativnímu výstupovému kódu frameworku.

3 VÝVOJ VLASTNÍ APLIKACE

Tato kapitola se bude zabývat představením aplikace, specifikací požadavků a vlastním návrhem. Vyvíjená aplikace bude sloužit jako jednoduchý program pro vytváření úkolů nebo poznámek.

3.1 Představení aplikace

Aplikace "Úkolovníček" umožní uživatelům vytvářet úkoly s popisky a k jednotlivým úkolům ukládat symbolické odkazy nebo přímo soubory. U každé poznámky je možné naplánovat připomínku, která zvukovým signálem oznámí blížící se konec. Data jsou perzistentně ukládány do databáze, a uživateli jsou umožněny nad nimi provádět běžné CRUD (Create, Read, Update, Delete) operace pro manipulaci s daty.

3.2 Požadavky

První kroky vývoje, by měly zahrnout analýzu. Analýza se zaměřuje na zodpovězení otázek "co se vyvíjí" a nebo "co by mělo být implementováno" a její součástí je určení funkčních a nefunkčních požadavků. Požadavky tvoří základ každého systému a vyjadřují jeho stěžejní funkci. Měli by vyjadřovat co by měl systém dělat, ale nikoli jakým způsobem. Každý požadavek je doporučeno značit jedinečným identifikátorem, klíčovým slovem "bude" a jeho příkazovou funkcí. [1]

3.2.1 Funkční požadavky

V následující tabulce jsou funkční požadavky, které určují výsledné chování a funkcionalitu aplikace.

Tabulka 4: Požadavky - funkční

Značení	Popis
REQ01	Uživatel bude moci vytvářet úkoly.
REQ02	Uživatel bude moci upravovat vytvořené úkoly.
REQ03	Uživatel bude moci smazat vytvořené úkoly.
REQ04	Uživatel bude moci nastavit upozornění u jednotlivých úkolů.
REQ05	Uživatel bude moci aktivovat / deaktivovat funkci upozornění.
REQ06	Uživatel bude moci přidat soubory k jednotlivým úkolům.
REQ07	Uživatel bude moci přidat symbolické odkazy na soubory k jednotlivým úkolům.
REQ08	Uživatel bude moci otevírat soubory či odkazy pomocí výchozích aplikací počítače.
REQ09	Uživatel bude moci skrýt aplikaci do systémové lišty.
REQ10	Uživatel bude moci importovat / exportovat stav aplikace.

3.2.2 Nefunkční požadavky

V tabulce 5 nalezneme požadavky nefunkční. Ty definují omezující podmínky nebo specifické vlastnosti vyvíjené aplikace.

Tabulka 5: Požadavky - nefunkční

Značení	Popis
REQ11	Aplikace bude vyvíjena ve frameworku Electron.
REQ12	Aplikace bude disponovat grafickým uživatelským rozhraním.
REQ13	Aplikace bude umožňovat persistenci svého stavu.
REQ14	Aplikace bude zajišťovat zabezpečení stavu aplikace při exportu pomocí zašifrování.

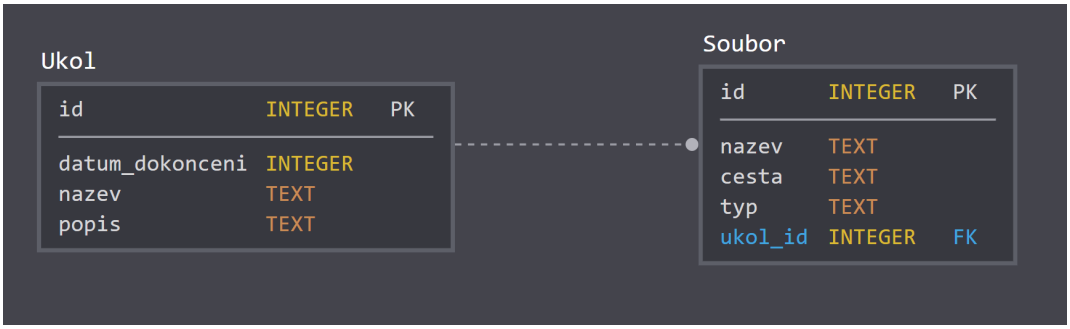
3.3 Návrh

Druhým krokem vývoje aplikace je návrh. Během návrhu je snaha vytvořit takovou strukturu, která by vedla ke splnění všech definovaných požadavků. V návrhu aplikace budou představeny databázový model, struktura aplikace a grafické uživatelské rozhraní.

3.3.1 Databázový model

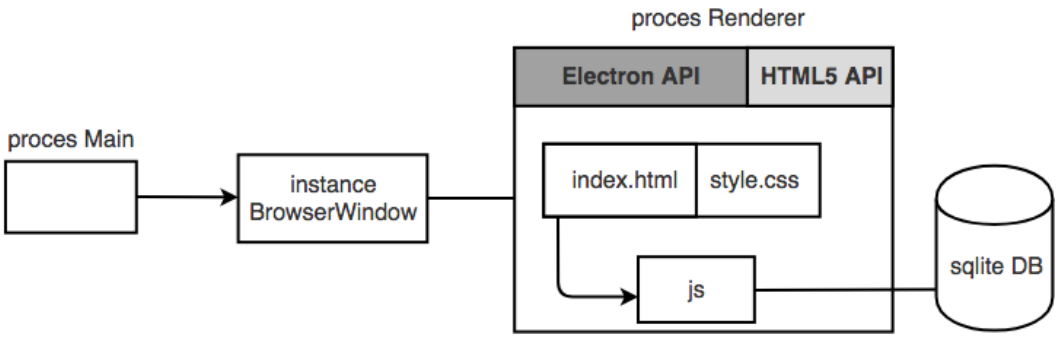
Pro zpracování a ukládání dat byl zvolen databázový systém SQLite. Ten byl původně vydán v roce 2000 a navržen takovým způsobem, který by zprostředkoval pohodlný přístup ke všem datům společně s manipulací bez vysokých režijních nákladů. Jedná se o relační databázi s otevřeným kódem, zajišťující veškerou funkcionalitu z jediného

souboru. SQLite má vybudovanou kladnou reputaci pro vysokou přenositelnost, snadné použití, kompaktnost, efektivnost a spolehlivost. [8]



Obrázek 9: Databázový model

Na obrázku 9 je znázorněno schéma databázového modelu, který obsahuje dvě použité relace (tabulky), které nesou názvy "Ukol" a "Soubor". Atributy relací (sloupce tabulek) jsou omezena integritním omezením typu. Integritní omezení zařazují atribut do přiřazené domény, čímž následně zabraňují vložení nepovolených hodnot do tabulek. Každá tabulka uchovává informace o jednom objektu, který je identifikován primárním klíčem "id". Pro každý úkol jsou vedeny údaje jako jeho název, podrobný popis, případně datum ukončení pro možnosti upozornění. Každý úkol může uchovávat libovolný počet souborů, u který lze sledovat název, cestu a typ, který značí přípona souboru. Tabulka "Soubor" také disponuje cizím klíčem zvaným "ukol_id", který odkazuje na primární klíč tabulky "Ukol". Díky němu je možné určit které záznamy jedné tabulky patří ke kterým záznamům tabulky druhé, a navíc umožní databázi zajistit referenční integritu zúčastněných tabulek. Referenční integrita zabraňuje vložení hodnoty cizího klíče, pro které neexistuje záznam primárního klíče na který odkazuje.



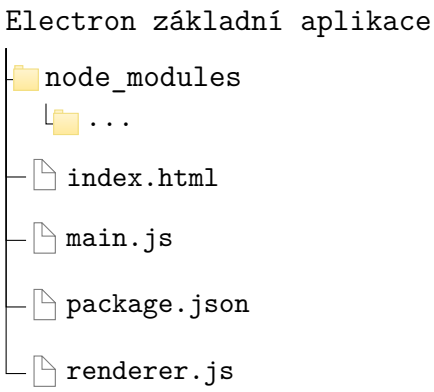
Obrázek 10: Přístup k databázi

Schéma na obrázku 10 znázorňuje jakým způsobem aplikace psaná ve frameworku *Electron* přistupuje k databázovému systému SQLite. Hlavní proces frameworku vytváří

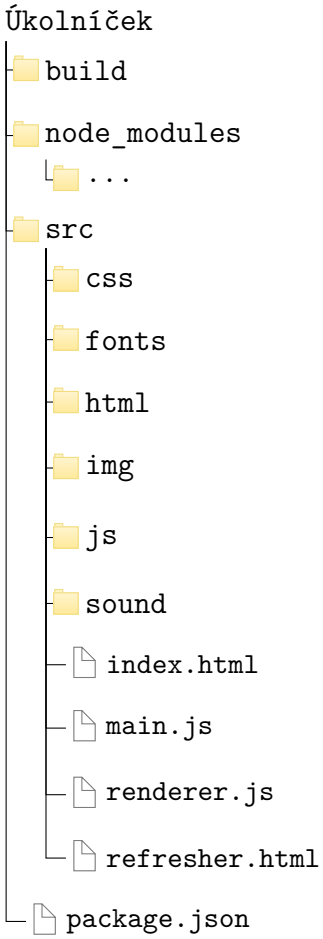
instance aplikačních oken BrowserWindow. Samotná okna běží ve vlastních vykreslovacích procesech Renderer, které zajišťují chod webových stránek. Pro manipulaci nebo přístup k datům jsou použity funkce napsané v JavaScriptovém souboru.

3.3.2 Struktura aplikace

Aplikační struktura základní aplikace bez jakékoliv úpravy se skládá z kořenového adresáře, ve kterém je obsažena pouze složka node_modules, která obsahuje knihovny třetích stran. V adresáři je povinný soubor package.json, který slouží jako zaváděcí pro hlavní proces aplikace. Jedná se o JavaScriptový soubor main.js, který obstarává chod celé aplikace a mezi-procesní komunikaci. Vytváří také instanci okna, která pomocí renderovacího procesu vykresluje obsah webové stránky index.html. Tento postup je možné vidět na obrázku 10. Celá struktura je graficky znázorněna v následujícím schématu.



Předchozí struktura struktura byla upravena, aby připomínala strukturu běžných webových aplikací. Zdrojové soubory aplikace jsou odděleny od souborů, které jsou používány k zavádění a testování samotné aplikace. Např. veškeré obrázky, které jsou v aplikaci použity, nalezneme na cestě src\img. Stejně intuitivně jsou rozmístěny skripty, fonty, styly a zvukové soubory. Kompletní schéma je znázorněno níže.



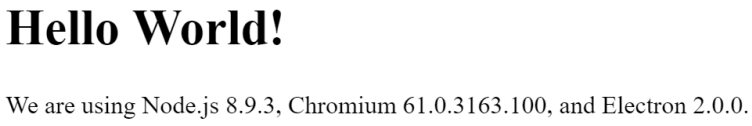
3.3.3 Grafické uživatelské rozhraní

Po návrhu databáze, specifikaci přístupu k ní, a zavedení základní struktury aplikace je dalším úkolem navrhnout její vhodné grafické uživatelské rozhraní (dále GUI). Největší výzvou je vytvoření GUI pomocí webových technologií takovým způsobem, aby aplikace působila a ovládala se jako desktopová. Zároveň musí být užito komponent a ovládacích prvků, které by vhodně plnily všechny funkční požadavky, které byly definovány.

Vzhled aplikace je značně ovlivněn záhlavím a rámečkem aplikačního okna. Tyto prvky se navíc na různých platformách liší. Z tohoto důvodu, bylo pro sjednocení vzhledu použito okno bez rámečku. Aplikace díky tomu působí méně jako webová, ale je pak nutné implementovat vlastní panel pro ovládací prvky např. pro vypnutí aplikace. Klasické okno s webovým rámečkem je možné vidět na obrázku 11.

Obrázek 11: Okno s ohraničením

Pro vytvoření okna bez rámečku je nutné, při vytváření hlavní okna, nastavit parametr frame na “false”. Alternativně lze na systému *MacOS* rámeček pouze skrýt, pak by ale nebylo možné mít vzhled konzistentní napříč platformami. Následující obrázek zobrazuje okno bez ohraničení.

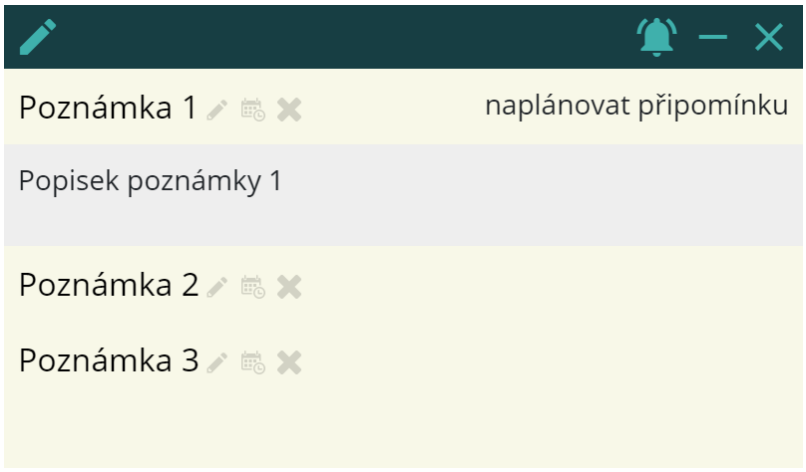


Obrázek 12: Okno bez ohraničením

Nastavení parametru je možné provést v hlavním skriptu “main.js”, kde se okno vytváří. Kód může vypadat následovně:

```
const {BrowserWindow} = require('electron')
let win = new BrowserWindow({width: 800, height: 600, frame: false})
win.show()
```

Jak již bylo zmíněno, celý framework *Electron* využívá webové technologie pro vývoj desktopových aplikací. Celá struktura je tedy tvořena pomocí značkovacího jazyka *HTML5*, zatímco vzhled a vizuální uspořádání je sestaveno prostřednictvím kaskádových stylů *CSS*. S těmito technologiemi je pak propojen *JavaScript*, který zajistí dynamičnost a funkcionalitu celé aplikace. Výsledný vzhled je možné vidět na následujícím obrázku.



Obrázek 13: Výsledný vzhled aplikace

4 IMPLEMENTACE

V této kapitole se nachází implementace aplikace ve zvoleném frameworku *Electron*. V této části budou představeny hlavně zajímavé prvky a funkce, které řeší nejběžnější problémy jako jsou práce s databází, funkcionality spojené s přístupem k operačnímu systému, mezi-procesní komunikace, a další specifika frameworku *Electron*.

4.1 Vlastní ovládací prvky

Při grafickém návrhu byl odstraněn rámeček spolu s ovládacími prvky okna, proto bylo nutné je nahradit vlastními. Prvky jsou zasazeny do vrchní panelu aplikace, kde intuitivně zaujímají místo původních prvků. Předešlá funkcionalita je obnovena voláním události při stisknutí ikony požadované akce. Pro zavření okna postačuje zavolat metodu close() na aktuálním okně. Pro minimalizace nelze jednoduše zavolat metodu jako v případě zavření okna. Tato funkce není prohlížeči podporována. Je tedy nutné získat pomocí Electronu referenci na aktuální okno, a pomocí vzdálené procedury minimize() okno minimalizovat. Následuje ukázka implementace této části:

```
<script>
    const electron = require("electron").remote;
</script>
<nav>
   <div>
      <i onclick="remote.getCurrentWindow().minimize()">remove</i>
      <i onclick="window.close()">clear</i>
   </div>
</nav>
```

4.2 Mezi-procesní komunikace

V Electronu je použit multi-procesní model architektury. Proto je velice důležité zajistit komunikaci mezi všemi procesy. Jak již bylo zmíněno, existují dva druhy procesů. Hlavní proces aplikace, a renderovací pro každou samostatnou stránku. Komunikaci mezi nimi zajišťuje modul IPC (*Inter-Process Communication*), který posílá serializované zprávy

ve formátu JSON. Následuje ukázka kódu zpracování, a odeslání požadavku na získání nastavení notifikací, pro zobrazení správné informační ikony stavu:

```
const ipc = electron.ipcRenderer;

ipc.send("requestNotificationSettings");

ipc.on("responseNotificationSettings", (ev, bool) => {
  setNotify(bool);
});
```

Předchozí ukázka obsahuje odeslání požadavku pomocí metody ipc.send(), kdy se renderovací proces dotazuje hlavního. A zpracování odpovědi metodou ipc.on(). Obdržení požadavku a odeslání odpovědi v hlavním procesu vypadá následovně:

```
const ipc = electron.ipcMain;

ipc.on("requestNotificationSettings", ev => {
  ev.sender.send("responseNotificationSettings", mainWindow.notifications);
});
```

4.3 Otevření souborů pomocí výchozí aplikace systému

Pro využití uživatelsky definovaných aplikací, k otevření souborů přímo z aplikace, je využit modul shell. Ten poskytuje funkce integrace aplikace se samotným systémem. Umožňuje například:

- zobrazit soubor v souborovém systému;
- otevřít soubor výchozí aplikací;
- otevřít a zobrazit výsledek zadané url adresy;
- přesunout soubor do koše;
- přehrát systémový zvuk pro pípnutí;
- a další.

Pro každý úkol je možné uchovávat soubory nebo pouhé odkazy na soubory. Pokud byl vymazán, zobrazí se uživateli dialog o ponechání či smazání souboru s neplatnou cestou. Po kliknutí na název souboru s platnou cestou se volá následující funkce:

```
const ipc = electron.ipcRenderer;
const shell = electron.shell;

function openInShell(cesta, idUkol, idSoubor) {
  if (!shell.openExternal("file://" + cesta)) {
    ipc.send("open-file-error", idUkol, idSoubor);
  }
}
```

4.4 Práce s DB

Připojení a práci s SQLite databází obstarává node.js balíček sqlite-sync. Jedná se o jednoduchý modul umožňující synchronní a asynchronní provedení SQL příkazů. Vytvoření databázového souboru je možné funkcí connect(), která vyžaduje cestu jako parametr. Pokud v dané cestě soubor neexistuje, tak je automaticky vytvořen. V opačném případě se pouze vytvoří spojení, které je nutné v obou případech po ukončení práce uzavřít. Následující kód naznačuje způsob práce s databází.

```
const sqlite = require("sqlite-sync");
const table_ukol = "Ukol";
// otevreni spojeni
sqlite.connect(dbPath);
// nova tabulka
sqlExecute(`CREATE TABLE ${table_ukol}(id INTEGER PRIMARY KEY
AUTOINCREMENT, datum_dokonceni INTEGER, nazev TEXT, popis TEXT);`);
// select
sqlite.run(`SELECT * FROM ${table_ukol};`);
// uzavreni spojeni
sqlite.close();
```

4.5 Minimalizování aplikace

Funkce skrytí aplikace do oznamovací oblasti je uskutečněna zpracováním události minimalizace po stisknutí minimalizačního tlačítka, které bylo implementováno v podkapitole 4.1 Vlastní ovládací prvky. Prvním krokem minimalizace je skrytí hlavního okna, následně je nutné inicializovat ikonu aplikace pro oznamovací oblast, a v neposlední řadě zaregistrovat událost pro obnovení aplikace zpět z minimalizovaného stavu. Ukázka implementace:

```
const Tray = electron.Tray;
mainWindow.on("minimize", ev => {
        mainWindow.hide();
        appIcon = new Tray(iconPath);
        appIcon.on("click", restoreMinimized);
});

function restoreMinimized() {
  mainWindow.show();
  appIcon.destroy();
}
```

Pro vytvořenou ikonu aplikace je možné definovat kontextové menu, pomocí modulu electron.Menu, a zobrazit definovaný popisek po najetí myší na ikonu. Kód kontextového menu a popisku:

```
const menu = electron.Menu;
appIcon.setToolTip("App tooltip");
const contextMenu = menu.buildFromTemplate([{
     label: "Export",
     click: exportData
   },{
     label: "Import",
     click: importData
   },{
     label: "Ukoncit",
     click: quitApp
   }]);
appIcon.setContextMenu(contextMenu);
```

4.6 Notifikace

Každému úkolu je možné naplánovat připomínku na určitý den a hodinu. Využitý balíček, implementující notifikace, nese název electron-notify. Vykreslí jednoduché oznámení do pravého spodního rohu obrazovky systému. Umožňuje zobrazit obrázek aplikace, přehrát zvukový soubor, otevřít URL a je podporován všemi platformami. Po naplánování připomínky, se automaticky obnovuje zbývající čas do jejího konce. Pět minut před koncem, je vyvolána notifikace se zvukovou signalizací, aby upozornila na končící úkol. Ukázka vyvolání z hlavního procesu:

```
const eNotify = require("electron-notify");

eNotify.setConfig({
    appIcon: path.join(__dirname, "img/windows-icon.png"),
    displayTime: 10000,  // doba zobrazeni v milisekundach
    defaultStyleClose: {
        cursor: "pointer",
        position: "absolute",
        top: 1,
        right: 3,
        fontSize: 11,
        color: "#777"
    }
});
eNotify.notify({
    title: "Nadpis notifikace",
    text: "Text notifikace"
});
```

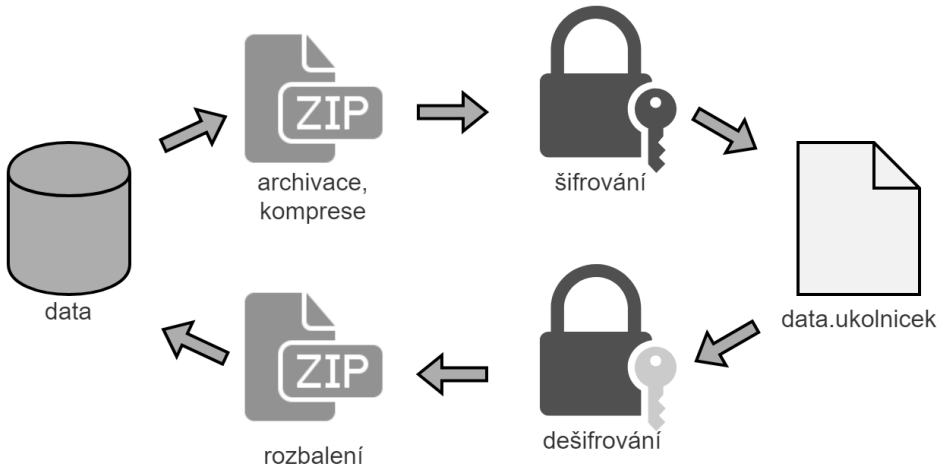
4.7 Hlavní proces aplikace po vzoru singleton

Framework *Electron* umožňuje využít metodu, která zajistí spuštění maximálně jedné instance naší aplikace. Metoda makeSingleInstance je volána skrze modul app z hlavního procesu, který mimo jiné zajišťuje řízení životního cyklu aplikace. Uvnitř této metody je, v případě opětovného spuštění, aplikace přepnuta do popředí a sekundární instance je vždy ukončena. Použití může vypadat následovně:

```
const app = electron.app;

let shouldQuit = app.makeSingleInstance((cmdLine, wrkDirectory) => {
  if (mainWindow) {
    if (mainWindow.isMinimized()) {
      restoreMinimized();
    }
    mainWindow.focus();
  }
});

if (shouldQuit) {
  app.quit();
}
```

4.8 Import a export

Při umožnění exportování dat, je nutné, aby byla zajištěna bezpečnost těchto dat při jejich přenášení. To je zajištěno následujícím způsobem. Nejprve jsou veškerá data archivována a následně komprimována do vyrovnávací paměti, balíčkem zip-local. Data v paměti jsou následně zašifrována heslem od uživatele společně se symetrickým šifrovacím algoritmem AES256-CTR. Výsledný soubor s koncovkou ".ukolnicek" je tedy nečitelný, a bez znalosti hesla uživatele není možné data přečíst. Následující schéma znázorňuje jednotlivé kroky importování a exportování dat.



Obrázek 14: Import a export dat

Kryptografická funkcionalita pro šifrování a dešifrování je poskytována node.js modulem crypto, jehož využití je možné vidět v následující ukázce:

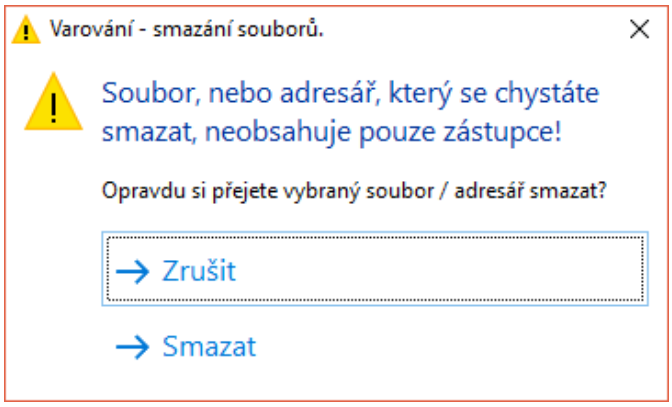
```
const crypt = require("crypto");
const algorithm = "aes-256-ctr";

function encrypt (buffer, password) {
   const cipher = crypt.createCipher(algorithm, password);
   const crypted = Buffer.concat([cipher.update(buffer), cipher.final()]);
   return crypted;
}

function decrypt(buffer, password) {
   const decipher = crypt.createDecipher(algorithm, password);
   const dec = Buffer.concat([decipher.update(buffer), decipher.final()]);
   return dec;
}
```

5 APLIKAČNÍ MANUÁL

V této části bude představeno samotné ovládání aplikace. V následujícím obrázku jsou znázorněny základní funkce aplikace.

Obrázek 15: Ovládání - základní funkce

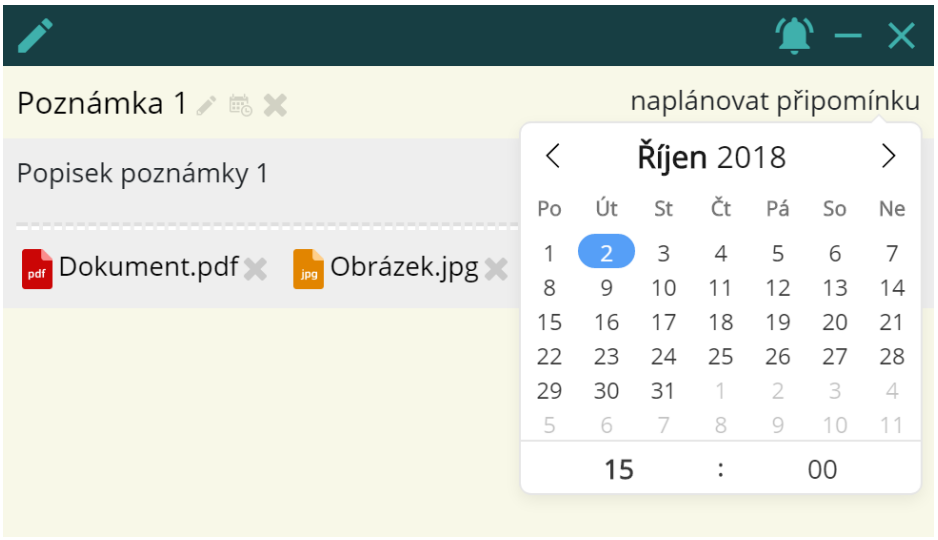
Ikona tužky (značení č. 1) slouží pro vytvoření nové poznámky. Vedle každé poznámky jsou nalezeny její ovládací ikony (značení č. 2). Ty slouží pro editaci, zapnutí/vypnutí naplánování, a smazání poznámky. Popisek pod názvem poznámky (značení č. 3), lze editovat po kliknutí na něj. Pod tímto popiskem je umístěn seznam veškerých souborů poznámky (značení č. 4). Každý soubor je možné smazat pomocí křížku vedle jeho názvu, po stisknutí se objeví dialog zobrazený v obrázku 16. Klíčové ovládací prvky aplikace jsou umístěny na hlavní liště (značení č. 5). Mezi tyto funkce patří aktivace/deaktivace upozornění, minimalizace a ukončení aplikace. Po zapnutí funkce naplánování, u jednotlivých poznámek, je umožněno naplánovat upozornění (značení č. 6).



Obrázek 16: Ovládání - mazání souboru

5.1 Plánování připomínky

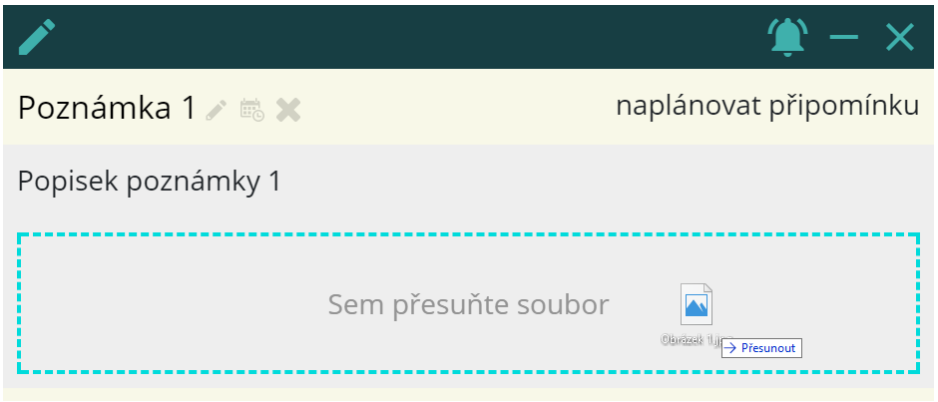
Naplánování připomínku je možné provést pro každou jednotlivou poznámku. Nejprve je nutné zapnout možnost naplánování, pomocí ikony kalendáře, vedle názvu poznámky. Po aktivaci se objeví nápis "naplánovat připomínku". Dialog, který se objeví po kliknutí na tento nápis, je zobrazen na obrázku 17. V dialogu lze naplánovat upozornění pro určitý čas ve vybraný den v roce.



Obrázek 17: Ovládání - naplánování připomínky

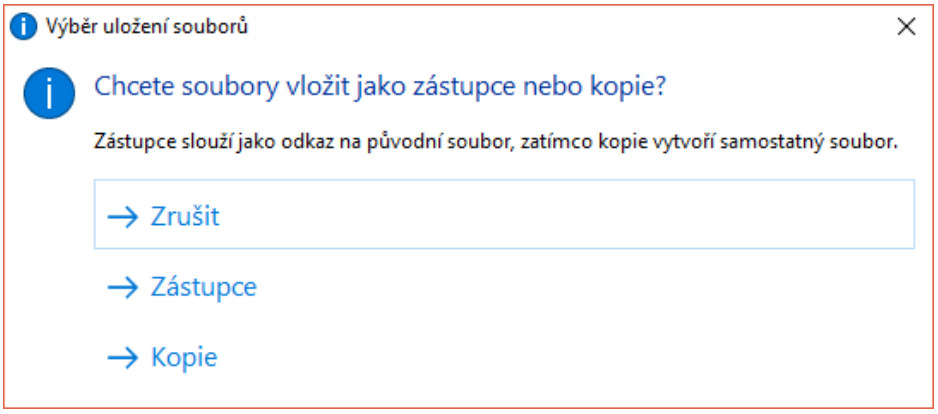
5.2 Vložení souborů

Po otevření poznámky, pomocí kliknutí na její název, může uživatel vložit jeden či více souborů a jednoduše je přiřadit poznámce. K vložení nejsou využity dialogy, nýbrž intuitivní drag & drop funkcionalita jak je možné pozorovat na obrázku níže.



Obrázek 18: Ovládání - drag & drop

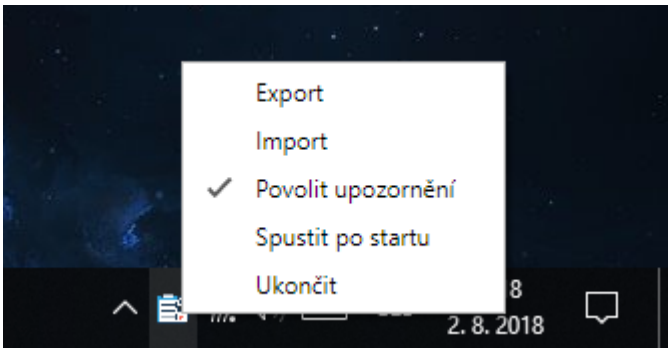
Soubory jsou následně dle volby zpracovány. Pro jednotlivé zástupce jsou pouze ukládány cesty k těmto souborům, a po smazání původních souborů už není možné je otevřít. Kopie je možné plnohodnotně otevírat bez původního souboru. Výběr uložení, po přesunutí souborů do definované oblasti, se zobrazí jako dialog v následujícím obrázku.



Obrázek 19: Ovládání - vložení souboru

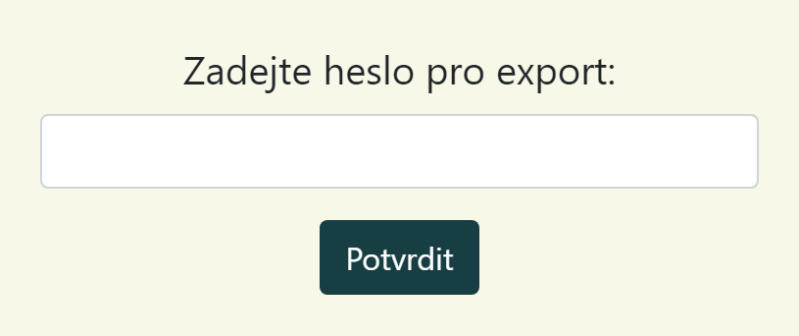
5.3 Doplňkové funkce

Funkce, které nejsou využívány příliš často, byly skryty do kontextového menu ikony aplikace v oznamovací oblasti systému. Vzhled menu je možné vidět na obrázku 20 Jedná se o funkce import, export, spuštění po startu a povolení upozornění, která zastává stejnou funkci jako první ikona v oblasti značení č. 5 na obrázku 15.



Obrázek 20: Ovládání - kontextové menu

Po volání funkcí import a export jsou volány standardní dialogová okna pro výběr umístění souboru. Po definování správné cesty k souboru, je uživatel dotázán na heslo, které je použito k zašifrování nebo dešifrování souboru v závislosti na vybrané funkci. Následující obrázek zobrazuje jednoduché dialogové okno pro zadání hesla.



Obrázek 21: Ovládání - dialog pro heslo při exportu

ZÁVĚR

První část práce zahrnuje seznámení s klasickými frameworky pro vývoj nativních aplikací. Jedná se konkrétně o framework .NET pro platformu *Windows*, a framework *Cocoa* pocházející z prostředí *MacOS*. Další kapitola obsahuje úvod do problematiky vývoje na více platforem. V této části jsou představeny různé přístupy společně s jejich výhodami a nevýhodami. Následuje představení vybraných existujících frameworků. U sledovaných frameworků byly podrobně vysvětleny jejich využívané technologie, programovací jazyky, architektury, cílové platformy a vývojová prostředí. V neposlední části této kapitoly jsou odhaleny odlišnosti cross-platform frameworků od těch klasických. V závěru této části jsou vybrané multiplatformní frameworky porovnány dle zvolených specifik. Třetí kapitola se zabývá představením vyvíjené aplikace společně se specifikací požadavků a vlastním návrhem. Následující kapitola je věnována implementaci navržené aplikace ve zvoleném frameworku *Electron*. Obsaženy jsou hlavně zajímavé funkce a neobvyklé aspekty frameworku. V závěru práce se nachází grafický aplikační manuál aplikace.

Pro multiplatformní frameworky je nutné se naučit nové principy vývoje a případně nový jazyk například u frameworku *Haxe*. Pokud má však vývojář zkušenosti s použitím webových technologií, tak by mu nemělo dělat potíže osvojit si tyto nové způsoby. Navíc zvyšují znovupoužitelnost veškerého kódu, protože je možné použít již napsané webové aplikace pro vytvoření těch desktopových. Webové technologie také umožňují využít širokou škálu knihoven a frameworků, které mohou nyní být využity pro cross-platformové desktopové aplikace. *Electron* a *NW.js* jsou tedy vhodné, pokud budou pro vývoj využity jakékoliv knihovny a frameworky napsané v jazyce *JavaScript*, mezi které patří např. React, jQuery, Angular a další. Pro vytváření uživatelského rozhraní využívají frameworky nejčastěji čisté *HTML* a *CSS*. Oproti tomu *JavaFX* na platformě Java disponuje popisným jazykem FXML, který umožňuje rozdělit aplikační logiku od uživatelského rozhraní. Díky tomu lze využít například nástroje Scene Builder pro vizuální tvorbu GUI. Ten umožní vývojáři navrhnout vzhled aplikace bez psaní jakéhokoliv kódu. Výsledné aplikace na platformě *Java* však pro svůj běh vyžadují běhové prostředí *Java Runtime Environment*, oproti tomu se aplikace z cross-platformových frameworků tváří nativně.

V práci se úspěšně podařilo vyvinout aplikaci pomocí multiplatformního frameworku *Electron*, i bez větších předchozích zkušeností se skriptovacím jazykem *JavaScript*. Funkcionalita aplikace byla otestována na systémech *Windows* a *Ubuntu*. Na obou systémech fungovala totožně a bez problémů. Všechny stanovené cíle práce tím byly tedy splněny.

Aplikace by mohla být rozšířena o formátované vnitřní struktury úkolů např. zaškrtávání splněných úkolů, formátovaný text, odrážky v textu apod. Další rozšíření by mohlo být cíleno na možnosti přizpůsobení vzhledu aplikace dle uživatele.

POUŽITÁ LITERATURA

[1] ARLOW, Jim a Ila NEUSTADT. *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky*. 2., aktualiz. a dopl. vyd. Brno: Computer Press, 2007. ISBN 978-80-251-1503-9.

[2] BENOIT, Alessandro. *NW.js Essentials: Build native desktop applications for Windows, MacOS, or Linux using the latest web technologies*. Packt Publishing, 2015. ISBN 978-1-78528-086-3.

[3] FAIN, Yakov. *Java programming 24-hour trainer*. Hoboken, IN: John Wiley, 2011. ISBN 978-0470889640.

[4] HILLEGASS, Aaron a Adam PREBLE. *Cocoa programming for MacOS X*. 4th ed. Upper Saddle River, NJ: Addison-Wesley, c2012. ISBN 978-0-321-77408-8.

[5] JENSEN, Paul B. *Cross-platform desktop applications: using Electron and NW.js*. Shelter Island, NY: Manning Publcations Co., 2017. ISBN 978-1617292842.

[6] MANNING, Jon, Paris BUTTFIELD-ADDISON a Tim NUGENT. *Swift development with Cocoa*. Sebastopol, CA: O'Reilly Media, 2014. ISBN 978-1-491-90894-5.

[7] MCCURDY, Jeremy. *Haxe Game Development Essentials: Create games on multiple platforms from a single codebase using Haxe and the HaxeFlixel engine*. Packt Publishing, 2015. ISBN 978-1-78528-978-1.

[8] OWENS, Michael *The definitive guide to SQLite*. New ed. Berkeley, Calif: Apress, 2006. ISBN 978-1-59059-673-9.

[9] PIPER, Ian. *Learn Xcode tools for MAC OS X and iPhone development*. New York: distributed by Springer-Verlag, c2009. ISBN 978-1-4302-7220-5.

[10] RICHTER, Jeffrey. *Applied Microsoft.NET framework programming*. Redmond, Wash.: Microsoft Press, c2002. ISBN 0-7356-1422-9.

[11] THAI, Thuan L. a Hoang Q. LAM. *.NET framework essentials*. 3nd ed. Sebastol, CA: O'Reilly, c2003. ISBN 0-596-00505-9.

[12] Build and debug cross platform applications using Haxe - HaxeDevelop [online]. Haxe Foundation, c2018 [cit. 2018-05-12]. Dostupné z: https://haxedevelop.org

[13] Electron website - Build cross platform desktop apps with JavaScript, HTML, and CSS [online]. [cit. 2018-05-12]. Dostupné z: https://electronjs.org

[14] Haxe - The Cross-platform Toolkit [online]. Haxe Foundation, c2018 [cit. 2018-05-12]. Dostupné z: https://haxe.org/#learn-more

[15] Java Platform, Standard Edition (Java SE) 8: *Client Technologies: Java Platform* [online]. c2016 [cit. 2018-05-12]. Dostupné z: https://docs.oracle.com/javase/8/javase-clienttechnologies.htm

[16] Java Powers Our Digital World: *Java Resources for Students, Hobbyists and More* [online]. Oracle, c1995-2018 [cit. 2018-05-12]. Dostupné z: https://go.java/index.html

[17] NW.js [online]. NW.js community, c2015-2018 [cit. 2018-05-12]. Dostupné z: https://nwjs.io/