Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Multiplatformní čtečka elektronických knih se synchronizací

Diplomová práce

Autor: Bc. Jan Bareš

Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Pavel Kříž, Ph.D.

Hradec Králové duben 2018

Prohlášení:

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a uvedl jsem všechny použité prameny a literaturu.

V Hradci Králové dne 15. dubna 2018 Bc. Jan Bareš

Poděkování:

Rád bych poděkoval vedoucímu diplomové práce Ing. Pavlu Křížovi, Ph.D. za metodické vedení práce, cenné rady a připomínky. Děkuji také rodině, přátelům a kolegům za vstřícnost a morální podporu.

Anotace

Práce se zabývá pojmem elektronická kniha a jejími formáty. Dále je zmíněno několik čteček těchto knih, které jsou dostupné pro platformy Android a Windows. Je zde zkoumána možnost použití nástroje Xamarin k vytvoření multiplatformní čtečky e-booků a využití podpory více platforem k synchronizaci mezi zařízeními.

V praktické části této práce byla taková aplikace pro platformy Android a Windows vytvořena. Čtenáři umožňuje synchronizaci postupu čtení mezi jeho zařízeními. Čtečka dále nabízí běžné funkce, jako přidávání záložek, změnu velikosti textu, noční režim a další. Synchronizace je podporována pomocí Dropboxu nebo s využitím cloudové databáze Firebase. Hotová aplikace byla umístěna do Google Play a Microsoft Store.

Annotation

Title: Cross platform e-book reader with synchronization

The thesis focuses on electronic books and their formats. It mentions several e-book readers that are available for Android and Windows platforms. Next, the thesis explores possibilities to create a cross platform e-book reader with Xamarin tool and the use of multiple platform support to sync between devices.

In the practical part of the thesis, this app for Android and Windows was created. It allows readers to synchronize the reading progress between their devices. The e-book reader also offers common features such as bookmarking, text size change, night mode, etc. Synchronization is provided by Dropbox or the Firebase cloud database. The completed app was published to Google Play and the Microsoft Store.

Obsah

Seznam obrázků

Seznam ukázek kódu

Použité zkratky a pojmy

C# C# je moderní objektový programovací jazyk od společnosti Microsoft. Použitelný je k vývoji webových, mobilních, desktopových a dalších aplikací.

CSS CSS je jazyk, pomocí kterého lze nastavovat vzhled HTML stránky. Je možné měnit různé vlastnosti, například barvu textu, velikosti elementů či používat pokročilé funkce, jako automatické rozdělení textu do sloupců.

GUI Grafické uživatelské rozhraní aplikace.

HTML HTML je značkovací jazyk, který popisuje strukturu webové stránky nebo jiného dokumentu, například elektronické knihy.

IoC IoC neboli *Inversion of Control* je programovací paradigma, které řeší předávání závislostí mezi jednotlivými částmi kódu.

UWP UWP neboli Universal Windows Platform je architektura aplikací vytvořená společností Microsoft. Jejím cílem je, aby je bylo možné spouštět stejné aplikace na zařízeních s Windows 10, Windows 10 Mobile, Xbox One nebo Hololens.

XAML XAML je značkovací jazyk založený na XML, který slouží k popisu grafického rozhraní aplikací postavených na platformě .NET.

XML XML je značkovací jazyk sloužící k popisu dat a jejich struktury.

ZIP ZIP je rozšířený souborový formát sloužící ke kompresi dat. Adresář může být po zabalení distribuován jako jeden ZIP soubor, který může být zase rozbalen.

1 Úvod

Elektronické knihy jsou v dnešní době velký fenomén. Pro některé čtenáře je jejich čtení pohodlnější než čtení klasických tištěných knih. Tyto knihy lze číst na různých zařízeních, ať už jsou to mobilní telefony, tablety, počítače či speciální čtečky. Na trhu ovšem není mnoho dostupných možností, jak číst elektronickou knihu na více zařízeních tak, aby se postup čtení automaticky synchronizoval.

Cílem této práce je ověřit možnost použití nástroje pro tvorbu **multiplatformních aplikací** k vytvoření **čtečky elektronických knih** a využít podpory více platforem k **synchronizaci postupu čtení** mezi jednotlivými zařízeními. Záměrem je vytvořit funkční a jednoduchou aplikaci, která bude sloužit k pohodlnému čtení elektronických knih, bude synchronizovat postup čtení a bude nabízet běžné funkce, jako například přidávání záložek, změnu velikosti textu a podobně.

Druhá kapitola (2) volně navazuje na úvod a zabývá se pojmem elektronická kniha a formáty, které se k její distribuci běžně používají. Největší pozornost je věnována formátu EPUB. Ve **třetí kapitole** (3) jsou představeny vybrané existující čtečky elektronických knih. U aplikací, které mají otevřený zdrojový kód, bylo zkoumáno, jakým způsobem elektronické knihy zobrazují. **Čtvrtá kapitola** (4) analyzuje způsob, jak vytvořit vlastní čtečku elektronických knih. Jsou zde stanoveny požadavky pro tvorbu takové aplikace. V **páté kapitole** (5) je čtenář seznámen s nejdůležitějšími technologiemi a nástroji, které byly použity při tvorbě této aplikace. **Šestá kapitola** (6) se zaměřuje na popis implementace vlastního řešení čtečky elektronických knih. V **sedmé kapitole** (7) je zhodnoceno dodržení požadavků na aplikaci stanovených v analýze. Dále je zde uživatelský návod k použití vytvořené aplikace. **Osmá kapitola** (8) shrnuje výsledky práce a nastiňuje možné způsoby budoucího rozvoje aplikace.

2 Elektronická kniha a její formáty

2.1 Elektronická kniha

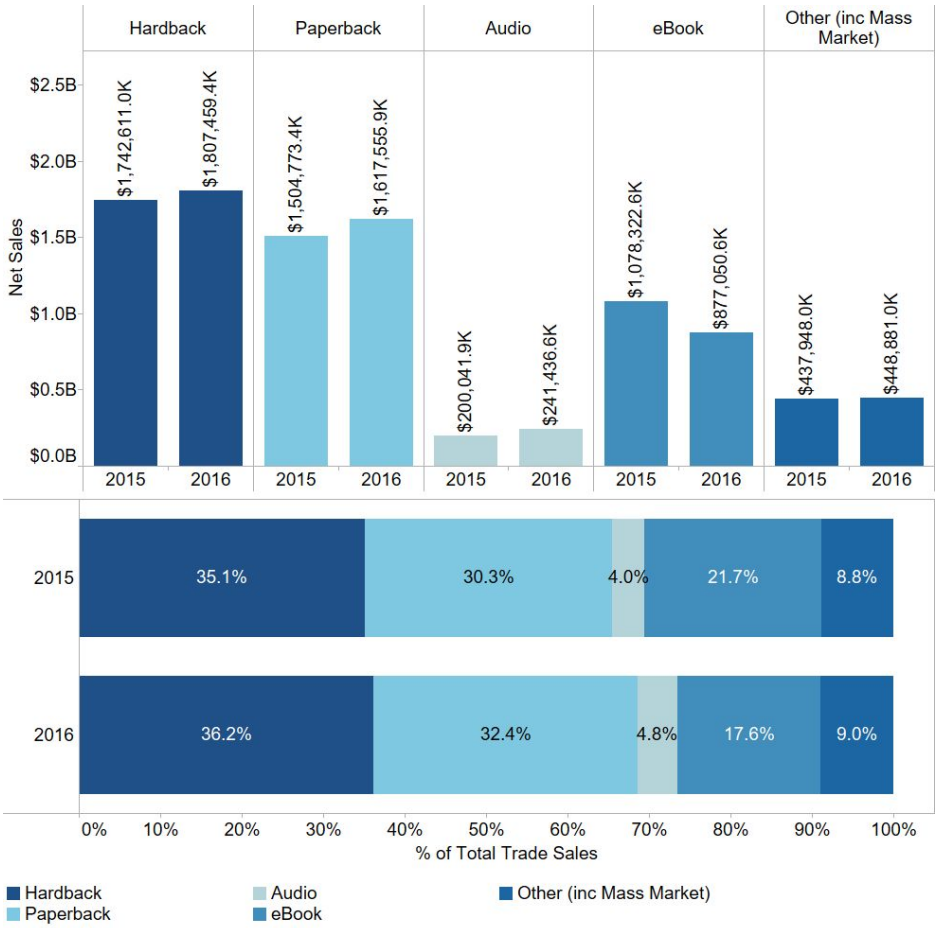
Elektronická kniha (anglicky e-book[1]) je digitální obdobou tištěné knihy. Je uložena v souboru, který má speciální formát, sloužící k distribuci těchto knih. Tyto soubory lze jednoduše šířit, ovšem k jejich otevření je zapotřebí čtečka. Může se jednat o mobilní telefon či tablet, které mají nainstalovanou potřebnou aplikaci. Je také možné použít počítač nebo speciální čtečku typu Amazon Kindle.

Podíl elektronických knih oproti knihám tištěným je na světovém trhu klesající, ale přesto v prvních třech čtvrtletích roku 2016 tvořil jejich prodej 17,6 % [1]. V porovnání s rokem 2015 se jedná o pokles zhruba o 18 % (viz obrázek 2.1) [1]. Podle Jiřího Vlčka [2], zakladatele českého obchodu s e-knihami Palmknihy.cz, za problémy s klesajícím prodejem elektronických knih stojí jejich cena. Ta bývá stejná nebo vyšší než u knihy tištěné a to přestože zakoupenou elektronickou knihu nelze dle legislativy nikomu zapůjčit, ani ji znovu prodat.

Možnost jednoduchého šíření elektronických knih může být pro vydavatele problém. Knihy lze umístit na internet a kdokoliv si je může stáhnout, aniž by za ně zaplatil. Proto součástí elektronických knih bývá **ochrana DRM**[2]. Ta má za cíl zabránit nelegálnímu šíření elektronických knih. Existují dvě základní podoby ochrany DRM. První z nich je sociální DRM. Do knihy je vložena jednoznačná identifikace kupujícího a transakce, ve které byla kniha zakoupena. Pokud čtenář tuto knihu začne nelegálně šířit, je možné ho jednoznačně dohledat. Silnější ochranou je Adobe DRM. Zakoupená kniha je svázána s konkrétním čtenářem a lze ji otevřít pouze na autorizovaném zařízení. Kdyby se ji tedy čtenář snažil šířit dál, nikdo jiný si tuto knihu nemůže otevřít. [3]

[1] Je možné používat varianty e-book, ebook či eBook.

[2] Digital Rights Management



Obrázek 2.1: Podíl elektronických knih na trhu v prvních třech čtvrtletích roku 2016 [1]

Pro čtenáře představují elektronické knihy oproti tištěným několik výhod. Lze si je uložit do mobilního telefonu, tabletu či speciální čtečky a nosit tak v kapse tisíce knih. Čtenář může kdykoliv a kdekoliv pokračovat ve čtení a nemusí si pamatovat, kde naposled skončil. Čtečka si jeho poslední pozici pamatuje místo něho. Displej čtečky či mobilního telefonu je podsvícený a je tedy možné číst i v noci bez rozsvíceného světla. Výhodou pro čtenáře s horším zrakem je možnost nastavit si velikost textu, která mu vyhovuje.

Elektronické knihy mají také své nevýhody. Existuje několik formátů, ve kterých se mohou distribuovat, a ne každá čtečka dokáže zobrazit všechny. Při koupi elektronické knihy je tedy nutné zkontrolovat, že její formát dokáže daná čtečka přečíst a zobrazit. Kniha se také může na každé čtečce zobrazit trochu jinak a ne vždy to může být v souladu se záměrem vydavatele knihy.

2.2 Formát EPUB

EPUB je dnes jedním z nejrozšířenějších formátů, ve kterém jsou elektronické knihy distribuovány. Jedná se o **otevřený formát** za kterým nejprve stála organizace IDPF[3], která se starala o jeho rozvoj. V lednu 2017 byla sloučena s organizací W3C[4], která udržuje pro EPUB klíčové technologie HTML a CSS [5].

Formát EPUB byl poprvé standardizován v roce 2007 jako nástupce formátu OEB[5]. V roce 2010 vyšla verze 2.0.1, která je posledním zástupcem druhé generace. Roku 2011 vyšla verze 3, která byla roku 2014 následována verzí 3.0.1. V lednu 2017 vyšla zatím poslední verze 3.1. [4].

2.2.1 Struktura formátu EPUB

Specifikace formátu EPUB je velmi obsáhlá a zde je popsán pouze její nejdůležitější výtah. Formát EPUB je **speciální adresářová struktura** zabalená do ZIP souboru. Struktura formátu EPUB je definována pomocí tří otevřených standardů: Open Publication Structure (OPS)[6], Open Packaging Format (OPF)[7] a Open Container Format (OCF)[8] [14]. Pomocí nich je určeno, jakým způsobem jsou definovány metadata knihy, navigace a další. Samotný text knihy je uložen v HTML souborech.

Každý EPUB musí obsahovat soubor `META-INF/container.xml`. Jedná se o XML soubor, ve kterém je uvedena cesta k OPF souboru, kde jsou uloženy všechny informace o knize, jako název nebo autor či seznam všech souborů a kapitol v knize. Ten se obvykle jmenuje `content.opf` a také se jedná o XML soubor, který má jeden kořenový element `package`. Tento element nastavuje pro své potomky jmenný prostor *http://www.idpf.org/2007/opf*. Dále má atribut `version`, pomocí kterého je definována použitá verze formátu EPUB. Obsahuje tři povinné potomky, jsou to `metadata`, `manifest` a `spine`.

V elementu `metadata` jsou uvedeny **meta informace o knize**. Princip je podobný jako metadata v HTML stránce. Jedná se o elementy, které v atributech obsahují název meta informace a její hodnotu. Pro některé potomky je použit jmenný prostor

[3] International Digital Publishing Forum
[4] World Wide Web Consortium
[5] Open eBook Publication Structure
[6] http://www.idpf.org/epub/20/spec/OPS_2.0.1_draft.htm
[7] http://www.idpf.org/epub/20/spec/OPF_2.0.1_draft.htm
[8] http://www.idpf.org/doc_library/epub/OCF_2.0.1_draft.doc

dublin core[9]. Ukázka zápisu těchto metadat je ve zdrojovém kódu 2.1.

```
1 <dc:title>Pán prstenů. Společenstvo Prstenu</dc:title>
```

Ukázka kódu 2.1: Ukázka meta informace obsahující název knihy

Další součástí souboru `content.opf` je `manifest`. Jedná se o **seznam souborů**, které jsou v EPUBu obsaženy. Jednotlivé položky jsou reprezentovány pomocí elementu `item`, který má další atributy. Prvním z nich je `id`, který představuje unikátní identifikátor daného souboru. Pomocí něho se lze na soubor odkázat například ze sekce `spine`. Dalším atributem je `href`, který určuje cestu k souboru. Ta je relativní vůči umístění `content.opf`. Posledním atributem je `media-type`, pomocí kterého je definován typ souboru. Může se jednat o HTML soubor, obrázek, CSS a jiné. Na pořadí elementů v rámci `manifest` nijak nezáleží a nejde pomocí něho nastavit pořadí kapitol v knize. Jedná se pouze o seznam souborů, který kromě jednotlivých kapitol obsahuje například i obrázky a podobně.

Posledním elementem v rámci souboru `content.opf` je `spine`. Pomocí tohoto elementu je možné nastavit **pořadí kapitol** v knize. Obsahuje potomky `itemref`, které se pomocí atributu `idref` odkazují na jednotlivé soubory ze sekce `manifest`. Pořadí těchto elementů určuje pořadí kapitol v knize. Element `spine` také může obsahovat atribut `toc`, který obsahuje identifikátor souboru, ve kterém je definována navigace v knize.

2.2.2 Navigace v knize

V rámci formátu EPUB je rozdíl mezi pořadím souborů v knize a navigací. Pořadí souborů pouze určuje, jak na sebe navazují, a v tomto pořadí by je také čtečka měla zobrazit. Oproti tomu navigace je další vrstva, která definuje **vlastní seznam kapitol** a jejich pořadí. Tento seznam může být zobrazen čtenáři a ten se pomocí něho může navigovat v knize. V této navigaci nemusí být zahrnuty všechny soubory. Příkladem může být kniha, která v prvních souborech obsahuje titulní obrázek, impresum[10] a podobně. V navigaci budou zahrnuty vlastní kapitoly textu, které jsou v následujících souborech. Když si tedy čtenář otevře tuto elektronickou knihu, čtečka nejprve zobrazí soubor s titulním obrázkem, ale pokud si čtenář zobrazí navigaci, uvidí zde pouze kapitoly textu a může se na ně přesouvat.

[9] http://purl.org/dc/elements/1.1/

[10] Vydavatelský záznam o knize obsahující copyrightové informace

K zapsání navigace se používá soubor NCX[11]. Jedná se o XML soubor s definovanou strukturou, který popisuje hierarchickou navigaci v rámci knihy. Hlavním elementem je `navMap`, který obsahuje elementy `navPoint`. Tyto představují položku navigace. V elementu `navLabel` mají nastavený popisek, který se v navigaci zobrazí, a v elementu `content` je cesta k souboru, který je s touto položkou svázán. Mohou obsahovat další elementy `navPoint` a díky tomu může vzniknout hierarchická navigace.

2.3 Další formáty

EPUB není jediným dostupným formátem pro distribuci elektronických knih. V českých obchodech[12] lze běžně zakoupit i další formáty, například PDF či MOBI.

2.3.1 PDF

Portable Document Format neboli PDF byl vytvořen společností Adobe v roce 1993. Na začátku roku 2008 byl tento formát standardizován organizací ISO[13] a stal se standardem ISO 32000-1 [11]. Formát PDF je spjat zejména s programem Adobe Acrobat Reader, což je oficiální software pro zobrazení tohoto formátu. V dnešní době není problém soubor ve formátu PDF otevřít v některém z alternativních prohlížečů, například Foxit Reader[14]. Existuje také implementace v jazyce JavaScript[15]. Soubory PDF lze také běžně otevírat i v internetovém prohlížeči.

PDF je multiplatformní formát a jeho cílem je, aby se obsah zobrazil na každém zařízení ve stejné podobě. Mimo jiné je možné v něm vytvářet formuláře, 3D objekty nebo video či zvuk.

2.3.2 MOBI

Formát MOBI byl původně vyvinut pro čtečku MobiPocket. Společnost Amazon si na jeho základě vytvořila vlastní formát AZW, který používá na svých čtečkách Amazon Kindle. Formát AZW se oproti MOBI liší zejména použitým DRM [12]. Podobně jako EPUB také MOBI obsahuje OPF soubor, ve kterém jsou uvedeny

[11] Navigation Center eXtended
[12] Na serveru Alza.cz, eReading.cz a Palmknihy.cz
[13] International Organization for Standardization
[14] https://www.foxitsoftware.com/pdf-reader/
[15] https://mozilla.github.io/pdf.js/

metadata knihy a seznam kapitol.

Text knihy je uložen ve formátu HTML, pro renderování knihy lze tedy využít podobný přístup, jako u formátu EPUB. MOBI nabízí oproti formátu EPUB několik pokročilých funkcí. Například je možné mít jeden obrázek v několika rozlišeních, a čtečka si sama vybere, která varianta je pro ni nejlepší. MOBI také nabízí vysokou míru komprese obsahu, aby nebyl soubor v tomto formátu příliš velký.

3 Přehled dostupných čteček

Pro platformu Windows a Android existuje několik čteček elektronických knih. Ne všechny jsou podporovány na obou platformách a nenabízí proto synchronizaci postupu čtení. V této kapitole je popsáno jen několik z nich.

3.1 Knihy Google Play

Velmi používaná čtečka[1] je Knihy Google Play[2]. Jedná se o produkt společnosti Google, která provozuje vlastní obchod s elektronickými knihami a k nim dodává i vlastni čtečku. Ta slouží primárně pro čtení zakoupených knih ve vlastním obchodě, ale lze do ní pomocí webového rozhraní naimportovat vlastní soubory.

Knihy Google Play lze používat na mobilní platformě Android (také vyvíjena společností Google), iOS a v podobě webové aplikace. Nativní aplikace pro platformu Windows však chybí. Mezi aplikacemi na podporovaných platformách probíhá synchronizace postupu čtení.

Aplikace nemá otevřené zdrojové kódy, takže není možné zjistit, jakým způsobem pracuje a zobrazuje knihy.

3.2 Amazon Kindle

Společnost Amazon, která původně začínala jako internetové knihkupectví, dnes mimo jiné prodává elektronické knihy. Pro tyto knihy má vlastní čtečku Amazon Kindle. Jedná se o aplikaci pro platformu Android a iOS, ale také o hardwarovou čtečku. Uživatel má svoje knihy mezi aplikací a hardwarovou čtečkou synchronizovány.

[1] Mobilní aplikace má 1 000 000 000–5 000 000 000 instalací

[2] https://play.google.com/store/apps/details?id=com.google.android.apps.books

Amazon Kindle je primárně určen pro čtení knih zakoupených na serveru Amazon Kindle Store a nikoliv na čtení vlastních knih.

Vzhledem k tomu, že se nejedná o aplikaci s otevřeným zdrojovým kódem, není možné zjistit, jakým způsobem aplikace interně pracuje.

3.3 eReading

Jeden z českých prodejců elektronických knih eReading.cz má svoji vlastní čtečku pro platformu Android[3]. Dříve bylo možné si na tomto portálu knihy nejen kupovat, ale také půjčovat, a vypůjčené knihy bylo možné otevřít pouze v této aplikaci. V současné době byla možnost výpůjček omezena pouze na zapůjčení knihy skrze veřejnou knihovnu.

V aplikaci (obrázek 3.1) jdou otevřít pouze knihy zakoupené nebo vypůjčené na portálu eReading. Aplikace umožňuje měnit velikost textu, odsazení, barvu písma a pozadí. Dále aplikace umí přidávat záložky a vyhledávat v textu. Nejedná se o aplikaci s otevřeným zdrojovým kódem a proto není možné zjistit, jakým způsobem byla vytvořena a na jakém principu funguje.

3.4 Moon+

Další aplikací je Moon+[4]. Ta je dostupná pouze pro platformu Android. Podporuje synchronizaci postupu čtení pomocí Dropboxu, ale pouze mezi zařízeními s operačním systémem Android.

Tato aplikace existuje ve verzi zdarma, která obsahuje reklamy. Lze zakoupit i placenou verzi, která kromě odstranění reklam nabízí například podporu PDF nebo možnost přidat knihu na plochu zařízení.

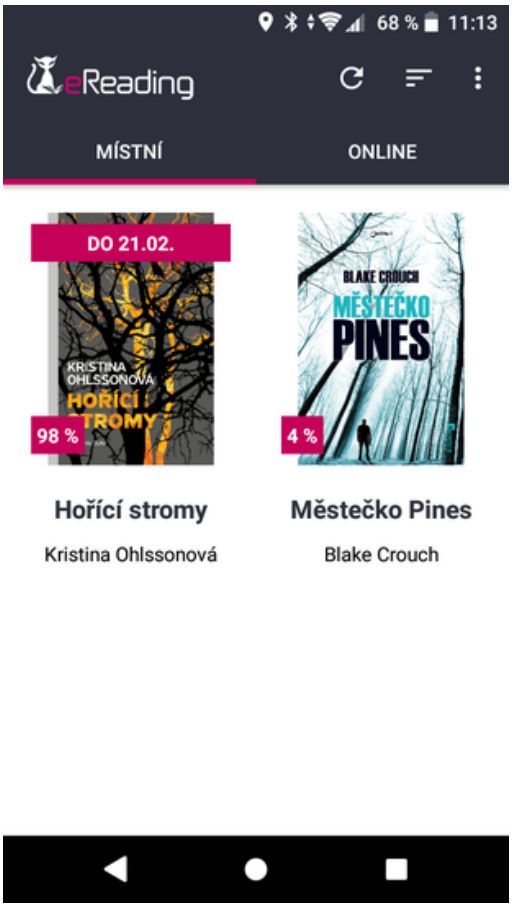
Aplikace nemá otevřený zdrojový kód a proto není možné zjistit, jakým způsobem vnitřně funguje.

3.5 Cool Reader

Další aplikací pro čtení elektronických knih je Cool Reader. Kromě formátu EPUB umí zobrazit také MOBI, DOC a další. Jedná se o aplikaci pro Windows (Win32

[3] https://play.google.com/store/apps/details?id=cz.ereading.android

[4] https://play.google.com/store/apps/details?id=com.flyersoft.moonreader



Obrázek 3.1: Hlavní stránka aplikace eReading

aplikce) a pro Android[5]. Cool Reader obsahuje velkou spoustu uživatelského nastavení, ale neobsahuje žádnou možnost synchronizace postupu čtení mezi zařízeními čtenáře.

Tato aplikace má otevřený zdrojový kód. Verze pro Windows je napsána pomocí jazyka C++ a dle dostupných zdrojových kódů na serveru sourceforge.net nepoužívá žádné externí WebView, ale obsahuje vlastní parsování CSS[6] a HTML[7].

Aplikace v dnešní době není moc udržovaná a vyvíjená. Poslední změna v oficiálním repozitáři[8] Windows aplikace proběhla v březnu 2017 a nejnovější verze pro Android pochází z července 2015.

[5]https://play.google.com/store/apps/details?id=org.coolreader
[6]https://sourceforge.net/p/crengine/crengine/ci/master/tree/crengine/src/lvstsheet.cpp
[7]https://sourceforge.net/p/crengine/crengine/ci/master/tree/crengine/src/lvxml.cpp
[8]https://sourceforge.net/projects/crengine/

3.6 Microsoft Edge

Microsoft Edge není primárně čtečka elektronických knih, ale jedná se o internetový prohlížeč od společnosti Microsoft. Přišel spolu s operačním systémem Windows 10, jako nástupce staršího prohlížeče Internet Explorer. Do prohlížeče MS Edge byla přidána podpora pro otevírání souborů EPUB a lze v něm tak číst elektronické knihy. Podporovány jsou funkce jako zobrazení obsahu knihy, přidávání záložek nebo změna vzhledu knihy. MS Edge je kromě Windows dostupný také pro platformy Android a iOS a je možné mezi těmito platformami využívat synchronizaci.

Aplikace MS Edge nemá otevřený zdrojový kód a není tedy možné zjistit, jakým způsobem zpracovává a renderuje knihy. Vzhledem k tomu, že se jedná o internetový prohlížeč, lze se domnívat, že bude k renderování využívat jádro, které se používá i k zobrazení internetových stránek v tomto prohlížeči.

4 Analýza vlastního řešení

4.1 Požadavky na aplikaci

V rámci této práce by měla být vytvořena multiplatformní čtečka elektronických knih s podporou synchronizace postupu čtení. Základní myšlenka je taková, že čtenář začne číst knihu například na mobilním telefonu či tabletu a automaticky může ve čtení pokračovat ze stejného místa na počítači. Aplikace by také měla podporovat běžné funkce, jako nastavení zobrazení knihy, přidávání záložek a podobně. Z toho vychází následující požadavky na aplikaci.

Funkční požadavky:

- Podpora formátu EPUB a možnost rozšíření o další formáty
- Možnost přidávat záložky do textu
- Automatická synchronizace postupu čtení a záložek
- Podpora synchronizace pomocí více poskytovatelů, například Dropbox
- Uživatelské nastavení, kde půjde nastavit vzhled a ovládání čtečky
- Možnost synchronizaci zakázat či omezit pouze na WiFi

Non-funkční požadavky:

- Podpora alespoň jedné mobilní platformy a desktopu
- Aplikace bude mít otevřený zdrojový kód, který bude dostupný na GitHubu
- Dostupnost aplikace v obchodě s aplikacemi

4.2 Nativní nebo multiplatformní aplikace

Aplikace by měla být dostupná na více platformách. Je několik způsobů, jak toho docílit. Může být pro každou požadovanou platformu vytvořena vlastní **nativní aplikace** nebo jen jedna **multiplatformní**, která bude fungovat na všech.

4.2.1 Nativní aplikace

První přístup k tvorbě aplikací je nativní vývoj. Jedná se o aplikace, které jsou určeny pro jednu konkrétní platformu a které jsou vyvíjeny prostředky, které tato platforma poskytuje.

Aplikace může jednoduše využívat všechny možnosti, které daná platforma nabízí. Používá běžné ovládací prvky a dodržuje standardní chování platformy. Také je taková aplikace vytvořena ve stylu jednotného designu platformy, jako je například Material Design na Androidu.

Zásadní nevýhoda nativního řešení se ukáže ve chvíli, kdy má aplikace podporovat více platforem. V takovém případě je potřeba aplikaci se stejnou funkčností vyvinout a udržovat několikrát, dle počtu požadovaných platforem. Vývoj aplikace tak bude náročnější na čas a finance.

Zvolit cestu nativního vývoje by bylo vhodné, pokud by aplikace měla fungovat pouze na jedné platformě, pokud by měla využívat její specifické vlastnosti nebo pokud by byl zásadní důraz kladen na čistě nativní vzhled a chování aplikace.

4.2.2 Multiplatformní aplikace

Druhou možností je tvorba multiplatformní aplikace, která funguje na více platformách. Pokud chce vývojář podporovat více platforem, nemusí stejnou aplikaci vytvářet a udržovat několikrát, ale stačí pouze jednou. Většina kódu v aplikaci by měla být napsána na jednom místě a je třeba vyřešit zejména rozdíly v grafickém rozhraní na jednotlivých platformách.

Vývoj multiplatformní aplikace s sebou nese výzvy, které je potřeba vyřešit. Dle Petzolda [10, stránka 3–4] se jedná o:

- **Rozdílné principy GUI**: například jiný způsob práce s uživatelskou navigací na každé platformě

- **Rozdílné vývojářské nástroje**: pro vývoj Android aplikace se používá Android Studio a pro vývoj UWP aplikace Visual Studio
- **Rozdílné prvky GUI**: pro podobnou funkcionalitu jsou na každé platformě rozdílné prvky, například pro přepínač nabízí Android `Switch` a UWP aplikace `ToggleSwitch`
- **Rozdílné programovací jazyky**: Android používá Kotlin nebo Javu a UWP má C#

Výhodou multiplatformní aplikace je možnost spustit ji na více platformách. Vývojář nemusí vyvíjet a udržovat několik aplikací, ale pouze jednu. Vývoj jedné aplikace je rychlejší a levnější než vývoj několika. Navíc k vývoji multiplatformní aplikace není potřeba několik týmů vývojářů, kde by se každý z nich specializoval na jednu platformu. Stačí pouze jeden tým, který vytvoří jednu aplikaci.

Nevýhodou multiplatformní aplikace je, že nemusí umožňovat použití specifických možností každé platformy. Příkladem je *floating action button*, tedy plovoucí tlačítko, které je primárním ovládacím prvkem Material Designu a tedy platformy Android [15]. Některé nástroje pro tvorbu multiplatformních aplikací tento prvek nemusí znát a vývojář ho nemůže jednoduše použít. Konkrétně Xamarin neumí s tímto tlačítkem pracovat a pokud ho chce vývojář použít, musí si toto tlačítko od základů vytvořit a nebo najít nějaké hotové řešení od třetí strany. V tom případě je potřeba pohlídat, aby splňovalo všechny parametry popsané v dokumentaci Material Designu. Multiplatformní aplikace může být také pomalejší, jelikož k její tvorbě nejsou použity nástroje, které jsou na dané platformě nativně podporovány.

Mezi nástroje pro tvorbu multiplatformních aplikací patří Xamarin [1], React Native [2], Ionic[3], Flutter[4], Apache Cordova[5] a další.

Pro vývoj čtečky elektronických knih se zdá vhodné využít cestu multiplatformní aplikace. Není potřeba používat žádné specifické možnosti podporovaných platforem a pokud by bylo vytvořeno několik nativních aplikací, velká část kódu by se musela v těchto aplikacích opakovat, což by zvyšovalo časovou náročnost vývoje, ale také by to mohlo vést k chybám v implementaci na jednotlivých platformách.

[1] https://www.xamarin.com/
[2] https://facebook.github.io/react-native/
[3] https://ionicframework.com/
[4] https://flutter.io/
[5] https://cordova.apache.org/

4.3 Přístupy k vykreslení textu knihy

Text elektronické knihy může být uložen ve formátu HTML. To je případ například formátu EPUB, který je ve své podstatě pouze ZIP se speciální adresářovou strukturu, obsahující HTML soubory. Pro zobrazení textu knihy je tedy nutné tento **HTML obsah vyrenderovat**. Jsou dvě možností, jak toho docílit. První způsob je vytvořit si vlastní algoritmus, který bude číst a vykreslovat HTML. Druhou možností je k renderování HTML využít existující řešení v podobě WebView, které pracuje podobně jako internetový prohlížeč. Na vstupu má HTML dokument a jeho obsah správně vykreslí.

4.3.1 Vlastní vykreslení

Vlastní řešení vykreslení obsahu spočívá v přečtení zdrojové HTML struktury a její správné interpretaci. Tato struktura může být velmi složitá, a pokud má být podporována celá HTML specifikace, bude se jednat o poměrně komplexní řešení. Pro zjednodušení nemusí být podporována celá specifikace, ale pouze její část. Algoritmus může zpracovávat pouze některé elementy a ostatní mohou být ignorovány. Požadované elementy může vykreslit vlastním způsobem a tím je získána plná kontrola nad zobrazeným obsahem.

Kromě textu mohou být v knize i další prvky, jako jsou obrázky nebo tabulky. Z toho vychází nutnost algoritmus průběžně upravovat dle toho, jak se bude v čase rozrůstat specifikace HTML. Pokročilejší řešení by také mělo přečíst kaskádové styly, které jsou součástí HTML. Ty určují jeho podobu, například barvu nebo velikost textu.

Další výzvou je výsledný obsah **rozdělit na jednotlivé stránky** a umožnit čtenáři přechod mezi nimi. Dlouhý text by měl být správně rozdělen, aby na okraji stránek nezůstávala useknutá slova. Je potřeba rozhodnout, kolik textu se vejde na jednu stránku a v závislosti na tom ho rozdělit. Již to samo o sobě je nelehký úkol a pokud se navíc bude jednat o složitější HTML strukturu, řešení tohoto problému bude muset být velice komplexní.

Výhoda tohoto řešení je plná kontrola nad výslednou podobou zobrazeného obsahu. Jednotlivé prvky mohou být renderovány dle konkrétních požadavků aplikace a nežádoucí elementy, například JavaScript v elementu `script`, mohou být zcela ignorovány. Další výhodou je nezávislost na externím programu. Aplikace ručí za zob-

razení obsahu na jednotlivých zařízeních a nemělo by tedy dojít k tomu, že se kniha na různých zařízeních zobrazí rozdílně.

Zásadní nevýhodou tohoto řešení je jeho složitost. Jazyky HTML a CSS jsou velmi komplexní a můžou se průběžně měnit. Vytvořit algoritmus, který by je uměl správně přečíst a vykreslit, by byl velmi náročný úkol.

4.3.2 Použití externího vykreslovacího jádra

Použití existujícího řešení v podobě externího vykreslovacího jádra je druhou možností. V kontextu mobilních aplikací se komponenta s takovým jádrem nazývá WebView. V podstatě se jedná o internetový prohlížeč bez grafického rozhraní, který má na vstupu HTML a dokáže ho zobrazit.

Princip použití spočívá ve vytvoření vlastní HTML šablony obsahující CSS a JavaScript. Šablona je připravena tak, aby zobrazovala obsah ve sloupcích. To budou jednotlivé stránky, na které se jde pomocí JavaScriptu přesouvat. Vložený JavaScript může také ze stránky odstranit nežádoucí elementy a pomocí CSS lze změnit vykreslení požadovaných prvků. Do šablony se poté vloží obsah knihy a celé HTML se zobrazí ve WebView. Výsledek je stejný, jako kdyby se HTML zobrazilo v internetovém prohlížeči.

Výhodou použití tohoto řešení je jeho jednoduchost. Není potřeba číst a zpracovávat HTML a CSS, ale ty se pouze předají hotové komponentě, která je zobrazí. Zároveň v ní lze běžně spouštět i JavaScript, což by při použití vlastního řešení byl problém. Další výhodou je jednoduché rozdělení knihy na jednotlivé stránky. Stačí pouze použít několik CSS vlastností a samotné **rozdělení obsahu provede WebView**.

Nevýhodou tohoto řešení je závislost na externím vykreslovacím jádru. Záleží na cílovém zařízení, jaké jádro k vykreslení použije. Z toho vyplývá riziko, že se kniha zobrazí na každém zařízení rozdílně. Jedná se o podobný problém, jako při vývoji webových stránek, kde může mít také každý uživatel jiný internetový prohlížeč.

4.4 Získání pozice v textu

Další z problémů k vyřešení je způsob získání pozice v textu. Ta musí být obousměrná, tedy je možné zjistit aktuální pozici a zároveň je možné se na požadovanou pozici přesunout. Podmínkou je, aby byla získaná **pozice nezávislá na velikosti**

textu, velikosti displeje a dalších parametrech. Pozice vypočítaná na mobilním telefonu by měla ukazovat na stejnou část textu také na počítači a naopak. Takto získaná pozice je důležitá pro synchronizaci postupu čtení mezi zařízeními. Tato synchronizace je v podstatě pouze pozice v textu, která se pošle na jiné zařízení a tam se text na dané pozici zobrazí. Nabízí se několik způsobů, jak tuto pozici zjistit.

4.4.1 Aktuální stránka vyjádřená procentem

První možností je číslo aktuální stránky vyjádřené v procentech. Například pokud je čtenář na páté stránce z deseti (tedy na 50 %), při zvětšení velikosti textu a následném zvýšení počtu stránek např. na dvanáct by se měl opět dostat na 50 %, tedy na stránku šest. Tento způsob zaručuje jednoduchý převod mezi pozicí a číslem stránky.

Ve skutečnosti tato metoda nepočítá pozici v textu, ale pozici místa, kde se daný text vyskytuje. Tedy zmíněných 50 % neznamená přesně polovinu textu, ale stránku, na které se text zobrazuje. Zcela zde chybí informace o tom, jak jsou jednotlivé stránky textu zaplněny. Například pokud by byl obsah na velkém rozlišení rozdělen na dvě stránky a většina textu by byla na první stránce, pozice druhé stránky bude 50 %. Pokud se tato pozice přenese na menší rozlišení, kde by byl text rozdělen na deset stránek, pozice 50 % odpovídá stránce pět. Text na této pozici ovšem nebude odpovídat textu na stejné pozici na prvním zařízení. Tato vlastnost se projevuje zejména při přenosu pozice mezi zařízeními, kde je velký rozdíl ve velikosti displeje.

Výhodou tohoto řešení je jednoduchost implementace a rychlost algoritmu. K výpočtu stačí znát pouze číslo aktuální stránky a celkový počet stránek. Dosazením do jednoduchého vzorce lze vypočíst aktuální pozici a tu naopak převést na číslo stránky v textu.

Nevýhodou je občasná nepřesnost vypočtené pozice, ke které v některých případech dochází, jak je popsáno výše.

Tato metoda by byla použitelná pouze pro přepočet pozice na jednom zařízení, například aby čtenář zůstal na stejné pozici při změně velikosti textu. Pro přenos pozice mezi zařízeními s různě velkým displejem se nejedná o vhodný způsob výpočtu pozice.

4.4.2 Pozice znaku v textu

Druhou možností je počítat pozici znaku v celém textu. Nebude se jednat pouze o pozici místa, kde se daný text nachází, ale o **skutečnou pozici textu**. Díky tomu je získaná pozice přenositelná a není ovlivněna faktory jako velikost displeje nebo textu.

Při každé změně zobrazení, která ovlivňuje rozdělení textu na stránky (tedy velikost displeje, velikost a odsazení textu a podobně) je potřeba vypočítat počet znaků na všech stránkách. Aktuální pozice pak bude součet znaků na předcházejících stránkách plus jedna. Například pokud má text čtyři stránky, které obsahují deset, devět, jedenáct a osm znaků, pozice textu na začátku třetí stránky bude dvacet. Součet počtů znaků na stránkách jedna a dva je devatenáct. Začátek třetí stránky začíná na dalším znaku, je tedy potřeba přičíst jedničku a tím je získána výsledná pozice dvacet.

Převod opačným směrem, tedy z pozice na stránku, předpokládá, že jsou známy velikosti všech stránek. Od hledané pozice stačí postupně odečítat velikosti stránek a tím zjistit, zda se hledaný text nachází na dané stránce. Například při hledání stránky pro pozici dvacet se nejprve odečte velikost první stránky, tedy deset. Výsledek je také deset a od něho se odečte velikost druhé stránky, tedy devět. Od výsledku jedna se odečte velikost třetí stránky, tedy jedenáct. Výsledek je záporné číslo a to znamená, že hledaná pozice byla na třetí stránce.

Výhoda tohoto řešení je jeho přesnost a nezávislost na velkosti displeje, velikosti textu a dalších parametrech. Vypočtená pozice je jednoduše přenositelná a povede vždy na stejné místo v textu.

Nevýhoda tohoto řešení je časová náročnost výpočtu. Pro zjištění pozice i pro převod pozice na číslo stránky je potřeba vypočítat velikost všech stránek a s těmito hodnotami nadále pracovat. To znamená projít celý text stránku po stránce a na každé z nich spočítat počet znaků. Pokud se změní jakýkoliv parametr, který velikost stránek ovlivňuje, je potřeba všechny tyto hodnoty znovu přepočítat, protože text může být do stránek rozdělen jinak. To nastane například v případě, kdy si čtenář změní velikost textu. Čím více bude mít kniha stránek, tím pomalejší tento algoritmus bude.

Přestože má tento způsob výpočtu u delšího textu problém s rychlostí, jedná se o vhodnější metodu. Vypočtená pozice je vždy přesná a ukazuje na stejnou část

textu. Je tedy možné použít ji pro synchronizaci mezi zařízeními, které mají různou velikost displeje. Problém s rychlostí by navíc mohl být řešitelný. Program by si pro každou kombinaci parametrů mohl pamatovat velikost každé stránky a při změně těchto parametrů by nemusel vše znovu počítat, ale pouze by použil dřívější hodnoty. Pomalejší výpočet by se tak projevil pouze při prvním použití kombinace parametrů ovlivňující velikosti stránek.

5 Použité technologie a nástroje

5.1 C#

C# je objektový kompilovaný programovací jazyk. V dnešní době se jedná o jeden z nejrozšířenějších jazyků, v TIOBE indexu v lednu 2018 dosáhl pátého místa [8]. Lze pomocí něho vytvářet webové, mobilní, desktopové a další aplikace.

Jazyk C# vznikl jako moderní a jednoduchá alternativa k jazyku Java pro platformu Windows. První verze jazyka ještě neměla některé dnes běžně používané funkce, jako LINQ nebo generické třídy. Ty byly přidány až v roce 2005 s druhou verzí. LINQ se svého uvedení dočkal koncem roku 2007 s verzí C# 3. Zatím poslední majoritní verzí je **C# 7**, který přinesl například lokální funkce nebo *pattern matching*, koncept známý z některých funkcionálních jazyků. [7]

Pomocí jazyka C# byla vytvořena podstatná část celé aplikace[1]. Důvodem pro výběr tohoto programovacího jazyka byla jednoduchost jeho použití a možnost vytvářet pomocí něho multiplatformní aplikace.

5.1.1 .NET Framework

Jedná se o platformu od společnosti Microsoft, na které může běžet několik jazyků. Microsoft aktivně vyvíjí **C#** (objektový jazyk), **F#** (funkcionální jazyk) a **Visual Basic** (výukový jazyk) [9]. Součástí této platformy je běhové prostředí *CLR*[2] [9]. Aplikace napsaná v jednom z těchto jazyků je nejprve přeložena do *CIL*[3]. To je sada instrukcí, která se následně spouští nad běhovým prostředím *CLR* [6, stránka 218]. Díky tomu může například program v jazyce F# volat knihovnu napsanou v C#. Nezáleží tedy na tom, v jakém jazyce z rodiny .NET byl původně program napsán, protože je vždy nejprve přeložen do společného *CIL*.

[1] Dle statistik repozitáře na serveru GitHub přibližně 75 %

[2] Common Language Runtime

[3] Common Intermediate Language

5.2 Xamarin

Xamarin je nástroj pro tvorbu multiplatformních aplikací primárně pomocí jazyka C#. Lze také použít ostatní jazyky na platformě .NET, například F#. V této práci byl Xamarin spolu s C# použit pro tvorbu aplikace.

Roku 2011 byl Xamarin vytvořen vývojáři okolo produktu *Mono*[4]. Nejprve bylo nutné za použití Xamarinu platit, ale v roce 2016 ho koupila společnost Microsoft a následně byl uvolněn k použití zdarma.

Xamarin byl pro tvorbu aplikace zvolen díky možnosti jednoduše vytvořit multiplatformní aplikaci pro platformu Android a Windows pomocí jazyka C#.

5.2.1 Xamarin.Forms

Od roku 2014 obsahuje Xamarin nástroj Xamarin.Forms, který umožňuje vytvářet jednotné uživatelské rozhraní, které se kompiluje na cílové zařízení. Podporované platformy nyní jsou: iOS, Android, UWP, Windows 8.1 a Windows Phone 8.1. [10, stránka 8].

Pro tvorbu GUI lze využít **XAML**, značkovací jazyk postavený nad XML (více v kapitole 5.2.3). Druhou možností je psát grafické rozhraní pomocí kódu v C#.

Výhodou použití Xamarin.Forms je, že v aplikaci stačí napsat pouze jedno grafické uživatelské rozhraní, které bude fungovat na všech platformách. Navíc se jeho obsah na jednotlivých platformách renderuje tak, aby se co nejvíce podobal nativním prvkům dané platformy [10, stránka 1127]. Nevýhodu použití Xamarin.Forms je, že dokáží pouze to, co je možné vykreslit na každé podporované platformě [10, stránka 15]. To znamená, že pokud například Android umožňuje použít pro tlačítko nějaké specifické nastavení, které není na jiných platformách dostupné, nelze ho pomocí Xamarin.Forms nastavit.

Řešením pro tvorbu grafického rozhraní specifického pro platformu je vytvořit vlastní renderer a v něm požadované chování nastavit [10, stránka 1127]. Jeho podoba je naznačena v ukázce 5.1. Pokud by takových požadavků bylo hodně, nemusí být použití Xamarin.Forms vhodné. Bylo by lepší GUI napsat pro každou platformu zvlášť. Ztrácí se pak výhoda v podobě GUI napsaného na jednom místě. Pokud by například bylo potřeba do aplikace přidat další tlačítko, muselo by se přidat na několik místech, pro každou platformu zvlášť.

[4]Open source multiplatformní implementace prostředí .NET

```
1  [assembly: ExportRenderer(typeof(SettingsPage),
       typeof(SettingsPageRenderer))]
2  namespace EbookReader.Droid {
3      class SettingsPageRenderer : TabbedPageRenderer {
4          public override void OnViewAdded(Android.Views.View child)
               {
5              base.OnViewAdded(child);
6              var tabLayout = child as TabLayout;
7              if (tabLayout != null) {
8                  /* Tato vlastnost je dostupná pouze na platformě
                       Android, nelze ji proto nastavit pomocí
                       Xamarin.Forms */
9                  tabLayout.TabMode = TabLayout.ModeScrollable;
10             }
11         }
12     }
13 }
```

Ukázka kódu 5.1: Ukázka tvorby vlastního rendereru

5.2.2 PCL nebo Shared Project

Solution pro Xamarin aplikaci se skládá z několika projektů. První z nich obsahuje sdílený kód, který bude použit na všech platformách. Další projekty jsou vytvořeny pro jednotlivé platformy, které má aplikace podporovat. V případě aplikace, která má fungovat na mobilních telefonech s operačním systémem Android a na Windows, bude solution obsahovat tři projekty. Jeden společný, jeden pro Android a jeden pro Windows.

Jsou dva způsoby, jak může vypadat sdílený projekt. Prvním je **PCL**[5] a druhým **Shared Project**. První způsob, tedy PCL, znamená, že projekt se zkompiluje pouze jednou a výsledné DLL se používá na všech platformách. V době kompilace tedy není možné rozlišit cílovou platformu. Tu lze zjistit až za běhu aplikace pomocí volání `Device.RuntimePlatform`, jak je naznačeno v ukázce 5.2. Při použití tohoto řešení není možné ze sdíleného projektu volat API specifické pro nějakou platformu, ale lze volat pouze společný kód. Toto omezení lze vyřešit pomocí principu *Inversion of Control*. Požadovaná funkčnost, která je specifická pro nějakou platformu, se vy-

[5] Portable Class Library

člení do společného rozhraní. Toto rozhraní je implementováno v každém projektu specifickém pro platformu. Díky dependency injection získá společný projekt správnou implementaci a když volá nějakou metodu nad společným rozhraním, zavolá se kód pro danou platformu.

Řešení PCL má dnes již svého nástupce v podobě .NET Standard. Princip fungování a kompilace projektu ale zůstává velmi podobný, jako u PCL.

```
1 if (Device.RuntimePlatform == Device.UWP) {
2     /* Tento kód se provede pouze v UWP aplikaci */
3     BrightnessWrapper.IsVisible = false;
4 }
```

Ukázka kódu 5.2: Ukázka dotazování se na platformu za běhu aplikace

Druhou možností je použít **Shared Project**. To znamená, že sdílený projekt bude zkompilován pro každou platformu zvlášť. V době kompilace je tedy možné ptát se na cílovou platformu a na jejím základě vytvářet v kódu podmínky typu `#if __ANDROID__` a podobně. Použití této možnosti je zobrazeno v ukázce 5.3. Díky tomu je možné ve společném projektu volat i platformově specifické API. Pomocí podmínek na aktuální platformu lze obalit načítání balíčku pro danou platformu a volání jejich API. Nevýhodou toho řešení je velká nepřehlednost kódu. Na jednom místě se vyskytuje společný kód a zároveň kód pro všechny platformy. Přitom v jednu chvíli se vždy použije pouze společný kód a kód pro aktuální platformu. Zbylý kód se sice do výsledného DLL nedostane (díky podmínkám pro kompilátor), ale ve zdrojovém kódu stále bude. Toto řešení je vhodné pouze pro projekty, kde má být velká část kódu pro všechny platformy rozdílná. Pokud větší část kódu bude společná, je vhodnější použít PCL.

```
1 #if __ANDROID__
2 /* Tento kód se zkomplikuje pouze pro platformu Android */
3 using Android.App;
4 #endif
```

Ukázka kódu 5.3: Ukázka dotazování se na platformu v době kompilace

V této aplikaci je **použito PCL**. Aplikace začala být vytvářena v době, kdy ještě PCL bylo doporučovaným řešením. Nyní je doporučováno použít .NET Standard a aplikace by na toto řešení mohla být převedena. Také je zde využito výše popsané volání za pomocí *Inversion of Control*. V aplikaci je možné si nastavit jas displeje. K tomu je nutné mít možnost zjistit a nastavit jas zařízení. Xamarin nic ta-

kového neposkytuje a proto je nutné přistoupit k volání specifickému pro platformu. Ve společném projektu je vytvořeno rozhraní `IBrightnessProvider`. Toto rozhraní je pak implementováno na všech platformách. Ze společného kódu se volá pouze toto rozhraní a není důležité, jak a kde je implementováno.

5.2.3 XAML a binding

Pro vytváření uživatelského rozhraní v rámci Xamarin.Forms lze využít XAML. Jedná se o jazyk založený na XML, pomocí kterého lze zapsat **strukturu a částečně i chování GUI**. Obsahem XAMLu jsou elementy, které představují jednotlivé prvky rozhraní. Tyto elementy mohou mít atributy, které upřesňují jejich význam. V podstatě se jedná o podobný princip jako u jazyka HTML, který je také založen na XML.

Na rozdíl od HTML u XAMLu nelze použít kaskádové styly. Vzhled jednotlivých elementů lze ovlivnit pomocí atributů (např. barva písma a podobně). Tyto atributy je možné vyčlenit do tzv. `Resources` a hromadně je aplikovat na více elementů. Použít je lze buď na elementy nějakého typu (například na všechny elementy `Button`) a nebo pouze na vybrané. Skupině `Resources` lze přiřadit klíč a tento klíč lze poté nastavit u jednotlivých elementů. Poté se budou daná nastavení aplikovat pouze pro elementy s nastaveným klíčem. Nelze ale pracovat se selektory podobně jako v jazyce CSS. Tento princip je naznačen v ukázce 5.4.

XAML je navržen tak, aby podporoval tzv. binding. Pomocí něho lze například propojit hodnoty dvou elementů. Pokud je na stránce element `Entry` (sloužící k zadávání textu, podobně jako `input` v HTML) a `Label` (textový řetězec), tak je možné, aby se automaticky zadaná hodnota z `Entry` propisovala do textu v `Label`. Není třeba odposlouchávat změnu hodnoty v `Entry` a ručně nový text zapisovat do `Label`. Toto vše bude automatické.

Binding jde mnohem dál než je pouhé propojení elementů v rámci GUI. Je možné do struktury XAML pomocí atributu `BindingContext` předat instanci třídy obsahující vlastnosti, které lze bindovat. Této třídě se říká ModelView a její výhoda spočívá v tom, že k ní přistupuje nejenom XAML, ale i C#. Díky tomu lze oddělit datovou a prezentační část aplikace. Prezentační část (grafické rozhraní) je zcela nezávislá na ostatních vrstvách aplikace, se kterými komunikuje pouze pomocí bindingu. V rámci něho lze používat i tzv. `commands`. Pokud v GUI nastane

nějaká událost, na kterou by měl zareagovat zbytek aplikace, prezentační část spustí příslušný `command`, který je nadefinován v ModelView, a kde může událost zpracovat. Tímto způsobem tedy může prezentační část komunikovat se zbytkem aplikace. Příkladem využití tohoto principu je například klik na tlačítko v aplikaci, jak je naznačeno v ukázkách 5.5 a 5.6.

```
1  <!--- Definice stylu v Resources -->
2  <ContentPage.Resources>
3      <ResourceDictionary>
4          <Style x:Key="settingsWrapper" TargetType="StackLayout">
5              <Setter Property="Orientation" Value="Horizontal" />
6          </Style>
7      </ResourceDictionary>
8  </ContentPage.Resources>
9
10 <!--- Použití stylu -->
11 <ContentPage.Content>
12     <StackLayout Style="{StaticResource settingsWrapper}" />
13 </ContentPage.Content>
```

Ukázka kódu 5.4: Ukázka nastavení vzhledu elementů pomocí Resources. Nastavuje se vlastnost Orientation na hodnotu Horizontal a toto nastavení se použije pro všechny elementy StackLayout s nastaveným klíčem settingsWrapper

```
1  <Button Text="Připojit Dropbox"
2          Command="{Binding ConnectToDropboxCommand}" />
```

Ukázka kódu 5.5: Použití commandu v XAMLu

```
1  public ICommand ConnectToDropboxCommand { get; set; }
2
3  public SettingsSynchronizationVM() {
4      ConnectToDropboxCommand = new Command(ConnectToDropbox);
5  }
6
7  void ConnectToDropbox() {
8      /* Pošle se zpráva s žádosti o otevření okna s přihlášením k
          Dropboxu */
9      IocManager.Container.Resolve<IMessageBus>().Send(new
          OpenDropboxLogin());
10 }
```

Ukázka kódu 5.6: Zachycení a zpracování commandu v ModelView

Šablona XAML může být propojena s C# třídou, která je umístěna ve stejnojmenném souboru jako XAML, jen má navíc příponu *.cs*. Tato třída má atribut `[XamlCompilation(XamlCompilationOptions.Compile)]` a v konstruktoru volá metodu *InitializeComponent*. Jedná se o takzvanou *partial* třídu. To znamená, že je rozdělena do více souborů. Do druhé části této třídy je automaticky zkompilován příslušný XAML a díky tomu je možné z této třídy přistupovat k veškerým prvkům, které jsou jeho součástí. Pokud nejsou v XAMLu použity `commands`, tak se mohou pro zpracování akcí používat události. Ty jsou zpravidla umístěny v této třídě.

5.2.4 Použité pluginy a další balíčky

Práce na platformě .NET je založena na tom, že jednotlivé komponenty programu jsou rozděleny do balíčků, kterým se říká `assembly`. Důvodem pro toto rozdělení je snazší údržba jednotlivých balíčků [6, stránka 17]. Balíčky, které nejsou součástí základního .NET Frameworku, se zpravidla instalují pomocí nástroje Nuget[6]. Samotný Xamarin lze také rozšiřovat pomocí Nuget balíčků, kterým se říká plugin. V této kapitole jsou popsány nejdůležitější balíčky a pluginy, které byly potřebné k vytvoření aplikace. Zdaleka to nejsou všechny, které byly použity. Mezi další lze jmenovat například balíčky pro práci s formáty JSON nebo ZIP, SDK[7] pro Dropbox, Firebase a další.

[6] https://www.nuget.org/

[7] Sada vývojových nástrojů

Autofac

Autofac[8] je IoC kontejner pro platformu .NET. Inversion of Control je programátorský návrhový vzor, ve kterém třídy nevytváří přímo instance jiných tříd, ale ty jsou jim předány. Například třída `EpubLoader`, která načítá EPUB, potřebuje mít přístup k souborovému systému, který zajišťuje rozhraní `IFileService`. To je implementováno třídou `FileService` a tato implementace se zaregistruje do IoC kontejneru. Samotná třída `EpubLoader` si nevytvoří instanci `IFileService`, ale pouze požádá IoC kontejner o instanci tohoto rozhraní. Kontejner poté vrátí správnou instanci, se kterou může třída `EpubLoader` pracovat. Výhoda takového řešení mimo jiné spočívá v tom, že implementaci rozhraní lze změnit na jednom místě. Stačí do IoC kontejneru zaregistrovat jinou implementaci a všude, kde je rozhraní vyžadováno, se bude automaticky používat nová implementace.

```
1  /* Registrace pojmenované třídy, která implementuje EpubParser, do
       IoC kontejneru */
2  ContainerBuilder.RegisterType<Epub200Parser>()
3      .Keyed<EpubParser>(EpubVersion.V200);
4  ContainerBuilder.RegisterType<Epub300Parser>()
5      .Keyed<EpubParser>(EpubVersion.V300);
6
7  /* Vyžádání instance EpubParser s konkrétním jménem z IoC
       kontejneru. Konstruktoru požadované třídy lze předat další
       parametry. */
8  var epubParser = IocManager.Container.ResolveKeyed<EpubParser>(
9      epubVersion, // Zaregistrované jméno
10     // Parametry pro konstruktor
11     new NamedParameter("package", package),
12     new NamedParameter("folder", epubFolder),
13     new NamedParameter("contentBasePath", contentBasePath)
14 );
```

Ukázka kódu 5.7: Použití IoC kontejneru pro pojmenované třídy

Pro platformu .NET existuje několik kontejnerů, které nabízí velmi podobné funkce a možnosti. Mimo Autofac jsou to například Castle Windsor[9] nebo Ninject[10]. Samotný Xamarin také obsahuje vlastní Dependency Service, ale ta nedosahuje ta-

[8] https://autofac.org/
[9] https://github.com/castleproject/Windsor
[10] http://www.ninject.org/

kového komfortu, jako zmíněný Autofac. Neumí například vkládat závislosti pomocí konstruktoru třídy nebo mít pro nějaké rozhraní více implementací a za běhu aplikace vybírat to, které se má zavolat. Toto je vhodné například v situaci, kdy existuje obecné rozhraní pro čtení souboru a jeho implementace pracují s konkrétním formátem. Za běhu aplikace se pak musí zavolat taková implementace, která odpovídá formátu otevřeného souboru. Využití této možnosti je naznačeno v ukázce 5.7.

PCLStorage

Každá aplikace na platformě Android i UWP dostává k dispozici prostor v souborovém systému, kam může ukládat data. PCLStorage[11] je balíček, který umožňuje jednoduchý a sjednocený přístup k tomuto prostoru v Xamarin aplikaci. V rámci přiděleného prostoru je možné vytvářet, editovat nebo mazat soubory a adresáře. Uložená data jsou perzistentní, při dalším spuštění aplikace tedy zůstanou zachována. V této aplikaci je plugin využíván k práci s e-booky, které si čtenář přidal do knihovny. Ty jsou překopírovány do úložiště vyčleněného pro aplikaci a odtud se s nimi pracuje pomocí pluginu PCLStorage.

WebView

WebView je komponenta, která umí renderovat webové stránky, tedy HTML, CSS a JavaScript. Obsah knihy ve formátu EPUB je uložen pomocí HTML a součást aplikace, která se stará o samotné vykreslení knihy (viz kapitola 6.3), je napsána pomocí jazyka JavaScript a CSS. Proto je nutné mít k dispozici WebView, které toto vše dokáže správně zpracovat a zobrazit. V Xamarin.Forms je k dispozici výchozí WebView, ale pro potřeby této aplikace bylo potřeba využít pokročilejší nástroj. Zvolen byl plugin Xam.Plugin.WebView[12], který mimo jiné umožňuje jednoduchou komunikaci mezi JavaScriptem ve WebView a kódem v C# nebo zvládá renderovat HTML předané v textovém řetězci, a nikoliv pouze webovou stránku. To je pro aplikaci klíčové, protože když se otevírá e-book, tak se nejprve načte připravená HTML šablona (kapitola 6.3) a ta se vloží do WebView. Potom se do něho pomocí zpráv (kapitola 6.5) pošle text knihy.

[11] https://www.nuget.org/packages/PCLStorage

[12] https://www.nuget.org/packages/Xam.Plugin.WebView/

Settings

Perzistentní ukládání uživatelského nastavení je další z problémů, který je třeba vyřešit. V této aplikaci byl použit plugin Xam.Plugins.Settings[13], který ukládání a načítání uživatelského nastavení pro všechny platformy sjednocuje. Vývojář je odstíněn od toho, jakým způsobem se nastavení ukládá na jednotlivých platformách, a místo toho pracuje s jednotným API. Jednotlivá uživatelská nastavení jsou reprezentována dvojicí klíč a hodnota. Klíč je textový řetězec a hodnota může být jeden ze základních datových typů, jako je boolean, textový řetězec nebo číslo. Pokud není hodnota pro nějaký klíč nastavena, je možné si nadefinovat výchozí hodnotu, kterou bude plugin vracet. Použití tohoto pluginu je ukázáno v ukázce 5.8.

```
1 public static class UserSettings {
2     private static ISettings AppSettings => CrossSettings.Current;
3
4     public static bool OnlyWifi {
5         get => AppSettings.GetValueOrDefault("onlyWifi", false);
6         set => AppSettings.AddOrUpdateValue("onlyWifi", value);
7   }
8 }
```

Ukázka kódu 5.8: Práce s uživatelským nastavením

FilePicker

Přestože je Xamarin nástroj pro tvorbu multiplatformních aplikací a sjednocuje spoustu prvků, v jeho základu není obsažen žádný jednotný FilePicker, tedy komponenta pro výběr souboru uživatelem. Ten je potřeba proto, aby si mohl čtenář vybrat, jakou knihu chce v aplikaci otevřít. Vývojář má dvě možnosti. Buď pro každou podporovanou platformu použít nativní FilePicker a to znamená napsat nějaký kód, který práci s nimi sjednotí. Druhou možností je použít plugin třetí strany, kde již takový kód napsán je a stačí ho použít. V rámci této práce bylo zkoušeno několik takových pluginů, ale většinou se na systému Android potýkaly s problémy při načítání souborů z externího úložiště nebo z Dropboxu. Nakonec byl zvolen balíček Xamarin.Plugin.FilePicker[14], který těmito problémy netrpí.

[13] https://www.nuget.org/packages/Xam.Plugins.Settings
[14] https://www.nuget.org/packages/Xamarin.Plugin.FilePicker/

SQLite

SQLite je velmi rozšířená multiplatformní SQL databáze, která je uložena v jednom souboru. Vzhledem ke své malé velikosti a velké rychlosti je vhodná k použití v mobilních zařízeních [13]. V aplikaci je pro práci s databází použit balíček sqlite-net-pcl[15]. Ten nabízí jednoduchou ORM[16] vrstvu. Díky tomu lze k databázi přistupovat pomocí jazyka LINQ a pro základní použití není potřeba psát žádné SQL příkazy. V aplikaci je databáze využita k uložení knihovny uživatele a záložek v knihách.

5.3 JavaScript

JavaScript je objektově orientovaný skriptovací jazyk. Dříve se používal zejména na klientské části webu k rozpohybování statických stránek. Například je možné pomocí něho vytvářet animace, manipulovat s DOM[17] nebo načítat další části webové stránky na pozadí. V dnešní době dostává JavaScript také velký prostor i mimo svět webových aplikací. Na platformě Node.js[18] lze psát aplikace v JavaScriptu, které nejsou určeny pro webové prohlížeče.

V této práci byl JavaScript použit pro tvorbu části aplikace, která zobrazuje text knihy, umožňuje přecházet mezi stránkami nebo reagovat na uživatelská gesta. Podrobněji je tato část aplikace popsána v kapitole 6.3.

5.4 CSS

CSS se používá v kombinaci s jazykem HTML. Ten určuje strukturu dokumentu a pomocí CSS je možné nastavit jeho vzhled. Například velikost písma, barvy, odsazení, ale také složitější vlastnosti, jako jsou plovoucí elementy a podobně. Užitečným nástrojem v CSS 3 je možnost automaticky rozdělit text do sloupců, aniž by na začátku nebo konci sloupce zůstával uříznutý text. Právě této vlastnosti bylo využito při tvorbě čtečky elektronických knih.

[15] https://www.nuget.org/packages/sqlite-net-pcl
[16] Automatické mapování mezi třídou v kódu a tabulkou v databázi
[17] Document Object Model
[18] https://nodejs.org/en/

6 Návrh a implementace aplikace

6.1 Knihovna

Aplikace obsahuje knihovnu, ve které jsou uloženy **čtenářovi knihy**. Ten ji nejprve přidá do aplikace (podrobněji popsáno v kapitole 6.1.1) a poté ji může otevírat z hlavní obrazovky. Aplikace si pomocí SQLite databáze uchovává údaje o knize, jako například její název, jméno autora či poslední pozici čtenáře v textu. Díky tomu je možné se při příštím otevření knihy vrátit na pozici, kde čtenář skončil.

Každá kniha je v rámci knihovny identifikována unikátním ID, které je automaticky vytvářeno při přidání knihy do knihovny. Jedná se o MD5 hash, který je generován z pole bajtů, ve kterém je uložen obsah souboru. Toto ID identifikuje knihu nejen v knihovně na jednom zařízení, ale také v rámci všech zařízení čtenáře. Pokud mají knihy na různých zařízeních stejné ID, jsou považovány za stejnou knihu a může mezi nimi probíhat synchronizace postupu čtení a záložek.

Pro práci s knihovnou je v projektu připravena třída `BookshelfService`, která obsahuje metody pro přidání, smazání a úpravu knihy a dále pro načtení knihy z knihovny. Samotná data jsou uložena v SQLite databázi v tabulce `Book`. Třída `BookshelfService` nepracuje s touto databází přímo, ale využívá k tomu další třídu `BookRepository`. Ta obsahuje základní CRUD operace[1] nad touto tabulkou. Třída `BookshelfService` provádí navíc další operace, jako například generování ID knihy při přidání do knihovny, kontrola zda kniha se stejným ID již v knihovně neexistuje či načtení metadat knihy.

6.1.1 Otevření souboru

Na hlavní obrazovce obsahuje aplikace tlačítko pro přidání knihy do knihovny. Toto tlačítko je obslouženo FilePickerem (kapitola 5.2.4), ve kterém si uživatel

[1] Create, Read, Update, Delete

vybere soubor, který chce otevřít. Následně dostane aplikace od FilePickeru dva údaje: název vybraného souboru a pole bajtů. Nejprve je zavolána statická metoda `EbookFormatHelper.GetBookLoader`, která na základě přípony souboru v jeho názvu rozhodne o typu knihy (podporovány jsou soubory EPUB, TXT a HTML) a vrátí správnou instanci rozhraní `IBookLoader`.

Rozhraní `IBookLoader` obsahuje několik metod pro práci s e-booky. Pro každý podporovaný formát knihy existuje jeho vlastní implementace. Důležitou metodou tohoto rozhraní je `GetBook`. Ta má na vstupu název souboru a pole bajtů, které jsou získány z FilePickeru. Úkolem této metody je vytvořit si z pole bajtů soubor a uložit ho do uživatelského prostoru v souborovém systému (viz kapitola 5.2.4). Dále si ze souboru načte metadata knihy a ty tato metoda vrací. Konkrétní implementace záleží na typu souboru. Například třída pro práci s formátem EPUB musí souboru přidat příponu *.zip* a rozbalit ho. Získá speciální strukturu souborů, které poté překopíruje na souborový systém. Pro získání metadat knihy je potřeba otevřít OPF soubor (který byl uložen v rámci ZIP souboru) a z něho požadované informace získat (viz popis struktury formátu EPUB v kapitole 2.2). Implementace, která pracuje s formáty v jednom souboru, jako jsou HTML či TXT, je jednodušší. Není potřeba rozbalovat ZIP ani procházet OPF soubor. Stačí pouze soubor překopírovat do uživatelského prostoru a jako název knihy vrátit název souboru bez přípony. Žádná další metadata pro tyto soubory není třeba načítat.

Další důležitou metodou je `OpenBook`, která má na vstupu cestu ke knize (kam byla překopírována při prvním otevření knihy) a výstupem jsou podobně jako u metody `GetBook` metadata knihy. Na rozdíl oproti předchozí metodě je tato volána, pokud je potřeba otevřít knihu z knihovny. Například u formátu EPUB již nemusí rozbalovat ZIP a kopírovat soubory do uživatelského úložiště, ale pouze čte OPF soubor a vrací data získaná z něj. Při prvním otevření knihy nejsou všechny její metadata uložena do databáze, proto je potřeba některá číst při opětovném otevření.

Jednou z informací, kterou `IBookLoader` o knize načítá, je seznam kapitol. Proto deklaruje toto rozhraní metodu `GetChapter`, která má na vstupu požadovanou kapitolu a vrací její text. V případě formátu EPUB je třeba načíst příslušný soubor a vrátit jeho obsah. Pokud se jedná o TXT nebo HTML, kniha obsahuje pouze jednu kapitolu a vrací stále obsah jednoho souboru.

Než je obsah kapitoly poslán do čtečky, je potřeba nejprve daný **text upravit**.

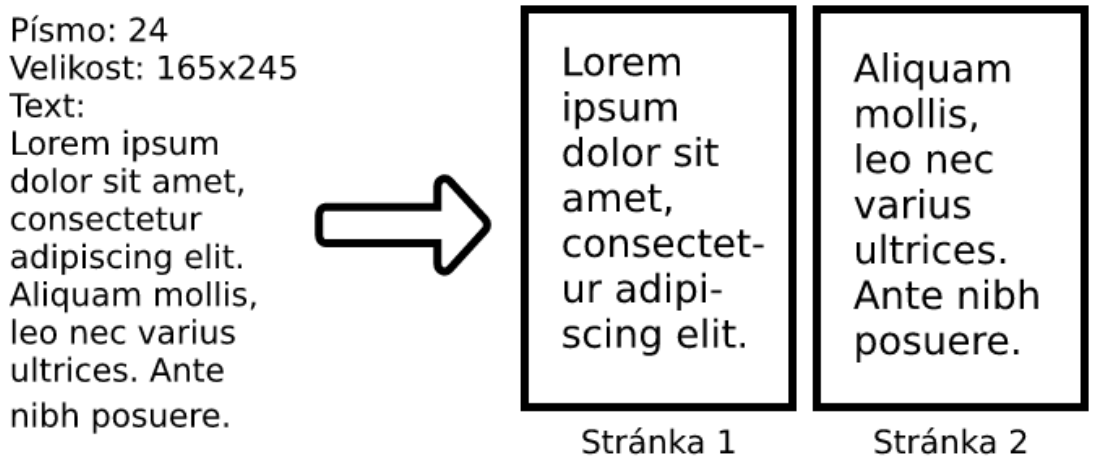
Pokud se jedná o TXT soubor, je potřeba text dle nových řádků rozdělit do HTML odstavců. Čtečka totiž renderuje HTML a nové řádky z textového soubory by ignorovala. Pokud se jedná o soubor EPUB, je potřeba připravit obrázky, které se v textu mají zobrazit. Jsou procházeny všechny HTML elementy `img` a načteny obrázky, na které se odkazují. Ty jsou převedeny do Base64 a jsou jim přiřazeny identifikátory. Těmito identifikátory jsou označeny i elementy `img`. Při zaslání obsahu do čtečky jsou zaslány i obrázky v kódování Base64 a čtečka si následně přiřadí obrázky dle identifikátorů do správných elementů. Díky tomu nemusí čtečka načítat obrázky ze souborového systému sama a řešit problémy s různým zápisem cesty, ale všechny obrázky jsou jí předány. Společnou úpravou pro textové a HTML soubory (včetně formátu EPUB) je odstranění nežádoucích HTML značek. Jedná se o `script`, `style` a `iframe`. Tyto značky by mohly obsahovat JavaScript či kaskádové styly, které by mohly způsobit nefunkčnost čtečky, která právě JavaScript a kaskádové styly využívá. Mohlo by se jednat i o bezpečnostní riziko. V knize by bylo možné načítat webové stránky či spouštět JavaScript, který by například zobrazil falešný formulář s přihlášením k Dropboxu a sbíral by přihlašovací údaje. Proto je vhodné tyto značky z HTML před zobrazením odstranit.

6.2 Renderování obsahu

6.2.1 Vlastní řešení renderování

V rámci práce na aplikaci bylo zkoušeno vlastní renderování obsahu. Implementace byla zaměřena na rozdělení textu do jednotlivých stránek zobrazených v aplikaci. Na vstupu tohoto algoritmu byl text, velikost displeje a požadovaná velikost textu. Výstupem byl text rozdělený na jednotlivé stránky. Algoritmus si na základě velikosti displeje a textu spočítal, kolik znaků se vejde na řádek, a kolik řádků textu se vejde na stránku. Poté si vstupní text rozdělil na jednotlivá slova a postupně z těchto slov sestavoval řádky. Pokud bylo slovo příliš dlouhé, bylo rozděleno na dvě. V takovém případě bylo nutné hlídat, aby jedna část rozděleného slova nebylo pouze jedno písmeno. Takto by bylo slovo rozděleno pouze v případě, že by se nevešlo na samostatný řádek. Jinak bylo slovo, ze kterého by rozdělením vzniklo jen jedno písmeno, celé posunuto na další řádek. Princip fungování algoritmu je zobrazen na obrázku 6.1.

Toto řešení se ukázalo jako velmi komplikované. Jeden z problémů bylo například



Obrázek 6.1: Princip fungování algoritmu pro rozdělení textu na stránky

použití proporcionální písma, ve kterém má každé písmeno v textu jinou šířku. V takovém případě není možné spočítat, kolik znaků se vejde na řádek, protože záleží na použitých znacích. Další komplikace by nastaly při zpracování HTML dokumentu, kde by bylo nutné řešit zanoření elementů nebo nastavené CSS vlastnosti. Toto řešení se tedy neukázalo jako vhodné a bylo nahrazeno použitím WebView, ve kterém jsou již tyto problémy vyřešeny.

6.2.2 Využití existujícího řešení

Při tvorbě aplikace byla využita druhá cesta renderování obsahu, tedy využití existujícího jádra v podobě WebView. To existuje na všech potřebných platformách a přístup k němu je sjednocen pomocí pluginu Xam.Plugin.WebView. WebView funguje podobně jako internetový prohlížeč. Dokáže zobrazit HTML včetně CSS. Umí také spouštět JavaScript a dokáže zprostředkovat komunikaci mezi ním a okolním kódem. Do tohoto WebView je poslána šablona knihy včetně obslužného JavaScriptu (viz kapitola 6.3) a zároveň obsah knihy. V šabloně se poté kniha zobrazí rozdělená na jednotlivé stránky, mezi kterými lze přecházet.

Pro **rozdělení textu na jednotlivé stránky** je využito CSS. Pomocí vlastnosti `column-width` je možné nastavit požadovanou šířku sloupce a obsah je poté do nich automaticky rozdělen tak, aby všechny sloupce měly požadovanou velikost a aby se do nich vešel veškerý obsah [16]. Hodnota této vlastnosti je nastavena v závislosti na velikosti displeje zařízení. Následně je možné spočítat **celkový počet stránek**. Na konec textu je umístěn označený element, který bude na po-

slední stránce. Poté se spočítá jeho horizontální pozice od levého okraje stránky a po vydělení šířkou jedné stránky je získán celkový počet stránek. Podobně funguje i **posun na jednotlivé stránky**. V rámci elementu, který je rodičem elementu rozděleného na sloupce, se lze posouvat horizontálně do stran. Pozice jednotlivých stránek jsou násobky šířky stránky a po posunutí na tyto násobky budou zobrazeny požadované stránky textu. Toto posunutí je možné realizovat hned nebo ho lze pomocí JavaScriptu animovat. Implementace kódu pro posun v textu je naznačena v ukázce 6.1. Rodičovskému elementu je také možné nastavit CSS vlastnost `will-change` s hodnotou `scroll-position`. Tím je vykreslovacímu jádru oznámeno, že v rámci elementu se bude posouvat do stran a může tomu být přizpůsobena optimalizace práce s DOM [17]. Díky tomu je zvýšena rychlost posouvání na jednotlivé stránky.

```
1 $('#columns-outer').animate({
2     scrollLeft: (page - 1) * this.pageWidth,
3 }, duration);
```

Ukázka kódu 6.1: Posun na stránku textu

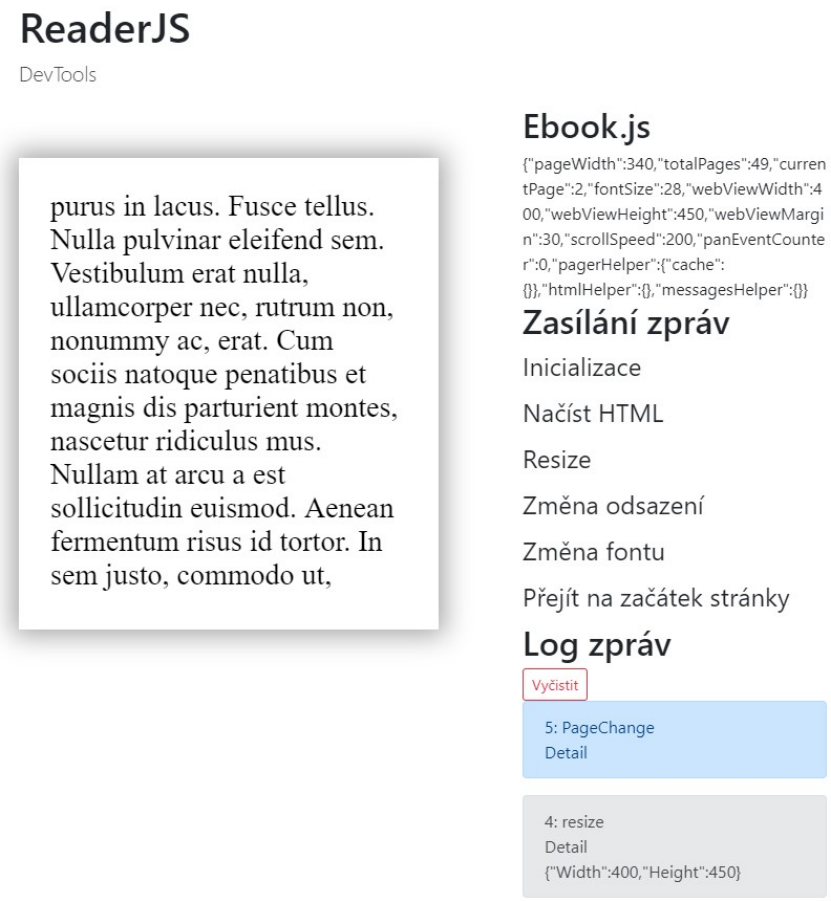
Toto řešení se ukázalo jako vhodnější pro použití v této aplikaci. Není potřeba renderovat složitou HTML strukturu, ale tato práce je přenechána hotové komponentě. Pro rozdělení textu na stránky stačí využít několika CSS vlastností. Nevýhodou tohoto řešení je závislost na konkrétní implementaci renderovacího jádra na cílovém zařízení, které se může mírně lišit. Ale i přes tento problém je použití tohoto řešení jednodušší a rychlejší. Proto bylo k tvorbě aplikace zvoleno.

6.3 ReaderJS

V rámci aplikace je komponenta, která má na starosti samotné vykreslení HTML, vyčleněna do samostatného projektu, který se jmenuje ReaderJS. Ten se skládá z HTML a CSS šablony e-booku a JavaScriptu. Na vstupu se očekává HTML jedné kapitoly knihy a výstupem je potom tento text vyrenderovány do HTML šablony. JavaScript, který je součástí projektu, se stará o posun stránek, výpočet aktuální pozice a posun na požadovanou pozici, komunikaci s kódem v C# nebo zachycení uživatelských gest.

Pro efektivnější vývoj JavaScriptové částí aplikace byl vytvořen interní nástroj DevTools (obrázek 6.2). Jedná se o HTML stránku, která má v sobě pomocí elementu

`iframe` vloženou JavaScriptovou čtečku. Velkou výhodou tohoto nástroje je možnost ladit její JavaScriptový kód pomocí webového prohlížeče a například Chrome DevTools. Dále je možné pomocí tohoto nástroje simulovat komunikaci mezi C# a JavaScriptem. Do JavaScriptové čtečky je vložen kousek kódu, který vytváří vlastní funkci `csCallback`, která je potřebná k zasílání zpráv do C# (viz kapitola 6.5), zjišťuje a vrací interní stavy čtečky a podobně. Nástroj se nachází v adresáři DevTools. Pro jeho spuštění stačí otevřít soubor *index.html* ve webovém prohlížeči.



Obrázek 6.2: Ukázka vlastního nástroje DevTools

K projektu ReaderJS jsou také napsány testy. Ty jsou vytvořeny pomocí testovacího frameworku Jest[2] od společnosti Facebook. V jednotlivých testech se pomocí nástroje Chromeless[3] načítá stránka DevTools a následně se pomocí zasílání zpráv a simulací klikání ve čtečce testují její jednotlivé funkce. Například se do čtečky načte HTML a testuje se správné zasílání zprávy s pozicí v textu, správný text zobrazený ve čtečce, posun na další stránky a podobně. Testy se nachází ve složce *Tests*. K je-

[2]https://facebook.github.io/jest/
[3]https://github.com/graphcool/chromeless

jich spuštění stačí zavolat příkaz `yarn test` v kořenovém adresáři projektu ReaderJS. Předpokladem je mít nainstalovaný nástroj yarn a všechny závislosti aplikace.

6.4 Implementace algoritmu pro zjištění pozice

Princip implementace tohoto algoritmu je v celku jednoduchý. Je potřeba spočítat, kolik textových znaků je na každé stránce. Pozice začátku stránky je pak součet velikosti všech předešlých stránek. Nejprve je potřeba projít všechny stránky v textu. Na začátku každé stránky se pomocí rozhraní `Range`[4] umístí HTML element, který značí začátek stránky (viz ukázka 6.2). Poté je celý obsah HTML uložen do řetězce a rozdělen na podřetězce, dle výskytu vložené značky, která odděluje jednotlivé stránky. Nyní každý podřetězec reprezentuje obsah jedné stránky textu. Stačí pouze ze všech podřetězců odstranit zbylé HTML značky a vytvořit z nich čistý text. Ve všech podřetězcích je již možné spočítat počet znaků. Výstupem pak je pole, ve kterém je délka každé stránky textu. Posledním krokem je odstranění pomocných značek z DOM, které tam byly vloženy.

Vypočtené délky jednotlivých stránek lze použít i v opačném směru, tedy přesun na danou pozici v knize. Princip spočívá v postupném sčítání velikostí jednotlivých stránek od první až do té doby, kdy součet bude vyšší než hledaná pozice. Ta se bude nacházet na poslední stránce, která v součtu figurovala. Tento algoritmus je zobrazen v ukázce 6.3.

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

```
var rect = {
    top: Ebook.webViewMargin,
    left: Ebook.webViewMargin,
};

var ranges = [];
for (var i = 1; i <= Ebook.totalPages; i++) {
    Ebook.goToPageFast(i);

    var range = document.caretRangeFromPoint(rect.left, rect.top);
    if (range !== null) {
        ranges.push(range);
    }
}

ranges.forEach(function(range) {
    var mark = document.createElement("span");
    mark.setAttribute("class", "js-ebook-page-begin");
    range.insertNode(mark);
});
```

Ukázka kódu 6.2: Vložení značky na začátek každé stránky

```
Ebook.pagerHelper.computeLengthOfAllPages();
var page = 0;
var currentPosition = 0;
while (currentPosition < position) {
    page++;
    var length = Ebook.pagerHelper.cache.get(page);
    if (length !== undefined) {
        currentPosition += length;
    }
}
```

Ukázka kódu 6.3: Převod pozice na číslo stránky

6.4.1 Řešené problémy

V rámci implementace algoritmu pro výpočet pozice bylo řešeno několik problémů. Rozhraní `Range` obsahuje metodu, která vrací obsah zobrazený na stránce. Prvotní

implementace tedy tuto metodu využívala, ale brzy se ukázalo, že má svoje limity. Pokud text nezačínal na začátku stránky, ale byl trochu posunutý, tak metoda rozhraní `Range` vrátila prázdný obsah. Také se často vyskytoval problém s koncem stránky. Ve vráceném obsahu nebyl pouze text z dané stránky, ale i několik řádků ze stránky následující. Proto se tato metoda ukázala jako nevyhovující a bylo nutné použít jiný způsob, tedy značit si začátky stránek a text dle těchto značek ručně rozdělit.

S ručním dělením obsahu souvisí další problém. Pokud na přelomu dvou stránek je jeden textový řetězec, je nutné pro označení začátku stránky tento textový řetězec rozdělit a na přesně danou pozici vložit HTML element. Při následném odstranění těchto značek by ale obsah nezůstal v původní podobě. Místo jednoho textového uzlu by nyní DOM obsahoval dva sousední textové bloky, které mohou být vyrenderovány jinak. Z první stránky se může přesunout poslední řádek na druhou stránku a změní se tak umístění všech pozic a celkový počet stránek. Proto je nutné při mazání pomocných značek spojovat sousední textové bloky do jednoho, aby nedocházelo k tomuto problému.

Posledním problémem této metody je její rychlost. Manipulace s DOM může být velmi náročná, zejména pokud se s ním pracuje neefektivním způsobem. Špatným přístupem je do DOMu nějakou změnu zapsat, potom z něho číst, potom znovu zapsat a takto dále v cyklu. Lepší je všechny zápisy provést najednou. Důvodem je to, že pokud je z DOMu čten nějaký parametr, například velikost elementu, prohlížeč si tuto hodnotu cachuje a při příštím čtení ji rovnou vrátí. Tato cache se ale invaliduje při zápisu a pokud se tedy do DOMu střídavě zapisuje a čte se z něho, cache nemá žádný účinek. Na toto je pamatováno i ve výše popsané implementaci algoritmu a nejprve se projdou všechny stránky, připraví se HTML značky a do DOMu jsou uloženy společně.

6.5 Komunikace mezi C# a JavaScriptem

Programy v C# a JavaScriptu spolu mohou komunikovat pomocí zasílání zpráv. Ty jsou zasílány ve formátu JSON a skládají se ze dvou částí. První je `action`, což je textový název dané zprávy. Díky tomuto parametru příjemce ví, jak má danou zprávu zpracovat. Druhou částí je parametr `data`. Zde je uložen obsah zaslané zprávy, například číslo stránky ve zprávě, která oznamuje změnu stránky. Celý JSON je pře-

veden na textový řetězec a zakódován pomocí Base64. Výsledek je poté zaslán druhé straně. Ta musí nejprve Base64 dekódovat, převést textový řetězec zpět na JSON a ten následně dle nastaveného parametru `action` zpracuje.

Zprávy jsou zasílány asynchronně a nedochází k potvrzování příjetí. To znamená, že když například C# požádá o posun na pozici v knize, nedostane zpět informaci o tom, zda k přesunu opravdu došlo.

6.5.1 Komunikace na straně C#

K zasílání a příjmu zpráv v C# se používá třída `WebViewMessages`. Ta potřebuje ke svému správnému fungování instanci WebView, se kterým bude komunikovat. Tuto instanci přijímá ve svém konstruktoru. Třída obsahuje veřejnou metodu `Send`, která jako parametry přijímá název zprávy a její obsah. Zpráva nemusí být poslána hned, ale je zařazena do fronty. Ta čeká, až bude inicializováno WebView a poté jako první zašle zprávu `init`. Když jsou tyto dvě podmínky splněny, je zaslán zbytek fronty. Samotné předání zprávy JavaScriptu je provedeno zavoláním metody `InjectJavascriptAsync` nad instancí WebView. V parametru této metody je předán řetězec obsahující JavaScript, který se má spustit. V tomto případě je to volání `Messages.parse()`, kde je v parametru Base64 se zprávou. Toto je zobrazeno v ukázce 6.4.

```
1 await _webView
2     .InjectJavascriptAsync($"Messages.parse('{toSend}')");
```

Ukázka kódu 6.4: Asynchronní předání zprávy JavaScriptu

Tato třída si dále zaregistruje pomocí metody `AddLocalCallback` nad instancí WebView callback pro volání funkce `csCallback` v JavaScriptu (ukázka 6.5). V parametru tohoto volání je zaslaná zpráva z JavaScriptu. Třída `WebViewMessages` ji zpracuje a spustí událost, která je nadefinována pro každý typ zprávy, který lze přijímat. Ke konkrétní události se mohou přihlásit zbylé části aplikace a mohou tak jednotlivé zprávy zpracovávat.

```
1 _webView.AddLocalCallback("csCallback", (data) => {
2     this.Parse(data);
3 });
```

Ukázka kódu 6.5: Přijetí zprávy z JavaScriptu

Zasílané zprávy

init K inicializaci JavaScriptové čtečky je zasílána zpráva `init`. Její parametry jsou šířka a výška okna, velikost textu, velikosti odsazení a další nastavení, jako rychlost posunutí nebo noční režim. Po přijetí zprávy si JavaScript nastaví tyto základní proměnné a zaregistruje listenery pro uživatelská gesta.

loadHtml `LoadHtml` je zpráva sloužící k zasílání HTML do čtečky v JavaScriptu. Kromě textu obsahuje také seznam všech obrázků převedených do Base64. Tyto obrázky jsou následně vloženy do HTML. Dále je možné v této zprávě nastavit, na jakou pozici se má čtečka posunout v zaslaném obsahu. Možnosti jsou konkrétní pozice vyjádřena číslem, poslední stránka textu nebo značka v textu (ID nějakého elementu). Pokud žádná z těchto možností není nastavena, provede se posun na první stránku textu.

goToPosition Zpráva `goToPosition` je zasílána, pokud C# požaduje posun na určitou pozici v textu. To může nastat například pokud přijde synchronizace knihy a je potřeba se posunout na přijatou pozici.

changeFontSize Zpráva `changeFontSize` je zasílána, pokud si uživatel změní velikost textu. Obsah zprávy je nová hodnota velikosti textu.

resize Pokud se změní velikost obrazovky (například změna orientace telefonu), tak je do JavaScriptu zaslána zpráva `resize`, která jako parametry obsahuje novou šířku a výšku okna.

changeMargin Zprávu `changeMargin` odesílá C#, když si uživatel v nastavení změní odsazení textu, a je potřeba tuto změnu reflektovat v zobrazeném textu. Obsahem zprávy je nová hodnota odsazení.

goToPage Zpráva `goToPage` je zasílána, pokud je potřeba se přesunout na jinou stránku. V parametrech zprávy je možné zaslat konkrétní stránku a nebo požádat o posun na následující či předchozí stránku. Využití této zprávy je při ovládání aplikace mimo WebView. Například mobilní aplikaci lze ovládat pomocí tlačítek pro změnu hlasitosti. Kód v C# si odchytne jejich stisknutí a zašle JavaScriptu zprávu s žádosti o posun na další či předchozí stránku.

6.5.2 Komunikace na straně JavaScriptu

Ke komunikaci na straně JavaScriptu se používá objekt `Messages`, který je přidán do instance `Window`. Jedná se tedy o globální objekt, který lze kdekoliv zavolat. Obsahuje metodu `send`, která přijímá název a obsah zprávy. Zpráva je zaslána ihned a to pomocí volání funkce `csCallback`, kde je jako parametr předán obsah zprávy (zakódován pomocí Base64). Jedná se o speciální funkcí ve `WebView`, která zapříčiní zavolání callbacku na straně C#. Tento kód je zobrazen v ukázce 6.6.

```
1 var json = JSON.stringify({
2     action: action,
3     data: data,
4 });
5
6 csCallback(Base64.encode(json));
```

Ukázka kódu 6.6: Zaslání zprávy v JavaScriptu

Pro příjem zpráv je k dispozici metoda `parse`. Ta je volána přímo z kódu v C#. Funkcí této metody je dekódovat Base64, převést text na JSON, dle parametru `action` zavolat příslušnou funkci a předat jí obsah v parametru `data` (ukázka 6.7).

```
1 var json = JSON.parse(Base64.decode(data));
2 this.actions[json.Action](json.Data);
```

Ukázka kódu 6.7: Přijetí zprávy v JavaScriptu

Zasílané zprávy

PageChange Při změně stránky zasílá JavaScript zprávu `PageChange`. Pomocí této zprávy informuje o aktuální stránce, celkovém počtu stránek a aktuální pozici. C# na ni může reagovat tak, že si uloží aktuální pozici (aby při příštím otevření knihy mohla být otevřena na stejném místě) nebo začne tuto pozici synchronizovat. Také díky této zprávě je možné zobrazovat v horní liště čtečky aktuální stránku a celkový počet stránek v textu.

NextChapterRequest Pokud je uživatel na poslední stránce zobrazeného textu a vyžádá si další stránku, tak JavaScript pošle zprávu `NextChapterRequest`. Kód v C# si pamatuje aktuální kapitolu a díky tomu může zpět zaslat následující kapitolu pomocí zprávy `loadHtml`, pokud taková kapitola existuje.

PrevChapterRequest Zpráva `PrevChapterRequest` má podobnou funkci jako zpráva `NextChapterRequest`. Pokud je uživatel na první stránce a vyžádá si předchozí stránku, tak se odešle tato zpráva. Odpovědí ze strany C# je zpráva `loadHtml` s předchozí kapitolou, pokud existuje (pokud uživatel nemá zobrazenou první kapitolu knihy).

OpenQuickPanelRequest Zpráva `OpenQuickPanelRequest` je zasílána v případě, kdy si uživatel gestem vyžádá zobrazení rychlého panelu (ten zobrazuje obsah, záložky a rychlé nastavení). O zachycení uživatelských gest se stará JavaScript a ten informuje kód v C#, který následně rychlý panel otevře.

ChapterRequest Zpráva `ChapterRequest` se zasílá, pokud uživatel v textu klikne na odkaz, který vede na jinou kapitolu v knize. Toto se stane například v případě, kdy je v textu odkaz na vysvětlivky. Obsahem zprávy je název dané kapitoly. Odpovědí ze strany C# je pak zpráva `loadHtml`, která obsahuje požadovanou kapitolu.

OpenUrl Zpráva `OpenUrl` se zasílá, pokud uživatel v načteném HTML klikne na externí odkaz. Parametrem této zprávy je daný odkaz. Kód v C# po přijetí této zprávy zavolá funkci `Device.OpenUri()`, která v závislosti na aktuální platformě zpracuje otevření odkazu.

PanEvent Posunem po okraji obrazovky ve svislém směru lze ovládat jas displeje. Gesto posunu zaznamenává JavaScript a zasílá zprávu `PanEvent`, která obsahuje souřadnice, na kterých k události došlo. Kód v C# pak na základě těchto souřadnic a uživatelského nastavení může změnit jas displeje.

6.6 Synchronizace

Vývoj aplikace byl od začátku zamýšlen tak, aby byla podporována synchronizace mezi všemi zařízeními, které čtenář používá. Dostupná je synchronizace postupu čtení a synchronizace záložek v knize. V rámci budoucího rozvoje by také bylo možné přidat synchronizaci celých knížek. Čtenář by si na jednom zařízení přidal knihu do knihovny a ta by se automaticky stáhla do jeho dalších zařízení.

Synchronizace neprobíhá mezi zařízeními přímo, ale skrze sdílené úložiště, které je přístupné ze všech zařízení. Jednotlivá zařízení tedy spolu nijak přímo nekomunikují, ale pouze si ukládají data na společné místo a odtud je také čtou. Tyto data

jsou ve formátu JSON a mohou být ukládána kamkoliv na internet. V aplikaci je připravena podpora pro Dropbox a cloudovou databázi Firebase. Podmínkou fungující synchronizace tedy je, aby bylo zařízení připojeno k internetu a aby se uživatel ze všech svých zařízení přihlásil ke stejnému úložišti.

Každé zařízení má také svůj název, který si může uživatel libovolně nastavit. Tento název se přenáší v synchronizacích a díky němu lze poznat, jestli daná synchronizace pochází z aktuálního zařízení či nikoliv. Využití tohoto názvu je například v synchronizaci postupu čtení. Pokud si čtečka v úložišti přečte svoji vlastní pozici, nemusí ji čtenáři ani nabízet.

Pro fungování synchronizace jsou důležité dvě komponenty aplikace. Jedná se o třídu `SyncService` a rozhraní `ICloudStorageService`. Zodpovědností třídy `SyncService` je generovat data k uložení, porovnávat synchronizace a rozhodovat o použití jednotlivých dat či kontrolovat správné nastavení synchronizace. Uložení a načtení dat z konkrétního úložiště mají na starosti implementace rozhraní `ICloudStorageService`. Ty pouze dostanou cestu a data, která načtou či uloží. Samotná logika synchronizace je tedy naprogramována nezávisle na konkrétním úložišti. Díky tomuto rozdělení je možné do aplikace jednoduše přidávat další služby, pomocí kterých lze synchronizaci realizovat. Stačí implementovat rozhraní `ICloudStorageService`, ve kterém je potřeba ukládat, načítat a mazat data ve formátu JSON.

6.6.1 Připojení k synchronizační službě

Aplikace nyní podporuje připojení na Dropbox a Firebase. Aby mohl uživatel využívat synchronizaci přes tyto služby, je třeba se do nich nejprve přihlásit.

Dropbox

Dropbox je webové úložiště souborů. K přístupu na něj je využíván oficiální balíček Dropbox.Api[5]. Aby bylo možné komunikovat s rozhraním Dropboxu, je nutné si nejprve na jeho webových stránkách vytvořit aplikaci. U ní je nutné nastavit, jestli má mít přístup k celému Dropboxu uživatele a nebo pouze k vyhrazenému prostoru. V takovém případě bude uživateli na jeho Dropboxu automaticky vytvořena speciální složka a aplikace bude mít přístup pouze do ní. Varianta s použitím vyhrazeného prostoru je využita v této čtečce knih. Po nastavení aplikace vygeneruje Dropbox

[5] https://www.nuget.org/packages/Dropbox.Api/

její identifikátor, pomocí kterého je nutné se autentizovat při veškeré komunikaci s rozhraním Dropboxu.

Připojení uživatele k Dropboxu probíhá pomocí protokolu OAuth2[6]. Nejprve je přesměrován na speciální stránku aplikace, kde je zobrazen přihlašovací formulář k Dropboxu. Uživatel zadá svoje přihlašovací údaje a ty jsou spolu s identifikátorem aplikace odeslány Dropboxu. Pokud je přihlášení úspěšné, je aplikaci vrácen access token, pomocí kterého je možné přistupovat na uživatelův Dropbox. Aplikace si neuchovává uživatelský login ani heslo, ale pouze získaný access token. Veškerá data jsou ukládána na uživatelův Dropbox do vyhrazené složky. Každý uživatel má tak všechny svoje data odděleny od dat jiných uživatelů.

Firebase

Firebase je cloudová JSON databáze od společnosti Google. Pro práci s ní je používán balíček Firebase.Xamarin[7]. Podobně jako u Dropboxu, je nutné si nejprve ve webovém rozhraní Firebase vytvořit aplikaci. Následně je vygenerována její adresa a klíč. Pomocí těchto dat lze poté s databází Firebase komunikovat.

Čtenář v aplikaci zadá e-mail a heslo, pomocí kterého se bude přihlašovat do databáze. Aplikace se připojí k databázi Firebase a zjistí, zda již účet s daným e-mailem existuje. Pokud neexistuje, tak ho vytvoří. Následně se pokusí k účtu přihlásit a pokud bude přihlášení úspěšné, může probíhat synchronizace. V aplikaci zůstane uložen uživatelský e-mail a heslo, protože je nutné tyto údaje zasílat při každém připojení k databázi. Firebase vygeneruje pro uživatele unikátní identifikátor, díky kterému jsou odděleny data jednotlivých uživatelů. Ty jsou sice uloženy v jedné společné JSON struktuře, ale pro každý uživatelský identifikátor existuje vlastní uzel, který obsahuje všechny data daného uživatele. Aplikace si potom stahuje pouze data, která jsou potomky tohoto uzlu. Na obrázku 6.3 je zobrazena ukázka této struktury. První potomek uzlu *users* je uživatelský identifikátor, pod kterým jsou uložena další data.

6.6.2 Synchronizace postupu čtení

Synchronizace s postupem čtení je reprezentována JSON strukturou, která se skládá ze třech částí. Jedná se o název zařízení, které vygenerovalo pozici (D), dále o číslo

[6] https://oauth.net/2/

[7] https://www.nuget.org/packages/Firebase.Xamarin/

Obrázek 6.3: Struktura dat v databázi Firebase

kapitoly (S) a pozici v rámci kapitoly (P). V ukázce 6.8 je taková synchronizace zobrazena.

```
1 {
2     "D": "Computer",
3     "S": 15,
4     "P": 563
5 }
```

Ukázka kódu 6.8: JSON s postupem čtení

Pro správné fungování synchronizace je potřeba do sdíleného úložiště ukládat aktuální polohu v knize a také si ji odtud načítat. K tomu slouží metody `SaveProgress` a `LoadProgress`, které jsou součástí třídy `SyncService`.

Pro uložení aktuální pozice do úložiště je určena metoda `SaveProgress`. Ta si nejprve připraví data k uložení, vygeneruje cestu pro uložení synchronizace a pomocí instance implementace rozhraní `ICloudStorageService` synchronizaci uloží. V případě

použití Dropboxu bude soubor s pozicí uložen pod názvem *progress.json* v adresáři, který je pojmenován dle identifikátoru knihy (viz kapitola 6.1). Při použití databáze Firebase bude tato cesta převedena do struktury JSON. Bude použit uzel pojmenovaný dle identifikátoru knihy s potomkem *progress*. Pozice je uložena při zavření knihy nebo zavření celé aplikace. Dále probíhá ukládání na pozadí každou minutu. Není tedy nutné knihu zavírat, aby se provedla synchronizace její pozice.

K načítání synchronizované pozice slouží metoda `LoadProgress`. Ta si vygeneruje cestu, na které očekává synchronizovanou pozici. Bude se jednat o cestu skládající se z identifikátoru knihy a uzlu *progress*. Následně zavolá nad instanci rozhraní `IClousStorageService` metodu `LoadJson`, která vrátí JSON nacházející se na této cestě. Načtení pozice se volá při otevření knihy a následně na pozadí každou minutu. Aplikace si pamatuje poslední získanou pozici ze synchronizace a při načtení nové pozice si zkontroluje, že se nejedná o předešlou pozici. Dále kontroluje, že se nejedná o současnou pozici na aktuálním zařízení a zda jde o synchronizaci z jiného zařízení. Pokud jsou tyto podmínky splněny, čtenáři je zobrazen modální dialog s otázkou, zda chce načíst pozici. Pokud souhlasí, je na ni přesunut. Aplikace si zapamatuje získanou pozici a při další synchronizaci se již nebude na tuto pozici uživatele dotazovat.

6.6.3 Synchronizace záložek

Kromě pozice v knize lze synchronizovat také záložky. V podstatě se jedná o pozici v knize, ke které je navíc přidaný název. Dále u knihy může být více záložek a lze je mazat. Synchronizace záložek jsou uloženy v JSON struktuře, která je zobrazena v ukázce 6.9. Je zde identifikátor záložky (I), příznak smazání (D), číslo kapitoly (S), pozice v rámci kapitoly (P), název záložky (N) a datum poslední změny (C). Každá záložka má svůj unikátní identifikátor, který je při použití Dropboxu využit jako název souboru, kde je uložen JSON. Všechny záložky jsou pak uloženy ve složce pojmenované dle identifikátoru knihy a v podsložce *bookmarks*. Pokud je k synchronizaci využita databáze Firebase, jsou záložky uloženy v JSON struktuře v uzlu knihy a potomku s názvem *bookmarks*. Zde jsou jednotlivé záložky v uzlech pojmenovaných dle identifikátoru záložky.

```
{
    "I": 36341323,
    "D": false,
    "S": 16,
    "P": 0,
    "N": "25.02.2018 15:48:43",
    "C": "2018-02-25T14:48:43.222273"
}
```

Ukázka kódu 6.9: JSON se synchronizací záložky

Stejně jako u synchronizace pozice, i pro záložky se používá třída `SyncService`. Důležitou metodou je `SynchronizeBookmarks`, která má za úkol stáhnout všechny záložky ze sdíleného úložiště a porovnat je se záložkami v zařízení. Jako stejné záložky jsou považovány takové, které mají shodný identifikátor. Metoda při porovnání záložek rozhodne, které je třeba přidat, upravit nebo smazat v zařízení a které ve sdíleném úložišti.

Protože k porovnávání záložek dochází na uživatelském zařízení, musí se vždy z úložiště stáhnout všechny. K této synchronizaci dochází periodicky každou minutu a vždy by se z úložiště přenášely všechny záložky, i když by to nemuselo být nutné. Aby bylo takovým přenosům dat zabráněno, tak se do úložiště ukládá čas, kdy došlo k poslední změně nějaké záložky. Ten je uložen buď v souboru *bookmarkslastchange.json* nebo ve stejnojmenném uzlu v JSON struktuře. Kdykoliv je uložena jakákoliv změna některé záložky, uloží se aktuální datum. Čas poslední synchronizace se ukládá také v zařízení a proto stačí každou minutu z úložiště pouze stahovat čas poslední změny. Pokud je novější než čas uložený v zařízení, pak došlo k synchronizaci ze strany jiného zařízení a je nutné provést synchronizaci i na aktuálním zařízení. Dojde ke stažení všech záložek z úložiště, k jejich porovnání a k synchronizaci. V opačném případě se žádná další data přenášet nebudou.

U každé záložky se také uchovává datum poslední změny. Pokud si zařízení z úložiště stáhne záložku, která má například jiné jméno než v zařízení, podle data změny lze určit, která varianta je aktuálnější. Pokud je u záložky v úložišti novější datum než u záložky v zařízení, je záložka v zařízení upravena. Naopak pokud je novější datum u záložky v zařízení, bude upravena záložka v úložišti.

Pokud si uživatel smaže nějakou záložku, není ve skutečnosti zcela odstraněna, ale je u ni pouze nastaven příznak smazání. Taková záložka se potom nikde nezobra-

zuje, ale dál se přenáší v synchronizaci, aby si ji i ostatní zařízení označila jako smazanou. Důvodem tohoto způsobu mazání záložek je jednoduchost zaslání informace o smazání ostatním zařízením. Kdyby záložka byla sesynchronizována na dvou či více zařízeních a na jednom z nich byla skutečně odstraněna, další zařízení by si nemělo jak vyhodnotit chybějící záložku. Mohlo by se jednat o odstraněnou záložku, kterou si má smazat ze zařízení a nebo by se naopak mohlo jednat o novu záložku, kterou musí poslat do úložiště. Tento problém je díky tomuto způsobu mazání vyřešen. Zařízení si stáhne změnu záložky stejně, jako kdyby se jednalo například o změnu jejího názvu. Ke skutečnému odstranění záložek dojde až při smazání knihy z knihovny.

Další metodou pro synchronizaci záložek je `SaveBookmark`. Ta porovnání záložek neprovádí, ale pouze uloží jednu záložku do úložiště. K volání dochází při přidání nové záložky, změně názvu nebo jejím smazání. Kromě uložení samotné záložky je také potřeba v cloudovém úložišti upravit čas poslední synchronizace záložek, aby si ostatní zařízení tuto záložku v rámci pravidelné synchronizace stáhla.

7 Výsledky

7.1 Zhodnocení splnění požadavků z analýzy

Vytvořená aplikace dle požadavků v analýze podporuje formát EPUB. Navíc podporuje také otevření souborů HTML a TXT a je rozšiřitelná o podporu dalších formátů. Do knihy lze přidávat záložky, které se spolu s postupem čtení synchronizují. K synchronizaci lze využít čtenářův Dropbox nebo cloudovou databázi Firebase. K jejímu použití se stačí na všech zařízeních přihlásit se stejným e-mailem a heslem. Aplikace je také rozšiřitelná o další úložiště, pomocí kterých by šlo synchronizaci realizovat. V aplikaci je obsaženo uživatelské nastavení, kde si může čtenář upravovat vzhled otevření knihy, ovládání čtečky či nastavení synchronizace. Mezi možnosti ovládání patří například posun na stránky pomocí tlačítek pro ovládání hlasitosti nebo změna jasu displeje pomocí posunu prstem po okraji displeje.

Aplikace byla vytvořena pro platformy Android a Windows. Je možné ji spouštět na desktopu jako UWP aplikaci, pro mobilní Windows není optimalizována. Platforma Android byla vybrána díky přibližně 70% podílu na trhu chytrých telefonů [18]. Desktopová platforma UWP byla pro tvorbu aplikace vybrána díky jednoduché tvorbě těchto aplikací pomocí nástroje Xamarin a také díky rozšíření platformy Windows. Podíl tohoto systému činí přibližně 82 % [19], ale aplikaci lze spouštět pouze na systému Windows 10, jehož podíl činí v rámci platformy Windows zhruba 43 % [20]. Uživatel tedy může používat aplikaci a využívat synchronizaci na mobilním telefonu a počítači, pokud používá tyto dvě platformy.

Zdrojové kódy aplikace jsou umístěny na GitHubu[1]. Výsledná aplikace byla uveřejněna v obchodě Google Play[2] a Microsoft Store[3].

Vzhledem k umístění čtečky do celosvětových obchodů s aplikacemi bylo rozhraní

[1] https://github.com/bares43/thesis-ebook-reader

[2] https://play.google.com/store/apps/details?id=cz.janbares.onesyncreader

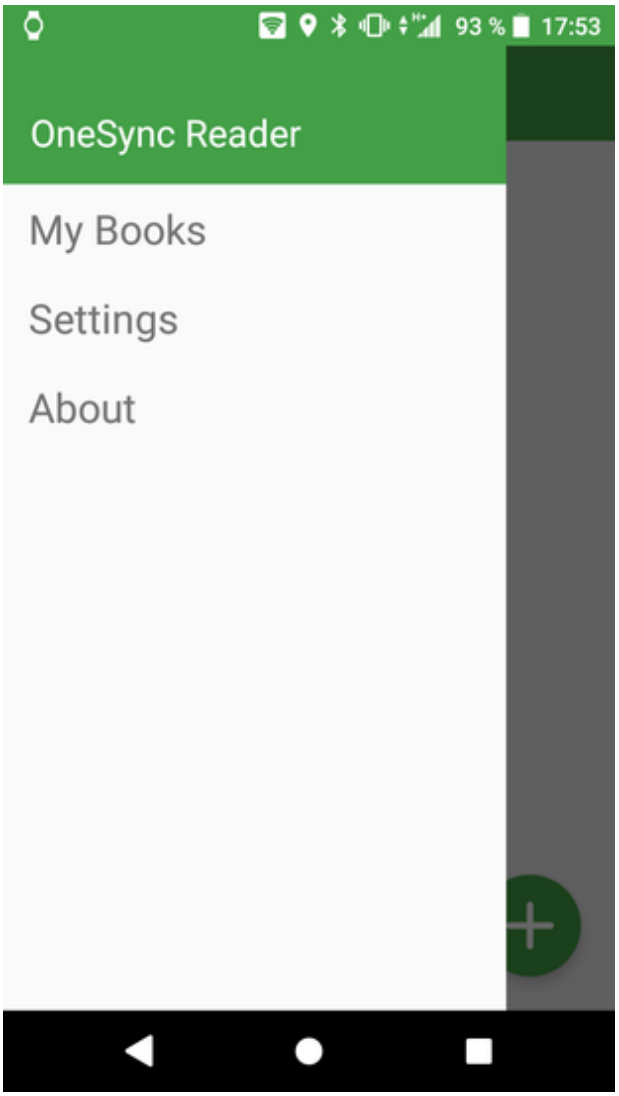
[3] https://www.microsoft.com/cs-cz/store/p/onesync-reader/9pltc7z5g1b4

aplikace vytvořeno v angličtině. Aplikace neumožňuje přidávání překladů do jiných jazyků, ale v budoucnu by o tuto možnost mohla být rozšířena.

Lze konstatovat, že všechny požadavky na aplikaci byly splněny. Aplikace má dokonce oproti požadavkům několik funkcí navíc, jako například podporu HTML a TXT či ovládání jasu displeje.

7.2 Uživatelský popis aplikace

Aplikace je rozdělena na několik stránek, mezi kterými může uživatel přecházet. Na zařízení Android je k dispozici nabídka vyjíždějící ze strany (obrázek 7.1) a v UWP aplikaci se nabídka zobrazuje v horní liště.



Obrázek 7.1: Navigace na zařízení Android

7.2.1 Hlavní stránka a knihovna

Hlavní stránka (obrázky 7.2 a 7.3) je vstupní místo do celé aplikace. Jedná se o první obrazovku, kterou uživatel po otevření aplikace uvidí. Na této stránce je uživatelova knihovna. Je možné zde do aplikace přidat novou knihu. V UWP aplikaci k tomu slouží obyčejné tlačítko a v Android aplikaci je k dispozici plovoucí tlačítko, takzvaný *floating action button*. Dále jsou zde zobrazeny všechny knihy, které má čtenář uloženy v knihovně. Knihy jsou vedle sebe zobrazeny horizontálně a lze se mezi nimi posouvat. U každé knihy je ikonka koše, pomocí které lze knihu z aplikace odstranit. Uživatel bude dotázán, zda chce knihu odstranit pouze z aktuálního zařízení nebo zda chce odstranit také veškeré synchronizace ze sdíleného úložiště. Pokud si čtenář odstraní knihu pouze z aktuálního zařízení, při opětovném přidání do aplikace se mu zpět načte jeho pozice a uložené záložky.

7.2.2 Čtečka

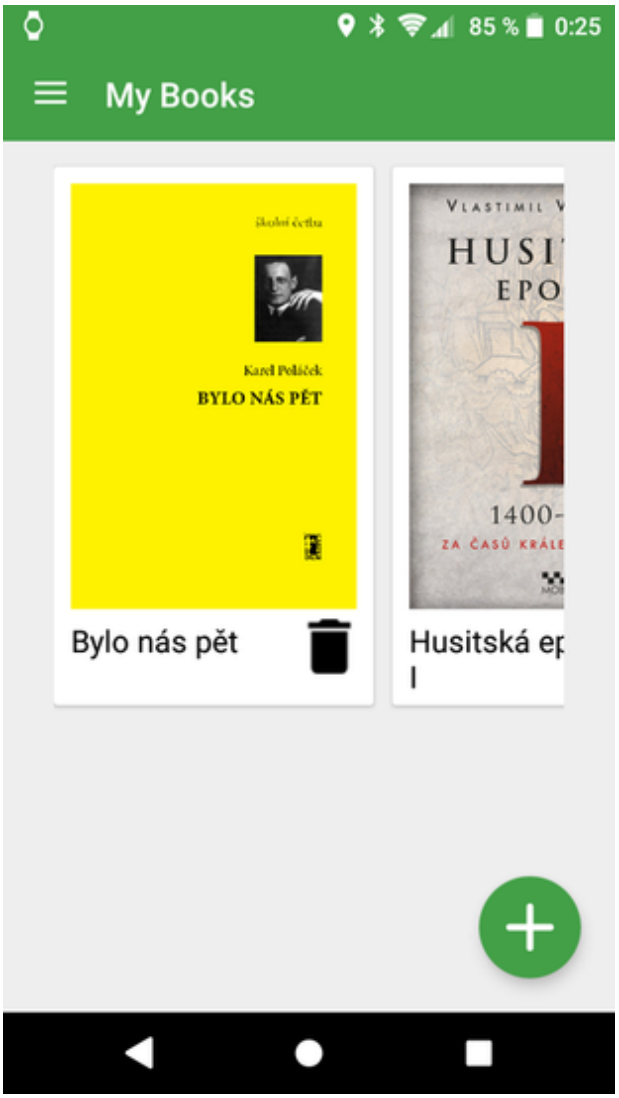
Když uživatel v aplikaci otevře knihu, zobrazí se mu na speciální stránce se čtečkou. Ta se v závislosti na uživatelském nastavení může otevřít v režimu fullscreen, tedy bude zobrazena přes celou obrazovku zařízení. Může se také zobrazovat v nočním nebo denním režimu. Na této stránce je zobrazen pouze text knihy a informační panel.

Informační panel

U horního okraje stránky s otevřenou knihou je zobrazen informační panel. Ten čtenáři zobrazuje číslo aktuální stránky a jejich celkový počet v kapitole. Dále zobrazuje aktuální čas a na telefonu s Androidem také stav baterie zařízení. Ten je znázorněn pomocí pěti ikonek, které se zobrazují v závislosti na procentuálním stavu baterie. Panel má průhledné pozadí, získá tedy barvu pozadí čtečky, a vizuálně vypadá jako součást otevřené knihy.

Rychlý panel

Další částí stránky se čtečkou je takzvaný rychlý panel. Ten lze zobrazit dvojím způsobem. První způsob je dvojité poklepání na displej telefonu či poklepání myší (takzvaný doubleclick). Druhou možností je dlouhým stiskem prstem nebo myší. Panel



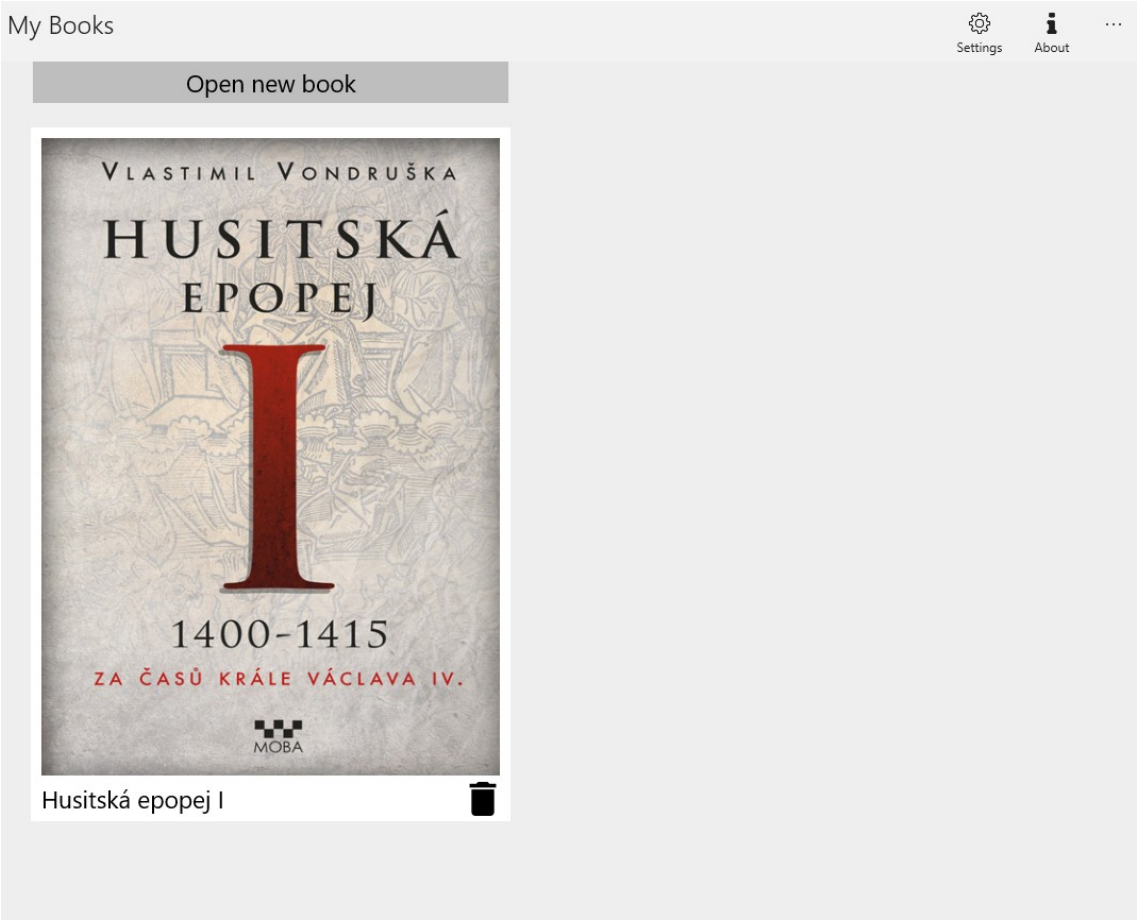
Obrázek 7.2: Hlavní stránka aplikace a knihovna na zařízení Android

lze následně zavřít ťuknutím na tlačítko zpět nebo někam mimo oblast tohoto panelu. Podobně, jako lze zavřít modální okno na některých webových stránkách.

Rychlý panel se skládá ze dvou částí. Nahoře je několik vybraných možností nastavení čtečky. Jedná se o změnu jasu, nastavení velikosti písma a velikosti odsazení textu. Ve spodní části je možné se přepínat mezi obsahem knihy a záložkami. Lze přidávat nové záložky či mazat nebo upravovat stávající. Po poklepání na název kapitoly či záložky se zobrazí příslušná část textu knihy. Tento panel tedy slouží k navigaci čtenáře v otevřené knize. Na obrázku 7.4 je tento rychlý panel zobrazen.

7.2.3 Nastavení

Nastavení aplikace je rozděleno do čtyř tématických sekcí. Ty jsou reprezentovány pomocí záložek, mezi kterými se může uživatel přepínat. Záložku lze otevřít kliknutí

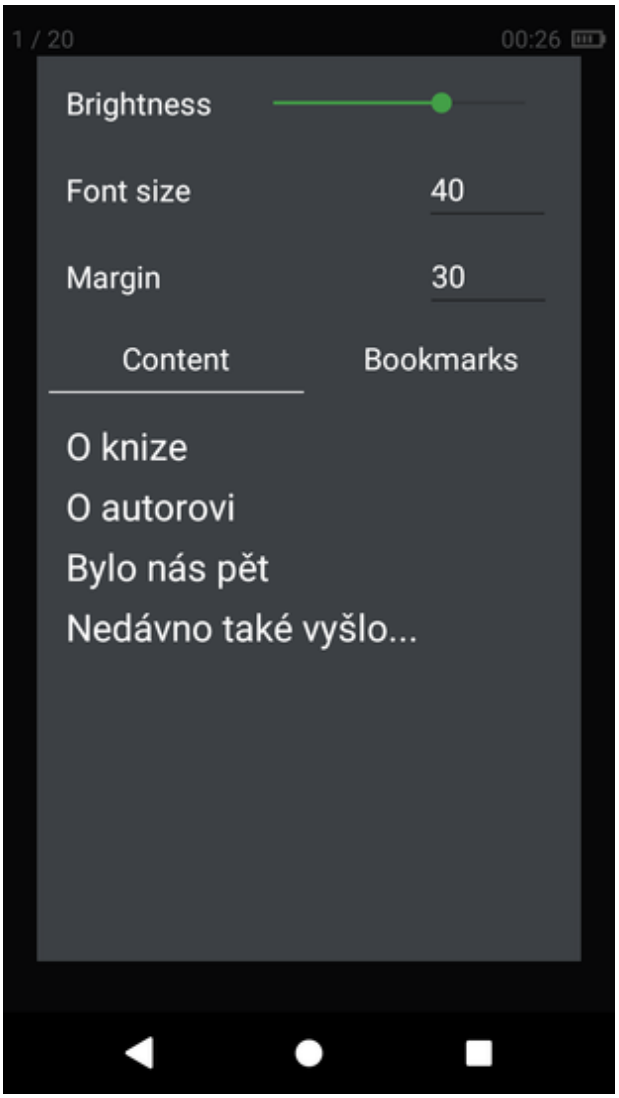


Obrázek 7.3: Hlavní stránka aplikace a knihovna v UWP aplikaci

na její název, případně posunem prstem na displeji do stran na zařízení se systémem Android. Veškeré nastavení se ukládá ihned a je perzistentní. Aplikace si tedy při příštím spuštění nastavené preference pamatuje. Podoba stránky s nastavením na zařízení Android je zobrazena na obrázku 7.5.

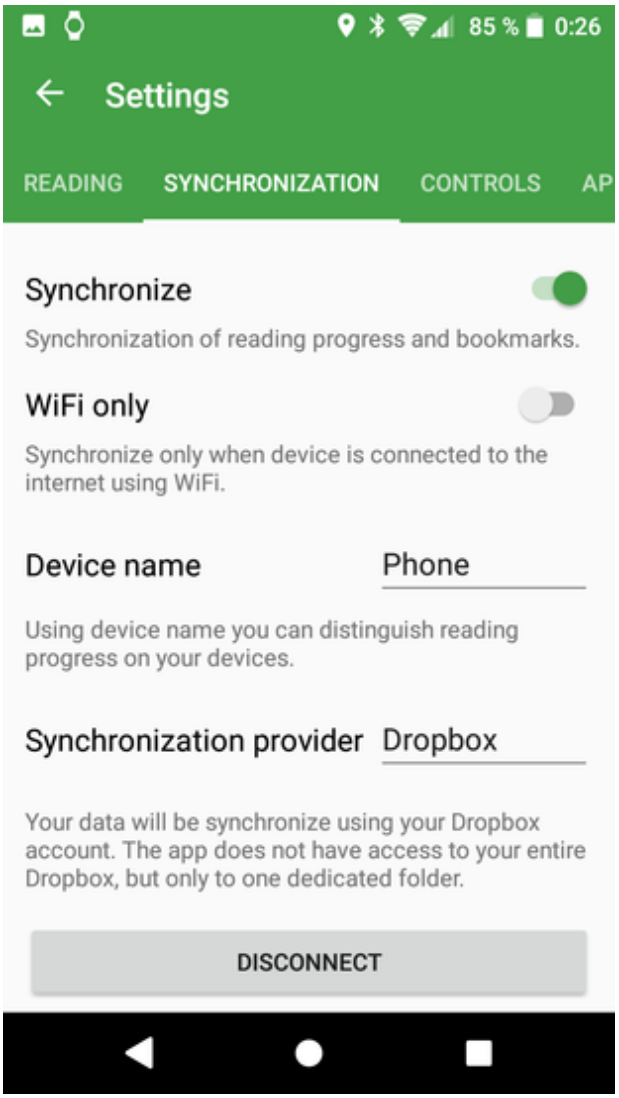
První záložka se jmenuje **reading**. Zde lze nastavit vzhled otevřené knihy, jako velikost textu, velikost odsazení textu nebo rychlost animace při posunu stránek. Dále je možné si zde zapnout noční mód čtečky (text bude zobrazen bílým písmem na tmavém pozadí) nebo režim fullscreen. To znamená, že otevřená kniha se zobrazí přes celou obrazovku, nebude tedy vidět status bar na zařízení Android a hlavní panel na počítačích s Windows.

Na druhé záložce s názvem **synchronization** je soustředěno veškeré nastavení synchronizace čtečky. Samotná synchronizace zde lze povolit nebo zakázat. Dále je možné povolit synchronizaci pouze pokud je zařízení připojeno na internet pomocí WiFi. To je vhodné pro případy, kdy má uživatel omezený datový tarif a chce zredukovat datové přenosy, které nejsou nezbytné. Uživatel si také může nastavit název



Obrázek 7.4: Otevřený rychlý panel na zařízení Android

aktuálního zařízení. Ten se přenáší spolu se synchronizacemi a při načtení pozice v knize z jiného zařízení bude uživatel dotázán, zda ji chce z takto pojmenovaného zařízení načíst. Uživatel díky tomu ví, z jakého zařízení daná pozice pochází. V nastavení je dále možné nakonfigurovat připojení ke sdílenému úložišti, pomocí kterého bude synchronizace probíhat. Uživatel si může vybrat ze dvou možností: Dropbox nebo e-mail. Pokud si vybere Dropbox, zobrazí se tlačítko pro připojení k němu. Po kliku na něj je uživatel přesměrován na přihlašovací stránku Dropboxu, kde zadá e-mail a heslo. Když je uživatel úspěšně přihlášen, začne se mu zobrazovat tlačítko pro odhlášení. Druhou možností je zvolit jako poskytovatele synchronizace e-mail a heslo. Synchronizace pak bude probíhat s využitím cloudové databáze Firebase. Uživateli jsou zobrazeny políčka pro zadání e-mailu a hesla, se kterými se aplikace k databázi bude přihlašovat. Není nutné se nijak registrovat, stačí na všech svých



Obrázek 7.5: Stránka s nastavením na zařízení Android

zařízeních zadat stejný e-mail a heslo. Aplikace sama rozpozná, zda má e-mail zaregistrovat nebo zda se stačí pouze přihlásit. Pokud uživatel zapomene heslo, může si z aplikace zažádat o jeho obnovu. Na e-mail mu je poté službou Firebase zaslán odkaz, na kterém si heslo může obnovit.

Třetí záložka se jmenuje **controls**. Zde je možné nastavovat ovládání čtečky s otevřenou knihou. První dostupná funkcionalita je posun na další stránku při ťuknutí kdekoliv na displej. Při běžném režimu ťuknutí na pravou polovinu displeje znamená posun na následující stránku a na levou polovinu posun na předchozí stránku. To nemusí být pro čtenáře příjemné, pokud drží mobil v levé ruce a nedosáhne na pravou polovinu displeje. Při aktivování této funkce je možné na displej ťuknout kdekoliv, třeba i v levé polovině, a kniha se posune o stránku vpřed. Pro posun na předchozí stránku lze využít gesto *swipe*, tedy posun prstem z levé poloviny obrazovky

do pravé. Dále je možné povolit ovládání čtečky pomocí tlačítek pro ovládání hlasitosti. Poté bude tlačítko pro snížení hlasitosti posunovat knihu o stránku zpět a tlačítko pro zvýšení hlasitosti o stránku vpřed. Během čtení tedy nebude možné ovládat hlasitost telefonu a protože ne každý uživatel může tuto funkci vyžadovat, je volitelná. Další možností je nastavit funkci změny jasu displeje, pomocí přejíždění prstem po okraji displeje ve vertikálním směru. Toto ovládání je možné nastavit na levý či pravý okraj displeje nebo zcela vypnout. Poslední možnost ovládání je gesto *swipe* pomocí dvou prstů. To slouží pro přesun mezi kapitolami, čtenáře přesune na předchozí nebo následující kapitolu textu.

Poslední záložka v nastavení se jmenuje **application** a slouží pro obecné nastavení aplikace. V současné době zde lze povolit nebo zakázat zasílání anonymních dat o používání čtečky. Tyto data jsou zasílány do služby Microsoft App Center, kde mohou být vyhodnocena. Je zde také popsáno, co je obsahem těchto dat. Zaznamenává se úspěšné a neúspěšné přihlášení k Dropboxu či službě Firebase. V těchto datech nejsou obsaženy žádné citlivé údaje, jako login nebo heslo. Pokud se v aplikaci nepodaří otevřít nějakou knihu, zaznamená se také název souboru, který se uživatel pokusil otevřít. Tyto údaje slouží ke snadnému zjištění problémů s otevřením některých knih v aplikaci. Díky tomu bude možné případně problémy vyřešit a opravit.

Obrázek 7.6: Stránka s nastavením v UWP aplikaci

Celá stránka s nastavením je interaktivní a přizpůsobuje se nastaveným volbám. Například pokud si uživatel vypne synchronizaci dat, nebudou se na stránce s nastavením synchronizace zobrazovat žádné další možnosti, protože by neměly žádný význam. Stejně tak se zobrazí políčka pro přihlášení k Dropboxu pouze pokud si uživatel zvolí jako poskytovatele Dropbox. Podobně fungují také políčka pro zadání přístupových údajů k službě Firebase. Nastavení je také přizpůsobeno zařízení, na kterém aplikace běží. Například na počítači s Windows chybí možnost ovládání čtečky pomocí tlačítek pro ovládání hlasitosti (obrázek 7.6), protože využití této funkce je zejména na mobilu.

7.2.4 O aplikaci

Stránka *O aplikaci* slouží jako prostor, kde lze zobrazit informace o verzi aplikace, kontakt na autora, ale také licence k použitým komponentám třetích stran a podobně. V budoucnu by mohla být tato stránka rozšířena například o nápovědu k ovládání aplikace.

8 Závěr

Cíl této práce byl splněn a aplikace pro čtení elektronických knih byla úspěšně vytvořena. Je použitelná pro čtení knih v rozšířeném formátu EPUB a podporuje synchronizaci skrze Dropbox nebo databázi Firebase. Hotová aplikace byla také umístěna do Google Play a Microsoft Store, takže si ji může kdokoliv stáhnout a začít používat.

Tvorba mobilní a desktopové aplikace je velmi komplexní záležitost. Neustále se objevují nové požadavky a funkce, o které lze aplikaci rozšířit. Jedna z nejzásadnějších funkcí je rozšíření podpory o **mobilní platformu iOS**, která nyní podporována není. Vzhledem k tomu, že je aplikace vytvořena pomocí nástroje Xamarin, nebylo by toto rozšíření velký problém. Další možné rozšíření je přidání podpory dalších služeb pro synchronizaci, například **Google Drive**, **OneDrive** či dalších. Také by byla zajímavá funkce, která by čtenáři zobrazovala **čas do dočtení knihy**. Nejprve by bylo nutné měřit, kolik čtenář průměrně přečte znaků za jednu minutu. Poté by se dle počtu znaků v knize dopočítal a zobrazil čas, který ještě bude čtenář knihu číst.

Aplikace má otevřený zdrojový kód a nové funkce tak mohou přidávat samotní uživatelé.

Literatura

[1] Publisher Book Sales Were $11.13 Billion in the First Three Quarters of 2016. ASSOCIATION OF AMERICAN PUBLISHERS [online]. 2017 [cit. 2018-03-31]. Dostupné z: http://newsroom.publishers.org/publisher-book-sales-were-1113-billion-in-the-first-three-quarters-of-2016/

[2] VLČEK, Jiří. E-kniha nevzkvétá. H7O - vzorec pro literaturu [online]. 2018 [cit. 2018-03-31]. Dostupné z: http://www.h7o.cz/e-kniha-nevzkveta/

[3] Formáty elektronických knih a ochrana proti kopírování. Alza.cz [online]. 2018 [cit. 2018-03-31]. Dostupné z: https://www.alza.cz/formaty-elektronickych-knih-a-drm

[4] EPUB. International Digital Publishing Forum [online]. 2017 [cit. 2018-01-17]. Dostupné z: http://idpf.org/epub

[5] IDPF Has Combined With W3C. International Digital Publishing Forum [online]. 2017 [cit. 2018-01-17]. Dostupné z: http://idpf.org/news/idpf-has-combined-with-w3c

[6] SHARP, John. Microsoft Visual C# Step by Step. 8th Edition. Redmond: Microsoft Press, 2015. ISBN 978-1-5093-0104-1.

[7] The history of C#. Microsoft Docs [online]. 2017 [cit. 2018-01-21]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-version-history

[8] TIOBE Index for January 2018. TIOBE [online]. 2017 [cit. 2018-01-21]. Dostupné z: https://www.tiobe.com/tiobe-index/

[9] Tour of .NET. Microsoft Docs [online]. 2017 [cit. 2018-01-21]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/standard/tour

[10] PETZOLD, Charles. Creating Mobile Apps with Xamarin.Forms: Cross-platform C# programming for iOS, Android, and Windows. Redmond: Microsoft Press, 2016. ISBN 978-1-5093-0297-0.

[11] PDF Reference and Adobe Extensions to the PDF Specification. Adobe [online]. 2018 [cit. 2018-01-25]. Dostupné z: http://www.adobe.com/devnet/pdf/_reference.html

[12] MOBI. MobileRead [online]. 2018 [cit. 2018-01-25]. Dostupné z: https://wiki.mobileread.com/wiki/MOBI

[13] About SQLite. SQLite [online]. [cit. 2018-03-01]. Dostupné z: https://www.sqlite.org/about.html

[14] EPUB 2.0.1. International Digital Publishing Forum [online]. 2017 [cit. 2018-03-31]. Dostupné z: http://idpf.org/epub/201

[15] Buttons: Floating Action Button. Material Design [online]. [cit. 2018-03-31]. Dostupné z: https://material.io/guidelines/components/buttons-floating-action-button.html

[16] Column-width. MDN Web Docs [online]. 2018 [cit. 2018-04-01]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS/column-width

[17] Will-change. MDN Web Docs [online]. 2018 [cit. 2018-04-01]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS/will-change

[18] Mobile Operating System Market Share Worldwide. StatCounter Global Stats [online]. 2018 [cit. 2018-04-01]. Dostupné z: http://gs.statcounter.com/os-market-share/mobile/worldwide

[19] Desktop Operating System Market Share Worldwide. StatCounter Global Stats [online]. 2018 [cit. 2018-04-01]. Dostupné z: http://gs.statcounter.com/os-market-share/desktop/worldwide

[20] Desktop Windows Version Market Share Worldwide. StatCounter Global Stats [online]. 2018 [cit. 2018-04-01]. Dostupné z: http://gs.statcounter.com/os-version-market-share/windows/desktop/worldwide

Příloha č.1 - Obsah přiloženého CD

Součástí CD je ZIP archiv (*ebook-reader.zip*) obsahující **zdrojové kódy** aplikace. Pro její spuštění je nutné nejprve ve složce *ReaderJS* spustit příkazy `yarn install` a `grunt build`. Následně je potřeba v adresáři *EbookReader/EbookReader* doplnit do souborů *ReaderApp.config* a *ReaderApp.Release.config* připojení k službám Dropbox, Firebase a AppCenter. Poté je možné projekt otevřít v programu Visual Studio, které má nainstalovanou podporu pro Xamarin.

Součástí CD je také **tato práce** ve formátu PDF (*thesis.pdf*).

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2017/2018

Studijní program: Aplikovaná informatika
Forma: Kombinovaná
Obor/komb.: Aplikovaná informatika (ai2-k)

Podklad pro zadání DIPLOMOVÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Bc. Bareš Jan	Gen. Kratochvíla 1008, Červený Kostelec	I1600259

TÉMA ČESKY:

Multiplatformní čtečka elektronických knih se synchronizací

TÉMA ANGLICKY:

Cross platform e-book reader with synchronization

VEDOUCÍ PRÁCE:

Ing. Pavel Kříž, Ph.D. - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Cíl: Cílem je vytvořit pomocí Xamarinu multiplatformní čtečku elektronických knih, která si bude mezi zařízeními uživatele synchronizovat postup čtení. Do textu knihy půjde jednoduše přidat záložky, které budou také synchronizované. Čtečka bude umožňovat změnu vzhledu e-booku (například barva písma nebo pozadí).

Osnova:
1. Úvod
2. Teoertický popis e-booků a jejich formátů
3. Přehled čteček
4. Popis Xamarinu a dalších použitých technologií
5. Analýza, návrh a implementace vlastního řešení
6. Výsledky a testování
7. Závěr

SEZNAM DOPORUČENÉ LITERATURY:

1. Charles Petzold: Creating Mobile Apps with Xamarin.Forms
2. John V. Richardson Jr: eBook readers: user satisfaction and usability issues
3. Peter Gasston: The Book of CSS3, 2nd Edition: A Developer's Guide to the Future of Web Design

Podpis studenta: ..... **Datum:** 11. 10. 17

Podpis vedoucího práce: ..... **Datum:** 11. 10. 17