Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Tvorba multiplatformní aplikace za použití frameworku Ionic

Martin KUSTL

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Martin Kustl

Systémové inženýrství a informatika
Informatika

Název práce

Tvorba multiplatformní aplikace za použití frameworku Ionic

Název anglicky

Multiplatform application development using Ionic framework

Cíle práce

Práce se zabývá problematikou vývoje multiplatformních aplikací s využitím frameworku Ionic. Hlavním cílem práce je navrhnout a s využitím této technologie implementovat mobilní aplikaci pro zaznamenávaní a sdílení textových, či foto poznámek. Dílčím cílem je popsat možnosti využití frameworku Ionic při tvorbě aplikací.

Metodika

Bakalářská práce sestává ze dvou částí – teoretické a praktické. Metodika zpracování teoretické části spočívá ve studiu odborných informačních zdrojů. Na základě syntézy zjištěných poznatků budou formulována teoretická východiska práce. Bude poskytnut přehled možností frameworku Ionic a popsáno jeho použití při vývoji.

V praktické části bude navržena a implementována aplikace, která bude umožňovat uživatelům zaznamenávat, spravovat a sdílet textové, či foto poznámky. Při návrhu a vývoji bude využito standardních metod softwarového inženýrství, pro vývoj bude využit framework Ionic a mj. nástroje Xcode, Android Studio, Capacitor, Firebase, React, Redux, styled-components, GitHub a Visual Studio Code. Aplikace bude prakticky nasazena a otestována, budou shrnuty poznatky získané při její tvorbě a navrženy možnosti jejího případného budoucího rozvoje.

Doporučený rozsah práce

35-40 stran

Klíčová slova

Ionic, Multiplatformní, iOS, Android, Poznámky, Aplikace

Doporučené zdroje informací

Capacitor: Universal Web Applications [online]. Drifty Co., 2019 [cit. 2019-11-17]. Dostupné z: https://capacitor.ionicframework.com/

Ionic – Cross-Platform Mobile App Development [online]. Drifty Co., ©2019 [cit. 2019-11-17]. Dostupné z: https://ionicframework.com/

React: A JavaScript library for building user interfaces [online]. Menlo Park, California, United States: Facebook, ©2019 [cit. 2019-11-17]. Dostupné z: https://reactjs.org/

SIMPSON, Kyle. You Don't Know JS: ES6 & Beyond [online]. 1005 Gravenstein Highway North, Sebastopol, CA 95472.: O'Reilly Media, 2015 [cit. 2019-11-20]. ISBN 978-1491904244. Dostupné z: https://www.amazon.com/You-Dont-Know-JS-Beyond-ebook-dp-B019HRGOPQ/dp/B019HRGOPQ/ref=mt_kindle?_encoding=UTF8&me=&qid=

Předběžný termín obhajoby

2019/20 LS – PEF

Vedoucí práce

Ing. Jiří Brožek, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 19. 2. 2020

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 19. 2. 2020

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 02. 03. 2020

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Tvorba multiplatformní aplikace za použití frameworku Ionic" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci, a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 10.3.2020 _____

Poděkování

Rád bych touto cestou poděkoval vedoucímu práce, panu Ing. Jiřímu Brožkovi, Ph.D., za odborné vedení, ochotu, věcné připomínky a cenné rady, které mi během konzultací věnoval.

Tvorba multiplatformní aplikace za použití frameworku Ionic

Abstrakt

Bakalářská práce se zabývá tématem vývoje multiplatformních aplikací prostřednictvím frameworku Ionic. Hlavním předmětem této práce je s využitím frameworku Ionic implementovat multiplatformní mobilní aplikaci pro zaznamenávání a sdílení textových, či foto poznámek. Dílčím cílem je popsat možnosti využití frameworku Ionic při tvorbě aplikací. V teoretických východiscích práce pojednává o současných možnostech multiplatformního vývoje. Poskytuje tak přehled nejpoužívanějších frameworků pro multipatformní vývoj včetně frameworku Ionic. Na základě tohoto přehledu pak tyto frameworky srovnává. Teoretická východiska jsou završena srovnáním nativního, webového a multiplatformního vývoje. Ve vlastní práci je popsán průběh vývoje multiplatformní aplikace od počátečního souhrnu požadovaných funkcí až po samotnou implementaci.

Klíčová slova: Ionic, Capacitor, React, Multiplatformní, iOS, Android, Student, Poznámky

Multiplatform application development using Ionic framework

Abstract

The bachelor's thesis is focused on multiplatform application development via framework Ionic. Main goal of this thesis is to use framework Ionic for development of multiplatform mobile application for creating text and photo notes. Secondary goal is to describe framework Ionic and its possibilities for application development. The theoretical part of this thesis describes currently existing options for multiplatform application development. It provides overview of most popular frameworks for multiplatform application development including framework Ionic. Previously mentioned overview is concluded by comparison of frameworks for multiplatform application development. The theoretical part is completed by summary of native, multiplatform and web development. The practical part of this thesis describes whole multiplatform application development process from defining application functionality to its final implementation.

Keywords: Ionic, Capacitor, React, Multiplatform, iOS, Android, Student, Notes

Obsah

Seznam obrázků

1 Úvod

V současné době je těžké si představit život bez chytrých zařízení. Chytrý telefon dnes vlastní přes 3,2 miliardy lidí. (1) Což je opravdu nezanedbatelné číslo, které činí trh s mobilními aplikacemi velice zajímavým. Tomuto trhu vládnou dvě platformy, iOS a Android. (2) A tak je pro největší dosah aplikace nutné, aby fungovala na obou těchto platformách. Dříve byl možný pouze nativní vývoj, a tak bylo nutné danou aplikaci vytvořit zvlášť pro obě tyto platformy. Toto řešení je nákladné a časově náročné, což vývojáře vedlo k myšlence vytvoření nástrojů, které by umožňovaly vytvářet aplikace tak, aby stačil pouze jeden kód pro obě platformy. Díky tomu vznikly frameworky jakým je například Ionic, React Native nebo Flutter.

Tato myšlenka napsání jednoho kódu, který bude fungovat napříč platformami, se mi zdá velice zajímavá. Proto jsem se rozhodl prozkoumat téma multiplatformního vývoje aplikací a následně vytvořit aplikaci za pomocí frameworku Ionic.

V teoretických východiscích budou nejdříve uvedeny veškeré možnosti tvorby mobilní aplikace. Bude popsáno, jak se tvoří nativní aplikace. Dále budou zmíněny progresivní webové aplikace, které dokáží do jisté míry nahradit mobilní aplikace a v neposlední řadě bude popsán multiplatformní vývoj.

V praktické části bude vytvořena multiplatformní mobilní aplikace pro vytváření a sdílení poznámek za pomoci frameworku Ionic, na které bude demonstrováno, co takový vývoj obnáší.

2 Cíl práce a metodika

2.1 Cíl práce

Práce se zabývá problematikou vývoje multiplatformních aplikací. Zejména se zabývá multiplatformním vývojem aplikací s využitím frameworku Ionic. Hlavním cílem práce jenavrhnout a s využitím této technologie implementovat multiplatformní mobilní aplikaci pro zaznamenávání a sdílení textových či foto poznámek. Dílčím cílem je popsat framework Ionic. Konkrétně jeho možnosti při tvorbě aplikací a popis jeho slabých a silných stránek oproti ostatním frameworkům pro multiplatformní vývoj a poskytnout tak čtenáři jasnou představu o tom, kdy je framework Ionic vhodné použít a kdy nikoliv.

2.2 Metodika

Bakalářská práce sestává ze dvou částí – teoretické a praktické. Metodika zpracování teoretické části spočívá ve studiu odborných informačních zdrojů. Bude poskytnut přehled možností vývoje aplikací. Dále se práce bude blíže zabývat multiplatformním vývojem. Pro detailnější pochopení problematiky multiplatformního vývoje nebude popsán pouze framework Ionic. Společně s frameworkem Ionic budou popsány další populární frameworky umožňující multiplatformní vývoj. Bude popsán princip, jak fungují a budou uvedeny jejich výhody a nevýhody v porovnání s nativním vývojem, případně s ostatními frameworky. V závěru teoretických východisek bude formulováno shrnutí multiplatformních frameworků a následně také shrnutí všech možností vývoje aplikací. V těchto shrnutích bude uvedeno, v jakých případech je vhodná volba multiplatformního vývoje.

V praktické části bude navržena a implementována aplikace, která bude umožňovat studentům zaznamenávat, spravovat a sdílet textové a foto poznámky. Budou jasně definovány požadavky na aplikaci, načež bude uveden grafický návrh aplikace. Následně bude popsáno, jak aplikace dosahuje požadovaných funkcí. Pro vývoj bude využit framework Ionic a mj. nástroje Xcode, Android Studio, Capacitor, Firebase, React, Redux, styled-components, GitHub a Visual Studio Code. Aplikace bude prakticky nasazena a otestována, budou shrnuty poznatky získané při její tvorbě a navrženy možnosti jejího případného budoucího rozvoje.

3 Teoretická východiska

3.1 Nativní vývoj mobilních aplikací

První možností pro vývoj mobilních aplikací je samozřejmě nativní vývoj. Tím se rozumí vývoj pro konkrétní platformu. Nativní vývoj tak umožňuje plný přístup k hardwaru a softwaru dané platformy čili například ke kameře, mikrofonu, nebo úložišti. Každá platforma má svůj programovací jazyk.

3.1.1 iOS

Vytváření aplikací pro platformu iOS obnáší znalost programovacích jazyků Swift nebo Objective-C. iOS má ještě další specifika. Prvním je vývojové prostředí, iOS aplikace je totiž možné vytvářet pouze ve vývojovém prostředí Xcode. Druhým je také fakt, že iOS aplikace je možné vytvářet pouze na operačním systému MacOS. (3)

3.1.2 Android

Pro vytváření Android aplikací je nutné znát programovací jazyk Kotlin nebo Java. Také je nutné mít Android Software Development Kit. Ten je dostupný zdarma ke stažení. Dále je samozřejmě nutné mít vývojové prostředí jakým je například IntelliJ IDEA, Android Studio nebo Eclipse. (4)

3.2 Vývoj progresivních webových aplikací

Progresivní webová aplikace je v podstatě internetová stránka. Nicméně uživatelovi je umožněno si tuto stránku přidat na domovskou obrazovku zařízení. Jakmile je takto aplikace přidána, umožní skrýt uživatelské rozhraní prohlížeče, takže se jeví jako nativní aplikace. Webové prohlížeče dnes také umožňují přístup k nativním funkcím telefonu, takže je možné přistoupit například ke kameře telefonu.

Podstatnou součástí progresivní webové aplikace jsou tzv. „service workers“, které se starají o to, aby aplikace fungovala, i když je uživatelovo zařízení bez připojení k internetu. Přidávají tak větší spolehlivost, že aplikace nerušeně a plynule poběží i bez rychlého připojení k internetu. Progresivní webová aplikace není tak výkonná, jako nativní aplikace, a výše zmíněný přístup k nativním funkcím telefonu závisí na prohlížeči, ve kterém

je spuštěna. Na druhou stranu lze webovou aplikaci spustit z jakéhokoliv prohlížeče bez ohledu na platformu. Což z ní dělá vhodného adepta, pokud je potřeba vytvořit aplikaci, která využívá přístupu k nativním funkcím pro vylepšení uživatelova zážitku z používání aplikace, ale není na těchto funkcích závislá, či není graficky náročná. (5)

3.3 Multiplatformní vývoj a frameworky

Multiplatformní vývoj je velice zajímavým tématem. Umožňuje totiž vytvořit aplikaci pro více platforem najednou. Toto řešení, kdy je k podpoře platforem potřeba pouze jeden kód, šetří čas strávený vývojem a také redukuje množství bugů. Přece jen méně kódu znamená méně prostoru pro tvorbu chyb.

Vytvoření multiplatformní aplikace dosahuje každý framework pomocí jiných principů, proto bude následovat přehled vybraných frameworků pro multiplatformní vývoj společně s jejich výhodami a nevýhodami.

3.3.1 Flutter

Flutter je UI Software development kit pro vývoj multiplatformních mobilních aplikací, který byl vyvinut firmou Google. První stabilní verze byla vydána 4. prosince 2018. (6) Tudíž se jedná o relativně mladý framework. Nutno ale dodat, že Flutter rapidně nabývá na popularitě. Pro srovnání, na GitHubu má téměř 86 tisíc hvězd, kdežto například React Native má 84,5 tisíce hvězd a Ionic má 40,1 tisíce. (7) (8) (9)

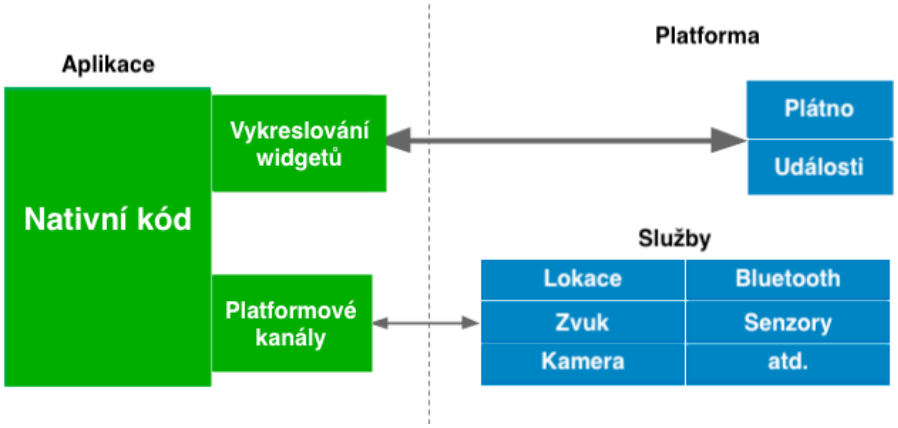
3.3.1.1 Princip Flutter aplikace

Přímo na oficiálních stránkách je zmíněno, že Flutter aplikace je nativně kompilovaná. (10) A tak by bylo dobré si říct, co to vlastně znamená, a jak Flutter funguje.

Naprosto zásadním rozdílem od většiny frameworků pro multiplatformní vývoj, které jsou zmíněny v této práci, je že Flutter nepoužívá žádný Javascript most pro komunikaci se zařízením.

Jediné, co si Flutter aplikace od zařízení žádá, je prázdné „plátno“, na které bude moci pomocí grafické knihovny Skia (knihovna pro 2D vykreslování) vykreslovat své vlastní widgety. Widget je jakýmsi stavebním blokem Flutter aplikace. Každý widget v sobě nese informace o tom, jak má vypadat a jak se má chovat na základě jejich aktuální konfigurace a stavu. Jedná se tak například o tlačítko, textové pole, či rozložení celé stránky. (11) (12)

Některé widgety samozřejmě potřebují přístup k funkcím telefonu, jako například ke kameře, nebo lokaci. Flutter aplikace tedy musí se zařízením komunikovat, aby měla k těmto nativním funkcím přístup. Mezi aplikací a zařízením je tak rozhrání, které pro tuto komunikaci vytváří kanály, kde komunikace probíhá. (13)



Obrázek 1 - schéma komunikace mezi Flutter aplikací a zařízením. Přeloženo z: (13)

Co se vzhledu widgetů týká, Flutter vytváří své vlastní namísto toho, aby používal widgety dané platformy, lze je tak volně upravovat a rozšiřovat. Za použití Cupertina pro iOS a Material design pro Android, je také možné, widgetům dodat nativní vzhled. (14) (13)

Flutter engine v sobě nese dvě základní technologie. První je výše zmíněná grafická knihovna Skia, která má na starosti vykreslování widgetů, a druhou je jazyk Dart, ve kterém jsou Flutter aplikace napsány. (14)

Právě užití programovacího jazyka Dart, z Flutter aplikací dělá nativně kompilované. Jazyk Dart totiž umožňuje „ahead-of-time“ kompilaci. Tento druh kompilace kompiluje kód do strojového nativního kódu před spuštěním aplikace, což způsobuje nativní spuštění výsledného binárního souboru. (14) (15) Samotný engine je pak kompilován za pomocí kompilátorů NDK u Androidu a LLVM u iOS. (14)

3.3.1.2 Výhody frameworku Flutter

- První výhodou je samozřejmě sdílení kódu mezi platformami iOS a Android. (16)
- Poskytuje Hot reload. Ten umožňuje spustit vyvíjenou aplikaci, která se při jakékoliv změně kódu znovu načte, a dané změny promítne, což značně urychluje vývoj. (16)

- Jak bylo řečeno v kapitole „3.3.1.1. Princip Flutter aplikace“, Flutter nepoužívá pro komunikaci se zařízením žádný most. (13) To zkracuje čas nutný pro prvotní načtení aplikace a také celkově zlepšuje výkon aplikace. (16)

3.3.1.3 Nevýhody frameworku Flutter

- Flutter je velice populárním frameworkem, ale je stále vcelku mladý. Díky tomu v porovnání s ostatními frameworky uvedenými v této práci neposkytuje tak velkou podporu komunity a nenabízí tolik knihoven třetích stran. (16)
- Jedná se o multiplatformní framework, ale zatím podporuje pouze mobilní platformy iOS a Android. (16) Google sice vydal podporu pro web, ta je ale zatím v beta verzi. (17)
- Fakt, že Flutter používá své vlastní widgety, obnáší jednu nevýhodu, engine pro jejich vykreslování musí být také součástí aplikace, díky čemuž jsou Flutter aplikace oproti nativním aplikacím o něco větší. (14) (13)

3.3.2 Xamarin

Xamarin je vývojářská platforma, určená pro vývoj multiplatformních aplikací za pomoci .NET, z čehož vyplývá, že Xamarin aplikace jsou vytvářeny prostřednictvím jazyka C#. Pro dosažení multiplatformního vývoje Xamarin využívá několik produktů. Xamarin.Android, Xamarin.iOS a Xamarin.Forms.

3.3.2.1 Xamarin.Android a Xamarin.iOS

Xamarin.Android a Xamarin.iOS plně zpřístupňují Android a iOS SDK pro vývojáře .NET. Díky Xamarin.Android a Xamarin.iOS je tak možné vytvářet aplikace pro Android a iOS za pomoci jazyka C#. (18) (19) Pokud vývojář použije pouze tyto dva produkty pro vývoj Xamarin aplikací, tak může sdílet logiku aplikace, ale uživatelská rozhraní musí vytvářet pro každou platformu zvlášť. Užití je tak vhodné ve chvíli, kdy je třeba, aby aplikace působila na každé platformě skutečně nativně čili zde není moc prostor pro sdílení uživatelského rozhraní. (20)

3.3.2.2 Xamarin.Forms

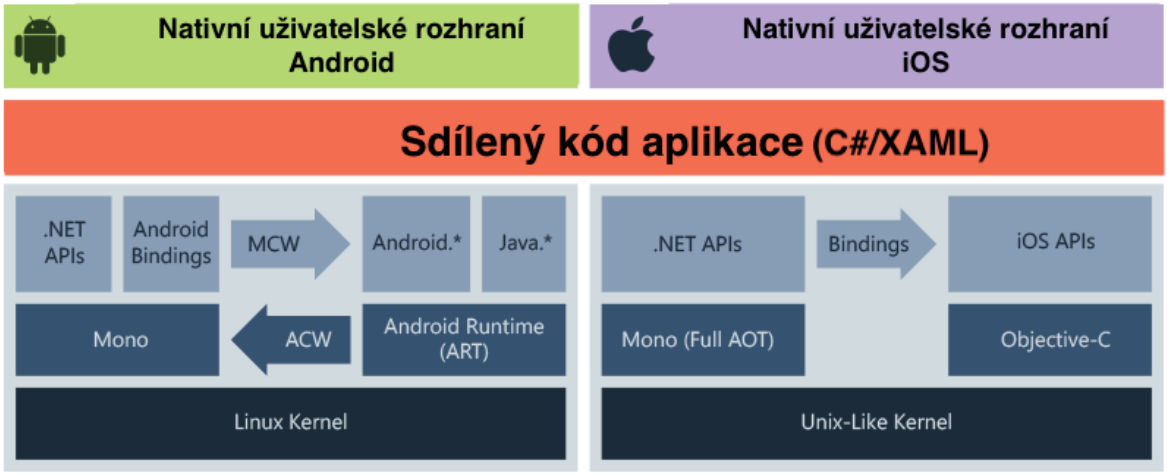
Xamarin Forms je jakousi vylepšenou verzí Xamarin.iOS a Xamarin.Android. Mimo sdílení logiky aplikace totiž umožňuje také sdílení uživatelského rozhraní prostřednictvím XAML (variace značkovacího jazyka XML, určená pro vytváření uživatelského rozhraní). (21) (22) Je tak vhodný pro aplikace, ve kterých není potřeba příliš aplikaci upravovat pro každou platformu zvlášť.

3.3.2.3 Princip frameworku Xamarin

Jak je vidět na obrázku č. 2, princip není pro obě platformy stejný. Jedno ale mají společné, běží v prostředí Mono. Mono je open-source verze .NET a funguje na většině platformách. (23)

iOS aplikace běží v Mono prostředí, které běží současně s Objective-C Runtime (Objective-C Runtime je knihovna, která umožňuje tvorbu mostu mezi jazykem Objective-C a ostatními jazyky). (24) (25) Obě tyto prostředí běží nad Unix jádrem, což umožňuje vývojářům přístup k nativnímu kódu zařízení. Komunikace mezi Objective-C a C# probíhá přes takzvané „bindings“. Aplikace je kompilována plně ahead-of-time z jazyka C# do nativního ARM assembly kódu. (24) Co je ahead-of-time kompilace již bylo popsáno v kapitole „3.3.1.1 Princip Flutter aplikace“.

Android aplikace také běží v Mono prostředí, namísto Objective-C Runtime knihovny je zde virtuální stroj Android Runtime. Tyto dvě prostředí zde běží nad jádrem Linuxu, což má ve výsledku stejný efekt jako u iOS. Zde komunikace probíhá tak, že na začátku .NET poskytne Androidu a Javě jmenné prostory. Mono následně do těchto jmenných prostorů volá skrze MCW (managed callable wrappers), jejichž součástí jsou ACW (android callable wrappers), pomocí kterých může nativní část odpovědět. Na rozdíl od iOS aplikace, Android aplikace jsou kompilovány nejdříve do intermediate-language, který je pak za chodu kompilován formou Just-in-time kompilace, což znamená, že je nativní kód dynamicky generován až v samotném zařízení. (23) (26) U iOS toto není možné z bezpečnostních důvodů nastavených firmou Apple. (24)



Obrázek 2 - Schéma Xamarin aplikace. Přeloženo z: (23)

3.3.2.4 Výhody frameworku Xamarin

- Stejně tak jako ostatní frameworky pro multiplatformní vývoj, Xamarin nabízí sdílení kódu. Xamarin aplikace fungují na platformách iOS, Android a Windows. (27)
- Xamarin.Forms umožňuje sdílení logiky i uživatelského rozhraní aplikace. Je tak vhodným kandidátem pro tvorbu prvotní verze prototypu aplikace, kdy je spíše podstatné vyzkoušet, zdali aplikace vůbec bude napříč platformami žádaná. (28)

3.3.2.5 Nevýhody frameworku Xamarin

- Xamarin aplikace v sobě obsahují řadu knihoven nezbytných pro jejich chod. Z toho důvodu jsou také o pár megabitů větší oproti nativním. (28)
- Další nevýhoda vyplývá ze samotného principu, jak Xamarin funguje. Xamarin aplikace nevyužívají přímo SDK daných platforem. Namísto toho využívají verze těchto SDK, které byly přepsány do jazyka C#. To znamená, že pokud se na některé z platforem objeví nová funkce, je nutné počkat až ji Xamarin tým implementuje do frameworku. (29)
- V případě, že je pro vývoj použit Xamarin.Android a Xamarin.iOS, tak má Xamarin ještě jednu nevýhodu. Tou je nutnost tvorby uživatelského rozhraní pro každou platformu zvlášť.

3.3.3 Ionic

Ionic je open source framework určen pro vývoj mobilních, desktopových a webových aplikací za pomocí webových technologií HTML, CSS a JavaScript. Soustředí se hlavně na uživatelské rozhraní čili na gesta, interakci s aplikací a animace. (30)

3.3.3.1 Ionic v4

V roce 2019 byl spuštěn Ionic v4, který se výrazně liší od předchozích verzí. V předchozích verzích byl vývoj Ionic aplikací možný pouze ve spojení s dalším frameworkem, který nese název Angular. To se ale ve čtvrté verzi změnilo. V této verzi byl Ionic předělán tak, aby mohl fungovat jako samostatná sada webových komponent a bylo jej možné použít s jakýmkoliv jiným frameworkem, či zcela bez frameworku. (30) (31) Nutno ovšem dodat, že v době, kdy byla tato práce vytvořena, Ionic oficiálně podporoval pouze frameworky Angular, React a Vue (podpora pro Vue ale byla v beta verzi)

3.3.3.2 Využití Ionic UI komponentů

Jak bylo řečeno v předchozím odstavci. Ionic je sadou webových komponent, a tak je nejdříve nutné vysvětlit, co to vlastně ty webové komponenty jsou. Webové komponenty představují soubor technologií, které nám umožňují vytvářet vlastní upravovatelné HTML elementy. Tyto komponenty je pak následně možné v moderních prohlížečích používat jako klasický HTML element. (32) (33)

Pokud tedy vývojář chce využít framework Ionic ve své aplikaci, ve které používá pouze JavaScript bez frameworku, tak mu stačí pouze importovat Ionic framework prostřednictvím CDN či nainstalovat skrze NodeJs a následně používat Ionic komponenty jako kterýkoliv jiný HTML element na stránce. (34) (35) Nutno dodat, že jména pro jednotlivé Ionic komponenty se liší na základě toho, jaký JavaScript Framework byl společně s Ionic frameworkem použit.

Ionic komponenta použitá v React aplikaci

```
<IonButton color="secondary" fill="outline" onClick={handleClick} />
```

Ionic komponenta použitá v aplikaci bez JavaScript Frameworku

```
<ion-button color="secondary" fill="outline" onclick="handleClick" />
```

Obrázek 3 - Porovnání použití Ionic komponenty v React aplikaci a v aplikaci bez JavaScript frameworku. Zdroj: vlastní tvorba

3.3.3.3 Přístup k nativním funkcím pomocí Capacitoru

Z předchozího textu vyplývá, že Ionic sám o sobě neumožňuje tvorbu multiplatformní mobilní aplikace. K tomuto účelu potřebuje most, který poskytne aplikaci přístup k nativním funkcím zařízení. Pro aplikaci tohoto mostu je na výběr ze dvou možností. První je Capacitor, který byl vyvinut týmem, který stojí za Ionicem, druhou možností je Apache Cordova. Vzhledem k tomu, že Capacitor je novější, a dle týmu Ionic má jednoho dne Apache Cordovu v Ionic aplikacích nahradit, tak bude dále popisován pouze Capacitor. (36)

Capacitor funguje vcelku jednoduše. iOS i Android poskytují takzvaný WebView. WebView umožňuje nativním aplikacím zobrazovat interaktivní obsah webu. (37) Tohoto faktu Capacitor využívá. Capacitor v podstatě vezme celou Ionic aplikaci a zabalí ji do nativní schránky. Tato nativní schránka obsahuje pouze zmíněný WebView, který v sobě nese celou Ionic aplikaci. WebView také umožňuje komunikaci mezi JavaScriptem ve webové aplikaci, a nativním kódem. Tudíž pro přístup k požadované nativní funkci zařízení stačí napsat malou část nativního kódu a propojit ji s Javascriptem, který pak nativní kód volá. Toto má také na starosti Capacitor. Poskytuje totiž řadu již připravených nativních API, například API pro přístup ke kameře, či k souborovému systému zařízení. Případně také umožňuje tvorbu vlastních pluginů. (36) (38) Capacitor také podporuje framework Electron (tato podpora je zatím ve zkušební verzi), díky kterému je možné vytvořit i desktopovou aplikaci, která bude fungovat na platformách Windows, MacOS a Linux. (39)

Obrázek 4 - Schéma Ionic Capacitor aplikace. Přeloženo z: (36)

3.3.3.4 Ionic a Javascript frameworky Vue, Angular a React

Použití Ionicu spolu s frameworky Vue, Angular či React je vcelku jednoduché. Stačí pouze pomocí Node.js vytvořit Ionic aplikaci, a při této tvorbě zadat jako typ aplikace jeden z podporovaných JavaScript frameworků. Konkrétní příklad použití je popsán v kapitole 4.4.1 Vytvoření projektu.

Jak již bylo řečeno, Ionic se stará hlavně o uživatelské rozhraní, a tak z hlediska použití Ionicu není podstatné, který JavaScript framework je pro vývoj zvolen, ovšem až na určité výjimky. Jedná se zejména o Routing aplikace a lifecycle metody.

Routing aplikace je upraven proto, že JavaScript frameworky byly vytvořeny pro klasické webové aplikace, které zpravila nevyžadují animace při přechodu mezi jednotlivými stránkami. Pro lepší nativní pocit z Ionic aplikace, je tak Routing o tyto animace obohacen. (40)

Proč má Ionic i své lifecycle metody? Prvně by bylo dobré říct, co to lifecycle metody vlastně jsou. Každá komponenta v aplikaci má svůj životní cyklus. Během tohoto cyklu je komponenta připojena, následně v průběhu svého života aktualizována, a na konci je odpojena. S tímto cyklem se pojí funkce, které v jednotlivých částech cyklu lze spustit. Například zavolat dotaz na databázi, když se komponenta připojí. JavaScript frameworky tyto lifecycle metody samozřejmě už mají v sobě zabudované, existují ale případy, kdy

lifecycle metody závisí také na Routingu aplikace, například při přechodu na novou stránku. O Routing se ale v Ionic aplikaci stará sám Ionic. Čili pokud chceme volat lifecycle metody, které se mají spustit při přechodu mezi stránkami, je nutné použít lifecycle metody Ionicu. (41)

3.3.3.5 React

V praktické části této práce je společně s frameworkem Ionic také použita knihovna React. Proto je v této kapitole React knihovna více přiblížena.

React je deklarativní, výkonná, flexibilní a komponentově založená JavaScript knihovna. Byl vyvinut firmou Facebook a je určen pro tvorbu uživatelského rozhraní. (42)

Tvorba uživatelského rozhraní ale není jeho jediné využití. Pomocí Reactu v kombinaci s knihovnami, které umožňují například routing nebo state management, se běžně tvoří webové Single Page aplikace, či Progresivní webové aplikace. Pro psaní React aplikací se používá Javascript a JSX. JSX je v podstatě z hlediska syntaxe, vylepšený JavaScript pro Ajaxovou tvorbu HTML elementů. JSX se tedy ve finále zkompiluje na klasický JavaScript (42)

Obrázek 5 - Ukázka JSX kódu, a Javascript kódu, na který je převeden kompilátorem. Přeloženo z: (42)

3.3.3.6 Výhody frameworku Ionic

- Ionic framework je ze všech frameworků uvedených v této práci nejvíce multiplatformní. Ionic aplikace v kombinaci s Capacitorem a Electronem funguje na webu, iOS, Androidu, Windowsu, Linuxu či MacOS. (43)
- Aplikace jsou nejvíce oddělené od samotných zařízení. K tvorbě je tak potřeba zejména znalost webových technologií. (44)
- Ionic a Capacitor nabízí řadu připravených řešení pro použití různých prvků uživatelského rozhraní, či pro přístup k nativním funkcím zařízení. Také je zde řada pluginů, vytvořených komunitou. Fakt, že je Ionic z velké části webovou aplikací, také umožňuje použít řadu knihoven a pluginů určených pro web. Díky tomu je málokdy nutné přistoupit k tvorbě vlastního pluginu, ke kterému je potřeba nativní vývoj.

3.3.3.7 Nevýhody Frameworku Ionic

- Vytvořit Ionic aplikaci tak, aby působila nativně může být opravdovou výzvou. Přeci jen webové technologie nebyly zamýšleny pro tvorbu mobilních aplikací.
- Díky tomu, že se jedná zejména o webovou aplikaci vykreslovanou pomocí prohlížeče, která je se zařízením propojena pouze pomocí mostu, je výkon menší v porovnání s nativním vývojem. A nejen s ním, také některé multiplatformní frameworky (například React Native, Flutter či Xamarin) mají blíže k nativní aplikaci, tudíž mají lepší výkon. Ovšem pokud aplikace nevyžaduje použití vylepšené reality, či není jinak vysoce graficky náročná, tak je výkon dostačující. (45)
- Chybí hot reload. (45) Aplikace lze sice vyvíjet jako klasické webové aplikace čili zejména v prohlížeči, kde je live reload, což je v podstatě to samé jako hot reload, akorát je aplikace po změně znovu vykreslena v prohlížeči. Avšak pro skutečné testování aplikace je nutné ji zkoušet přímo na zařízení, což u Ionic Capacitor aplikací obnáší proces zkompilování aplikace a její instalaci na zařízení, či spuštění v simulátoru.
- Občas je debugging velice obtížný. V Ionic aplikaci je často použita řada knihoven a pluginů, což může vyústit v řadu nečekaných bugů. (44) A nejen to, i při tvorbě vlastního kódu lze občas narazit na situaci, kdy se prvek chová ve webové aplikaci

dle očekávání, zatímco na některém ze zařízení vykazuje určité odlišnosti. K odhalení chyby je tak kolikrát nutné důkladně procházet vlastní kód, dokumentaci samotného frameworku, použitých knihoven a pluginů a v neposlední řadě dokumentaci daného zařízení. Díky tomu se může vývoj nečekaně protáhnout.

3.3.4 Apache Cordova a Phonegap

Apache Cordova a PhoneGap jsou v podstatě jedna a ta samá věc. Jediný rozdíl je v tom, že Apache Cordova je open-source verze frameworku, zatímco framework PhoneGap je placenou verzí, kterou nabízí společnost Adobe, a má oproti Apache Cordova určité výhody. (46)

Tyto frameworky fungují na stejném principu jako framework Ionic společně s Capacitorem. Aplikace jsou tedy také zabalené do WebView a k jejich tvorbě se také používají webové technologie HTML, CSS a JavaScript. (47) Z tohoto důvodu zde nebude blíže popsán princip frameworku ani jeho výhody a nevýhody.

3.3.5 React Native

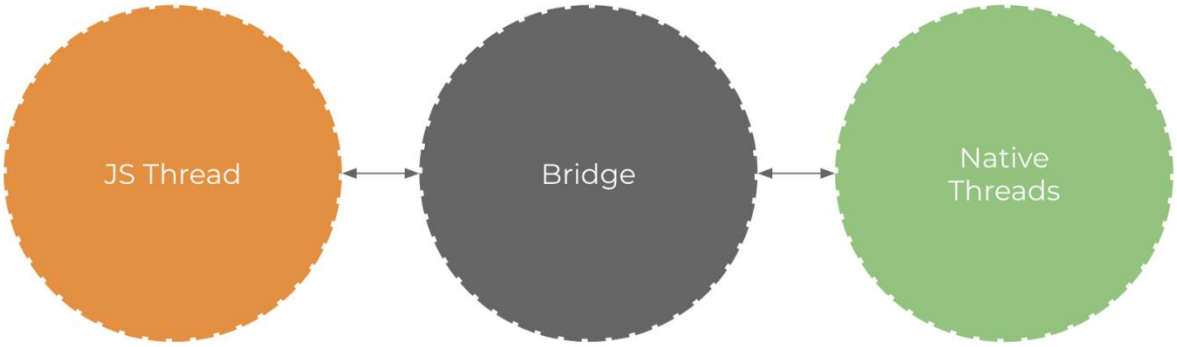
Framework React Native kombinuje nativní vývoj mobilních aplikací, s knihovnou pro webové aplikace React. (48) Více o této knihovne se lze dočíst v kapitole „3.3.3.5 React“

3.3.5.1 Princip frameworku React Native

Jak již bylo zmíněno, React Native se skládá ze dvou částí, z nativní a JavaScriptové. Při tvorbě React Native aplikace je vytvořena instance UIView, což je ta samá instance, která je vytvořena při tvorbě nativní aplikace. Čili část aplikace, která má na starosti zobrazování elementů a zpracovávání gest uživatele React Native aplikace, je nativní kód, zatímco veškerá logika aplikace je tvořena v JavaScriptu. (49) (50)

Zatím neexistují žádné kompilátory, které by převáděli JavasScript na nativní kód. Tudíž Javascript nekomunikuje s nativním kódem přímo. Komunikace probíhá skrze most, který je navržen tak, aby byl schopen komunikovat s nativními platformami a je tak tou nejpodstatnější částí, která dělá z React Native aplikace, multiplatformní aplikaci. Tento most poskytuje asynchronní a obousměrnou komunikaci. Komunikace probíhá tak, že Javascript část posílá asynchronní JSON zprávy, ve kterých jsou v podstatě instrukce, co má

nativní strana vykonat. Jakmile je nativní strana připravena, tak vykoná to, oč ji logická čili JavaScript část žádá. (49)



Obrázek 6 - Schéma komunikace mezi JavaScriptí logickou částí a nativní částí. Zdroj: (49)

3.3.5.2 Výhody frameworku React Native

- Jako u ostatních frameworků, pomocí React Native lze tvořit aplikaci pro iOS a Android, ve které bude značná část kódu sdílena. (51) Nejen to, React Native aplikace je tvořena zejména pomocí jazyku JavaScript a knihovny React. A tak za předpokladu, že bude třeba vytvořit i webovou verzi aplikace, a pro její tvorbu bude použita knihovna React, lze část kódu sdílet i s touto webovou verzí. (52)
- Část React Native komponent je vykreslena pomocí nativních API daného zařízení. Díky tomu je React Native rychlejší a působí více nativně oproti frameworkům Ionic (kapitola 3.3.3 Ionic) či Apache Cordova (kapitola 3.3.4 Apache Cordova a Phonegap), které pro svůj chod používají Web View. (53)
- React Native stejně jako některé další frameworky poskytuje Hot Reloading. (51)

3.3.5.3 Nevýhody frameworku React Native

- Komunikace se zařízením a přístup k nativním funkcím se neobejde bez nativního vývoje. Dnes je již řada těchto funkcí zpřístupněna frameworkem samotným, ale neplatí to pro všechny funkce. Pokud je třeba přistoupit k něčemu, co framework nenabízí, tak je nutné danou funkci vytvořit pomocí nativního kódu pro každou platformu zvlášť a dané řešení následně integrovat do aplikace. (51)

- Další problém souvisí s předchozím bodem. Mimo volbu vlastního řešení je ještě možné zvolit řešení třetích stran. Ty ale bývají často nedoladěná, neaktualizovaná, či špatně zdokumentovaná. (54)

3.4 Shrnutí frameworků pro multiplatformní vývoj

V době tvorby této práce patří mezi nejoblíbenější frameworky Flutter a React Native, to ale zdaleka neznamená, že jsou tyto frameworky tou nejlepší volbou. (55) Každý z frameworků uvedených v předchozí kapitole má své výhody a nevýhody. Xamarin poskytuje .NET vývojářům možnost tvorby i mobilních aplikací, ale plně spoléhá na C# verze SDK pro jednotlivé platformy. React Native se příliš neliší od knihovny React a tak umožňuje React vývojářům snadněji tvořit mimo webových aplikací také mobilní aplikace, ale nativnímu vývoji se lze těžko vyhnout, protože uživatelské rozhraní je tvořeno pomocí nativních prvků. Flutter slibuje nejlepší výkon, je ale zároveň nejmladším ze zmíněných frameworků, což se sebou nese nevýhody ve formě menší komunity, podpory menší škály podporovaných platforem a menšího množství nabízených již vytvořených pluginů (resp. Flutter Widgetů). Framework Ionic umožňuje vytvořit aplikaci pro web, iOS, Android, MacOS, Linux i Windows. Ionic komponenty také mění svůj vzhled dle platformy, na které aplikace běží. Ionic tak umožňuje vcelku rychlou tvorbu uživatelského rozhraní. Ionic aplikace má ale ze všech zmíněných zástupců nejblíže k webové aplikaci, a tak je za tento rychlý počáteční vývoj a široké pokrytí platforem placeno nižším výkonem a horším nativním dojmem.

Při volbě frameworku je tak vždy nutné stanovit, co je od aplikace požadováno a také je samozřejmě dobré zohlednit, jaké technologie ovládají vývojáři, kteří se mají na vývoji podílet. Přeci jen nemá smysl, aby se například firma, která se zabývá hlavně vývojem webových aplikací, rozhodla vyvíjet Flutter aplikaci.

3.5 Shrnutí webového, multiplatformního a nativního vývoje

Multiplatformní aplikace dnes nedílně patří k vývoji aplikací, ale rozhodně nejsou vhodné pro vývoj jakékoliv aplikace. Kde tedy mají své místo? Pro dostatečné zodpovězení této otázky je dobré rozebrat každý přístup uvedený v titulu této kapitoly.

První na řadu přichází webové aplikace. Při vývoji webových aplikací není třeba jakkoliv hledět na platformu, na které běží. Stačí pouze zohlednit prohlížeč. Progresivní webové aplikace umožňují offline přístup k aplikaci. Dokonce umožňují vypnout rozhraní

prohlížeče a umístit ikonu aplikace na telefon či počítač, čímž vzbuzují nativní dojem. Dnes již také poskytují přístup například ke kameře telefonu. Dokonce i z hlediska použitých technologií se příliš neliší od některých frameworků pro multiplatformní vývoj. Ionic či Apache Cordova aplikace jsou přeci také vyvíjeny pomocí HTML, CSS a JavaScriptu. Ovšem jak již ale bylo řečeno v kapitole „3.2 Vývoj progresivních webových aplikací“, podpora těchto funkcí není u progresivních webových aplikací stoprocentní. Mimo tuto omezenou podporu má progresivní webová aplikace ještě jednu nevýhodu. Přístup k nativním funkcím je omezen zabezpečením prohlížeče. Tato omezení se ale na multiplatformní aplikace nevztahují. Čili webové aplikace, potažmo progresivní webové aplikace se hodí ve chvíli, kdy je hlavním cílem předat uživatelům určité sdělení, případně poskytnout určitý bonus ve formě funkce, která přistupuje k funkci zařízení, čímž obohatí uživatelovu zkušenost s aplikací, zároveň ale aplikace na této funkci není závislá.

Ve chvíli, kdy je pro aplikaci přístup k funkcím zařízení klíčový, přichází na scénu multiplatformní aplikace. Hlavní výhodou, kterou poskytují všechny frameworky pro multiplatformní vývoj aplikací, je možnost sdílení kódu napříč verzemi aplikace pro jednotlivé platformy. Sdílení kódu redukuje čas a peníze, které je nutné do vývoje investovat. (56) Není tomu tak ale vždy. Vývojáři musejí aplikaci vytvořit tak, aby působila nativně, což mnohdy může být výzvou.

Tato výzva je zapříčiněna několika faktory:

1. Všechny frameworky pro multiplatformní vývoj jsou založeny na propojení dvou světů. Jedním je samotná aplikace a druhým je nativní platforma, na které aplikace běží. Multiplatformní frameworky toto propojení realizují pomocí mostu či komunikačních kanálů, což se více či méně podepisuje zejména na výkonu aplikace. Nejlépe se s tímto problémem vypořádává Flutter, ale je to za cenu toho, že jsou Flutter aplikace znatelně větší, oproti nativním. To také není ideální, protože velikost aplikace bývá často důvodem k odinstalování.
2. Je nutné vytvořit uživatelské rozhraní, které na každé platformě vypadá nativně. Jinými slovy je nutné uživatelské rozhraní pro každou platformu zvlášť upravit. Většina frameworků sice poskytuje řadu již připravených prvků, které lze do aplikace jednoduše vložit, ale pokud vývojář potřebuje něco specifického, musí si to buďto vytvořit sám, nebo spoléhat na pluginy třetích stran. Pluginy třetích stran ale často nebývají pravidelně aktualizované a bez chyb.

3. Vývojář se občas nevyhne tvorbě nativního kódu. Frameworky sice nabízejí přístup k funkcím zařízení, ale ne ke všem. To samé se u některých frameworků, jako například React Native, týká i výše zmíněných specifických prvků uživatelského rozhraní.

Z těchto faktorů je tak zřejmé, že multiplatformní vývoj není vhodný pro všechny druhy aplikací. Hodí se ve chvíli, kdy firma potřebuje vytvořit aplikaci, která není graficky náročná, a jejím hlavním cílem je pokrýt co největší segment trhu s aplikacemi. (57)

Aplikace, které mají vysoké požadavky na výkon, nejsou vhodné z důvodu omezeného výkonu. Také aplikace, které jsou hodně specifické vzhledem a přístupem k nativním funkcím zařízení, nemá smysl tvořit multiplatformní, protože by to stejně obnášelo velké množství nativního kódu. V těchto případech je vhodný spíše nativní vývoj.

V některých článcích také uvádí, že multiplatformní aplikace není vhodná, pokud je aplikace jádrem našeho podnikání. (58) To ale není zcela přesné, protože například mobilní aplikace Facebook a Instagram jsou vytvořeny pomocí frameworku React Native. (59)

Z této kapitoly tak vyplývá, že každý způsob vývoje má své uplatnění. Nelze tak jednoznačně říct, který přístup je nejlepší. Zvolení vhodného přístupu tak vždy závisí na požadavcích aplikace a úvaze samotných vývojářů. Ostatně tak už to v informačních technologiích bývá.

4 Vlastní práce

4.1 Požadavky na aplikaci

Vývoj aplikace zpravidla trvá několik měsíců. Je tak dobré si nejdříve jasně stanovit, co má daná aplikace umět a jak svých funkcí má dosáhnout. Bez jasného návrhu je totiž velice snadné během vývoje přicházet se stále novými vylepšeními, která prodlužují celkovou dobu vývoje.

Hlavním cílem této aplikace je poskytnout uživatelům možnost zaznamenávat textové, či foto poznámky a ty pak následně sdílet s ostatními uživateli. Takovýto popis ale rozhodně nestačí. Bude tak následovat přesnější popis, jak aplikace jednotlivých funkcí dosahuje.

4.1.1 Zaznamenávání textových poznámek

Aplikace musí umožňovat tvorbu textových poznámek. Pro tuto funkci nestačí klasické textové pole. To totiž sice umožňuje vkládání textu, ale postrádá možnost text jakkoliv formátovat. Z toho důvodu bude v aplikaci použit WYSIWYG (What You See Is What You Get) editor, který je blíže popsán společně i s názornou ukázkou v kapitole „4.4.4.1 Text poznámky“.

4.1.2 Zaznamenávání foto poznámek

Uživatel bude mít v aplikaci možnost vložit do poznámky také fotku či obrázek. Bude mít na výběr ze dvou možností, buďto fotku pořídí pomocí kamery telefonu, či fotku vybere z galerie telefonu.

4.1.3 Vyhledávání poznámek

Předpokládá se, že uživatelé budou mít řadu poznámek. Pro snadnější a rychlejší práci tak bude v aplikaci zabudováno vyhledávání, které uživatelům umožní vyhledávat poznámky na základě jejich nadpisů.

4.1.4 Sdílení poznámek

V neposlední řadě je nutné stanovit, jak bude v aplikaci probíhat sdílení poznámek, s touto funkcí se také pojí otázka, kde budou samotné poznámky (data aplikace) uchovávány a jak k nim budou uživatelé přistupovat.

Bylo rozhodnuto, že uživatelé budou mít možnost spravovat, s kým poznámku mohou sdílet. Na druhou stranu uživatelům, se kterými je poznámka sdílena, bude umožněno poznámku také upravovat.

Pro poskytnutí této funkcionality tak není možné poznámky ukládat přímo na zařízení. Poznámky tedy budou ukládány na serveru, ke kterému budou uživatelé přistupovat skrze internet. Přístup budou mít k poznámkám, ve kterých jsou zapsáni buďto jako vlastníci, či jsou v seznamu spolupracovníků. S čímž se pojí nutnost, aby každý uživatel měl jednoznačný identifikátor, pomocí kterého ho lze určit jako vlastníka poznámky, či spolupracovníka, který k ní má také přístup. Aplikace tedy bude obsahovat registraci a přihlášení.

Aplikace není primárně určena pro hromadnou úpravu jedné poznámky několika uživateli najednou. I tak je ale nutné počítat s možností, že dva a více uživatelů přistoupí najednou k jedné poznámce. Je tak nutné, aby aplikace kontrolovala, že verze zachycených poznámek je aktuální. V případě, že někdo pozmění obsah nějaké poznámky, se tato změna musí promítnout i ostatním uživatelům, kteří mají k poznámce přístup.

4.1.5 Offline funkce a synchronizace dat

V předchozí kapitole „4.1.4 Sdílení poznámek“ bylo napsáno, že poznámky budou uloženy na serveru, ke kterému se uživatelé budou přihlašovat a následně přistupovat k poznámkám, u kterých bude průběžně kontrolováno, zdali nebyly nějakým dalším uživatelem upraveny. V ideálním případě, by všichni uživatelé měli neustále kvalitní připojení k internetu, tak tomu ale v reálném světě není.

Z důvodu přihlašování k serveru, sdílení poznámek a synchronizaci dat není možné, aby aplikace plně fungovala offline, zároveň ale musí alespoň částečnou offline podporu nabízet. Přeci jen krátkodobé výpadky připojení nejsou výjimečné a aplikace by se stala vcelku nepoužitelnou, pokud by při každém takovém výpadku přestala kompletně fungovat.

Aplikace tak v případě, že je uživatel přihlášen, bude umožňovat přístup k již zachyceným poznámkám, u nich bude mít uživatel možnost upravovat text. Dále bude uživateli umožněna tvorba nových poznámek. Při následném opětovném připojení k internetu aplikace odešle změny, které se uloží na server.

Samozřejmě zde hrozí scénář, že dva uživatelé přistoupí ke stejné poznámce najednou. Jeden z nich během úprav poznámky přejde do offline režimu a pokračuje v úpravách poznámky. Během toho druhý uživatel zůstane online a také nadále poznámku upravuje.

Ve chvíli, kdy uživatel, který byl offline, přejde do online režimu, se jeho verze dané poznámky bere jako aktuální, čímž přemaže veškeré změny, které učinil uživatel, který byl celou dobu online. To je nepříjemné, je ale dobré zopakovat, že aplikace není primárně určena pro hromadnou úpravu jedné poznámky několika uživateli najednou.

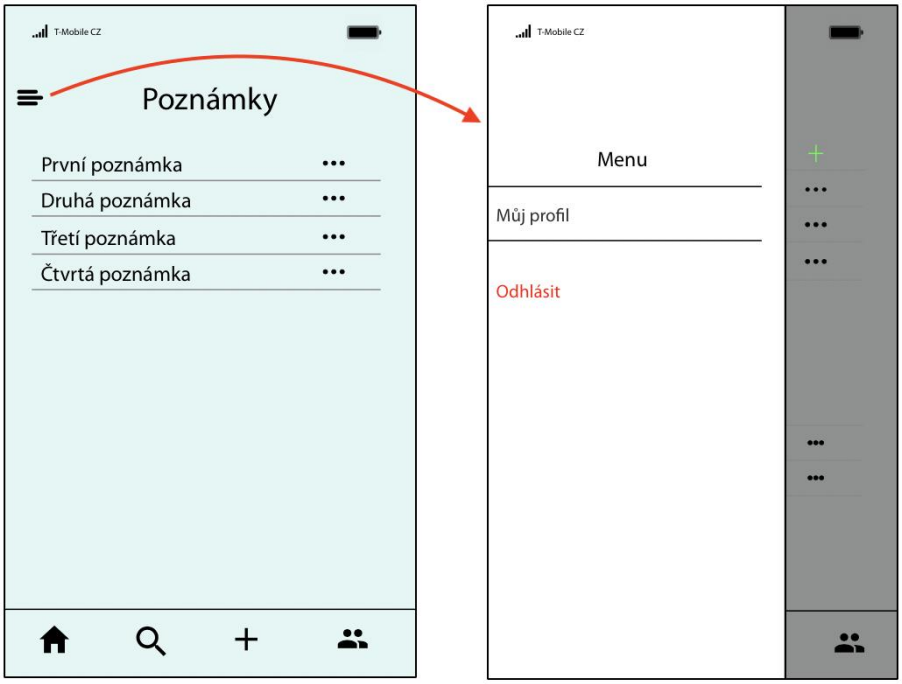
Fotky či obrázky, které chce uživatel vložit do poznámky, budou ukládány v úložišti Firebase, které nevytváří lokální kopii. Z tohoto důvodu bude uživatelům umožněno vkládat fotky či obrázky pouze v případě, že budou online.

4.2 Návrh uživatelského rozhraní aplikace

Před samotným vývojem byl vytvořen nákres aplikace. Pro následnou jednodušší implementaci návrhu, nebyl vytvořen pouze wireframe, ale spíše grafický návrh. V tomto návrhu byly vytvořeny pouze nejpodstatnější stránky, které se od sebe vzhledově výrazně liší. Nutné také říci, že se jedná pouze o návrh, tudíž je ve finální aplikaci možné najít odlišnosti.

4.2.1 Domovská obrazovka

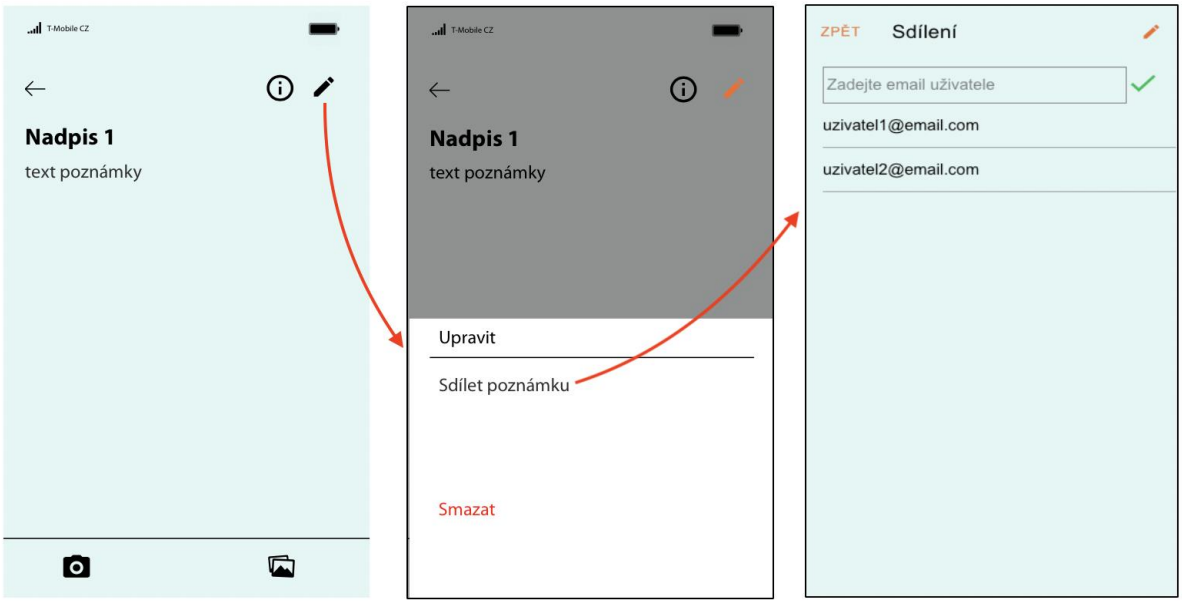
Dnes již nikdo nechce číst dlouhé návody a manuály, a tak byla při tvorbě grafického rozhraní hlavním cílem intuitivnost a jednoduchost ovládání aplikace. Z tohoto důvodu se všechna hlavní tlačítka pro funkci aplikace, nachází ve spodní části obrazovky, kde jsou za každé příležitosti snadno dostupná. Tlačítko pro zobrazení menu k obecné správě aplikace bylo po vzoru většiny mobilních aplikací umístěno do levého horního rohu. Po kliknutí na tzv. „hamburger“ tlačítko, vyjede menu z levé strany, jak je ilustrováno níže na obrázku 7.



Obrázek 7 - Návrh domovské obrazovky aplikace a hlavního menu aplikace. Zdroj: vlastní zpracování

4.2.2 Poznámka

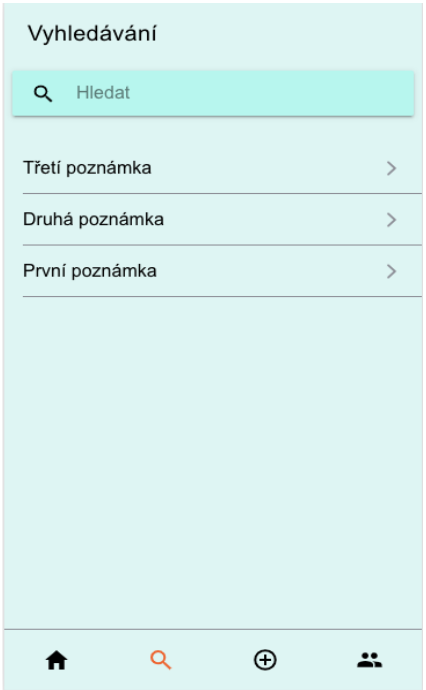
U poznámky byl opět kladen důraz na jednoduchost. Je důležité, aby měl uživatel všechna podstatná tlačítka ihned na dosah. Návrh je možné vidět na obrázku 8. Levá část obrázku reprezentuje návrh samotné poznámky. V prostřední části je ukázáno menu, které se objeví po klinutí na tužtičku v pravém rohu. Pravá část představuje návrh obrazovky pro správu sdílení poznámky, která se zobrazí po klinutí na tlačítko „sdílet poznámku“.



Obrázek 8 - Návrh obrazovky pro poznámku. Zdroj: vlastní zpracování.

4.2.3 Vyhledávání

Vyhledávání je poslední výrazně odlišnou obrazovkou aplikace. Jedná se o vcelku jednoduchý návrh, který lze nalézt na obrázku 9.



Obrázek 9 - Návrh obrazovky pro vyhledávání poznámek.
Zdroj: vlastní zpracování

4.3 Nástroje využité pro tvorbu aplikace

V této kapitole je pouze uveden soupis uvedených nástrojů společně s jejich stručným popisem. Praktické příklady použití přímo v aplikaci jsou uvedeny v pozdějších kapitolách.

4.3.1 Node package manager

Jedná se o správce balíčků, který byl napsán v JavaScriptu. Kromě správy JavaScript balíčků, také poskytuje celé event-driven prostředí, které prostřednictvím nástrojů jako například Webpack, umožňuje automatizovat některé lokální úlohy. (60) Event-driven znamená, že jsou činnosti vykonávány na základě událostí (eventů). Tato událost může být například zmačknutí klávesy. (61)

4.3.2 Ionic a Capacitor

Tyto nástroje již byly popsány v teoretických východiscích. Ionic byl v aplikaci využit pro tvorbu uživatelského rozhrání. Pro přístup ke kameře a galerii telefonu, byl využit Capacitor.

4.3.3 React

Tato knihovna byla zmíněna v teoretických východiscích. Samotný Ionic se totiž stará pouze o vzhled a chování jednotlivých komponent. O to, za jakých okolností se mají Ionic komponenty zobrazit se stará React. Například výše zmíněné tlačítko pro úpravu poznámky a zespod vyjíždějící nabídka (viz. obrázek 8). Ionic nabízí ikonku tužky a vyjíždějící menu, kdežto React hlídá ty samotné akce a stav daného komponentu, v tomto případě hlídá akci kliknutí a přepíná stav mezi otevřeným a zavřeným.

4.3.4 Firebase a Cloud Firestore

Pro backend aplikace byla zvolena platforma Firebase. Samozřejmě by bylo možné vytvořit vlastní backend, ale platforma Firebase je pro začínající projekty velmi zajímavým, funkčním, a hlavně časově i finančně efektivním řešením. (62)

4.3.4.1 Firebase

Firebase je Backend-as-a-Service platforma, která nabízí hotové řešení backendu. Tato platforma umožňuje vývojářům do svých aplikací implementovat například synchronizaci dat, posílání upozornění na zařízení, databázi, úložiště a autentizaci uživatelů. (62) (63)

4.3.4.2 Cloud Firestore

Jedná se o databázi poskytovanou od Firebase. Mimo tuto databázi, nabízí Firebase ještě jedno starší řešení nazvané Firebase Realtime Database. Obě dvě tyto databáze jsou NoSQL databáze. Velký rozdíl je v tom, že Realtime Database ukládá data jako jeden velký JSON strom, kdežto Firestore ukládá data v kolekcích, ve kterých lze najít jednotlivé záznamy nazvané dokumenty. (64) Díky této struktuře dokumentů a kolekcí, nabízí Firestore oproti Realtime Databse například lepší a efektivnější tvorbu dotazů, nebo lepší strukturu dat a případnou rozšiřitelnost databáze. (62)

4.3.5 Redux, Redux-thunk, React-redux-firebase a redux-firestore

Jedná se o 4 knihovny, které se dohromady starají o správu globálního stavů aplikace.

4.3.6 Styled-components

Styled-components je knihovna, která ulehčuje práci při stylování jednotlivých komponent. Umožňuje vytvářet styly přímo v komponentách. Dále se stará o dynamické vkládání stylů. To znamená, že jsou vloženy pouze styly komponent, které jsou skutečně na stránce v danou chvíli vykreslené. Mimo tuto funkci také umožňuje do stylů vložit Javascript podmínky, díky kterým je umožněno zobrazovat styly podmíněně. Také pro každou css třídu generuje unikátní jména.

4.3.7 Uuid

Knihovna, která vytváří jednoznačné identifikátory.

4.3.8 Quilljs

Jedná se o textový editor, který je v aplikaci využit pro tvorbu a editování poznámek.

4.3.9 Xcode a Android Studio

Bez těchto dvou IDE, nelze vyvíjet mobilní aplikace. Starají se o samotnou kompilaci kódu pro mobilní zařízení a nelze bez nich aplikaci spustit na mobilním telefonu.

4.3.10 Použité telefony

Do jisté míry lze Ionic aplikaci vytvářet pouze za použití webového prohlížeče. Jak již bylo řečeno, jedná se pouze o webovou aplikaci zabalenou do webview. Nicméně pro opravdové testování, je dobré aplikace zkoušet přímo na mobilních zařízeních. Během vývoje byla použita zařízení iPhone SE a Samsung Galaxy J5.

4.4 Tvorba aplikace

4.4.1 Vytvoření projektu

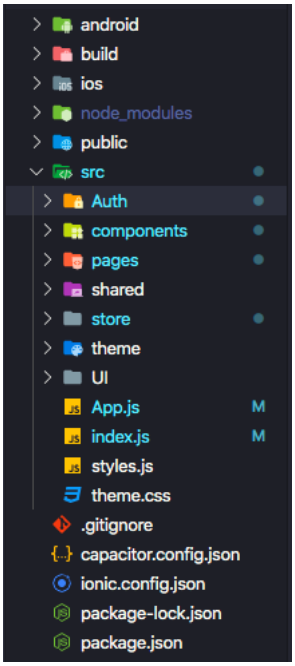
Ionic má svou vlastní command line utilitu. Toho je v poznámkové aplikaci také využito, protože Ionic command line příkazy výrazně ulehčují počáteční nastavení aplikace. Vytvoření projektu, ve kterém je Ionic propojen s knihovnou React je tak uskutečnitelné pomocí jednoho příkazu. V něm je nejdříve uvedeno jméno projektu, následně počáteční rozložení aplikace (tabs automaticky vytvoří rozložení, ve kterém se tlačítka pro základní navigaci v aplikaci nachází v dolní části obrazovky. Toto rozložení je demonstrováno na obrázku 7 – Návrh domovské obrazovky aplikace, který se nachází v kapitole 4.2.1 Domovská obrazovka), a jako poslední je uveden typ aplikace, což je zde React aplikace.

```
ionic start notes-app tabs --type=react
```

Obrázek 10 - Příkaz pro vytvoření Ionic projektu. Zdroj: vlastní zpracování

4.4.2 Struktura souborů projektu

Struktura souborů je během programování velice podstatná z hlediska přehlednosti, ale jak již bylo řečeno, pro tvorbu aplikace je společně s frameworkem Ionic, použita knihovna React. Tato knihovna nemá jasně dáno, jak mají být soubory strukturovány. (42) Proto je dobré si popsat strukturu souborů, která byla v poznámkové aplikaci použita.



Obrázek 11 - Souborová struktura poznámkové aplikace.

Zdroj: vlastní zpracování.

- **Android** – Složka android obsahuje kód pro nasazení aplikace na operační systém android
- **Build** – Zde se nachází verze aplikace, která je použitelná v klasických prohlížečích.
- **ios** – Tato složka obsahuje kód pro nasazení aplikace na operační systém iOS
- **Node_modules** – Zde se nachází veškeré soubory spravované Node package managerem
- **Public** – V této složce je pouze .html soubor, který v elementu <body>, vykresluje pouze jeden element , který v sobě nese atribut id. To je vše, co React potřebuje, všechno ostatní v aplikaci se právě na tento element napojuje dynamicky skrze JavaScript.
- **Src** – Zde je veškerá logika aplikace. Je to také místo, pro které není jasně definována struktura, proto je teď bude následovat podrobnější popis tohoto souboru.
    - **Auth** – V tomto souboru je umístěna komponenta pro registraci a přihlášení uživatele.
    - **Components** – Do této složky jsou uloženy všechny takzvané „presentational components“. Tyto komponenty mají na starost pouze

vykreslování obsahu. Obsahují pouze stav aplikace, který je spojen se zmíněným vykreslováním obsahu (např. otevřeno / zavřeno). (65) Veškerý ostatní stav, tyto argumenty pouze přebírají jako argumenty od „Container components“, které jsou vysvětleny hned v následujícím bodě.

- **Pages** – Zde se nacházejí všechny stránky aplikace. Tyto stránky plní funkci „Container components“. Tyto komponenty spravují stavy aplikace, a mohou obsahovat mnoho „presentational components“, jejichž stavy také spravují. (65) Toto rozdělení na „container a presentational components“, je velice výhodné z hlediska udržování kódu. Vývojář totiž nemusí složitě hledat jednotlivé stavy napříč celou aplikací, stačí mu pouze nahlédnout do těchto „container“ komponent a následně změnit co potřebuje.
- **Shared** – V této složce jsou veškeré funkce, které sdílí vícero komponent.
- **Store** – Zde se nachází úložiště knihovny Redux
- **Theme** – Tato složka je dána Ionicem a nachází se v ní CSS soubor, pro globální stylování.
- **UI** – Složka UI obsahuje pouze komponenty, které vykreslují nějaký určitý HTML element. Například Input.
- **App.js** – Tento soubor obsahuje routování, skrze které propojuje jednotlivé stránky aplikace.
- **Index.js** – V tomto souboru je komponenta vytvořená v souboru App.js, napojena na element , který se nachází v složce Public, v jejím HTML souboru. Mimo to je zde také napojení na platformu Firebase a úložiště Redux. Je dobré ukázat na praktickém příkladu, jak toto propojení vypadá. V následující ukázce kódu je použita funkce „ReactDOM.render“, ta má dva argumenty, v prvním řekneme, co se má vykreslit, a druhým, k jakému elementu se má komponenta napojit. Je možné si všimnout komponenty „Provider“, takto vypadá napojení Redux úložiště. Komponenta „ReactReduxFirebaseProvider“, propojuje Redux úložiště s platformou Firebase, a nese v sobě přihlašovací údaje objekt „rrfProps“ obsahující přihlašovací údaje

k Firebase, na jejichž základě propojí aplikaci s platfromou Firebase a Firestore databází.

```
ReactDOM.render(
  <Provider store={store}>
    <ReactReduxFirebaseProvider {...rrfProps}>
      <App />
    </ReactReduxFirebaseProvider>
  </Provider>,
  document.getElementById('root')
);
```

- **Capacitor.config.json, ionic.config.json, package-lock.json, package.json** – Tyto soubory v sobě nesou informace o použitých knihovnách společně s jejich verzemi. Případně také nesou skriptovací příkazy, které lze použít například pro spuštění aplikace či kompilaci produkčních verzí pro mobilní telefony nebo web.

4.4.3 Ověřování uživatelů

Jak již bylo řečeno, pro backend aplikace je využita platforma Firebase. Ta uchovává veškerá data o vytvořených poznámkách. Z bezpečnostních důvodů ale samozřejmě není možné, aby měli všichni uživatelé přístup ke všem datům. Tudíž bylo nutné, aby měl každý uživatel jednoznačný identifikátor, na jehož základě se mu budou zobrazovat pouze data, ke kterým opravdu má mít přístup.

To znamená, že k tomuto údaji bude potřebovat přístup vícero komponent, například aplikace jako taková, která podle něj bude určovat, zdali je uživatel přihlášen, nebo obrazovka s poznámkami kde bude tento identifikátor vyslán společně s požadavkem o data, aby mohl Firebase posoudit, ke kterým datům má uživatel přístup. Je tak lepší, mít tento identifikátor uložený globálně, aby k němu mohla kterákoliv komponenta přistupovat. Pro toto úložiště jsou v aplikaci využity výše zmíněné knihovny Redux a Redux-thunk. Tyto dvě knihovny by stačily, nicméně pro ulehčení komunikace s Firebase byly použity i knihovny React-redux-firebase a redux-firestore.

4.4.3.1 Registrace uživatele

Pro vytvoření takového identifikátoru zde slouží registrace či přihlášení uživatele. Firebase pro tuto funkcionalitu nabízí několik hotových řešení, například přihlašování přes Facebook

či Google účet. V poznámkové aplikaci bylo využito přihlašování přes email. Nejdříve byla samozřejmě implementována registrace, pro ni byla vytvořena níže ukázaná funkce. Ta bere jako argument údaje o novém uživateli. Pokud je uživatel vytvořen úspěšně, obdrží aplikace od Firebase odpověď, ve které je uživatelovo ID, které následně zašle do Firestore, kde společně s tímto ID uloží i případné dodatečné informace o uživateli, v tomto případě je to jméno uživatele. Pokud vše proběhne v pořádku, je vyslána úspěšná akce, která do globálního úložiště uloží údaje o uživatelovi. Pokud proběhne neúspěšně, vyšle se neúspěšná akce společně s chybovou zprávou.

```
export const signUp = newUser => {
  return (dispatch, getState, getFirebase) => {
    const firebase = getFirebase();
    firebase
      .auth()
      .createUserWithEmailAndPassword(newUser.email,
newUser.password)
      .then(res => {
        return firebase
          .firestore()
          .collection('users')
          .doc(res.user.uid)
          .set({
            userName: newUser.name
          });
      })
      .then(() => {
        dispatch({ type: actionTypes.SIGNUP_SUCCESS });
      })
      .catch(err => {
        dispatch({ type: actionTypes.SIGNUP_ERROR, error: err });
      });
  };
};
```

4.4.3.2 Přihlášení existujících uživatelů

Níže uvedený kód vezme zadané přihlašovací údaje z přihlašovacího formuláře a zavolá z knihovny React-redux-firebase, funkci „firebase.login()“, která pošle zadané údaje platformě Firebase, ze které v případě, že jsou údaje správné, přijde v odpovědi mimo jiné i identifikační číslo uživatele.

```
const submitLogin = e => {
        e.preventDefault();
        firebase.login({
          email: loginForm.email.value,
          password: loginForm.password.value
        });
      };
```

4.4.3.3 Přepínání mezi stavem přihlášen/odhlášen

Přihlášení a registrace pouze řešily ověření z hlediska backendu aplikace. Bylo ještě nutné vyřešit zobrazování správných obrazovek na základě toho, zdali je uživatel přihlášen. Toho bylo docíleno pomocí Reactu a routování v React aplikacích. V následující ukázce kódu jsou proměnné „routes“ přiřazeny routovací komponenty podle toho, jestli existuje identifikační číslo uživatele.

Kód je tak i jasnou ukázkou toho, jak React funguje a k čemu je v Ionic aplikaci využíván. Popisované sledování stavu „auth.uid“, a následné přiřazení hodnot proměnné „routes“, má na starosti právě React.

```
    let routes = (
        <IonRouterOutlet id="main">
          <Route path="/" component={Auth} />
          <Redirect to="/" />
        </IonRouterOutlet>
      );

      if (auth.uid) {
        routes = (
          <Fragment>
            <HomeMenu />
            <IonTabs>
              <IonRouterOutlet id="main">
                <Route
                  path="/note/usernote/:id"
                  exact={true}
                  render={props => (
                    <Note
                      {...props}
                      onIsNoteOpenChange={onIsNoteOpenChange}
                      isNewNote={false}
                    />
                  )}
                />
```

```
                <Route
                  path="/shared/sharednote/:id"
                  exact={true}
                  render={props => (
                    <Note                                          {...props}
onIsNoteOpenChange={onIsNoteOpenChange} />
                  )}
                />
                <Route
                  path="/newnote"
                  exact={true}
                  render={props => (
                    <Note
                      {...props}
                      isNewNote={true}
                      onIsNoteOpenChange={onIsNoteOpenChange}
                      testauth={auth}
                    />
                  )}
                />
                <Route path="/home" exact={true} component={Home} />
                <Route  path="/search"  component={Search}  exact={true}
/>
                <Route  path="/shared"  component={Shared}  exact={true}
/>
                <Route path="/" component={Home} exact={true} />
                <Redirect to="/" />
              </IonRouterOutlet>
              {tabButtons}
            </IonTabs>
          </Fragment>
        );
      }
```

4.4.4 Tvorba poznámky

Tvorba poznámky obnášela několik úkolů. Bylo nutné uživatelovi umožnit psát a formátovat text, pořídit fotku či vybrat fotku z galerie telefonu a také umožnit sdílení poznámky.

4.4.4.1 Text poznámky

Toto bylo nejtěžší částí na celé aplikaci. Umožnit uživateli psát klasický text je vcelku jednoduché, stačí klasický input, ale pro vytváření poznámek, je klasický input, který nelze formátovat, nedostačující. Je dobré připomenout, že je tvořena aplikace za pomoci webových technologií. Jednou z těchto technologií, je jazyk HTML, který vývojářovi

umožňuje psát text, a pomocí daných značek udávat, který text má být například nadpis. Je samozřejmě nepředstavitelné po uživatelích aplikace chtít, aby vkládali text poznámky ve formě HTML kódu.

Bylo třeba vytvořit editor, který bude zmíněné HTML značky vkládat za něj, a uživatel uvidí již jen výsledný naformátovaný text. Pro tyto účely existují takzvané „What you see is what you get“ editory, které přesně toto umožňují. (66)

Tyto editory využívají atribut „contenteditable“, který lze vložit do HTML elementu, což umožní daný element editovat. (67) Tento atribut má ale spoustu chyb, například nestandardizované chování formátu textu. Pokud například uživatel vytvoří tučný text, tak některý prohlížeč tučné písmo interpretuje pomocí elementu <b>, jiný prohlížeč zase pomocí elementu . (68) Toto chování je problematické a z toho důvodu byl využit editor Quilljs, který zaručuje podporu ve všech hlavních prohlížečích i na mobilních zařízeních. (69) Mimo to také poskytuje řadu možností pro formátování, například nadpisy do třetí úrovně, tučný text a číslované či nečíslované seznamy. Pro implementaci Quilljs do poznámkové aplikace byla využita komponenta react-quill.

Implementace vypadá následovně. Proměnná „noteText“ je React stav. Je zde řečeno, že komponenta StyledReactQuill při prvním načtení zobrazí veškerý obsah stavu „noteText“, při jakékoliv změně v této komponentě, bude zavolána funkce „handleNoteTextChange“, která aktualizuje hodnotu stavu „noteText“. V následující ukázce kódu záměrně není uvedena funkce „handleNoteTextChange“, protože tato funkce má na starosti víc než jen aktualizaci stavu, proto bude uvedena až v následující kapitole „Práce s obrázky v poznámce“. Také je zde funkce „handleSelectionChange“, která pouze hlídá pozici kurzoru v rámci komponenty StyledReactQuill. Proměnná „quillRef“ obsahuje skutečný HTML obsah komponenty „StyledReactQuill“.

```
<StyledReactQuill
                defaultValue={noteText}
                onChange={handleNoteTextChange}
                onChangeSelection={handleSelectionChange}
                ref={quillRef}
                modules={modules}
              />
```

Za povšimnutí stojí název komponenty. „StyledReactQuill“ je zvolen proto, že je zde použito vlastní stylování za pomoci knihovny styled-components. V ukázce je možné si všimnout určitých odlišností od klasického CSS stylování. Ale v podstatě je zde převzata

komponenta „ReactQuill“, u které je řečeno, že třída „quill-editor“, která se nachází uvnitř této komponenty, bude mít velikost písma 30 pixelů.

```
const StyledReactQuill = styled(ReactQuill)`
  &.quill-editor {
    font-size: 30px;
  }
`;
```

4.4.4.2 Práce s obrázky v poznámce

Bylo nutné přistoupit ke kameře telefonu. Pro tuto funkcionalitu byl využit Capacitor, o kterém bylo hovořeno v teoretických východiscích. V následující ukázce kódu je ukázána funkce, která ke kameře přistupuje. V první podmínce je možné si všimnout, že se ověřuje, zdali vůbec zařízení kameru má. Dnes samozřejmě každý telefon kameru má, proto se z podmínky pouze vrací „console.error“. Je tak učiněno z důvodu budoucího rozvoje aplikace. Ionic aplikace totiž, jak již bylo řečeno, funguje i v klasických prohlížečích, kde ne každé zařízení kameru má. Dále je v kódu uvedena kvalita fotografie, zdroj odkud fotografie pochází, výška a šířka fotografie, a v neposlední řadě také výsledný formát fotografie. Pro tento formát byl zvolen base64. Tento typ je zvolen kvůli následnému nahrání obrázku na platformu Firebase.

```
const handleTakePhoto = () => {
    if (!Capacitor.isPluginAvailable('Camera')) {
      return console.error('Camera not available');
    } else {
      Plugins.Camera.getPhoto({
        quality: 50,
        source: CameraSource.Camera,
        correctOrientation: true,
        height: 320,
        width: 200,
        resultType: CameraResultType.Base64
      })
        .then(photo => {
          uploadPhoto(photo);
        })
        .catch(error => {
          console.error(error);
          return false;
        });
    }
  };
```

Po úspěšném pořízení fotky je nejdříve nutné danou fotku nahrát úložiště Firebase. V ukázce kódu je nejdříve vytvořena reference pro místo, kde bude fotka uložena, společně s jejím jménem. To musí být dopředu definováno, takže bylo nutné využít nějaký generátor jednoznačných jmen. Na to byla nakonec zvolena knihovna „uuid“.

```
  const uploadPhoto = photo => {
        const ref =
firebase.storage().ref(`images/${uuid()}.${photo.format}`);
        const uploadTask = ref.putString(`${photo.base64String}`,
'base64');
        uploadTask.on(
          'state_changed',
          snapshot => {},
          err => {
            console.log(err);
          },
          () => {
            console.log('Image uploaded');
            uploadTask.snapshot.ref.getDownloadURL().then(url => {
              insertPhotoToNote(url);
            });
          }
        );
      };
```

Po nahrání na Firebase bylo také třeba vyřešit, jak vložit obrázek na správné místo v poznámce. Samotný text poznámky je totiž uložen ve Firestore databázi, kdežto obrázky jsou ve Firebase úložišti. V následujícím kódu se obrázek uloží buďto na místo kurzoru, či na konec poznámky (v případě, že uživatel do poznámky neklikl).

```
  const insertPhotoToNote = url => {
        if (cursorPosition || cursorPosition === 0) {
          quillRef.current.getEditor().insertEmbed(cursorPosition,
'image', url);
        } else {
          quillRef.current
            .getEditor()
            .insertEmbed(quillRef.current.getEditor().getLength(),
'image', url);
        }
      };
```

Tímto ale práce s poznámkou nekončí. Výše popsaný kód se stará pouze o správné vložení obrázku na uživatelově straně, bylo tedy třeba ještě vyřešit aktualizování obsahu poznámky ve Firestore databázi. Pro tuto aktualizaci byl využit fakt, že vložení obrázku

v předchozím kódu, vyvolá událost změny komponenty. V následující ukázce je ve spodnější části funkce „handleNoteTextChange“, která se spustí pokaždé, kdy se v textovém editoru stane nějaká změna. V této funkci je nejdříve řešeno mazání již existujících obrázků. To funguje tak, že se pro aktuální obsah poznámky a obsah poznámky před vyvolanou změnou zavolá funkce, která v obsazích poznámek hledá text „<img src=“, z nich si bere hodnoty uložené v atributu „src“ a utváří z nich pole těchto hodnot. Je dobré říci, že tato funkce vybírá opravdu jen skutečné HTML obrázky. Quilljs totiž veškerý text vložený do jeho komponenty kontroluje, takže hranaté závorky (> nebo <), které vloží uživatel, jsou předělány na jejich znakové entity.

Po vykonání funkce „getAllImages“ je porovnáno množství obrázků, které se nachází v novém a starém obsahu poznámky. Pokud se tyto délky neshodují, tak jsou přebytečné obrázky vymazány. Zde bylo třeba myslet i na to, že uživatel může buď obrázky mazat postupně tím, že drží klávesu pro mazání, také ale může označit více fotek najednou a následně je hromadně smazat. Po této úpravě je zavolána funkce „handleSubmitNote“, která vloží data do Firestore databáze. Funkce pro vložení do Firestore není moc zajímavá, a tak zde není uvedena. Nicméně nutno dodat, že data nevloží do databáze po každé změně, ale až ve chvíli, kdy uživatel nevkládá žádný nový obsah po dobu 1,5 sekundy.

```
const getAllImages = str => {
        let regex = new RegExp('<img .*?src="(.*?)"', 'gi'),
          result,
          res = [];
        while ((result = regex.exec(str))) {
          res.push(result[1]);
        }
        return res;
      };


const handleNoteTextChange = content => {
        if (noteText !== '') {
          const newImageArray = getAllImages(content);
          const oldImageArray = getAllImages(noteText);
          if (oldImageArray.length > newImageArray.length) {
            const deletedImages = oldImageArray.filter(
              image => !newImageArray.includes(image)
            );
            deletedImages.forEach(image => {
              deleteImageFromFirebase(image);
            });
          }
        }
```

```
        setNoteText(content);
        if (noteHeading && content) {
          handleSubmitNote(
            noteHeading,
            content,
            note,
            uid,
            ownerName,
            isNew,
            match.params.id
          );
        }
      };
```

4.4.4.3 Sdílení poznámek

Všechny poznámky obsahují informace o tom, kdo je vytvořil. Autor poznámky má samozřejmě od začátku k poznámce přístup. Mimo to je mu také umožněno k poznámce přidávat další uživatele, kterým chce povolit přístup k poznámce. K jejich přidání mu pouze stačí v příslušném okně zadat jejich email.

V následující ukázce kódu aplikace převezme input uživatele (e-mailovou adresu uživatele, se kterým má být poznámka sdílena) a k poznámce přidá tento údaj k poli „collaborators“, případně pokud toto pole zatím neexistuje, tak ho vytvoří.

```
    const submitShareForm = e => {
        e.preventDefault();
        let isValid = true;
        if (shareInput && note.id) {
          let share;
          if (note.collaborators) {
            share = {
              id: note.id,
              collaborators: [...note.collaborators, shareInput],
              updatedAt: new Date()
            };
          } else {
            share = {
              id: note.id,
              collaborators: [shareInput],
              updatedAt: new Date()
            };
          }
          if (note.collaborators) {
            note.collaborators.forEach(collab => {
              if (collab === shareInput) {
```

```
                isValid = false;
              }
            });
          }
          if (isValid) {
            setNote(prevState => {
              return        {        ...prevState,        collaborators:
[...share.collaborators] };
            });
            firestore.update({  collection:  'notes',  doc:  share.id  },
share);
            setShareInput();
          }
        }
      };
```

5 Výsledky a diskuse

Poznámkových aplikací je samozřejmě celá řada. Mou motivací ale bylo vytvořit aplikaci s co nejjednodušším uživatelským rozhraním. Řada lidí, včetně mě vytváří krátké poznámky, například seznam úkolů na další den či zaznamenání nějakého nápadu. Právě pro tuto skupinu lidí je aplikace určena.

Výsledná aplikace tak umožňuje uživatelovi psát textové poznámky, případně i pořídit fotku, či vybrat obrázek z galerie. Výsledné poznámky ukládá do databáze Firestore, díky čemuž je v aplikaci umožněno sdílení poznámek a synchronizace poznámek napříč uživatelovými zařízeními.

Jedná se o multiplatformní aplikaci, která je zatím určena pro mobilní platformy iOS a Android. Aplikace je funkční, ale má i svá slabá místa. Prvním takovým místem je synchronizace s Firestore databází. V případě neshody mezi verzí poznámky uložené v databázi a verzí v uživatelově zařízení se jako platná verze považuje ta s nejaktuálnějším datem úpravy. To není vyloženě špatně (viz. kapitola „4.1.5 Offline funkce a synchronizace dat“). Možná by ale stálo za snahu určité vylepšení, například by se v případě takovéto kolize mohly vytvořit dvě verze, z nichž by si autor poznámky mohl vybrat tu, kterou považuje za správnou.

Dalším slabším místem je zobrazování a nahrávání obrázků. Ty jsou uloženy v úložišti Firebase, které v zařízení nevytváří lokální kopii, což způsobuje, že je pořizování a nahrávání obrázků přístupné pouze ve chvíli, kdy je uživatel online. Uživatel je o této skutečnosti samozřejmě zpraven, avšak jako další možné vylepšení aplikace se nabízí vytvoření lokální kopie alespoň doposud načtených a nově vytvořených obrázků.

Aplikace také byla otestována řadou uživatelů. Ti v první řadě vznesli připomínky k drobným nedostatkům, které byly následně odstraněny. Jednalo se například o nejasnou tvorbu hesla, chyběla instrukce, že heslo musí mít minimálně 6 znaků a uživatelé tak nechápali, proč nemůžou účet vytvořit. Dalším požadavkem byla tvorba poznámky bez nadpisu. V původní verzi byla poznámka automaticky vymazána v případě, že neobsahuje nadpis či text poznámky. Ve finální verzi se poznámka smaže pouze tehdy, pokud nemá žádný obsah. V případě chybějícího nadpisu je na jeho místo vložen text „bez názvu“. Mimo menší nedostatky uživatelé také vznášeli požadavky na přidání nových funkcí. Aplikace tak v budoucnu může být rozšířena například o umožnění tvorby audio poznámky, či přidání možnosti kreslení do obrázků.

Poznámková aplikace byla autorovým prvním větším počinem na poli mobilních aplikací, což z ní učinilo velice zajímavou výzvu. Během vývoje se vyskytlo pár problémů způsobených zejména tím, že se jedná o multiplatformní aplikaci. Nejzajímavější byla situace, kdy změna selekce v textovém editoru na iOS zařízení vyvolala kompletní zrušení selekce a skrytí klávesnice. Otázkou tak bylo, co dané chování způsobuje. Platforma iOS, textový editor, framework Ionic, či WebView, ve kterém aplikace běží? Nakonec bylo objeveno, že problém způsobuje framework Ionic. Bylo tak nutné toto rušení selekce vypnout. Jedná se o zajímavý problém zejména proto, že jde o modelový příklad problému, se kterým se musí vývojář multiplatformní aplikace potýkat.

Bylo velice zajímavé si vyzkoušet tvorbu takovéto aplikace. Vývoj za použití frameworku Ionic přináší řadu výhod i nevýhod. Mezi výhodami je jednoznačně rychlá tvorba uživatelského rozhraní. Framework Ionic poskytuje řadu již hotových komponentů, které automaticky přizpůsobují svůj vzhled na základě platformy, na které je aplikace spuštěna. Také přístup k nejběžnějším funkcím telefonu je velice snadný díky nativnímu mostu Capacitor. Ionic aplikace je z velké části webová aplikace, a v kombinaci s Capacitorem a Electronem je možné ji rozšířit na všechny důležité platformy. To z Ionicu tvoří perfektní nástroj pro tvorbu aplikace, kterou je nutné v krátkém časovém horizontu dostat na vícero platforem a začít testovat, zdali by o ni měli uživatelé zájem. Ionic také díky svým komponentům může značně ulehčit tvorbu čistě webových aplikací a poskytnout tak uživatelům lepší zážitek při používání webové aplikace na mobilním zařízení.

Na druhou stranu právě fakt, že Ionic aplikace má velice blízko k webové aplikaci, značně ztěžuje tvorbu aplikace, která má působit nativně. Například ve chvíli, kdy Ionic aplikace potřebuje přistoupit ke kameře telefonu se občas stává, že aplikace na chvíli zamrzne. Také je nutné zmínit samotný vývoj. Při snaze vytvořit nativní vzhled aplikace je nutné přizpůsobit některé komponenty pro každou platformu zvlášť. Občas se také vyskytne chyba, která se vyskytuje pouze na některé z platforem a co víc, jsou i případy, kdy se liší chování mezi aplikací spuštěnou v prohlížeči a aplikací spuštěnou ve WebView toho samého zařízení. Ionic tak může značně zkomplikovat a prodloužit vývoj aplikací, které mají působit opravdu nativně a poskytovat řadu nativních funkcí. U takových aplikací je tak lepší zvolit nativní vývoj.

6 Závěr

Dílčím cílem bakalářské práce bylo popsat framework Ionic, konkrétně jeho možnosti při tvorbě multiplatformních aplikací. Tento cíl byl postupně splněn v teoretických východiscích práce. V těchto východiscích byly v úvodu představeny všechny možnosti vývoje aplikací. Konkrétně se jednalo o vývoj nativní, multiplatformní či vývoj progresivní webové aplikace. Následoval popis nejpoužívanějších frameworků pro multiplatformní vývoj aplikací, kde byly společně s frameworkem Ionic také představeny frameworky Xamarin, Flutter, React Native a Apache Cordova. Teoretická východiska byla zakončena celkovým shrnutím vývoje aplikací. V tomto shrnutí autor na základě nabitých znalostí uvádí, v jakých případech je vhodné vyvinout progresivní webovou aplikaci, nativní aplikaci, či multiplatformní aplikaci.

Hlavním cílem bakalářské práce bylo vyvinutí multiplatformní aplikace pro zaznamenávání textových a foto poznámek. Vývoj této aplikace byl popsán v druhé části práce od počátečních požadavků na funkce aplikace, až po samotné naprogramování.

Aplikace byla také otestována reálnými uživateli. Na základě tohoto testování a nápadů autora ji tak lze nadále rozšiřovat. Jedno z možných rozšíření je přidání dalších možností pro zaznamenávání poznámek, konkrétně by se mohlo jednat o možnost pořizovat video či audio poznámky. Dalším možným zdokonalením je lepší synchronizace dat. Aplikace by tak mohla uživateli umožnit náhled do proběhlých změn dané poznámky. Uživatel by následně mohl proběhlé změny zrušit, nebo se vrátit k některé z předchozích verzí poznámky.

Výsledná aplikace demonstruje možnosti frameworku Ionic, jehož využití je dle názoru autora vhodné v několika případech. První se nabízejí aplikace, které potřebují přistoupit k řadě nativních funkcí, ale nejsou graficky náročné. Druhá situace je zatím s otazníkem, avšak pokud v budoucnu Ionic začne oficiálně podporovat propojení s frameworkem Electron, stane se také vhodný pro začínající firmy, které nemají velký počáteční kapitál a potřebují v rozumném čase vyvinout aplikaci pro všechny podstatné platformy. V neposlední řadě je framework Ionic vhodné použít i ve webových aplikacích, kterým poskytuje řadu již připravených komponent, vhodných zejména pro menší obrazovky.

7 Citovaná literatura

1. Number of smartphone users worldwide from 2016 to 2021. *statista.* [Online] [Citace: 17. Listopad 2019.] https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/.

2. statcounter. *Mobile Operating System Market Share Worldwide.* [Online] [Citace: 17. Listopad 2019.] https://gs.statcounter.com/os-market-share/mobile/worldwide .

3. Białecki , Paweł . 10 steps to become a professional iOS developer. *Medium.* [Online] 25. Květen 2015. [Citace: 17. Listopad 2019.] https://medium.com/app-coder-io/10-steps-to-become-a-professional-ios-developer-11b82b6aea4c.

4. Allison, Stadd. Udacity. *How to Become an Android Developer* . [Online] 29. Květen 2015. [Citace: 17. Listopad 2019.] https://blog.udacity.com/2015/05/become-android-developer.html .

5. Nath, Deepu S. Medium. *4 important points to know about Progressive Web Apps (PWA)* . [Online] 24. Květen 2017. [Citace: 17. Listopad 2019.] https://medium.com/@deepusnath/4-points-to-keep-in-mind-before-introducing-progressive-web-apps-pwa-to-your-team-8dc66bcf6011.

6. Google Inc. FAQ. *Flutter.* [Online] [Citace: 29. Leden 2020.] https://flutter.dev/docs/resources/faq.

7. © 2020 GitHub, Inc. flutter / flutter. *GitHub.* [Online] [Citace: 29. Leden 2020.] https://github.com/flutter/flutter.

8. —. facebook / react-native. *GitHub.* [Online] [Citace: 29. Leden 2020.] https://github.com/facebook/react-native.

9. —. ionic-team / ionic. *GitHub.* [Online] [Citace: 29. Leden 2020.] https://github.com/ionic-team/ionic.

10. Google Inc. Flutter. *Flutter.* [Online] [Citace: 1. Únor 2020.] https://flutter.dev/.

11. Gupta, Deepak K. Medium. *Flutter : From Zero To Comfortable.* [Online] 2. Březen 2018. [Citace: 1. Únor 2020.] https://proandroiddev.com/flutter-from-zero-to-comfortable-6b1d6b2d20e.

12. Google Inc. Flutter. *Introduction to widgets* . [Online] [Citace: 1. Únor 2020.] https://flutter.dev/docs/development/ui/widgets-intro.

13. Leler, Wm. Hackernoon. *What's Revolutionary about Flutter* . [Online] 20. Srpen 2017. [Citace: 1. Únor 2020.] https://hackernoon.com/whats-revolutionary-about-flutter-946915b09514.

14. Gehrer, Ralf. How Flutter works under the hood and why it is game-changing . *Medium.* [Online] 5. Květen 2019. [Citace: 2. Únor 2020.] https://medium.com/@ralfgehrer/how-flutter-works-under-the-hood-and-why-it-is-game-changing-2335954a5bfc.

15. HOANG, TRAN SON. Medium. *Just-in-Time (JIT) and Ahead-of-Time (AOT) Compilation in Angular.* [Online] 2. Září 2019. [Citace: 2. Únor 2020.] https://levelup.gitconnected.com/just-in-time-jit-and-ahead-of-time-aot-compilation-in-angular-8529f1d6fa9d.

16. Inc., SteelKiwi. Flutter: Pros and Cons for Seamless Cross Platform Development. *HACKERNOON.* [Online] 11. září 2018. [Citace: 7. únor 2020.] https://hackernoon.com/flutter-pros-and-cons-for-seamless-cross-platform-development-c81bde5a4083.

17. Google Inc. Web support for Flutter . *Flutter.* [Online] [Citace: 7. únor 2020.] https://flutter.dev/web.

18. Microsoft. Xamarin.Android . *Microsoft.* [Online] [Citace: 5. Únor 2020.] https://docs.microsoft.com/cs-cz/xamarin/android/.

19. —. Xamarin.iOS . *Microsoft.* [Online] [Citace: 5. Únor 2020.] https://docs.microsoft.com/en-us/xamarin/ios/.

20. PS, Divya. Xamarin Native vs. Xamarin.Forms: How to Choose. *dzone.* [Online] 21. Srpen 2019. [Citace: 5. Únor 2020.] https://dzone.com/articles/xamarin-native-vs-xamarinforms-how-to-choose.

21. Microsoft. What is Xamarin.Forms? . *Microsoft.* [Online] 18. Září 2019. [Citace: 5. Únor 2020.] https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin-forms.

22. © wpf-tutorial.com 2007-2020. What is XAML? . *WPF Tutorial.* [Online] [Citace: 5. únor 2020.] https://www.wpf-tutorial.com/xaml/what-is-xaml/.

23. Microsoft. What is Xamarin? . *Microsoft.* [Online] 16. červen 2019. [Citace: 5. únor 2020.] https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin.

24. —. iOS App Architecture . *Microsoft.* [Online] 21. březen 2017. [Citace: 5. Únor 2020.] https://docs.microsoft.com/en-us/xamarin/ios/internals/architecture.

25. Copyright © 2020 Apple Inc. Objective-C Runtime . *Developer.* [Online] [Citace: 5. únor 2020.] https://developer.apple.com/documentation/objectivec/objective-c_runtime.

26. Microsoft. Architecture. *Microsoft.* [Online] 25. Duben 2018. [Citace: 5. únor 2020.] https://docs.microsoft.com/en-gb/xamarin/android/internals/architecture.

27. Lauschenko, Oleksandr. The Pros and Cons of Xamarin for Cross-Platform Development. *HACKERNOON.* [Online] 5. říjen 2018. [Citace: 7. únor 2020.] https://hackernoon.com/the-pros-and-cons-of-xamarin-for-cross-platform-development-2a31c6610792.

28. AltexSoft. The Good and The Bad of Xamarin Mobile Development. *altexsoft.* [Online] 26. duben 2019. [Citace: 7. únor 2020.] https://www.altexsoft.com/blog/mobile/pros-and-cons-of-xamarin-vs-native/.

29. Vartanova, Nika. Pros and Cons of Mobile Development with Xamarin. *iflexion.* [Online] 16. prosinec 2019. [Citace: 7. únor 2020.] https://www.iflexion.com/blog/xamarin-pros-and-cons.

30. elylucas, camwiegert. ionicframework. *What is Ionic Framework?* [Online] 11. Listopad 2019. [Citace: 2. Únor 2020.] https://ionicframework.com/docs/intro.

31. Lynch, Max. Ionic. *Introducing Ionic 4: Ionic for Everyone.* [Online] 23. Leden 2019. [Citace: 2. Únor 2020.] https://ionicframework.com/blog/introducing-ionic-4-ionic-for-everyone/.

32. © 2005-2020 Mozilla and individual contributors. MDN web docs. *Web Components.* [Online] [Citace: 4. Únor 2020.] https://developer.mozilla.org/en-US/docs/Web/Web_Components.

33. Introduction. *WEBCOMPONENTS.ORG.* [Online] [Citace: 4. Únor 2020.] https://www.webcomponents.org/introduction.

34. dwieeb, brandyscarney, rtpHarry, camwiegert. Installing Ionic . *ionicframework.* [Online] [Citace: 4. Únor 2020.] https://ionicframework.com/docs/installation/cli.

35. —. Ionic Packages. *ionicframework.* [Online] [Citace: 4. Únor 2020.] https://ionicframework.com/docs/installation/cdn.

36. Lynch, Max. Announcing Capacitor 1.0. *ionicframework.* [Online] 22. Květen 2019. [Citace: 5. Únor 2020.] https://ionicframework.com/blog/announcing-capacitor-1-0/.

37. Copyright © 2020 Apple Inc. WKWebView . *Developer.* [Online] [Citace: 4. Únor 2020.] https://developer.apple.com/documentation/webkit/wkwebview.

38. dotnetkow, jcesarmobile. Using Capacitor Plugins . *capacitor.* [Online] [Citace: 5. únor 2020.] https://capacitor.ionicframework.com/docs/basics/using-plugins.

39. mlynch. Building Electron Apps with Capacitor . *capacitor.* [Online] [Citace: 6. Únor 2020.] https://capacitor.ionicframework.com/docs/electron.

40. dotNetkow, mhartington, elylucas. React Navigation . *ionicframework.* [Online] 22. únor 2019. [Citace: 5. únor 2020.] https://ionicframework.com/docs/react/navigation.

41. elylucas, dotNetkow. React Lifecycle . *React Lifecycle* . [Online] 19. listopad 2020. [Citace: 5. Únor 2020.] https://ionicframework.com/docs/react/lifecycle.

42. Facebook. React: A JavaScript library for building user interfaces. [Online] [Citace: 17. Listopad 2019.] https://reactjs.org/.

43. Asia. Development With Ionic Framework — Pros and Cons . *APPSTRONAUTS.* [Online] 29. říjen 2018. [Citace: 7. únor 2020.] https://www.appstronauts.co/development-with-ionic-framework-pros-and-cons/.

44. UKAD. 6 Pros and 3 Cons of Ionic Development. *UKAD.* [Online] [Citace: 7. únor 2020.] https://www.ukad-group.com/blog/6-pros-and-3-cons-of-ionic-development/.

45. AltexSoft. The Good and the Bad of Ionic Mobile Development. *altexsoft.* [Online] 21. květen 2019. [Citace: 7. únor 2020.] https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-ionic-mobile-development/.

46. Griffith, Chris. What is Apache Cordova? *ionicframework.* [Online] [Citace: 6. únor 2020.] https://ionicframework.com/resources/articles/what-is-apache-cordova.

47. © 2012, 2013, 2015 The Apache Software Foundation. Overview . *Cordova.* [Online] [Citace: 6. únor 2020.] https://cordova.apache.org/docs/en/latest/guide/overview/index.html.

48. Facebook. React Native . [Online] 2019. [Citace: 17. Listopad 2019.] https://facebook.github.io/react-native/ .

49. Frachet, Marvin. Hackernoon. *Understanding the React Native bridge concept* . [Online] 10. Listopad 2017. [Citace: 17. Listopad 2019.] https://hackernoon.com/understanding-react-native-bridge-concept-e9526066ddb8 .

50. Evkoski , Blagoja. React Native: What it is and how it works . *Medium.* [Online] 2017. [Citace: 17. Listopad 2019.] https://medium.com/we-talk-it/react-native-what-it-is-and-how-it-works-e2182d008f5e .

51. Chrzanowska, Natalia. React Native Pros and Cons - Facebook's Framework in 2019 (update). *netguru.* [Online] 16. květen 2019. [Citace: 7. únor 2020.] https://www.netguru.com/blog/react-native-pros-and-cons.

52. Harutyunyan, Nairi. Sharing code between React and React Native using SDK. *Medium.* [Online] 6. květen 2018. [Citace: 7. únor 2020.] https://medium.com/hackernoon/share-code-between-react-and-react-native-using-sdk-85c2dab552f8.
53. AltexSoft. The Good and the Bad of ReactJS and React Native. *altexsoft.* [Online] 10. září 2018. [Citace: 7. únor 2020.] https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-reactjs-and-react-native/.
54. Dyvliash, Vadim. React Native Advantages. *BELITSOFT.* [Online] 20. prosinec 2018. [Citace: 7. únor 2020.] https://belitsoft.com/react-native-development/react-native-advantages.
55. Where Do Cross-Platform App Frameworks Stand in 2020? *netsolutions.* [Online] 6. leden 2019. [Citace: 6. únor 2020.] https://www.netsolutions.com/insights/cross-platform-app-frameworks-in-2019/.
56. Klubnikin, Andrew. Cross-platform vs. Native Mobile App Development: Choosing the Right Dev Tools for Your App Project. *Medium.* [Online] 24. květen 2017. [Citace: 6. únor 2020.] https://medium.com/all-technology-feeds/cross-platform-vs-native-mobile-app-development-choosing-the-right-dev-tools-for-your-app-project-47d0abafee81.
57. Korolev, Sergey. Mobile App Development Approaches Explained. *railsware.* [Online] 19. červen 2019. [Citace: 6. únor 2020.] https://railsware.com/blog/native-vs-hybrid-vs-cross-platform/.
58. BLOG . *Skelia.* [Online] 12. duben 2019. [Citace: 6. únor 2020.] https://skelia.com/articles/the-startup-dilemma-native-vs-cross-platform-apps/.
59. Facebook Inc. Who's using React Native? . *React Native.* [Online] 2020. [Citace: 6. únor 2020.] https://facebook.github.io/react-native/showcase.
60. Wanyoike, Michael a Dierx, Peter. A Beginner’s Guide to npm — the Node Package Manager. *sitepoint.* [Online] 15. duben 2019. [Citace: 26. Leden 2020.] https://www.sitepoint.com/beginners-guide-node-package-manager/.
61. Copyright ©2020 Red Hat, Inc. What is event-driven architecture? *RedHat.* [Online] [Citace: 26. Leden 2020.] https://www.redhat.com/en/topics/integration/what-is-event-driven-architecture.
62. GeekyAnts. Introduction to Firebase. *Hackernoon.* [Online] 26. Prosinec 2017. [Citace: 25. Leden 2020.] https://hackernoon.com/introduction-to-firebase-218a23186cd7.

63. punyavashist. An Introduction to Firebase: Part One . *Medium.* [Online] 2. Září 2017. [Citace: 25. Leden 2020.] https://medium.com/thecyberfibre/an-introduction-to-firebase-part-one-12e47c7eab65.

64. Sharma, Ashish. Realtime Database vs. Cloud Firestore — Which Database is Suitable for your Mobile App. *Medium.* [Online] 29. Září 2018. [Citace: 25. Leden 2020.] https://medium.com/datadriveninvestor/realtime-database-vs-cloud-firestore-which-database-is-suitable-for-your-mobile-app-87e11b56f50f.

65. Copes, © 2020 Flavio. React: Presentational vs Container Components. *flavioscopes.* [Online] 21. Prosinec 2018. [Citace: 26. Leden 2020.] https://flaviocopes.com/react-presentational-vs-container-components/.

66. Rouse, Margaret. WYSIWYG (what you see is what you get). *WhatIs.com.* [Online] Březen 2011. [Citace: 25. Leden 2020.] https://whatis.techtarget.com/definition/WYSIWYG-what-you-see-is-what-you-get.

67. contributors., © 2005-2020 Mozilla and individual. contenteditable. *MDN web docs.* [Online] 19. Leden 2020. [Citace: 25. Leden 2020.] https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes/contenteditable.

68. Koszuliński , Piotrek. ContentEditable — The Good, the Bad and the Ugly. *Medium.* [Online] 13. srpen 2015. [Citace: 25. Leden 2020.] https://medium.com/content-uneditable/contenteditable-the-good-the-bad-and-the-ugly-261a38555e9c.

69. Why Quill. *Quill.* [Online] [Citace: 25. Leden 2020.] https://quilljs.com/guides/why-quill/.

8 Přílohy

Příloha 1 – CD se zdrojovými kódy poznámkové aplikace a souborem „android-verze-aplikace.apk“, ze kterého lze aplikaci nainstalovat na platformu Android.