Specifika vývoje aplikací za pomocí technologie React a React Native

Marek Novák

Bakalářská práce
2022

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Marek Novák**
Osobní číslo:	**A19078**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Specifika vývoje aplikací za pomocí technologie React a React Native**
Téma práce anglicky:	**Specifics of Application Development using React and React Native Technologies.**

Zásady pro vypracování

1. Nastudujte a stručně charakterizujte React a React Native. Dále popište jejich možné využití v praxi.
2. Popište hlavní rozdíly mezi React a React Native.
3. Implementujte aplikaci pro zadávání poznámek pomocí obou uvedených technologií.
4. Během praktické části pozorujte hlavní odlišnosti ve vývoji, implementaci kódu a popište je.
5. Porovnejte časovou náročnost vývoje těchto aplikací. Přizpůsobte webovou aplikaci tak, aby měla charakter PWA a následně za pomocí Capacitor frameworku z ní vytvořte mobilní aplikaci pro Android. Porovnejte také výhody či nevýhody těchto dvou vytvořených mobilních aplikací a popište využití těchto technologií v praxi.
6. Webovou i mobilní aplikaci otestujte na reálném mobilním zařízení (Android OS) a ověřte responzivitu.

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. GASSTON, Peter. Moderní web. Přeložil Ondřej BAŠE. Brno: Computer Press, 2015. ISBN 978-80-251-4345-2.
2. ROLDAN, Carlos. React Cookbook : create dynamic web apps with React using Redux, Webpack, Node.js, and GraphQL. Birmingham, UK: Packt Publishing, 2018. Print. ISBN 9781783980727.
3. Ward, Dan. React Native Cookbook : Step-By-step Recipes for Solving Common React Native Development Problems, 2nd Edition. Birmingham: Packt Publishing Ltd, 2019. Print. ISBN 9781788991926
4. Getting Started. React JS [online]. Menlo Park: Facebook, c2021 [cit. 2021-11-29]. Dostupné z: https://reactjs.org/docs/getting-started.html
5. Introduction. React Native [online]. Menlo Park: Facebook, c2021 [cit. 2021-11-29]. Dostupné z: https://reactnative.dev/docs/getting-started
6. Firebase Documentation. Firebase [online]. Mountain View: Google, c2021 [cit. 2021-11-29]. Dostupné z: https://firebase.google.com/docs

Vedoucí bakalářské práce: **Ing. Radek Vala, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **3. prosince 2021**
Termín odevzdání bakalářské práce: **23. května 2022**

doc. Mgr. Milan Adámek, Ph.D. v.r.
děkan

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 24. ledna 2022

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 19. 5. 2022

Marek Novák v. r.
podpis studenta

ABSTRAKT

Cílem této bakalářské práce je porovnat vývoj webové a mobilní aplikace za pomocí moderní technologie React a React Native. Porovnání je provedeno na aplikaci pro zadávání poznámek. Dalším obsahem práce jsou grafické návrhy těchto aplikací, responzivní design a ukládání dat do databáze. Hlavní porovnání se zaobírá především odlišností vývoje těchto aplikací, strukturou kódu nebo časovou náročností celého vývoje. Pro další porovnání časové náročnosti bylo využito Capacitoru, díky kterému můžeme vytvořit mobilní aplikaci z původní webové aplikace.

Klíčová slova: Vývoj webové aplikace, vývoj mobilní aplikace, React, React Native, Firebase, JavaScript, Capacitor, Android, iOS

ABSTRACT

The aim of this bachelor thesis is to compare the development of a web and mobile application using the modern technology React and React Native. The comparison is done on a note taking application. The graphical design of these applications, responsive design and data storage in the database are other topics of the thesis. The main comparison deals with the differences in the development of these applications, the structure of the code or the time consumption of the whole development. To further compare the time requirements, Capacitor was used, which allows us to create a mobile application from the original web application.

Keywords: Web application development, mobile application development, React, React Native, Firebase, JavaScript, Capacitor, Android, iOS

Tímto bych rád poděkoval Ing. Radku Valovi Ph.D. za ochotu a cenné rady při vedení mé bakalářské práce.

OBSAH

ÚVOD

V dnešní době rychlého vývoje internetu je nedílnou součástí vývoj webových a mobilních aplikací. Jedná se o součást internetu, která se stala nedílnou součástí našich životů. Je téměř nepředstavitelné, že bychom je nemohli používat na denní bázi. Hledání a dostupnost informací či komunikace s ostatními se tak v průběhu posledních let stalo samozřejmostí. Je tak tedy v zájmu všech vývojářů těchto aplikací a velkých technologických společností, aby byl vývoj, údržba či škálovatelnost tohoto typu softwaru co možná nejefektivnější. Velkým tématem je také multiplatformní vývoj aplikací. Tedy aplikace přizpůsobené pro konkrétní platformy, jako jsou nejčastěji počítače či mobilní zařízení. V tomto odvětví se podstatně výrazně angažuje *Meta* (bývalý *Facebook*), díky kterým vznikla technologie, které nese název *React*. Jelikož je *Facebook* velice rozsáhlá aplikace, která obsahuje mnoho funkcí, je nutné, aby běžel zcela bezchybně. V roce 2011 se jeden z vývojářů *Facebooku* potýkal s problémy s údržbou kódu, a tak vznikla první myšlenka *Reactu*, která údržbu kódu výrazně usnadňuje. Další z výhod *Reactu* (a nejen *Reactu*, ale i jiných *JavaScriptových frameworků*) je zkrácení napsaného kódu či usnadnění jeho rozšíření. Z *Reactu* dále vychází další z technologií, která nese název *React Native*. Jedná se o *framework*, který umožňuje vývoj aplikací pro mobilní platformu. Obě tyto technologie jsou právě tématem této bakalářské práce.

Hlavním cílem této práce je představení moderní technologie *React* a *React Native* a následné porovnání vývoje webové a mobilní aplikace za pomocí právě těchto technologií. Porovnání je v praktické části provedeno na implementaci aplikace pro zadávání poznámek. Hlavní porovnání se zabývá především odlišností při vývoji těchto aplikací, strukturou kódu či efektivitou vývoje a následné údržby. Autor si dává za cíl pochopení nejefektivnějšího, a především vhodného využití těchto technologií v praxi a rovněž se naučit vyvíjet webové a mobilní aplikace díky těmto technologiím. Veškeré poznatky nasbírané při této práci chce předat čtenáři a využít je v budoucím pracovním životě.

Tato práce je orientována na lidi, kteří se pohybují v odvětví vývoje webových a mobilních aplikací a kteří se chtějí dozvědět více o praktickém využití a specifik vývoje díky technologiím React a React Native.

I. TEORETICKÁ ČÁST

1 VÝVOJ WEBOVÝCH APLIKACÍ

Webové stránky a webové aplikace dnes používáme zcela běžně na denním pořádku a dalo by se říct, že se tak jedná o jednu z největších informačních a nejrychleji rostoucích technologií. Tato technologie se dá také popsat jako *World Wide Web (WWW)*, což je systém vzájemně propojených *hypertextových dokumentů*, které jsou přístupné pomocí internetu. Každodenní aktivity, jako je komunikace, vzdělávání, hledání informací, nakupování či vytváření obsahu, si tak nacházejí místo na internetu v podobě právě webových stránek a aplikací.

Z pohledu uživatele se webové stránky berou jako samozřejmost, takže očekávají bezproblémové dohledání informací, snadnou orientaci na stránkách či v aplikaci, přívětivý design, bezpečnost, stabilitu či optimalizaci pro jejich zařízení. Pro samotného vývojáře to znamená navrhnout a vytvořit produkt, který bude bezchybný ve všech těchto disciplínách a nejen těchto, ale i dalších, které uživatel nevidí, ale jsou důležitou součástí celého projektu. Tyto součásti jsou dále rozebrány v následujících kapitolách, kde je vysvětlen jejich princip a jakou hrají roli při tvorbě webových aplikací.

1.1 Kde se zrodil web

Původním autorem webu je *Tim Berners-Lee*, který pracoval pro institut *CERN*. Vznik můžeme datovat do roku 1989, kdy se pokusil o vytvoření konceptu pro automatizované sdílení informací mezi vědci na univerzitách a v ústavech po celém světě. Základní myšlenkou tak bylo spojit rozvíjející se technologie počítačů, datových sítí a hypertextu do výkonného a snadno použitelného globálního informačního systému. První webová stránka odkazovala na informace užitečné pro vědce v *CERNu* (například *telefonní seznam CERNu* nebo návody na používání *centrálních počítačů*). Tímto byla představena celá vize statických webových stránek, které se stále vyvíjí a dnes už je známe také ve formě dynamických webových aplikací [1].

1.2 Technologie pro vývoj webů

Webový vývoj má oproti vývoji mobilních aplikací velkou škálu technologií, které je možné využít. Vývoj a zároveň technologie můžeme rozdělit do dvou základních kategorií. Jedná se o *frontend* neboli klientskou část aplikace a *backend*, což je serverová část. Obě tyto části mají své specifické technologie, které je nutné dobře vybrat ještě před celým vývojem. Pro klientskou část zde máme technologie, jako je *HTML*, který reprezentuje základní kostru každého webu, *CSS* pro stylování *HTML elementů* nebo *JavaScript*, který můžeme rovněž

využít pro serverovou část. Serverová část obsahuje technologie jako *PHP*, *Java*, *C#*, *ASP.NET*, *.NET* a jiné. Tyto technologie nám umožňují vytvořit logiku webové aplikace, což zajišťuje funkčnost triviálních funkcí, jako je kontaktní formulář až po komplexnější funkce, jako je třeba práce s databází nebo autentizace uživatelů.

1.2.1 Frontend – klientská část

Klientská část obsahuje to, co vidí uživatel a s čím může provést nějakou interakci. Vše se zobrazuje ve webovém prohlížeči, které jsou dnes dostupné téměř na každém zařízení. Vývojáři zabývající se *frontendem* mají za úkol vytvořit přívětivý a intuitivní prostředí webových stránek a aplikací, které se musí přizpůsobit všem zařízením, na kterých je možné je navštívit. Základním kamenem je značkovací jazyk *HTML*, *CSS* a *JavaScript*, díky kterým můžeme přizpůsobit vzhled webů pro každé zařízení. Toto přizpůsobení se nazývá *responzivní design*, který je popsán níže v další kapitole společně s ostatními technologiemi.

1.2.1.1 HTML

HTML je základ každého webu. Jedná se o značkovací jazyk, který nám umožňuje vytvořit elementy tvořící obsah celé stránky. *HTML* poskytuje sadu značek pro přidávání tohoto obsahu na stránku. Obsahem chápeme text, obrázky či jiná média. Značky do sebe můžeme vzájemně vnořovat a tvořit tak komplexnější elementy. Každá značka může přebírat parametry, jako například parametr třídy, který nám umožní přiřadit třídu ke konkrétní značce a tu následně stylovat díky *CSS*. Značky můžeme rozdělit na dvě základní skupiny:

- **Párové** – Obsahuje dvě značky, kdy první tvoří začátek elementu a druhá její konec. Mezi nimi se nachází obsah (texty, obrázky, případně další vnořené elementy). Poslední značka obsahuje zpětné lomítko, které označují konec elementu. Jako příklad může být základní značka pro vytvoření nadpisu: <h1>Ukázka nadpisu</h1>.
- **Nepárové** – Tyto značky se neuzavírají a nenesou tak v sobě žádný obsah. Typickým příkladem může být
, tedy značka pro zalomení řádku textu [2].

1.2.1.2 CSS

CSS je zkratka pro *Cascading Style Sheets*, neboli kaskádové styly. Jedná se o jazyk stylů, který umožňuje popsat vzhled a formátování dokumentu napsaného v *jazyce HTML*. Tento typ stylování se využívá k designování a rozvržení webových stránek. Styly můžeme k elementu přiřadit buď využitím *atributů třídy*, *identifikátoru* nebo názvu samotné *HTML značky* a píšeme je zvlášť do odděleného souboru s koncovkou *.css*.

1.2.1.3 JavaScript

JavaScript je programovací jazyk, který umožňuje implementaci komplexnějších funkcí na webové stránce. Kdykoliv, začne mít stránka dynamický charakter a přestane být pouze statická, například manipulace s multimédii, animace nebo aktualizace obsahu, je to pravděpodobně právě díky *JavaScriptu*. Tvoří tak třetí vrstvu webových technologií, kde doplňuje *HTML* a *CSS*, které tak vzájemně tvoří uživatelské rozhraní. *JavaScript* je stále více populární, a to zejména díky kvalitním a populárním frameworkům, jako je *React*, *Angular* nebo *Vue.js*. Své místo si také našel při vývoji mobilních aplikací, což je přívětivé, jelikož vývojáři dokážou s jedním programovacím jazykem pokrýt jak webovou, tak mobilní platformu [3].

1.2.2 Backend – serverová část

Backend se rozumí jakákoliv část webové stránky nebo softwarového programu, kterou uživatelé nevidí. Tohle je právě jeho klíčová vlastnost, díky které můžeme izolovat funkce, do kterých nemá mít uživatel přístup a můžeme tak například zpracovávat data. V programátorské terminologii se dá *backend* charakterizovat jako vrstva přístupu k datům, zatímco *frontend* je vrstva prezenční. V dnešní době se webové stránky chovají dynamicky, což znamená, že se obsah stránky generuje za chodu. Tento typ generování se provádí za pomocí jednoho nebo více skriptů, které se pouštějí na webovém serveru při přístupu na stránku. Skripty vygenerují obsah stránky a ty se následně odešlou do webového prohlížeče, kde se zobrazí uživateli. Veškerý proces, který se děje před zobrazením stránky ve webovém prohlížeči, je součástí *backend*. Do základních funkcí *backend* patří:

- Zpracování příchozího požadavku na webovou stránku.
- Spouštění skriptů, které generují *HTML*.
- Přístup k datům z databáze, nejčastěji přes dotazy *SQL*.
- Ukládání, odstranění nebo aktualizace dat v databázi.
- Validace příchozích dat od uživatele.
- Šifrování či dešifrování dat [4].

1.2.2.1 Firebase

Firebase spravuje a vyvíjí společnost *Google* od roku 2014, kdy tuto platformu odkoupil. *Firebase* je jednoduchá *backend* technologie, kterou v této práci využívám v kombinaci s *Reactem* a *Reactem Native*. Celá implementace je velice jednoduchá, stačí pouze přidat *Firebase* do projektu díky *JavaScriptu*. Tato platforma je známá především díky své databázi v

reálném čase, což je databáze fungující na bázi uzlů klíčových hodnot optimalizovaná pro synchronizaci dat mezi zařízením uživatele a centralizovaným úložištěm v *cloudu*. *Firebase* se také stará o odesílání a stahování dat, čímž vývojářům usnadňuje práci. *Firebase* poskytuje řadu užitečných funkcí, jako je například:

- **Autentizace** – funkce pro registraci, přihlášení a vytvoření identity uživatele. Tato funkce je obsažena v praktické části.
- **Realtime databáze** – jednoduchá databáze hostovaná v *cloudu*. Rovněž je klíčovou funkcí v praktické části, kde slouží pro ukládání dat, a proto je více rozepsána v následující kapitole.
- **Cloud storage** – oddělené úložiště pro ukládání dat, jako jsou například profilové fotky uživatelů, či jiná média.
- **Firebase hosting** – rychlá nabídka hostingu vytvořených aplikací.

1.2.2.1.1 Firebase autentizace

Autentizace je pro velkou většinu webových aplikací klíčová. Umožňuje uživatelům vidět pouze jejich obsah, přidávat obsah, zapisovat do databáze a mnoho dalších užitečných funkcí. *Firebase* se stará o bezpečnou autentizace uživatelů velice efektivně a implementace celé funkce je jednoduchá. Celá autentizace funguje díky emailu a heslu uživatele, nicméně *Firebase* umožňuje také přihlášení přes platformy třetích stran, jako je například *Google* nebo *Facebook*. Ukázka implementace přihlášení se nachází v praktické části [17].

1.2.2.1.2 Firebase realtime databáze

V aplikaci můžeme také chtít nechat uživatele vytvářet obsah a data tohoto obsahu ukládat do databáze. I pro tyto potřeby má *Firebase* řešení, a to v podobě ukládání dat na jejich *cloud*. Celá tato databáze funguje v reálném čase, což znamená, že se data aktualizují hned, jak je provedena změna. Tato aktualizace se rovněž dynamicky propisuje do samotné aplikace. V praxi to znamená, že provedená změna na jednom zařízení je hned vidět na všech ostatních zařízeních [18].

2 VÝVOJ MOBILNÍCH APLIKACÍ

Mobilní zařízení je dnes velkou součástí našich životů, které využíváme denně a samozřejmostí jsou rovněž mobilní aplikace. K samotnému vývoji můžeme přistupovat několika způsoby v závislosti na konkrétní platformě. Můžeme zvolit nativní vývoj a soustředit se tak na jednu platformu nebo jít cestou multiplatformního vývoje a pokrýt tak všechny přístupné platformy.

2.1 Mobilní platformy

Postupným vývojem se podařilo ustálit na dvou primárních platformách. Jedná se o operační systém *Android*, který vyvíjí společnost *Google* a *iOS*, který spadá pod společnost *Apple*. Další dobře známá platforma je například *Windows Phone*, nicméně tato platforma postupně upadá, a proto je tahle kapitola věnována pouze platformě *Android* a *iOS*.

2.1.1 Android

Android je operační systém založený na jádře *Linuxu* a jedná se o nejrozšířenější operační systém pro mobilní platformu. *Android* si však našel cestu i k dalším zařízením, jako jsou tablety či chytré televize. O vývoj systému se stará společnost *Google*, která celý systém nabízí jako *open source*, což umožňuje komukoliv nahlédnout do zdrojového kódu. To zajistí jak technickou dostupnost, tak legální licenční dostupnost. Tyhle faktory umožňují firmám za určitých podmínek upravit systém dle jejich potřeby a následně jej distribuovat do svých zařízení.

Cílem *Androidu* je rozvoj mobilních technologií za nižší náklady na vývoj a jeho následnou distribuci. Platforma dává k dispozici nejen operační systém, ale také kompletní řešení pro nasazení systému pro výrobce zařízení, ale také pro vývojáře aplikací poskytuje nástroje pro jejich vývoj. Primární cesta pro stahování mobilních aplikací do zařízení je *Google Play*, který rovněž spravuje *Google*. I po letech fungování *Androidu* jsou aplikace této platformy kontroverzní téma. Služba *Google Play* nemá tak přísné podmínky pro publikování aplikací, jako například jejich konkurent *Apple* na své platformě *iOS*. Vzniká tak mnoho aplikací, které jsou nekompatibilní se zařízením, některé funkce uvnitř aplikace nemusí fungovat správně nebo v sobě může aplikace obsahovat nebezpečný *malware*. Tohle může být pro některé uživatele velká překážka, a proto raději zvolí konkurenční platformu. Bezpečnost je poslední dobou u *Androidu* velké téma, situace se však začíná zlepšovat ku prospěchu jeho uživatelů [5].

2.1.2 iOS

iOS je druhý nejpoužívanější operační systém pro mobilní zařízení. Systém byl vytvořen společností *Apple*, který ho stále vyvíjí a oproti *Androidu* není nabízen jako *open source*, takže je exkluzivní pro *Apple* zařízení. Nejdříve byl vytvořen speciálně pro první *iPhone*, který byl představen v roce 2007, později byl systém upraven i pro další Apple produkty. Dnes jsou z *iOS* odvozeny další operační systémy, které pohání produkty od Applu. Patří mezi ně *iPadOS* pro *iPad* nebo *watchOS* pro hodinky *Apple Watch*. Apple se o své operační systémy dobře stará, a tak se mohou uživatelé těšit z každoroční velké aktualizaci nové verze operačního systému pro svá zařízení. Architektura systému je unixového typu a je tak odlehčenou verzí *macOS*, což je operační systém pro počítače Apple. Operační systém se také vyznačuje svou uzavřeností, takže do zařízení není možné nainstalovat jiné aplikace než ty, co jsou dostupné na *App Store*, což je distribuční služba od Applu pro stahování aplikací. Apple se tak snaží zajistit integritu celého systému a chránit své uživatele před škodlivým obsahem. Společnost také klade důraz na testování samotných aplikací, a to vše ještě předtím, než povolí vývojáři nabízet aplikaci volně ke stažení [6].

2.2 Volba přístupu pro vývoj mobilních aplikací

Výběr technologií a celkového přístupu, je důležitou součástí celého vývojového procesu. Při webovém vývoji nemusíme řešit tolik faktorů, jako je tomu u vývoje mobilních aplikací. U vývoje mobilních aplikací musíme brát v potaz zejména operační systém, na kterém daná aplikace bude fungovat. Je proto vhodné pokrýt dvě základní a zároveň nejrozšířenější platformy, kterými jsou již zmíněné *Android* a *iOS*. V následující části je popsána charakteristika konkrétních přístupů k vývoji.

2.2.1 Aplikace vytvořené nativně

Nativní mobilní aplikace jsou aplikace vyvinuté exkluzivně pro konkrétní platformu nebo operační systém. Tyto aplikace jsou vytvořené pomocí programovacích jazyků, které jsou specifické pro danou platformu. Například pro *nativní Android aplikace* se jedná o *Javu* nebo *Kotlin*, kde navíc běžně využijeme vývojové prostředí *Android Studio*. V případě *iOS* aplikací zvolíme *Swift* nebo *Objective-C* a to vše v prostředí *XCode*, které je rovněž specifické tím, že funguje pouze na Apple zařízeních. Jednou z velkých výhod nativních aplikací je přístup ke všem funkcím, které poskytuje dané zařízení, jako je fotoaparát nebo GPS. Obliba nativních aplikací stále roste, protože poskytují výjimečný uživatelský zážitek, jelikož jsou

tyto aplikace obecně vysoce výkonné. Uživatelský zážitek je také lepší díky vizuální stránce celé aplikace, která se přizpůsobuje dané platformě a může tak být pro uživatele intuitivnější. V případě, že budeme chtít nativní aplikaci jak pro platformu *iOS* tak i *Android*, může se tento přístup stát ne zcela vhodným, jelikož zde vznikají podstatně vyšší náklady na souběžný vývoj pro obě platformy.

2.2.2 Aplikace založené na webových technologiích

Tento přístup je kompletně založený na webových technologiích, jako je *HTML*, *CSS* nebo *JavaScript*. Vývojář tak nemusí mít znalost programovacích jazyků, které se využívají pro nativní vývoj, a i tak bude schopen vytvořit plnohodnotnou mobilní aplikaci. Vytvořená aplikace se bude ve výsledku chovat stejně jako okno webového prohlížeče. Díky těmto prvkům se dá tento přístup zařadit mezi nejjednodušší vývoj mobilních aplikací. Nevýhodou však může být jeho větší požadavky na výkon či nekompatibilita nativních prvků konkrétní platformy.

2.2.3 Aplikace založené na překladu do nativního kódu

S tímto přístupem se můžeme setkat při práci s *frameworky*, jako je *React Native* nebo *NativeScript*, kde pro vývoj využijeme programovací jazyk *JavaScript*. Vyznačuje se především tím, že stačí napsat pouze jednou zdrojový kód aplikace a ten se následně za běhu interpretuje podle platformy, na které je aplikace spuštěna. Tento přístup s sebou nese další výhody, jako je například využití nativních prvků, a to zcela bez nutnosti zasahovat do uživatelského prostředí každé z platforem. Bohužel jsou takhle vytvořené aplikace náročnější na výkon mobilního zařízení, stejně jako je tomu u přístupu založeném na webových technologiích. Tato práce se zabývá právě tímto přístupem, a to díky technologii *React Native*, která je zde více rozebrána v dalších kapitolách.

3 REACT

React je *JavaScriptová* knihovna pro tvorbu uživatelského rozhraní. Tato knihovna byla vytvořena společností *Meta* (dříve známá jako *Facebook*) v roce 2013. *Facebook* i nadále *React* vyvíjí a udržuje. *React* usnadňuje práci díky vytváření komponentů, které lze při vývoji opakovaně používat. Knihovna si klade za cíl zefektivnit vývoj aplikací a zjednodušit jejich následnou údržbu díky zkrácení napsaného kódu. *React* tak můžeme využívat jak pro tvorbu jednoduchých webových stránek, tak i pro komplexní webové aplikace. Díky těmto charakteristickým rysům je dnes React velice populární a disponuje také skvělou komunitou, která se neustále rozšiřuje a díky které existují návody a články, které doplňují oficiální dokumentaci [15].

3.1 Základní koncept Reactu

Tradiční vývoj uživatelského rozhraní pro webové aplikace využívá strukturu *HTML* nebo předem vytvořené šablony. *React* k vytváření uživatelského rozhraní přistupuje odlišným způsobem, a to tak, že rozděluje celou aplikaci na menší celky zvané komponenty. Tyto komponenty jsou následně vykreslovány pomocí *JavaScriptu*, tedy plnohodnotného programovacího jazyka [7].

3.2 Renderování a update rozhraní

Při klasické práci s jazykem *JavaScript* je nutné sledovat, jaká data se změnila a následně provádět změny v *DOM*, aby se zachovala jeho aktuálnost. *React* jde však jinou cestou. V momentě, kdy se inicializuje jeden z vytvořených komponentů, zavolá se metoda *render*. Tato metoda vygeneruje odlehčenou reprezentaci našeho vytvořeného *View*. Z této reprezentace se vytvoří řetězec, který se vloží do dokumentu. Když se data změní, znovu se zavolá metoda *render*. Aby se provedla aktualizace co nejefektivněji, porovnáme návratovou hodnotu z předchozího volání metody *render* s novou hodnotou a vygenerujeme minimální nutnou sadu změn, které se následně aplikují v *DOM*. Data, která vrací metoda *render* nejsou ani řetězcem, ani uzlem *DOM*, ale jedná se o odlehčený popis toho, jak by měl *DOM* vypadat a označuje se jako *virtuální DOM* [7].

3.3 DOM

DOM neboli *Document Object Model* je objektově orientovaná reprezentace dat. Toto rozhraní poskytuje standardní způsob reprezentace a přístup k dokumentům *HTML* a *XML*.

Dynamicky tak můžeme přistoupit k obsahu, struktuře a stylu konkrétního dokumentu, případně jej aktualizovat, *DOM* je vlastně vestavěné *API* rozhraní, díky kterému mají vývojáři plnou kontrolu nad webovým obsahem. K veškerým objektům můžeme přistupovat a manipulovat jimi prostřednictvím *JavaScriptu*. *DOM* umožňuje přistupovat k dokumentu jako ke stromu, kde jsou jednotlivé části spojeny pomocí uzlů. Díky *DOM API* můžeme využívat již vytvořené funkce pro lepší práci s elementy. Jedna z těchto funkcí je dobře známá *getElementById* [8].

3.4 Virtuální DOM

React nevyužívá klasický *DOM*, ale *virtuální DOM (VDOM)*, který funguje zcela nezávisle. Jedná se o koncept, kdy se virtuální reprezentace *UI* uchovává v paměti a synchronizuje se s klasickým typem *DOM*. V praxi to funguje tak, že pokud řekneme *Reactu*, v jakém stavu se má uživatelské rozhraní nacházet, React se dokáže postarat o to, aby DOM tomuto stavu odpovídal. Tímto odpadá manipulace s atributy, zpracování eventů nebo manuální aktualizace DOMu. Díky těmto funkcím dokážeme zlepšit výkon *frontend* aplikací [9].

3.5 JavaScript XML – JSX

Tohle je jedna z dalších technologií, které s sebou přináší React. *JSX* nám umožňuje vytvářet tak zvané elementy, které následně můžeme *renderovat* jako uživatelské rozhraní. *JSX* se tak dá chápat jako rozšíření JavaScriptové syntaxe, která dokáže zobrazit HTML elementy na uživatelském rozhraní. HTML tak můžeme psát přímo do JavaScriptu. Po samotné kompilaci se veškeré výrazy napsané v *JSX* souboru transpilují do čistého *JavaScriptu*. To znamená, že JSX můžeme využívat uvnitř podmínek, cyklů nebo ho také přiřazovat do proměnných. Není však nutné při práci s Reactem využívat pouze JSX, ale můžeme použít čistě jen JavaScript [10].

3.6 Znovupoužitelné komponenty

React umožňuje rozdělit uživatelské rozhraní do menších celků zvaných *komponenty*. Díky těmto komponentům můžeme celou komplexitu programu výrazně zjednodušit. Je velice důležité tyto celky vhodně a efektivně rozdělit pro jejich další využití. Komponenty jsou od sebe navzájem izolované a nezávislé. Tyto komponenty se definují zvlášť do JavaScriptových souborů a fungují tak jako znovupoužitelné části, které se do projektu importují dle potřeby. Jako příklad může být komponenta tlačítka, která se na stránce většinou využívá

hned na několika místech, a právě React umožňuje takové tlačítko vytvořit jen jednou a následně ho implementovat na požadovaná místa. Pro vytvoření takové komponenty v nejjednodušším případě stačí elementární *JavaScriptová funkce*, která bude vracet element k vykreslení. Mohou tak přebírat vstupy zvané *props* a vracet element popisující co se má vykreslit. Pro vykreslení komponenty musí funkce obsahovat *return* [16].

3.7 Props

Props jsou data, která jsou předávána do komponentů. Jsou tak ekvivalentem argumentů, které se předávají do funkcí, stejně jako je tomu i v jiných programovacích jazycích. Data se předávají z rodičovské komponenty, a to při její implementaci. Můžeme je tak použít ke konfiguraci a předávání dat do podřízené komponenty. Mimo jiné můžeme díky těmto argumentům definovat vzhled komponenty. Jedná se tak o jednosměrný tok dat [16].

3.8 Stavy

Komponenty vytvořené díky Reactu v sobě mají funkci *state*. Vykreslení jednotlivých komponentů závisí právě na jejich stavu. Stav definujeme jako objekt v *JavaScriptu*, který se může libovolně v průběhu času měnit. Stav tedy slouží pro ukládání dat aplikace, která je uložena v proměnné a která může být předána podřízené komponentě. Stav komponenty lze aktualizovat za pomocí funkce *setState*. Pokud je změna stavu validní, aktualizuje se celý komponent, který se znovu vykreslí a uchová v sobě nový stav. Stav by se neměl zaměňovat s *props*, které jsou narozdíl od stavu neměnné [11].

4 REACT NATIVE

React Native je *framework* pro tvorbu mobilních aplikací pro platformy *Android* a *iOS*. Celý framework vychází z *Reactu*. *React Native* umožňuje vývojářům vytvářet mobilní aplikace díky JavaScriptu. Výhodou frameworku je sdílení kódu mezi platformami, čímž se usnadní vývoj a případné náklady na vytvoření a údržbu aplikací. *React Native* vznikl v roce 2015, tedy dva roky po zveřejnění *JavaScriptové* knihovny *React*. Jelikož tento *framework* vychází z Reactu, může být pro vývojáře kteří již pracují s Reactem jednoduché pochopit základní principy *React Native* a začít tak vyvíjet nativní mobilní aplikace. *Framework* také podporuje nativní vývoj, z čehož vyplývá, že vývojáři mohou vytvářet uživatelská rozhraní podobná rozhraním specifické platformy, na které byla aplikace vytvořena. To znamená, že pokud vytvoříme aplikaci pro platformu *Android*, tak prvky rozhraní budou mít specifický vzhled právě pro tuto platformu. Příkladem mohou být běžná dialogová okna či tlačítka, které se pro každou platformu liší.

Podobně jako je tomu při práci s *Reactem*, i zde můžeme využít technologii známou jako *JSX*, tedy kombinaci *JavaScriptu* a *XML*. React Native dokáže zpracovat JavaScript a následně nativně *renderovat* rozhraní pro *iOS* (díky *Objective-C*) a *Android* (díky *Javě*). To znamená, že vytvořená aplikace bude vykreslovat skutečné nativní komponenty, takže bude vypadat a chovat se jako jakákoliv jiná aplikace. React Native rovněž dokáže přistoupit k funkcím jako je fotoaparát telefonu nebo poloha uživatele [12].

4.1 Vytváření komponentů

Komponenty vytvořené v Reactu Native jsou totožné s komponenty, které jsou vytvořené pomocí Reactu, ovšem s pár zásadními rozdíly, kterými jsou renderování a stylování. Komponenty vytvořené v Reactu, které slouží pro webové rozhraní využívají klasických HTML tagů (například * , <p>, *). Ovšem v případě Reactu Native se těchto tagů zbavíme a místo nich budeme využívat tagy specifické pro mobilní platformu. Základní *cross-platformní tag* je *<View>*, což je jednoduchý a flexibilní element pro tvorbu uživatelského rozhraní, který můžeme chápat jako ekvivalent tagu * *, který běžně používáme v *HTML* [12].

4.2 Stylování komponentů

Když stylujeme *HTML* elementy v Reactu, tak využíváme klasické *CSS*, jako je tomu při jakémkoliv vytváření webu. CSS se tak stává nedílnou součástí webové platformy. Když ale

pracujeme s Reactem Native a vytváříme tak aplikace pro jinou než webovou platformu, nemůžeme využít klasické *CSS*, a proto musíme zvolit jiný alternativní přístup, který nám React Native nabízí. Naštěstí jde v tomto případě o jeden standardizovaný přístup ke stylování. Jedná se o ořezanou podmnožinu *CSS*, která se zaměřuje spíše na jednoduchost než na implementaci celého rozsahu pravidel CSS, které známe z webové platformy. Oproti webu, kde se může podpora jednotlivých stylů lišit dle prohlížeče, React Native tak dokáže vynutit konzistentní podporu stylů.

Celá implementace stylu funguje za pomocí *JavaScriptu*. Všechny stylované komponenty přijímají *props* s názvem *style*, kde definujeme název, ke kterému si následně přiřadí konkrétní styly. Názvy a hodnoty jednotlivých stylů fungují podobně jako v *CSS*, jen s výjimkou, kdy každý název zapisujeme formou *camel casing* (velké písmeno pro každé slovo). Například místo *background-color*, jako bychom napsali v CSS, napíšeme *backgroundColor*. Styly můžeme také definovat jako jednoduché JavaScriptové objekty. S tím, jak ale jednotlivé komponenty nabírají na komplexitě, je vhodné spíše rozdělení stylů za pomocí *StyleSheet.create*, kde seskupíme několik stylů na jedno místo a následně jen přiřazujeme názvy k elementům, které chceme stylovat [13].

4.3 Navigace mezi obrazovkami aplikace

Je logické, že při tvorbě aplikace budeme chtít uživateli nabídnout více než jednu obrazovku s obsahem. Právě pro řízení prezentace jednotlivých obrazovek, se v Reactu Native běžně využívá knihovna *React Navigation*, která jde jednoduše nainstalovat a následně integrovat do projektu. Knihovna poskytuje snadné navigační řešení jak pro platformu *Android*, tak *iOS*. Navigace se implementuje v souboru *App.jsx*, kde se definují jednotlivé obrazovky. Každá z obrazovek může přebírají hned několik parametrů, jako je například *name* pro nastavení názvu dané obrazovky, *component* pro přiřazení konkrétního obsahu, který se má vykreslit a v neposlední řadě parameter *options*, který umožňuje přiřadit další nastavení, jako je například *title* pro vytvoření nadpisu každé obrazovky. Nadpis se následně zobrazí v hlavičce obrazovky. Hlavička se zobrazuje vždy při vytvoření nové obrazovky, ale můžeme nastavit parametr *headerShown* na *false*, který hlavičku skryje. *React Navigation* využívá nativní rozhraní, takže se celá navigace bude chovat dle nativních specifik konkrétní platformy a nabídne tak uživateli stejný požitek při navigování mezi obrazovkami, na který je zvyklý z jiných nativních aplikací [14].

4.4 Testování aplikace

Testování je nutnou součástí, ať už vytváříme jakýkoliv typ softwaru. Testování mobilních aplikací se liší od testování webových aplikací, a to zejména prostředím, ve kterém se testuje. Webovou aplikaci můžeme otestovat v běžném webovém prohlížeči, naopak testování mobilních aplikací se může stát daleko komplexnější. Testování aplikací nám umožňují nástroje jako je *Android Studio* nebo *XCode*. Při testování musíme vzít v potaz řadu proměnných, jako je například operační systém, verze operačního systému, typ zařízení a jeho hardwarové specifikace. Testování je důležité nejen na konci celého vývoje, kde se testuje konečný produkt před zveřejněním na trh, ale také během celého vývojového procesu. Testování během vývoje nám zajistí zachycení jednotlivých chyb, takže se můžeme aktivně podílet na jejich opravě ještě předtím, než se podobných chyb vyskytne více a jejich oprava pak bude podstatně složitější. *React Native* nabízí pro testování aplikací *Expo*, což je nástroj pro vývojáře, který umožní jednoduché prvotní vytvoření nové aplikace. *Expo* s sebou přináší řadu dalších výhod, jako je například:

- Otevření aplikace na několika zařízeních najednou, a to vše během toho, co na aplikaci stále pracujeme.
- K otevření aplikace není nutné mít *Android Studio* nebo *XCode*, stačí pouze stáhnout *Expo* aplikaci z *Google Play* nebo *App Store* a naskenovat vygenerovaný QR kód nebo odkaz, díky kterým se na zařízení otevře aplikace.
- Aplikaci je možné otevřít také ve webovém prohlížeči. Zde se ale nedoporučuje aplikaci testovat, jelikož prohlížeč nemusí zachytit některé chyby
- *Expo* umožňuje build *.apk* a *.ipa* souborů pro distribuci aplikace.

5 VYUŽITÍ REACTU A REACTU NATIVE

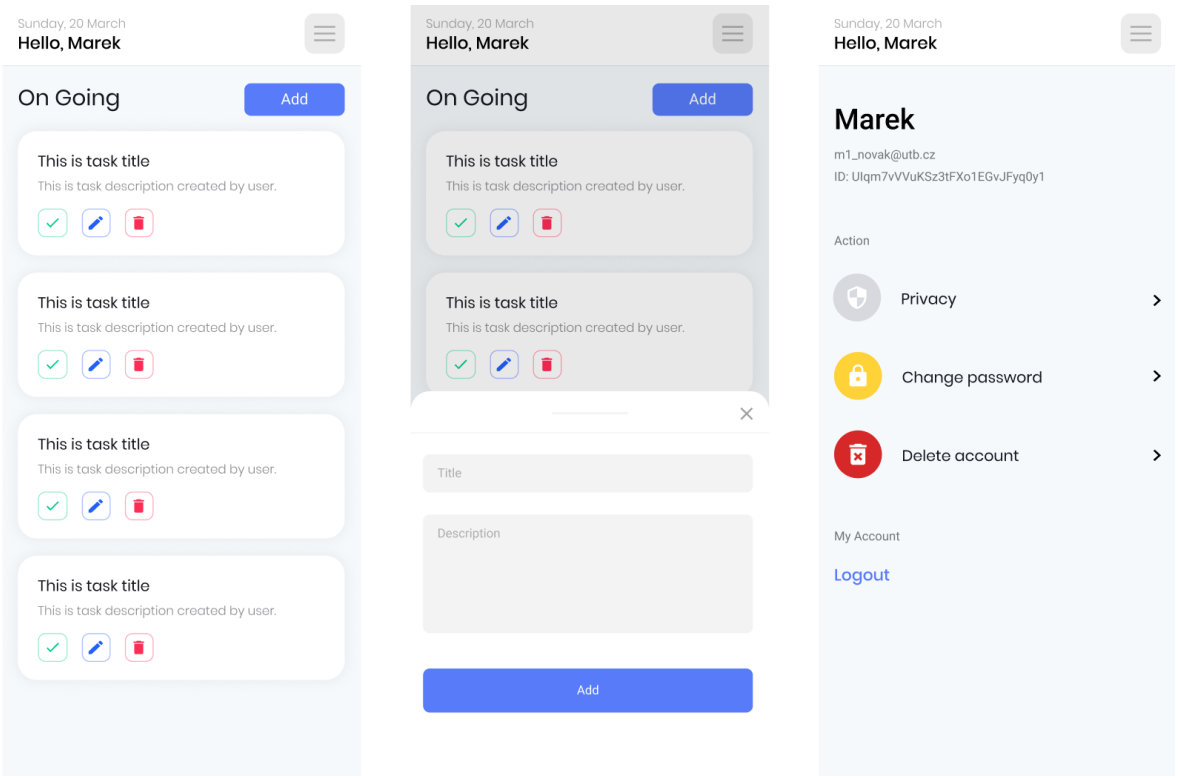
Popularita Reactu stále stoupá a pořád si drží první stupeň nejoblíbenějšího JavaScriptového frameworku. React dokáže při správném použití ušetřit velké množství času, úsilí a nákladů při vývoji uživatelského rozhraní, které bude rozděleno do několika menších izolovaných celků. React tak zajistí rychlé načítání stránek a aplikací, díky *virtuálnímu DOMu*. Na *React* a *React Native* sází několik známých společností. Mezi ně se samozřejmě řadí *Facebook* a *Instagram*, kde původně tato technologie vznikla, ale časem se přidali i další, jako *Coinbase*, *Tesla*, *Pinterest*, *Oculus* nebo *Discord*, kteří využívají *React Native* pro své mobilní aplikace na platformy Android a iOS. React je daleko používanější, jelikož s ním můžeme vytvářet jak jednoduché webové stránky, tak velké komplexní aplikace. Známé společnosti využívající React pro své webové aplikace jsou například *Netflix* a *Airbnb*. Netflix vytvořil řadu přednášek od svých inženýrů, kteří popisují jak a proč využívají React.

II. PRAKTICKÁ ČÁST

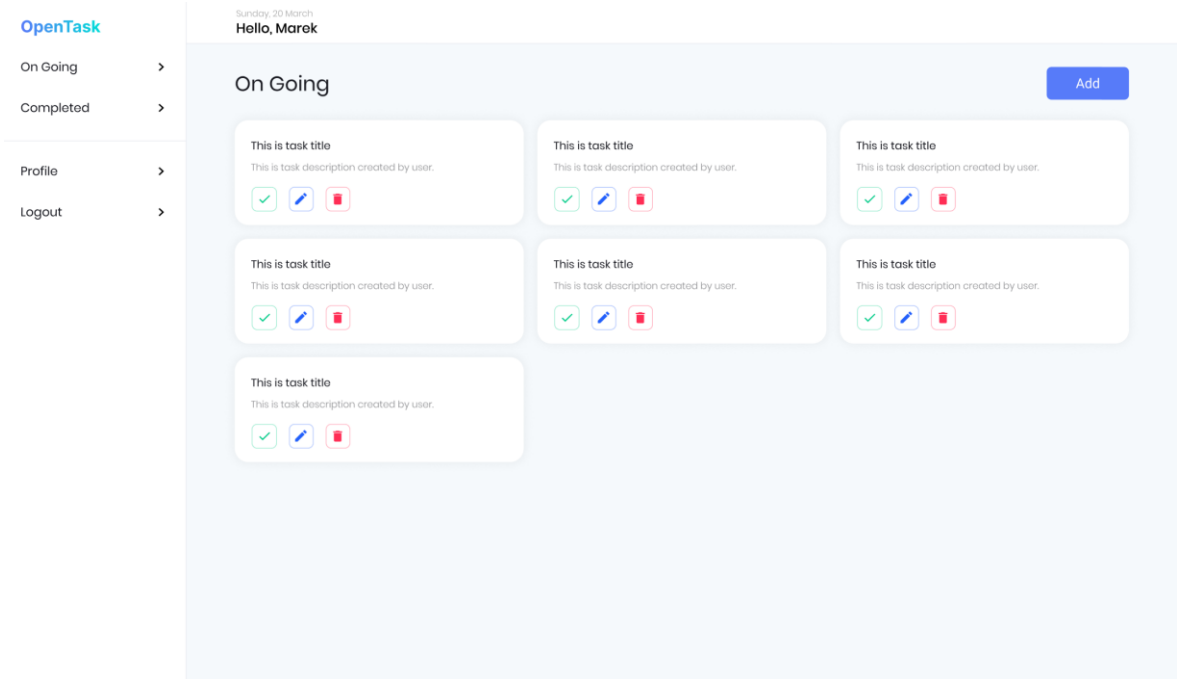
6 NÁVRH GRAFICKÉHO ROZHRANÍ

Před samotnou implementací mobilní a webové aplikace bylo důležité vytvořit grafické návrhy uživatelského rozhraní, ze kterých následně vycházejí tyto aplikace při implementaci. Jelikož byly návrhy vytvořeny již před implementací, stačilo se při vývoji držet těchto návrhů uživatelského rozhraní, čímž se při vývoji ušetřil čas.

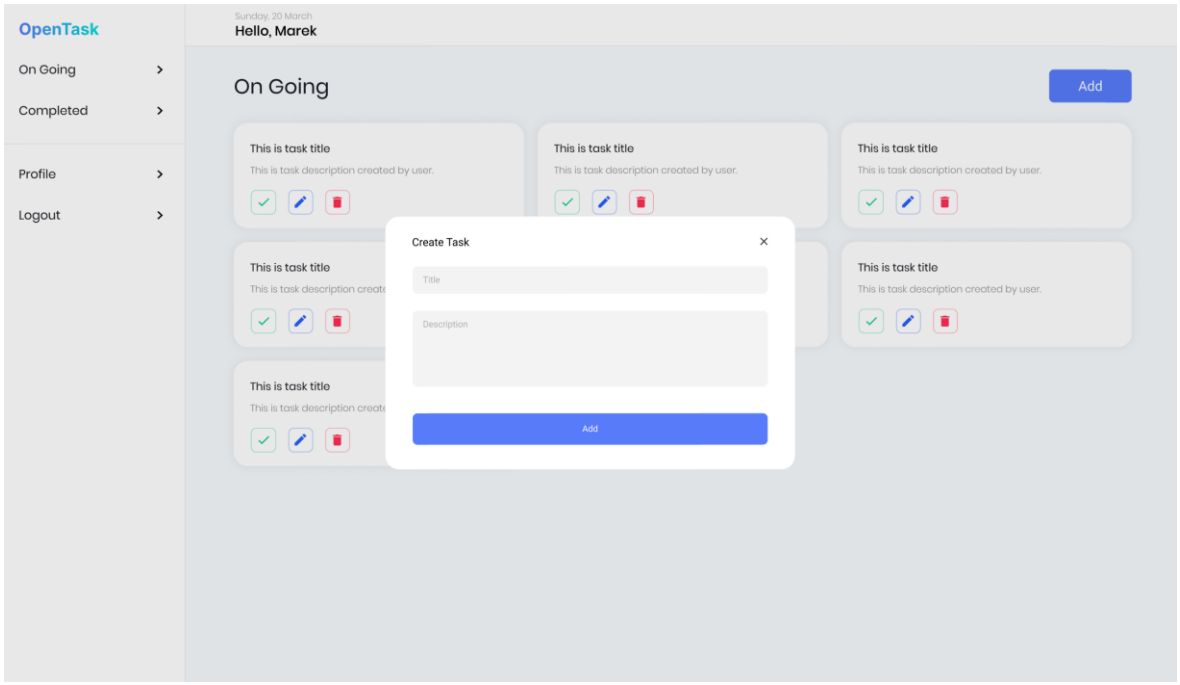
Toto rozhraní bylo vytvořeno díky programu *Figma*, který umožnil vytvořit návrhy jak webové, tak mobilní aplikace. Hlavním cílem návrhu uživatelského rozhraní, bylo vytvořit intuitivní vzhled aplikací pro uživatele, a zjednodušit tak jejich používání.



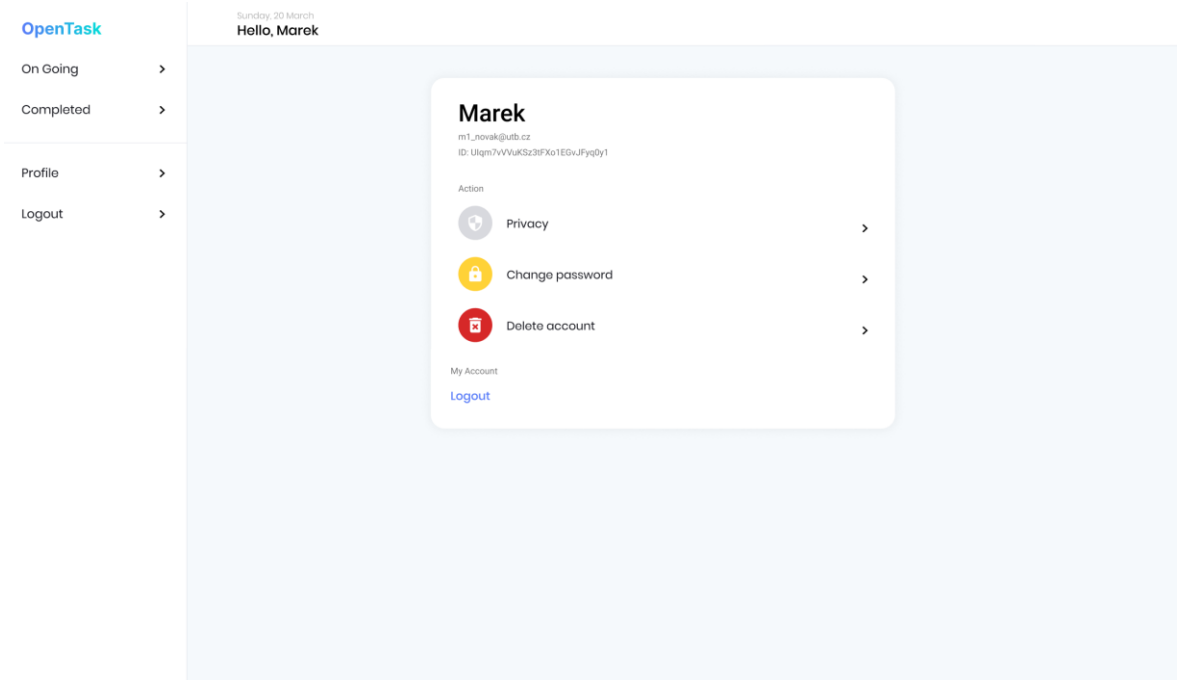
Obrázek 1 Návrh uživatelského rozhraní klíčových obrazovek pro mobilní aplikaci (vlastní zpracování)



Obrázek 2 Návrh uživatelského rozhraní hlavní stránky webové aplikace (vlastní zpracování)



Obrázek 3 Návrh uživatelského rozhraní pro přidání poznámek ve webové aplikaci (vlastní zpracování)



Obrázek 4 Návrh uživatelského rozhraní stránky profilu ve webové aplikaci (vlastní zpracování)

7 IMPLEMENTACE APLIKACÍ

Cílem praktické části bylo implementovat aplikace pro zadávání poznámek, a to právě díky technologii *React* a *React Native*. Samotné implementaci předcházel grafický návrh těchto aplikací pro webovou a mobilní platformu. Aplikace byly porovnávány a testování při vývoji a následně na reálných zařízeních, kde se ověřilo, že aplikace funguje správně.

7.1 Firebase – serverová část aplikace

Jelikož jsou výsledkem praktické části dvě aplikace, bylo dobré umožnit uživatelům vidět stejná data jak na webové, tak mobilní platformě. Z tohoto důvodu byla nutná implementace serverové části, které vyřešila nejen ukládání samotných poznámek do databáze, ale také autentizaci uživatelů a dalších dílčích funkcí.

Jako serverové řešení bylo využito služeb Firebase, které poskytuje cloudové služby pro jednoduchou a intuitivní implementaci. Firebase se do projekt instaluje stejně jako jiné extérní balíčky, a to díky příkazu *npm install firebase.*

7.1.1 Implementace Firebase do projektu

Jak bylo zmíněno, tak implementace celé serverové části je díky *Firebase* jednoduchá, kde vše začíná vytvořením projektu ve webovém rozhraní *Firebase Console*. Zde dostaneme vygenerovaný konfigurační soubor, který nakopírujeme do našeho projektu. Tento konfigurační soubor nám umožní komunikaci mezi aplikací a serverem. Konfigurační soubor obsahuje unikátní identifikátory, které jsou specifické pro každý vytvořený projekt.

```
1   import { initializeApp } from "firebase/app";
2   import { getDatabase } from "firebase/database";
3
4   const firebaseConfig = {
5     apiKey: "API_KEY",
6     authDomain: "PROJECT_ID.firebaseapp.com",
7     databaseURL: "https://DATABASE_NAME.firebaseio.com",
8     projectId: "PROJECT_ID",
9     storageBucket: "PROJECT_ID.appspot.com",
.0    messagingSenderId: "SENDER_ID",
.1    appId: "APP_ID",
.2  };
.3
.4  const app = initializeApp(firebaseConfig);
.5  export const db = getDatabase(app);
```

Obrázek 5 Firebase konfigurace (vlastní zpracování)

7.1.2 Autentizace

Aby uživatelé viděli pouze své vytvořené poznámky a nemohli si tak vzájemně narušovat svůj obsah, je nutné využít autentizace a identifikovat každého uživatele. *Firebase* pro autentizaci nabízí řadu moderních řešení. Primární přihlášení probíhá přes email uživatele, který musí být pro každého uživatele unikátní a také díky heslu k jeho účtu. Tento typ přihlášení je využit v praktické části a uživatel se tak může přihlásit ke svému účtu napříč mobilní či webovou aplikací. *Firebase* podporuje také řadu moderních řešení pro autentizaci, jako je přihlášení přes *Gmail, Facebook, GitHub, Twitter, Apple, Microsoft* nebo také přes telefonní číslo.

7.1.2.1 Registrace nového uživatele

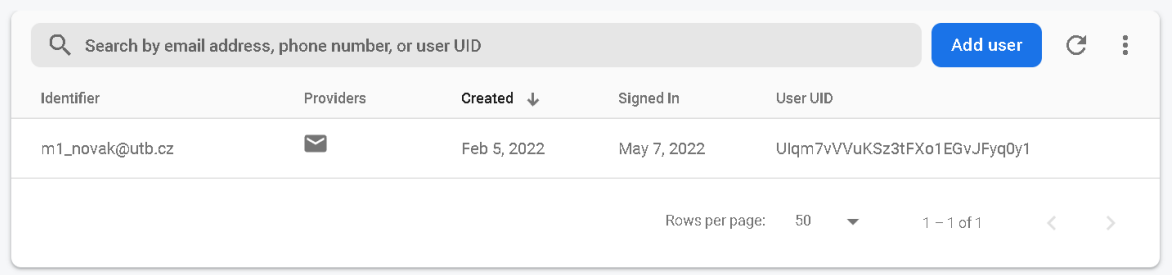
Pro vytvoření nového uživatele stačí implementace funkce *signInWithEmailAndPassword*, kterou lze importovat přímo z nainstalovaného balíčku *Firebase*. Tato funkce od uživatele přebírá dva parametry, jedná se o email a heslo. Vstupy pro registraci uživatel zadává do standardního formuláře, který je v aplikaci vytvořený. Funkce ověří, zda již není nějaký uživatel s tímto emailem zaregistrovaný. V případě, že je email unikátní a v databázi se nenachází, uživatel je úspěšně zaregistrován a přesměrován do aplikace. Funkce pro registraci vypadá následovně.

```
const handleRegister = () => {
  if (registerInformation.email !== registerInformation.confirmEmail) {
    alert("Email is not the same");
    return;
  } else if (
    registerInformation.password !== registerInformation.confirmPassword
  ) {
    alert("Password is not the same");
    return;
  }

  createUserWithEmailAndPassword(
    auth,
    registerInformation.email,
    registerInformation.password
  )
    .then(async () => {
      await updateProfile(auth.currentUser, { displayName: username });
      navigate("/tasks");
      window.location.reload();
    })
    .catch((err) => alert(err.message));
};
```

Obrázek 6 Funkce pro registraci nového uživatele (vlastní zpracování)

Ve webové administraci *Firebase* můžeme najít přehled všech uživatelů. V tabulce se nacházejí informace o uživateli, jako je *email, datum vytvoření účtu, datum posledního přihlášení* a *UID*, což je unikátní kombinace znaků a čísel pro každého uživatele. Díky unikátnosti *UID* můžeme tento parametr rovněž využít pro identifikaci uživatele v rozhraní databáze, stejně jako emailovou adresu.



Obrázek 7 Přehled zaregistrovaných uživatelů v administraci

7.1.2.2 Přihlášení uživatele

Přihlášení probíhá podobně jako registrace, a to díky existující funkci z *Firebase*. Využijeme tak funkci *signInWithEmailAndPassword*, která bude rovněž přebírat email a heslo ze vstupu, následně se na serveru provede validace těchto parametrů, jestli se uživatel s touto emailovou adresou již v databázi nenachází. V případě, že jsou data validní, bude uživatel přesměrován dále do aplikace. Celá funkce pro přihlášení vypadá následovně.

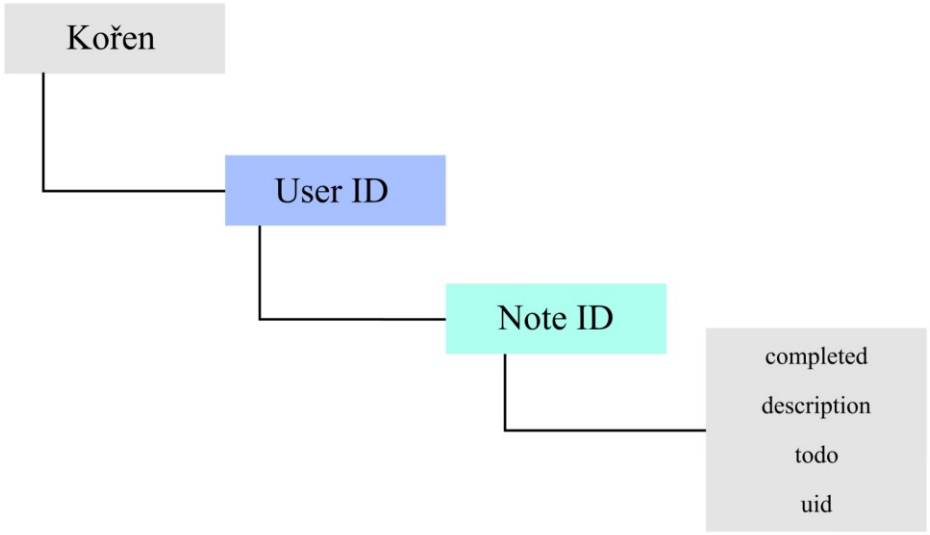
```
1  const handleSignIn = () => {
2    signInWithEmailAndPassword(auth, email, password)
3      .then(() => {
4        navigate("/tasks");
5      })
6      .catch((err) => alert(err.message));
7  };
```

Obrázek 8 Funkce pro přihlášení do aplikace (vlastní zpracování)

7.1.3 Realtime databáze

Firebase nabízí dva druhy databáze, v tomto projektu je však využita pouze *realtime databáze*, která má stromovou strukturu jako soubory *JSON*. Tento typ databáze je vhodný pro menší projekty, které nepracují s velkým množstvím dat. Jelikož se v projektu do této databáze ukládají pouze poznámky, je tento typ vhodný. V případě většího projektu nebo případné škálovatelnosti aplikace je vhodné využít jiný typ databáze, které Firebase nabízí.

Realtime databáze umožňuje synchronizaci dat napříč aplikacemi v reálném čase, a to bez nutnosti aktualizace stránky nebo aplikace. Můžeme tak zadat poznámky v mobilní aplikaci a změna se ihned projeví i na dalších zařízeních.



Obrázek 9 Stromová struktura uložených dat v databázi (vlastní zpracování)

Každý uživatel má pod svým ID uložené všechny poznámky, které mají také své unikátní ID. Díky ID pak můžeme jednotlivé poznámky mazat či aktualizovat. Každá z poznámek je objektem a obsahuje následující hodnoty:

- **Completed**: stav poznámky, jestli je hotová nebo ne. Obsahuje hodnotu *True* nebo *False*.
- **Description**: popis poznámky.
- **Todo**: název samotné poznámky.
- **UID**: unikátní ID poznámky.

Obrázek 10 Ukázka stromové struktury přímo z přehledu databáze (vlastní zpracování)

7.1.4 Vytváření, mazání a aktualizace poznámek

Uživatelé mají v aplikaci možnost poznámky *vytvořit*, *mazat* či případně provádět jejich *úpravy*. Jak již bylo zmíněno, tak poznámky jsou uloženy jako *stromová struktura* v databázi. K těmto poznámkám přistupujeme za pomocí jejich unikátního *identifikačního klíče*, který je nutné využít i při manipulaci s poznámkami v implementovaných funkcích.

Veškeré operace s poznámkami jsou řešené díky funkcím, které nabízí *Firebase* pro manipulaci s daty mezi klientem a databází, stejně jako tomu je při registraci a přihlášení uživatele.

7.1.4.1 Vytvoření poznámek

Pro zapisování do databáze je využita funkce *set*, kde je nastaven parametr v podobě cesty, kde se má vytvořit nový uzel, který reprezentuje poznámku. Abychom vytvořili poznámku pro konkrétního uživatele, musíme do cesty v parametru zadat *ID* právě přihlášeného uživatele, pod kterým se poznámka vytvoří.

Abychom mohli s poznámkou do budoucna pracovat a manipulovat s ní, musíme ji ve funkci nastavit unikátní identifikační klíč. Tento klíč je v aplikaci generován díky

externímu balíčku, který se instaluje za pomocí *Node Package Manageru (npm).* Tento vygenerovaný klíč se následně využije jako součást cesty pro vytvoření poznámky.

```
const writeToDatabase = () => {
  const uidd = uid();
  set(ref(db, `/${auth.currentUser.uid}/${uidd}`), {
    todo: todo,
    description: description,
    uidd: uidd,
    completed: false,
  });

  setTodo("");
  setDescription("");
};
```

Obrázek 11 Funkce pro vytvoření nové poznámky v databázi (vlastní zpracování)

7.1.4.2 Mazání poznámek

Funkce pro mazání poznámek je poměrně jednoduchá na implementaci, jelikož stačí předat do funkce *remove* dva parametry, kde jedním z nich je inicializace databáze, kde se má smazání provést a jako druhý parametr je cesta k poznámce, tedy cesta k uzlu, který se má smazat. Uzel je v cestě opět reprezentován v podobě identifikačního klíče.

```
const handleDelete = (uid) => {
  remove(ref(db, `/${auth.currentUser.uid}/${uid}`));
};
```

Obrázek 12 Funkce pro smazání poznámky v databázi (vlastní zpracování)

7.1.4.3 Aktualizace poznámek

Ze strany uživatele je možné editovat některé parametry poznámek. Jedná se o název poznámky (parametr *todo*) a popis poznámky (parametr *description*). Dále může uživatel editovat stav poznámky, jestli poznámku považuje za uzavřenou či nikoliv. Tento parametr má dva stavy *True* a *False.* Toto základní rozdělení umožňuje v aplikaci třídit poznámky na dvě kategorie. Pro každou z těchto kategorií je vytvořena vlastní stránka jak na webu, tak v mobilní aplikaci.

```
const handleEditConfirm = () => {
  update(ref(db, `/${auth.currentUser.uid}/${tempUidd}`), {
    todo: todo,
    description: description,
    tempUidd: tempUidd,
  });

  setTodo("");
  setDescription("");
  setIsEdit(false);
};
```

Obrázek 13 Funkce pro aktualizaci poznámky v databázi (vlastní zpracování)

```
const completeTodo = (uid, isCompleted) => {
  update(ref(db, `/${auth.currentUser.uid}/${uid}`), {
    uidd: uid,
    completed: !isCompleted,
  });
};
```

Obrázek 14 Funkce pro aktualizaci stavu poznámky v databázi (vlastní zpracování)

7.2 Webová aplikace

Webová aplikace vychází z původního grafického návrhu, a tak jsem se při implementaci snažil dodržel její vzhled. Jeden z mých osobních cílů byl co nejvíce sjednotit aplikace napříč platformami, aby měl uživatel stále pocit, že se jedná o tu samou aplikaci, ať si ji zobrazí ve webovém prohlížeči nebo přímo na mobilním zařízení.

Aplikace je závislá na *NodeJS*, díky kterému bylo možné nainstalovat externí balíčky nebo spustit a testovat aplikaci na lokálním serveru. Při výběru technologií pro tento projekt bylo klíčové vybrat případné vhodné doplňky k Reactu, kde jsem zvažoval různé typy externích knihoven pro stylování, ale jelikož aplikace není tak komplexní a chtěl jsem co nejvíce designem přiblížit ke grafickému návrhu, zvolil jsem pro stylování webové aplikace čisté CSS. V projektech založených na Reactu je také důležité zvážit rozdělení komponentů, stylů a logiky aplikace. Tohle rozdělení a celá struktura projektu je pro každý projekt individuální. Já jsem pro webovou aplikaci zvolil vlastní přístup, který vychází z *Atomic Design Pattern.*

7.2.1 Struktura komponentů

Jak bylo zmíněno výše, tak pro webovou aplikaci jsem zvolil strukturu vycházející z *Atomic Design Pattern*, který jsem mírně upravil pro požadavky této aplikace. Tato struktura se skládá z pěti částí, jejichž kombinací můžeme zkompletovat aplikaci. Těchto pět bloků dělíme na atomy, molekuly, organismy, šablony a stránky.

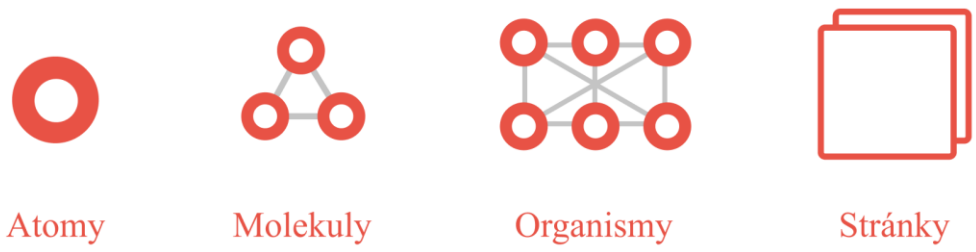
V projektu jsem se rozhodl tohle rozdělení dodržet, až na výjimku šablon, jelikož samotný návrh aplikace má pouze opakující se menší komponenty (například tlačítka a vstupy) než nějaké větší celky. Chtěl jsem se tak vyvarovat zbytečnému sofistikovanému rozdělení pro aplikaci těchto rozměrů.

Atomy jsou základní stavební prvky, jako jsou tlačítka, vstupy nebo popisky.

Molekuly jsou vytvořeny za pomocí seskupení více atomů, za účelem vytvoření funkčnosti.

Organismy jsou tvořeni kombinací molekul, které tvoří část rozhraní, například navigační panel.

Stránky tvoří ekosystém, který je složený z několika organismů a následně se jako celek prezentují uživateli.



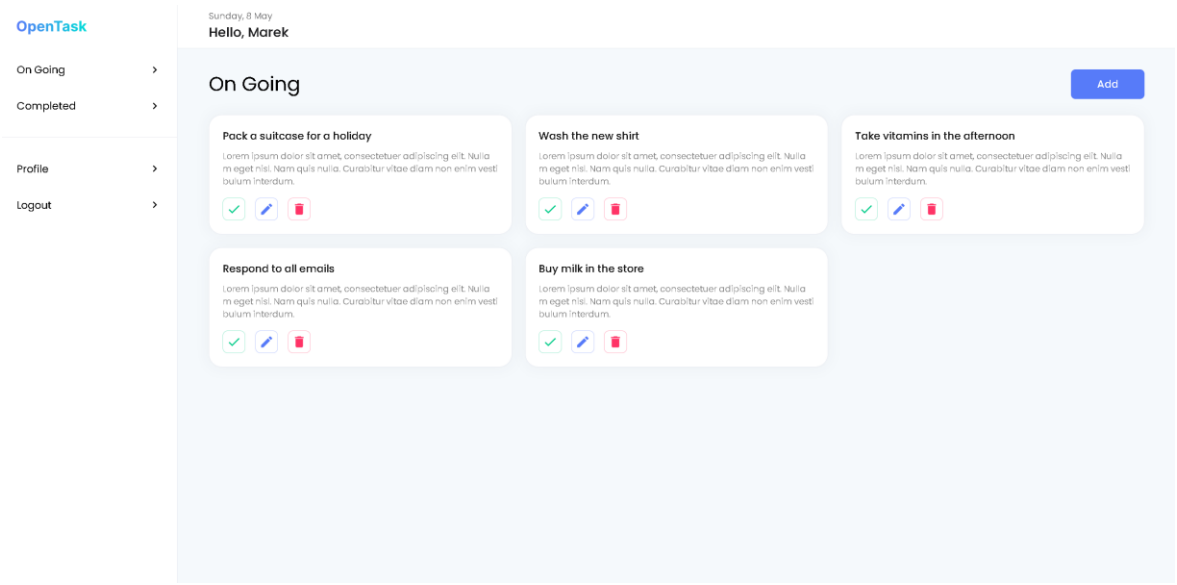
Obrázek 15 Vizualizace struktury komponentů (vlastní zpracování)

7.2.2 Uživatelské rozhraní webové aplikace

Každá stránka aplikace obsahuje tři základní sekce. Na levé straně se nachází navigační menu pro přechod mezi jednotlivými stránkami aplikace. V horní části se pak nachází sekce, která obsahuje informace o aktuálním dni v týdnu, měsíci, dni v měsíci a uživatelské jméno právě přihlášeného uživatele. Poslední a zároveň nejdůležitější část je tvořena obsahem každé stránky, na kterou se uživatel naviguje.

7.2.2.1 Přehled poznámek

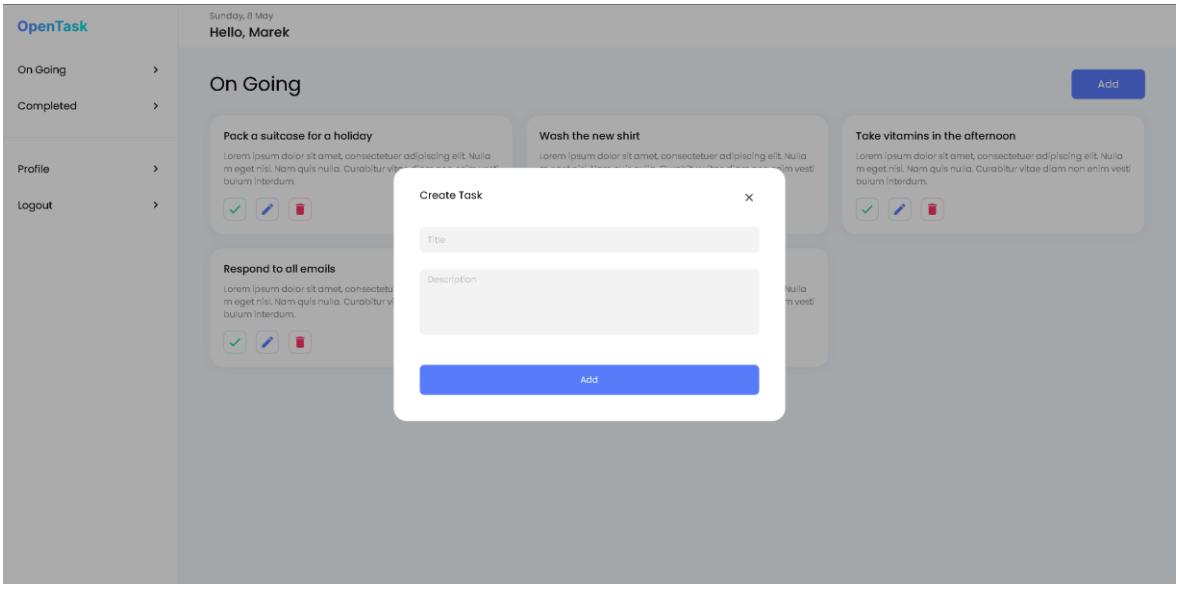
Po přihlášení do aplikace je uživatel přesměrován přímo na primární stránku, kterou je přehled vytvořených poznámek na stránce *On Going*, tedy poznámek, které uživatel kategorizuje jako nevyřízené. Na této stránce může provádět manipulace s poznámkami, jako je editace, mazání či přidání nové poznámky.



Obrázek 16 Ukázka hlavní stránky webové aplikace (vlastní zpracování)

7.2.2.2 Přidání nové poznámky

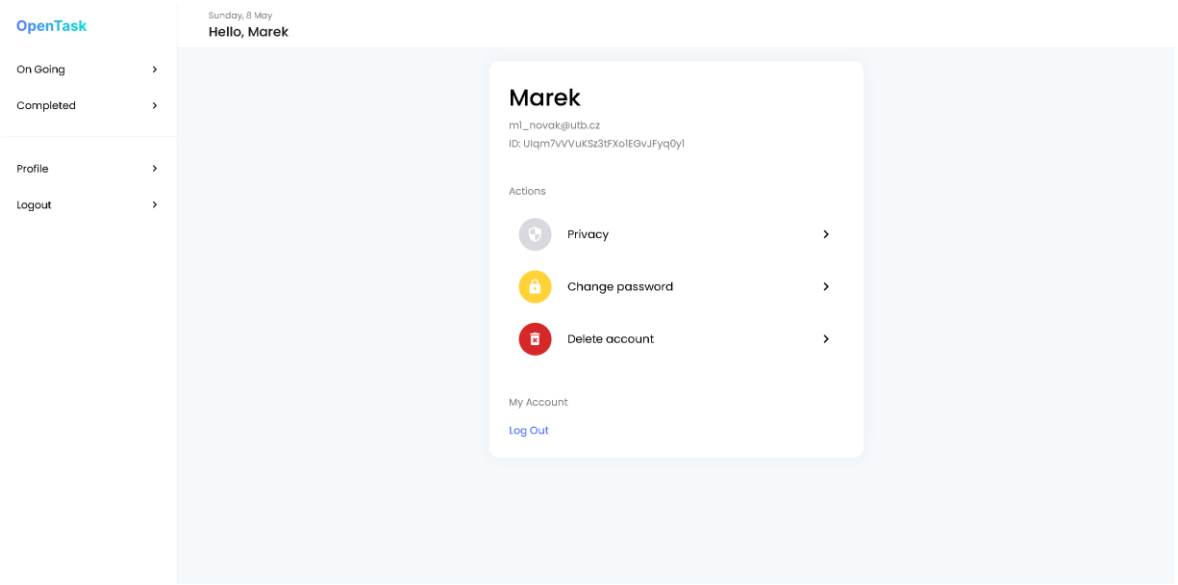
Po kliknutí na tlačítko *Add* se uživateli otevře formulář, který slouží k vytvoření nové poznámky. Obsahuje dvě základní pole pro vyplnění názvu poznámky a jeho podrobnějšího popisu. Stejný formulář se zobrazí i v případě editace poznámky, jen s výjimkou, že textová pole již budou předvyplněna aktuálními parametry editované poznámky. Tyto parametry lze přepsat či zcela odstranit. Aktualizace se propíše do databáze ihned po potvrzení úpravy díky tlačítku *Confirm*.



Obrázek 17 Formulář pro vytvoření nové poznámky (vlastní zpracování)

7.2.2.3 Uživatelský profil

Jelikož aplikace umožňuje přihlášení uživatele, bylo nutné vytvořit i základní funkce, jako je změna hesla, zapomenuté heslo nebo smazání účtu. Tyto funkce můžeme najít na stránce uživatelského profilu. Stránka obsahuje základní informace o uživateli, jako je uživatelské jméno, email a jeho ID, díky kterému můžeme každého uživatele identifikovat.

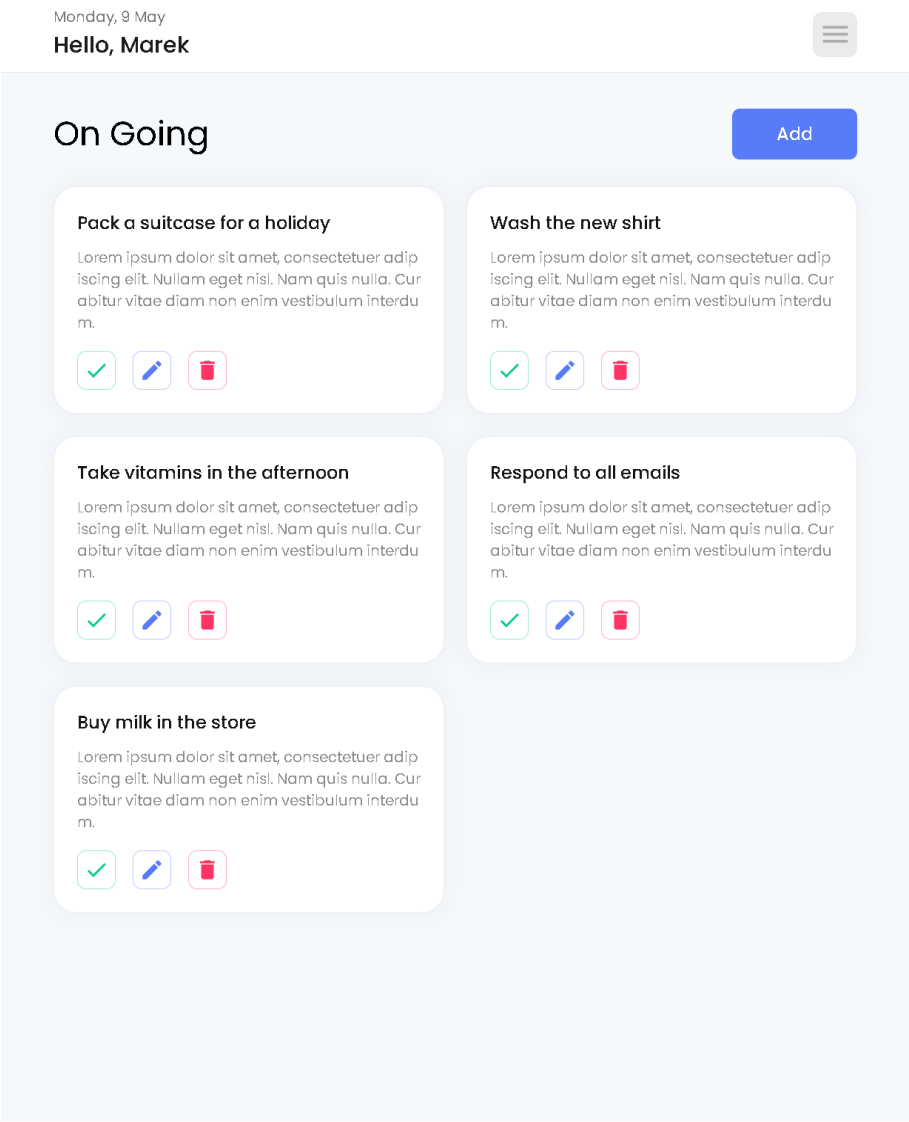


Obrázek 18 Uživatelský profil (vlastní zpracování)

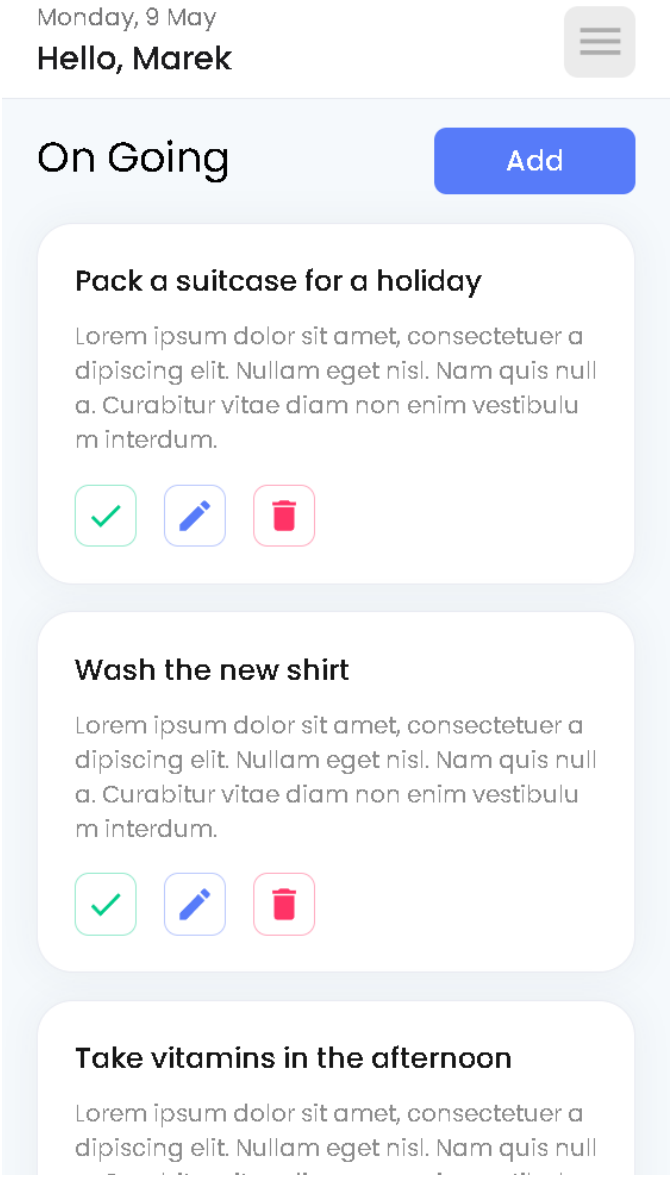
7.2.3 Responzivita

Aby mohla webová aplikace fungovat na jakémkoliv zařízení, bylo nutné vyřešit responzivní design uživatelského rozhraní. Přizpůsobení rozhraní tak bylo docíleno díky CSS a *media queries*, ve kterých je definované, jak se mají jednotlivé části měnit či přizpůsobovat na základě šířky obrazovky zařízení.

Rozvržení stránky je pro tablety a mobily téměř totožné. Boční navigační menu je skryté v levé části obrazovky a pro jeho otevření bylo přidané tlačítko, které menu otevře. Poznámky jsou vytvořené jako *grid*, což velice usnadnilo přizpůsobení jak pro mobilní zařízení, tak pro tablety. *Grid* je pro vykreslování poznámek nastaven tak, aby se na počítačích zobrazoval ve třech sloupcích, na tabletech ve dvou sloupcích a na mobilním zařízení jsou všechny poznámky sjednoceny do jednoho sloupce.



Obrázek 19 Ukázka responzivity pro tablety (vlastní zpracování)



Obrázek 20 Ukázka responzivity pro mobilní zařízení (vlastní zpracování)

7.3 Mobilní aplikace

Aplikace pro mobilní zařízení byla primárně navržena a testována na zařízení s operačním systémem Android. Serverová část aplikace funguje na stejném principu jako navržená webová aplikace, takže logika je stejná v obou aplikacích.

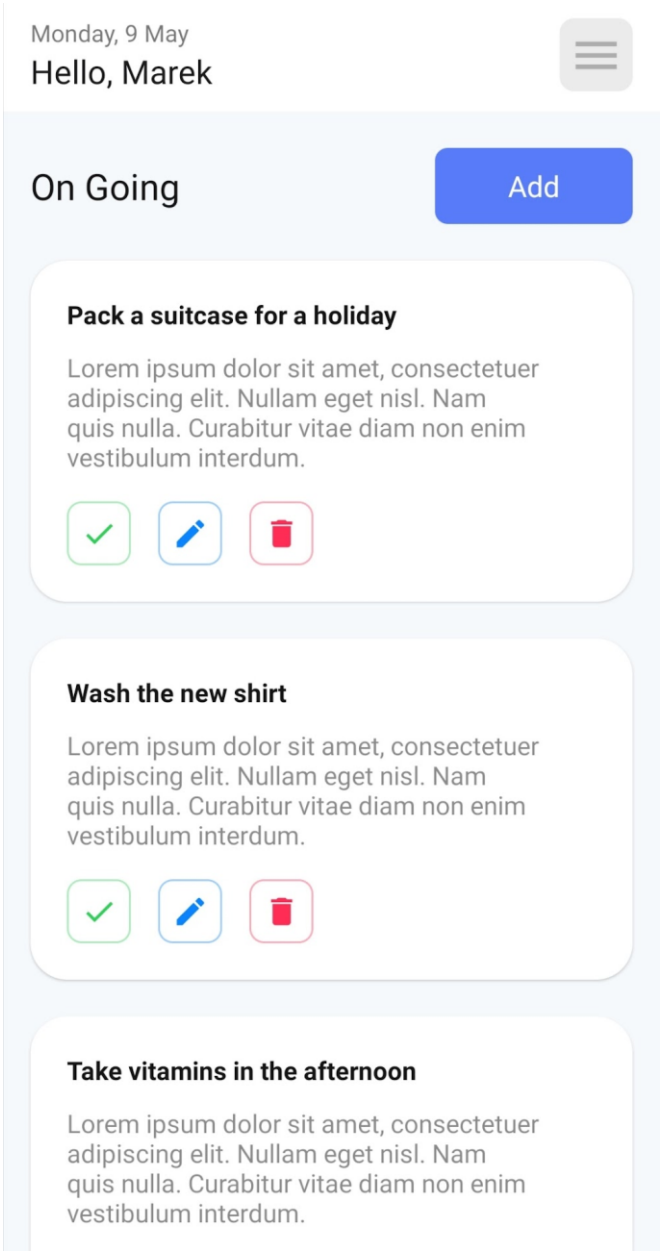
7.3.1 Struktura komponentů

Při implementaci mobilní aplikace jsem zvolil jiný přístup k rozdělení struktury komponentů. Struktura je tak rozdělena na dvě základní části, kde první část je tvořena adresářem

s komponenty, které se v aplikace využívají na více místech a druhá část je tvořena obrazovkami aplikace, kde se implementují jednotlivé komponenty a ty pak tvoří rozhraní aplikace. Tento přístup je vhodný při vývoji menších projektů.

7.3.2 Uživatelské rozhraní mobilní aplikace

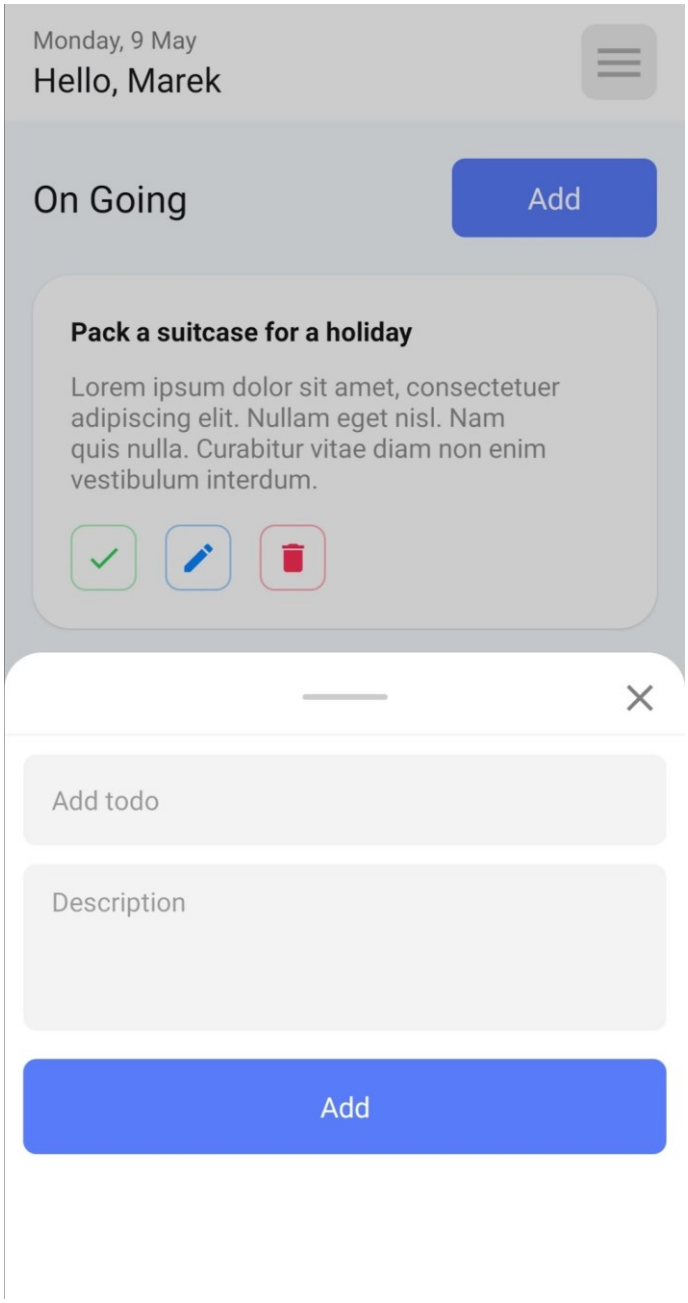
Vývoj uživatelského rozhraní bylo jednodušší u mobilní aplikace, jelikož jsem při vývoji nemusel brát v potaz ostatní zařízení, jako je tablet a zobrazení na počítači. Cílem tak bylo vytvořit stejnou aplikaci po vzhledové stránce, jako je zobrazení webové aplikace na mobilním zařízení.



Obrázek 21 Ukázka hlavní stránky mobilní aplikace (vlastní zpracování)

7.3.2.1 Přidání nové poznámky

Pro zobrazení formuláře na vytvoření nových poznámek jsem využil populární řešení zvané *bottom sheet*, tedy komponentu, která se po stisknutí tlačítka vysune ze spodní části obrazovky a překrývá tak původní obsah stránky. Tento typ komponenty jsme využil také v dalších částech aplikace, jako je například funkce pro smazání profilu uživatele.



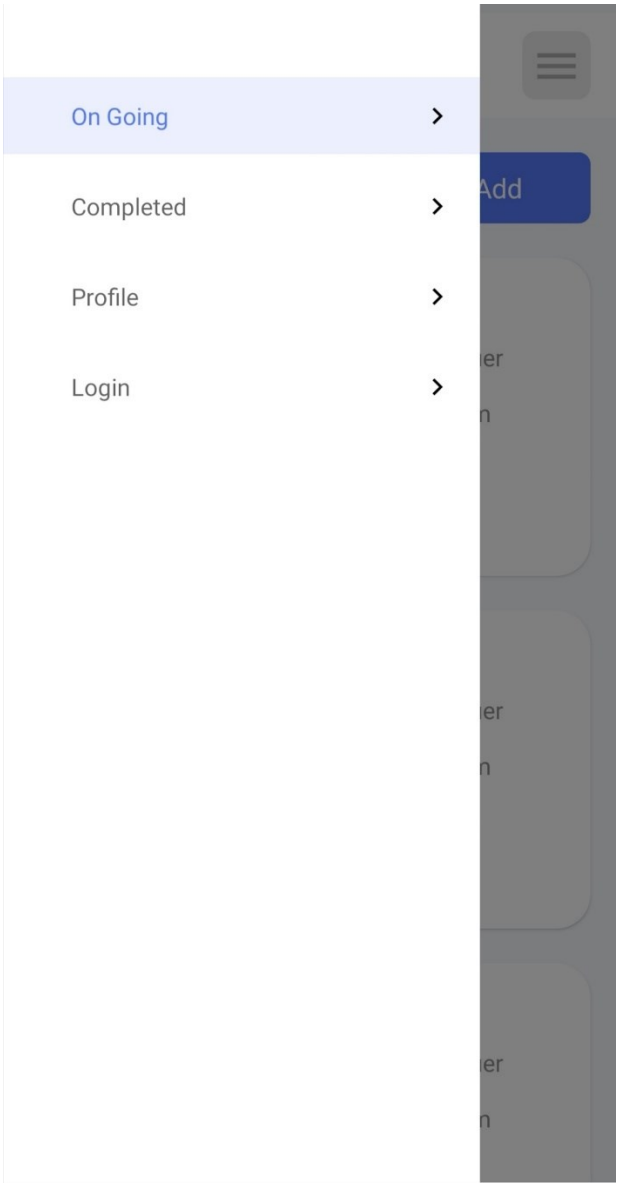
Obrázek 22 Formulář pro vytvoření nové poznámky v mobilní aplikaci (vlastní zpracování)

7.3.2.2 Navigační menu

Vytvoření navigačního menu se v mobilní aplikaci na rozdíl od webové aplikace liší svou implementací. Před vytvořením navigace bylo nutné do projektu nainstalovat balíček *@react-navigation/drawer*, který umožnil vygenerovat boční navigační menu. I přesto, že se menu generuje automaticky a má již předdefinovaný základní vzhled, můžeme ho díky vlastnímu stylování upravit.

```
26    return (
27      <NavigationContainer theme={MyTheme}>
28        <Drawer.Navigator initialRouteName="Home">
29          <Drawer.Screen
30            name="Home"
31            component={HomeScreen}
32            options={{
33              headerShown: false,
34              drawerActiveBackgroundColor: "#ebeffe",
35              drawerActiveTintColor: "#577bf9",
36              drawerItemStyle: {
37                borderRadius: 0,
38                width: "100%",
39                marginLeft: 0,
40              },
41            }}
42          />
43          <Drawer.Screen
44            name="Profile"
45            component={ProfileTaskScreen}
46            options={{
47              headerShown: false,
48              drawerActiveBackgroundColor: "#ebeffe",
49              drawerActiveTintColor: "#577bf9",
50              drawerItemStyle: {
51                borderRadius: 0,
52                width: "100%",
53                marginLeft: 0,
54              },
55            }}
56          />
57        </Drawer.Navigator>
58      </NavigationContainer>
59    );
60  }
```

Obrázek 23 Ukázka implementace navigačního menu (vlastní zpracování)



Obrázek 24 Navigační menu mobilní aplikace (vlastní zpracování)

7.3.2.3 Instalační balíček pro Android

Po dokončení této aplikace jsem vytvořil instalační balíček pro platformu *Android*, aby bylo možné aplikaci nainstalovat a otestovat na reálném zařízení. Díky tomuto vznikla plnohodnotná mobilní aplikace, kterou je možné umístit volně ke stažení na *Google play*, nicméně publikace aplikace není součástí této bakalářské práce, a proto jsem tento krok vynechal. Před vytvořením instalačního balíčku bylo nutné splnit některé náležitosti jako je např vytvoření *ikony* pro aplikaci či *splash screenu*. Vytvoření instalačního balíčku probíhalo přes webové rozhraní *Expo* díky příkazu *expo build:android*.

7.4 Capacitor – mobilní aplikace

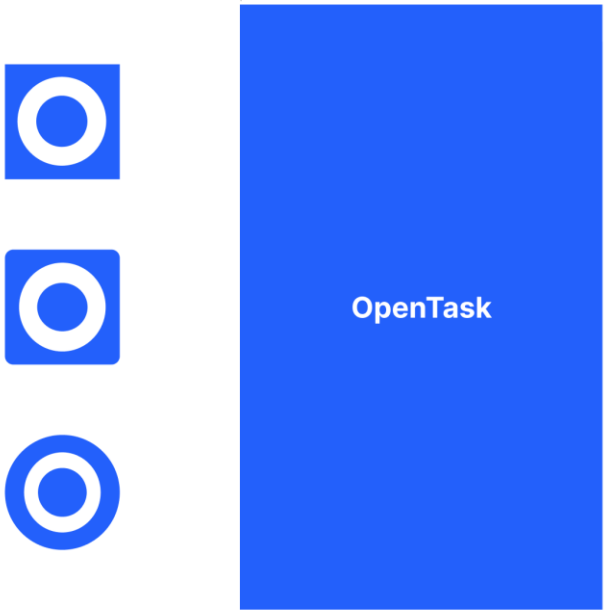
Dalším cílem praktické části bylo přizpůsobit původní webovou aplikaci tak, aby měla charakter mobilní aplikace a následně z ní vytvořit aplikaci pro platformu *Android*.

Pro vytvoření této aplikace bylo nutné ověřit responzivní chování původní webové aplikace pro mobilní zařízení a následně díky *Capacitoru* vytvořit instalační balíček. Jelikož se aplikace neinstaluje jako typické *PWA* přes webový prohlížeč, ale instaluje se přímo do mobilního zařízení, nebylo tak nutné vytvářet *servis worker*, jelikož aplikace v offline módu nestahuje data z webového serveru a jiné změny ve struktuře webové aplikace nebyly nutné.

7.4.1 Přidání Android platformy do projektu

Aby byla aplikace instalovatelná, nejdříve jsem musel do projektu přidat balíček *@capacitor/android*, který nainstaloval *Capacitor*. Následně bylo nutné přidat platformu díky příkazu *npx cap add android*, což umožnilo specifikovat pro jakou platformu bude aplikace určená. Díky tomuto bylo možné spustit aplikaci v Android studiu, kde se dala aplikace testovat na emulátoru a vytvořit z ní instalační balíček.

Dalším krokem bylo vytvořit *splash screen* a *ikonu* aplikace, které se budou uživateli zobrazovat po nainstalování do reálného mobilního zařízení. Pro tento účel jsem opět využil aplikaci Figma, kde jsem vytvořil *splash screen* a *ikony* pro různé typy rozlišení, aby se tato část aplikace zobrazovala správně na každém zařízení.



Obrázek 25 Návrh ikon a splash screenu (vlastní zpracování)

7.4.2 Vytvoření instalačního balíčku

Po úspěšném přidání platformy do projektu jsem díky *Android studiu* vytvořil instalační balíček, který umožnil nainstalovat mobilní aplikaci do reálného zařízení. Těchto instalačních balíčků jsem v průběhu vytvořil hned několik, a to zejména proto abych v průběhu implementace otestoval, zda aplikace funguje korektně po instalaci na reálných zařízeních a opravil případné chyby, a to vše před vytvořením konečné verze aplikace. Konečnou verzi aplikace jsem dále po nainstalování testoval v roli uživatele.

8 TESTOVÁNÍ VYTVOŘENÝCH APLIKACÍ

Každá z vytvořených aplikací se testovala zcela individuálně. Aplikace byly testovány jak během jejich implementace, tak po jejich dokončení, kde veškeré testy probíhaly na reálných zařízeních. Cílem tohoto testování bylo zjistit případné nedostatky aplikací či jejich kompatibilitu na testovaném zařízení.

8.1 Testování během vývoje

Mobilní aplikace byly testovány během vývoje díky aplikaci *Expo*, která umožnila testovat právě vyvíjenou aplikaci na reálném zařízení bez nutnosti její instalace. Mobilní aplikaci šlo rovněž testovat ve webovém prohlížeči, nicméně tento typ testování nezachytil některé chyby v implementaci aplikace, jako je například špatná syntaxe při stylování komponentů v *React Native*. Technologie, která také podpořila testování je *Android studio*, díky jehož *emulátoru* bylo možné otestovat aplikaci na téměř libovolném typu zařízení a verzi Androidu, které *Android studio* podporuje. Vybral jsem tak emulátor mobilního zařízení *Pixel 2* a *Android 10.0.0*. Naopak testování webové aplikace bylo při vývoji prováděno primárně ve webovém prohlížeči na počítači a také v prohlížeči na mobilním zařízení.

Během vývoje jsem testoval, zda se veškerá data z aplikací zapisují do databáze korektně, kde jsem veškerý zápis mohl sledovat přes webové rozhraní *Firebase*. Ve webové aplikaci bylo rovněž důležité otestovat její *responzivitu*. Tyto testy probíhaly rovněž ve webovém prohlížeči *Chrome*, a to díky funkci *DevTools*, kde je možné ověřit, jak se aplikace přizpůsobuje pro různé typy rozlišení.

8.2 Uživatelské testování

Testování z role uživatele probíhalo po konečné implementaci všech aplikací, a to tak že jsem se řídil možným scénářem, který reprezentoval chování uživatele v aplikaci. Ověřil jsem tak všechny funkce aplikace jako je vytvoření uživatelského profilu, přihlášení, vytváření poznámek či jiná manipulace s nimi nebo odstranění uživatelského profilu.

Mobilní aplikace byly testovány na reálných zařízeních s operačním systémem *Android*, a to tak že byly do zařízení přímo nainstalovány. Mezi tyto zařízení patří například *Honor 10*, který disponuje verzí *Androidu 10.0.0* nebo *Redmi Note 9 Pro*, ve verzi *Androidu 11.0.0*. Webová aplikace byla testována ve webovém prohlížeči *Chrome*, *Opera* nebo také *Microsoft*

Edge a to přímo na počítači. Dále byla tato verze aplikace testována i ve webovém prohlížeči na mobilním zařízení z pohledu koncového uživatele.

9 ZHODNOCENÍ VÝSLEDNÝCH APLIKACÍ

V této kapitole se zabývám zhodnocením aplikací, které byly vytvořeny v rámci praktické části.

Z pohledu časové náročnosti, bylo podstatně rychlejší vytvoření mobilní aplikace díky technologii *React Native* a to zejména proto, že tato aplikace pokrývá pouze mobilní zařízení a nebylo tak nutné přizpůsobovat uživatelské rozhraní aplikace pro větší rozlišení jako jsou například tablety či počítače. Naopak u webové aplikace bylo nutné přemýšlet nad tím, jak má aplikace vypadat, případně se chovat na různých typech zařízení a všechny části aplikace tak vhodně přizpůsobit. Pro časovou náročnost vývoje aplikací hrálo také velkou roli rozdělení struktury komponentů a stránek, díky čemuž bylo možné usnadnit opravy komplexnějších částí aplikace během vývoje. U webové aplikace jsou komponenty rozděleny do podstatně menších celků, než je tomu u mobilní aplikace, což bylo přínosné zejména v momentě, kdy některá z částí aplikace nefungovala dle očekávání, a tak bylo jednoduché chybu ihned identifikovat a opravit pouze příslušnou komponentu, bez nutnosti zasahovat do funkční části kódu, jelikož jednotlivé části jsou od sebe více izolované.

Zajímavé bylo pracovat s technologií *Capacitor*, kde bylo poměrně jednoduché vytvoření mobilní aplikace z původní webové verze aplikace, a to bez nutnosti zásahu do struktury projektu. Tato metoda vývoje může být vhodná pro firmy, které potřebují v krátkém čase vytvořit jak webovou, tak mobilní aplikaci, či nemají tak velký rozpočet na vývoj.

Pro další porovnání se níže nachází tabulka, která obsahuje data o velikosti instalačních balíčků mobilní aplikace vytvořené díky React Native a Capacitor. Tabulka také obsahuje průměrnou rychlost spuštění každé z aplikací. Uvedená hodnota je průměrem deseti spuštění pro každou aplikaci. Test rychlosti spuštění aplikace probíhal v Android Studiu, kde byly aplikace spuštěny na emulátoru.

Tabulka 1 porovnání mobilních aplikací

Porovnání	React Native	React + Capacitor
Velikost APK balíčku	62 MB	2,15 MB
Průměrná rychlost spuštění	1,6451 s	1,9916 s

Všechny aplikace jsem vytvořil tak, aby bylo možné je dále rozšířit nebo případně upravit. Jelikož *React Native* podporuje nativní funkce, je tak možné rozšířit celou aplikaci například o *push notifikace* pro nastavení upozornění pro každou z poznámek. I přesto, že byla druhá verze mobilní aplikace vytvořená z původní webové aplikace, a to zcela za pomocí webových technologií, i tak je možné využít nativní funkce zařízení díky *Capacitoru*, který přidal do celého projektu platformu Android. Kdyby byla aplikace vytvořena pouze jako *PWA* a instalovala by se do mobilního zařízení přes webový prohlížeč, nebylo by tak možné využít tyto nativní funkce.

Webová aplikace je dostupná na adrese: *https://open-task.vercel.app*

ZÁVĚR

Tato bakalářská práce se zabývala specifiky vývoje aplikací pomocí technologií React a React Native, díky kterým byla v praktické části vytvořena aplikace pro zadávání poznámek.

Teoretická část práce se v úplném úvodu zabývá základní charakteristikou důležitých pojmů pro pochopení zbytku celé práce, jako jsou například definice webové a mobilní platformy, operačních systémů mobilních zařízení nebo konceptů pro vývoj multiplatformních aplikací. Následně je rozebráno primární téma této práce, kterým je React a React Native, kde jsou rovněž popsány jejich základní koncepty, hlavní funkce a případné rozdíly mezi těmito technologiemi.

Praktická část se zabývá implementací již zmíněné aplikace pro zadávání poznámek. Před samotnou implementací této aplikace byly vytvořeny grafické návrhy uživatelského rozhraní jak pro webovou, tak mobilní verzi. Tato aplikace byla vytvořena ve webové verzi díky Reactu a následně jako mobilní aplikace pro platformu Android za pomocí Reactu Native. Z webové aplikace byl vytvořen instalační balíček pro mobilní zařízení s operačním systémem Android, a to díky technologii Capacitor a vývojovému prostředí Android studio.

Pro serverovou část aplikací bylo využito technologie Firebase, která vyřešila autentizaci uživatelů a ukládání jejich poznámek do databáze.

Tyto aplikace se testovaly jak během jejich vývoje, tak po jejich dokončení, kde testy probíhaly z role uživatele. Výstupem praktické části je tak webová aplikace, která je hostována na Vercelu a dva instalační balíčky této aplikace pro Android.

SEZNAM POUŽITÉ LITERATURY

[1] A short history of the Web. CERN [online]. Switzerland: CERN, c2022 [cit. 2022-05-16]. Dostupné z: https://home.cern/science/computing/birth-web/short-history-web

[2] HTML basics. Resources for Developers, by Developers [online]. San Francisco: Mozilla Corporation, c1998–2022 [cit. 2022-05-16]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics

[3] What is JavaScript?. Resources for Developers, by Developers [online]. San Francisco: Mozilla Corporation, c1998–2022 [cit. 2022-05-16]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

[4] CHRISTENSSON, Per. Backend definition. TechTerms: The Computer Dictionary [online]. Minneapolis: Sharpened Productions, c2022, April 11, 2020 [cit. 2022-05-16]. Dostupné z: https://techterms.com/definition/backend

[5] Android (operační systém). Wikipedie, otevřená encyklopedie [online]. San Francisco: Wikipedia, c2022 [cit. 2022-05-16]. Dostupné z: https://cs.wikipedia.org/wiki/Android_(opera%C4%8Dn%C3%AD_syst%C3%A9m)

[6] IOS. Wikipedie, otevřená encyklopedie [online]. San Francisco: Wikipedia, c2022 [cit. 2022-05-16]. Dostupné z: https://cs.wikipedia.org/wiki/IOS

[7] Why did we build React?. React a JavaScript library for building user interfaces [online]. Menlo Park: Meta Platforms, c2022 [cit. 2022-05-16]. Dostupné z: https://reactjs.org/blog/2013/06/05/why-react.html

[8] Document Object Model. Wikipedie, otevřená encyklopedie [online]. San Francisco: Wikipedia, c2022 [cit. 2022-05-16]. Dostupné z: https://cs.wikipedia.org/wiki/Document_Object_Model

[9] Virtual DOM and Internals. React a JavaScript library for building user interfaces [online]. Menlo Park: Meta Platforms, c2022 [cit. 2022-05-16]. Dostupné z: https://reactjs.org/docs/faq-internals.html

[10] Introducing JSX. React a JavaScript library for building user interfaces [online]. Menlo Park: Meta Platforms, c2022 [cit. 2022-05-16]. Dostupné z: https://reactjs.org/docs/introducing-jsx.html

[11] BANKS, Alex a Eve PORCELLO. Learning React: functional web development with React and Redux. Beijing: O'Reilly Media, 2017. ISBN 978-1-491-95462-1.

[12] EISENMAN, Bonnie. Learning React Native: building mobile applications with JavaScript. Beijing: O'Reilly, [2016]. ISBN 978-1-491-92900-1.

[13] Style. React Native Learn once, write anywhere [online]. Menlo Park: Meta Platforms, c2022 [cit. 2022-05-16]. Dostupné z: https://reactnative.dev/docs/style

[14] Navigating Between Screens. React Native Learn once, write anywhere [online]. Menlo Park: Meta Platforms, c2022 [cit. 2022-05-16]. Dostupné z: https://reactnative.dev/docs/navigation

[15] Thinking in React. React a JavaScript library for building user interfaces [online]. Menlo Park: Meta Platforms, c2022 [cit. 2022-05-19]. Dostupné z: https://reactjs.org/docs/thinking-in-react.html

[16] Components and Props. React a JavaScript library for building user interfaces [online]. Menlo Park: Meta Platforms, c2022 [cit. 2022-05-19]. Dostupné z: https://reactjs.org/docs/components-and-props.html

[17] Firebase Authentication. Make your app the best it can be [online]. Mountain View: Google, c2022 [cit. 2022-05-19]. Dostupné z: https://firebase.google.com/docs/auth

[18] Firebase Realtime Database. Make your app the best it can be [online]. Mountain View: Google, c2022 [cit. 2022-05-19]. Dostupné z: https://firebase.google.com/docs/database

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

DOM	Document Object Model
API	Application Programming Interface
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
SQL	Structured Query Language
VDOM	Virtual Document Object Model
UI	User Interface
JSX	JavaScript XML

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: PŘILOŽENÉ CD

Přiložené CD k bakalářské práci obsahuje:

- bakalářskou práci ve formátu PDF,
- zdrojové soubory všech aplikací,
- instalační balíčky mobilních aplikací.