FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Implementace mobilní aplikace FLOWIO s využitím konceptu WebView
Student:	Bc. Luka Lukašević
Vedoucí:	Ing. Petra Pavlíčková, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce zimního semestru 2021/22

Pokyny pro vypracování

Cílem práce je implementace nativních prvků mobilních aplikací do webové aplikace vykreslené ve WebView tak, aby se výsledná aplikace z uživatelského hlediska chovala co nejvíce jako mobilní aplikace.

1. Nastudujte a analyzujte koncept WebView.
2. Na základě poznatků prozkoumejte možnosti propojení konceptu WebView s programovacím jazykem React JS za pomoci nativních prvků z React Native.
3. Navrhněte kombinaci těchto tří technologií tak, aby se výsledná aplikace chovala z uživatelského hlediska jako nativní aplikace. S tím, že hlavní snahou je co nejméně měnit kód existující webové aplikace, kterou chceme zobrazovat na mobilních zařízeních.
4. Návrh aplikujte na podnikovou mobilní aplikaci FLOWIO.
5. Výsledek otestujte a vyhodnoťte náročnost použitého způsobu vývoje mobilní aplikace v porovnání s klasickým vývojem pomocí nativního jazyka.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 31. července 2020

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Implementace mobilní aplikace FLOWIO s využitím konceptu WebView

Bc. Luka Lukašević

Katedra softwarového inženýrství
Vedoucí práce: Ing. Petra Pavlíčková, Ph.D.

22. prosince 2020

Poděkování

Nejdříve bych chtěl poděkovat paní Petře Pavlíčkové, za skvělé vedení práce a za velmi příjemnou spolupráci i v této nelehké době. Dále bych chtěl poděkovat kolegům z práce Jakubovi Petrovi a Martinovi Pongráczovi, za vnuknutí nápadu práce a za cenné rady při jejím vypracovávání.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 22. prosince 2020 . . . . . .

Odkaz na tuto práci

Lukašević, Luka. *Implementace mobilní aplikace FLOWIO s využitím konceptu WebView*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Tato diplomová práce se zabývá studiem konceptu WebView a následnou aplikací poznatků na podnikovou aplikaci FLOWIO. Cílem práce je funkční mobilní aplikace FLOWIO, která bude z uživatelského hlediska vystupovat jako nativní mobilní aplikace. Hlavní myšlenkou bylo usnadnění procesu vývoje pro společnosti, které potřebují k existující webové aplikaci vytvořit i odpovídající mobilní aplikaci. Po detailní analýze konceptu WebView jsem zpracoval požadavky na mobilní aplikaci FLOWIO a implementoval adekvátní řešení. V práci jsem ukázal, že lze využít WebView k zobrazení i složitějších webových aplikací, než jsou pouhé statické webové stránky.

Klíčová slova WebView, FLOWIO, cross-platformní aplikace, React JS, React Native

Abstract

This diploma thesis deals with the study of the WebView concept and the subsequent application of knowledge to the enterprise application FLOWIO. The aim of the work is a functional mobile application FLOWIO, which will act as a native mobile application from the user's point of view. The main idea was to facilitate the development process for companies that need to create

a corresponding mobile application for an existing web application. After a detailed analysis of the WebView concept, I processed the requirements for the FLOWIO mobile application and implemented an adequate solution. In my work I showed that WebView can be used to display even more complex web applications than just static web pages.

Keywords WebView, FLOWIO, cross-platform applications, React JS, React Native

Obsah

Seznam obrázků

Úvod

V dnešní době je snahou vše digitalizovat a automatizovat. Vzniká stále víc a víc aplikací, které se nám snaží usnadnit život a urychlit nebo ulehčit práci kterou děláme. Jsou různé typy aplikací, ale nejrozšířenějšími jsou v dnešní době zcela určitě webové a mobilní aplikace. V posledních letech nabývají na oblíbenosti právě mobilní aplikace, jelikož každý z nás má mobilní telefon a používá ho každý den. Čteme si na nich zprávy, posloucháme hudbu, hledáme informace, komunikujeme s jinými lidmi a mnoho dalších věcí. Tím pádem vzniká na trhu i větší poptávka po programátorech, kteří programují mobilní aplikace. Poptávka je ale větší, než je počet schopných a spolehlivých programátorů mobilních aplikací. Firmy, jenž vyvíjí webové aplikace, musí řešit uživatelskou spokojenost a musí sledovat nové trendy a trh. Tím pádem je zapotřebí vedle webové aplikace mít i odpovídající mobilní aplikaci pro vyšší spokojenost uživatelů. Veliké firmy s dobrou pověstí a vysokým majetkem tento problém snadno vyřeší, jelikož finanční prostředky pro ně nejsou problém a mohou si dovolit na zakázku mobilní aplikaci pro svůj produkt vytvořit na míru a nebo založit celé nové oddělení pro mobilní vývoj. Na trhu je ovšem mnoho menších firem a startupů, které takové prostředky nemají a vyvíjet dvě aplikace současně je pro ně velmi časově i finančně náročné.

Existují ale možnosti, které tento problém mohou do začátku alespoň částečně vyřešit. Takové možnosti vychází ze skutečnosti, že stačí mít hotovou webovou aplikaci a za použití určitých postupů a konceptů z ní vytvořit mobilní aplikaci. A to bez nutné znalosti nativních programovacích jazyků a nebo s použitím pouhých základů těchto jazyků. Výsledek ovšem nebývá tak uspokojivý, jako když by aplikace byla naprogramována přímo jako mobilní aplikace. Splní ale svůj účel, kterým je předat uživatelům funkční produkt do doby, než bude možno dodat nativní aplikaci. Z možností, které máme v této době dostupné, jsem si vybral koncept WebView a budu se jím v této práci zabývat.

WebView je obecně používaný koncept a mnoho mobilních aplikací ho využívá tak dobře, že uživatelé ani nepoznají rozdíl od nativní aplikace. Je to jednoduché, když je zapotřebí pouze zobrazit jednu nebo dvě webové stránky sloužící spíše k informativním účelům. Já budu ve své práci zkoumat, jak nejlépe využít konceptu WebView k implementaci funkční mobilní aplikace. Po důkladné analýze budu poznatky aplikovat na podnikovou aplikaci FLOWIO. Výsledkem práce bude funkční mobilní aplikace, která se bude z uživatelského hlediska chovat zcela jako nativní mobilní aplikace.

Hlavním přínosem této práce by měla být podrobná studie konceptu WebView a jeho propojení s vybraným programovacím jazykem. Kromě seznámení s konceptem, se budu snažit především proniknout do jeho hloubky a maximálně využít jeho možnosti k dosažení požadovaného výsledku, kterým je funkční mobilní aplikace vycházející z již existující webové aplikace. Motivací k výběru tohoto tématu mi byl nedostatek materiálů, které by popisovaly řešení zadaného problému. Jak už jsem zmínil výše, nebyl jsem schopný dohledat podrobnější informace, kromě ukázek využití základních prvků konceptu a samotné dokumentace různých typů komponent WebView. Taková dokumentace avšak může být nepřehledná a bez ukázek použití nemusí na první pohled dávat potřebné řešení. Při podrobnějším prozkoumání dokumentace, lze zjistit, že WebView má daleko větší možnosti, než je pouhé vykreslování webové stránky, na zadané URL (Uniform Resource Locator). Aplikace poznatků z analytické části práce na podnikovou aplikaci FLOWIO krásně ukáže využití těchto možností.

Kapitola 1

Seznámení se s zadanými technologiemi a souvisejícími pojmy

Tato kapitola popisuje technologie a postupy, které jsou v práci zmíněny a kterými se práce zabývá. Nejedná se o analýzu, ale pouze o seznámení čtenáře s termíny, aby pak v částech jako je analýza a aplikace zjištěných poznatků na reálný problém, bylo vše srozumitelné a jasné. Hlavním a nejdůležitějším termínem práce je zcela jistě WebView. Tento koncept se ale pojí s programovacími jazyky, které tento koncept implementují. V tomto případě je použit programovací jazyk JavaScript a jeho knihovny React, která slouží pro tvorbu uživatelského rozhraní a React Native, která slouží k tvorbě mobilních aplikací. Poté budou v analytické části rozebrány způsoby propojení těchto programovacích jazyků s konceptem WebView. Tato práce by mohla do budoucna sloužit jako ukázka možných způsobů použití WebView pro složitější projekty, proto je nutné možnosti WebView dopodrobna rozebrat. Začnu ale od základních termínů, jako například co jsou aplikace a jaké typy aplikací se v práci objeví.

1.1 Aplikace

Aplikace, nebo aplikační software, je software, který je vyvinut k tomu aby prováděl specifické úkoly se specifickým účelem. V podstatě je to program, nebo kolekce programů, které jsou využívány koncovými uživateli. V podstatě každý software, který není systémový software nebo programovací software je aplikační software. Tyto dva další typy softwaru se v práci nebudou objevovat, proto se jimi dále nebudu zabývat. V závislosti na činnosti, pro kterou byla aplikace navržena, může aplikace manipulovat s textem, čísly, zvukem, grafikou a nebo tyto prvky kombinovat. Některé balíčky aplikací, se zaměřují pouze na jeden úkol, například na zpracování textu (MS Word). Jiné aplikace,

které zahrnují více aplikací, nazýváme integrovaný software.

Příklady aplikačních softwarů:

- Software pro zpracování textu
- Databázové programy
- Zábavný software
- Obchodní software
- Vzdělávací software
- Software pro počítačový design (CAD)
- Tabulkový software a další[2]

Webové a mobilní aplikace patří do skupiny integrovaných aplikací a budou blíže popsány v této kapitole.

1.2 Webová aplikace

Webová aplikace je typ aplikačního softwaru, který je uložen na vzdáleném serveru a je doručován přes internet prostřednictvím rozhraní internetového prohlížeče. Webové služby jsou podle definice webové aplikace, a mnoho webových stránek, ne ale všechy, obsahují webové aplikace. Podle editora portálu Web.AppStorm Jarela Remicka se jakákoliv webová součást, která pro uživatele vykonává nějakou funkci, klasifikuje jako webová aplikace.

Webové aplikace mohou být navrženy pro nejrůznější účely a může je využívat kdokoliv, od organizace po jednotlivce, kvůli různým důvodům. Mezi běžně využívané aplikace patří webmaily, online kalkulačky nebo e-shopy. K některým webovým aplikacím mají přístup pouze konkrétní prohlížeče nebo pouze určité země, většina je však volně dostupná.

1.2.1 Webová stránka vs. webová aplikace

Webová aplikace

Webová aplikace je software nebo program, který je přístupný pomocí libovolného internetového prohlížeče. Její front-end je většinou vytvářen pomocí programovacích jazyků jako je HTML, CSS a JavaScript, které jsou podporovány všemi prohlížeči. Back-end může využívat jakýkoliv programovací stack jako je LAMP (Linux, Apache, MySQL, PHP/Perl/Python), MEAN (bezplatný open source JavaScriptový software stack) a další. Na rozdíl od mobilních aplikací, nepotřebují žadný specifický SDK (Software Development Kit) pro vývoj. Webové aplikace se dostaly do popředí s příchodem Software as a Service (SaaS).[3]

Webová stránka (website)

Webová stránka (website) je skupina globálně dostupných, provázaných webových stránek, které mají stejnou doménu. Můžou být vyvíjeny a udržovány jednotlivcem, podnikem nebo celou organizací. Cílem webové stránky (website) je sloužit různým účelům, například jako blog. Webová stránka (website) je hostována na jednom nebo na více webových serverech. Jsou přístupné prostřednictvím sítě, jako je internet nebo pokud je síť privátní tak pomocí IP adresy. Jelikož je WebView používáno v případě této práce v kombinaci s webovou aplikací, nebudu se webovými stránkami dále zabývat.[4]

1.2.2 Výhody webové aplikace

Webové aplikace můžeme využívat na mnoho způsobů, a s těmito způsoby přichází mnoho výhod. Mezi základní výhody webových aplikací patří:

- Umožňují přístup více uživatelům ve stejnou chvíli k stejné verzi aplikace.
- Webové aplikace není potřeba instalovat.
- Webové aplikace nezávisí na platformě. Můžeme k nim přistupovat pomocí počítače, telefonu, tabletu a dalších zařízení, které se dokážou připojit k síti.
- Jsou přístupné skrze různé internetové prohlížeče

1.2.3 Webové aplikace vs. jiné typy aplikací

V rámci odvětví mobilních počítačů, jsou webové aplikace někdy v kontrastu s nativními aplikacemi, což jsou aplikace vyvinuté speciálně pro konkrétní platformu nebo zařízení a jsou na něm nainstalované. Nativní aplikace mohou běžně využívat hardware specifický pro dané zařízení, například GPS nebo fotoaparát na mobilním telefonu.

Programy, které kombinují oba dva přístupy, jsou nazývány hybridní aplikace. Hybridní aplikace fungují podobně webovým aplikacím, jsou ale nainstalovány na zařízení jako nativní aplikace. Hybridní aplikace mohou také využívat výhod speciálních prvků pro zařízení pomocí interní API (Application Programming Interface). Stažené nativní aplikace mohou fungovat i bez připojení do sítě, hybridní aplikace tuto výhodu bohužel nemají. Hybridní aplikace typicky sdílí podobné prvky v navigaci jako webové aplikace, jelikož jsou postaveny spíše nad webovou aplikací než nad nativní.[3]

1.3 Mobilní aplikace

Mobilní aplikace jsou rostoucí odvětví, které přitahuje společnosti a podniky ze všech různých trhů. Není se čemu divit, podle portálu statista.com, se předpokládá, že zisky mobilních aplikací dosáhnou za rok 2020 až k hranici 600 miliard dolarů [5]. Díky narůstající popularitě chytrých telefonů a tabletů, se mobilní aplikace staly extrémně populárními mezi podnikovými majiteli po celém světě.

1.3.1 Co je mobilní aplikace

Mobilní aplikace je typ aplikace, která je navržena tak, aby běžela na mobilním zařízení, kterým může být chytrý telefon nebo tablet. I přes to, že jsou mobilní aplikací většinou malé softwarové jednotky s limitovaným počtem funkcí, stále dokážou dopřát uživatelům kvalitní služby a zkušenosti. Na rozdíl od aplikací, které jsou navrženy pro stolní počítače a notebooky, jdou mobilní aplikace směrem pryč od integrovaných softwarových systémů. Každá mobilní aplikace dodává nějakou limitovanou a izolovanou funkcionalitu.

Například to může být hra, kalkulačka nebo internetový prohlížeč. Jelikož mobilní zařízení ze začátku neměly tolik výkonný hardware, museli se vývojáři vyhnout multifunkčnosti aplikací. Dnes už jsou mobilní zařízení daleko výkonnější, někdy i více než některé počítače, i přes to ale zůstala mobilním aplikacím jednoduchá funkcionalita. Takto majitelé mobilních aplikací umožňují zákazníkům vybírat přesně ty funkce, které by jejich zařízení mělo mít.

1.3.2 Typy mobilních aplikací

Mobilní aplikace mají různé účely a jsou různě velké. Zde je výpis nejpopulárnějších typů aplikací, na které můžeme na trhu narazit.

- **Herní aplikace** - Nejpopulárnější kategorie aplikací. Podniky investují stále více a více zdrojů do vývoje mobilních aplikací, jelikož jsou z tohoto typu aplikací na trhu největší zisky. Podle nedávné studie představují mobilní aplikace 33% všech stažených aplikací, 74% zisku peněz od uživatelů a 10% veškerého času při používání aplikací [6].
- **Podnikové a produktivní aplikace** - Tyto aplikace se drží hned za herními aplikacemi co se poptávky na trhu týče. Lidé jsou stále náchylnější k používání chytrých telefonů a tabletů k provádění mnoha složitých úkolů na cestách. Tyto aplikace jim například pomáhají s rezervací lístků, posíláním emailů nebo sledováním stavu jednotlivých úkolů v práci. Podnikové aplikace jsou vytvářeny tak, aby podpořily produktivitu a minimalizovaly výdaje tím, že umožní uživatelům provádět širokou škálu úkolů, od nákupu firemních potřeb přes nabírání nových zaměstnanců.

- **Vzdělávací aplikace** - Tato kategorie zahrnuje aplikace, které pomohou uživatelům získat nové zkušenosti a znalosti. Vzdělávací herní aplikace jsou výborným nástrojem pro děti. Mnoho vzdělávacích aplikací usnadňuje kantorům jejich práci. Někteří je používají pro organizaci a pro jejich vlastní vzdělání do budoucna.

- **Aplikace pro životní styl** - Aplikace, které se soustředí na různé aspekty osobního života. Můžou se zaměřovat na nákup, módu, virtuální kabinky, fitness, online seznamky nebo dietní jídelníčky.

- **Nákupní aplikace** - Nejpopulárnější nákupní aplikace jakou jsou Amazon nebo eBay dodávají verze jejich počítačových řešení i pro mobilní telefony. Umožňují tím uživatelům přistupovat k jejich produktům prostřednictvím chytrých telefonů a tabletů a využívat platebních možnosti těchto zařízení jako je Apple Pay nebo Android Pay.

- **Aplikace pro zábavu** - Tyto aplikace umožňují uživatelům sdílet video obsah, hledat události, chatovat nebo sledovat obsah online. Sociální media jako Facebook nebo Instagram jsou tím nejlepším příkladem. Dalšími jsou aplikace jako Netflix nebo HBO Go, které umožňují uživatelům sledovat video obsah a udržují si zájem uživatelů tak, že často aktualizují jejich obsah.

- **Nástrojové aplikace** - Tyto aplikace jsou tak běžné, že občas zapomínáme, že je vůbec používáme. Jsou to například zdravotní aplikace, skenery čárových kódů nebo sledovače.

- **Cestovní aplikace** - Hlavním nápadem těchto typů aplikací je umožnit uživatelům cestovat jednoduše. Cestovní aplikace udělají z chytrého telefonu nebo tabletu cestovní deník a uloží všechny potřebné informace o cestě přehledně na jedno místo. Dále také pomáhají například v přepravě a ubytování osob, jako je Uber nebo AirBnb.[7]

1.3.3 Vývoj mobilních aplikací

Vývoj mobilních aplikaci je proces, který se velmi podobá procesu vývoje klasických aplikací. Vývojáři se avšak musí zaměřit na to, aby vytvořili software, který bude využívat výhody unikátních funkcí mobilních zařízení. Nejjednodušším scénářem pro vývoj mobilních aplikací je vzít klasickou počítačovou aplikaci a vložit jí do mobilního zařízení. Čím je ale klasická aplikace robustnější, tím problematičtější je tato technika. Lepší přístup je vyvíjet aplikace dělané přímo pro mobilní prostředí. Tato technika může využívat všech výhod, které mohou mobilní zařízení poskytnout. Tento proces bere v úvahu jejich omezení a pomáhá majitelům firem nastolit rovnováho mezi náklady a funkčností.

Moderní chytré telefony a tablety jsou vybaveny funkcemi jako je Bluetooth, NFC (Near Field Communication), GPS, gyroskopické senzory, kamery a mnoho jiných. Vývojáři pak mohou tyto funkce využívat pro vytvoření aplikací s technologiemi jako je virtuální realita, snímání čárových kódů, služby pro kontrolu polohy a další. Nejúspěšnější a nejpopulárnější mobilní aplikace využívají tyto funkce chytrých telefonů na ten nejlepší možný způsob. Ne všechny chytré telefony mají ale stejný hardware a proto jsou zde i komplikace. Vývojáři vyvíjející aplikace pro iOS mohou očekávat, že tyto aplikace poběží na dvou typech zařízení, kterými jsou IPhony a IPady. Android vývojáři musí ale počítat s tím, že každý chytrý telefon a tablet mohou mít různý hadrware a různé verze operačních systémů. Vedle toho je zde problém pro podniky, které se rozhodnou vedle existující webové aplikace vytvořit odpovídající mobilní aplikaci a to takový, že musí myslet na různorodost typů mobilních zařízení a vytvořit tak hned dvě mobilní aplikace (iOS a Android)[8].

1.4 Hybridní aplikace

Hybridní aplikace fungují trošku jako hybridní auta. Kombinují starší technologie (HTML, CSS, JavaScript) s inovativními novými technologiemi (mobilní zařízení) a vytváří produkt, který pomáhá snižovat spotřebu paliva (zdroje vývojářů). Hybridní aplikace používá jedinou kódovou základnu ke sdílení logiky aplikace mezi více zařízeními (mobilní vs. webové) a platformami (iOS vs. Android). Jinak řečeno, vývojáři nemusí řešit na jakým zařízení aplikaci poběží a jediné o co se musí starat, je aby se aplikace dokázala jednotlivým zařízením přizpůsobit. Hybridní aplikace toto dokážou pomocí naprogramování aplikace v běžných jazycích jako je HTML, CSS a JavaScript v kombinaci s malým množstvím nativního kódu sloužícím pro práci s prvky specifickými pro zařízení, jako jsou senzory, GPS, data a sítě.

1.4.1 Výhody hybridních aplikací

Když se při vývoji mobilních aplikací rozhoduje, jestli se bude aplikace vyvíjet jako nativní aplikace nebo jako hybridní aplikace, je třeba určit jaké jsou výhody a jaké jsou nevýhody jednotlivých přístupů. Technologie se rychle vyvíjí a trh se musí efektivně a svižně přizpůsobovat aby udržel krok s nejnovějšími trendy. Hybridní aplikace nám umožňují dostat se na požadovaný trh rychleji, levněji a efektivněji.

1.4.1.1 Rychleji

Sdílený kód znamená méně času. Schopnost hybridních aplikací sdílet jednu kódovou základnu (codebase) napříč několika platformami nabízí tempo vývoje, které by nebylo v případě vývoje oddělených aplikací pro různé platformy

možné. Vývojáři hybridních aplikací jsou také schopni využívat dobře definované a dokumentované API a velkou komunitu uživatelů, kteří pomůžou diagnostikovat příčinu problému a vyřešit ho. Dále jsou schopní využít existující knihovny pro použití základních funkcionalit jako je přihlašování, autentizace, platby a další.

1.4.1.2 Levněji

Čas jsou peníze a hybridní aplikace nejsou levnější jenom díky tomu, že jsou rychleji vytvořeny. Vzhledem k tomu, že jsou hybridní aplikace vyvíjeny pomocí široce používaných programovacích jazyků, je také jednodušší najít vývojáře za rozumnou mzdu nebo smluvní sazbu. Talentovaní vývojáři nativních aplikací jsou horkým zbožím na trhu a proto mohou požadovat za své služby vyšší ceny.

Hybridní aplikace umožňují také jednomu vývojovému týmu efektivně pracovat na vytváření aplikace pro více platforem ve srovnání s dvěma nebo více odlišnými týmy, kde každý vyvíjí trošku odlišnou aplikaci. Je nutno podotknout, že hybridní aplikace mohou být levnější, ale hrozí zde to, že se postupem času stanou nákladnějšími. Pokud tým udělá zásadní chyby v architektuře nebo dostane nějaký technický dluh, může dojít k celkovému přepsaní kódové základny.

1.4.1.3 Lépe

Je sporné říkat lépe v každém případě, jelikož v některých případech prostě dává smysl vyvíjet nativní aplikaci. První verze aplikace Instagram byla například k dispozici pouze pro iOS, protože byla původní verze vytvořena pro dobový fenomén známý jako iPhonografie - fotky pořízené pomocí iPhone zařízení. Nakonec se však Instagram rozhodl aplikaci předělat na hybridní aplikaci, aby bylo možné nabídnout sdílenou základnu kódu, lepší offline zážitek a sdílené WebView. Hybridní aplikace mohou být jednoduše konfigurované a nasazené jako Progresivní Webová Aplikace (PWA), což umožňuje aplikaci běžet v webovém prohlížeči jako je Chrome, Safari, Firefox a mnoho dalších.

1.4.2 Nevýhody hybridních aplikací

Neexistuje zde žádná jednoduché řešení při vývoji mobilních aplikací. Ekosystém je neustále v pohybu a programátoři se musí starat o to, aby byl stále vyvážený. Firma může v průběhu vývoje čelit různým omezením, jako je například čas do uvedení aplikace na trh, vyschnutí finančních zdrojů a dalším problémům. Musí proto pečlivě zvážit výhody a nevýhody strategických rozhodnutí v rámci životního cyklu aplikace.

1.4.2.1 Grafika a animace

Hybridní aplikace zvládají jednoduše transformovat základní elementy jako jsou tlačítka, formuláře, checkboxy a další do nativního UI. Na druhou stranu komplexní animace, grafiky a interakce s hardwarem je složitější. Animace je možné vykreslit v hybridních aplikacích pomocí WebView, kód se ale provádí pomaleji než by tomu tak bylo u nativního kódu, který má přímý přístup k hardwaru zařízení. To znamená, že grafika a animace nemusí být tak plynulé při používání aplikace. To je hlavní důvod toho, proč vývojáři mobilních her upřednostňují nativní jazyky před hybridním přístupem.

1.4.2.2 Pluginy a buginy

Hybridní aplikace nejsou vyvíjeny pomocí nativních programovacích jazyků, které jsou přirozeně kompatibilní s operačním systémem zařízení, na kterém aplikace poběží. Proto musí hybridní aplikace využívat pluginy. Pluginy pomůžou aplikaci běžet plynule, ale nesou sebou pár úskalí. Vzhledem k tomu, že pluginy vytvářejí třetí strany, je možné že funkcionalita, které je důležití pro chod aplikace, bude zastaralá a nebo nebude vůbec v pluginu implementovaná. Tato funkcionalita může být tak důležitá, že se tým rozhodne pro přepsání pluginu nebo jeho vytvoření, a to může vést k další finanční zátěži. Pluginy jsou také velmi náchylné k chybám v aplikaci. Tyto chyby většinou bývá velmi obtížné nalézt a opravit, jelikož chybové hlášky a dokumentace jsou psány různými lidmi s různými konvencemi a každý plugin je většinou dílem někoho jiného.

1.4.2.3 Limitace v produkci

Už jsem diskutoval limity hybridních aplikací, které zahrnují kvalitu grafiky a animací a problémy, které mohou nastat při použití pluginů. Celkově by se tedy dalo říci, že hybridní aplikace nejsou tak rychlé jako nativní mobilní aplikace. Hybridní aplikace potřebují nějaký zabalený interface a pluginy, aby bylo možné aplikaci zkompilovat a používat jí na mobilním zařízení. Tyto kroky navíc podstatně zpomalují rychlost načítání aplikace.

I navzdory těmto nevýhodám hybridních aplikací, jsou zde možnosti, jak tyto omezení obejít a dodat tak uživatelům plynulou a funkční aplikaci. Například Instagram přišel s velmi efektivním a elegantním řešením, jak využít správně čas, potřebný pro načtení fotografie. Když uživatel přidá fotografii, začne se automaticky nahrávat na pozadí, zatímco uživatel píše popisek a vybírá tagy. Poté když je uživatel připraven fotografii nahrát, bude to vypadat, že se nahrála okamžitě, bez jakýchkoliv prodlev.[9]

1.5 WebView

Když se firma rozhodne pro vývoj mobilní aplikace, musí myslet na dvě rozdílné platformy jako jsou iOS a Android. Pokud už existuje android aplikace, je potřeba vyvinout podobnou aplikaci pro iOS a obráceně. Když dojde k aktualizaci v jednom kódu, musí se aktualizovat i ten druhý. Všechnu tu těžkou práci je potřeba vykonávat dvakrát. Není to ideální stav, je ale potřeba dělat vše co se dá, aby se obešla omezení kladená na to, co je možno vyvinout.

Je zde ale naděje. Je tu cesta, jak použít 80% stejného kódu na vývoj iOS i Android aplikace. Strategie je taková, že se aplikace napíše pomocí programovacích jazyků jako je JavaScript, HTML a CSS. Poté se použije koncept WebView (kontejner pro webové stránky) v nativní mobilní aplikaci k tomu, aby zobrazil stránky v aplikaci (ukládání uživatelských preferencí. dynamického obsahu, grafiky a dalších). HTML, CSS a JavaScript by měly být vyhovující, aby prováděly všechny tyto operace uvnitř WebView, na jakémkoliv zařízení, které podporuje WebView.

Aplikace, které využívá konceptu WebView je primárně utvořena z JavaScriptu, HTML a CSS souborů. Webovou aplikaci tvoří klasicky jedna nebo více webových stránek. Tyto stránky utvoří front-end rozhraní. Dalo by se tedy říci, že WebView je okno, skrz které vaše zařízení zobrazuje tyto webové stránky. Normálně uživatelé používají pro zobrazení webových stránek webové prohlížeče. WebView v tomto případě nahrazuje webové prohlížeče. Místo toho, aby uživatelé mohli měnit webové stránky, WebView zobrazí pouze stránky, které souvisí přímo s aplikací. Mechanicky je aplikace pouze směsice webových stránek. Esteticky se ale aplikace chová a vypadá jako nativní aplikace. To je koncept WebView.

WebView může ale pokrýt nejvýše 80% kódu pro oba systémy. Zbylých 20% bude potřeba pokrýt kódem, který je specifický pro daný operační systém, aby se WebView chovalo správně jak by mělo. WebView pro iOS se liší od WebView pro Android a je proto zapotřebí psát specifický kód pro každý jiný typ WebView. Jestli se někdo rozhodne pro vytvoření aplikace pro obě dvě platformy ve WebView, musí pracovat s unikátními technikami WebView, pro dva různé světy. Je třeba podotknou, že odhad 80% ku 20%, je velmi hrubá aproximace poměru WebView kódu a kódu specifického pro různé platformy. Procenta mohou kolerovat pro různé projekty s ohledem na to, co bude jejich cílem a čeho budou chtít dosáhnout.[10]

1.6 Použité programovací jazyky a podpůrné technologie

Jazyky, jako je HTML a CSS, které jsou úzce spjaté s WebView, jsem se rozhodl v práci blíže nepopisovat, jelikož nemají na cíl práce žádný vliv ani dopad. Pro seznámení se s použitými programovacími jazyky jsem se rozhodl zvolit obecnou kapitolu a jednotlivé jazyky popsat pouze okrajově. Těmito jazyky se budu hlouběji zabývat v analytické části práce, kde budu popisovat jejich možnosti propojení pro získání uživatelsky přívětivé nativní aplikace ve formě WebView. Popíši zde jazyky jako jsou JavaScript, React a React Native. Dále zde popíši technologie, které s těmito jazyky jdou ruku v ruce a jejich používání usnadňuje programátorům každý další den.

1.6.1 JavaScript

JavaScript je scriptovací jazyk, který se používá ke kontrole a vytváření dynamických částí webové aplikace. Jinak řečeno, vše co se pohybuje nebo jakkoliv jinak mění na obrazovce bez manuální aktualizace webové aplikace nebo stránky. Může to být animovaná grafika, prezentace fotek, našeptávání textu, interaktivní formuláře a další. Skriptovací jazyky jsou programovací jazyky, které slouží k automatizaci práce, kterou by bylo jinak zapotřebí provádět krok za krokem. Díky tomu, že je JavaScript tolik integrovaná webová funkcionalita, všechny hlavní prohlížeče už mají v základu vestavěné nástroje, které umožňují vykreslovat JavaScript. JavaScript je možné psát přímo do HTML dokumentu a není zapotřebí stahovat dodatečné kompilátory. JavaScript se používá jak pro front-end tak pro back-end aplikace.[11]

1.6.2 React JS

React je JavaScriptová knihovna, která se využívá pro vytváření rychlých a interaktivních rozhraní pro weby a mobilní aplikace. Je to pro všechny dostupná (open-source), komponentově založená, front-endová knihovna, které se stará pouze o View vrstvu aplikace. V MVC architektuře je View vrstva zodpovědná za to, jak aplikace vypadá a jakou dává uživatelům zkušenost. React byl vytvořen Jordanem Walkerem, softwarovým inženýrem z Facebooku. React rozděluje UI (uživatelské rozhraní) na komponenty, kde každá komponenta má svoji unikátní funkcionalitu.

Hlavní sílou reactu jsou dva vestavěné typy proměnných, které se nazývají stav (state) a vlastnosti (properties, zkraceně props). State slouží k ukládání dat a props k předávání dat mezi komponentami. React si uchovává lehčí verzi opravdového DOMu (Document Object Model) v paměti, jinak také známou jako Virtuální DOM (VDOM). Tím pádem, když dojde v některé z komponent ke změně stavu nebo vlastnosti komponenty (dojde ke změně

proměnných state nebo props), VDOM změní pouze tu jednu dotyčnou komponentu v DOM a ušetří tím znovunačítání celého DOM. To značně urychluje chod aplikace.[12]

1.6.3 NPM

NPM je zkratka pro Node Package Manager a je to základní metoda, jak řídit a zacházet s balíčky v běhovém prostředí Node.js. Používá se prostřednictvím klienta z příkazového řádku a přistupuje k databázi veřejných a prémiových balíčků známých jako registr npm. Většina balíčků je open-source a uživatelé je mohou dle libosti používat. Npm i jeho registr jsou spravovány společností npm, Inc.[13]

1.6.4 Yarn

Yarn byl vyvinut společností Facebook s cílem vyřešit nějaké neduhy výše zmíněného npm. Technicky vzato, Yarn není náhražkou za npm, jelikož závisí na některých modulech z npm registrů. Yarn by se dal tedy brát jako nový instanční prvek, který stále vychází se stejné struktury npm. Registr jako takový se nezměnil, ale pouze jeho instalační metoda. Hlavní rozdíl oproti npm je jeho rychlost, yarn je skoro dvakrát rychlejší než poslední verze npm. Díky tomu, že yarn využívá stejné balíčky jako npm je přechod z npm na yarn plynulý a není potřeba měnit workflow.[14]

1.6.5 React Native

React Native nebo také jinak známý jako RN, je velmi populární, rozšířená, JavaScriptově založená knihovna, určená pro vývoj mobilních aplikací jak pro iOS tak pro Android. Knihovna umožňuje vytvářet aplikace pro různé aplikace použitím stejného kódového základu. React Native byl vyvinut stejně jako React JS společností Facebook v roce 2015. Už za pár let se z něj stal fenomén ve vývoji mobilních aplikací. V React Native jsou napsány jedny z největších mobilních aplikací na světě, jako jsou Facebook, Instagram nebo Skype. Díky tomu, že v React Native lze vyvíjet aplikace pro iOS i pro Android souběžně, mohly firmy, které ho začaly používat značně ušetřit peníze i čas. React Native je založen na React JS, který je stejně populární v oddělení webových aplikací. Díky tomu není problém pro zkušené React vývojáře přejít na vývoj mobilních aplikací, jenž byl dříve záležitostí jazyků, které se značně lišily.[15]

Kapitola 2

Analýza použitých technologií

V této části bych se chtěl věnovat podrobné analýze WebView. Nejdříve zmíním další možnosti, jak lze v dnešní době vyvinout mobilní aplikace jinak, než za použití nativního programovacího jazyka a poté také zhodnotím, z jakého důvodu jsem se rozhodl zrovna pro koncept WebView. Budu se v této kapitole také věnovat jednotlivým programovacím jazykům, které jsem zmínil výše a popíšu způsob, jak nejlépe je využít v kombinaci s WebView.

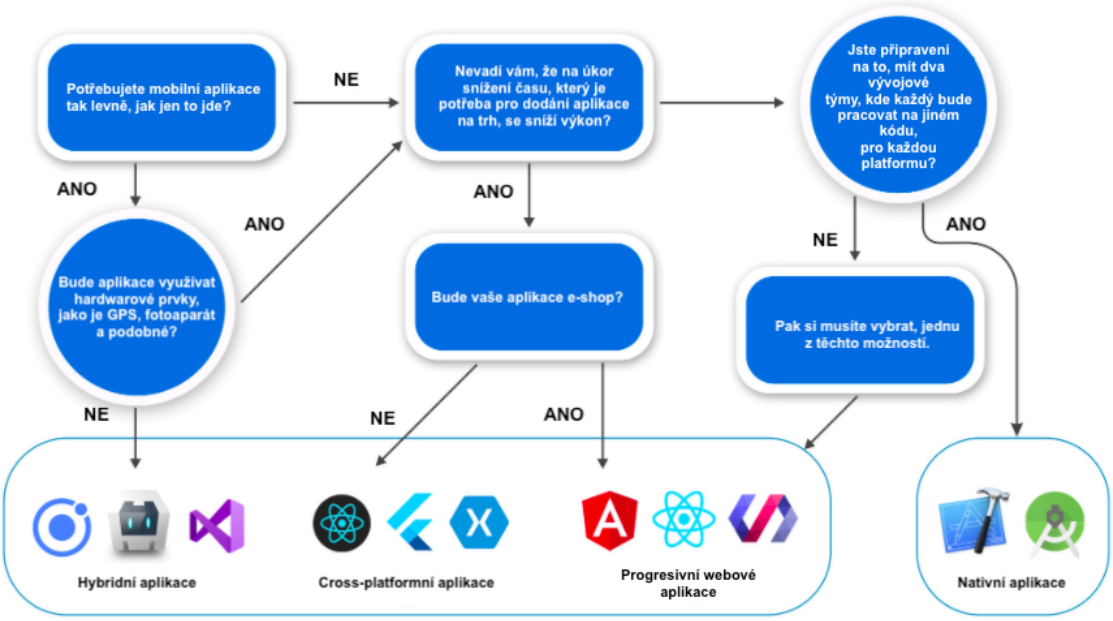
2.1 Další způsoby, jakými lze dosáhnout stejného výsledku jako s WebView

Jak už je zmíněno v názvu práce i v úvodu, pro vývoj mobilní aplikace, která bude mít stejnou kódovou základnu jak pro systém iOS tak pro Android, jsem zvolil koncept WebView. Mobilní aplikace, která je implementována formou WebView, je typ hybridní mobilní aplikace. Existují ale další možnosti, jak docílit stejného výsledku. Rád bych tedy předtím než se dostanu k analýze samotného WebView zmínil tyto další možnosti, vyzdvihl jejich výhody a nevýhody a vysvětlil, z jakého důvodu jsem zvolil právě koncept WebView.

V dnešní době, již není nativní aplikace jediná cesta, pro vytvoření funkční a uživatelsky přívětivé mobilní aplikace. Firmy se mohou rozhodnout, jestli zvolí cestu hybridní aplikace, která spočívá na webových technologiích. Nebo mohou využívat benefity cross-platformních vývojových nástrojů jako je React Native nebo Flutter. Na obrázku 2.1 můžete vidět diagram, jak postupovat při rozhodování, jaký způsob pro vývoj mobilní aplikace by měl nejvíce vyhovovat požadavkům společnosti.

2.1.1 Nativní vývoj mobilní aplikace

Tento způsob vývoje mobilní aplikace je samozřejmě nejžádanější, ale ne všichni si takový luxus mohou dovolit. Zahrnuje to vytvoření mobilní aplikace, která je



Obrázek 2.1: Zvolení přístupu vývoje mobilní aplikace [1]

přizpůsobena každé platformě zvlášť. V tomto případě programátoři využívají programovací jazyky, které jsou nativní k danému operačnímu systému. Může to být Java nebo v poslední době Kotlin pro Android, a Swift nebo Objective-C pro iOS. Uživatelé vždy ocení nativní aplikace kvůli jejich výkonu a na míru šitém UX (User Experience). Tento přístup však odradí především startupy kvůli finanční náročnosti. Cena je klíčovým rozdílem mezi nativním a hybridním vývojem mobilních aplikací.

Při vývoji nativní aplikace mají vývojáři plný přístup ke všem možnostem daného zařízení. To jim umožňuje vyvíjet pokročilejší funkcionality jako je správa paměti, komplexní síťování a další. Největší výzvou je zde ale rozběhnout aplikaci na dvou platformách. K docílení toho je zapotřebí oddělené kódová základna. Některá nativní rozšíření umožňují sdílení kódu. Například můžete sdílet C++ knihovny mezi iOS a Android aplikací za pomoci Java Native Interface.

Vývojáři při vývoji iOS aplikací používají nástroje jako je XCode, AppCode, Atom a pro Android aplikace zase Android Studio, Android IDE nebo IDEA od Intellij. Android aplikace lze vyvíjet jak na macOS tak na Windows, iOS aplikace lze ale vyvíjet pouze na systému macOS a to také může být další finanční zátěží pro firmy, jelikož Apple počítače bývají dražší než ty na kterých běží Windows. Čistě nativní aplikace jsou pak například Artsy, Pinterest nebo iOS kalkulačka.[16]

2.1.2 Hybridní vývoj mobilní aplikace

Aplikace nemusí být nativní, aby uživatelům dopřála dobrou mobilní zkušenost. Projekt může mít jiné priority, jako je například dostat aplikaci rychle na trh. Hybridní aplikace není jenom levnější alternativa, pro její vývoj je zapotřebí menší množství času a umožňuje sdílení kódu. Druhou stranou mince může být pomalý výkon a neoptimální uživatelská zkušenost. Teoreticky je však možné dosáhnout skvělého UX a navigačních vzorů z vizuálního hlediska, je zapotřebí ale zkušených programátorů aby aplikace splňovala takové požadavky. Více o hybridních aplikacích je popsáno v sekci 1.4.

Dvěma nejznámnějšími frameworky pro psaní hybridních mobilních aplikací jsou zcela jistě Ionic a Apache Cordova.[17]

2.1.2.1 Apache Cordova

Apache Cordova je open-source mobilní framework. Umožňuje využívat standardní webové technologie jako je HTML, CSS a JavaScript pro cross-platformní vývoj. Aplikace je vždy zaobalena do kontejneru, který odpovídá platformě, na které je aplikace spouštěna. Při přístupu k senzorům, datům a síťovým prvkům zařízení spoléhá na standardy vyhovujícího API. Aplikace psané v tomto frameworku vychází z souboru config.xml, kde jsou uloženy informace o aplikaci a specifikuje parametry, které ovlivňují chod aplikace. Aplikace jako taková je implementované jako webová stránka, která začíná souborem index.html, které odkazuje na styly CSS, JavaScript, soubory, fotografie a další prvky, které jsou důležité pro chod aplikace.

Aplikace je spouštěna jako WebView v nativním kontejneru, ze kterého se vytvoří binární soubor a je vložen na Apple Store nebo Google Play. WebView s podporou Apache Cordova může tvořit celou aplikaci, ale může také být použito pouze jako komponenta ve větší hybridní aplikaci, kde se kombinuje WebView s nativním komponentami aplikace.[18]

2.1.2.2 Ionic

Ionic je framework který se používá k vývoji mobilních aplikací za použití HTML, CSS a JavaScriptu. Kromě těchto jazyků ho lze kombinovat také s jinými front-endovými jazyky, jako je Angular, React a další. Ionic vychází a vždycky vycházel z Apache Cordova viz 2.1.2.1. Ionic samotný je spíše UI (User Interface) framework, který poskytuje upravené vstupní a výstupní elementy, které jsou navrženy pro mobilní zařízení.[19]

2.1.3 Cross-platformní vývoj mobilní aplikace

Je nutné hned ze začátku říci, že hybridní a cross-platformní způsob vývoje nejsou to samé. Jediným společným rysem těchto dvou přístupů je sdílení kódu pro různé platformy. Díky tomu oba dva způsoby umožňují značně urychlit vývoj. Cross-platformní způsob vývoje využívá nativní vykreslovací modul. Kódová základna napsaná v JavaScriptu se napojí na nativní komponenty přes tak zvané mosty. Díky tomuto přístupu je možné docílit skoro stejné uživatelské zkušenosti jako při používání nativní aplikace. Nabízí bezproblémovou funkčnost, snadnou implementaci a náklady na vývoj aplikace nemusí být vysoké. Přizpůsobení různým platformám je však omezováno používaným frameworkem a to si může vybrat svojí daň na výkonu aplikace. Takovými frameworky mohou být React Native, Xamarin nebo Flutter. Mnoho velmi používaných aplikací bylo vytvořeno tímto přístupem a klasický uživatel by nepoznal rozdíl. Je to například aplikace Bloomberg, Insightly nebo Reflectly.[20]

React Native

React Native už jsem popisoval v sekci 1.6.5 a dále v práci se blíže budu zabývat pouze jednotlivými komponentami, které poskytuje pro řešení zadaného problému. Jediné co zmíním je to, že s použitím tohoto frameworku lze dosáhnout dobrého výkonu aplikace. Na rozdíl od Apache Cordova totiž převádí zdrojový kód na nativní aplikaci místo toho, aby aplikace běžela v zabudovaném prohlížeči jako je WebView. V tomto frameworku ale existuje knihovna, které umožňuje využívat WebView jako komponentu a lze ji kombinovat s nativním přístupem, který dodává React Native. Tyto dvě technologie spolu mohou vzájemně komunikovat a lze tedy využít lepší výkon který dodává React Native v kombinaci s daleko větším množstvím znovupoužitelného kódu, kterým se pyšní WebView.[21]

Xamarin

Xamarin je nástroj, jenž se využívá pro cross-platformní vývoj mobilních aplikací a umožňuje vývojářům sdílet až 90% kódu napříč hlavními platformami. Xamarin přišel na svět v roce 2011 pod vedením Miguela Icaza jako open-source vývojová platforma založená na framworku .NET. V roce 2016 získal Xamarin Microsoft a udělal z něj balíček pro vývoj softwaru (SDK), který je dostupný jako doplněk ve svých vývojových prostředích. Technicky řečeno, Xamarin používá nativní knihovny jazyka C# zabalené v .NET vrstvě určené pro cross-platformní mobilní vývoj.

Zatímco kód, spojený s business logikou, přístupy do databáze a síťovými prvky může být sdílený mezi jednotlivými platformami, Xamarin umí vytvářet vrstvy UI specifické pro každou platformu. Díky tomu cross-platformní apli-

kace vytvořené pomocí Xamarinu vypadají 100% jako nativní aplikace a poskytnou uživatelům lepší uživatelský zážitek než hybridní aplikace.[22]

Flutter

Flutter je bezplatný a veřejně dostupný UI framework, používaný pro vývoj cross-platformních aplikací, vydaný společností Google v roce 2017. Jako podobné frameworky, umožňuje vytvářet mobilní aplikace pro různé platformy za použití jedné kódové základny. Flutter se skládá ze dvou částí a to z

1. kolekce nástrojů, určené pro jednodušší vývoj aplikací zvané SDK. To zahrnuje nástroje potřebné pro kompilaci kódu do nativního strojového kódu (kód pro Android nebo pro iOS).

2. A knihovny založené na widgetech. Což je kolekce znovupoužitelných UI elementů jako jsou tlačítka, textová vstupy, nadpisy a další, které lze upravovat dle potřeby.

Flutter používá programovací jazky Dart, což je jazyk, který představila společnost Google v roce 2011 a za ty roky se z něj stal velmi používaný jazyk. Dart se zaměřuje na front-end, lze jej tedy použít k programování mobilních a webových aplikací. Je to objektově orientovaný jazyka a má podobnou syntaxi jako JavaScript. [23]

2.1.4 Progresivní webová aplikace

Progresivní webová aplikace, zkráceně PWA, je nová technologie se kterou přišla na trh firma Google. Díky této technologii lze na mobilním zařízení přidat webovou aplikaci na domovskou stránku chytrého telefonu ve formě ikonky a umožňuje k této stránce přistupovat i bez připojení k síti. Aby se z webové aplikace stala PWA je zapotřebí pouze tří věcí.

1. Přidat ikonku pro webovou aplikaci na domovské stránce zařízení.

2. Přidat manifest pro webovou aplikaci.

3. Přidat nějakého správce služeb, který se bude starat o načítání aplikace i bez připojení k síti, o rychlejší načítání a o zasílání push notifikací (notifikace vyskakující když uživatel nepoužívá aplikaci).

Z pohledu rychlosti vývoje a finanční náročnosti je tento způsob zaručeně nejlepší. Pokud existuje webová aplikace, stačí ji pouze doplnit o tři aspekty nutné pro PWA a mít responzivní design pro mobilní aplikace. Když se uživatel ve svém chytrém telefonu dostane přes internetový prohlížeč na webovou stránku nebo aplikaci, která má upravený zdrojový kód podle PWA požadavků, vyskočí lišta, jenž dá uživateli vědět, že je tato webová aplikace PWA a může si jí uložit na domovskou stránku zařízení. Tato lišta obsahuje většinou krátkou

informaci a tlačítko, na které když se klikne, tak se provede akce uložení. Jak to vypadá lze vidět na obrázku 2.2.[24]

Obrázek 2.2: Vyskakovací lišta pro stažení PWA

Nastavení aplikace musí splňovat jistá pravidla, aby internetový prohlížeč vyhodnotil aplikaci jako PWA a zobrazil vyskakovací lištu. Prohlížeč zobrazí lištu když:

- Webová aplikace ještě není stažena do zařízení.
- Splňuje heuristiku zapojení uživatelů.
- Webová aplikace používá https protokol.
- Zahrnuje manifest webové aplikace, který musí mít atributy jako:
  - *short_name* nebo *name*

  - *icons* - musí zahrnovat 192px a 512px ikonu
  - *start_url*
  - *display* - musí být jedno z *fullscreen*, *standalone* nebo *minimal-ui*
- Registrovaný správce služeb [25]

Tyto kritéria platí pro prohlížeč Chrome, kde pro ostatní se mohou v nějakých aspektech lišit. Tyto odchylky nejsou ale důležité a proto se jimi nebude dále zabývat. Výhodou tohoto přístupu je zcela určitě jednoduchost řešení, možnost spouštění webových stránek bez přístupu do sítě, push notifikace a aplikace, které jsou PWA, mají lepší ranking při hledání klíčových slov v prohlížeči. Tato technologie ale pouze zaobaluje prohlížeč, není to plně funkční nativní aplikace a technicky to je pořád webová stránka. Metriky jako interakce, animace a výkon se nemohou rovnat nativním aplikacím jelikož je uživatelské rozhraní pouze internetový prohlížeč bez URL části v horní části zařízení. A největším nedostatkem této technologie je nulová podpora pro iOS systémy (iPhony a iPady).

Avšak i přes tyto nedostatky je tato technologie využívána a našla uplatnění především v sektoru internetových obchodů. PWA jsou například Flipkart, Forbes nebo AliExpress.[26]

2.1.5 Výsledné shrnutí a důvod proč bylo zvoleno WebView

Ať už má společnost již hotovou webovou aplikaci a k ní chce vytvořit odpovídající mobilní aplikaci a nebo chce vytvořit pouze mobilní aplikaci, měla by před samotným vývojem určit, jaká možnost se pro danou firmu a typ aplikace nejlépe hodí. V sekcích výše, jsem zmínil ty nejčastěji a nejvíce užívané postupy a technologie když dojde na řešení tohoto problému. Na čem ale záleží nejvíce, jsou požadavky společnosti nebo zákazníka na aplikaci. Má být aplikace rychle vydaná, je dostatek finančních prostředků na vývoj mobilní aplikace, má být aplikace připravena na vysokou uživatelskou zátěž, to znamená má být výkonná? Jestli dokáže společnost jasně určit tyto aspekty, tak poté může snáze určit správný postup pro vývoj aplikace. Pro lepší přehled lze v tabulce 2.1 vidět krátkou rekapitulaci všech metod na jednom místě.

Typ aplikace	Nativní	Hybridní	Cross-platformní
Nástroje	XCode	Ionic	React Native
	AppCode	Apache Cordova	Xamarin
	Android Studio	Visual Studio	Flutter

Způsob vykreslení	Nativní	V prohlížeči	Nativní
Knihovny	Nezávisí moc na open-source knihovnách a na platformě (každá platforma má svůj kód)	Silně závisí na knihovnách a frameworcích	Silně závisí na knihovnách a frameworcích
Ladění (debugging)	Nativní debuggovací nástroje	Nativní + webové debuggovací nástroje	Závísí na použitém frameworku
Kódová základna	Oddělený kód - pro každou platformu jeden kód	Jeden kód s potencionálními platformě specifickými prvky	Jeden kód s potencionálními platformě specifickými prvky
Výhody	• Plný přístup k funkcím zařízení • Výborný výkon • Nativní UI • Efektivní běh aplikací • Vysoká kvalita funkcionalit a UX	• Nízší cena za vývoj • Podpora různých OS • Znovupoužití kódu • Cenově efektivní vývoj	• Podpora různých OS • Výkon UI je skoro tak rychlý jako u nativního přístupu • Znovupoužití kódu • Cenově efektivní vývoj (je potřeba se ale naučit nový jazyk nebo přístup)

Nevýhody	• Nemá žádnou podporu pro multi-platformní vývoj • Vysoké výdaje na vývoj • Žádná znovupoužitelnost kódu	• Pomalejší výkon • Limitovaný přístup k prvkům OS • Žádná interakce s jinými nativním aplikacemi	• Pomalejší výkon • Ještě limitovanější přístup k OS prvkům než hybridní aplikace • Slabá interakce s nativními aplikacemi

Tabulka 2.1: Tabulka porovnání jednotlivých metod

WebView jsem si jako téma práce nezvolil pouze kvůli tomu, že je jeho použití špatně dokumentováno a kombinace s jinými technologiemi chabě popsaná, ale také proto, že ho budu aplikovat na vývoj mobilní aplikace k podnikové aplikaci FLOWIO. Aplikace FLOWIO byla vyvinuta jako webové aplikace k tomu, aby zjednodušovala a automatizovala procesy nad různými datovými zdroji. Aby se aplikace dala používat snadno a efektivně i jinde než v pohodlí kanceláře z počítače, bylo zapotřebí vyvinout i odpovídající mobilní aplikaci. Hlavním požadavkem bylo, aby existovala funkční mobilní aplikace, která by se mohla začít používat pro testovací účely v rámci interního prostředí firmy. Rychlost byla hlavním parametrem z toho důvodu, že bylo nutné předvést fungování aplikace v provozu a tím si získat větší finanční podporu k rozvinutí potenciálu. Aby bylo možné vyvinout funkční mobilní aplikaci pro iOS i pro Android souběžně, je zapotřebí dvou větších týmů schopných vývojářů. To by znamenalo nábor nových lidí a vysoké finanční náklady a ani jednoho se v tomto případě nedostávalo. Mobilní verze aplikace FLOWIO navíc nevyužívá žádných nativních prvků mobilních zařízení jako je GPS, Bluetooth nebo kamera a v kombinaci s požadavky, které byly na aplikaci kladeny z hlediska vývoje, bylo jednoduché určit, že ta správná cesta nepovede přes vývoj aplikace pro každý operační systém zvlášť v nativním jazyce.

Mobilní aplikace by se měla ve většině případů chovat stejně jako webová, je tam ale pár prvků, ve kterých se přeci jen budou lišit. Z toho důvodu nepřipadalo v úvahu zvolit postup jakým je PWA, kdy dojde pouze k vykreslení obsahu webové aplikace do té mobilní. Mobilní verze aplikace potřebuje

nějaký kontejner, ve kterém bude definována rozdílná logika od té webové. Přesně takový problém řeší hybridní přístup vývoje mobilních aplikací s použitím technologie Apache Cordova nebo Ionic, kde dojde k vykreslení webové aplikace pomocí WebView, ale je možné z kontejneru s WebView komunikovat a měnit obsah webové aplikace dle potřeb té mobilní. Front-end část webové aplikace je vyvinuta za použití frameworku React JS a tím pádem i celý vývojový tým front-end části je orientovaný na tento jazyk. Ionic i Cordova jsou sice vhodnými kandidáty, syntax je ale jiná a vývojáři by strávili drahocenný čas studiem nových technologií. Ideální by v tomto případě byl zmiňovaný framework React Native, který vychází z React JS a má velmi podobnou syntaxi. Jelikož komunita React Native vývojářů vytvořila WebView knihovnu i pro tento jazyk, bylo rozhodnuto zvolit kombinaci jazyku React Native, který poslouží jako kontejner pro vykreslení webové aplikace jako mobilní, za použití WebView konceptu. S použitím tohoto přístupu budu možné souběžně s webovou aplikací vyvíjet i tu mobilní a při správné kombinaci těchto technologií lze dosáhnout podobného výsledku, jako při vývoji nativní aplikace, za mnohem kratší dobu.

2.2 Koncept WebView

V této sekci se budu podrobně věnovat analýze samotné komponenty WebView a popíšu její důležité metody a atributy, pomocí kterých lze snadno komunikovat s webovou částí uvnitř WebView a s vnější React Native částí. Při správném použití těchto vlastností WebView můžeme dosáhnout uživatelsky velmi přívětivého výsledku.

WebView bylo součástí React Native už od samého začátku jazyka. Mělo ale velmi základní funkčnost, kterou bylo pouhé vykreslení webové stránky ze zadané url do React Native kontejneru. Vývojáři viděli v této komponentě ale mnohem větší potenciál a proto se rozhodli přijít se samostatným modulem, který přichází s celou řadou nových a vylepšených funkcionalit. Primárním cílem práce, je prozkoumat a aplikovat možnosti propojení WebView s webovou aplikací a jazykem React Native, tím pádem se budu zabývat pouze parametry, které jsou důležité pro samotnou WebView komponentu. Tím pádem budu předpokládat, že existuje funkční forma React Native aplikace, kde je potřeba přidat pouze WebView komponentu. WebView modul pro React Native se dá nainstalovat jako každý jiný javascriptový modul a to pomocí **npm** 1.6.3 nebo **yarn** 1.6.4 balíčkových manažerů. V tomto případě jsem se rozhodl používat pro instalaci balíčků yarn, jelikož je to novější a rychlejší technologie oproti npm. WebView knihovna se tedy nainstaluje provedením následujícího příkazu z příkazové řádky v kořenovém souboru aplikace:

```
1  yarn add react-native-webview
```

Pro použití WebView v iOS aplikaci nejsou zapotřebí žádné další kroky. Pro android platformy verze 6.x.x a vyšší, je zapotřebí upravit soubor *android/-gradle.properties* tak, že přidáte následující dva řádky kódu:

```
android.useAndroidX=true
android.enableJetifier=true
```

První řádek indikuje, že v projektu chceme používat AndroidX. Druhý řádek automaticky konvertuje existující knihovny třetích stran tak, jako když by byly napsané pro AndroidX.[27]

2.2.1 Props komponenty WebView

Komponenty jsou soběstačné datové struktury, které by měly mít jednu hlavní funkci v aplikaci. Komponent v aplikaci bývá z pravidla velmi mnoho a musí spolu nějakým způsobem umět komunikovat. Jak jsem popisoval v sekci 1.6.2 může mít každá komponenta definované vstupní proměnné nazývané props. Pomocí těchto proměnných může komponenta získávat informace a data od nadřazených komponent a může jim díky funkcím data posílat zpět. WebView z balíčku react-native-webview je také komponentou a správné pochopení a vyzkoumání jejích props je stěžejní pro tuto práci. V následujících sekcích popíši interface důležitých props WebView komponenty a budu diskutovat možnosti jejich použití. Webová aplikace, která je zobrazena pomocí WebView, má díky JavaScriptu lokání proměnnou s referencí na WebView komponentu, pomocí které lze přistupovat k WebView metodám, přes které lze komunikat s kontejnerem, ve kterém je WebView zobrazeno. Tyto metody nebudu diskutovat v rámci této sekce, ale v sekcích, které patří jazykům kde se tyto metody používají.

2.2.1.1 source

Source je hlavním atributem WebView komponenty. Pomocí tohoto atributu totiž komponenta dostane informaci o tom, co se má vlastně ve WebView vykreslit. Datový typ tohoto atributu je *object* a obsah webové stránky, kterou chceme pomocí WebView zobrazit, můžeme v tomto objektu poslat třemi způsoby.

Jednoduché jednořádkové HTML

Nejjednodušším způsobem, pro využití WebView, je jednoduše napsat kus HTML kódu, který chceme zobrazit. Objekt je v JavaScriptu brán jako asociativní pole, což znamená, že každý prvek pole (objektu) má k sobě přiřazenou právě jednu hodnotu. Těmto prkvům se říká vlastnosti a dále budu používat tento termín. Pokud tedy chceme zobrazit obsah pomocí jednořádkového HTML, musíme v objektu vytvořit vlastnost s názvem *html* a přiřadit jí HTML ve formě datového typu *string*. Při použití tohoto způsobu zobrazení webové

stránky je zapotřebí WebView kompontě poslat prop *originWhiteList* a nastavit mu hodnotu na `['*']`. Tento prop bude blíže popsán v sekci 2.2.1.2.

```
<WebView
     originWhitelist={['*']}
     source={{ html: '<h1>Toto je staticky HTML zdroj<\h1>' }}
/>
```

Listing 2.1: Ukázka použití propu source s inline HTML

Změna stringu vyvolá znovuvykreslení WebView obsahu, lze tedy zobrazovat jednoduché html stránky dynamicky v interakci s uživatelem.

URL zdroj

Tento způsob je nejčastějším případem užití pro WebView. Objektu posílanému do propu source je nastavena vlastnost *uri* a přiřazena hodnota ve formátu odkazu na webovou stránku, kterou chceme začít vykreslovat ve WebView. Datový typ této hodnoty je *string*. Po zobrazení webové stránky na dané adrese se WebView začne chovat jako prohlížeč a navigace mezi jednotlivými webovými stránkami probíhá v rámci dané aplikace. Ve WebView neexistuje tlačítko zpět jako v internetovém prohlížeči a proto musí dostatečné navigace být obstarána v rámci webové aplikace. Pro návrat zpět do nativní aplikace lze použít různé triky, kterými se budu zabývat v dalších sekcích.

Spolu v kombinaci s vlastností *uri* lze objektu posílanému do propu source přidat ještě další tři doplňující vlastnosti.

- **method** (datový typ *string*) - Specifikuje jakou HTTP metodu pro danou url adresu použít. Pokud není uvedeno, je použita automaticky GET metoda. Jedinými možnými metodami jsou POST a GET.
- **headers** (datový typ *object*) - V této datové struktuře lze specifikovat dodatečné HTTP hlavičkové atributy pro použitý dotaz. Lze nastavovat například cookies nebo jiné atributy jako u klasických HTTP dotazů. Je možné používat pouze pro GET dotazy. Pokuď chceme posílat do WebView cookies, je třeba na komponentě WebView přidat prop *sharedCookiesEnabled* a nastavit jeho hodnotu na `true`.
- **body** (datový typ *string*) - Atribut HTTP body, který se odešle se zadaným dotazem. Hodnota této vlastnosti musí být validní UTF-8 string a bude odeslán přesně jak je specifikován. To znamená bez aplikování dodatečného kódování (URL-escaping nebo base64). Pouze pro POST dotazy.[28]

```
    <WebView
          source={{
              uri: 'http://example.com',
```

```
            headers: {
              Cookie: 'cookie1=asdf; cookie2=dfasdfdas',
            },
          }}
          sharedCookiesEnabled={true}
        />
```

Listing 2.2: Ukázka načtení dat z URL odkazu s posíláním cookies

Načtení lokálních HTML souborů

Tato možnost v aktuální verzi knihovny react-native-webview není dostupná a je diskutována mezi vývojáři v repozitáři knihovny na Githubu. Tato metoda je velmi zřídka používána a není vhodným řešením problému, kterým se zabývá tato práce, proto se tímto způsobem načtení webové stránky nebudu dále zabývat.

2.2.1.2 originWhiteList

Tato vlastnost komponenty WebView slouží ke kontrole, kam všude se může uživatel v rámci WebView navigovat. WebView nemá žádnou hlavičku jako internetový prohlížeč, kde by se dala měnit url webové stránky a pohybovat se tak různě po internetu. Jestli ale webová stránka obsahuje nějaké odkazy, které vedou mimo danou stránku, mohlo by se stát, že se uživatel už zpět do aplikace nedostane a musel by jí celou restartovat. Prop `originWhitelist` má datový typ pole stringů a hodnotami může být seznam původních URL stringů, kde se ve WebView může uživatel pohybovat. V případě, že uživatel klikne na odkaz, který se neshoduje s žádným stringem ze zadaného pole, otevře se stránka v zvoleném prohlížeči mobilního zařízení (Safari, Chrome, Mozilla a další) a WebView bude stále zobrazovat požadovaný obsah.

Například pokud chcete, aby se uživatel nedostal v rámci WebView jinam, než na stránku https://mojestranka.com a její pod stránky, tak stačí předat vlastnostem hodnoty viz následující kód

```
<WebView
  source={{ uri: 'https://mojestranka.com/' }}
  originWhitelist={['https://mojestranka.com/*']}
/>
```

Listing 2.3: Příklad použití propu originWhiteList

2.2.1.3 renderLoading

Získání obsahu webové stránky z URL adresy předané v `source` propu může nějakou chvíli trvat a proto je velmi důležité aby uživatel věděl, že se něco děje a že se něco zobrazí. K tomu existuje prop `renderLoading`, jehož hodnota má

datový typ funkce. Tato funkce vrátí uživatelem definovaný obsah, který informuje uživatele o načítání HTML obsahu ze zadané URL. Ve většině případů funkce vrací animaci ve formě načítacího kolečka, kterou můžeme jednoduše získat z nějakého balíčku dostupného z npm. Spolu s propem `renderLoading` musíme dodat WebView komponentě i prop `startInLoadingState` a nastavit jeho hodnotu na *true*. Tímto propem oznamujeme WebView, že ji dodáváme indikátor načítání a chceme ho zobrazit do té doby než se načte obsah.

```
<WebView
    source={{ uri: 'https://reactnative.dev' }}
    startInLoadingState={true}
    renderLoading={() => <Loading />}
/>
```

Listing 2.4: Použití propu renderLoading

2.2.1.4 injectedJavaScript

Tento prop je jedním ze způsobů, jak lze komunikovat s webovou aplikací, kterou WebView zobrazuje. Tento prop dělá přesně to, jak je pojmenovaný. Pomocí tohoto propu můžeme předat script ve formě stringu, který se zavolá v zobrazované webové aplikaci ihned poté, co poprvé dojde k jejím načtení do WebView. Tento script se zavolá pouze jednou a to při prvním načtení obsahu, kde nezáleží, jestli dojde k přenačtení stránky nebo se naviguje někam jinam.

Forma scriptu musí respektovat JavaScriptové konvence pro správnou kompilaci a je možné používat veškeré zabudované JavaScriptové příkazy, které se provedou v rámci webové aplikace. Důležité je, aby script končil vždy příkazem *true*. To zabrání případnému spadnutí webové aplikace, pokud by se ve scriptu vyskytla nějaké syntaktická chyba nebo přístup k nedefinovaným proměnným. Správnost provedení scriptu by měla být ošetřena na straně webové aplikace.

injectedJavascriptForMainFrameOnly

Jestli přidáme do WebView tento prop a nastavíme jeho hodnotu na *false*, docílíme toho aby se script zavolal na každé obrazovce, ne pouze na té hlavní. Výchozí hodnota tohoto propu je *true*. Tato funkcionalita není podporována na zařízeních s operačním systémem Android a proto je doporučováno se bez toho propu obejít.

Následující kus kódu by po načtení stránky https://google.com ve WebView změnil barvu pozadí na červenou a po uplynutí dvou sekund, by vypsal do okénka upozornění s textem „Hello World“.

```
const INJECTED_JAVASCRIPT = `
  document.body.style.backgroundColor = 'red'
  setTimeout(function() { window.alert('Hello World'), 2000);
  true;
`

<WebView
  source={{ uri: 'https://google.com" }}
  injectedJavaScript={INJECTED_JAVASCRIPT}
/>
```

Listing 2.5: Ukázka použití propu injectedJavaScript

2.2.1.5 injectedJavaScriptBeforeContentLoaded

Tento prop má stejný interface a velmi podobnou funkci jako prop 2.2.1.4. Jsou rozdílné pouze v době zavolání předaného scriptu. Jak to má tento prop v názvu, tak zavolá script před tím, než dojde poprvé k načtení obsahu WebView. Tento prop je vhodný, když je zapotřebí aby webová stránka měla určité informace před tím, než dojde k jejímu načtení. Tudíž se hodí přidávat vlastnosti do systémových proměnných JavaScriptu *window* a *document* nebo nastavovat data do lokálního uložiště *localstorage*.

injectedJavaScriptBeforeContentLoadedForMainFrameOnly

Tento prop přiřadíme WebView komponentě a nastavíme jeho hodnotu na *false* v případě, že chceme aby se script zavolal před načtením každé obrazovky.

2.2.1.6 onFileDownload

Tento prop je možné používat v případě, že je aplikace spouštěna na mobilním zařízení s operačním systémem iOS. Hodnotou je funkce, která se zavolá v případě, že operační systém detekuje žádost o stažení souboru. Parametrem funkce je datová struktura, kde jsou obsaženy informace potřebné k tomu, aby programátor mohl správně zpracovat ukládaná data.

Operační systém iOS verze 13 a novější už přichází s vlastím API, které dokáže přistupovat k záhlaví HTTP odpovědí, kde dokáže určit jestli se jedná o stahování nebo ne. Na operačních systémech iOS verze 12 a nižší pouze MIME (typ internetového média) typy, které nemohou být vykresleny pomocí WebView komponenty, spustí funkci předanou v propu onFileDownload.

K tomu aby bylo možné ukládat fotografie do galerie zařízení, je zapotřebí specifikovat povolení v souboru, který se nachází v kořenovém adresáři aplikace ve složce *ios/[název-projektu]/Info.plist*

```
<key>NSPhotoLibraryAddUsageDescription</key>
<string>Save pictures for certain activities.</string>
```

Android má zabudovanou integraci s manažerem stahování a žádosti o stahování souborů jsou řešeny na úrovni operačního systému. V případě, že chce programátor tyto akce odchytávat, existuje mnoho knihoven třetích stran které toto umožňují. Například balíček `rn-fetch-blob` nebo `react-native-fs`.

```
<WebView
  source={{ uri: 'https://google.com" }}
  onFileDownload={({ nativeEvent }) => {
    const { downloadUrl } = nativeEvent;

    // --- kod pro zpracovani odkazu a pripadne ulozeni dat
  }}
/>
```

Listing 2.6: Ukázka použití propu onFileDownload

2.2.1.7 onMessage

Tento prop je zřejmě nejdůležitějším propem WebView komponenty co se komunikace s webovou aplikací týče. Přes tento prop totiž dostává WebView data z webové části. Hodnotou propu je funkce, která se zavolá, když dojde k zavolání WebView metody `postMessage` viz 2.2.2.1 z webové aplikace. Funkce jenž je hodnotou tohoto propu má jediný argument, kterým je string obsahující komprimovaná data z webové části aplikace. Je na programátorovi aby zajistil, že webová část pošle data ve formě stringu, jinak dojde k chybě a aplikace spadne. V těle funkce dojde pak k získání dat z tohoto stringu a nativní část určí, jak s nimi dále naloží. Zde je velmi důležitá správně zvolené strategie komprese dat, velikost stringu není nijak omezena a proto lze posílat velký obnos dat tímto způsobem. Více se těmito technikami budu zabývat v části vývoje.

```
<WebView
  source={{ uri: 'https://google.com" }}
  onMessage={({ nativeEvent }) => {
    const { data } = nativeEvent;

    // --- kod pro zpracovani komprimovanych dat
  }}
/>
```

Listing 2.7: Ukázka použití propu onMessage

2.2.1.8 onLoad

Hodnotou tohoto propu je funkce, která se zavolá, když WebView komponenta dokončí načítání obsahu. Jediným argumentem funkce je objekt zvaný `nativeEvent`, který má následující vlastnosti.

- canGoBack
- canGoForward
- loading
- target
- title
- url

V těle funkce pak lze provádět další akce, které budou pracovat s informací, že WebView dokončilo načítání.

2.2.1.9 onError

Hodnotou tohoto propu je funkce, která se zavolá v případě, že načítání WebView obsahu selže. Argumentem funkce je `nativeEvent` který má následující vlastnosti.

- canGoBack
- canGoForward
- loading
- target
- title
- url
- description
- didFailProvisionalNavigation
- domain (pouze pro iOS)
- code

2.2.1.10 onScroll

Hodnotou tohoto propu je funkce s jedním argumentem. Argumentem je objekt datového typu `NativeScrollEvent`. K zavolání funkce dojde v případě, že definovaný posluchač události zaznamená posunutí v obsahu zobrazovaném pomocí WebView.

2.2.2 Metody komponenty WebView

Pomocí props můžeme předat komponentě informace se kterýma dále pracuje a jsou to její vlastnosti. Komponenty mohou mít ale i svoje metody a právě díky těm dokáže s WebView komunikovat jak nativní kontejner, tak webová aplikace. K tomu, aby se daly tyto metody používat, je zapotřebí mít veřejnou instanci WebView komponenty. Tato instance je dostupná jak v nativní části kódu, kde se volá samotná WebView komponenta, tak v webové aplikaci. Pokud je webová stránka nebo aplikace vykreslena formou WebView, vytvoří WebView referenci na sebe sama a uloží tu referenci do globální proměnné `window`. Tuto proměnnou lze používat pomocí JavaScriptu kdykoliv je potřeba a instance WebView je uložena pod vlastností `window.ReactNativeWebView`. V nativní části aplikace je vytvoření reference na programátorovi. WebView komponenta se používá k vykreslení webového obsahu a zaujímá místo ve vykreslovací části funkce zvané render. Programátor vytvoří proměnou, kterou prováže s komponentou WebView přes její prop zvaný `ref`. Tím uloží referenci na komponentu do proměnné a zpřístupní si tak celý interface veřejných metod, které WebView poskytuje. V části, kde budu popisovat vývoj jednotlivých metod blíže, popíšu vytvoření reference a ukážu správný postup jejího vytvoření.

2.2.2.1 postMessage

Tato metoda je důležitá především pro webovou aplikaci, která je zobrazována pomocí WebView. K této funkci lze přistupovat ve webové aplikaci pomocí globální proměnné `window.ReactNativeWebView`. Funkce přijímá jeden argument jehož datový typ musí být vždy string. Tato metoda je jediným způsobem jak může webová aplikace komunikovat s nativní částí. Provedení této funkce aktivuje na straně WebView prop `onMessage` viz 2.2.1.7 kde se zpracuje string, který funkce `postMessage` odešle.

Zde je základní ukázka fungování komunikace mezi webovou aplikací a nativní částí. WebView vykreslí jednoduché HTML, kde se po dvou sekundách spustí skript, který zavolá funkci `postMessage` s textem. Zavolání funkce WebView odchytí a zavolá funkci uloženou v propu `onMessage`, který zprávu zpracuje a text poslaný z webové aplikace vypíše do informativního okna.

```
import React, { Component } from 'react';
import { View } from 'react-native';
import { WebView } from 'react-native-webview';

export default class App extends Component {
  render() {
    const html = `
      <html>
      <head></head>
      <body>
        <script>
          setTimeout(function () {
            window.ReactNativeWebView.postMessage("Hello!")
          }, 2000)
        </script>
      </body>
      </html>
    `;

    return (
      <View style={{ flex: 1 }}>
        <WebView
          source={{ html }}
          onMessage={(event) => {
            alert(event.nativeEvent.data);
          }}
        />
      </View>
    );
  }
}
```

Listing 2.8: Ukázka komunikace mezi webovou částí a nativní částí pomocí funkce postMessage a propu onMessage.[29]

2.2.2.2 injectJavascript

V případě propu `injectedJavaScript` viz 2.2.1.4 jsme předali WebView komponentě skript, kerý se provedl po načtení obsahu nebo před načtením obsahu v případě propu `injectedJavaScriptBeforeContentLoaded` pouze jednou. Tato metoda nám umožňuje poslat do WebView skript kdykoliv je to zapotřebí. Je to ideální způsob jak posílat data do webové aplikace z nativní části po načtení WebView obsahu. Funkce přijímá jediný argument, kterým je skript, jehož datový typ musí být *string*. Stejně jak tomu bylo u skriptu, posílaném v propu `injectedJavaScript`, musí být poslední hodnotou v předávaném stringu `true`, kvůli předcházení nežádoucím chybám.

Níže lze vidět kus kódu, který zobrazí webovou stránku ve WebView a po třech sekundách od zobrazení zavolá metodu `injectJavaScript` a změní barvu pozadí WebView obsahu na modrou.

```
import React, { Component } from 'react';
import { View } from 'react-native';
import { WebView } from 'react-native-webview';

export default class App extends Component {
  render() {
    const run = `
      document.body.style.backgroundColor = 'blue';
      true;
    `;

    setTimeout(() => {
      this.webref.injectJavaScript(run);
    }, 3000);

    return (
      <View style={{ flex: 1 }}>
        <WebView
          ref={(r) => (this.webref = r)}
          source={{
            uri:
              'https://github.com/react-native-community/react-native-webview',
          }}
        />
      </View>
    );
  }
}
```

Listing 2.9: Ukázka komunikace mezi webovou částí a nativní částí pomocí metody injectJavaScript[30]

2.2.2.3 reload

Tato metoda způsobí přenačtení aktuální stránky. Je nutné podotknout, že pouze aktualizuje obsah WebView podle zadané URL, nezavolá znovu metody `injectedJavaScript` nebo `injectedJavaScriptBeforeContentLoaded`.

2.2.2.4 stopLoading

V případě, že dojde na straně obsahu zobrazovaného ve WebView k chybě, která není správně ošetřena, může se webová aplikace zacyklit a nedojde nikdy k úplnému načtení. Stav načítání můžeme ve WebView sledovat například pomocí propu `onLoad` viz 2.2.1.8 a v případě, že do určité doby nedojde k úplnému načtení, vyvolat zastavení načítání manuálně. K tomu může posloužit metoda `stopLoading`.

2.2.2.5 goBack

Jelikož ve WebView není žádné tlačítko zpět, jako je tomu v klasickém prohlížeči, je zde metoda `goBack`. Tato metoda je volána z nativní části a může být použita například při detekci chyby, kterou indikuje prop `onError` viz 2.2.1.9. Dále mohou být tlačítka pro navigaci zpět a dále viz 2.2.2.6 implementována do WebView a k navigaci poslouží přesně tyto metody.

2.2.2.6 goForward

Tato metoda vyvolá načtení stránky, která je v historii WebView před tou aktuální. V případě, že žádná taková stránka není, zachová se stejně jako akce dopředu u klasického prohlížeče a to tak, že se nestane nic.

2.2.2.7 clearCache

Metoda, která smaže cache pro WebView obsah. Tato metoda je dostupná pouze pro operační systém Android. V případě iOS je zapotřebí vyřešit smazání cache jiným způsobem, například poslat potřebná data webové aplikaci pomocí funkce `injectJavaScript`.

2.2.2.8 clearHistory

Indikace pro WebView aby smazalo pole, ve kterém si drží informace o historii prohlížení pro funkce `goBack` a `goForward`. Opět, jako u metody `clearCache`, je tato metoda funkční pouze pro Android a v případě iOS je to třeba řešit jinak.

2.3 Důležité JavaScriptové metody a proměnné pro komunikaci s WebView

Nebudu popisovat JavaScript jako jazyk, jelikož přímo tím se práce nezabývá, vyberu ale jeho metody a proměnné, které jsou používány pro komunikaci s WebView. Některé prvky už byly v práci zmíněny, jako například proměnné `window` nebo `document`. Nebyly ale nijak blíže specifikovány a jsou zde i další prvky JavaScriptu, které jsou pro tuto práci velmi důležité.

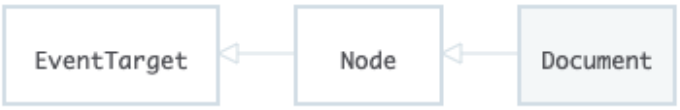
2.3.1 Proměnná window

Proměnná `window` je globální objekt, který v sobě uchovává vlastnosti týkající se aktuálního dokumentu DOM, což je v překladu vlastně to, co vidíme na záložce v prohlížeči. Díky tomu, že je tento objekt globální, je možné k němu přistupovat libovolně z jakékoliv části aplikace. Když je proměnná vytvořena bez přiřazení klíčového slova `var`, `let` nebo `const`, stává se z ní automaticky globální proměnná a stává se součástí proměnné `window`. Objekt

window má v sobě mnoho vlastností jako jsou konstruktory, proměnné nebo metody. V prohlížečích, které mají záložky, má každá záložka svoji vlastní instanci proměnné window. K vlastnostem proměnné window nemusíme přistupovat přes referenci této proměnné, ale můžeme přistupovat k vlastnosti přímo. Například k obecně známé vlastnosti window.console, kterou používáme nejčastěji pro vypisování kontrolních textů do konzole, přistupujeme napřímo jako console.log() .[31]

2.3.2 Proměnná document

Tato proměnná je jednou z vlastností proměnné window a tím pádem i globální proměnnou, ke které nemusíme přistupovat pomocí instance window. Proměnná document je rozhraní, které reprezentuje jakoukoliv webovou stránku načtenou v prohlížeči a slouží jako vstupní bod k obsahu webové stránky, která je DOM strom. DOM strom zahrnuje HTML elementy jako $< body >$, $< table >$ a mnoho dalších. Poskytuje globální funkcionality pro celý dokument, jako například jak získat URL aktuální webové stránky nebo vytvářet nové elementy v dokumentu. DOM je cross-platformní a jazykově nezávislá cesta, jak manipulovat s HTML a XML dokumenty.[32]



Obrázek 2.3: Struktura document elementu

2.3.3 EventTarget a jeho metody

EventTarget je rozhraní DOM viz obrázek 2.3, které je implementováno objekty, jenž mohou dostávat události (eventy) a mít pro ně posluchače (listenery). Document a window jsou nejběžnějšími event targety, ale existují i jiné. Například XMLHttpRequest, AudioNode, AudioContext a další. Ty nebudou pro tuto práci zapotřebí a proto se jimi dále nebudu zabývat. Mnoho event targetů (včetně window a document) podporují nastavování obsluh událostí jako je vytváření, mazání nebo volání událostí.

EventTarget má pro obsluhu událostí 3 metody. Tyto metody jsou

- addEventListener,
- removeEventListener,
- dispatchEventListener.

Pomocí těchto metod, lze nastavit obsluhu událostí pro globální proměnnou `window`, které může poslouchat a čekat na různé typy úkolů, které může odesílat například WebView nebo nativní část aplikace.

2.3.3.1 addEventListener

Tato metoda slouží k tomu aby nastavila funkci, které se zavolá kdykoliv detekuje doručení specifikované události k cíli (targetu). Základními targety je například proměnná `window` nebo `document`. Funguje to tak, že se funkce nebo objekt, který implementuje EventListener, přidá do seznamu event listenerů specifikovaného podle typu události v EventTargetu, nad kterým byla funkce zavolána. Do takového seznamu může být přidán jakýkoliv počet obsluh událostí bez toho, aniž by se vzájemně překrývaly.

Syntax metody addEventListener

target.addEventListener(type, listener [, options])
target.addEventListener(type, listener [, useCapture])

Parametry

- **type** - String reprezentující typ události, na kterou má poslouchat. Typ může být ze seznamu typu událostí, které jsou definované JavaScriptem nebo může být definován uživatelem.
- **listener** - Objekt, který dostane upozornění o tom, když se objeví událost se specifikovaným typem. Musí to být objekt implementující EventListener rozhraní nebo JavaScriptová funkce. Parametrem této funkce je objekt typu Event, který obsahuje všechny vlastnosti a akce, které lze provádět v JavaScriptu s událostmi.
- **options** - Objekt, který specifikuje informace o posluchači událostí. Tento parametr není povinný. Objekt `options` má následující vlastnosti:
  - **capture** - Logická hodnota označující, že události tohoto typu budou odeslány registrovanému posluchači před odesláním do libovolného jiného EventTarget elementu, který se nachází v DOM stromu pod ním.
  - **once** - Logická hodnota označující, že posluchač by měl být po přidání zavolán maximálně jednou. Pokud je hodnota nastavena na `true`, posluchač je po vyvolání automaticky odebrán.
  - **passive** - Logická hodnota, které když je nastavena na hodnotu `true`, označuje že funkce specifikovaná posluchačem nezavolá preventDefault() funkci.

- **useCapture** - Logická hodnota označující, zda budou události tohoto typu odeslány registrovanému posluchači před odesláním do libovolného EventTarget elementu, který se nachází v stromu DOM pod ním. Události, které probublávají stromem nahoru nespustí posluchače, který má vlastnost `useCapture` nastavenou na hodnotu `true`. Tato vlastnost není povinná, ale dříve byla a proto se doporučuje tuto vlastnost nastavovat z důvodu prevence chyb v starších prohlížečích.[33]

2.3.3.2 removeEventListener

Metoda **`EventTarget.removeEventListener()`** odstraní z EventTargetu posluchače události, který byl registrovaný pomocí metody **`EventTarget.addEventListener()`**. Posluchač události který má být odstraněn, je identifikován pomocí kombinace typu události, funkce posluchače a různých volitelných možností, které mohou ovlivnit proces párování.

Syntax metody removeEventListener

target.removeEventListener(type, listener [, options])

target.removeEventListener(type, listener [, useCapture])

Parametry

- **type** - String určující typ události, ze které má odstranit posluchače.
- **listener** - Funkce posluchače, která má být odstraněna za EventTarget elementu.
- **options** - Objekt možností specifikujících charakteristiky o posluchači. Tento parametr není povinný. Vlastnosti objektu jsou:
  - **capture** - Logická hodnota označující, že události tohoto typu budou odeslány registrovanému posluchači před odesláním do libovolného jiného EventTarget elementu, který se nachází v DOM stromu pod ním.
- **useCapture** - Určuje, zda je EventListener, který má být odebrán, registrován jako zachycovací posluchač či nikoliv. Pokud tento parametr chybí, předpokládá se výchozí hodnota `false`.[34]

2.3.3.3 dispatchEvent

Metoda, která odešle událost na zadaném EventTargetu vyvoláním (synchronně) ovlivněných posluchačů událostí v příslušném pořadí. Normální pravidla pro zpracování událostí (včetně fáze zachycování a volitelného probublávání) se vztahují také na události odeslané manuálně pomocí funkce `dispatchEvent()`.

Syntax metody dispatchEvent

cancelled = !target.dispatchEvent(event)

Parametry

- **event** - Objekt typu Event co se má odeslat. Event objekt reprezentuje jakoukoliv událost, která je součástí DOM. Event obsahuje mnoho vlastností a metod, které jsou základní pro všechny události.

Návratová hodnota metody

Návratová hodnota je `false` pokud je událost zrušitelná a alespoň jeden z obslužných programů událostí, který obdržel událost, zavolal metodu `Event.preventDefault()` .

Metoda vyhodí chybu `UNSPECIFIED_EVENT_TYPE_ERR` v případě, že typ události nebyl specifikován inicializací před tím, než byla metoda zavolána nebo je typ události `null` nebo prázdný string. [35]

2.4 Vybrané principy z jazyka React JS

React JS je frameworkem JavaScriptu a proto v něm lze používat veškeré zabudované metody a proměnné, které byly zmiňovány výše. React ale posouvá JavaScript na úplně jinou úroveň a spolu s tím přichází s novými funkcemi, jenž usnadňují práci programátorům a zpřehledňují kód. Stejně jako JavaScript, tak i React je velmi rozsáhlý jazyk a nebudu proto popisovat všechny jeho možnosti. Zaměřím se pouze na metody důležité pro tuto práci, tedy jak nejlépe využít možností jednotlivých jazyků k dosáhnutí uživatelsky přívětivé mobilní aplikace.

V sekci WebView jsem zmiňoval metody, kterými můžeme uvědomit webovou část aplikace o tom, že posíláme nějaká data nebo že si nějaké data žádáme. V JavaScript sekci jsem představil možnosti jak těmto požadavkům naslouchat a jak jiné požadavky posílat zpátky. Jelikož je front-end aplikace na kterou budeme poznatky z analytické části aplikovat napsaný v jazyce React JS, je nutné se tohoto jazyka držet a použít správné prvky k zpracování a uložení získaných dat z nativní části aplikace. Jelikož jsou všechny části webové aplikace rozděleny na komponenty, umožňuje nám React sledovat stav vykreslení jednotlivých komponent a na základě toho můžeme vykonávat určité akce. K tomu, abychom mohli přistupovat k instanci WebView v nativní části, musíme umět vytvořit správnou referenci. Proměnné můžeme v Reactu ukládat do lokálního stavu komponenty a podle toho, jak se hodnota proměnné bude

měnit, vyvolávat její znovuvykreslování a aktualizaci. Všechny tyto metody a operace můžeme v Reactu vyvolávat pomocí tak zvaných React Hooks.

2.4.1 React Hooks

S React Hooks přišli vývojáři z Facebook po pěti letech používání jazyka React jako s řešením určitých problémů, které za tu dobu používání jazyka vyplavaly na povrch. V Reactu existují dva typy komponent kterými jsou třídní komponenty a funkcionální komponenty. Funkcionální komponenty sloužily pro jednoduché přímočaré prvky, které měly vždy jeden jasný úkol, byly znovupoužitelné a dostávaly informace od svých rodičů. Třídní komponenty sloužily pro uchovávání stavu, mohly se napojovat na různá uložiště a sloužily jako HOC (High Order Components) komponenty. Třídní komponenty v sobě měly a pořád mají, zabudované metody pro sledování tak zvaného životního cyklu komponenty. Životní cyklus komponenty se dělí na dvě části a to na připojení nebo přihlášení komponenty do DOMu a odpojení nebo odhlášení komponenty z DOMu. Každá metoda se zavolá v určitou chvíli životního cyklu komponenty a programátoři mohou provádět potřebné akce v závislosti na stavu. Toto však nebylo možné sledovat u funkcionálních komponent a také nemohly uchovávat vlastní lokální stav. Řešením tohoto problému jsou React Hooks.

Hooky jsou funkce, které umožňují používat React stav a sledování životního cyklu pro funkcionální komponenty. Hooky nefungují v třídních komponentách, umožňují používat React bez třídních komponent. React přichází s několika zabudovanými hooky, jako je třeba `useState` nebo `useEffect`, je možné ale vytvářet i vlastní hooky.

Pravidla hooků

Hooky jsou JavaScriptové funkce, existují ale jistá pravidla, na které musí vývojáři při jejich používání dbát.

- Hooky se volají pouze na nejvyšší úrovni komponenty. To znamená, že není možné volat hooky uvnitř smyček, podmínek nebo vnořených funkcí.
- Hooky je možné volat pouze z funkcionálních komponent nebo z jiných hooků. Není možné volat hooky v obyčejných JavaScriptových funkcích.

2.4.1.1 useState

Tento hook je jedním z hooků, které jsou součástí React knihovny. Používá se k spravování lokálního stavu v funkcionálních komponentách. Přijímá pouze jeden argument a tím je počáteční stav. Návratovou hodnotou je pole o velikosti dvou prvků, kde prvním prvkem pole je aktuální stav a druhým prvkem

je funkce, která aktualizuje starý stav za nový, jenž je jejím argumentem. K uložení těchto prvků do uživatelem definovaných proměnných se používá tak zvaná destrukce pole. Je to akce, pomocí které lze prvky z pole jednoduše přiřazovat novým proměnným za použití hranatých závorek.[36]

```
const foo = ['jedna', 'dva', 'tri'];

const [a,b, c] = foo;
console.log(a); // "jedna"
console.log(b); // "dva"
console.log(c); // "tri"
```

Listing 2.10: Ukázka inicializace proměnných pomocí destrukce pole

```
  import React, { useState } from 'react';

    function Example() {
    const [count, setCount] = useState(0);

    return (
      <div>
        <p>You clicked {count} times</p>
        <button onClick={() => setCount(count + 1)}>
         Click me
         </button>
      </div>
    );
  }
```

Listing 2.11: Ukázka použití useState hooku pro jednoduchý čítač

2.4.1.2 useEffect

Tento hook umožňuje pracovat s životním cyklem komponenty v funkcionálních komponentách. Je ekvivalentem třídních metod pro zpracování životního cyklu komponenty, jako jsou `componentDidMount`, `componentDidUpdate` a `componentWillUnmount`. Hook má jeden povinný a jeden volitelný argument.

- **callback** - Je povinným a prvním argumentem hooku. Hodnotou je funkce, které se provede pokaždé, když dojde ke změně na obrazovce. Volání pak také závisí na druhém argumentu hooku. Tělem funkce je uživatelem definovaná logika. Jestli je potřeba, aby se provedla akce podobná funkci `componentWillUnmount` z třídní komponenty, musí funkce callback vrátit funkci. Funkce, která bude v návratové hodnotě funkce callback, se provede po odpojení komponenty z DOM stromu.

- **dependencies** - Je volitelný argument, jehož hodnota je pole závislostí. Hook `useEffect` zavolá `callback` funkci, pouze pokud zaznamená změnu v poli závislostí mezi jednotlivými vykresleními komponenty. Podle hodnoty tohoto argumentu rozlišujeme tři různé možnosti volání `callback` funkce.
  - Když je argument dependencies nedefinovaný, dojde k zavolání `callback` funkce po každém vykreslení komponenty, což znamená při každé změně stavu.
  - Když je hodnotou argumentu prázdné pole, dojde k zavolání funkce `callback` při prvním vykreslení komponenty.
  - Když je hodnotou argumentu pole závislostí, kde závislost je stavová nebo prop proměnná, dojde k zavolání funkce `callback` pokaždé, když dojde ke změně nějaké závislosti ze zadaného pole.[37]

Definice tohoto hooku může být těžká pro představu a proto ukážu jeho použití, pro dosáhnutí ekvivalentního efektu jako s funkcemi, které slouží k zpracování životního stavu komponenty v třídním přístupu.

Přidání komponenty do DOM (componentDidMount)

Jsou případy, kdy potřebujeme aby se nějaké akce provedla pouze při prvním vykreslení komponenty. Abychom toho dosáhli, zavoláme hook, kde prvním argumentem bude funkce, jenž akci provede a druhým argumentem bude prázdné pole.

```
1 import { useEffect } from 'react';
2
3 function Example({ jmeno }) {
4   useEffect(() => {
5     console.log(`Ahoj ${jmeno}`);
6   }, []);
7 }
```

Listing 2.12: Ukázka použití useEffect hooku jako ekvivalenta k componentDidMount

Komponenta vypíše do konzole hodnotu proměnné *jmeno* pouze pro první vykreslení komponenty. V případě, že dojde ke změně propu, nic se nevypíše.

Aktualizace komponenty při změně stavu (componentDidUpdate)

Může se stát, že potřebujeme, aby se nějaké akce provedla pouze při změně určitého stavu nebo stavů. Toho docílíme tak, že zavoláme hook, kde prvním argumentem bude funkce jenž akci provede a druhým argumentem bude pole závislostí, při jejichž změně dojde k zavolání funkce v callback argumentu.

```
import { useEffect } from 'react';

function Example({ jmeno }) {
  useEffect(() => {
    console.log(`Ahoj ${jmeno}`);
  }, [jmeno]);
}
```

Listing 2.13: Ukázka použití useEffect hooku jako ekvivalenta k componentDidUpdate

Komponenta vypíše hodnotu proměnné *jmeno* pokaždé, když dojde k její změně.

Odebrání komponenty z DOM (componentWillUnmount)

Tato funkcionalita je požadována především v případě, že potřebujeme za sebou tak zvaně uklidit nadělaný nepořádek. Jelikož se JavaScript stará o uvolňování paměti sám, jedná se především o akce jako odebírání přidaných listenerů nebo vypínání nějakých nekonečných smyček. Pokud chceme, aby došlo k uklizení nepořádku pouze při finálním odstranění komponenty z DOMu, tak definice useEffect hooku bude stejná jako v prvním případě, s tím rozdílem, že callback funkce bude vracet funkci co provede úklid.

```
import { useEffect } from 'react';

function Example() {
  function resizeFnc() {
    console.log("zmena velikosti okna");
  }
  useEffect(() => {
    window.addEventListener('resize', resizeFnc, true);

    return () => {
      window.removeEventListener('resize', resizeFnc, true);
    }
  }, []);
}
```

Listing 2.14: Ukázka použití useEffect hooku jako ekvivalenta k componentWillUnmount

Při prvním vykreslení komponenty dojde k vytvoření posluchače události, který zavolá funkci při změně velikosti okna prohlížeče. Při odpojení komponenty z DOM dojde k odstranění posluchače pomocí funkce z návratové hodnoty.

2.4.1.3 useRef

Tento hook je funkce, jenž vrací proměnlivý objekt typu reference. Má jeden nepovinný argument, kterým může být hodnota jakéhokoliv typu. Vlast-

nost objektu s názvem `current` je pak inicializována na hodnotu daného argumentu. Vrácený objekt bude existovat do té doby než bude komponenta odebrána z DOMu. Tento hook má v reactu dvě hlavní využití.

Přístup k DOM uzlům nebo React elementům

V klasickém JavaScriptu se přistupuje k DOM uzlům nebo ke klasickým HTML elementům pomocí proměnných `window` nebo `document`. To je ale JavaScriptový přístup, React místo toho využívá tento hook. Objekt, který hook vrátí, se naváže na libovolný HTML element v render funkci a lze pak přistupovat k jeho vlastnostem. Hook vrátí při každém vykreslení novou referenci.

```
import React, { useRef } from "react";

const CustomTextInput = () => {
  const textInput = useRef();

  focusTextInput = () => textInput.current.focus();

  return (
    <>
      <input type="text" ref={textInput} />
      <button onClick={focusTextInput}>Focus the text input</
    button>
    </>
  );
}
```

Listing 2.15: Použití useRef hooku pro vytvoření reference k HTML elementu

2.4.1.4 Udržování proměnlivé proměnné

Mezi jednotlivým vykreslováním komponenty můžeme v Reactu uchovávat data dvěma způsoby. Jedním je uchování lokálního stavu pomocí hooku `useState`, kde každá změna stavu způsobí znovuvykreslení komponenty a druhým způsobem je uchování stavu v ref objektu. Ref objekt je ekvivalent instančních proměnných v třídních komponentách. Změna instanční proměnné nevyvolá znovuvykreslení komponenty. K hodnotě uložené v referenci se pak přistupuje přes její vlastnost `current`. Změna této proměnné může způsobit vedlejší účinky a proto je zapotřebí provádět změnu proměnné vždy uvnitř `useEffect` hooku.[38]

```
import React, { useRef } from "react";

const RenderCounter = () => {
  const counter = useRef(0);

  useEffect(() => {
    // provede si pri kazdem vykresleni komponenty
    counter.current = counter.current + 1;
```

```
9   });
10
11  return (
12    <h1>{`Komponenta byla vykreslena ${counter} krat`}</h1>
13  );
14 };
```

Listing 2.16: Použití useRef hooku jako instační proměnou

2.5 Prvky použité z React Native pro správné fungování WebView

Jelikož React Native vychází z React JS lze i v React Native používat hooky a všechny přístupy, které jsem zmiňoval v analýze použitých prvků z jazyka React JS. V této práci je hlavní funkcí React Native to, že slouží jako kontejner, do kterého WebView vykresluje svůj obsah. Ve výsledné aplikaci bude celá část přihlašování a notifikací implementována pomocí React Native, ale v tomto případě je použita pouze jedna komponenta, které nějakým způsobem upravuje zobrazení WebView. Jedná se o komponentu, jenž má na starosti zobrazení delšího obsahu než je velikost displeje zařízení. V klasickém prohlížeči to funguje samo od sebe a stránka může být nekonečně dlouhá. V mobilním přístupu musíme obsah delší než se vejde na displej zařízení zabalit do komponenty, která se stará o postupné načítání dalších dat. Tato komponenta nám také umožňuje aktualizovat zobrazovaná data. Jedná se o komponentu ScrollView.

2.5.1 ScrollView

Jedná se o komponentu, která obaluje platformu ScrollView a zároveň poskytuje integraci se systémem "odpovědi"zamykání dotykem. Tato komponenta dědí vlastnosti komponenty View. Je třeba myslet na to, že ScrollView komponenta musí mít pevně danou výšku kontejneru aby správně fungovala, jelikož obsahuje většinou potomka, který nemá předem známou délku obsahu a tím pádem i výšku zobrazení. K tomu aby se výška ScrollView komponenty svázala spolu s velikostí obrazovky je zapotřebí nastavit přímo pevnou velikost samotné komponentě, což se nedoporučuje a nebo se ujistit, že výška obsahu potomka bude odpovídat velikosti zařízení. Tato komponenta také umožňuje kontrolovat akci, která je ve většině mobilních aplikacích brána jako aktualizace obsahu stažením obrazovky gestem ze shora dolů.

V následujících sekcích zmíním vlastnosti, které budu později v části kde se budu zabývat vývojem používat.

2.5.1.1 onLayout

Tento prop je jednou z vlastností, kterou komponenta podědila od komponenty View. Jedna se o velmi jednoduchou metodu, která se zavolá pokaždé když dojde ke změně rozložení komponenty která na sobě danou metodu má. Hodnotou propu je funkce s jedním argumentem. Argument je typu `LayoutChangeEvent`, což je objekt, který vypadá následovně.

```
1  { nativeEvent: { layout: { x, y, width, height }}}
```

Tato událost se aktivuje okamžitě po výpočtu nového rozložení. Je nutné si ale dát pozor na to, že nově vypočítané rozložení se nemusí projevit na obrazovce v době přijetí události, zejména pokud probíhá animace nastavení rozložení. V těle funkce, které je hodnotou tohoto propu, pak lze provádět akce spojené s novou velikostí a vzhledem rozložení. [39]

2.5.1.2 refreshControl

Hodnotou tohoto propu je komponenta `RefreshControl`, používaná k zajištění funkce pull-to-refresh (stáhni a aktualizuj) pro komponentu `ScrollView`. Tato funkce je dostupná pouze pro vertikální typ `ScrollView` (prop `horizontal` musí být nastaven na *false*).

2.5.1.3 RefreshControl

Tato komponenta se používá uvnitř komponenty `ScrollView` pro přidání pull-to-refresh funkcionality. V případě, že je hodnota vertikální pozice `ScrollView` rovna nule, potažením dolů prstem na obrazovce dojde k vyvolání události `onRefresh`.[40]

onRefresh

Hodnotou tohotu propu je funkce s jedním argument. Argument je typu `NativeScrollEvent` a obsahuje všechny důležité informace o vlastnostech rozložení.

enabled

Hodnotou tohoto propu je logická hodnota, která indikuje zda se může akce v `onRefresh` propu provést nebo ne.

refreshing

Hodnotou je logická hodnota, indikující zda má načítací kolečko zůstat zobrazené nebo ne. Manipulovat s touto hodnotou můžeme například při načítání dat ze serveru, abychom dali uživateli přesně vědět, kdy skončilo načítaní dat.

Kapitola 3

Analýza podnikové aplikace FLOWIO

V předešlých kapitolách jsem se věnoval především pojmům a technologiím, kterými se práce zabývá. V této kapitole představím aplikaci a vymezim případy užití, na které budu poznatky z minulých kapitol aplikovat.

3.1 Co je FLOWIO

FLOWIO je podniková, multiplatformí aplikace, která má za úkol sjednotit veškerou pracovní agendu do jednoho nástroje. Umožňuje sjednotit více podnikových systémů přehledně do jednoho prostředí. Aplikace dbá především na jednoduchost používání, nehledě na to, na jakém zařízení je používána. Uživatelé mohou v rámci aplikace nastavovat rychle jakýkoliv firemní proces z jakéhokoliv přiadného systému a mohou to provádět sami. Většina podnikových systémů jsou obrovské informační systémy, kde se k nastavování procesů ve firmě využívá služeb externích společností, jelikož nastavování může být zdlouhavé a složité, ve FLOWIO toho už nebude zapotřebí, jelikož díky jednoduchosti aplikace to zvládne každý zaměstnanec sám.

3.2 Vymezení případů užití, kde bude potřeba upravit webovou část, aby správně komunikovala s nativní částí a WebView

Existuje tedy funkční webová aplikace FLOWIO. V aplikaci existují dva typy účtů, administrátorský a klasický uživatelský. Přihlášení do aplikace je pro oba dva účty stejné, jediný rozdíl je pak v přístupech k jednotlivým funkcionalitám aplikace. Administrátor se stará o nastavování a úpravu procesů. Klasický uživatel si pak pouze nastavuje a upravuje podle vlatních potřeb

svojí agendu v rámci různých datových zdrojů. V aplikaci si uživatelé vytváří tak zvané šablony, což jsou tabulky, vytvořené ze zvolených datových zdrojů jako jsou databáze a tabulky v nich. Dostupné datové zdroje a tabulky definuje administrátor v administrátorské části. Aplikace je koncipovaná tak, aby bylo možné se připojit na libovolný datový zdroj, který uchovává informace potřebé k plynulému chodu firmy. Šablony znázorňují nejrůznější procesy ve firmě, jako je třeba schvalování docházky, žádosti o dovolené, výpisy firemních cest, žádosti o vybavení a podobné. Nad těmito tabulkami je možno provádět nejrůznější akce, které nad těmito šablonami definuje administrátor v administrační části. V aplikaci funguje také notifikační systém, který uživatele informuje o jakékoliv změně v procesu, který uživatel založil nebo je jakýmkoliv způsobem jeho součástí.

Hlavním cílem aplikace je především urychlení a zjednodušení firemních procesů pro zaměstnance. Z toho důvodu je zcela jasné, že je zapotřebí mít k webové aplikaci i mobilní, pro větší flexibilitu používání. Jak už jsem popisoval výše, tak vzhledem k tomu, jaké byly kladeny nároky na vývoj mobilní aplikace, jsme museli zvolit altnernativu ve formě využití jazyka React Native, pro vykreslení webové aplikace jako obsah WebView komponenty. Jelikož je FLOWIO komplexní aplikace, bylo zapotřebí přizpůsobit webovou aplikaci tak, aby zvládala komunikovat s WebView a dala se obalit nativním kódem. V této sekci proto vymezím ty části webové aplikace, které bylo zapotřebí upravit, aby se dosáhlo požadovaného výsledku ve formě mobilní aplikace.

3.2.1 Přihlašování, respektive odhlašování uživatele do, respektive z aplikace

V webové aplikaci funguje přihlašování do FLOWIO klasicky, jako u všech jiných webových aplikací. Pokuď uživatel má vytvořený účet v aplikaci, zadá přihlašovací jméno a heslo, v případě že tyto údaje napsal správně, tak je přihlášen do aplikace a na server se pošlou cookies o tom, aby zůstal přihlášený i když odejde ze stránky nebo vypne prohlížeč. V případě, že zadal špatné přihlašovací údaje, zobrazí se chybová hláška. Když uživatel nemá vytvořený uživatelský účet, odkazem se dostane na stránku pro registraci. Po vyplnění všech políček se uživatel registruje a na zadaný email se odešle odkaz, pro aktivaci účtu. Po kliknutí na odkaz v emailu je uživatel automaticky přihlášen do aplikace. Podobný proces funguje i pro změnu nebo nastavení nového hesla k již existujícímu účtu.

3.2.1.1 Nadstavba pro mobilní aplikaci

Jedná se ale o přihlášení do webové aplikace. Mohli bychom ve WebView vykreslovat už samotnou přihlašovací stránku, ale pak bychom se připravit o možnost přihlašování do aplikace možnostmi, které umožňuje nativní přístup

jako je faceID, touchID a nebo kód. Kromě toho, musí přihlašování do mobilní verze aplikace FLOWIO zajišťovat jednu rozdílnou funkcionalitu navíc. V případě webové aplikace, je možno přistupovat do různých prostředích aplikace FLOWIO. Mám tím na mysli prostředí produkční, testovací nebo vývojové. Společnost, jenž bude využívat FLOWIO, může tyto různá prostředí používat ke kontrole nastavování procesů v aplikaci před tím, než jí umožní zaměstnancům používat. V případě webové aplikace se jedná o rozdíl v URL adrese, podle které se přistupuje do back-end části aplikace pro správná data. Jelikož ve WebView není přítomna část s URL adresou, je zapotřebí umožnit uživatelům přepínat mezi prostředími jiným způsobem, než změnou URL adresy. Proto bylo nutné zvolit jiný přístup a přidat novou funkcionalitu do mobilní aplikace. Proto v případě, že uživatel není přihlášen v mobilní aplikaci, bude první obrazovkou výběr prostředí nebo serveru, na kterém bude dostupné API pro získávání dat v aplikaci. Samotné přihlášení komunikuje s zadaným API a ukládá informace o přihlášení, proto je nutné, aby stránka s výběrem prostředí byla jako první.

3.2.1.2 Samotné přihlášení a odhlášení

Jelikož chceme v mobilní aplikaci při přihlášení využívat nativních prvků zařízení, je nutné aby i stránka pro přihlášení do aplikace, registraci i změnu hesla byla vytvořena nově za použití nativního jazyka. Logika i vzhled stránek zůstanou indetické, jako v případě webové aplikace a navíc bude třeba přidat tlačítko, pro přechod na obrazovku s výběrem prostředí. Po prvním přihlášení, před vpuštěním uživatele do aplikace, vyskočí okénko s otázkou, jestli uživatel chce uložit přihlašovací údaje do používaného zařízení a jestli pro příště zvolí pro přihlášení jeden ze způsobů nativní autentizace jako je face ID, touch ID nebo kód. Při každém dalším přihlášení pak dojde mnohem rychleji než při ručním zadávání přihlašovacích údajů.

Tlačítko odhlášení z aplikace je pak stejné jako ve webové aplikaci s tím rozdílem, že provede jinou akci. V webové aplikaci po stisknutí tlačítka pro odhlášení se pošle žádost na server a uživatel je odhlášen. Webová aplikace ale pozná, jestli je zobrazována formou WebView. V případě že tomu tak je, pošle nejdřív zprávu WebView o tom, že chce odhlásit uživatele. WebView zprávu zpracuje a odhlášení provede až nativní aplikace. Pro lepší představu lze vidět na sekvenčním diagramu na obrázku 3.1.

3.2.2 Zpracování notifikací

Notifikace jsou další částí webové aplikace, kterou bude potřeba upravit, pro správné fungování v kombinaci s WebView a nativní částí. Ve webové aplikaci fungujou notifikace klasicky, jako u všech jiných aplikací, kde jsou notifikace přítomny. V hlavičce aplikace je ikonka zvonečku, s počtem nezobrazených notifikací. Po kliku na tento zvoneček dojde k zobrazení modálního okna,

kde si uživatel může zobrazit seznam všech nových notifikací. Po kliknutí na libovolnou notifikaci se lze dostat na detail šablony, které se notifikace týká. Po kliknutí na notifikaci dojde k jejímu označení za přečtenou a nadále se nebude oběvovat v seznamu nových notifikací. Notifikace jsou vázány k šablonám. K jedné šabloně může být více notifikací a ve FLOWIO aplikaci existují tři úrovně shlukování notifikací.

- **Notifikace k šabloně** - značí intuitivně, kolik nových událostí přibylo k dané šabloně.
- **Notifikace na modulu** - Moduly slouží k tomu aby shlukovaly šablony do logických celků. Číslo indikující počet notifikací na modulu je součet všech notifikací, které přísluší šablonám jenž jsou součástí daného modulu.
- **Hlavní ukazatel všech notifikací** - Číslo zobrazující se u zvonečku v záhlaví aplikace. Ukazuje celkový počet všech nových notifikací v aplikaci.

Takto by mohly notifikace fungovat i ve WebView, tím by se ale aplikace připravila například o možnost používání push notifikací. Dále je potřeba myslet na novou funkcionalitu pro mobilní verzi aplikace FLOWIO, kterou jsem popisoval v sekci 3.2.1.1, kde se lze do aplikace připojovat nad různými datovými zdroji. Uživatel může mít přidaných najednou vícero datových zdrojů a může mezi nimi přepínat. Je proto velmi důležité, aby měl uživatel přehledně zobrazené všechny notifikace, ze všech přidaných datových zdrojů. Tuto funkcionalitu samotná webová aplikace neumožňuje a proto je zapotřebí tuto část zpracovat v mobilní aplikaci jinak.

Jelikož chceme využívat v mobilní aplikaci push notifikací, je zapotřebí přesunout logiku zpracování notifikací do nativní části kódu. Push notifikace jsou notofikace, které jsou zaregistrovány při prvním přihlášení do aplikace a zobrazují se v liště oznámení na mobilním zařízení. Push notifikace pracují přímo s hardwarem zařízení a proto je nutné používat pro jejich zpracovávání programovací jazyky, které mají odpovídající knihovny pro komunikaci s hardwarem mobilních zařízení. React Native poskytuje takové knihovny a proto dává smysl přesunout práci s notifikacema do mobilní části kódu.

Veškerá komunikace se serverem v rámci notifikací tedy probíhá v nativní části aplikace. Po přihlašení uživatele, se odešle na server dotaz k získání notifikací pro zadaný datový zdroj. Výsledkem je seznam notifikací, jenž se uloží do lokálního uložiště a počet získaných notifikací je přičten k počtu notifikací z jiných datových zdrojů, které má uživatel v rámci mobilní aplikace přidané. Toto celkové číslo nyní nativní část pomocí WebView metody odešle do webové aplikace. Webová část si také zažádala o notifikace pro přihlášeného

uživatel, tato informace ale nebude použita v hlavičce aplikace, kde se indikují všechny nové notifikace, ale budou sloužit pouze jako indikátor notifikací k příslušným šablonám a modulům v aplikaci. Hlavní indikátor všech notifikací totiž nebude záviset na dotazu, který se odeslal na server z webové části, ale bude záležet na informaci, kterou webová část dostane od WebView. Toto číslo totiž bude indikovat počet všech notifikací napříč přidanými datovými zdroji v mobilní aplikaci. Obsluha přehledu všech notifkací bude také zpracována v nativní částí. Tento seznam musí být rozčleněn na části, podle toho, z jakého datového zdroje notifikace pochází. Tlačítko, které je třeba stisknout pro zobrazení seznamu je ale ve webové části a seznam je v nativní části. Opět webová aplikace rozpozná, že se jedná o zobrazení aplikace pomocí WebView a místo otevření klasického modálního okna v webové aplikaci, odešle informaci pomocí WebView metody, že uživatel chce zobrazit seznam všech notifikací. WebView akci zpracuje a nativní část zobrazí seznam notifikací.

Zobrazení notifikací je tedy vyřešené, ale je zapotřebí také zpracovat kliknutí na notifikaci ze zobrazeného seznamu. Seznam notifikací je zobrazován v rámci nativní části aplikace, ale šablona, které se notifikace týká je už součástí webové části. Je proto nutné správně tuto akci ošetřit, aby se při kliku na notifikaci vytvořila URL s detailem notifikace, která se předá WebView komponentě pro vykreslení správného obsahu. Opět je zapotřebí synchronizovat počty notifikací v nativní části s počtem notifikací ve webové části, jelikož po kliku na notifikaci se sníží počet celkových notifikací o jedna. Na stejném principu funguje zobrazení detailu notifikace i při kliku na notifikaci zobrazenou pomocí push notifikace. Používá se stejná logika jako při kliku na notifikaci z aplikace. Push notifikace jsou spíše záležitostí samotného nativního programovacího jazky, proto se jimi nebudu dále zabývat. Vše co bylo výše popsáno lze vidět graficky zobrazeno na sekvenčním diagramu na obrázku 3.2.

3.2.3 Menší úpravy

Notifikace a přihlašování jsou dvě největší úpravy, které bylo potřeba ve webové aplikaci provést. Dále je zde pár menších úprav, kterým bych se chtěl věnovat krátce v této části.

3.2.3.1 Změna serveru z webové aplikace

Jak už jsem zmiňoval výše, tak v mobilní aplikaci přibyla nová funkcionalita pro vyběr datového zdroje neboli serveru, ze kterého bude uživatel získávat data. Proto je zapotřebí umožnit uživateli, aby mohl přidávat nové datové zdroje a přepínat mezi nimi přímo z webové aplikace zobrazené ve WebView. V záhlaví aplikace je možné rozkliknout uživatelské nastavení, kde jsou nejrůznější akce jako je jméno přihlášeného uživatele, možnost odhlásit se, změna hesla a další. Tato část je ideální pro přidání tlačítka, pomocí kterého se uživatel dostane na stránku s výběrem serveru. Tato možnost se zobrazí

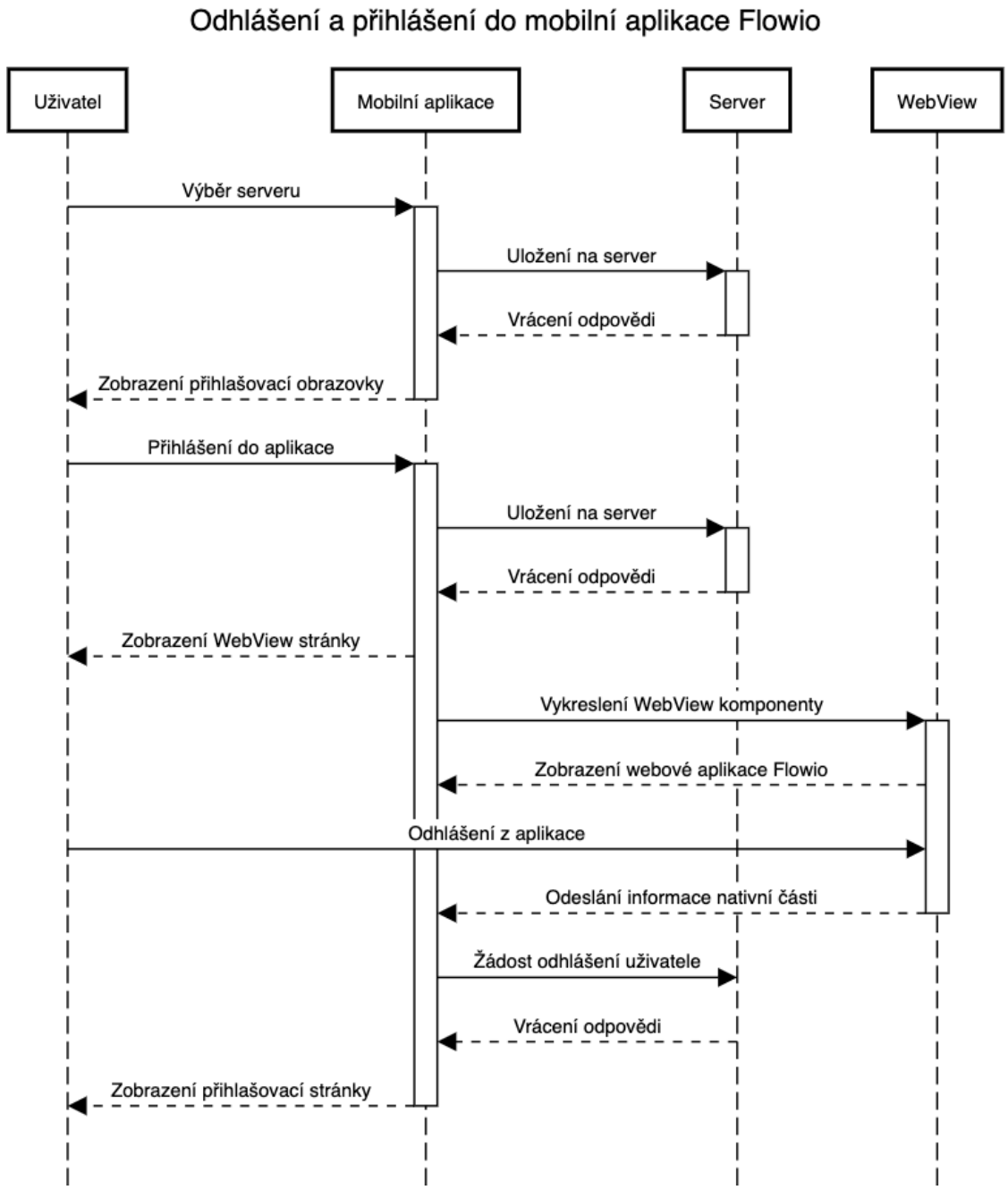
pouze v případě, že je webová stránka zobrazována ve WebView, tudíž pokuď se jedná o mobilní aplikaci. Kliknutím na tlačítku se zavolá WebView metoda, která uvědomí nativní část o tom, že uživatel chce změnit server. Nativní část akci zpracuje a zobrazí nativní stránku s výběrem serveru. Tuto akci lze vidět znázorněnou graficky formou sekvenčního diagramu na obrázku 3.3.

3.2.3.2 Vyčištění lokálního uložiště

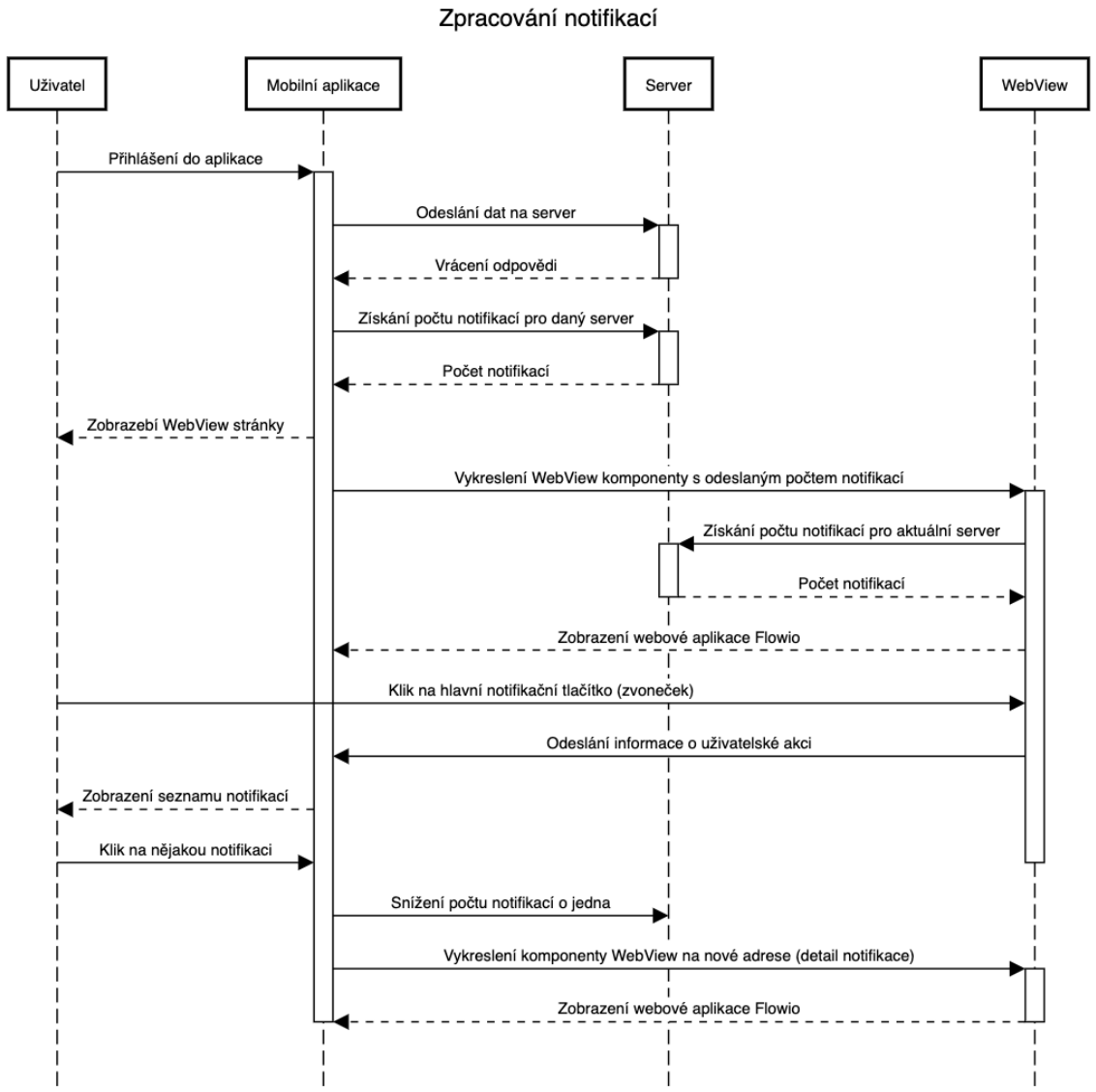
K tomu, aby si mobilní aplikace pamatovala například to, že se uživatel jednou do aplikace přihlásil a nebyl pak při vypnutí aplikace znovu nucen provádět stejnou akci, existuje lokální uložiště zvané `AsyncStorage`. Dalo by se říci, že se jedná o ekvivalenta `LocalStorage` v webovém prohlížeči, akorát pro mobilní zařízení. V případě mobilní aplikace FLOWIO se do tohoto lokálního uložiště kromě přihlašovacích údajů ukládají přidané servery, notifikace a další informace důležité pro správné fungování aplikace. Tyto informace jsou uložené v uložišti do té doby, dokud nejsou smazány programátorem pomocí kódu. Je proto důležité umožnit uživateli vrátit aplikaci do takového stavu, jako když si ji poprvé nainstaloval. Ideálním místem pro tlačítko, jenž provede tuto akci, je opět do uživatelského nastavení. Stejně jako tlačítko pro změnu akce viz 3.2.3.1, je tlačítko pro tuto akci zobrazeno pouze v případě mobilní aplikace. Kliknutím na tlačítko ve webové části se zavolá WebView metoda, které pošle nativní části informaci o tom, že má vyčistit lokální uložiště. Uživatel je poté přesměrován na první stránku aplikace, kde probíhá výběr serveru. Tuto akci lze vidět znázorněnou graficky formou sekvenčního diagramu na obrázku 3.4.

3.2.3.3 Skrytí administrační části

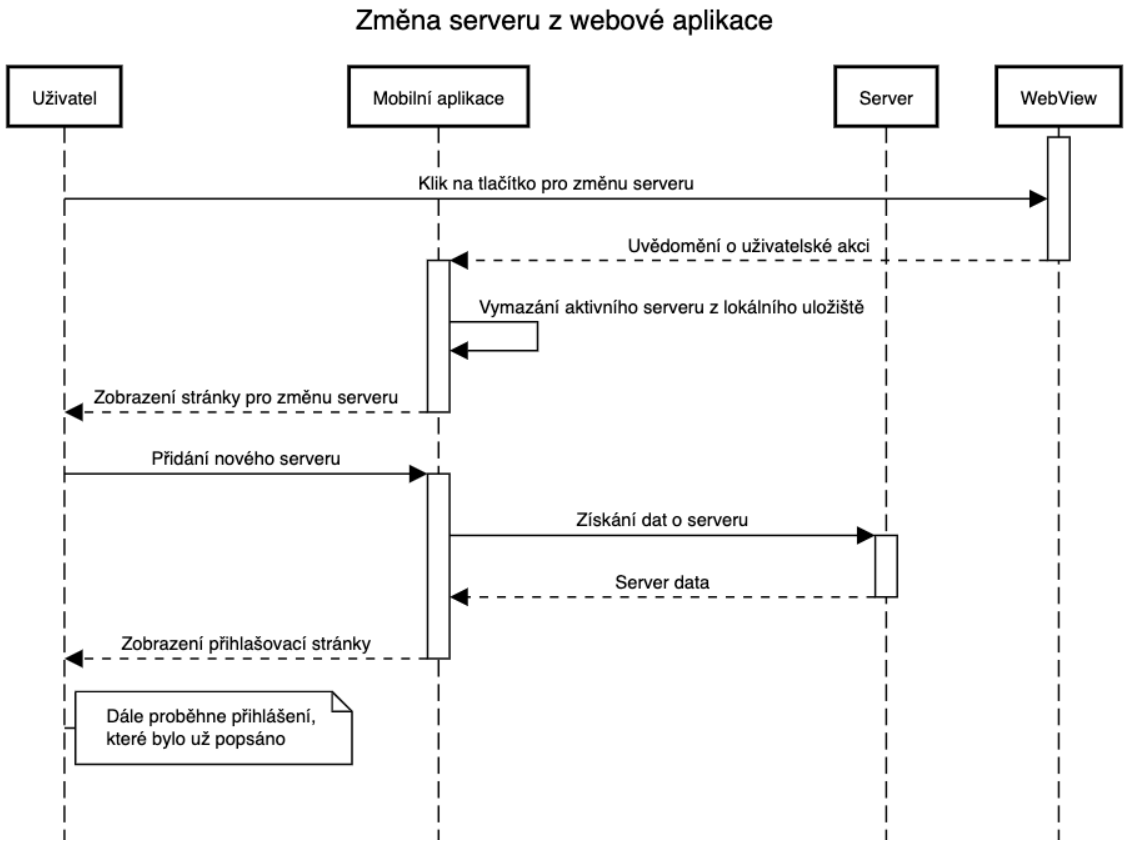
V klasické webové aplikaci FLOWIO má uživatel, s uživatelským účtem typu administrátor, přístup do administrátorské částí, kde probíha veškeré vytváření a nastavování procesů. Ostatní účty do této sekce přístup nemají. Operace, jenž se tam provádí, jsou velmi složité na to, aby je bylo možné provádět na mobilních zařízeních. Proto v případě, že je aplikace zobrazována na mobilním zařízení, nebude přístup do této části umožněn.



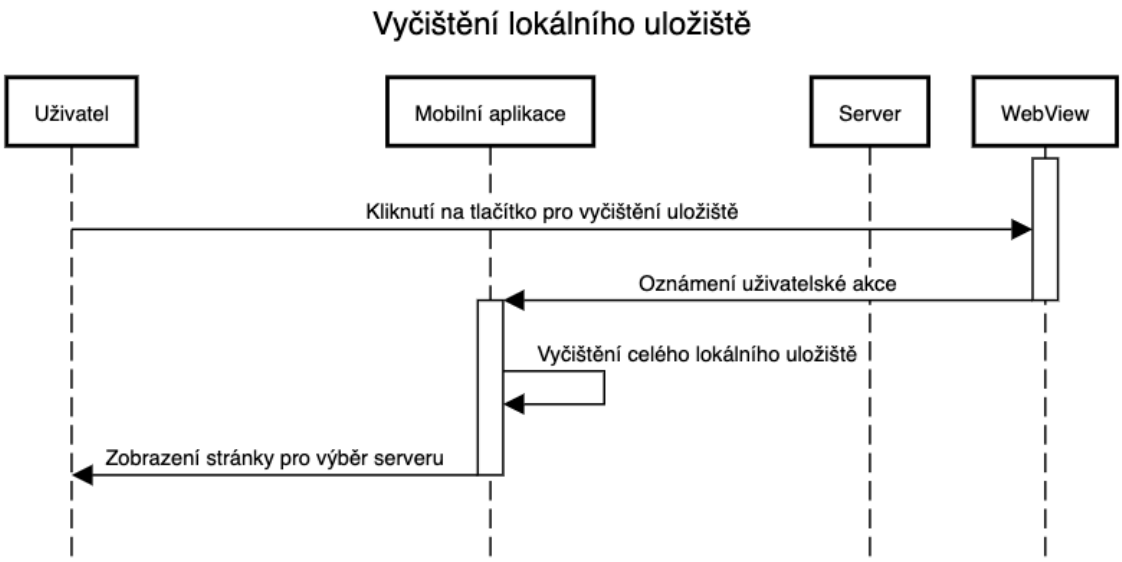
Obrázek 3.1: Sekvenční diagram uživatelského přihlášení a odhlášení v mobilní aplikaci FLOWIO



Obrázek 3.2: Sekvenční diagram zpracování notifikací v mobilní aplikaci FLOWIO



Obrázek 3.3: Sekvenční diagram akce změny serverů



Obrázek 3.4: Sekvenční diagram akce vyčištění lokálního uložiště

Kapitola 4

Aplikování zjištěných poznatků na mobilní aplikaci FLOWIO

V minulých částech jsem analyzoval WebView a prvky z jazyků, které lze v kombinaci s WebView používat. Dále jsem vyčlenil případy, kde bylo zapotřebí použít kombinaci WebView a jeho prvků, s nativními prvky aplikace, aby se webová aplikace chovala mobilní aplikace a uživatelé z jejího použití měli co možná nejlepší uživatelský zážitek. Tato část bude věnována samotné implementaci jednotlivých požadavků, které jsem zmínil v kapitole 3.2. Budu zde popisovat způsob, jakým jsem řešil jednotlivé části v mobilní a webové aplikaci. Každý případ bude mít svoji vlastní sekci a tato sekce bude rozdělena na to, co bylo zapotřebí implementovat v mobilní části a co bylo zapotřebí upravit nebo přidat v té webové. Cílem bylo, co nejméně měnit kód webové aplikace, aby nedocházelo k nekonzistentním verzím. Webová část musí být primárně založena pro zobrazení ve webových prohlížečích a pouze se přizpůsobit zobrazení ve WebView. Řešení, která budu v této kapitole popisovat, nejsou určitě jediná možná a přistupoval jsem k nim z hlediska účinnosti tak, aby co nejvíce vyhovovala řešenému problému. Budu ale používat všechny základní funkce a vlastnosti komponenty WebView pro dosáhnutí správného výsledku a prvky z použitých jazyků, které se pro tento způsob komunikace nejlépe hodí. Tato část může proto být užitečná i pro různé typy úkolů, kde WebView figuruje a to z toho důvodu, že jsem přesně popsal jak jednotlivé metody a vlastnosti fungují a nyní předvedu i jejich použití. Pro řešitele podobných problémů nebude nic jednoduššího, než můj popisovaný postup upravit a aplikovat na svůj problém.

4.1 Konfigurace WebView komponenty

WebView komponenta je stavebním kamenem celé aplikace. Díky této komponentě je zobrazován celý webový obsah FLOWIO aplikace v té mobilní.

Proto je velmi důležitá základní konfigurace této komponenty. V kapitole 2.2 jsem popsal všechny důležité vlastnosti a metody WebView komponenty. V mobilní aplikaci FLOWIO nebude zapotřebí použít všechny, které jsem v kapitole 2.2 zmiňoval. WebView komponenta je vykreslována v rámci komponenty nazývané FlowioWebViewScreen, která zaobaluje WebView do potřebných kontejnerů, definuje lokální stavové proměnné, používá různé hooky a definuje funkce používané ve Webview. Nejdříve ukážu vzhled kusu kódu, kde je WebView komponenta definována a poté se budu věnovat každé vlastnosti a metodě použité v WebView zvlášť a budu diskutovat použité hodnoty. Některé vlastnosti mají jasně dané hodnoty, které jsou vytvářeny přímo při vykreslování komponenty. Jiné vlastnosti mají jako hodnoty konstanty nebo funkce, které jsou definovány v souborech zvlášť kvůli zachování přehlednosti kódu. Definici těchto hodnot ukážu a budu diskutovat v sekcích, jenž odpovídají jednotlivým vlastnostem WebView komponenty. Dále v této sekci ukážu využití komponenty ScrollView z jazyka React Native, která zaobaluje WebView komponentu a umožňuje uživatelům provádět aktualizaci WebView obsahu za použití mobilního gesta.

4.1.1 Definice samotné WebView komponenty

```
<WebView
    style={styles.webView}
    ref={webViewRef}
    source={{ uri: params.url }}
    originWhitelist={["https://flowio.popronsystems.cz"]}
    onMessage={(event)=> onMessage(event.nativeEvent.data)}
    injectedJavaScriptBeforeContentLoaded={injectedJsBeforeLoad}
    onLoadEnd={updateNotificationCount}
    onScroll={handleScroll}
    renderLoading={() => <ActivityIndicator />}
/>
```

4.1.1.1 style

Hodnotou tohoto propu je datová struktura, jenž obsahuje styly pro WebView kontejner. Aby byl obsah WebView vždy správně zobrazen přes celou výšku zařízení, je nutné mít spravné styly a v případě, že je WebView komponenta zaobalená v ScrollView komponentě, musí být styl který udává výšku WebView nastavován vždy dynamicky.

```
\\ soubor useStyles.js
import { StyleSheet } from "react-native";

const useStyles = (height: number) => {
  return StyleSheet.create({
    webView: {
```

```
      flex: 1,
      height,
    },
    scroll: {
      flex: 1,
      height: "100%",
    },
  });
};

export default useStyles

\\ soubor s WebView komponentou index.js
import { useStyles } from "./useStyles"

const styles = useStyles(height);
```

Listing 4.1: Definice stylů pro WebView komponentu

Vlastnost scroll ze stylů se aplikuje na komponentu ScrollView, jenž bude popsána dále v textu a nastaví jí fixní výšku na 100% výšky zařízení. Parametr height vytvořeného hooku je hodnota z lokálního stavu, kterou upravuje akce z ScrollView komponenty podle toho, jak uživatel posouvá obsah. Více o této implementaci v sekci 4.1.2.

4.1.1.2 ref

Pomocí hooku useRef jsem vytvořil refenrenci, kterou jsem uložil do proměnné `webViewRef`. Přiřazením proměnné k vlastnosti ref dosáhneme toho, že se vytvoří reference na WebView komponentu a uloží se právě do proměnné `webViewRef`. Díky tomu lze pak přistupovat k vlastnostem komponenty a k jejím metodám kdekoliv v FlowioWebViewScreen komponentě.

4.1.1.3 source

Z možností, jak zobrazovat obsah WebView, jsme se rozhodli pro zobrazení pomocí URL adresy. Datová struktura `params` v sobě má vlastnost url, kde je uložena URL adresa, ze které WebView vykreslí svůj obsah. Tato struktura je uložena v local storage aplikace a je upravována na základě toho, jaký si uživatel zvolí server pro zobrazování dat. Tato proměnná v local storage je upravována kdykoliv je zapotřebí změnit vykreslenou stránku ve WebView. Mám tím na mysli například to, když uživatel klikne na notifikaci, jenž se zobrazuje v modálním okénku, které zobrazuje nativní kód. Nativní kód změní url pro WebView na detail notifikace, WebView komponenta zaznamená změnu hodnoty vlastnosti `uri` a vykreslí požadovanou stránku.

4.1.1.4 originWhitelist

Hodnotou je string *https://flowio.popronsystems.cz* , který odkazuje na to, že v rámci WebView komponenty se uživatel nemůže dostat jinam než na zadanou URL adresu a její pod adresy.

4.1.1.5 onMessage

Hodnotou je funkce onMessage , kde se odehrává hlavní komunikace s webovou částí, kterou WebView zobrazuje. V ukázce výše přiřazuji tomuto propu funkce, která vrací funkci onMessage. Dělám to takto, jelikož pro zpracování zprávy z webové části potřebuji pouze to, co je uložené v datové struktuře event.native pod vlastností data . Poté definice funkce onMessage vypadá takto.

```
const onMessage = (jsonData) => {
    const data = JSON.parse(jsonData);
     if (data.type === INVOKE_CHANGE_SERVER) {
      navigation.navigate("ChangeServer");
    } else if (data.type === INVOKE_CLEAR_STORAGE) {
      clearStorage();
    } else if (data.type === INVOKE_SHOW_NOTIFICATIONS) {
      toggleModal(true);
    } else if (data.type === INVOKE_LOG_OUT) {
      logOutUserAction();
    }
  };
```

Tato funkce se provede pokaždé, když je zavolána metoda WebView metoda *postMessage()* viz 2.2.2.1. Parametr jsonData má datový typ *string* a má v sobě uložená komprimovaná data poslaná webovou částí. Pomocí funkce JSON.parse() dekomprimuju data a dostanu objekt. Objekt má v sobě dvě vlastnosti a vypadá následovně.

```
{
  type: String,
  payload: Any \\libovolny datovy typ
}
```

Jelikož jsou webová a nativní část dva rozdílné projekty, musím se jako vývojář starat o to, abych dodržel definovanou datovou strukturu v obou částech.

Vlastnost type určuje typ akce, které se provedla ve webové části a na základě této informace se funkce *onMessage* rozhoduje jaké akce se provedou v nativní části. Vlastnost type může nabývat následujících hodnot.

- `INVOKE_CHANGE_SERVER`
- `INVOKE_CLEAR_STORAGE`
- `INVOKE_SHOW_NOTIFICATIONS`
- `INVOKE_LOG_OUT`

Tyto hodnoty odpovídají pevně definovaným konstantám v samostatném souboru a jsou stejné pro webovou i nativní část.

Ve vlastnosti `payload` může být obsaženo cokoliv, co je potřeba odeslat z webové části do nativní. Hodnotou mohou být jednoduché datové typy jako je *integer* nebo *string*, ale mohou to být také složené datové struktury obsahující více informací.

4.1.1.6 injectedJavaScriptBeforeContentLoaded

Zde je v proměnné `injectedJsBeforeLoad` uložen javascriptový kód ve formě stringu který vypadá následovně.

```
const injectedJsBeforeLoad = `
  window.isNativeApp = true;
  true;
`;
```

Jediné, co je potřeba v webové části před tím než dojde k načtení obsahu nastavit, je globální proměnná `isNativeApp`. Tato proměnná je dostupná pak v webové části a s její pomocí můžeme určit, zda se aplikace vykresluje ve WebView nebo ne. Jak jsem popisoval v analýze této vlastnosti v sekci 2.2.1.4, je zapotřebí přidat na konec každého takového stringu hodnotu *true*, aby nedošlo ke spadnutí aplikace při používání takového typu komunikace.

4.1.1.7 onLoadEnd

Funkce *updateWebviewNotifCount* se jak říká její název, stará o aktualizaci počtu notifikací ve webové části při načtení obsahu WebView. Definice funkce vypadá následovně.

```
const updateWebviewNotifCount = () => {
    if (webViewRef && webViewRef.current) {
      webViewRef.current.injectJavaScript(
        updateNotificationsCount(allNotificationsCount),
      );
    }
  };
```

Funkce pracuje s referencí WebView komponenty a používá metodu WebView *injectJavascript*, jenž provede javascriptový kód, který dostane v argumentu funkce. Na začátku, před tím než se do proměnné webViewRef uloží reference na WebView komponentu, je hodnota proměnné *null* a proto je potřeba ošetřit přístup k vlastnostem reference, aby nedošlo k chybě. Blíže budu funkci *updateWebviewNotifCount* popisovat v sekci **??**.

4.1.1.8 renderLoading

Jako indikátor načítání WebView obsahu jsem zvolil komponentu ActivityIndicator z knihovny react-native.

4.1.1.9 onScroll

Funkce *handleScroll()* se stará o zpracování akce když uživatel začne listovat obsahem WebView. Jak už jsem zmiňoval, tak WebView komponenta je v našem případě zaobalena do komponenty ScrollView, díky které umíme aktualizovat obsah WebView gestem na obrazovce mobilního zařízení. Je ale nutné kontrolovat kdy může k aktualizaci dojít a kdy ne. K aktualizaci dat může dojít pouze v případě, že už se nejde posunout výše na obrazovce, což znamená, že ypsilonová souřadnice polohy WebView kontejneru má hodnotu 0. Proto má komponenta FlowioWebViewScreen v lokálním stavu definovanou proměnnou refreshEnabled. Jelikož při prvním vykreslení WebView komponenty začíná hlavní kontejner se souřadnicemi x a y v bodě 0, je výchozí hodnota proměnné refreshEnabled nastavená na *true*. Pokaždé když uživatel pohne obsahem WebView dolu, respektive nahoru, dojde k zavolání funkce handleScroll, která dostane informace o nové poloze WebView kontejneru. Poté dojde ke kontrole polohy kontejneru WebView a na aktualizaci proměnné refreshEnabled.

```
1 const handleScroll =  (e ) => {
2     if (e.nativeEvent.contentOffset.y === 0 && !refreshEnabled) {
3       setRefreshEnabled(true);
4     } else if (e.nativeEvent.contentOffset.y > 0 &&
      refreshEnabled) {
5       setRefreshEnabled(false);
6     }
7   };
```

Proměnná refreshEnabled se používá v komponentě ScrollView a indikuje povolení aktualizace obsahu WebView pomocí gesta stažením dolů na obrazovce mobilního zařízení. Více o komponentě ScrollView a jejích vlastnostech v sekci 4.1.2.

4.1.2 Konfigurace komponenty ScrollView

ScrollView kompomta zaobaluje Webview komponentu a má na WebView vliv především ze dvou důvodů. Jedním je už zmiňovaná možnost aktualizace obsahu WebView pomocí gesta na obrazovce zařízení a druhým je nastavování pevné velikosti WebView komponenty při posouvání. V sekci 4.1.1.1 jsem zmiňoval parametr `height`, který je lokálním stavem komponenty FlowioWebViewScreen a právě tento parametr upravuje jedna z vlastností ScrollView komponenty.

4.1.2.1 Definice samotné ScrollView komponenty

```
<ScrollView
   onLayout={onLayoutChange}
   refreshControl={
     <RefreshControl
         onRefresh={refresh}
         enabled={refreshEnabled}
         refreshing={false}
     />
   }
   style={styles.scroll}
>
    <WebView {...props} />
</ScrollView>
```

4.1.2.2 onLayout

Funkce `onLayoutChange` dostane v argumentu funkce událost s informaceme o kontejneru komponenty ScrollView. Tato funkce se zavolá pokaždé když dojde ke změně zobrazení kontejneru. Funkce pak porovná hodnotu lokální proměnné `webViewHeight` s novou velikostí ScrollView kontejneru. Konkrétně se zde porovnává výška. V případě že se nerovnají, tak se aktualizuje lokální proměnná `webViewHeight` a tím pádem se přenastaví i velikost WebView zobrazení viz 4.1.1.1.

```
const onLayoutChange = (e) => {
    if (webViewHeight !== e.nativeEvent.layout.height) {
      setWebViewHeight(e.nativeEvent.layout.height);
    }
  };
```

Aby WebView budilo dojem mobilní aplikace musí pokrývat celou šířku a výšku mobilního zařízení. V případě, když bychom nepotřebovali funkci aktualizace WebView obsahu pomocí gesta, tak by stačilo nastavit ve stylech WebView komponenty atribut `height` na 100% a o posouvání obsahu by se staralo samotné WebView jako je tomu v prohlížeči. Tady taková funkcionalita ale je a tím, jak uživatel posouvá obsah, by mohlo dojít k tomu, že by

výšky WebView obsahu a kontejneru, který zaobaluje WebView komponentu, nebyly konzistentní. Z toho důvodu je vždy nutné kontrolovat výšku WebView a v případě, že se liší od velikosti kontejneru tak ji aktualizovat.

4.1.2.3 refreshControl

Komponenta RefreshControl je z modulu `react-native` a jedná se o načítací kolečko, které se zobrazí v případě, že uživatel provede gesto stažení dolů obsahu pro jeho aktualizaci. Tato komponenta se zároveň stará i o to, co se při takové akci má stát. Intuitivně bude potřeba aktualizovat patřičná data, ale v těle funkce, která se zavolá, lze definovat libovolnou akci.

onRefresh

Funkce `refresh` nemá žádný argument a slouží pouze k jedinému účelu a to k přenačtení WebView obsahu. Nejdřív je třeba zkontrolovat existenci reference na WebView komponentu a v případě její existence se zavolá metoda `reload`.

```
1 const refresh = () => {
2     if (webViewRef && webViewRef.current) {
3       webViewRef.current.reload();
4     }
5   };
```

enabled

Hodnotou tohoto propu je hodnota proměnné `refreshEnabled`, což je lokální proměnná komponenty FlowioWebViewScreen a nastavuje jí funkce `handleScroll`, kterou volá vlastnost `onScroll` z WebView komponenty viz 4.1.1.9. Když bude hodnota *false* nebude akce aktualizace dostupná a bude možné používat gesto posouvání bez vedlejších efektů.

refreshing

Hodnutu *false* jsem zde nastavil z toho důvodu, že potřebuji uživateli pouze jednorázově dát vědět o tom, že došlo k přenačtení dat. WebView má pak vlastní indikátory načítání jako je vlastnost `renderLoading` a nebo webová aplikace má také vlastní indikátory načítání obsahu.

4.2 Implementace komunikace jednotlivých částí pro zobrazení notifikací v mobilní aplikaci

Úprava notifikací pro mobilní aplikaci byla z programovacího hlediska největším oříškem. Ve všech ostatních případech stačilo pracovat s vlastnostmi a metodami, které WebView dodává pro komunikaci s webovou aplikací. Jedná se o posílání informací před prvním vykreslením WebView komponenty pomocí vlastnosti `injectedJavaScript` nebo pak volání akcí, ať už z webové části pomocí metody `postMessage` nebo z nativní části pomocí metody `injectJavaScript`. Tyto metody jsou zpravidla volány po interakci uživatele s tlačítkem nebo s jiným prvkem aplikace. Notifikace ale přichází asynchronně v závislosti na akcích ostatních uživatelů a proto je zapotřebí stále upravovat stav notifikací v aplikaci. Proces, jenž kontroluje stav notifikací na serveru a stará se o aktualizaci front-endu, funguje na webové části zcela soběstačně. Jak už jsem ale popisoval, v mobilní aplikaci se o zpracování notifikací stará nativní část a může zpracovávat notifikace i z více datových zdrojů. V sekci 3.2.2 jsem popsal, jak by se měly notifikace chovat a co je třeba změnit oproti webové části. Webová část musí uživateli zobrazovat součet všech notifikací napříč definovanými datovými zdroji. Toto číslo je získáno v nativní části, kde se o získávaní aktuálního počtu notifikací stará stejný mechanismus jako ve webové části a pak je zapotřebí toto číslo poslat webové části. Toto se ovšem děje při každé změně počtu notifikací, tudíž ve webové části bylo zapotřebí implementovat posluchače, jenž se stará o aktualizaci dat ve webové části. K docílení požadovaného výsledku bylo zapotřebí využít kombinaci principů, které jsem popisoval v sekci kde jsem analyzoval jednotlivé použité programovací jazyky a jejich principy.

Dále budu popisovat samotnou implementaci toho, co jsem znázornil na obrázku 3.2. Popis implementace rozdělím do dvou částí a to do webové a nativní části. Budu popisovat pouze komunikaci přes WebView a zpracování a ukládání informací, které byly získány pomocí WebView. Push notifikace, získávání notifikací ze serveru, zpracování jejich uložení a další zmiňovat nebudu, jelikož ty části nejsou nijak ovlivněny WebView komponentou.

4.2.1 Úprava webové části aplikace FLOWIO pro zpracování notifikací

Implementaci na straně webové části jsem se rozhodl popsat jako první z toho důvodu, že akce jenž definují naslouchání příchozích akcí jsou definovány ve webové části a popis akcí v nativní části bude později jasnější. Budu se zde zabývat komponentou `Notifications`, která se ve webová části aplikace stará o zobrazení celkového počtu nových notifikací. Nejdříve popíšu a ukážu implementaci této komponenty před tím než dojde k úpravě pro mobilní aplikaci.

Tím bych chtěl ukázat to, že stačí zasáhnout minimálně do původního kódu pro získání požadovaného výsledku, což je jedním z cílů této práce.

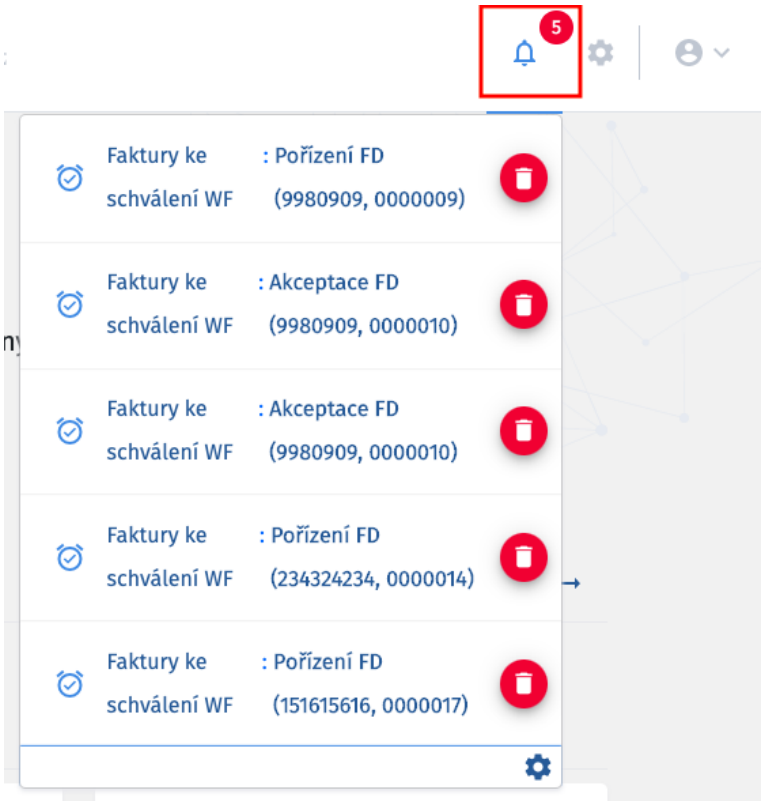
4.2.1.1 Komponenta Notifications před úpravou pro mobilní aplikaci

```
const Notifications = ({ notificationCount }) => {
  const [isOpen, toggleOpen] = useState(false);

  return (
     <StyledDropdown
       isOpen={isOpen}
       toggle={toggleOpen}
     >
       <DropdownToggle>
         <StyledNotificationsIcon isActive={isOpen} />
         <StyledNotificationIndicator notificationCount={
    notificationCount} />
       </DropdownToggle>
       <StyledDropdownMenu>
         <NotificationViewTemplateCardLinkList
           onNotificationClick={toggleOpen}
         />
       </StyledDropdownMenu>
     </StyledDropdown>
   );
};

export default Notifications;
```

Jediným propem této komponenty je prop `notificationsCount`, což je počet notifikací získaný ze serveru. Jelikož se jedná o vlastnost komponenty Notifications, dojde při každé změně vlastnosti `notificationsCount` k znovuvykreslení komponenty a tudíž i k aktualizaci počtu notifikací. Dále má komponenta jednu stavovou proměnnou a to `isOpen`. Tato proměnná obsahuje logickou hodnotu, jenž indikuje zda je seznam všech notifikací zobrazený nebo ne. Komponenta vykresluje komponentu StyledDropdown, která se stará o zobrazování seznamu notifikací. Má vlastnosti `isOpen` a `toggle`, jejichž hodnotami jsou stavové proměnné `isOpen` a `toggleOpen`, které uchovávají stav o otevřenosti nebo zavřenosti seznamu notifikací. Výchozí hodnota proměnné `isOpen` je nastavena na logickou hodnotu *false*, jelikož seznam má být zavřený při vykreslení aplikace. Komponenta StyledDropdown zaobaluje dvě komponenty, DropdownToggle a StyledDropdownMenu. DropdownToggle se stará o zobrazení ikonky zvonečku a počtu nových notifikací pomocí vlastnosti `notificationsCount`. StyledDropdownMenu se stará o zobrazení seznamu všech notifikací a jejich krátkého popisku pro uživatele. Uživatel kliknutím na

zvoneček nastaví hodnotu stavové proměnné `isOpen` na hodnotu *true* a dojde k rozbalení seznamu notifikací. Zavření pak probíhá zase kliknutím na zvoneček nebo kamkoliv mimo seznam notifikací. Vzhled komponenty Notifications ve webové aplikaci lze vidět na obrázku 4.1 a 4.2.

Obrázek 4.1: Vzhled komponenty Notifications ve webové aplikaci, kde má stavová proměnná `isOpen` hodnotu *false*.



Obrázek 4.2: Vzhled komponenty Notifications ve webové aplikaci, kde má stavová proměnná `isOpen` hodnotu *true* a je rozbalený seznam notifikací.

4.2.1.2 Komponenta Notifications po úpravě pro mobilní aplikaci

Když dojde k vykreslení webová aplikace ve WebView, znamená to že je aplikace zobrazována na mobilním zařízení a je potřeba upravit logiku v komponentě Notifications. V části 3.2.2 jsem popsal, že se bude lišit číslo, jenž se vykresluje jako počet nových notifikací a akce, která se provede po kliknutí na notifikační zvoneček. Dále je také zapotřebí přidat indikátor jenž rozhodne, že se jedná o nativní mobilní aplikaci. K tomu abych dosáhl požadovaného výsledku a připravil komponentu Notifications pro zobrazení ve WebView jsem především musel přidat nový stav komponety, kde se uloží počet notifikací získaný z nativní části a akci, která tento stav aktualizuje v případě že přibudou nové notifikace. Dále jsem musel zařídit aby se při kliku uživatele na notifikační zvoneček neotevřel seznam notifikací v rámci webové aplikace, ale aby se poslala akce nativní části a tam se provedlo zobrazení všech notifikací, napříč všemi přidanými datovými zdroji. Níže lze vidět upravený kód komponenty Notifications a poté se budu věnovat jednotlivým změnám.

```
1  const Notifications = ({ notificationCount }) => {
2    const [isOpen, toggleOpen] = useBooleanState(false);
3    const [nativeNotifCount, setNotifCount] = useState<number |
       undefined>(
4      undefined
5    );
6
7    const isNativeApp = useIsNativeApp();
8
9    const handleNotificationClick = () => {
10     invokeRNMessage(INVOKE_SHOW_NOTIFICATIONS, window.location.
       href);
11   };
12
13   const getNotificationsCount = (event) => {
14     if (event.detail) {
15       setNotifCount(Number(event.detail));
16     }
17   };
18
19
20   useEffect(() => {
21     window.addEventListener(
22         "notificationsChanged",
23         getNotificationsCount
24         );
25     return () => {
26       window.removeEventListener(
27       "notificationsChanged",
28       getNotificationsCount
29       );
30     };
31   }, []);
32
```

```
33   const renderWebNotifications = () => {
34     return (
35       <StyledDropdown
36         isOpen={isOpen}
37         toggle={toggleOpen}
38       >
39     {...}
40       </StyledDropdown>
41     );
42   };
43
44   const renderNativeNotifications = () => {
45     return (
46     <DropdownToggle
47             onClick={handleNotificationClick}
48           >
49             <StyledNotificationsIcon isActive={isOpen} />
50             <StyledNotificationIndicator
51               notificationCount={nativeNotifCount}
52             />
53           </DropdownToggle>
54     );
55   };
56
57   return isNativeApp
58     ? renderNativeNotifications()
59     : renderWebViewNotifications();
60 };
61
62 export default Notifications;
```

Funkce renderNativeNotifications a renderWebNotifications

Jak lze vidět výše, tak v kódu kromě nové stavové proměnné přibylo i několik funkcí. Jako první bych chtěl popsat funkce `renderNativeNotifications` a `renderWebNotifications` a proměnnou `isNativeApp` z hooku `useIsNativeApp`, která rozhoduje o tom jaká funkce vykreslí správný obsah Notifications komponenty.

Tyto dvě funkce se starají o vykreslení obsahu komponenty Notifications na základě toho, jestli je webová aplikace vykreslena ve WebView nebo ne. Funkce `renderWebNotifications` vrací stejný obsah, jaký vracela komponenta Notifications před úpravou. Z toho důvodu, že upravená Notifications komponenta využívá jiné stavové proměnné i jiné funkce, rozhodl jsem se že vykreslení rozdělím do dvou funkcí a vykreslím pouze jednu v závislosti na proměnné `isNativeApp`. Funkce `renderNativeNotifications` vykresluje opět zvoneček s počtem notifikací s tím rozdílem, že neobsahuje seznam notifikací, ale vykresluje počet notifikací uložený v proměnné `nativeNotifCount` a akce po kliknutí na zvoneček odešle zprávu do nativní části.

isNativeApp

Hodnotu této proměnné nastavuje hook `useIsNativeApp` a indikuje zda je webová aplikace vykreslena v WebView nebo klasicky v prohlížeči.

```
1  export const useIsNativeApp = () =>
2    window.isNativeApp === true;
```

Hook přistupuje ke globální proměnné `window` a kontroluje, jestli má vlastnost `isNativeApp`. Jestli takovou vlastnost v sobě má a její hodnota je nastavena na *true*, znamená to že se jedná o mobilní aplikaci a webová aplikace je zobrazena formou WebView. Vlastnost `isNativeApp` nastavuje do proměnné `window` WebView při vykreslení svého obsahu pomocí propu `injectedJavaScriptBeforeContentLoaded`.

Proměnná nativeNotifCount a funkce getNotificationsCount

Proměnná `nativeNotifCount` je stavová proměnná komponenty Notifications, které slouží k uložení počtu notifikací získaných z nativní části. Tuto hodnotu nelze uložit do globální proměnné `window` jako tomu bylo u vlastnosti `isNativeApp`. Bylo by to možné, pokud by se mělo zobrazit jedno číslo, známé už před vykreslením obsahu WebView. Zde je zapotřebí toto číslo aktualizovat v závislosti na nově příchozích notifikacích.

Jako možné řešení by se také mohlo zdát využití hooku `useEffect`, kde bych do pole závislostí dal proměnnou `window.notifCount`, kterou bych nastavoval v nativní části při změně počtu notifikací a při každé změně hodnoty této proměnné by se provedla definovaná akce v těle hooku `useEffect`. To by ale nefungovalo a to proto, že k zavolání hooku dojdu pouze v případě, že v poli závislosti jsou pouze stavové nebo prop proměnné komponenty. Proměnná `window` je globální proměnná a proto by změna této proměnné neměla na `useEffect` žádný vliv. Hook `useEffect` by se ale použít dal a to pro registraci a uvolnění posluchače události. Posluchače mohou být v rámci JavaScriptu předdefinované a nebo si je mohou programátoři definovat sami. Zde se hodí nadefinovat posluchač, který se vyvolá v případě že dojde v nativní části ke změně počtu notifikací. V kódu výše proto používám hook `useEffect`, ve kterém definuji posluchač s názvem *notificationChanged*, který v případě, že nějaký proces vyvolal událost s daným jménem, provede funkci `getNotificationCount`.

```
1  const getNotificationsCount = (event) => {
2     if (event.detail) {
3       setNotifCount(Number(event.detail));
4     }
5   };
```

Funkce `getNotificationsCont` má jeden argument, kterým je proměnná event. Kromě toho, že v sobě tato proměnná obsahuje všechny vlastnosti, které má klasická událost v JavaScriptu, tak obsahuje také uživatelem definovanou vlastnost `detail`. Do této proměnné je možné při vyvolání události uložit libovolnou hodnotu. V tomto případě tímto způsobem posílám z nativní části aktuální počet notifikací a po kontrole, na existenci proměnné, nastavuji novou hodnotu stavové proměnné `nativeNotifCount`. Tím, že se jedná o stavovou proměnnou, komponenta Notifications zaznamená změnu a znovu vykreslí notifikační zvoneček, teď už s novým počtem notifikací. Volání události a posílání počtu notifikací budu popisovat v části níže, jenž se bude týkat implementace v nativní části.

handleNotificationClick

Tato funkce se zavolá v případě, že uživatel klikne na notifikační zvoneček a jedná se o zobrazení webové aplikace v té mobilní.

```
export const invokeRNMessage = (type, payload) => {
  if (isNativeApp()) {
    const data = {
      type,
      payload,
    };
    window.ReactNativeWebView.postMessage(JSON.stringify(data));
  }
};

const handleNotificationClick = () => {
    invokeRNMessage(
        INVOKE_SHOW_NOTIFICATIONS,
        window.location.href
      );
  };
```

Funkce zavolá další funkci `invokeRNMessage`, která se stará o odeslání zprávy do WebView. Definuji zde typ zprávy `INVOKE_SHOW_NOTIFICATIONS` a spolu s typem posílám informaci, na které adrese došlo ke kliku na notifikační zvoneček. Informaci, na které adrese došlo ke kliku posílám z toho důvodu, že musím umožnit uživateli v nativní části krok zpět při kliku na nějakou notifikaci ze seznamu. Takovým kliknutím se uživatel dostane na detail zadané notifikace a když bych tuto informaci neposlal, nebyl by žádný záznam o předešlé stránce. Kontroluji, že se jedná o nativní aplikaci, tudíž existuje reference na WebView, pomocí které mohu použít metodu `postMessage`. Tato metoda bere jediný argument typu `string` a proto musím data před předáním do metody kompresovat. Pomocí této zprávy chci nativní části zdělit, že si uživatel chce

zobrazit seznam všech notifikací. Odeslanou zprávu zachytí vlastnost WebView komponenty `onMessage` a provede další akce. Tyto akce budou dále popsány v implementaci nativní části.

4.2.2 Implementace nativní části pro zpracování notifikací

V nativní části nebude zapotřebí ukazovat jak vypadala komponenta pro zpracování notifikací před tím než se aplikovaly změny pro WebView, jelikož se komponenta vyvíjela soustavně s vznikem mobilní aplikace spolu s implementací WebView. Nativní část ukládá aktuální počet notifikací v lokálním uložišti a stará se o získávání nových notifikací ze serveru. Pokaždé, když dojde ke změně počtu notifikací, odešle nový počet notifikací do webové části. Nativní část dále obsahuje seznam všech notifikací napříč přidanými datovými zdroji a zpracovává akci z webové části, která zažádá o zobrazení seznamu notifikací. Seznam notifikací je v nativní části reprezentován komponentou NotificationsModalContainer jenž je vykreslována spolu s WebView komponentou v komponentě FlowioWebViewScreen, kterou už jsem zmiňoval výše v sekci 4.1. Dále přidám ukázku celé implementace zpracování notifikací v kombinaci s WebView v nativní části. Poté se budu zabývat jednotlivými funkcemi a proměnnými.

```
1  const FlowioWebViewScreen = () => {
2    const [isOpen, toggleModal] = useState(false);
3    const webViewRef = useRef<any>(null);
4    const allNotificationsCount = useAllNotificationsCount();
5
6    const updateWebviewNotifCount = () => {
7      if (webViewRef && webViewRef.current) {
8        webViewRef.current.injectJavaScript(
9          updateNotificationsCount(allNotificationsCount),
10       );
11     }
12   };
13
14   useEffect(() => {
15     updateWebviewNotifCount();
16   }, [allNotificationsCount]);
17
18   const onMessage = (jsonData: string) => {
19     const data = JSON.parse(jsonData);
20
21     if (data.type === INVOKE_SHOW_NOTIFICATIONS) {
22       toggleModal(true);
23     }
24     ...
25   };
26
27   return (
28     <SafeAreaView>
```

```
      <NotificationsModalContainer
        toggleModal={toggleModal}
        isOpen={isOpen}
      />
      <ScrollView {...props }>
        <WebView
           ref={webViewRef}
          onMessage={
            (event) => onMessage(event.nativeEvent.data)
          }
          onLoadEnd={
            () => updateWebviewNotifCount()
          }
          {...props}
        />
      </ScrollView>
    </SafeAreaView>
  );
};

export default FlowioWebViewScreen;
```

4.2.2.1 Odesílání notifikací webové části

Počet notifikací je získán z lokálního uložiště pomocí hooku `useAllNotificationsCount` a uložen do proměnné `allNotificationsCount`. Chtěl bych podotknout, že hodnota uložená v této proměnné reprezentuje počet všech notifikací napříč všemi přidanými datovými zdroji uživatelem v mobilní aplikaci. Pro lepší přehlednost níže ukázka kódu proměnných a metod spojených s odesíláním notifikací.

```
const allNotificationsCount = useAllNotificationsCount();

const updateWebviewNotifCount = () => {
  if (webViewRef && webViewRef.current) {
    webViewRef.current.injectJavaScript(
       updateNotificationsCount(allNotificationsCount),
    );
  }
};

useEffect(() => {
    updateWebviewNotifCount();
  }, [allNotificationsCount]);
```

Tím, že je v proměnné `allNotificationsCount` uložená hodnota z hooku, stává se z proměnné stavová proměnná komponenty FlowioWebViewScreen a tím pádem změna její hodnoty bude mít dopad na životní cyklus komponenty a dojde k provedení funkce v hooku `useEffect`.

updateWebViewNotifCount

Tato funkce se provede pokaždé když dojde ke změně počtu notifikací a má za úkol poslat nový počet notifikací webové části. Ve funkci je použita metoda z WebView `injectJavaScript`, která přijímá v argumentu JavaScriptový kód ve formě stringu. K této metodě přistupujeme přes referenci, která odkazuje na WebView komponentu. Může se ale stát, že je její hodnota *null* a proto je třeba přístup řádně ošetřit. Níže lze vidět definici funkce `updateNotificationsCount`.

```
1 const updateNotificationsCount = (notifCount: number) => `
2   window.dispatchEvent(
3     new CustomEvent(
4       'notificationsChanged',
5       { detail: ${notifCount} }
6   ));
7   true;
8 `;
```

Argumentem funkce je nový počet notifikací a vrací kus javascriptového kódu ve formě stringu. Tento kód se pak provede ve webové části a má za úkol vyvolat událost s názvem `notificationsChanged`. V této události posílá nový počet notifikací a posluchač události ve webové části, který už jsem zmiňoval v sekci 4.2.1.2, tuto událost zpracuje a získá z proměnné `detail` aktuální počet notifikací.

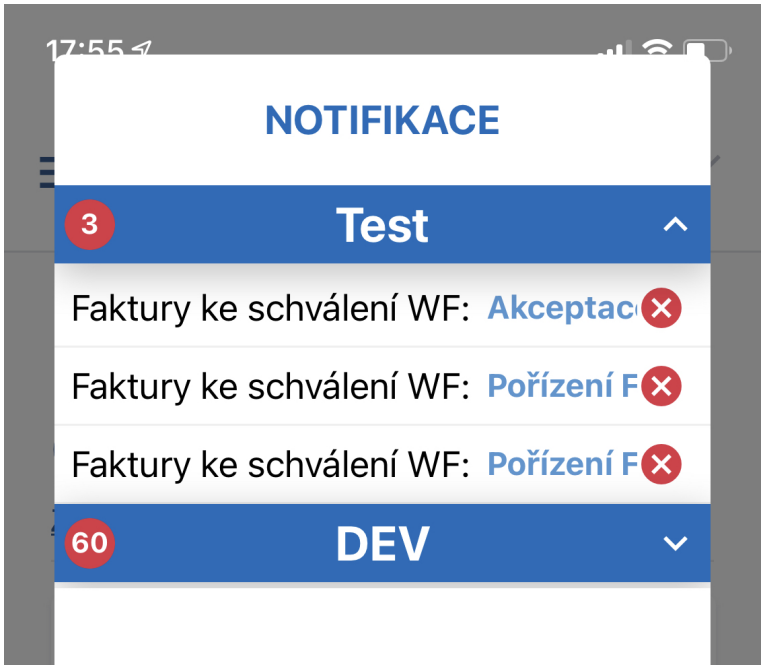
Tento způsob komunikace mezi nativní částí a webovou částí prostřednictvím WebView je poněkud netradiční, ale v kombinaci přístupů z jednotlivých jazyků lze takto dosáhnout požadovaného výsledku.

onMessage

Funkci `onMessage` už jsem popisoval při definici implementace WebView komponenty v sekci 4.1.1.5. Zde mne ale zajímá případ, kdy data odeslané z webové části obsahují typ `INVOKE_SHOW_NOTIFICATIONS`, což značí, že uživatel klikl na notifikační zvoneček ve webové části a chce si zobrazit seznam notifikací. Když se jedná o tento typ zprávy tak pomocí funkce `toggleModal` nastavím hodnotu stavové proměnné `isOpen` na *true*. Hodnota této proměnné indikuje, zda je seznam notifikací ve webové části zobrazený či nikoliv.

Komponenta NotificationsModalContainer

Tato komponenta se stará o zobrazení seznamu notifikací v nativní části aplikace. Má dva propy a to `isOpen` a `toggleModal`. Prop `isOpen` indikuje, zda je seznam otevřený nebo ne. Prop `toggleModal` nastavuje hodnotu proměnné



Obrázek 4.3: Vzhled seznamu notifikací v mobilní aplikaci (seznam obsahuje dva různé datové zdroje DEV a Test)

`isOpen` a stará se o uzavření seznamu notifikací. V případě, že dojde k označení notifikace za přečtenou nebo k jejímu smazání, upraví se počet notifikací v lokálním uložišti, změní se hodnota proměnné `allNotificationsCount` a dojde k aktualizaci počtu notifikací ve webové části.

4.3 Přihlášení do aplikace

Přihlášení do aplikace jsem sice popisoval v sekci 3.1, ale pouze z hlediska funkčnosti a provedení forkflow, jelikož se přístup v mobilní aplikaci odlišuje od toho ve webové. Z hlediska implementace ale nebylo potřeba přihlašování ve webové části nijak upravovat. WebView totiž přímo s samotným přihlašovaním do aplikace nijak nepracuje. Přihlášení proběhne v nativní části za použití nativního jazyka React Native. Na server se pošle informace o přihlášení uživatele a rovnou se tam ta informace uloží. Poté se nastaví URL adresa, která je předána do WebView jako začínající stránka. URL odkazuje na úvodní stránku webové aplikace FLOWIO. Webová aplikace FLOWIO má v hlavní komponentě mechanismus, který se stará o zjišťování, jestli uživatel už není v aplikaci přihlášený. Pokaždé když dojde k vykreslení aplikace, posílá se na server žádost o zjištění stavu přihlášení uživatele. V případě, že k přihlášení uživatele došlo už v nativní části, WebView vykreslí webovou aplikaci která od

serveru zjistí, že tento uživatel už je přihlášený a vykreslí úvodní stránku aplikace FLOWIO. Z toho důvodu, že se zde WebView nepoužívá jinak než pro vykreslení obsahu, nebudu dodávat ukázku a popis implementace přihlášení. Dále budu ale popisovat akci odhlášení z aplikace, jelikož ta s WebView už pracuje.

4.4 Implementace odhlášení z aplikace, změna serverů a vymazání uložiště

Tyto tři akce jsem se rozhodl seskupit do jedné sekce, jelikož jsou všechny zobrazovány ve webová části v rámci jedné komponenty a v nativní části se o přijímání těchto akcí stará jedna funkce. Opět rozdělím popis implementace do dvou částí a to do webové a nativní části. V nich se pak budu věnovat každé akci zvlášť.

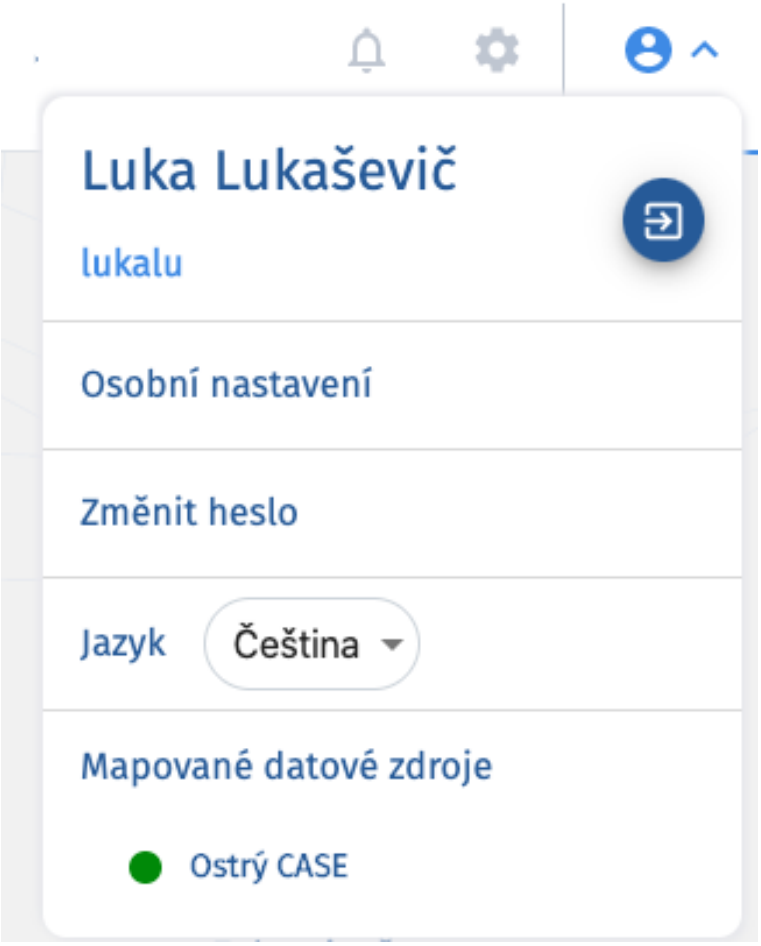
4.4.1 Implementace akcí v webové části aplikace FLOWIO

Všechny akce se ve webové části budou nacházet v uživatelské sekci, která je při klasickém používání aplikace neviditelná a zobrazí se kliknutím na ikonku uživatele v pravém horním rohu aplikace. Tato ikonka je součástí navigace a proto je viditelné na každé stránce webové aplikace a proto je možné k ní přistupovat napříč celou aplikací. Uživatelská sekce umožňuje provádět různé uživatelské akce, mezi které v případě zobrazení aplikace formou WebView přibudou další, týkající se nativních akcí. Jak můžete vidět na obrázcích 4.4 a 4.5, tak v mobilní aplikace přibudou v uživatelské sekci dvě akce z pohledu UI a akce odhlášení bude prováděna ze stejného tlačítka jako v případě webové aplikace. O vykreslení sekce se stará komponenta nazvaná UserSection. Ta se skládá z komponent, které odpovídají jednotlivým akcím. Každá akce má vlastní komponentu a společně tvoří celek komponenty UserSection. Budu zde popisovat pouze implementaci akcí v sekcích pro odhlášení uživatele, změnu serveru a vyčištění uložiště.

4.4.1.1 Odhlášení uživatele z aplikace

O odhlášení uživatele z aplikace se stará komponenta LogOutSection, která vykresluje tlačítko, po jehož stisknutí se provede příslušná akce. V případě mobilní aplikace odhlášení ale neproběhne ve webové části, ale v té nativní.

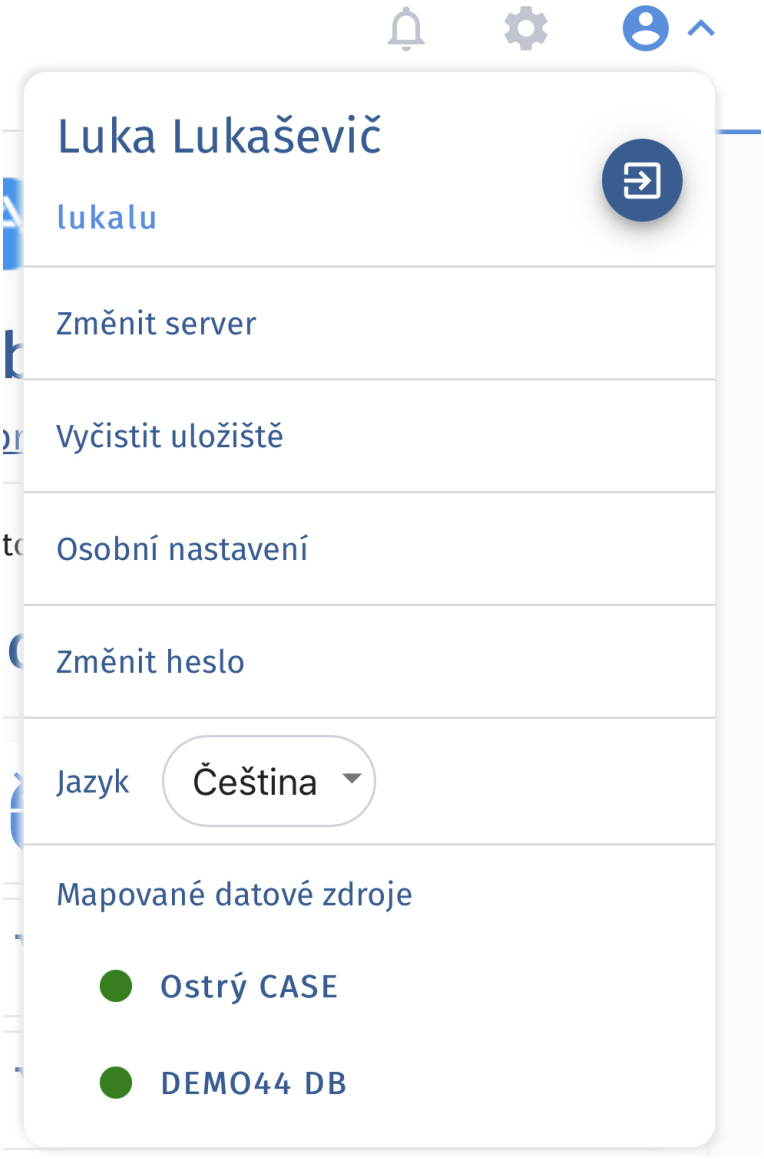
```
1 const UserSection = ({
2   requestLogout,
3 }) => {
4   const isNativeApp = useIsNativeApp()
5   const handleLogOut = () => {
6     if (isNativeApp()) {
7       invokeRNMessage(INVOKE_LOG_OUT);
```



Obrázek 4.4: Vzhled uživatelské sekce při zobrazení aplikace v prohlížeči

```
8       } else {
9         requestLogout();
10      }
11    };
12
13    return (
14      <SectionRow>
15        <UserSectionContainer>
16          <LogOutButton onClick={handleLogOut} />iv>
17        </UserSectionContainer>
18      </SectionRow>
19    );
20  };
21
22  export default UserSection;
```

Komponenta má jeden prop, kterým je funkce pro odhlášení uživatele v webové



Obrázek 4.5: Vzhled uživatelské sekce v mobilní aplikaci

aplikaci. Tato funkce je předána z kontejneru, který zaobaluje komponentu UserSection a má v sobě API akce. Opět zde využívám hook `useIsNativeApp`, díky jehož hodnotě zjistím, zda se jedná o zobrazení aplikace formou WebView nebo v klasickém prohlížeči. Tlačítko pro odhlášení je reprezentováno komponentou LogOutSectionButton. Při uživatelském kliknutí zavolá funkci `handleLogOut` a v případě, že se jedná o mobilní aplikaci, pošle zprávu na-

tivní části, jinak odhlásí uživatele klasickým způsobem. Funkci `invokeRNMessage` už jsem popisoval v sekci 4.2.1.2, kde v tomto případě odešle do nativní části zprávu `INVOKE_LOG_OUT`. Tento typ zprávy má uvědomit nativní část o tom, že se uživatel chce odhlásit z aplikace a je zapotřebí provést patřičné akce.

4.4.1.2 Změna serveru

Komponenta zobrazující sekci, ve které je tlačítko pro změnu serveru, se nazývá ChangeServerSection.

```
const ChangeServerSection = () => {

 const handleClickChangeServer = () => {
    invokeRNMessage(
    INVOKE_CHANGE_SERVER,
    window.location.href
   );
  };

 return (
  <SectionRow>
    <SectionLink onClick={handleClickChangeServer}>
       <FormattedMessage id="webview.change_server" />
     </SectionLink>
   </SectionRow>
 );
}
```

Komponenta nemá žádný prop a má velmi jednoduchý úkol. Poté co uživatel klikne na tlačítko se zavolá funkce `handleClickChangeServer`. Tato funkce zavolá funkci `invokeRNMessage`, kterou jsem popisoval v sekci 4.2.1.2. Zpráva `INVOKE_CHANGE_SERVER` signalizuje to, že uživatel zažádal o změnu serveru a nativní část musí zobrazit odpovídající nativní stránku pro výběr serveru. Spolu se zprávou posílám v komprimovaném objektu opět aktuální URL adresu pro případnou akci zpět.

4.4.1.3 Vyčištění lokálního uložiště

O vyčištění lokálního uložiště se ve webové části stará komponenta ClearStorageSection.

```
const ClearStorageSection = () => {

 const handleClickClearStorage = () => {
    invokeRNMessage(
    INVOKE_CLEAR_STORAGE,
    window.location.href
```

```
    );
  };

 return (
  <SectionRow>
    <SectionLink onClick={handleClickClearStorage}>
       <FormattedMessage id="webview.clear_storage" />
     </SectionLink>
   </SectionRow>
 );
}
```

Implementace této komponenty je velmi podobná implementaci komponenty ChangeServerSection v sekci 4.4.1.2. Jediné rozdílnost v jejich implementaci je pojmenování funkce a typ zprávy odesílaný nativní části. Tato zpráva má za úkol informovat nativní část o tom, že je zapotřebí vyčistit lokální uložiště aplikace. Je to jediný způsob jak vymazat data, která se uchovávají po celou dobu přítomnosti aplikace na zařízení.

4.4.2 Implementace akcí v nativní části aplikace FLOWIO

V nativní části byla implementace těchto tří akcí ještě jednodušší než v té nativní. Už jsem zmiňoval, že o zpracování zpráv poslaných z webové části prostřednictvím metody komponenty WebView se stará funkce `onMessage`. Tuto funkci jsem už zmínil v sekci 4.2.2, kde jsem popisoval příjímání zprávy z webové části týkající se notifikací.

```
const onMessage = (jsonData: string) => {
   const data = JSON.parse(jsonData);
   if (data.type === INVOKE_CHANGE_SERVER) {
      navigation.navigate("ChangeServer");
    } else if (data.type === INVOKE_CLEAR_STORAGE) {
      clearStorage();
    } else if (data.type === INVOKE_LOG_OUT) {
      logOutUserAction();
    }
    ...
  };
```

V kódu výše je implementace funkce `onMessage` pro akce odhlášení, změnu serveru a vyčištění uložiště v mobilní aplikaci. Funkce definuje poslech i pro jiné operace, ale ty už jsem v této práci popisoval a proto není zapotřebí je zde znovu zmiňovat. Funkce je zavolána vždy když je ve webové části zavolána metoda `postMessage` s argumentem ve formě objektu komprimovaného do stringu. Tento string je pak i argumentem funkce `onMessage` a po dekompresi získám potřebné data, která v sobě nese.

4.4.2.1 Odhlášení uživatele z aplikace

V případě, že data z webové části obsahují zprávu `INVOKE_LOG_OUT`, je zapotřebí odhlásit uživatele z mobilní aplikace. Tuto akci je nutné provést v nativní části, jelikož i přihlašování probíhá v této části. Funkce `logOutUserAction` je předána komponentě FlowioWebViewScreen formou props a není potřeba znát její definici. Tato funkce provede odhlášení uživatele a změní aktuální stránku na stránku, které slouží pro přihlášení do aplikace. Je důležité zmínit, že tato akce pouze provede odhlášení uživatele z jednoho, zrovna aktivního datového zdroje a nedojde k promazání uložených dat.

4.4.2.2 Změna serveru

Když je typ posílaných dat z webové části `INVOKE_CHANGE_SERVER`, nativní část musí přesměrovat uživatele na stránku pro změnu serveru. Opravdu je to tak jednoduché. Stránka pro změnu serveru obsahuje seznam už přidaných serverů a tlačítko pro přidání nového. V případě výběru jednoho z už přidaných serverů dojde k znovunačtení obsahu WebView, ovšem nyní s daty z nově zvoleného serveru. Jestli se požadovaný server v seznamu serveru nezobrazuje je zapotřebí ho přidat. Po stisku tlačítka pro přidání serveru je uživatel přesměrován na stránku pro přidání serveru a proces je pak stejný jako v případě prvního příchodu do aplikace.

4.4.2.3 Vyčištění uložiště

V případě, že je typ dat z webové části `INVOKE_CLEAR_STORAGE`, musí nativní část zařídit vymazání lokálního uložiště. O tuto akci se stará funkce `clearStorage`, kterou získám z hooku `useClearStorage`. Funkce smaže veškerá data z lokálního uložiště zařízení, tak i z lokálního uložiště aplikace. Po provedení tohoto mazání je uživatel přesměrován na úvodní stránku, kterou je stránka pro výběr serveru. Tato akce zařídí nastavení aplikace do stejného stavu, v jakém je aplikace při stažení z Apple Store nebo z Google Play.

4.5 Nástroje, které lze použít při vývoji nativní aplikace s WebView, pro zkoumání komponent a síťových prvků

Psaní kódu je jedna věc a druhá zas testování výsledku. U klasických webových aplikací máme k dispozici vývojářské nástroje ve všech moderních internetových prohlížečích. Tyto nástroje nám umožňují prohlížet DOM strukturu stránky, síťovou komunikaci, obsah cookies a lokálního uložiště a mnoho dalšího. Webové aplikace programované v jazyku React JS už jsou tak běžné, že jim prohlížeče vyšly vstříc a umožňují používat řadu doplňků pro kontrolu React komponent. Například Google Chrome má ve vývojářských nástrojích možnost

zkoumat stromovou strukturu komponent v react js aplikaci včetně stavů komponent a jejich props. Vývojáři přišli také s vlastními doplňky. Například pro zkoumání lokálního uložiště zvaného redux store, se používá doplněk redux devtools. Což je také jeden z doplňků, které v této sekci popíši blíže. Vývojová prostředí pro mobilní aplikace obsahují podobné nástroje, které usnadní programátorům vývoj. Problém ale nastává v případě, že programátor programuje hybridní mobilní aplikaci a nepoužívá IDE (aplikaci pro vývoj software), které slouží pro vývoj mobilních aplikací. Ještě hůř se zkoumá stav aplikace v případě, že se v mobilní aplikaci používá WebView, kde je vykreslována webová stránka.

Při vývoji mobilní aplikace FLOWIO jsem narazil na stejný problém a proto bych tuto sekci chtěl věnovat nástrojům, které jsem při vývoji používal. Nebudu zde popisovat nástroje jako základní vývojářské nástroje v Google Chrome nebo v Safari a ani vývojářské nástroje z Android Studio nebo XCode. Budu se zde věnovat spíše doplňkům od uživatelů třetích stran a jejich konfiguraci.

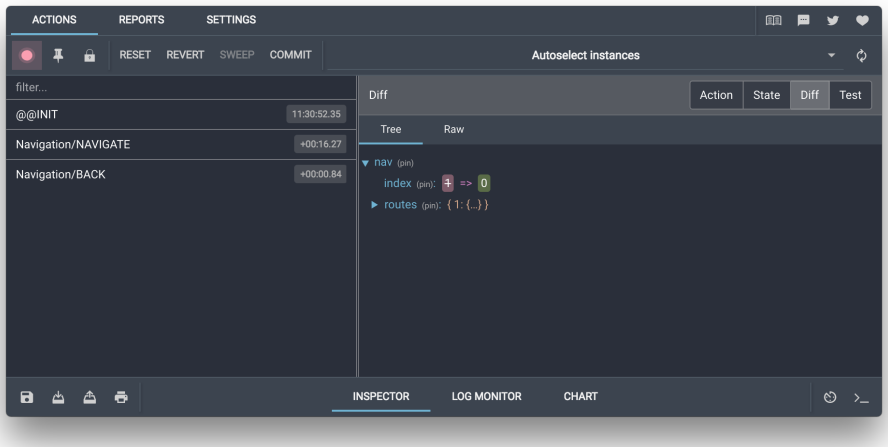
4.5.1 Redux DevTools

Redux DevTools jsem nepoužíval zrovna pro vývoj mobilní aplikace zde, ale jelikož je značná část mobilní aplikace zobrazena pomocí WebView, tedy jako webová aplikace, přišlo mi vhodné tento nástroj zmínit, jelikož za jeho pomoci vznikl celý front-end webové aplikace.

Jedná se o vývojářský nástroj, který podporuje vývoj lokálního uložiště zvaného Redux a usnadňuje vývoj s jakoukoliv architekturou. Tento nástroj sleduje změny stavu a přehledně zobrazuje veškeré části lokálního uložiště pomocí grafického rozhraní. Tento doplněk lze použít v prohlížečích jako jsou Chrome, Edge nebo Firefox, jako samostanou aplikaci nebo jako React komponentu integrovanou do klientovy aplikace. Pro zobrazení v Google Chrome je třeba si stáhnout doplněk z Google Chrome store v sekci rozšíření. Tím dostaneme DevTools do globální proměnné `window`, v případě že prohlížeč detekuje webovou aplikaci psanou v React JS. Dále je zapotřebí přidat nástroj do aplikace. Jestli aplikace používá základní konfiguraci Redux Store (lokální uložiště aplikace), stačí pouze při vytváření uložiště přidat následující kus kódu.

```
1 const store = createStore(
2    reducer, window.__REDUX_DEVTOOLS_EXTENSION__ &&    window.
     __REDUX_DEVTOOLS_EXTENSION__()
3    );
```

Jestli se jedná o aplikaci, která při vytváření uložiště používá tak zvané middlewary, je potřeba přidání Redux DevTools následovně upravit.



Obrázek 4.6: Grafické rozhraní doplňku Redux DevTools

```
const composeEnhancers =    window.
    __REDUX_DEVTOOLS_EXTENSION_COMPOSE__ || compose;

const store = createStore(
  reducer,
  composeEnhancers(
        applyMiddleware(...middleware)
    ));
```

V obou dvou případech se pomocí vlastností proměnné `window` odkazujících na Redux DevTools vytvoří middleware, který přijímá lokální uložiště a napojí ho na doplněk v prohlížeči. Poté stačí při prohlížení webové aplikace v prohlížeči zvolit v vývojových nástrojích záložku s názvem `redux`, kde budou přehledně vidět vlastnosti lokálního uložiště. [41]

4.5.2 React Native Debugger

Nyní už se opravdu dostávám k nástrojům pro mobilní vývoj. Tento nástroj vychází z nástroju React DevTools a je vytvořen stejnými vývojáři. Jelikož se ale při mobilním vývoji nepoužívá prohlížeč, je React Native Debugger samostatná aplikace. Stažení aplikace je možné v GitHub repozitáři pod jménem aplikace. Jsou dostupné verze pro Windows i pro macOS. Co je potřeba mít na paměti jsou verze aplikace a verze knihovny react-native. Integrace lokálního uložiště do aplikace probíhá stejným způsobem jako v případě Redux DevTools v sekci 4.5.1. Výhodou React Native Debugger je například přítomnost konzole, ze které můžete ručně mazat asynchronní lokální uložiště, ve kterém jsou data persistentní po celou dobu existence aplikace. Pro zobrazení uložiště mo-

React Native Debugger	react-native
>= 0.11	>= 0.62
<= 0.10	<= 0.61

Obrázek 4.7: Potřebné verze react-native-debugger aplikace a react-native

bilní aplikace v aplikaci React Native Debugger je zapotřebí mít spuštěnou mobilní aplikaci z konzole. Většinou je při vývoji mobilní aplikace v React Native v konzoli možné provádět operace jako znovunačtení nebo debug mód. Mobilní aplikace napojí svoje uložiště na aplikaci React Native Debugger pouze v případě když je v debug módu. Proto je nutné zvolit v konzoli debug mód a v simulátoru nebo na fyzickém zařízení, v závislosti na čem je aplikace testována, zvolit možnost `Debug JS Remotely`. Tím dáváme souhlas aplikacím třetích stran pro připojení k vyvíjené aplikaci pomocí portu. React Native Debugger běží na portu 8081. Tento port je výchozí a v případě, že na stejném portu poběží jiná aplikace nebo nástroj pro ladění chyb, nedojde k propojení lokálního uložiště mobilní aplikace a aplikace React Native Debugger. Aplikaci React Native Debugger v takovém případě můžeme pustit z konzole s definovaným vlastním portem pomocí následujícího příkazu.[42]

```
open "rndebugger://set-debugger-loc?host=localhost&port=8081"
```

Tento příkaz je možné nastavit do proměnné prostředí a spustit automaticky aplikaci React Native Debugger se spuštěním vyvíjené mobilní aplikace.

```
REACT_DEBUGGER="open -g 'rndebugger://set-debugger-loc?port=8001'
    ||"  react-native start
```

4.5.3 Reactotron

Reactotron je samostatné aplikace, která slouží ke stejnému účelu jako Redux DevTools a Redux Native Debugger. Kromě toho, že lze pomocí Reactotronu zkoumat lokální uložiště, tak umí sledovat jednotlivé dotazy na server a jeho odpovědi. Tuto funkcionalitu poskytuje také Google Chrome, ale v případě vývoje mobilní aplikace je potřeba hledat alternativu. Ideální alternativou v mém případě byla zrovna aplikace Reactotron. Konfigurace je velmi jednoduchá, kde není potřeba nijak zasahovat do vyvíjeného kódu, ale stačí pouze přidat vývojovou závislost (dev depedency).

Nejdříve je potřeba stáhnout si aplikaci z github uložiště. Tvůrci Reactotronu poskytují verze pro Windows, macOS i pro Linux operační systémy. Poté je potřeba stáhnout do projektu závislost pomocí npm nebo yarn z příkazové řádky. K tomu slouží příkaz

```
\\npm  ---->  npm i --save-dev reactotron-react-native
\\yarn  ---->  yarn add --dev reactotron-react-native
```

Po instalaci je možné konfiguraci vložit přímo do souboru `index.js` nebo do `App.js` a nebo konfiguraci pro přehlednost oddělit do samostatného souboru. Využil jsem možnost konfiguraci oddělit a vytvořil jsem soubor `ReactotronConfig.js`, do kterého jsem vložil následující kód.

```
import Reactotron from 'reactotron-react-native'

Reactotron
  .setAsyncStorageHandler(AsyncStorage) // nastaveni lokalniho
    uloziste
  .configure() // kontroluje pripojeni a komunikacni nastaveni
  .useReactNative() // prida vesktere vestavene doplnky pro React
     Native
  .connect() // finalni vytvoreni spojeni s aplikaci
```

Pak už stačí jenom pustit aplikaci Reactotron, spustit vyvíjenou aplikaci v debug módu a veškerá síťová komunikace bude zobrazena v aplikaci Reactotron. V případě, že je aplikace testována na zařízení android nebo na android emulátoru, je potřeba spustit následující příkaz k ujištění, že se správně připojí na aplikaci Reactotron.[43]

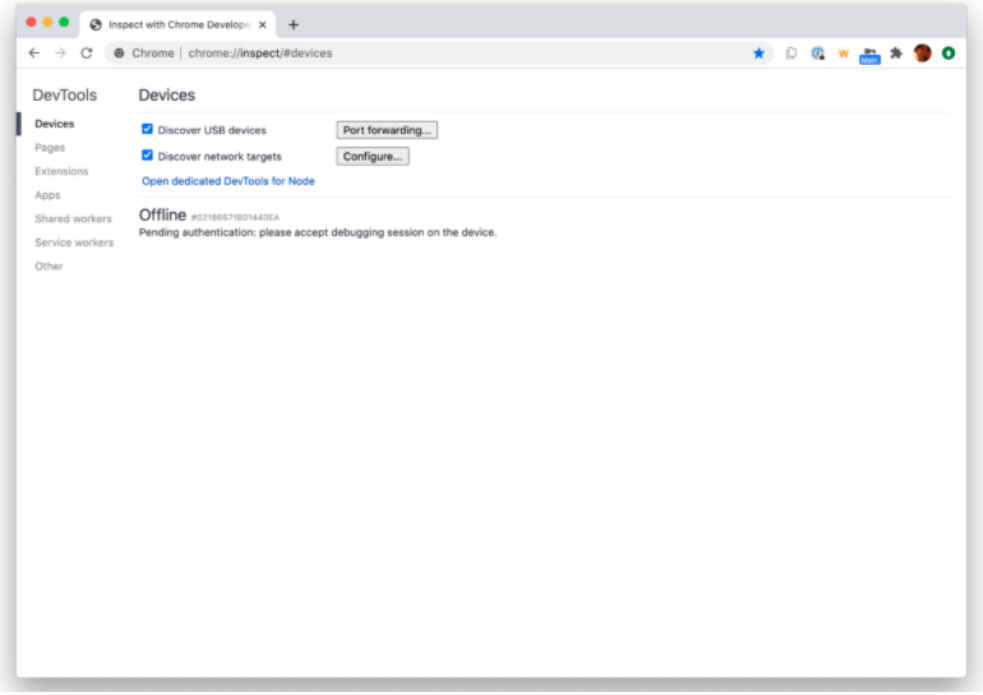
```
adb reverse tcp:9090 tcp:9090
```

4.5.4 Ladění WebView obsahu zobrazovaném v mobilním zařízení

Už jsem zde zmínil nástroje pro zkoumání lokálního uložiště, strukturu DOM elementu i nástroje pro zkoumání síťových akcí. Tato práce se zabývá ale konceptem WebView a to je zobrazení webová aplikace v mobilní aplikaci. Zkoumat webovou aplikaci z prohlížeče je jednoduché s pomocí vývojových nástrojů které poskytují prohlížeče. WebView se ale jako prohlížeč chová jen aby zobrazilo obsah webové stránky v mobilní aplikaci a proto není možné zkoumat prvky klasickým způsobem. S řešením tohoto problému přišla opět společnost Google. Přišli s nástrojem který umožňuje zkoumat webové prvky v mobilní aplikaci. Tento nástroj slouží primárně pro zkoumání webových

aplikací spouštěných z prohlížeče v mobilním zařízení. WebView je ale svým způsobem také takový webový prohlížeč a proto je možné obsah WebView zkoumat právě tímto způsobem. Jediným problémem je to, že nelze takto zkoumat prvky v iOS aplikaci, ale pouze na Android zařízeních. K tomu,

aby tento způsob ladění obsahu WebView mohl fungovat, je zapotřebí mít připojené mobilní zařízení, na kterém běží vyvíjená aplikace, pomocí USB kabelu do počítače. Díky tomu se stává mobilní zařízení zjistitelné pro účely ladění. Dále je potřeba mít zapnutý debug mód, aby bylo možné zkoumat jednotlivé prvky aplikace. Poté stačí v prohlížeči na počítači jít na stránku `chrome://inspect/#devices`, která v případě, že byly splněny kroky zmíněné výše, bude vypadat jako na obrázku 4.8. Na obrázku lze vidět nápis `offline`



Obrázek 4.8: Vzhled stránky pro vzdálené ladění mobilních aplikací - první přístup

s sériovým číslem nalezeného zařízení. Na zařízení pak stačí povolit přístup a dojde k automatickému připojení. Připojené zařízení se zobrazí v seznamu dostupných zařízení a bude možné zkoumat webové prvky na tomto zařízení.

V případě, že nástroj detekuje WebView v aplikaci, zobrazí automaticky možnost průzkumu stránky. V případě, že počáteční stránka je nativní, je možné se na WebView dostat zadáním URL stránky, na které se WebView zobrazí. Vzhled stránky v případě, že není detekováno WebView automaticky lze vidět na obrázku 4.9.

Obrázek 4.9: Vzhled stránky pro vzdálené ladění mobilních aplikací - připojené zařízení

4.6 Problémy, se kterými jsem se v průběhu implementace potýkal

Díky dobré analýze, dobrému vymezení požadavků na mobilní aplikaci a především, dobře postavené architektuře webové aplikace, byla implementace mobilní aplikace FLOWIO pomocí konceptu WebView vcelku přímočará a až na pár problémů probíhala hladce. I přes to, se v průběhu vývoje vyskytly určité překážky a problémy, které během implementace a při uživatelském testování vyplavaly na povrch. V další kapitole se budu věnovat celkovému výsledku aplikace a uživatelskému testování, v této části bych ale rád zmínil problémy, na které jsem při vývoji narazil.

4.6.1 Zkoumání dotazů na server volaných z obsahu WebView

V sekci výše, jsem popisoval nástroje použité pro zkoumání stavu mobilní i webové aplikace. Podařilo se mi nalézt nástroje pro zkoumání serverových dotazů z nativní částí aplikace. Dokonce jsem nalezl i způsob, jak zkoumat obsah, který zobrazuje WebView. Kromě struktury DOM elementů webové aplikace ve WebView, uložených cookies, obsahu localstorage a možnosti používat příkazovou řádku, nebylo v nástroji popisovaném v sekci 4.5.4 nic jiného. Chyběla základní funkcionalita a to pozorování dotazů na back-end aplikace, které byly volány z webové části aplikace zobrazované pomocí WebView. Tyto

akce jsem mohl pohodlně pozorovat z klasického zobrazení v prohlížeči a to pro většinu případů bohatě stačilo. Problém ale nastával v případě, že se nějaká akce prováděla pouze v případě zobrazení ve WebView. Jedním z těchto případů byl například dotaz, který slouží k získání počtu notifikací pro jednotlivé šablony v aplikaci. V případě chyby, jsem musel příčinu zjišťovat klasickými výpisy z aplikace do konzole a umisťováním krokovacích zarážek neboli jinak známých jako *debugging breakpoints*.

4.6.2 Testování nově vyvíjených funkcionalit ve webové aplikaci pro komunikaci s WebView

WebView v mobilní aplikaci zobrazovalo obsah ze zadané URL, kterou měla aplikace uložena v lokálním uložišti. Tato adresa odkazuje na webovou aplikaci, která je veřejně přístupná pomocí internetového prohlížeče. Tato url adresa odkazovala vždy na jedno prostředí aplikace FLOWIO podle potřeby vývojáře. Mám tím na mysli testovací prostředí, vývojové prostředí nebo produkční prostředí. Tyto prostředí jsou součástí cyklu každé nové verze webové aplikace, kterou vývojář vyvinul a byla schválena vedoucím programátorem. V případě vyvíjení funkcionalit pro mobilní aplikaci, které souvisely s WebView a webovou aplikací, bylo zapotřebí nově vyvinutý kód co nejrychleji nasadit na jedno ze zmíněných prostředí, aby bylo možné funkcionalitu otestovat proti mobilní aplikaci a WebView. Ideální případ by byla přítomnost nějakého prostředí, kam by vývojář pracující na nových funkcionalitách mohl nahrávat nový kód samostatně a změny by byly projeveny na aplikaci FLOWIO automaticky. Takové prostředí ovšem nebylo dostupné z hlediska finančních nákladů a hlavní programátor, který se staral o kontrolu nových funkcionalit pro nasazení na jedno z veřejných prostředí, neměl dostatečnou časovou kapacitu na veškeré žádosti. Jediným způsobem zde bylo využívat pro tyto účely lokální vývojovou verzi webové aplikace. Ta ovšem potřebovala k tomu, aby byla přístupná přes WebView, lokálně bežící server aplikace.

Tento problém se mi povedlo vyřešit pro operační systémy Windows. Nainstaloval jsem lokální kopii back-endu aplikace, který při spuštění přímo spuštěl i front-end aplikace, ke kterému se mohlo WebView po spuštění v mobilní aplikaci lokálně připojit. Tento proces byl ale vždy zdlouhavý a velmi náročný na výpočetní sílu počítače. Je to ale jeden způsob, jak v rámci možností efektivně vyvíjet komunikaci mezi WebView a webovou aplikací.

Větší problém nastal však při snaze použít stejný postup na počítačích s operačním systémem macOS pro testování iOS verze aplikace. V některých případech se určité prvky chovají jinak v iOS aplikaci a proto bylo zapotřebí testovat vývoj pro obě dvě zařízení. Z toho důvodu, že je back-end aplikace FLOWIO napsaný v jazyce .NET, což je framework vydaný firmou Microsoft a na operačním systému macOS se nám nepodařilo lokální server spustit.

Jediným způsobem bylo se připojit z mobilní aplikace spuštěné na macOS na lokální server, který byl spuštěný na počítači s Windows systémem. K tomu, abychom toho docílili, jsme museli připojit počítač s macOS na stejnou síť, kterou používal počítač se spuštěným serverem a nakonfigurovat jeho síť tak, aby se k němu mohlo připojit jiné zařízení. Tento postup byl daleko zdlouhavější a náročnější než pro Android aplikaci a i přes to, že jsme tímto způsobem dokázali určité funkcionality otestovat, se tento způsob vývoje jeví jako velmi neefektivní.

4.6.3 Nekonzistence ukládání cookies pro WebView na iOS a Android zařízeních

Tento problém se ani tak netýká WebView jako spíš prohlížečů, pomocí kterých WebView zobrazuje svůj obsah. To záleží samozřejmě na tom, jestli zařízení používá operační systém iOS nebo Android. V případě Android, je výchozím prohlížečem, který WebView používá Google Chrome. V případě iOS to je zase Safari. Na tento problém jsme narazili při zobrazování komponenty, která má za úkol představit aplikaci nově registrovanému uživateli. Je to komponenta, která je součástí webové aplikace a není nijak ovlivněna samotným WebView, proto jsem ji v práci nijak nepopisoval. V krátkosti její funkcionalita byla taková, že při prvním příchodu uživatele do aplikace se tato komponenta zobrazila a zeptala se jestli má uživatel zájem o představení aplikace. V případě zájmu, následoval krátký průvodce, který vysvětlil ty důležité aspekty aplikace. V případě nezájmu, stačilo zmáčknout tlačítko pro zrušení. V každém případě, po dokončení průvodce nebo po jeho okamžitém vypnutí, se vytvořila cookie, která signalizovala to, že uživatel už průvodce viděl. Tím pádem, se při dalším přihlášení nebo přenačtení stránky, tento průvodce již nezobrazoval.

Problém byl v tom, že u cookie se nastavuje její životnost. Pokud má danou životnost nějakým datem, trvá do toho zadaného data. Když není explicitně zadaná životnost cookie datem, nastaví se výchozí hodnota `session`. Znamená to, že cookie bude vytvořena po dobu otevření prohlížeče. Po jeho zavření se cookie smaže. Přesně takovou životnost měla cookie pro průvodce. Zde se ale projevilo rozdílné nastavení jednotlivých prohlížečů. Prohlížeč Chrome, má ve výchozím nastavení, že při spuštění aplikace, začne tam kde uživatel předtím skončil. To znamená, že nejen že bude mít uživatel k dispozici stránky, na kterých byl před zavřením, ale i cookie která má životnost po dobu `session` bude stále přítomna. Safari má základní nastavení takové, že při zavření prohlížeče se cookies s dobou expirace po dobu session opravdu vymažou. Tento problém samozřejmě existuje i ve webové verzi aplikaci FLOWIO v Safari prohlížeči. Tam se ale nastavení může změnit. WebView však bere pokaždé to základní nastavení aplikace a není možné ho nijak konfigurovat. Proto jsme na tuto chybu přišli díky ladění mobilní aplikace na iOS zařízení. Tento problém jsme opravili tak, že jsme při vytváření cookie nastavili její ex-

piraci na jeden rok. Mohlo by se to ale řešit i jinými způsoby, například ukládat takovou informaci na server nebo do persistentního lokálního uložiště.

4.6.4 Chybná navigace z nativní části do webové části

K uchovávání historie a k navigaci slouží v každé části aplikace jiná react knihovna. V případě webové aplikace k navigaci a uchování historie slouží knihovna React Router a v nativní části zase knihovna React Navigation. Nativní navigace se ve FLOWIO aplikace používá do té doby, než se uživatel dostane na obrazovku, kde začíná WebView a vykresluje webovou část. Tam přebírá navigaci webová část. Ve WebView funguje navigace tak jak má a chování je identické jako chování ve webové aplikaci. Zde jsme se ale setkali s problémem, že když uživatel v mobilní aplikaci zobrazil seznam notifikací a kliknutím na jednu z notifikací se dostal na její detail, kde je přítomno tlačítko zpět pro vrácení o jeden krok v historii dozadu. Když uživatel kliknul na toto tlačítko jednou, tak se nic nestalo a až když kliknul na tlačítko podruhé, dostal se zpět na stránku ze které šel na detail notifikace. Příčina chyby spočívala v tom, že seznam notifikací v mobilní aplikaci je vlastně seznam zobrazený v nativní části kódu. Kliknutím na notifikaci se vygenerovalo URL detail notifikace na základě dostupných metadat a do lokálního uložiště se nastavila nová URL, kterou má WebView zobrazit. Tím, že se změnila zobrazovaná URL došlo k přenačtení aplikace na novou adresu, na adresu kde je zobrazen detail notifikace. Očekávané chování bylo, že přibude nový záznam do historie a na URL, které bylo zobrazené jako poslední, se dostaneme jedním krokem v historii zpět. Přenačtení aplikace ale způsobilo, že FLOWIO aplikace nejdříve zkusila uživatele přihlásit a po zjištění, že uživatel už přihlášený je, se přesměrovala na stránku se zadaným URL. Tím pádem byly v historii nové dva záznamy, kde jeden byla URL stránky pro přihlášení a druhý URL s stránkou pro detail notifikace. Tím pádem bylo potřeba dvakrát kliknout na tlačítko zpět pro dvojitý posun v historii zpět. Toto chování bylo způsobeno tím, že WebView přímo modifikovalo URL a způsobilo přenačtení stránky. Když bychom stránku měnili pomocí akce, která je dostupná z knihovny React Router, která slouží k navigaci mezi stránkami ve webové aplikaci, nedošlo by l přenačtení celé aplikace, ale pouze ke změně zobrazované stránky. Tím pádem by v historii byl pouze jeden nový záznam a bylo by možné se vrátit tlačítkem zpět.

Problém jsem vyřešil tak, že jsem neměnil stránku z nativní části pomocí WebView, ale poslal jsem zprávu webové části, aby se tato změna provedla přes akce knihovny React Router. K tomu jsem využil stejnou techniku, kterou jsem používal pro posílání aktuálního počtu notifikací. Vytvořil jsem si ve webové části posluchače události, který čekal na událost s názvem `webViewDispatch`. V této události byla zpráva o tom, že je potřeba přidat nový záznam do historie spolu s URL adresou detailu notifikace. Webová část tuto zprávu zachytila

a přidala nový záznam do historie ve webové části. To způsobilo přenačtení WebView obsahu a zobrazení požadovaného detailu. Při stisku tlačítka zpět, je uživatel vrácen na stránku, ze které na detail přišel.

Tato chyba byla z velké části způsobena logikou ve webové aplikaci, která po každém načtení aplikace kontroluje přihlášení uživatele na jiné URL adrese. To ale nemění nic na tom, že by navigování v obsahu WebView mělo být integritně zachované pouze ve webové části. WebView by pomocí vlastnosti `source` mělo zobrazit pouze počáteční stránku webového obsahu a více tuto adresu neupravovat.

4.6.5 Stejné datové typy na dvou různých vývojových prostředích

Tento problém není ani tak problém, jako spíše taková uživatelská nepohodlnost při vývoji. Jde o to, že pro komunikaci mezi WebView a webovou částí se používá pár metod a to `postMessage` pro posílání dat z webové části, `onMessage` pro příjímání dat z webové části v nativní části, `injectJavascript` metoda pro posílání dat z nativní části do webová části a vlastnost komponenty WebView `injectedJavaScript`, které slouží pro nastavení dat do webové části. Tyto metody si navzájem posílají data ve formě komprimovaných stringů. Většina dat má v sobě uložené různé typy zpráv, podle čeho části určují, jakou akci mají provést. Tato posílaná data musí mít ale na obou dvou prostředích stejný formát, aby bylo možné je zpracovávat. O to, aby tento formát byl zachován se musí starat sám vývojář, kvůli rozdílným prostředím zde IDE nepomůže a proto se může velmi jednoduše stát, že se programátor přehlédne nebo upíše a aplikace kvůli tomu nebude fungovat. Oprava je zde velmi jednoduchá, ale dochází díky tomu k časovým prodlevám. V případě metody `injectJavaScript` to je ještě složitější, jelikož v posílaném stringu musí být validní javascriptový kód. Tím, že je kód psaný ve stringu, může velmi lehce dojít ke špatné syntaxi a k nefunkčnosti výsledného kódu. To, že je daný kód syntakticky špatně napsaný a nebo nedělá přesně to co si vývojář usmyslel, se vývojář dozví až při spuštění aplikace, což také není úplně časově výhodné viz sekce 4.6.2.

Kapitola 5

Zhodnocení výsledku a porovnání zvoleného přístupu s nativním vývojem

Mobilní aplikace FLOWIO má po čtyřech měsících vývoje svoji první verzi a tím pádem i první uživatelské testování v rámci interního provozu společnosti. Proto bych se v této kapitole rád ohlédnul za zvoleným postupem a zhodnotil výsledek jako takový i samotný postup. Dále bych pak rád porovnal zvolený postup s klasickým nativním vývojem aplikace.

5.1 Zhodnocení mobilní aplikace FLOWIO po vydání první verze

Po implementaci jednotlivých případů užití pomocí konceptu WebView a následnému testování na zařízeních s operačním systémem Android a iOS, byla mobilní aplikace připravena k internímu testování v rámci společnosti Popron Systems s.r.o.. Výsledná aplikace je dostupná prostřednictvím obchodu s aplikacemi jako je App Store a Google Play, kde je možné si první verzi aplikace FLOWIO zdarma do zařízení stáhnout. Z toho jasně vyplývá, že se jedná o validní mobilní aplikaci. Co se týče ale samotné konzistence aplikace, je tomu jinak. Aplikace se ve skutečnosti skládá z nativní části, komponenty WebView, která zobrazuje webovou část a celé webové části. Když bych měl procentuálně vyjádřit poměr celkového nativního kódu a klasického kódu pro webové aplikace, vyšlo by mi že pouze 10% celkového kódu je nativní kód, který slouží jako kontejner s určitými nativním prvky a zbylých 90% je využitá webová aplikace a její kód. Výsledek bych rád zhodnotil z vlastního pohledu a pak také z pohledu uživatelů, kteří aplikaci používali v rámci interního provozu společnosti.

5.1.1 Vlastní pohled na vývoj aplikace a na výslednou formu její první verze

Cílem práce bylo co nejméně měnit kód webové aplikace a toho jsem se také v průběhu vývoje držel. Požadavky na mobilní aplikaci tento cíl značně ulehčily, jelikož hlavním důvodem, proč nemohla být formou WebView vykreslena celá aplikace, byla možnost přepínání a přidávání různých datových zdrojů pro zobrazovaní WebView obsahu. Nebýt toho, tak by se i největší část implementace týkající se notifikací prováděla v rámci WebView a jediné co by bylo zapotřebí vyřešit, by bylo zpracování push notifikací a jejich interakce s uživatelem. Dokážu si ale představit, že když by byly požadavky na mobilní aplikaci složitější a architektura webová aplikace byla postavená tak, že se celá nedala využít k zobrazení ve WebView, byla by integrace WebView, nativní části a webové části značně složitější a nejsem si jistý, zda by byla vůbec možná.

Hlavním problémem je zcela jistě to, že knihovna `react-native-webview` je výtvorem programátorů, jenž knihovnu vytvořili kvůli vlastním potřebám a ve svém volném čase. Tím pádem vydávání nových verzí knihovny a uživatelská podpora závisí na volném času jejích vývojářů. Knihovna slouží především pro vykreslování jednoduchých webových stránek a k tomu je uzpůsoben i její interface. V průběhu práce jsem zjistil, že je možné tento způsob využít i k vykreslení složitější webové aplikace a výsledná mobilní aplikace FLOWIO je toho také důkazem. V oficiální dokumentaci knihovny je hlavní důraz kladen především na vlastnosti komponenty WebView, jako jsou `source` a `injectedJavaScript`, kde vlastnost `source` slouží k vykreslení obsahu WebView a vlatnost `injectedJavaScript` slouží k nastavení zobrazované webové stránky před prvním vykreslením. Tyto vlastnosti jsem v konfiguraci pro mobilní aplikaci použil pouze jako základní definici a i když jsou to jsou stěžejní vlastnosti komponenty tak na celkovém výsledku se podílely pouze svojí základní funkčností. Mnohem větší význam byl práci kladen na metody `onMessage`, `postMessage` a `injectJavaScript`, které sloužily ke komunikaci webové části s nativní částí. To bylo určitě hodně dáno účelem mobilní aplikace, ale i přes to jsou za mě tyto tři metody nejdůležitějšími funkcemi celé knihovny a měly by, kromě základní definice jejich rozhraní, být více popsány. Kvůli tomu, že jsem nemohl těžit z ukázek použití těchto metod, jsem se musel spolehnout na vlastní vymyšlená řešení. Komunikace probíhá výměnou stringů. V takovém stringu může být definované cokoliv co si vývojář zamane, jen musí zvolit správný způsob komprese a dekomprese dat uložených v stringu. Tento způsob komunikace bezchybně funguje a vývojáři na tom odvedli skvělou práci. Ve větší aplikaci kde bude potřeba vyměňovat více dat může nastat však problém v nepřehlednosti. Jednak z toho důvodu, který jsem popisoval v sekci 4.6.5, tak také proto, že metody zpracovávající tyto zprávy na jednotlivých prostředích mohou nabýt robustních rozměrů. V případě nativní části jsou tyto zprávy zpracovávány

metodou `onMessage`, kde se akce volají na základě zprávy, poslané z webové části, podle kondicionální logiky. V případě aplikace FLOWIO se jedná o pár zpráv a celá funkce nezabrala více než pár řádků. Když by ale aplikace záležela více na vzájemné komunikaci jednotlivých prostředí, stal by se zápis zdlouhavým a velmi nepřehledným.

V případě mobilní aplikace FLOWIO, se ale zvolený postup osvědčil a čtyři měsíce stačily k vytvoření funkční mobilní aplikace, schopné první pilotní verze. Mohu říci, že jsem soustavně s vývojem mobilní aplikace řešil i vývoj aplikace webové, tím pádem, by zkušenější vývojář, se zaměřením pouze na vývoj mobilní aplikace, s využitím poznatků získaných z této práce, dokázal vydat první verzi i mnohem rychleji. Všechny požadované funkcionality byly implementovány a grafické rozhraní mobilní aplikace bylo přizpůsobeno grafickému rozhraní webové aplikace, tím pádem i obsahu zobrazovanému ve WebView. Výsledek byl předán k internímu testování ve společnosti Popron Systems s.r.o., kde aplikaci měli testovat zaměstnanci, na bázi klasického uživatelského používání.

5.1.2 Výsledek uživatelského používání mobilní aplikace FLOWIO

Po týdenním používání aplikace byla pozitivní zprávou skutečnost, že z uživatelského hlediska nebylo poznat, že se nejedná ryze o nativní mobilní aplikaci. Jedním z hlavních cílů práce bylo dodat funkční mobilní aplikaci, která se na první pohled a při používání bude tvářit jako mobilní aplikace. Testování se účastnilo deset zaměstnanců společnosti a všichni do jednoho se shodli na tom, že se jedná čistě o mobilní aplikaci. Jedním důvodem pro takto kladný výsledek z hlediska uživatelské zkušenosti mohl být fakt, že ani jeden z testerů nebyl vývojářem nativních mobilních aplikací. Uživatelé byli dále spokojený s návrhem a jednoduchostí mobilní aplikace. O tuto skutečnost se postaral správný návrh webové aplikace a její responzivní implementace. Uživatelé se ale setkali s jedním nepříjemným prvkem, který by jim dával větší smysl, když by jim bylo známo rozhraní mobilní aplikace. Touto nepříjemností byla rychlost aplikace. Přihlašování, výběr serveru, změna serveru a listování v seznamu notifikací fungovalo z pohledu rychlosti zcela bezproblémově, to bylo ale dáno tím, že se jednalo o nativní prvky aplikace. Jelikož je WebView ve skutečnosti prohlížeč, který zobrazuje webovou stránku, závisí rychlost operací na kvalitě připojení zařízení k síti. V případě nejrychlejšího připojení, kterým je 4G, je rychlost načítání jednotlivých stránek adekvátní. Stejně tomu tak je při kvalitním připojení na Wi-Fi. V případě horšího připojení vznikají i větší prodlevy mezi načítáním jednotlivých stránek. V případě, že dojde k výpadku sítě je uživatel z aplikace automaticky odhlášen a přemístěn na první stránku aplikace, což není z hlediska mobilních aplikací zcela uživatelsky přívětivé chování. Tím, že je většina aplikace zobrazena pomocí WebView, mají na rychlost do-

pad také nativní gesta, jako je například krok zpět přetažením prstu zprava doleva na zařízeních s operačním systémem iOS. Rychlost připojení nelze nijak ovlivňovat pomocí kódu , ale obsah dat a načítání dat ze serveru lze optimalizovat a bude se na tom pracovat v dalších verzích aplikace.

Celkově při používání aplikace narazili uživatelé na řadu chyb, které se týkaly jak uživatelského dojmu z aplikace, tak samotné funkčnosti aplikace a jejích jednotlivých částí. Tyto chyby ale všechny spadaly do webové části, která je pomocí WebView pouze zobrazována. Tyto chyby nijak nesouvisí s implementovanými funkcionalitami a nativní aplikací, proto se těmito chybami dále nebudu zabývat a budou opraveny v rámci vývoje webové aplikace. Prvky vyvíjené pomocí konceptu WebView, se ukázaly být jako velmi funkční, stabilní a nebyly při testování zjištěny žádné závažné chyby, které by znamenaly nesplnění zadaného úkolu.

5.2 Porovnání zvoleného postupu s klasickým nativním vývojem

K implementaci mobilní aplikace FLOWIO, jsem se rozhodl využít konceptu WebView z toho důvodu, že bylo zapotřebí v krátkém čase a s omezenými finančními zdroji vyvinout funkční mobilní aplikaci, k podpoření existující webové aplikace. Toto řešení mělo být pro mobilní aplikaci FLOWIO dočasné, jelikož v případě úspěchu aplikace na trhu a při nárustu poptávky po aplikaci, by bylo dostatek finančních prostředků a bylo by reálné vyvinout ryze mobilní aplikaci. Na začátku práce, jsem v analytické části popisoval různé metody vývoje mobilních aplikací, ze kterých mi nejlépe pro řešení zadaného problému s omezujícími podmínkami vyšla kombinace WebView konceptu a programovacího jazyku React Native. Nyní po dokončení první stabilní verze aplikace, se mohu za vývojem ohlédnout a porovnat zvolený postup s klasickým vývojem nativní aplikace a určit, jestli jsem se na začátku práce rozhodl správně či nikoliv. Kvůli tomu, že jsem nenašel žádné podklady k tomu, jak vyvíjet aplikaci takového rozměru pomocí WebView, jsem také nemohl přesně určit, jak náročné to bude uskutečnit. Klasických postupů pro vývoj nativních mobilních aplikací je spoustu a proto nebude těžké porovnat vývoj mobilní aplikace FLOWIO za použití nativního programovacího jazyka s konceptem WebView použitým v práci.

Pro porovnání jsem se rozhodl použít jazyk React Native. Tento jazyk už jsem zde v práci zmiňoval a částečně je pomocí toho jazyka naprogramovaná i mobilní aplikace FLOWIO. React Native je z rodiny programovacích jazyků, které slouží k programování cross-platformních mobilních aplikací. To znamená, že jeden kód stačí pro vývoj mobilní aplikace na iOS i na Android. Když bych k porovnání použil čistě nativní jazyky jako jsou Kotlin, Java pro

Android a nebo Swift, Objective-C pro iOS, tak by byly nákladové hodnoty dvakrát vyšší, pro dosažení výsledku, kterého je možné dosáhnout za použití React Native. Další výhodou je to, že React Native vychází z knihovny React a syntax mají velmi podobnou. Díky tomu se může využít stejné architektura jako pro webovou aplikaci, což znamená další zachování času a finančních prostředků.

V porovnání budu z hlediska správnějšího vyhodnocení uvažovat pouze jednoho vývojáře pracujícího na projektu, jelikož i implementace mobilní aplikace FLOWIO, je dílem jednoho vývojáře. Dílem mobilní aplikace mám na mysli nativní část mobilní aplikace a komunikaci WebView s webovou částí, implementace webové části do porovnání nespadá. Informace z nativní části porovnání nemusí být úplně přesné, jelikož sám nejsem vývojář nativních aplikací a ani nikoho takového neznám. Informace jsou výsledkem nashromáždění dat z dostupných internetových zdrojů.

5.2.1 Časová náročnost vývoje aplikace

Rychlost vývoje byla jednou ze dvou hlavních podmínek, které byly na mobilní aplikaci FLOWIO kladeny. V této oblasti porovnání budu vycházet ze stavu, kdy existuje funkční webová aplikace, která může být zobrazena pomocí WebView a z jejíž architektury může vývojář v React Native vycházet. Stejně tak existuje i dostupné API z back-end části aplikace.

WebView

Vývoj jako takový, včetně nativní části aplikace, trval jednomu vývojáři, který neměl žádné zkušenosti s React Native ani s WebView, 4 měsíce. Nutno uvážit, že vývojář nepracoval jenom a pouze na vývoji mobilní aplikace, ale souběžně i na vývoji webové aplikace. Dalo by se tedy říci, že by jeden vývojář, zaměřený čistě jenom na vývoj mobilní aplikace, dokázal dodat produkt i dříve než za 4 měsíce.

React Native

Dle internetových zdrojů, může vývoj nativní mobilní aplikace s jasným plánem, návrhem a dostupným back-endem, jednomu vývojáři trvat v rozmezí od 3 týdnů do tří měsíců, vzhledem ke komplexnosti aplikace [44] [45]. S ohledem na to, že by zkušený programátor mohl vycházet z již existující webové aplikace, řekněme že by k vývoji první verze potřeboval 2 měsíce.

5.2.2 Finanční náročnost vývoje aplikace

Finanční prostředky byly druhou brzdou na projektu, se kterou se musel vývoj mobilní aplikace FLOWIO potýkat.

WebView

Finanční náklady na vývoj mobilní aplikace z hlediska firmy Popron Systems byly nulové, jelikož vývojář, pracující na vývoji mobilní aplikace pomocí WebView, byl primárně zaměstnán jako front-end vývojář a soustavně s webovou aplikací vyvíjel i mobilní aplikaci pomocí konceptu WebView.

React Native

Podle portálu nofluffjobs.com, se platové ohodnocení React Native vývojáře pohybuje průměrně kolem 70 tisíc korun českých za měsíc [46]. Když uvážíme fakt, že by vývoj mobilní aplikace zkušenému programátorovi trval 2 měsíce, bylo by to 140 tisíc, které by společnost musela vyplatit. Připomínám, že se jedná pouze o dobu do první verze aplikace, neuvažuji případné další nové verze a funkcionality aplikace.

5.2.3 Škálovatelnost aplikace

Škálovatelnost nebo také rozšiřitelnost je v dnešní době velmi žádoucí vlastnost jakéhokoliv systému. Jedná se o schopnost efektivně řešit přidávání nových funkcionalit a tvarovat aplikaci dle potřeb koncového uživatele.

WebView

Tento způsob vývoje dobře posloužil v případě základních požadavků na mobilní aplikaci FLOWIO. S rostoucími uživatelskými požadavky budou růst i nároky na WebView a množství nativních funkcionalit, které WebView podporuje, je závislé na autorech konceptu. V případě, že dojde k ukončení podpory WebView, bude projekt nepoužitelný. Co se týče samotné škálovatelnosti a přidávání nových funkcionalit, tak by funkce pro zpracování komunikace mezi jednotlivými prostředími bobtnaly a brzy by se staly velmi nepřehledné. Proto je škálovatelnost pro zvolený postup velmi špatná.

React Native

Jelikož se jedná o nativní programovací jazyk a celková aplikace by byla vytvořena pomocí tohoto jazyka, byla by i škálovatelnost značně jednodušší. Je zde ovšem důležitý zvolený postup vývojáře a celková architektura aplikace, jelikož špatně postavená architektura může vést ke špatné škálovatelnosti. V každém případě bude škálovatelnost daleko lepší a pohodlnější než v případě WebView.

5.2.4 Vývojové prostředí

Vývojové prostředí může mít značný dopad na rychlost vývoje aplikace. Jelikož oba dva přístupy využívají React Native tak mohou využívat i stejných

vývojových prostředí. V tomto ohledu neexistují žádné rozdíly a výběr je pouze na vlastním uvážení vývojáře podle toho, na co je zvyklý a co mu vyhovuje.

5.2.5 Testování aplikace v průběhu vývoje

WebView

Jak už jsem zmiňoval v kapitole 4.5, tak existuje řada možností jak zkoumat a testovat vyvíjenou aplikaci za použití tohoto přístupu. Je tu ale jeden problém, který může značně znepříjemnit vývoj. Jedná se o testování změn provedených ve webové části, které komunikují s WebView a mají dopad na nativní část. Tato forma testování probíhá v pořádku i když poněkud zdlouhavě, na zařízeních s operačním systémem Android. V případě operačních systémů iOS je to ale daleko složitější a ne vždy to musí skončit zdárně. Tento problém značně znepříjemňuje testování vývoje na zařízeních s operačním systémem iOS.

React Native

Při klasickém nativním vývoji lze používat stejné nástroje k testování jako za použití konceptu WebView. Zde se ale nebude vývojář potýkat s problémem testování na více prostředích. Pro snadnější a rychlejší vývoj může zvolit na začátku přístup pomocí platformy Expo, což je veřejná platforma sloužící k vývoji univerzálních nativních aplikací pro Android, iOS a web za použití React Native. Tuto platformu nebylo možné využít v případě této práce přesně kvůli důvodům testování webového prostředí v rámci WebView.

5.2.6 Zhodnocení porovnání

V tabulce 5.1 jsem shrnul porovnání implementace mobilní aplikace FLOWIO za použití WebView a za použití React Native. Ukázalo se, že by vývoj aplikace za použití React Native trvalo kratší dobu, než použitá implementace. To ale ovšem závisí na faktu, že by společnost dokázala alokovat na projekt schopného a zkušeného React Native vývojáře. To se také ale odráží na finanční stránce. Vývoj mobilní aplikace s využitím WebView konceptu sice potrvá delší dobu, ale za to bude vývoj značně levnější. Na výsledku bychom pak mohli pozorovat poměr cena kvalita, jelikož za pomoci WebView byla vyvinuta funkční mobilní aplikace, která perfektně poslouží pro předváděcí účely produktu potencionálním investorům. V případě zájmu bude ale potřeba aplikaci přeci jenom přepsat do ryze nativní formy, jelikož je škálovatelnost WebView postupu velice špatná a hlavním uživatelským problémem s aplikací byla rychlost a pomalejší nativní gesta. Vývoj nativní aplikace pomocí React Native bude zcela jistě finančně náročnější, ale ne na darmo se říká, že za kvalitu se platí. Díky investovaní do nativního vývoje bude výsledkem stabilní mobilní aplikace s jednoduchou udržovatelností a škálovatelností.

Porovnávaná oblast	WebView	React Native
Časová náročnost	4 měsíce	2 měsíce
Finanční výdaje	bez výdajů (vývojář není alokován jako vývojář nativních aplikací	140 000 kč
Škálovatelnost aplikace	špatná	dobrá
Vývojové prostředí	V obou dvou případech může být stejné	
Testování aplikace v průběhu vývoje	nepohodlné a pomalé	jednodušší a intuitivnější

Obrázek 5.1: Shrnutí porovnání implementace mobilní aplikace FLOWIO

Závěr

Cílem této diplomové práce bylo zanalyzovat koncept WebView, zanalyzovat programovací jazyky, které budou správně s WebView komunikovat a tyto získané poznatky aplikovat na vývoj mobilní podnikové aplikace FLOWIO. Snažil jsem se v práci ukázat, že je možné využít konceptu WebView i pro vykreslení složitější webové aplikace než jsou pouhé informační webové stránky, ke kterému je koncept vlastně určen. V kombinaci s programovacími jazyky React a React Native, se mi podařilo dosáhnout požadovaného výsledku a vytvořit funkční mobilní aplikaci FLOWIO. Mobilní aplikace vychází z již existující webové aplikace a většina obsahu mobilní aplikace je zobrazena právě pomocí WebView. V průběhu vývoje jsem zachovával integritu řešení webové části a jenom za pomoci malých úprav a přidání určitých funkcí se mi podařilo vytvořit stabilní komunikaci mezi WebView a webovou aplikací. Z následného uživatelského testování jsem dostal zpětnou vazbu, že aplikace opravdu na první pohled vypadá a chová se jako nativní mobilní aplikace. Tato verze mobilní aplikace FLOWIO poslouží přesně k tomu účelu, ke které byl její vývoj směrován a to k představení aplikace potencionálním investorům. Po schválení nápadu bude nutné aplikaci přeprogramovat do ryze nativní aplikace, k využití celého jejího potenciálu a k dosáhnutí maximální uživatelské spokojenosti.

Koncept WebView v sobě skrýval daleko větší potenciál, než je pouhé vykreslení statických webových stránek. Brzdou konceptu je však to, že není zamýšlen ve větším měřítku. Vývojáři konceptu odvedli skvělou práci, ale je toho ještě mnoho, co musí přidat a vylepšit, aby se mobilní aplikace používající tento koncept, dokázaly vyrovnat klasickým nativním aplikacím. Jedná se především o kompatibilitu mezi operačními systémy Android a iOS. Některé metody a vlastnosti WebView jsou dostupné pouze pro operační systémy Android a některé zase pouze pro iOS. Povýšení konceptu WebView na vyšší úroveň by stálo za zamyšlenou a mohlo by se stát dobrým námětem na téma jiné odborné práce.

Literatura

[1] Korolev, S.: Chose a dev approach for your mobile app. [online], 2019. Dostupné z: https://railsware.com/blog/native-vs-hybrid-vs-cross-platform/

[2] Definitions, I.: Application Software. [online]. Dostupné z: https://www.defit.org/application-software/

[3] Rouse, M.: Web application (Web app). [online], 2011. Dostupné z: https://searchsoftwarequality.techtarget.com/definition/Web-application-Web-app

[4] Guru99: Difference between Website and Web Application. [online], 2015. Dostupné z: https://www.guru99.com/difference-web-application-website.html

[5] Clement, J.: Worldwide mobile app revenues in 2014 to 2023. [online], 2014. Dostupné z: https://www.statista.com/statistics/269025/worldwide-mobile-app-revenue-forecast/

[6] Annie, A.: The State of Mobile 2020. [online], 2020. Dostupné z: https://www.appannie.com/en/go/state-of-mobile-2020/

[7] Mroczkowska, A.: What is a Mobile App? [online], 2019. Dostupné z: https://www.thedroidsonroids.com/blog/what-is-a-mobile-app-app-development-basics-for-businesses

[8] Peek, S.: What is mobile app development? [online], 2020. Dostupné z: https://www.businessnewsdaily.com/5155-mobile-app-development.html

[9] Karnes, K.: How Mobile Hybrid Apps Differ From Native. [online], 2020. Dostupné z: https://clevertap.com/blog/hybrid-apps/

[10] Briscoe, P.: The WebView strategy for creating mobile apps. [online], 2015. Dostupné z: https://www.human-element.com/webview-strategy-creating-mobile-apps-part-13

[11] Morris, S.: Tech 101: What is JavaScript. [online]. Dostupné z: https://skillcrush.com/blog/javascript/

[12] Sufyian, T.: What is React. [online], 2020. Dostupné z: https://www.simplilearn.com/what-is-react-article

[13] Arsenault, C.: What is npm? [online], 2017. Dostupné z: https://www.keycdn.com/blog/npm-vs-yarn

[14] Arsenault, C.: What is Yarn? [online], 2017. Dostupné z: https://www.keycdn.com/blog/npm-vs-yarn

[15] Budziński, M.: What is React Native. [online], 2019. Dostupné z: https://www.netguru.com/what-is-react-native

[16] Korolev, S.: Native app development. [online], 2019. Dostupné z: https://railsware.com/blog/native-vs-hybrid-vs-cross-platform/

[17] Korolev, S.: Hybrid app development. [online], 2019. Dostupné z: https://railsware.com/blog/native-vs-hybrid-vs-cross-platform/

[18] Foundation, T. A. S.: Architectural overview of Apache Cordova. [online], 2015. Dostupné z: https://cordova.apache.org/docs/en/latest/guide/overview/index.html

[19] AR, M.: What is Ionic framework app development? [online], 2019. Dostupné z: https://www.quora.com/What-is-Ionic-framework-app-development

[20] Korolev, S.: Cross-platform app development. [online], 2019. Dostupné z: https://railsware.com/blog/native-vs-hybrid-vs-cross-platform/

[21] Schevcheko, O.: React Native. [online], 2018. Dostupné z: https://hackernoon.com/web-apps-turn-website-into-mobile-app-your-four-best-options-78fcb2277be8

[22] Altexsoft: Xamarin. [online], 2019. Dostupné z: https://www.altexsoft.com/blog/mobile/pros-and-cons-of-xamarin-vs-native/

[23] Thomas, G.: What is Flutter and Why You Should Learn it in 2020. [online], 2020. Dostupné z: https://www.freecodecamp.org/news/what-is-flutter-and-why-you-should-learn-it-in-2020/

[24] Schevcheko, O.: Progressive Web Apps. [online], 2018. Dostupné z: https://hackernoon.com/web-apps-turn-website-into-mobile-app-your-four-best-options-78fcb2277be8

[25] LePage, P.: PWA Install criteria. [online], 2020. Dostupné z: https://web.dev/install-criteria/

[26] Korolev, S.: Progressive web app – as a bonus to mention:). [online], 2019. Dostupné z: https://railsware.com/blog/native-vs-hybrid-vs-cross-platform/

[27] (https://stackoverflow.com/users/2960788/christopher), C.: Can I use library that used android support with Androidx projects. [online], 2018. Dostupné z: https://stackoverflow.com/questions/52033810/can-i-use-library-that-used-android-support-with-androidx-projects

[28] Malbranche, T.: source, WebView prop. https://github.com/react-native-webview/react-native-webview, 2018.

[29] The window.ReactNativeWebView.postMessage method and onMessage prop. https://github.com/react-native-webview/react-native-webview, 2018.

[30] The injectJavaScript method. https://github.com/react-native-webview/react-native-webview, 2018.

[31] Au-Yeung, J.: Introducing the JavaScript Window Object — XML and Console. [online], 2019. Dostupné z: https://www.keycdn.com/blog/npm-vs-yarn

[32] Document element. https://developer.mozilla.org/en-US/docs/Web/API/Document, 2020.

[33] EventTarget.addEventListener(). https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener, 2020.

[34] EventTarget.removeEventListener(). https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/removeEventListener, 2020.

[35] EventTarget.dispatchEvent(). https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/dispatchEvent, 2020.

[36] Wieruch, R.: Introduction to React Hooks - REACT USE STATE HOOK. [online], 2020. Dostupné z: https://www.robinwieruch.de/react-hooks

[37] Pavlutin, D.: A Simple Explanation of React.useEffect(). [online], 2020. Dostupné z: https://dmitripavlutin.com/react-useeffect-explanation/

[38] Garcia, C. G.: React useRef Hook. [online], 2019. Dostupné z: https://medium.com/trabe/react-useref-hook-b6c9d39e2022

[39] Sessions, M.: onLayout in React Native. [online], 2017. Dostupné z: https://www.matthewsessions.com/blog/react-native-on-layout/

[40] RefreshControl. https://reactnative.dev/docs/refreshcontrol/, 2020.

[41] Redux DevTools. https://github.com/zalmoxisus/redux-devtools-extension, 2016.

[42] React Native Debugger. https://github.com/jhen0409/react-native-debugger, 2016.

[43] Reactotron. https://github.com/infinitered/reactotron, 2019.

[44] Gupta, D.: How Long Does it Take to Build a Mobile App? [online], 2020. Dostupné z: https://appinventiv.com/blog/how-long-does-it-take-to-build-a-mobile-app/

[45] Patel, C.: Know The Cost To Hire A React Native Developer. [online], 2018. Dostupné z: https://www.bacancytechnology.com/blog/cost-to-hire-react-native-developer

[46] React Native pracovní místa v České Republice. https://nofluffjobs.com/cz/jobs/prague/react?gclid=CjwKCAiAwrf-BRA9EiwAUWwKXmtZPawWs_KajozCIr_qaN6MWoDZwOUbSPm-aqp_tCURAPH2AclLwxoCEvMQAvD_BwE, 2020.

Příloha **A**

Seznam použitých zkratek

API Application Programming Interface

CSS Cascading Style Sheets

DOM Document Object Model

GUI Graphical User Interface

HOC High Order Component

HTML Hypertext Markup Language

IDE Integrated Development Environment

MVC Model View Controller

NPM Node Package Manager

SDK Software Development Kit

URL Uniform Resource Locator

UI User Interface

UX User Experience

VDOM Virtual Document Object Model

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
  - `impl` ..... zdrojové kódy použité v práci
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF