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

Multiplatformní vývoj mobilní aplikace

Diplomová práce

Autor: Jakub, Robotka
Studijní obor: Informační management 2

Vedoucí práce: Ing. Jiří Štěpánek

Hradec Králové Duben 2019

Prohlášení:

Prohlašuji, že jsem diplomovou práci *Multiplatformní vývoj mobilní aplikace* zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 29.4.2019 Jakub Robotka

Poděkování:

Děkuji vedoucímu diplomové práce Ing. Jiřímu Štěpánkovi za metodické vedení práce a cenné rady, které mi při psaní této diplomové práci poskytl.

Anotace

Cílem diplomové práce je představit možnost multiplatformního vývoje mobilní aplikace. V textu jsou blíže specifikovány aktuálně používané nástroje pro vývoj mobilních aplikací, a zároveň je provedeno jejich porovnání na základě stanovených kritérií. Praktická část obsahuje implementaci vybraného nástroje React Native pro tvorbu aplikace Banko. Banko slouží k vyhledání nejbližšího bankomatu podle aktuální polohy telefonu. Proces vývoje je popsán v praktické části, obsahující kritické situace spolu s jejich možným řešením a následným vyhodnocením procesu realizace mobilní aplikace v technologii React Native. Diplomová práce by mohla pomoci vývojářům jako rozšíření znalostí o současných možnostech multiplatformního vývoje.

Annotation

Title: Cross-platform development of mobile application

The goal of thesis is to introduce the possibility of multiplatform mobile application development. The currently used tools for development of mobile applications are presented in more detail and their comparison on the basis of established criteria. The practical part contains the implementation fot he selected tool React Native for creating the application named Banko. The Banko is used to find the nearest ATM based on your phone´s current location. The development proces is descripbed int he practical part, containing critical situations together with possible solutions and subsequent evalution of the implementation process of mobile application in React Native technology. The thesis could help developers to extend their knowledge of current possibilities of cross-platform development.

Obsah

1 Úvod

Mobilní zařízení v současné době zaznamenávají neustálý nárůst podílu na trhu po celém světě, kdy v některých případech už překonávají desktopové zařízení. S ohledem na danou informaci je důležité optimalizovat služby pro koncové uživatele. Optimalizace musí probíhat jak pro desktopové zařízení, tak i pro mobilní zařízení. (1)

S vývojem chytrých zařízení narůstá počet přidávaných funkcionalit, které mohou usnadňovat běžné denní aktivity. Až už jde o platbu mobilem přes NFC[1], nebo sledování sportovních aktivit. Koncoví uživatelé již očekávají, že aplikace používané v jejich chytrých telefonech umí využívat danou funkcionalitu. Samozřejmě to s sebou nese poptávku od klientů k vývoji komplexních aplikací v krátkém čase. Díky tomu vzrostla potřeba urychlit vývoj mobilních aplikací.

Z daného důvodu je zvoleno téma řešící otázky multiplatformního vývoje mobilních aplikací. Multiplatformní aplikace mají za úkol odstranit duplicitu v logické části právě sdílením mezi vybrané platformy. Zkracují tak dobu vývoje aplikace a finanční náročnost pro klienta. Se sdílenou logikou se snižuje náročnost na udržitelnost aplikace a vývoj nové funckionality je mnohem efektivnější.

Tato diplomová práce se zabývá rozborem vybraných technologií, které umožnují multiplatformní vývoj mobilních aplikace. Každá vybraná technologie využívá svůj přístup pro tvorbu aplikace. Vzniká zde možnost efektivního porovnání jednotlivých technologií a výběru vhodné platformy pro budoucí vývoj aplikace.
V další části se práce zabývá prozkoumáním frameworku React Native a jeho praktického použití pro předem definovanou aplikaci. Framework je představen jak z teoretického pohledu, tak i po praktické stránce. V teoretické části jsou vysvětleny základní prvky frameworku, které jsou následně používány v praktické části aplikace. Praktická část popisuje, kromě samotné funkcionality, problémy při implementaci logických nebo vzhledových částí, se kterými s může vývojář

[1] Near Field Communication – modulární technologie radiové bezdrátové komunikace mezi elektronickými zařízeními na velmi krátkou vzdálenost (55)

v průběhu své praxe setkat. Při používání nové technologie vývojáři často naráží na problémy nedostatečné znalosti technologie a tomu odpovídající otázky, jak problém řešit. Praktická část demonstruje některé problémy spolu s řešením, které mohou pomoci vývojáři při hledání vhodného řešení.

2 Cíl práce

Pro získání výsledků je tato práce rozdělena na hlavní a dílčí cíle. Mezi hlavní cíle práce patří:

- Analýza existujících frameworků pro vývoj multiplatformních aplikací. V současné době už existuje více nástrojů a možností, jak vyvíjet mobilní aplikace pro více platforem. Pro rozbor a porovnání jsou zvolený jedny z aktuálních a principiálně rozdílných frameworků. Následně je provedeno porovnání a vyhodnocení každého zvolené frameworku.
- Vytvoření funkční aplikace ve zvoleném frameworku React Native. Pro funkční aplikaci byl zvolen jeden z rozebíraných frameworků, se kterým je vytvořena aplikace s požadavky odpovídajícími nativní aplikaci.
- Pro větší přínos jsou analyzovány jednotlivé problémy při vývoji aplikace a následně nalezeno jejich optimálního řešení.

K dosažení hlavních cílů a lepšího pochopení výsledků, je potřeba se orientovat v základní problematice. Proto je potřeba nejprve vytyčit dílčí cíle, které čtenáře zasvětí do tématu.

- Vymezit pojem multiplatformní vývoj. Vysvětlit rozdíly vývoje mobilní aplikace způsobem multiplatformního oproti nativnímu vývoji. Následně je rozebráno samotné dělení multiplatformního vývoje a na jakém principu fungují.
- Přiblížení a vysvětlení principu React Native. Pro zvolený framework je potřeba uvést podrobnější chování a funkcionalitu.

2.1 Přínos

Výsledky práce přinesou informace o výhodách a nevýhodách jednotlivých nástrojů pro vývoj multiplatformní aplikace. Dává stručný přehled a porovnání při rozhodování výběru technologie. Čtenář se bude lépe orientovat v problematice mobilních aplikací a snadněji tak volit nástroj odpovídající požadavkům na jeho mobilní aplikaci.

Praktická práce se zaměřuje na vývoj reálné aplikace ve vybraném frameworku React Native. Využívají se zde teoretické znalosti k dosažení požadované aplikace. Vytvářená aplikace řeší širokou škálu požadavků, od práce s veřejnými API a zapojení nativní funkcionality mobilního zařízení, až po zaměření specificky na danou platformu a jiné pomocné ukázky. Získané znalosti lze pak následně využívat při řešení podobných situací na jiných aplikací.

3 Metodika zpracování

Pro dosažení cíle je provedena analýza jednotlivých frameworků. Je potřeba prostudování principů mobilních aplikací, architektury kompilace a jejich omezení. Dále jsou stanovená kritéria, pod kterými se budou frameworky hodnotit a z výsledků je vytvořena tabulka jejich porovnání. Vývojář dostane shrnuté informace o vybraných možnostech vývoje mobilních aplikací a může z nich lépe vybrat technologii, která efektivně vyřeší jeho požadavky na aplikaci.

V další části se představí jeden z vybraných frameworků podrobněji. Zjistí se z dostupných zdrojů způsob chování a vývoj aplikací. Veškeré znalosti se následně aplikují při vývoji aplikace. V aplikaci jsou použity komplexní prvky chování, které následně přiblíží skutečný vývoj aplikací. Ke každému vývoji patří chvíle řešení problémových situací. Hledání a aplikování řešení pro problémovou situaci je důležitá součást vývoje jakéhokoliv produktu. Vývojář se v danou chvíli sám vzdělává a zjišťuje nové situace a jak se daným problémům vyvarovat. Proto jsou tyto informace předávány na konci praktické části spolu s vývojem aplikace.

4 Úvod do mobilního vývoje

Kapitola přináší úvodní poznatky z oblasti vývoje mobilních aplikací. Primárně je představen úvod do operačních systémů, odkud se pokračuje k vysvětlení rozdílů jednotlivých přístupů vývoje mobilních aplikací.

4.1 Mobilní platformy

Definice mobilní platformy je lépe definována z hlediska tradičních operačních systémů. Operační systém je prostředí běžící mezi holým hardwarem a aplikačními programy. Operační systémy poskytují hardwarovou abstrakci, ovladače a síťové modely, bezpečnostní architekturu, zařízení pro správu procesů a paměti pro optimální využití hardwarového zdroje. Mobilní platforma je ekvivalentem tradičního operačního sytému pro mobilní zařízení – včetně tabletu nebo chytrého telefonu. (1)

V této kapitole jsou shrnuty dvě základní mobilní platformy. Jedná se o platformu Android a iOS, u kterých došlo k velkému nárůstu oproti jiným platformám. Jeden z hlavních faktorů je růst vývoje mobilních aplikací na daných platformách. Android a iOS představovaly skoro 88% podílu na mobilním trhu v roce 2012. Zatímco Blackberry, Symbian, Windows Phone a jiné společnosti pokrývaly zbylých 12% podílu na trhu. (2)

Nejnovější statistiky z roku 2018 aktuálně ukazují, že dominujícím operačním systémem je Android s podílem okolo 78 % na trhu. Na druhém místě se řadí iOS s podílem 16 %. Zbylých 6 % trhu se dělí mezi zbylé, jako jsou Windows Phone, Blackberry a jiné Linuxové zařízení. (3)

4.2 Typy mobilních platforem

Tato sekce se zabírá představením dvou platforem, pro které je následně vyvíjena aplikace.

4.2.1 Android

První veřejná beta verze byla vydána dne 12. listopadu 2007, avšak první mobilní telefon s operačním systémem vyšel až 23. září roku 2008. Android se tak

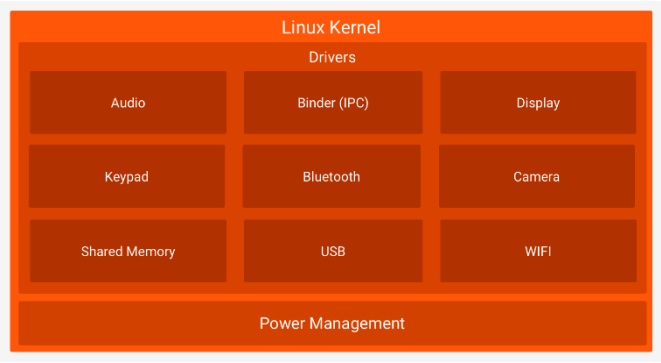
stal rychle vyvíjeným operačním systémem. Třetí verze byla představena počátkem roku 2011 a čtvrtá verze později téhož roku. (4) (5)

Operační systém Android je vyvinut pod záštitou společnosti Google a jádrem platformy je Linux. Vzhledem k tomu, že systém Android je otevřenou platformou (open source), umožnuje samotné provozování a vyvíjení různých variant na různých zařízení od různých výrobců. S tímto přístupem jsou méně závislí na jediném výrobci, ale samotní výrobci si navzájem konkurují při následném prodeji zařízení. (6)

Společnost Google učinila některé kroky, aby minimalizovala roztříštěnost svého operačního systému Android pomocí klauzule o neslučitelnosti u všech oficiálních prodejců využívající platformu. To znamená, že všechny změny v jejich operačním systému musí být před vydáním společností Google schváleny. (6)

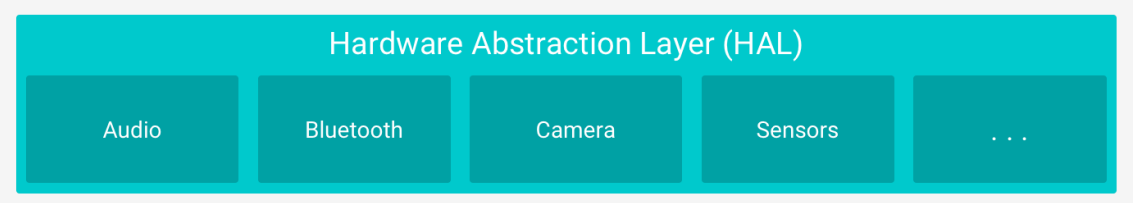
4.2.1.1 Architektura

Android je volně dostupný operační systém založený na Linuxu. Je tvořen pro širokou škálu zařízení. Následující obrázek ukazuje hlavní součásti platformy Android, ze kterých je tvořena.



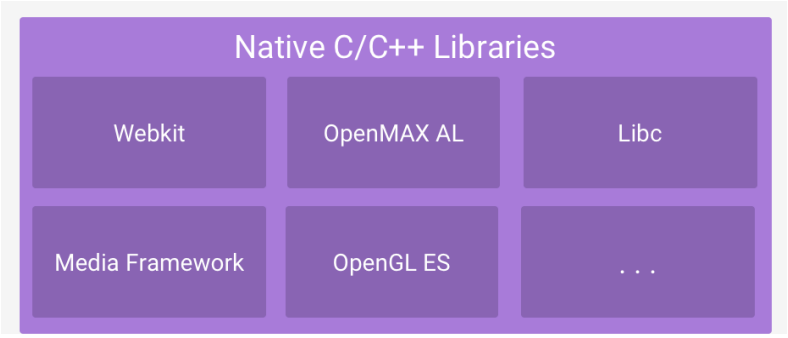
Obrázek 1: Architektura platformy Android – Linux Kernel (7)

Základem platformy Android je jádro Linuxu (**Chyba! Nenalezen zdroj o dkazů.**). Zahrnuje ovladače hardwaru pro různé komponenty mobilního telefonu a poskytuje služby jádrovému systému, jako je správa paměti, správa procesů a šítový zásobník. (7)



Obrázek 2: Architektura platformy Android –HAL (7)

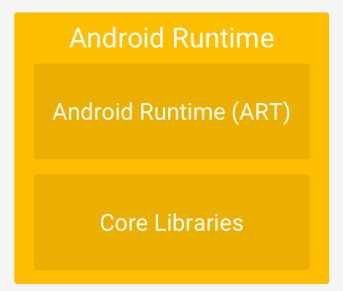
Hardwarová abstrakční vrstva (HAL) poskytuje standardní rozhraní, která vystavují vlastnosti hardwaru zařízení do vyšší úrovně rozhraní Java API[2]. Modul HAL se skládá z několika modulů knihoven, z nichž každá implementuje rozhraní pro konkrétní typ hardwarových komponentů, jako je zvuk nebo senzory.



Obrázek 3: Architektura platformy Android – C/C++ Knihovny (7)

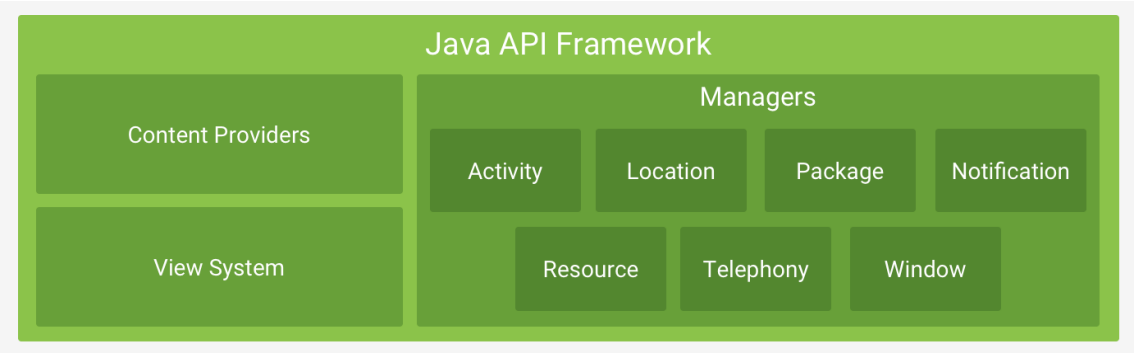
Fialová vrstva se skládá z knihoven, které mohou být použity u každé aplikace pro Android (podléhají bezpečnostním omezením). Tyto služby jsou psané v jazyce C nebo C++ a jsou vystavovány skrze API pro využívání aplikací v Androidu. Dostupné knihovny jsou například Webkit[3] a SQLite[4], které poskytují vložený webový pohled a odlehčenou databázi. (7)

[2] Application program interface – sada rozhraní pro tvorbu aplikací.
[3] Webkit – rendrovací jádro prohlížeče využívané v Apple aplikacích a jiných operačních systémech (např. Linux, Windows) (56)
[4] SQLite – relační databáze psaná určená pro aplikace. (50)



Obrázek 4: Architektura platformy Android: Runtime (7)

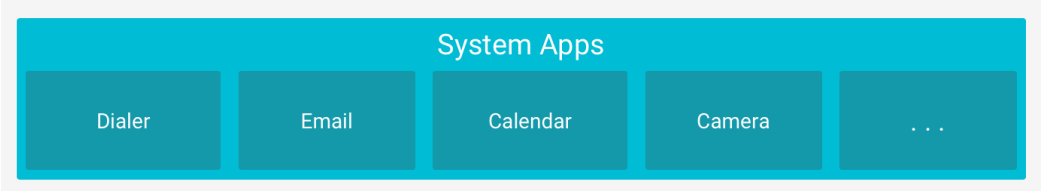
U zařízení se systémem Android verze 5.0 a vyšší se každá aplikace spouští ve svém vlastním procesu a s vlastní instancí aplikace Android Runtime (ART). ART je napsán pro spouštění souborů DEX, což je bytecode[5] formát navržený speciálně pro Android, který je optimalizován pro minimální paměť.



Obrázek 5: Architektura platforma Android: Runtime (7)

Celá sada funkcí operačního systému Android je k dispozici prostřednictvím rozhraní API psané v jazyce Java. Tato rozhraní API vytvářejí stavební bloky, které jsou potřebné k vytváření aplikací pro Android, a to zjednodušením opětovného použití jádrových, modulárních součástí systému a služeb. (7)

[5] Byte code – označení pro kompilovaný formát v jazyce Java. (57)



Obrázek 2: Architektura platformy Android: Systémové aplikace (7)

Android obsahuje sadu hlavních aplikací pro e-maily, SMS zprávy, kalendáře, internetového prohlížeče, kontakty a další aplikace. Aplikace zahrnuté do platformy nemají žádný zvláštní stav mezi aplikacemi, které se uživatel rozhodne nainstalovat. Proto aplikace třetích stran se mohou stát výchozími aplikacemi jako je nastavení webového prohlížeče. Tyto systémové aplikace tvoří nejvyšší vrstvu v architektuře Android platformy. Fungují jako aplikace pro koncového uživatele a poskytují klíčové funkce. (7)

4.2.1.2 Vývojový jazyk

Pro vývoj aplikace pro platformu Android si může vývojář vybrat z několika jazyků. Mezi nejvíce rozšířené se řadí:

- Java – Oficiální jazyk pro vývoj aplikace pro Android a je podporován v Android Studiu. Avšak tento jazyk má svojí učící křivku.
- Kotlin – Kotlin je novější vedlejší oficiální programovací jazyk. Velmi se podobá jazyku Java, avšak je jednodušší pro začátečníky.
- C/C++ – Android studio také podporuje C++ s použitím Java NDK[6]. Umožňuje nativní psaní aplikací, které mohou být užitečné pro psaní například her. Avšak tento jazyk může být komplikovaný.
- C# – Přátelská alternativa k C nebo C++. Je podporována užitečnými nástroji jako je Unity nebo Xamarin, které jsou skvělé pro vývoj her nebo aplikací pro více platforem.

[6] Java NDK – sada nástrojů umožňující implementovat aplikaci v nativním kódu, i když je například psaná v jazyce C. (58)

Existuje více možných jazyků, které lze využít pro vývoj mobilních aplikací určené pro Android. (8)

4.2.1.3 Vývojové prostředí

Android má výhodu oproti iOS platformě v tom, že neomezuje vývojáře a lze aplikace vyvíjet na kterémkoliv operačním systému (Windows, Mac, Linux). Hlavním vývojovým nástrojem je Android SDK, který obsahuje komplexní sadu vývojových nástrojů. Patří sem ladící program (debugger), knihovny, emulátory, dokumentace a jiné pomocné nástroje. (8)

Oproti tomu existuje mnoho nástrojů třetích stran podporujících vývoj Android aplikací. Mezi známé se zde může zařadit například *Xamarin*, *Corona SDK* nebo *Go-lang*, který podporuje vývoj Android aplikace od verze 1.4.

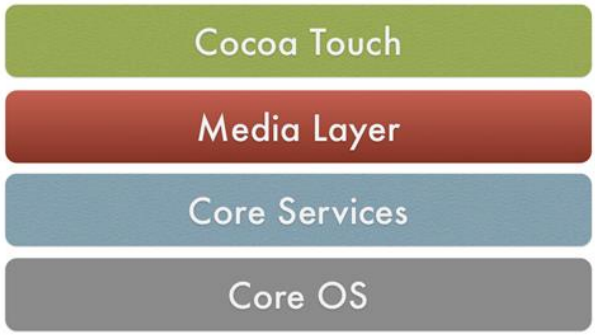
4.2.2 iOS

Dalším velkým hráčem je společnost Apple se svojí platformou iOS. iOS dříve znám spíše jako iPhone OS, byl představen v červnu roku 2007. Jeho základ je založen na úspěšném desktopovém operačním systému Mac Os X. Apple se svým produktem měl také velký úspěch vzhledem k tomu, že jejich platforma není volně dostupná jako je Android. Společnost Apple si vyrábí vlastní zařízení a svůj vlastní operační systém. Navíc Apple zvolil pouze výrobu menšího množství variant svého zařízení. To je hlavní důvod, proč Apple nemá problémy s fragmentací, kterou musí řešit platforma Android. Tato strategie poskytuje vyšší úroveň jednotnosti zařízení a zlepšuje standardizaci. To dává společnosti Apple větší kontrolu nad zařízeními a umožňuje tak pomáhat vývojářům zajistit, aby jejich aplikace fungovala podle očekávání. Nevýhodou uzavřenosti a užším výběrem společnosti Apple je následně menší podíl na trhu. Apple byl v roce 2010 jedinou platformou, která tento přístup využívala. (9)

Vzhledem k tomu, že Apple vyrábí vlastní telefony a rozvíjí své operační systémy samostatně, je pro ně jednodušší optimalizovat jejich operační systém s dostupnými zařízeními na jejich platformě. Existují však změny a obměny každé verze, ale většina z nich souvisí s funkcemi telefonu, jako je zvýšení výkonu, rozlišení fotoaparátu apod. Avšak vzhled telefonu se stále držel stejný. (9)

4.2.2.1 Architektura

Systém iOS se skládá ze čtyř základních vrstev, které zajištují základní funkčnost. Jednotlivé vrstvy jsou ukázány na následujícím obrázku. (10)



Obrázek 3: iOS architektura (10)

- Cocoa Touch
- Media
- Core Services
- Core OS

Vrstva Cocoa Touch je nejvyšší vrstva iOS architektury. Obsahuje některé klíčové frameworky, na kterých se nativní aplikace spoléhají. Tato vrstva definuje základní aplikační infrastrukturu a poskytuje řadu klíčových technologií, jako je multitasking a dotykový vstup. iOS aplikace se silně spoléhají na Framework UIKit. UIKit představuje grafickou infrastrukturu iOS aplikací. Také se stará o ostatní základní aspekty, jako jsou notifikace nebo dostupnost. Cocoa Touch vrstva poskytuje vývojáři klíčové vlastnosti, které lze při tvorbě aplikací využít. Při vývoji je doporučováno začínat s danou vrstvou a ostatní vrstvy používat pouze pokud je to nutné. (10)

Media vrstva se stará o výstupní technologie jako je grafika, audio nebo video. Technologie napomáhá aplikacím k vylepšení vzhledu a zvuku u jednotlivých koncových aplikací. Jednotlivé technologie spolupracují následně s hardwarem zařízení, aby poskytly co nejlepší výsledek, ať už v podobě zvuku, nebo videa. (10)

Vrstva Core Service je zodpovědná za správu základních systémových služeb, které používají nativní aplikace iOS. Vrstva Cocou Touch závisí právě na této vrstvě

pro některé z jejích funkcí. Vrstva Core Service také poskytuje řadu nepostradatelných funkcí, jako je využívání síťových protokolů, SQLite knihovnu nebo služba umožňující zobrazit soubory uživatele. (10)

Jako hlavní vrstva v iOS architektuře je považována vrstva Core OS, která poskytuje nízkoúrovňovou funckionalitu ostatních technologií, které jsou na ní postaveny. Vrstva Core OS poskytuje hrst frameworků, které vaše aplikace může používat přímo, například se může jednat o Bezpečnostní Framework. Vrstva zapouzdřuje prostředí jádra a rozhraní UNIX s nízkou úrovní. Tím zajištuje správu virtuální paměti, vlákna, rozhraní mezi dostupným hardwarem a systémovými frameworky. (10)

4.2.2.2 Vývojový jazyk

Platforma iOS je založená na Mac OSC a využívá Cocoa Programming prostředí. Cocoa automatizuje mnoho vzhledových prvkových elementů, tím aplikace vyvíjená v daném prostředí splňuje požadovaný Apple vzhled uživatelského prostředí. Základními jazyky pro vývoj aplikací jsou Objective-C nebo poměrně mladý jazyk Swift. Xcode nabízí vývojáři všechny editovací a testovací schopnosti, stejně jako dokumentaci SDK v jednoduchém rozhraní. (11)

4.2.2.3 Vývojové prostředí

Apple v souladu s vlastnickou povahou zajištuje omezený vývoj iOS aplikací pouze na Mac počítači. Nepostradatelným vývojovým prostředím nutné pro vývoj je XCode. I když se aplikace vyvíjí například v jazyce JavaScript, je stále nepostradatelný. XCode obsahuje funkce jako emulátor pro testování na Apple zařízení i proces pro vytvoření finální aplikace pro Apple Store. Testovat lze pouze ve virtuálním zařízení, v případě že vývojář chce testovat na reálném zařízení, musí mít zakoupenou vývojářskou licenci. S touto licencí může následně publikovat svoje aplikace do oficiálního Apple Storu pro veřejnost. (11)

4.3 Přístupy vývoje

V dnešní době existuje několik nástrojů pro vytváření aplikací pro více jak jeden operační systém, které využívají různé typy architektonických přístupů. Mezi

běžně používané přístupy lze zařadit webový, hybridní, cross-compile a interpretovaný přístup. Tyto přístupy jsou popsány v této části spolu s jejich výhodami a nevýhodami.

4.3.1 Nativní přístup

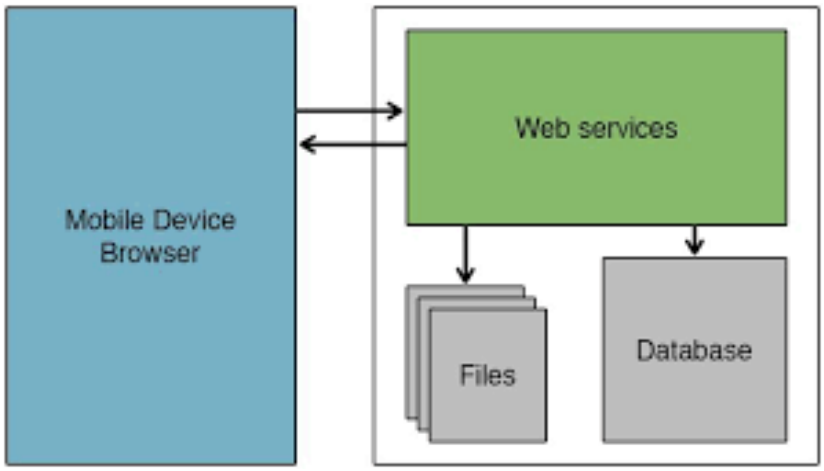
Nejčastější technikou, která se dnes používá k vytváření mobilních aplikací, je psaní aplikace pro každou platformu zvlášť. Tento způsob není multiplatformní, ale je to stále nejvíce výkonný způsob tvorby mobilní aplikace. (12)

Nativní přístup má výhody ve využívání veškerých funkcí pro platformu a samotný výkon aplikací se nedá srovnávat s ostatními přístupy. Proto tyto aplikace budou nejvýkonnější. Při psaní aplikací nativním přístupem má vývojář k dispozici specifické uživatelské rozhraní pro danou platformu. Výsledná aplikace tak vždy splňuje vzhled ke konkrétní platformě. Samozřejmě nativní aplikace mohou být umístěny do obchodů s aplikacemi, a tedy instalace může být lehce zpeněžena. (12)

Nevýhodou je však samotný vývoj aplikace pro každou platformu samostatně. Je nutné vyvíjet více aplikací a tím vzniká duplikační logika, akorát pokaždé napsaná v jiném jazyce. Dále pro vytváření nativních aplikací je potřeba vývojáře, případně více vývojářů, kteří dokážou vytvářet aplikace v různých programovacích jazycích. Jeden z důvodů vzniku multiplatformního vývoje mobilních aplikací. (13)

4.3.2 Webový přístup

Jeden z dalších přístupů k vývoji je tvorba aplikace jako webová aplikace, která je realizována v prohlížeči mobilního zařízení, jak je prezentováno na Obrázek 3: iOS architektura



Obrázek 4: Architektura webové mobilní aplikace (14)

Tento přístup využívá standardních webových technologií jako HTML, CSS a JavaScript k vytvoření aplikace, která se podobá nativní aplikaci. HTML a CSS využívají pokročilé funkce, jako jsou vestavěné databáze, localStorage[7] a animace (14). Webový přístup využívá například nástroje SensaTouch nebo jQuery Mobile. Švédská mobilní stránka IKEA je jedním z příkladů, který je vybudován pomocí jQuery mobile. (15)

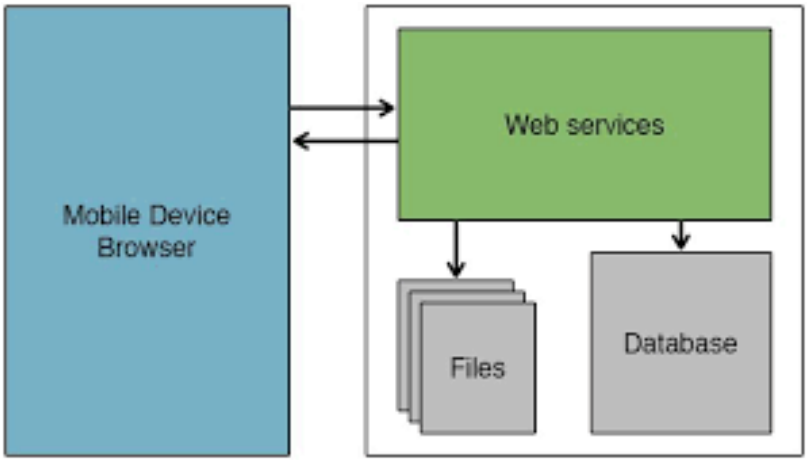
Webový přístup vyhovuje určitým druhům aplikací, neboť je levný a má potenciál vytvářet aplikace, které pracují na více platformách pouze s nepatrnými změnami. Další výhodou je okamžitá dostupnost aplikace, protože pro své spuštění vyžaduje pouze prohlížeč. Avšak tento přístup má také své nevýhody. Jedním z hlavních nedostatků je potíž emulovat přirozené uživatelské rozhraní, protože vývojář je omezen na standardní webové prvky. Použití funkcí zařízení je také omezeno ve srovnání s jinými přístupy. Nelze tedy využívat vestavěné funkce zařízení jako je GPS nebo kamera. Další výzvou pro vývojáře je malá kontrola nad tím, jak je aplikace zobrazována u různých prohlížečů. Chcete-li tuto situaci

[7] localStorage – jedná se o uložiště v prohlížeči klienta přístupné Javascriptem. (51)

překonat, musí se věnovat dostatek času na testování. Nakonec aplikace nemůže být vydána prostřednictvím obchodů s aplikacemi, což může mít negativní dopad na její distribuci (14).

4.3.3 Hybridní přístup

Existuje také hybridní přístup k vývoji více platformních aplikací (**Chyba! N enalezen zdroj odkazů.**), který spočívá mezi webovou a nativní metodikou. Tento přístup je založen na webových technologiích, ale je prováděn v nativním kontejneru, který umožňuje přístup k funkcím zařízení. Kontejner spouští WebView, lehký webový prohlížeč, kde je kód spouštěný. S pomocí frameworků mohou být prvky nativního uživatelského rozhraní vytvořeny pomocí HTML. Komunikace mezi webovou vrstvou a hybridní aplikací probíhá obvykle prostřednictvím rozhraní API jazyka JavaScript. (14) Populární hybridní nástroj je PhoneGap. PhoneGap byl použit k vytvoření mobilní aplikace TripCase. (16)

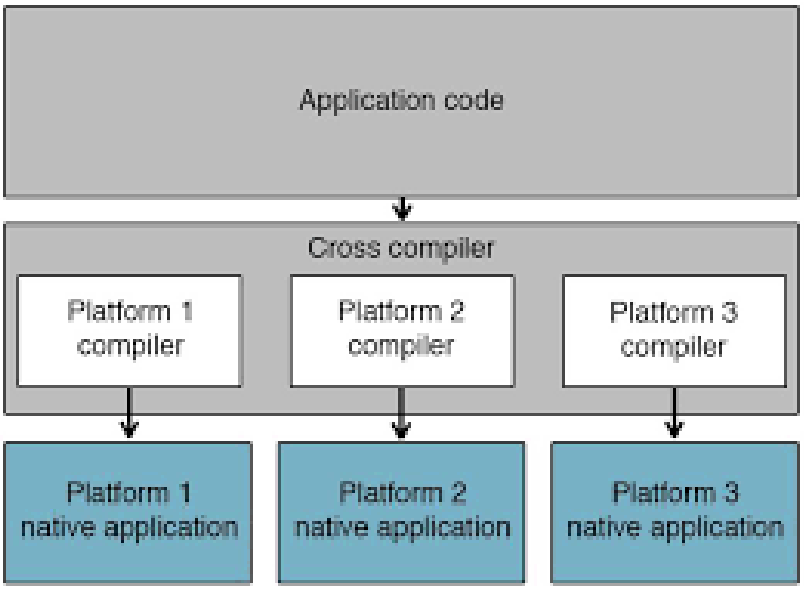


Obrázek 5: Architektura hybridní mobilní aplikace (14)

Výhodou hybridních aplikací je to, že jsou lokálně uloženy v zařízení a funguje tak efektivněji než standardní webové aplikace. Umožňuje tak využívat funkce zařízení, jako je kamera a GPS (13). Další výhodou je, že hybridní aplikace lze distribuovat prostřednictvím oficiálních obchodů s aplikacemi. Nevýhodou tohoto přístupu je obtížnost dosažení nativního uživatelského rozhraní a výkonnostně jsou na tom hůř než nativní aplikace (14).

4.3.4 Cross-compile přístup

Hartmann a kol (13) popisují cross-compiling jako techniku, která odděluje prostředí sestavení od cílového prostředí. Vývoj mobilních aplikací za využití cross-compiling znamená, že vývojáři využívají framework, který poskytuje API nezávislé na platformě. Rozhraní API by mělo používat hlavní programovací jazyk, jako je JavaScript, Java a jim podobné. Pomocí rozhraní API vývojář vytváří různé části aplikace: uživatelské rozhraní, strukturu dat a logiku samotné aplikace. Pomocí kompilace se poté transformuje kód na nativní aplikaci pro cílené platformy, viz (**Chyba! Nenalezen zdroj odkazů.**). Multiplatformní nástroje využívající cross-c ompile jsou například RhoMobile nebo Xamarin. Xamarin byl například použit k vybudování mobilní aplikace pro easyJet (17).

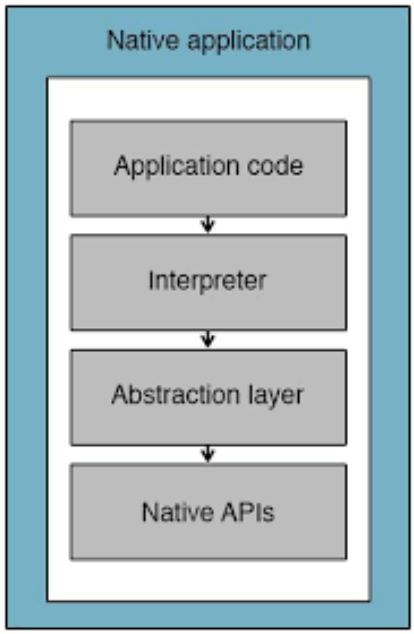


Obrázek 6: Architektura cross-compile mobilní aplikace (14)

Podle Hartmanna (13) má tento multiplatformní přístup několik výhod. První a nejdůležitější výhodou je výkon, jelikož aplikace se chová jako nativní aplikace s ohledem na uživatelské chování. Navíc tento přístup umožňuje přístup ke specifickým funkcím zařízení, jako je fotoaparát. Hlavní nevýhodou je samotná implementace a udržitelná konzistentnost s používanými mobilními platformami a operačními systémy. (14).

4.3.5 Interpretovaný přístup

Tento přístup využívá tlumočníka na cílovém zařízení, který interpretuje kód aplikace za běhu. Tlumočník je často virtuální stroj běžící na zařízení, který vykonává aplikaci jako fyzický stroj. Prostřednictvím abstrakční vrstvy lze přistupovat k nativním rozhraním pomocí API (**Chyba! Nenalezen zdroj odkazů.**) a vystavit vývojářům nativní funkce. Vzhledem k tomu, že každá platforma má vlastního tlumočníka specifického pro platformu, je možné využívat nativního uživatelského rozhraní. Jedním z nástrojů, které využívají tento přístup je React Native nebo NativeScript. (14)



Obrázek 7: Architektura interpretovaného přístupu (14)

Výhody tohoto přístupu, podobně jako cross-compilovaného, spočívají v tom, že výsledná aplikace s sebou nese pocit plnohodnotné nativní aplikace, schopnost zařízení lze využít jako abstrakční vrstvy a distribuovat prostřednictvím oficiálních obchodů. Podle Hartmanna (13) poskytuje toto řešení vynikající přenosnost ve srovnání s kompilací, jelikož tlumočník je snadnější na údržbu v případě přidání nové podpory a funkcionality. Pokud jde o výkonnost, interpretovaný přístup překonává webový i hybridní přístup. Protože je však kód aplikace interpretován za běhu, může dojít k přetížení výkonnosti v porovnání s cross-compile kompilací. Další výzvou je, že tento přístup omezuje vývojáře na funkce poskytované využívaným frameworkem nebo nástrojem. (14)

4.3.6 Porovnání přístupů

V následující tabulce í je provedeno srovnání jednotlivých přístupů podle poskytnutých informací. Posuzují se základní vlastností přístupů. První porovnávaná charakteristika je využívání nativního uživatelského rozhraní (v případě, že aplikace budou vypadat jako nativní pro danou platformu). V daném přístupu lze vidět, že dané kritérium nesplňuje přístup za využití webového prohlížeče.

Přístup	*Nativní UI*	*Nativní funkce*	*Obtížnost vývoje*	*Výkon*
Nativní	Ano	Ano	Vysoká	Vysoký
Webový	Ne	Ne	Nízká	Ne
Cross-compile	Ano	Ano	Vysoká	Vysoký
Interpretovaný	Ano	Ano	Střední	Střední

Tabulka 1: Charakteristika přístupů multiplatformních aplikací (13)

Druhým kritériem je přístup k funkcím daného mobilního zařízení. Jelikož se jedná o mobilní aplikace, je důležité, aby mohli vývojáři využívat a integrovat dostupné funkce, které mobilní telefon nabízí. Kritérium nativních funkcí splňují všechny, kromě Webového přístupu.

Dalším je rychlost a cena vývoje. Toto kritérium může být také důležité v rozhodování při volbě přístupu. Někdy je potřeba dodat dané aplikace v krátkém čase nebo je rozpočet na vývoj mobilní aplikace nějak omezený. V tomto případě dopadl nejhůř nativní vývoj. Avšak Webový přístup v této oblasti exceluje.

Výkon aplikace je dalším důležitým kritériem. Může se stát, že aplikace bude pracovat s velkým množstvím dat a provádět obtížné operace. Proto je potřeba při výběru přístupu vývoje mobilní aplikace brát v potaz toto kritérium. Zde vyhrává Nativní přístup, který je za každých okolností maximálně optimalizovaný pro danou platformu. Stejně tak i přístup využívající překladač je v tomto kritériu vyhovující a dokáže vytvářet výkonné aplikace. (14)

5 Multiplatformní vývoj

Tato kapitola se zaměřuje na populární frameworky pro tvorbu mobilních aplikací. Každý z vybraných frameworků spoléhá na jiný princip fungování a architektury, ale společně všechny umožňují vývojáři vytvářet mobilní aplikace. Hlavním výsledkem v této kapitole je srovnání jednotlivých frameworků podle daných kritérií. Získané informace se pak shrnou do tabulky výsledků pro utvoření přehledu nástrojů pro vývoj mobilních aplikací.

Mezi vybrané frameworky patří Ionic, Xamarin.Forms, React Native a Flutter. Každá z technologií má svoje výhody a nevýhody. Každý framework je představen, následně se vysvětluje samotný princip, pokračuje se architekturou, která stojí za frameworkem. Důležité je poskytované API frameworku, jaké jsou limity frameworku a ukázka reálných aplikací. Prvním rozebraným frameworkem je Ionic.

5.1 Ionic framework

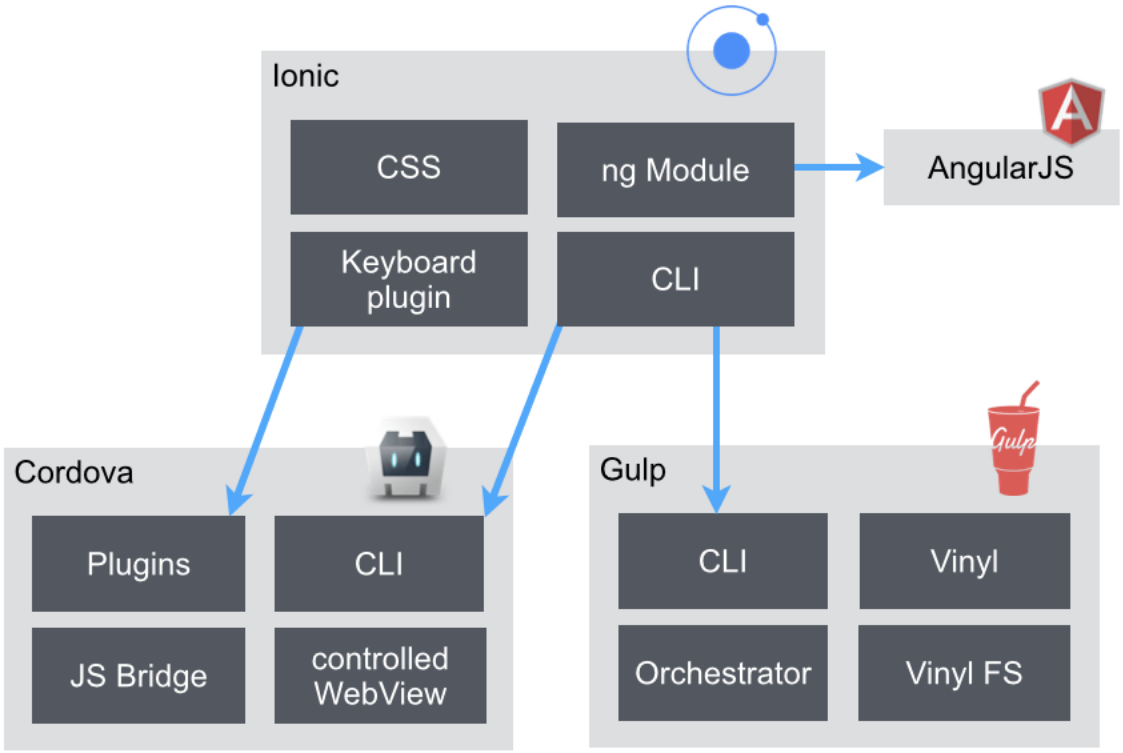
Ionic je volně dostupný framework pro tvorbu multiplatformních mobilních aplikací vytvořený společností Driffy Co. Pro tvorbu aplikací využívá webové technologie HTML5, CSS a JavaScript. S více než 37,000 hvězd na GitHubu, živou a aktivní komunitou se stává jedním z nejpoužívanějším hybridním frameworkem pro tvorbu mobilních aplikací. Ionic síla pochází z využití JavaScriptového frameworku Angular, jako motor Apache Cordova framework a stylování pomocí preprocesoru SASS. (18)

Ionic framework nabízí nejmodernější webové technologie a nativní komponenty aplikace pro vytváření vysoce interaktivních nativních a progresivních webových aplikací. Vývojář si může od nové verze mimo Angularu vybrat i jiné populární JavaScriptové frameworky pro budování vzhledu mobilních aplikací. Od 4. verze Ionicu lze využívat pro vytváření vzhledu Angular, Vue, React nebo čistý JavaScript. (18)

Cílem Ionicu je umožnit webovým vývojářům využívat webové technologie pro tvorbu mobilních aplikací, aniž by se museli učit nové technologie, ale mohli navazovat na již osvojené znalosti. Vyznačuje se svojí jednoduchostí, rychlostí vývoje a svojí velikostí. (3)

5.1.1 Architektura

Základní architektura Ionic Frameworku se dělí na několik částí. Skládá se z JavaScriptového frameworku Angular (aktuálně jako výchozí framework, ale od verze Ionic 4 lze využívat i jiné JavaScriptové frameworky), následuje využívání Apache Cordova a nástrojů pro tvorbu balíčku, závislostí balíčků, rozšíření css pomocí SASS preprocesoru a nástrojů pro automatizaci nasazení kódu. V této části jsou rozebrány kroky, jak architektura Ionic frameworku funguje. (19)



Obrázek 8: Architektura Ionic frameworku (19)

Principem Ionicu je využívat jiné frameworky a sám obstarávat jen zlomek funkcionality. Využívá tak více technologií pro tvorbu hybridních aplikací. Pro vzhled tedy využívá Angular spolu s HTML, CSS, a komunikaci s mobilním zařízením přenechává frameworku Apache Cordově. (19)

Angular je framework postavený na JavaScriptu a používá se převážně na webové aplikace. Umožňuje strukturovat aplikace a využívá návrhový vzor MVC.

Aplikace Apache Cordova je složena z webové aplikace využívající webových technologií poskytující Angular, HTML a CSS. Webová aplikace komunikuje s WebView, která pracuje s prohlížečem nativní platformy a umožní tak zpřístupnit

pro aplikaci potřebné nativní API. Následně WebView umožnuje komunikaci s Cordova pluginy, které rozšiřující funckionalitu frameworku o operačního systému. Lze tak vyvíjet vlastní pluginy pro nové nebo nepodporované nativní funkcionality v hlavním frameworku Cordova. (18)

Pluginy hrají důležitou roli. Skládají se ze dvou základních složek. První je JavaScriptové rozhraní pro přímou komunikaci s prohlížečem na dané platformě. Druhá část je nativní rozhraní, které obstarává implementaci funkcionality v jazyce platformy. Tím získá přístup k požadované funkcionalitě, které platforma nabízí. (19)

5.1.2 Poskytované API frameworku

Dostupnost API je závislá na vytvořených pluginech. Veškeré pluginy jsou tvořené pro konkrétní platformu a existuje jich celá řada. Vcelku pro všechny potřebné nativní funkce existuje již vytvořený plugin, který lze využít. (19)

S instalací čistého Apache Cordova frameworku se stáhnou následující pluginy pro podporu API.

Status baterie, Kamera, Konzole operačního systému, Kontakty, Device (umožňuje získat informace o zařízení), Akcelerometr, Kompas, Dialog, Soubory a Práci se soubory, Geolokace, Globalizace (umožňuje získat informace o zemi, jazyce a časové zóně), Zabudovaný prohlížeč, Média a zpracování, Sítově připojení, Obrazovka, Vibrace telefonu, Stavový řádek zařízení, Whitelist (určuje, ke kterým webovým stránkám má uživatel přístup). (19)

5.1.3 Vzhled a uživatelské rozhraní

Poskytované UI komponenty jsou stavební bloky vysoké úrovně vytvářející celé Ionic aplikace. Komponenty zjednodušují tvorbu rozhraní, které vypadá a funguje kvalitně. Všechny komponenty mohou být upravovány jak vzhledově, tak funkcionálně k dosažení potřebám vývojáře. Z velké části se komponenty skládají z HTML a CSS, ale v některých případech mají i JavaScriptovou funkcionalitu. Příkladem mohou být formulářové prvky, checkboxy, modální okna a jiné.

Ionic využivá technologii SASS (Syntatctically Awesome StyleSheets) pro stylování prvků. Umožnuje vývojáři využívat v CSS proměnné, podmínky,

zanořování, importy souborů nebo iterace nad polem. Vytváří se tak dobře strukturované a udržovatelné CSS pro projekt, které se může škálovat do větších projektů. Od druhé verze Ionicu nabízí předpřipravené styly pro iOS (Cupertino), Material Design pro Android a Windows. Jednotlivé moduly se aktivují podle aktuálně používané platformy. Jedna se o jednu z možností, jak dosáhnout nativního vzhledu u aplikací Ionic. Navíc každá z těchto připravených modulů může být modifikována podle vlastních požadavků. (18)

5.1.4 Silné a slabé stránky

Hlavní výhoda Ionic frameworku je v jeho jednoduchosti a snadném používání frameworku, což způsobuje využívání webových technologií HTML, CSS a JavaScript. Navíc Ionic má strukturovanou dokumentaci, kde lze snadno vyhledávat potřebné informace.

Testování za využití sdíleného odkazu. Ionic vytvořil mechanismus umožňující vývojářům poslat odkaz k testování vyvíjené aplikace. Testovací osoba může pomocí odkazu otevřít odkaz a spustit aplikace na svém zařízení. Umožnuje se tím rychle testovat bez vytvoření produkční aplikace.

Ionic poskytuje tvorbu uživatelského rozhraní pomocí drag-and-drop nástroje. Lze sestavit vzhled aplikace pouze za přetahování prvků a skládání bez psaní HTML kódu. K tomu Cordova dává na výběr ze své sbírky pluginů rozšiřujících rozhraní pro komunikaci s mobilním zařízení.

Avšak Ionic přichází i s mnoha nevýhodami. Jelikož se snaží využívat webové technologie, výsledná výkonost a rychlost aplikace oproti nativním je slabší. Nicméně ve většině případů, není výkonnostní rozdíl pozorovatelný. V případě vývoje aplikace pracující s citlivými daty, jako jsou bankovní aplikace, Ionic není vhodná volba. Neposkytuje takovou úroveň zabezpečení oproti nativnímu řešení.

Omezená nativní funkcionalita. Mohou existovat některé nativní funkce, které nemusí být dostupné v rámci Ionic frameworku. V takovém případě se musí vyvinout rozšíření k přidání podporované funkcionality. K dispozici je však mnoho pluginů pokrývajících nativní funkce.

Stejně tak Ionic není vhodný pro tvorbu komplexního grafického vzhledu, animací nebo her. Výkon není optimalizovaný ani tak výkonný jako nativní aplikace.

Proto pro vývoj herních aplikace nebo aplikace využívající vysoce komplexní grafické animace není Ionic vhodný. Nicméně, existuji JavaScriptové knihovny, které poskytují slušnou optimalizace animací na zařízení. (20)

5.1.5 Výkon aplikace

Nativní výkon aplikace je obtížné dosáhnout s frameworkem, který interpretuje kód za běhu aplikace. Aplikace napsané v Ionicu mají delší dobu načítání, špatný výkon a nejsou praktické při práci s vysokým zatížením CPU nebo pro komplexní využití grafiky. Nicméně celková velikost aplikace je podstatně menší než nativní aplikace nebo jiné kompilované hybridní aplikace.

Jelikož Ionic následuje princip zobrazování pomocí webového rozhraní, založeného na hybridním přístupu, výkon a vývoj aplikace je podobný tomu webovému. (20)

5.1.6 Reálné aplikace

Díky tomu, že Ionic existuje již několik let, má za sebou mnoho úspěšných aplikací. Mezi populární aplikace lze zařadit MarketWatch zaměřující se na akciové trhy a novinky v oblasti obchodování.

Další velmi známou aplikací pro monitorování a napomáhání při zvládání stresu je Pacifica. Nabízí úžasné nástroje pro sledování nálady uživatele, poslouchání audio cvičení a nahrávání myšlenek. Je to ukázková aplikace demonstrující využívání nativních funkcionalit za použití Ionic frameworku.

Pro vyhledávání streamovaného obsahu lze využívat aplikace JustWatch. Umožňuje vyhledávat filmy, přehlídky nebo objevovat nová a populární videa v celé řadě služeb. Využívá tak komunikaci s ostatními službami poháněnou frameworkem Ionic. (20)

5.2 Xamarin

Produkt Xamarin vznikl v roce 2011 jako nástroj pro tvorbu nativních aplikací s použitím jazyka C#. Xamarin ve své době razil cenovou politiku. Vývojář musel platit za možnost využívat daný nástroj. Nebyl tedy volně dostupný, ale i tak si získal mnoho vývojářů po celém světě. Koncem roku 2016 Xamarin odkoupila společnost

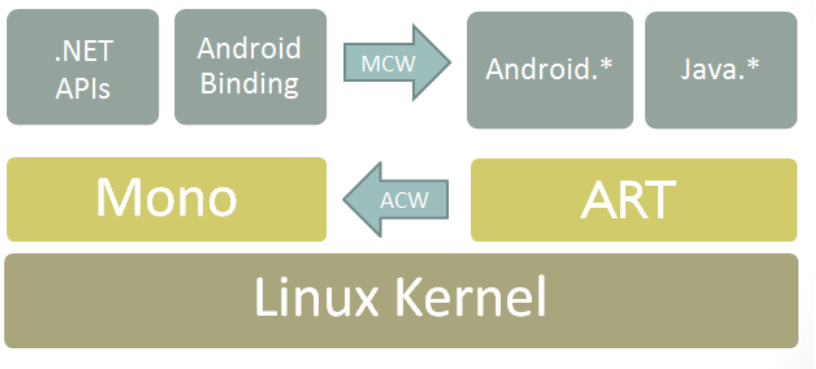
Microsoft a produkt se tak stal do určité míry volně dostupný pro menší vývojové týmy, akademické účely, výzkumy apod.

Prvotním účelem projektu bylo napojit .NET knihovny a překladače na jiné operační systémy. Cílem Xamarinu bylo vyvíjet, překládat a spouštět aplikace psané v C# pro platformy založené na Unixu. Xamarin přišel s možností vyvíjet C# aplikace pro jiné operační systémy a pojmenoval ji Mono (iOS – Mono Touch, Android – Mono for Android).

S využíváním překladače pro každou platformu je možné převést vývojový kód do nativního strojového kódu. Tento proces umožňuje přístup k funckionalitám mobilního zařízení pro jednotlivé platformy. (17)

5.2.1 Architektura

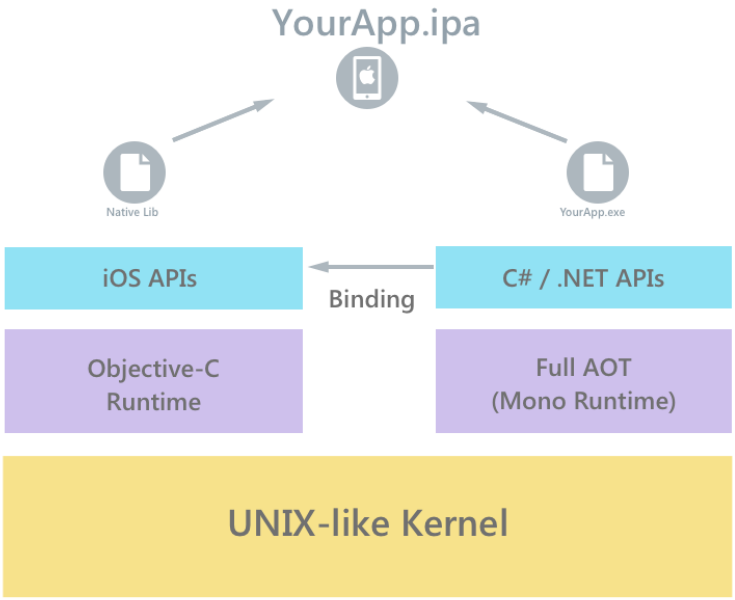
Architektura Xamarinu se skládá ze dvou hlavních částí. Obsahuje Xamarin.Android a Xamarin.iOS. Vývoj se podobá nativnímu vývoji pomocí nativních nástrojů. Jelikož využívá jazyk C#, vývojář může následně sdílet logiku mezi platformami. Je to běhové prostředí přidávající vrstvu mezi zdrojový a strojový kód a balíček knihoven. (21)



Obrázek 9: Xamarin architektura pro Android

Procesy pro kompilaci kódu do nativního prostředí se liší podle platformy. Android prostředí běží bok po boku s Android Runtime (ART) virtuálním strojem. Toto prostředí následně využívají vývojáři pro přístup k nativním API běžícím na vrcholu linuxového jádra. Jednotlivé API nejsou dostupné napřímo, musí se k nim přistupovat pomocí Android Runtime. Vývojáři tedy přistupují k funkcím operačního systému využíváním.NET API, které znají nebo využívají dostupné třídy v Androidu poskytující most k JavaAPI vystavené v Android Runtime. (22)

Architektura Xamarinu pro iOS je rozdílná oproti Androidu. U iOS zařízení všechny skripty musí být spouštěny na Apple Webkit frameworku.



Obrázek 10: Xamarin architektura pro iOS

Pro vývoj na platformu iOS je potřeba provést přechodový jazyk. Xamarin iOS kompilátor zkompiluje zdrojový kód do přechodového jazyka, který je známý jako CIL (Common Intermediate Language). Po překladu je potřeba provést znovu kompilaci, ale tentokrát do nativního strojového kódu běžícího na iOS zařízení. Výstupem kompilace je nativní aplikace, která má přístup k nativní funckionalitě mobilního zařízení. (23)

Xamarin Framework SDK umožňuje přidávat do projektu externí kompatibilní knihovny psané v jazyce C#. Kompatibilitu lze snadno ověřit pomocí databáze knihoven přímo pro Android nebo iOS platformu. Od verze 4.3 lze také přidávat do projektu i knihovny psané v nativním jazyce platformy – Java pro Android nebo Objektové C pro iOS. (21)

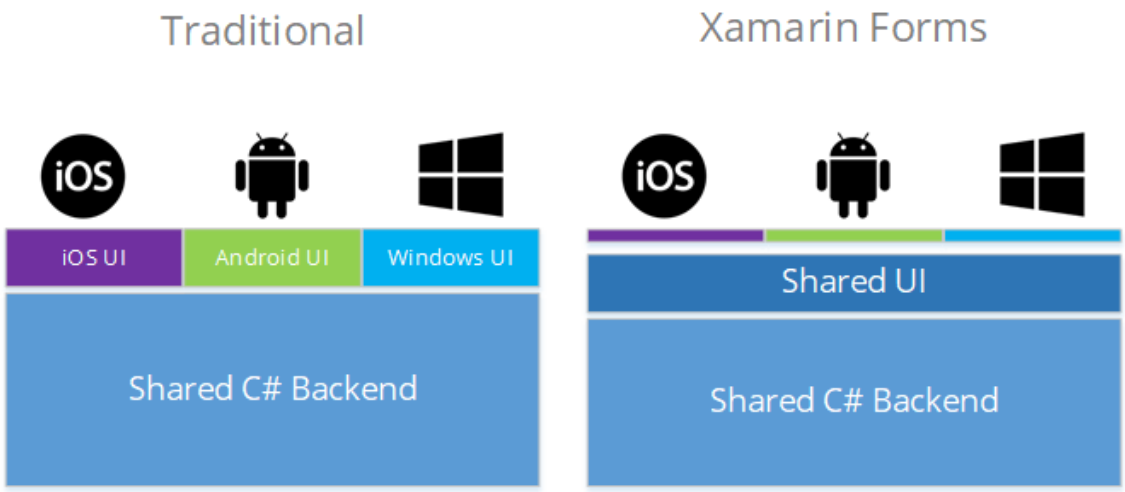
5.2.2 Poskytované API frameworku

Aplikace vytvářené technologií Xamarinu lze klasifikovat jako nativní. Proto není překvapením, když při tvorbě aplikace lze přistupovat ke všem dostupným funkcím mobilního zařízení. Xamarin poskytuje bohaté API ke zpracování služeb jako je kamera, souborový systém, stav telefonu nebo používání NFC.

V kódu lze využívat třídy pro nativní funkce, psané čistě v knihovně. Pokud je potřeba využívat obecná řešení, lze se podívat i po rozšiřujících pluginech, které zjednodušují požadovanou implementaci. Vcelku s frameworkem Xamarin lze využívat veškeré nativní funkce nabízené mobilní platformou a zařízením. (17)

5.2.3 Vzhled a uživatelské rozhraní

Pro každou platformu Xamarin poskytuje využívání základních prvků uživatelského rozhraní. Nedílnou součástí pro vývoj cross-platformní aplikace v Xamarinu je rozšíření frameworku Xamarin.Forms pro rychlé vytváření uživatelského rozhraní.



Obrázek 11: Architektura UI Xamarin.Forms

Xamarin.Forms poskytuje vlastní abstraktní vrstvu sdílenou mezi všechny platformy. Vrstva je následně vykreslena s použitím ovládacích prvků pro iOS, Android nebo Windows. Tím se usnadní sdílet velkou část zdrojového kódu pro uživatelské rozhraní, a přitom si ponechat vzhled a pocit nativní aplikace. Jak je Xamarin.Forms využit v aplikace, je znázorněno na Obrázek 11.

Framework svým způsobem umožnuje rychlé prototypování aplikací, které se postupně mohou rozvinout do komplexnějších aplikací. Jelikož jsou veškeré vykreslené prvky převedeny do nativního kódu, není potřeba se u Xamarin.Forms zatěžovat kompatibilitou s prohlížečem, omezením na API nebo sníženým výkonem.

Existují dva způsoby tvorby UI při použití Xamarin.Forms. Lze zapisovat rozhraní čistě v jazyce C# bez jakéhokoliv rozšíření, nebo využít XAML jazyk, který se používá k popisu uživatelského rozhraní. (24)

5.2.4 Silné a slabé stránky frameworku

Výsledná aplikace je velmi rychlá. Aplikace běží téměř stejně rychle jako nativní aplikace. Jelikož .NET podporuje potenciál znovupoužití serverového kódu, v případě dostupnosti internetu. Lze dosáhnout téměř maximálního znovupoužití kódu při použití Xamarin.Forms. Jako vedlejší efekt tato technologie snižuje počet vyskytnutých chyb v aplikaci. Framework zobrazuje standardní nativní uživatelské ovládací prvky, které odpovídají vzhledovým konvencím platformy.

Za základní funkci aplikace se dá považovat stoprocentní přístup k základním schopnostem zařízení a akceleraci hardwaru. Xamarin přidal i podporu pro ASP.NET Razor, šablonovací systém pro generování HTML v aplikacích.

Vzhledem k tomu, že Xamarin se stále vyvíjí a vydávají se nové aktualizace, je zapotřebí častých aktualizací během vývoje. Vývojář by měl také vědět o ztrácení paměti během skrytých chyb v implementaci uživatelského rozhraní. Xamarin.Forms nevyužívá uživatelské rozhraní pro vzhled, k zobrazení vzhledu a uživatelského rozhraní je nutná kompilace. Velká část Xamarin API jsou chráněny, což ponechává menší prostor při hledání řešení v případě chyb ve frameworku. Další problém je s licencí. Nejlevnější licence pro malé vývojáře nezahrnuje podporu pro Visual Studio, což může být omezením pro začínající projekty. (17)

5.2.5 Výkon aplikace

Oproti tradičnímu hybridnímu řešení založeném na webových technologií, aplikace postavené na Xamarinu mohou být klasifikované jako nativní. Měření výkonu jsou srovnatelná s Javou u Android platformy a Objektivním-C, nebo Swiftem u iOS aplikací. Navíc výkon Xamarinu je neustále vylepšován tak, aby plně odpovídal standardům nativního vývoje. (17)

Při využívání knihovny Xamarin.Forms se bohužel zpomaluje výkon výsledné aplikace, nicméně i tak je zde rozdíl vcelku zanedbatelný. Důvodem rychlého běhu aplikace je optimalizace .NET nebo Moro runtime. Výsledná velikost má okolo 12-

16 MB, kvůli samotné velikosti frameworku. Nicméně s rostoucí aplikací se rozdíl velikosti aplikace o tolik měnit nebude. (24)

5.2.6 Reálné aplikace

Jak již bylo napsáno, Xamarin je populární nástroj pro tvorbu hybridních mobilních aplikací s více než miliony stažení. Od malých týmů vyvíjejících klientské aplikace, až po velké společnosti vyvíjející ty komplexnější.

Jako referenční aplikace lze zařadit z kategorie podnikání – Insightly. Více než 1,5 miliónu uživatelů po celém světě důvěřuje Insightly, které pomáhá rozvíjet jejich podnikání řízením interakcí se zákazníky, ziskem potencionálních zákazníků, prodejními příležitostmi a projekty.

Pomocí Xamarin lze vyvinout i herní aplikace jako je Transistor. Jedná se o scifi akční hru vyvinutou prvně pro počítače a Xbox. Následně byla hra přetvořena v Xamarinu do mobilní aplikace se zachovanou prezentací a souboji jako původní originální hra. Pomocí Xamarinu lze vytvářet jednoduché i komplexní aplikace, ať už se jedná o aplikace se zvýšenou bezpečností, nebo herní aplikace s vysokými požadavky na grafiku. (17)

5.3 React Native

React Native je volně dostupný, multiplatformní framework pro vytváření nativních aplikací pomocí jazyka JavaScript a knihovny React. Je vytvořen a podporován společností Facebook, Inc. Tento framework umožňuje vývojáři vytvářet nativní aplikace pro Android a iOS s rozšířením umožňujícím vývoj i pro systém Windows.

Jako syntax React Native využívá vlastní specifický jazyk pro psaní komponent, který je následně transpilovaný do JavaScriptu za použití Babelu[8]. React Native nabízí deklarativní UI komponenty, které jsou kompilované do nativních prvků pro konkrétní platformu. (18)

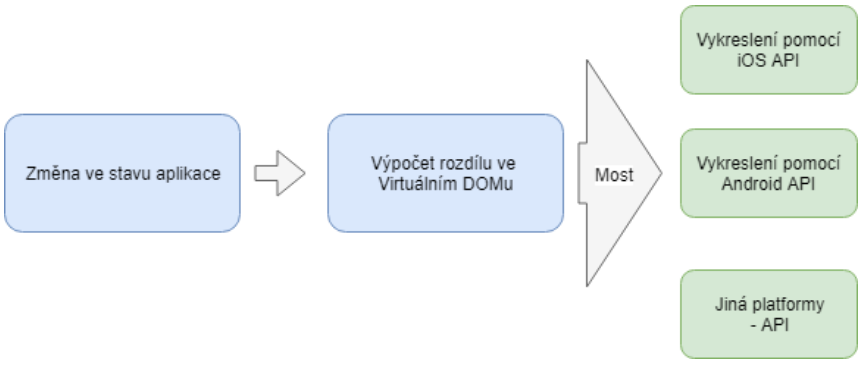
[8] Babel – nástroj pro konverzi nového (ECMAScript 2015+) Javascriptu na zpětně kompatibilní verze jazyka (25)

Důležité je zmínit, že React Native se může pochlubit velmi rozsáhlou a stále rostoucí komunitou. Výsledkem je četné množství článků na téma React Native, publikujících různé způsoby implementace frameworku, a stejně tak velký počet otázek ohledně na serveru „Stack Overflow“[9] pomáhající ostatním vývojářům při řešení problémů. (18)

5.3.1 Architektura

React Native staví na stejných konceptech jako React, ale namísto vykreslování HTML elementů, používá základní stavební prvky nativních platforem. Konečným výsledkem je umožnit vývojářům vyvíjet mobilní aplikace za využívání React syntaxe. (18)

Celý princip vykreslování React Native spočívá na komunikaci skrze API a techniky Virtuálního DOMu. Jedná se o techniku optimalizace výkonu, ale je toho mnohem víc než jen to. Je to abstraktní vrstva mezi kódem, který popisuje, jak má vypadat aplikace, a skutečným vykreslením těchto prvků. React Native funguje na principu vykreslování uživatelského rozhraní na hlavním podprocesu s příkazy odesílanými z kódu JavaScriptu. Jako takové musí existovat prostředí pro komunikaci mezi vlákny. K tomuto principu React Native využívá techniky takzvaného Mostu. Most slouží jako prostředník pro komunikaci mezi uživatelským rozhraním a nativním API platformou. Základní myšlenka je zobrazena na Obrázek 12.



Obrázek 12: Princip architektury React Native

[9] Stack Overflow –místo pro otázky a odpovědi mezi programátory.

Namísto JavaScriptového kódu, React Native volí API rozhraní dané platformy. Například na platformě Android Most komunikuje s Java API rozhraním pro vykreslování nativních prvků.

Pomocí komunikace přes Most je možné využívat nativní funkcionalitu platformy. React Native tak neomezuje funckionalitu a při integrování nové platformy nebo jiného rozhraní, je pouze potřeba připravit daný Most pro komunikaci. Příkladem je aktuální podpora pro Televizní aplikace, které v React Native lze vyvíjet. (26)

5.3.2 Poskytované API frameworku

React Native architektura umožňuje přistupovat k funkcionalitám mobilního zařízení. V samotném základu nabízí React Native základní funkcionality jako je Status Bar, Klávesnice, Kamera, Notifikace, Geolokace a mnohem více. V oficiální dokumentaci si lze najít v kompletní seznam podporovaných API. (26)

V případě potřeby využívat nadstandardní funckionalitu nebo nepodporovanou funkcionalitu v základu frameworku React Native, je možnost si stáhnout pluginy rozvíjející základní schopnosti. Komunita kolem frameworku je velice aktivní a stále se vyvíjí nové pluginy otevírající nové možnosti frameworku.

Pokud by neexistovala žádná knihovna nebo plugin pro potřebovanou funkcionalitu mobilního zařízení, React Native poskytuje možnost využívat přímo nativního jazyka a aplikovat plnou funckionalitu mobilního zařízení. (26)

5.3.3 Vzhled a uživatelské rozhraní

Aplikace vyvíjené pomocí frameworku React Native se mohou pochlubit propracovaným využíváním nativních komponent mobilního zařízení. React Native využívá API pro komunikaci s nativní funkcionalitou a umožnuje tak plně využívat odpovídající vzhled pro platformu. (26)

Momentálně existuje mnoho rozšíření pro nativní uživatelské rozhraní doplňující React Native. Už v základu React Native nabízí spoustu skladatelných a stavových řešení pro návrh uživatelského rozhraní. Ovšem přizpůsobení ke konkrétním případům se může stát obtížným úkolem, pokud se budou muset vyvíjet

komponenty znovu. Framework je naštěstí podporován pulzující komunitou, jejíž členové vyvinuli řadu možností pro UI Komponenty. Mezi nejlepší se mohou řadit:

- NativeBase – využívá již hodně předpřipravených knihoven s podporou pro iOS/Android.
- React Native Elements – slušný seznam menších komponent i komplexních.
- React Native Material Kit – knihovna s inspirujícím Material Designem a s rozšířeným rozhraním pro tvorbu vlastních komponent.

5.3.4 Výhody a nevýhody

React Native nabízí vysokou kvalitu uživatelského rozhraní a lze tak docílit nativního vzhledu i propracované animace. Stejně tak umožnuje přistupovat k nativním funkcionalitám mobilního zařízení. Framework nabízí již předem připravené elementy pro tvorbu prvků, které lze sdílet mezi platformami. V případě potřeby React Native nabízí přímé použití nativního kódu ke zvýšení optimalizace aplikace. Pomocnou funkcí při vývoji je nabízený Hot Reload, umožňující před načíst aplikace rychle bez opětovné kompilace.

Používáním frameworku React Native lze vyvíjet velmi kvalitní aplikace. Avšak aplikace postavené na React Native nejsou tak výkonné jako nativní. Jelikož aplikace v React Native jsou psané v JavaScriptu, je nutné dodržovat čistý a optimální kód pro dosažený rychlých výkonů aplikace a plynulých animací. Tvorba navigačního menu pro mobilní aplikace může být komplikovanější a nedosahuje takového výkonu při přechodech jako u nativních aplikací.

V jistých situacích je nutnost si vyvinout vlastní nativní komponentu. Poskytované komponenty někdy nesplňují požadavky na logiku a je nutné vyvinout komponentu pro každou platformu odděleně. (25)

5.3.5 Výkon aplikace

Jedním z nejzajímavějších přínosů React Native je poskytování vysokého a téměř nativního výkonu.

I když React Native není nativní jazyk pro mobilní aplikace, umožňuje dosáhnout podobného výkonu jako skutečně nativní aplikace s výhodami frameworku – zvýšení produktivity, rychlejší a snadnější vývoj. Umožňuje optimalizovat výkon aplikace zahrnutím nativního kódu v některých oblastech aplikace.

Ačkoliv oficiální stránky tvrdí, že React Native poskytuje nativní výkon, není to nutně pravda. React Native v některých případech trpí problémy s výkonem. Může se jednat o dobu spuštění aplikace, únik paměti, využívání CPU nebo celková velikost aplikace. Důvodem může být špatné používání komponent, velké množství prvků, neoptimálně navržená architektura aplikace. React Native poskytuje možnost zvýšit efektivnost vývoje, proto je následně potřeba se zaměřit na psaní optimálního kódu ke zvýšení výkonnosti. Výkon aplikace může ovlivňovat:

- Špatně napsaný JavaScript (spouštění nepotřebných skriptů)
- Přetížení Mostu (namísto posílání více požadavků, je optimální seskupit dotazy do dávky)
- Čtení celé paměti Cache, namísto aktuálně potřebované.

I přes všechny omezení se ale může vždy najít řešení k dosažení výkonných výsledků. (26)

5.3.6 Reálné aplikace

Jelikož React Native vzniklo pod společností Facebook, není žádným překvapením, že aplikace Facebook je napsaná pro mobilní telefony za využití frameworku React Native. Spolu s tím, další populární aplikace přes sociální sítě Instagram je vyvinuta právě za pomoci React Native. Ukazuje zde svoji sílu využívat nativní prvky mobilního zařízení. Uživateli dovoluje používat prvky jako je Kamera, Galerie nebo Geolokaci pro přidání aktuální polohy. Stejně tak využívat mapy a hromadu dalších menším funkčních komponent – například našeptávač.

React Native zvolilo velké množství velkých společností. Lze sem zařadit aplikace jako jsou Teslo, Uber, Walmart, Skype a hromadu dalších. React Native demonstruje, že zvládne vytvářet mobilní aplikace všeho druhu. (26)

5.4 Flutter

Flutter je multiplatformní framework, kde hlavním cílem je vývoj vysoce výkonných mobilních aplikací. Flutter byl zveřejněn společnostní Google v roce 2016, ale stabilní verze vyšla až koncem roku 2018. Nejen, že aplikace psané ve Flutteru mohou běžet na platformách Android a iOS, ale také na připravovaném operačním systému nové generace Fuschia. (27)

Flutter je jedinečný svým využíváním vlastních komponent za pomoci vlastního vysoce výkonného vykreslovacího nástroje, namísto toho, aby se spoléhal na webové vykreslení nebo komunikaci s nativním rozhraním. Tato povaha poskytuje možnost vytvářet aplikace, které jsou výkonnostně srovnatelné s nativními aplikacemi. Každý kód ve Flutteru je psaný za pomoci programovacího jazyku Dart, následně kompilovaného do nativního kódu.

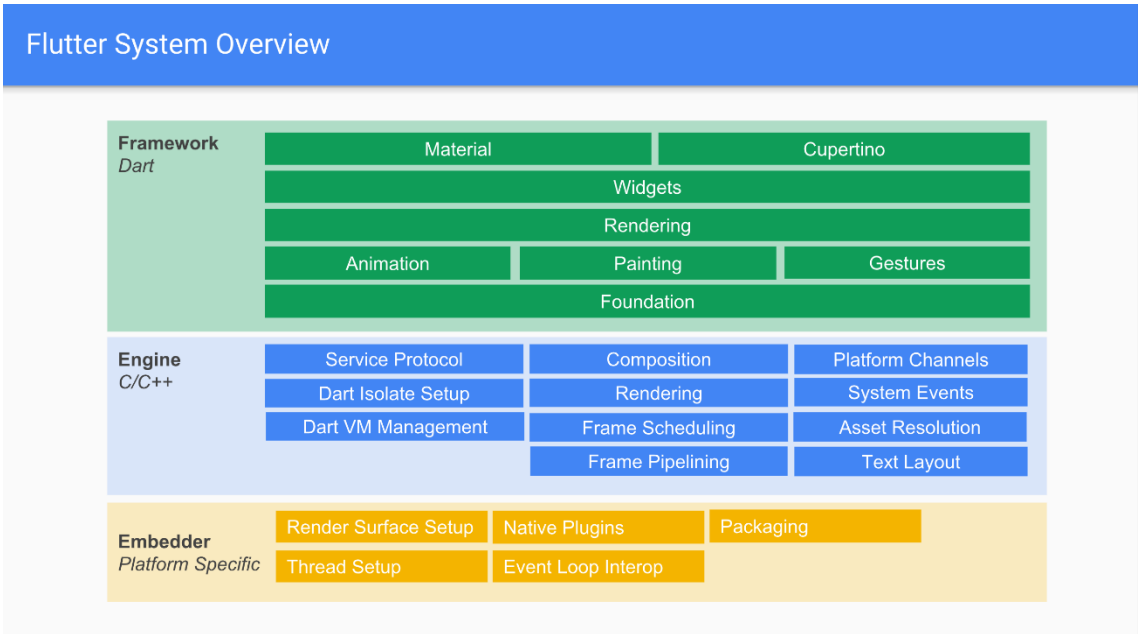
Při vývoji aplikace Flutter poskytuje hot-reload technologii. Jedná se o implementaci vkládání aktualizovaného zdrojového kódu na běžící aplikaci bez změny vnitřní struktury. Umožňuje tak aktualizovat kód při vývoji a zvýšit efektivitu vývojového procesu.

Hlavním principem Flutteru je to, že všechno je skládáno z widgetů. Jsou proto považovány za nejdůležitější prvek aplikace Flutteru. (26)

5.4.1 Architektura

Jádro Flutteru je přenosný Runtime pro vysoce kvalitní mobilní aplikace. Implementuje základní Flutter knihovny, včetně animací a grafiky, souborových a síťových I/O, podpory přístupnosti, architektury pluginů a Dart runtime pro vývoj, kompilaci a spouštění Flutter aplikací.

Jádro Flutteru se skládá z technologií: Skia – 2D grafická zobrazující knihovna, programovací jazyk Dart, VM pro správu paměti objektově orientovaného jazyka a hostující je v uživatelském rozhraní. Základní přehled je zobrazen na obrázku Obrázek 13.



Obrázek 13: Architektura Jádra Flutteru

Flutter využívá Dart pro vytváření komponent, který následně přivede k životu pomocí Skia knihovny. Flutter také obsahuje moderní reaktivní framework. Pro vykreslení uživatelského rozhraní využívá Skia a aplikační kód uvnitř odlehčeného Dart VM. Základní Framework je napsána v jazyce Dart, avšak vykreslovací jádro je implementováno v C++. (27)

Zdrojový kód Dart je zkompilován do nativního kódu pomocí kompilace Ahead of Time. Stále však potřebuje ke spouštění Dart VM (Virtuální počítač). Na Androidu je C / C++ kód kompilován s Android NDK, a všechen Dart kód je zkompilovaný AOT do nativního. Na iOS je jádro C/C++ kódu implementováno s LLVM (Low Level Virtual Machine) a kompletní Dart kód je také pomocí techniky AOT kompilovaný do nativního kódu. Aplikace v obou případech běží v nativním prostředí. (8)

5.4.2 Poskytované API

Flutter nabízí vývojářům přístup k některým specifickým službám a API v základním balíčku. Nicméně Flutter se snaží vyhnout problémům „pokrytí všech případů“ s podporou většiny API, takže nemá v úmyslu budovat rozhraní API pro všechny nativní služby.

Řada poskytujících služeb pro platformy a rozhraní už je vyvinuto ve formě balíčků dostupných na webu Pub (balíčky podporované pro Dart a Flutter).

Používání existujících balíčku je navíc velmi jednoduché. Stačí přidat do souboru název a IDE se postará o automatické stažení.

Nakonec Flutter doporučuje vývojářům, aby využívali asynchronní systém pro předávání zpráv k vytváření vlastní integrace funckionality platformy. Flutter připravil možnost pro vývojáře integrovat vlastní funckionalitu a mohou tak vystavovat tolik, kolik budou potřebovat rozhraní platformy, a vytvářet vrstvy abstrakcí, které nejlépe vyhovují danému projektu. (26)

5.4.3 Vzhled a uživatelské rozhraní

Využívá vlastní vykreslovací technologie pro uživatelské rozhraní. Nevyužívá na obalení komponenty pro specifickou platformu. Umožnuje vyvíjet vzhled pro každou platformu na pixel-perfect aplikace jak pro iOS, tak pro Android. Pro každou platformu má vlastní knihovnu se specifickými komponenty. Pro iOS využívá Cupertino a pro Android Material Design knihovny. Stejně tak Flutter vyniká svým rozsáhlým množstvím komponent pro tvorbu vzhledu.

I když Flutter umožňuje vzhledově přesné aplikace pro platformu, dosáhnout toho nemusí být jednoduché. Zaniká tak znovu použitelnost uživatelského rozhraní. I přesto nabízí vestavěné a výkonné animace s vlastním povědomím o platformě. (26)

5.4.4 Výhody a nevýhody frameworku

Výhodou frameworku je výkon výsledných aplikací. Dart využívá AOT kompilaci, která umožňuje přímou komunikaci s nativní platformou.

Funkce Hot reload pozitivně zvyšuje rychlost vývoje aplikací. Každá změna v kódu je automaticky vložena do již běžící aplikace bez nutnosti opětovného spouštění kompilace.

Celá řada předem připravených widgetů. Framework implementuje Material Design widgety pro Android a Cupertino widgety pro iOS, avšak vývojář se nemusí přesně řídit těmito pravidly. Cupertino widget bude vypadat a fungovat stejně tak i na Android platformě.

Oproti tomu Flutter má i své nedostatky a nevýhody.

Nevýhodou je velikost aplikace. Jelikož Flutter nespoléhá na propojení mezi vlastním kódem a nativní funkcionalitou, všechny UI komponenty jsou tvořeny v aplikace samotné. Ve výsledku tvoří velkou aplikaci. Průměrná velikost Flutter aplikace je okolo 5 MB, proto je dobré si promyslet smysl aplikace při volbě frameworku.

Jak již bylo zmíněno, Flutter vykresluje vlastní uživatelské rozhraní, nevyužívá nativní komponenty. Pokud by přišla grafická změna v UI některé platformy, Flutter by nereagoval na změnu a stále by vykresloval stejný vzhled. Musela by proběhnout aktualizace od vývojářů a následná aktualizace projektu s vydáním nové verze aplikace.

Menší počet pomocných knihoven rozšiřujících framework vytvořený komunitou. I když framework oproti ostatním je poměrně nový, stále nejsou vytvořeny podporované systémy pro Flutter. Příkladem může být integrovaný systém placení, software pro automatizování daní nebo Apple a Android TV. (26)

5.4.5 Výkon aplikace

Pokud jde o výkon, přístup Flutteru je zcela odlišný od přístupu ostatních multiplatformních frameworků. Aplikace Flutter je sestavena pomocí C / C++, takže je blíže strojovému jazyku a poskytuje lepší nativní výkon. Nejsou zkompilovány pouze komponenty uživatelského rozhraní, ale celá aplikace.

Programovací jazyk Dart je sám o sobě výkonným jazykem a z hlediska výkonu má Flutter převahu.

Jelikož Flutter nevyužívá komunikace s nativním API zařízením nebo jiné metody konverze, ale vykresluje si všechno za využití Skia 2D knihovny, dosahuje výkonných mobilních aplikací.

Pro zlepšení optimalizace výkonů aplikace Flutter nabízí profilování výkonnosti Flutteru. Při zapnutí se vykreslí v horní části aplikaci zatíženost a statistiky aplikace – FPS a CPU. (26)

5.4.6 Reálné aplikace

Jelikož Flutter framework je mladá technologie a mezi vývojáře se dostává až poslední rok, není až tak rozsáhlá mezi většími společnostmi. Komunita kolem Flutteru se však se rychle rozrůstá, proto vznikají neustále nové aplikace.

Mezi hlavní reference se řadí aplikace pro Alibabu, jako jedna z největších eCommerce společností v Číně. Vytvořila tak aplikaci, kterou používá více jak 50 miliónů uživatelů.

Google, jakožto tvůrce Flutteru využívá framework pro většinu svých interních projektů. Příkladem je Google Adds, aplikace napomáhající pro správu reklamních kampaní.

Mezi jiné populární aplikace lze zařadit AppTree, Hamilton Musical nebo Tencent. Lze očekávat větší nárůst nových velkých aplikací. (26)

5.5 Porovnání frameworků

Pro lepší rozlišení mezi jednotlivými vlastnostmi výše uvedených frameworků, je lepší rozdělit tyto funkce do menších podsekcí. Vybrané vlastnosti jsou rozděleny na základě hledisek, které pomáhají rozeznat jejich specifické kvality. Zejména se jedná o následující hlediska:

- Výkon aplikace
- Architektura frameworku
- Popularita
- Podporované API
- Přepoužitelnost kódu
- Tvorba uživatelského rozhraní

Při rozlišování frameworků je potřeba si porovnat i méně podstatné aspekty jako je komplexita, stabilita nebo využití frameworků.

5.5.1 Výkon aplikace

Základní požadavek na aplikaci je její samotný výkon. Mezi výkon lze řadit doba spuštění aplikace, rychlost zpracování a interakce nebo FPS animací.

Ionic v oblasti výkonu pokulhává. Výkon aplikace není srovnatelný s nativní aplikací. Důvodem je využití interpretovaného jazyka HTML a CSS ke stavbě GUI v rámci Apache Cordova. Tato architektura snižuje rychlost a rovněž nevyužívá nativní komponenty.

Aplikace Xamarin oproti Ionicu mají lepší výkon a neustále se zlepšují, aby odpovídaly standardům nativního vývoje. S pomocí Xamarin frameworku, aplikace pro Android a iOS jsou plně nativní s využitím rozhraní platformy. Pomocí knihovny Xamarin.Forms lze vyvíjet nativní mobilní aplikace se sdíleným uživatelským rozhraním a současně také přidávat nové funkce.

React Native nabízí téměř nativní výkon vyvíjených aplikací. Navíc umožňuje psaní v nativním kódu dané platformy a zvyšuje tak výkon aplikace. Umožnuje vyvinout jednu část aplikace v React Native a druhou část v nativním kódu pro dosažení většího výkonu aplikace.

Flutter aplikace mají dostatečně vysoký výkon a vývojář se tak nemusí zabývat ruční optimalizací. Výkon aplikací vyvinutých v tomto frameworku je uspokojivý i z uživatelského hlediska. Flutter umožňuje využívat nativní kód pro dosažení lepšího výkonu. Widgety zahrnují rozdíly v platformách, jako je posouvání, navigace nebo vykreslení písma. V souhrnu Flutter poskytuje bezproblémový výkon v systémech Android i iOS.

5.5.2 Architektura frameworku

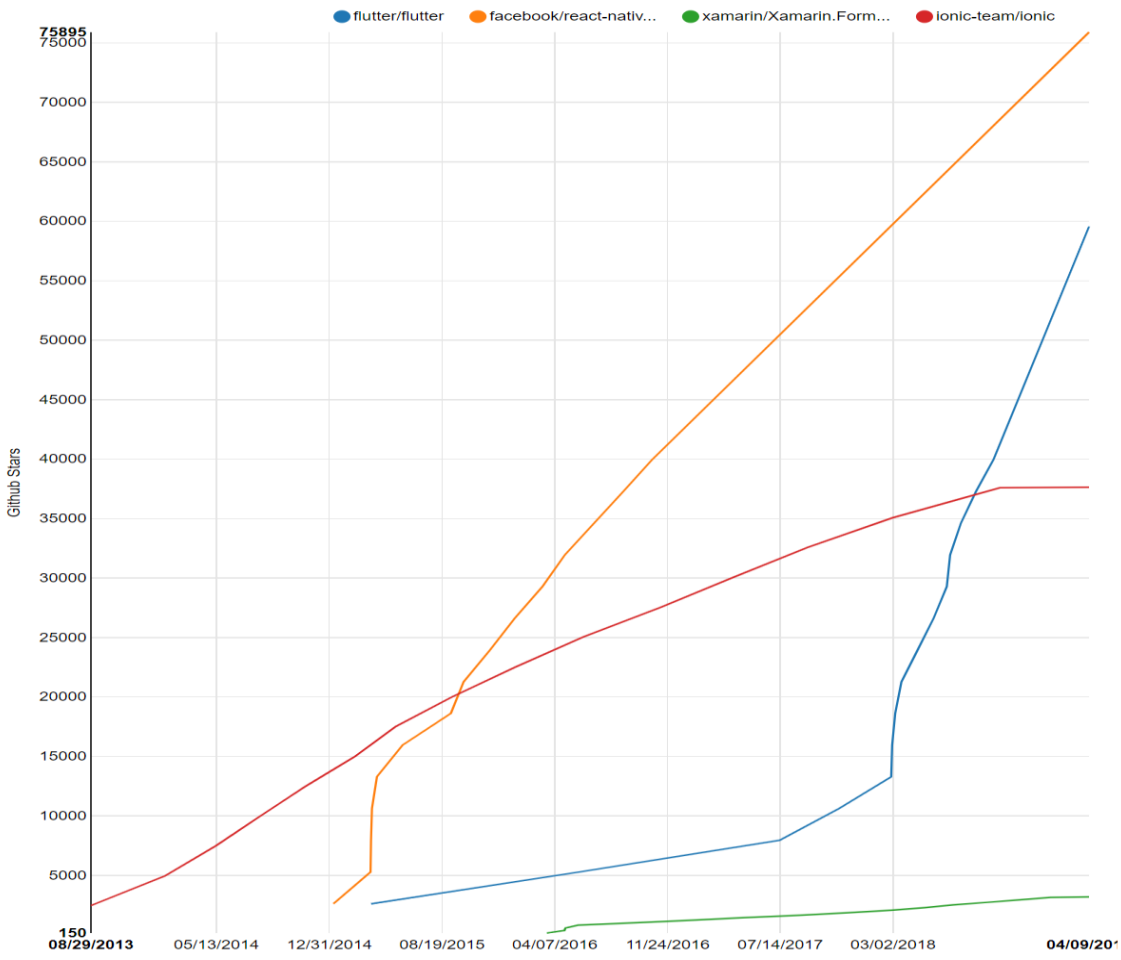
Architektura React Native je prostá. Framework komunikuje s nativním rozhraním pomocí technologie mostu. Umožňuje tak plně využívat nativní funkcionalitu. Další výhodou je možnost opětovného použití nativních prvků komponent a pro přidání nové podpory pro platformu se vytvoří pouze nový komunikační Most pro cílovou platformu.

Oproti tomu Xamarin architektura vyžaduje kompilaci z nativního kódu do přechodového, a ten je následně kompilován do nativního kódu. Využívá Xamarin.iOS a Xamarin.Android produkty založené na Mono. Veškerý kód je psaný v jazyce C#. Přidání podpory pro další framework by znamenalo nutnost naprogramovat další kompilátor, což je velmi náročná úloha.

Architektura Ionicu je založena na webových technologiích zakomponovaných do kontejneru Apache Cordova. Využívá vývojářům známé technologie spolu s nativní funckionalitou platformy. Tento přístup si získal popularitu svojí jednoduchostí, ale za cenu nižšího výkonu aplikace.

5.5.3 Popularita

Mít dynamický ekosystém je něco, za co stojí být vděčný. Na následujícím grafu lze vidět aktuální popularitu frameworků podle hvězd na GitHubu.



Obrázek 14: Průběžná popularita frameworků na GitHubu

React Native převzal první místo mezi vývojáři. Síť vývojářů rychle roste a od nynějška má mnoho zkušených React Native inženýrů. Umožňuje snadné spuštění projektu React. Využívá populární knihovny a nejzákladnější jazyk pro vývoj webu. Tyto vlastnosti z něj činí silnou platformu a jsou důvodem jeho slávy.

Největším vzrůstem popularity je Flutter. Za poslední rok vstoupil do povědomí vývojářů a ti si jej oblíbili. S rostoucí popularitou pomalu dotahuje React Native. Google se snaží svůj framework Flutter intenzivně podporovat a je pro něj prioritou do budoucna, hlavně kvůli nově vyvíjenému operačnímu systému Fuschia.

Ionic si drží svoji pozici klidným vzrůstem počtu vývojářů. Umožňuje vývojářům vytvářet nativní mobilní aplikace nejrychlejším možným způsobem.

Xamarin je spíše tichý framework. Microsoft stále vynakládá velké úsilí na rozvoj komunity Xamarin. Za poslední léta společnost Microsoft umožnila částečný open source pro framework, který přilákal nové vývojáře.

5.5.4 Podporované API

V případě podpory rozhraní, každý framework poskytuje dostatečnou podporu základních funkcí. Porovnává se zde základní podpora jednotlivých funckionalit. Zda framework poskytuje funkcionalitu v základu nebo je nutné si stáhnout potřebné rozšíření. V tabulce č. 2 jsou uvedené základní požadované funkcionality.

Funkce	Ionic	Xamarin	React Native	Flutter
Navigace	Ano	Ano	Plugin	Ano
Kamera	Ano	Ano	Ano	Plugin
Video	Ano	Ano	Plugin	Plugin
Zvuk	Ano	Ano	Plugin	Plugin
Kontakty	Plugin	Ano	Plugin	Plugin
Zařízení	Ano	Ano	Plugin	Plugin
Souborový systém	Ano	Ano	Plugin	Plugin
Geolokace	Ano	Ano	Ano	Plugin

Tabulka 2: Porovnání základní funkcionality frameworků

V další tabulce jsou porovnávány komplexnější a nové funkcionality. Dovoluje tak sledovat aktuální stav frameworku. Zda se autoři snaží framework pravidelně aktualizovat, jestli podporuje požadované rozhraní či jeli komunita aktivní.

Funkce	Ionic	Xamarin	React Native	Flutter
NFC	Ano	Ano	Plugin	Plugin
Notch support[10]	Ano	Ano	Ano	Ano
Grafika 3D	Plugin	Plugin	Plugin	Plugin
Platební brána	Plugin	Plugin	Plugin	Plugin

Tabulka 3: Porovnání rozšířené funkcionality frameworků

Z tabulky lze pozorovat, že všechny frameworky poskytují veškerou zmíněnou funkcionalitu. Pro všechny případy je buď podpora nativní nebo lze využít rozšíření. Z podpory funckionality platformy všechny frameworky splní požadované funkce.

5.5.5 Přepoužitelnost kódu

Je to jeden ze základních požadavků a důvod pro vznik multiplatformních frameworků. Určuje, kolik kódu lze sdílet mezi jednotlivými platformami. Z logické strany je možné sdílet kód bez většího problému, ovšem pokud se nevyužívá možnost využít nativního kód pro optimalizaci aplikace. Většina rozdílů se týká vzhledu a dosažení pocitu nativní aplikace.

Ionic je při sdílení kódu univerzální. Bez ohledu na daný operační systém bude na každém z nich fungovat stejně dobře. Ovšem styly mohou být určeny pouze pro danou platformu. Lze tak upřednostnit připravené téma pro platformu.

Všechny aplikace psané v Xamarinu jsou vyvíjeny v jazyce C# a kompilovány do nativního kódu. V průměru se sdílí až 96 % zdrojového kódu spolu s použitím Xamarin.Forms, který umožňuje sdílet vzhled pro jednotlivé platformy. (24)

React Native využívá nativní komponenty napsané v Objective-C, Swift nebo Java ke zvýšení výkonu aplikace. Avšak tyto komponenty nelze znovu použít na jiných platformách. Následkem musí vývojáři udělat práci navíc pro dosažení podpory na jiných platformách. Nicméně, přijmout tyto nativní komponenty znamená, že 90 % kódu může být znovu použit.

[10] Notch – Jedná se o nový layout, kde vršek mobilu je vyříznutý a je potřeba k tomu přizpůsobit vzhled.

Přepoužitelnost u frameworku Flutteru může být různá. Díky logice, struktuře dat a modelů lze docílit až plného sdílení. Ovšem při tvorbě vzhledu se může tato vlastnost snadno zhoršit. Každá operační systém má svoje definované rozšíření pro dosažení požadovaného vzhledu. Menší komponenty lze pak sdílet napříč platformami a docílit kódu bez velké duplicity.

5.5.6 Tvorba uživatelského prostředí

Uživatelé posuzují aplikaci během několika prvních sekund používání, proto uživatelské rozhraní tvoří důležitou část aplikace.

V porovnání s ostatními frameworku Ionic nevyužívá nativní prvky a vše zobrazuje za využívání HTML element stylovaných pomocí CSS. Pro dosažení nativního vzhledu je potřeba vynaložit velké úsilí.

Xamarin umožňuje vytvářet uživatelské rozhraní dvěma způsoby. První je klasickým využitím separace stylů pro každou platformu zvlášť. Tento způsob není účinný a vzniká duplicita v kódu. Dalším způsobem je rozšíření Xamarin.Forms ke sdílení stylů. Navíc obsahuje předem připravené komponenty vzhledově podobné prvkům nativního zařízení.

Na rozdíl od jiných frameworků pro vývoj mobilních aplikací, React Native framework implementuje vybrané nativní UI komponenty. To znamená, že aplikace budou vypadat a chovat se jako nativní aplikace. Pokud se React Native používá správně, uživatel nebude schopen rozeznat aplikace od těch nativních.

Nakonec Flutter razí odlišný způsob tvorby UI. Ve Flutteru je všechno widget a sám framework obsahuje sadu připravených unikátních widgetů. Navíc se Flutter nezaměřuje na využívání nativních komponent, ale vytváří si vlastní UI komponenty. Implementuje tak Material Design widgety pro Android a nabízí také Cupertino widgety pro iOS. Dokáže tak vytvářet perfektní vzhled s přesností na pixel.

5.5.7 Ostatní hlediska srovnání frameworků

Při srovnávání je potřeba brát v úvahu i jiné aspekty než výslednou aplikaci, nebo co všechno framework nabízí. Tyto aspekty by se daly nazvat jako „Developer Experience“. Jedná se o to, jak framework zpříjemňuje samotný vývoj aplikace. Mezi konkrétní aspekty lze zařadit komplexita frameworku a složitost programovacího

jazyka, podporované IDE s našeptáváním a kontrolou kódu, funkce urychlující vývoj jako je hot reload nebo stabilita frameworku.

Komplexita frameworku záleží na zkušenostech vývojáře. Jestliže vývojář začíná s vývojem, nejjednodušší na vývoj je Ionic. Využivá webové technologie a není potřeba se zabývat složitým programováním. Složitost React Native frameworku je podobná Ionicu. Vcelku pokud vývojář umí pracovat s JavaScriptem nebo Reactem pro vývoj webových aplikací, je tento framework určitě vhodná volba. Je jednoduchý na naučení a nevyžaduje mnoho úsilí, pouze potřebuje znalost JavaScriptu. Flutter a Xamarin na druhou stranu jsou komplexnější frameworky. Oba využívají komplexní programovací jazyky oproti JavaScriptu. Xamarin si zakládá na C#, kde mnoho věcí je podobné Javě. Flutter využívá Google objektově orientovaný programovací jazyk Dart. Avšak pokud se vývojář naučí základní koncept jazyka, bude schopen rychle vyvíjet mobilní aplikace. Xamarin a Flutter je jednoduší pro naučení, pokud má vývojář zkušenosti s programováním backendových[11] částí aplikací.

Další hledisko je vývojové prostředí. Je nutné, aby prostředí vyhovovalo vývojáři a urychlovalo tak proces vývoje. U frameworků Ionic a React Native nezáleží na vývojovém prostředí. Vystačí si s podporou JavaScriptu, kterou má v dnešní době většina editorů a IDE. Lze tak využívat populární Visual Studio Code, Sublime Text nebo i komplexní placené prostředí jako je WebStorm. Xamarin podporuje vývoj pouze ve svém prostředí Visual Studio. Flutter podporuje aktuálně IDE Android Studio, IntelliJ a přidal rozšíření do editoru Visual Studio Code.

Stabilita je důležitým prvkem opomíjeným při vývoji aplikací. Pokud je potřeba vyvíjet dlouhodobě fungující aplikaci, je potřeba vybrat stabilní framework. Stále rozšiřující se framework, který při aktualizaci neprovádí žádné „Breaking Change“, ani rozbíjející zpětnou kompatibilitu ve stejné verzi. Příkladem může být React Native a jeho stále nulová verze. Vývojáři z Facebooku po tolika letech stále nedospěli k tomu, aby vydali stabilní verzi frameworku. Oproti tomu Flutter

[11] Backend – Logická část aplikace, která není přístupná uživateli napřímo. Jedná se o uchovávání a manipulaci s daty.

nedávno zveřejnil první verzi Flutteru. Xamarin a Ionic jsou na trhu mnohem delší dobu a prokázali svoje schopnosti dlouhodobě vyvíjet mobilní aplikace.

5.5.8 Shrnutí

Pro multiplatformní vývoj mobilních aplikací si všechny čtyři frameworky si získaly důvěru mezi organizacemi, jejichž cílem bylo zkrátit čas a náklady na vývoj aplikací. Všechny frameworky poskytují možnost vyvíjet mobilní aplikace podporující alespoň obě základní platformy.

Z pohledu podpory nativních funkcí splňují všechny frameworky svoji úlohu. U všech lze využívat základní nativní funkcionalitu mobilního zařízení. Co lze rozlišovat u frameworků je jejich výkon při využívání funkcionality zařízení. S nejnižším výkonem přišel Ionic, oproti tomu poskytuje nejjednodušší používání a rychlost vývoje mobilní aplikace s plnou podporou veškerých nativních funkcí zařízení. Ostatní frameworky poskytují výkon přibližující se k nativním aplikacím.

Každý z frameworků má svoje silné a slabé stránky. Umožňuje se tak efektivně rozhodnout pro výběr správného frameworku pro potřeby vývoje aplikace a k dosažení jejich požadovaných funkcí. Pokud je požadavkem vyvíjet komplexní aplikace na grafiku, lze volit mezi Xamarin nebo Flutter frameworkem. Oba mají optimalizované vykreslování a zvládnou komplexní animace. V případě, že chce vývojář vytvořit jednoduchou aplikaci bez velkých požadavků na výkon a bezpečnost, lze zvolit Ionic. Pokud je v plánu vyvíjet dlouhodobě aplikaci s komplexní logikou, je vhodné si vybrat aktivně vyvíjený a komunitou podporovaný framework. Jelikož v případě záseku nebo nevyřešitelné chyby se lze obrátit na komunitu. V tomto případě je vhodné zvolit framework s aktivní komunitou, která dokáže poradit. Podle statistik jsou aktuálně populární dva frameworky – React Native a Flutter. Přičemž Flutter poslední rok získává velkou pozornost od vývojářů, stejně tak Google investuje do tohoto frameworku hodně úsilí.

Pokud někdo chce začít vyvíjet mobilní aplikace, je vhodné zvolit framework podle dosavadních vývojových znalostí. Pokud začínající vyvíjí webové technologie, může volit mezi React Native nebo Ionic u kterých jsou technologie podobné. Jestliže vývojář přichází z Java prostředí, lze zvolit framework Flutter. Programovací jazyk Dart se podobá též jazyku Java. Bez předchozích znalostí programování lze

doporučit React Native nebo Ionic pro svoji jednoduchost. Oproti tomu Flutter a Xamarin již vyžadují základní programovací znalosti.

S celkového hodnocení pro další pokračování byl zvolen framework React Native. Důvodem je požadovaný výkon finální aplikace, který využívá nativní prvky a komunikuje s nimi přes rozhraní most. Dodává aplikacím požadovaný vzhled a podpora komunity je dostačující. Využít se k tomu dá populární webový JavaScriptový framework ReactJS, tudíž získané znalosti se dají aplikovat na webové aplikace. Z čehož vzniká obrovská výhoda, vývojář tak dokáže vyvíjet výkonné a funkční mobilní aplikace, spolu s reaktivními webovými aplikacemi. V případě potřeby React nabízí možnost rozšíření o vývoj na desktopové aplikace.

V další kapitole bude rozveden React Native více do hloubky a budou vysvětleny jednotlivé principy vykreslování a používání jednotlivých funkcionalit. Následně jsou tyto znalosti použitý a demonstrovány na reálné aplikaci.

6 Vývoj mobilních aplikací s React Native

V návaznosti na předešlé rozhodnutí je tato kapitola zaměřená na hlubší představení frameworku React Native. Jelikož je framework založen na jazyku JavaScript, představíme si tento jazyk a jeho výhody či nevýhody. Následuje historie a rozšířený rozbor React Native frameworku spolu se základními prvky.

6.1 JavaScript

JavaScript je jazyk multiplatformní, objektově orientovaný a událostmi řízený skriptovací jazyk. Je považován za jednu ze tří hlavních technologií, které stojí za vývojem webových stránek. JavaScript je založen na standardu ECMAScript, jehož aktuální verze je ECMA-262 (23).

Důležitou vlastností tohoto programovacího jazyka je možnost být kompilovaný či interpretovaný jazyk. Motor JavaScriptu je V8 nebo Rhino – využívá konceptu známého jako kompilace JIT. Pokud je kus JavaScriptového kódu interpretován několikrát, daný kus kódu je zacachován[12]. Opakovaný kód již není znovu kompilovaný, ale využívá již zkompilovaný kód namísto znovu interpretování stejného kusu kódu.

```
1 function arraySum(array) {
2   var sum = 0
3   for (var i = 0; i < array.length; i++) {
4     sum += array[i]
5   }
6   return sum
7 }
8
```

Kód 1: Ukázka využití JIT kompilace v JavaScriptu

To se vztahuje na hlavní cíl kompilace JIT, a to je zvýšit výkonnost jazyka JavaScript. (24)

Jako multi-paradigmatický jazyk, podporuje JavaScript několik stylů programování a lze je dokonce kombinovat; podporuje jak objektově orientovaný, tak i imperativní a funkcionální programovací styl. Přestože byl navržen pouze jako

[12] Cache – uchovávání dat v paměti k urychlení znovu dostupnosti.

jazyk klientský (v prohlížečích), je v současné době také používán na webových serverech, databázích a v některých jiných webových programech nebo aplikacích. (24)

6.1.1 Výhody a nedostatky JavaScriptu

JavaScript, stejně jako jakýkoliv jiný programovací jazyk, přichází s vlastní řadou vymožeností a nedostatků, které je důležité zmínit.

Mezi výhodu, ale i nevýhodu lze zařadit dynamické přiřazení typu. Jelikož JavaScript není typový jazyk, jsou typy asociovány s hodnotami místo proměnných. Výhodou je jednoduché ladění kódu v prohlížeči. Momentálně prohlížeč Chrome poskytuje řadu nástrojů pro zjednodušení ladění; využívání breakpointů[13], logování výstupu v daný moment, bez zásahu do zdrojového kódu a mnohé další. Jak již bylo zmíněno, výhodou je multi-paradigmatický přístup. JavaScript umožňuje více přístupů, jak daný kód napsat. Dává následně možnost vývojáři využívat jakýkoliv přístup, který považuje za nejvhodnější pro svou specifickou situaci. Ukázka některých přístupů je uveden na **Chyba! Nenalezen zdroj odkazů.** (24)

Mezi nevýhody JavaScriptu patří potencionální hrozby v zabezpečení. JavaScript se spustí okamžitě poté, co uživatel vstoupí na webovou stránku, což otevírá dveře potenciálním škodlivým zneužitím uživatele. Ačkoliv moderní prohlížeče jsou obvykle omezující, pokud jde o tyto exploity[14], stále existují způsoby a případy zneužití uživatele. (24)

[13] Breakpoint – označuje místo, kde je JavaScript pozastaven pro snadnější ladění aplikačního kódu.

[14] Exploit – zneužití programátorské chyby, která způsobí nezamyšlenou činnost softwaru.

```
// Imperativní přístup
const text = "Hello World";
console.log(text)

//Funkcionální přístup
let log = (text) => console.log(text);
text("Hello World")

//Objektový přístup
const log = {
  text: "Hello World",
  logOut() {
    console.log(this.text)
  }
}
log.logOut();
```

Kód 2: Ukázka přístupu v JavaScriptu

Jednou z nevýhod je nekompatibilita prohlížečů. Různé prohlížeče mohou provádět stejný kus JavaScriptu odlišně, což by mohlo způsobit nesrovnalosti, které mohou mít nechtěné následky. Naštěstí se tento problém obvykle řeší využitím transpilátorů.

6.1.2 Transpilátor

Transpilátory – nebo překladače – jsou důležitou součástí ekosystému JavaScript. Překladač je nástroj, který čte zdrojový kód napsaný v jednom programovacím jazyce a vygeneruje ekvivalentní kód v jiném jazyce. (25)

Použití transpilátorů je dvojí. Na jedné straně se používají s jazyky typu TypeScript nebo Dart, které pomáhají implementovat různé specifické koncepty, které v jazyce JavaScript chybí. Druhým účelem je využít funkce z novějších edic ECMAScript, které nemusí nutně podporovat všechny prohlížeče. Každý prohlížeč využívá jiný zdroj zpracování JavaScriptu s různými výkonnostními charakteristikami a různými implementacemi standardu ECMAScriptu. Vývojář je následně nucen psát kód stejným způsobem a nemůže využít nových funkcionalit. Transpilátor odstraňuje tento problém, kde převádí nový zápis JavaScript (podle předem definovaných pravidel) do společného fungujícího zápisu. (25)

V následující sekci je popsán JavaScriptový framework React, který React Native rozšiřuje o vlastní funkcionalitu. (25)

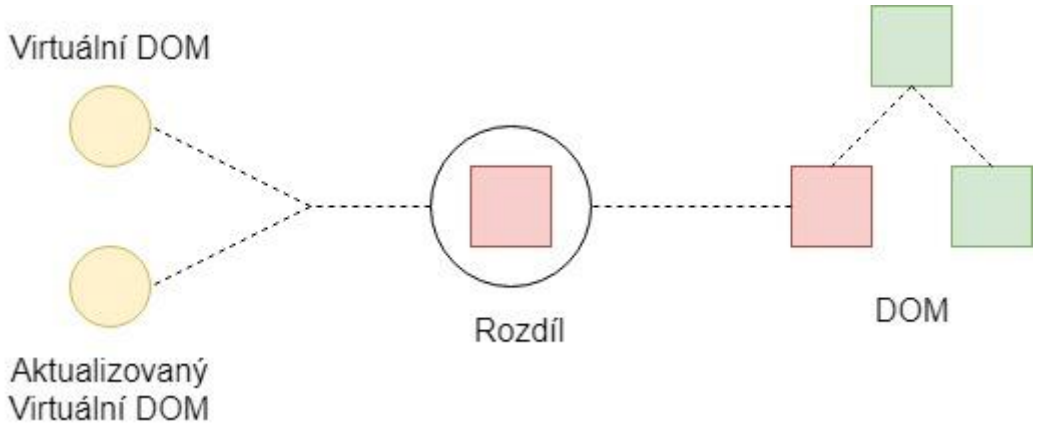
6.2 React

React je JavaScriptová knihovna, používaná pro vyváření uživatelských rozhraní. V tradiční architektuře MVC lze React považovat za View vrstvu. React byl vybudován společnostní Facebook, aby vyřešil jeden problém: vybudování velkých aplikací z dat, které se v průběhu aplikace mění. (26)

K vyřešení problému byl React vybudován jako deklarant. To znamená, že uživatelské rozhraní se definuje jednou a v případě změny stavu aplikace, React reaguje překreslením rozhraní. (26)

React využívá architekturu založenou na komponentách. Komponenty slouží k řešení problému komplexních stavů tím, že jednotlivé části rozdělí na menší součástky. Komponenta v Reactu je podobná funkci v jazyce JavaScript: vždy generuje výstup, když je volána. V podstavě generuje kód HTML, který se nakonec zobrazí v prohlížeči.

Pro dosažení vysokého výkonu React zaznamenává v paměti Virtuální DOM a využívá jednotlivé rozdíly Virtuálního DOMu pro minimalizaci změn ve skutečném DOMu prohlížeče. To umožňuje prohlížečům aktualizovat pouze prvky, které byly změněny, namísto aktualizování celého objektu DOMu. Proces je znázorněn na **Chyba! Nenalezen zdroj odkazů.** (26) (27)



Obrázek 15: Virtuální DOM (26)

Obrázek znázorňuje již zmíněný rozdíl Virtuálního DOMu. Virtuální DOM se uloží do paměti a čeká na změny. V případě změny, se vezme aktualizovaný Virtuální DOM a následně porovná s předešlým. Poté porovná rozdíly jednotlivých Virtuálních DOMů a provede změnu pouze na místech, kde se provedla skutečná

změna. Tento proces minimalizuje práci s DOMem a odlehčuje složitou manipulaci prohlížeči. Proces umožňuje provádět větší operace bez podstatné ztráty výkonu na klientské straně prohlížeče. (28)

6.3 Historie React Native

Facebook v květnu roku 2013 vydal první verzi React, někdy označovanou jako React.js. Jednalo se o průlomový framework k tvorbě JavaScriptových aplikací a sloužil primárně k posílení reaktivity webových stránek na front-endu. V té době nikdo netušil, jakou revoluci tento projekt sebou přinese. (29)

O dva roku později v březnu roku 2015 vychází React Native. Na popularitě nabývat vcelku rychle. Jednalo se o revoluci pro vývoj mobilních aplikací. Když byl React Native vydán, byla podpora pouze pro platformu iOS, avšak podpora pro Android byla přidána téhož roku. Od té doby se framework neustále vyvíjí a přináší stále nové funkce. Facebook následně přijal přístup open-source, kde se snaží zapojit komunitu pro zlepšení frameworku. (29)

Hlavním cílem React Nativu je zjednodušit vývoj mobilních aplikací a sjednotit tak vývoj pro všechny platformy. Vývojáři náhle nemusí mít znalost více programovacích jazyků a mohou vytvářet aplikace pro obě platformy zároveň.

6.4 Jak funguje React Native

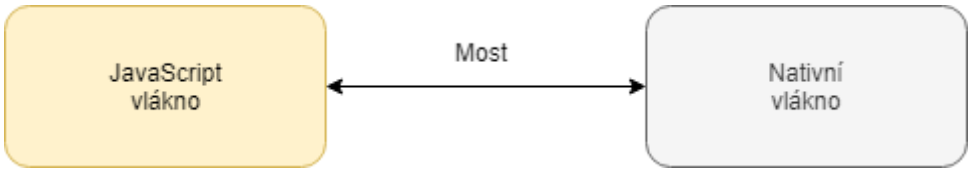
React Native staví na stejných konceptech jako React, ale namísto vykreslování HTML elementů, používá základní stavební prvky nativních platforem. Konečným výsledkem je, že vývojáři mohou vytvářet aplikace za využití nativních prvků platformy, zatímco píší aplikaci v jazyce vyšší úrovně, využívajících konceptů a nápadů z Reactu. (18)

JavaScriptová vlákna jsou zodpovědná za spuštění podnikové logiky aplikace, která je zapsána v jazyce JavaScript. Podproces Javascriptu může posílat / přijímat zprávy do / z hlavního podprocesu pro zobrazení pohledů a reagovat na události.

Hlavní podproces je zodpovědný za vykreslení uživatelského rozhraní na základě příkazů přijatých z podprocesu JavaScriptu. Dále zpracovává také přirozené funkce, jako je vytváření síťových požadavků, reaguje na dotykové události, přistupuje k nativním prvkům mobilního zařízení a další. Hlavní podproces může

v reakci na určité události posílat zprávy zpět na JavaScriptové podprocesy. Tyto zprávy jsou odesílány přes interní komponentu zvanou most. (18) (29)

React Native funguje na základě vykreslování uživatelského rozhraní na hlavním podprocesu s příkazy odesílanými z kódu JavaScriptu. Jako takové musí existovat nějaké prostředí pro komunikaci mezi vlákny. To je místo, kde vstupuje React Native Bridge, neboli most. most je poslední bariéra mezi JavaScriptem a Nativním procesem. Jakékoli údaje, které je třeba předat mezi nativním kódem a jazykem JavaScript, budou odeslány právě přes Most. (18)



Obrázek 16: Schéma vláknového systému implementující React Native

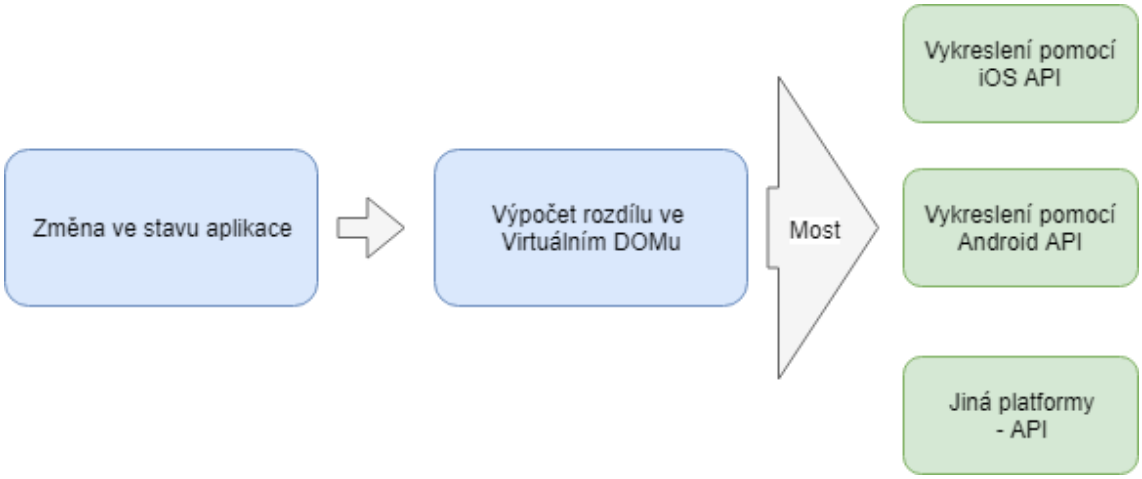
Komunikace přes most jsou obvykle asynchronní a dávkové, aby se omezilo volání na minimum. Most by mohl být považován za hlavní překážku aplikace React Native a doporučuje se minimálně využívat tohoto prvku pro zlepšení přirozeného chování aplikace. (18)

6.5 Základní prvky React Native

Tato kapitola uvádí a vysvětluje základní prvky a vlastnosti React Native, které se využívají při tvorbě aplikace. Slouží jako teoretický základ pro následující kapitolu, která popisuje tvorbu aplikace. Jsou zde demonstrovány ukázky kódu a syntax frameworku, které pomohou vysvětlit funkcionalitu.

6.5.1 Vykreslování

Jak již bylo zmíněno v kapitole 6.2, React Native využívá koncept zvaný Virtuální DOM. Jedná se o techniku optimalizace výkonu. Je to abstraktní vrstva mezi kódem, který popisuje, jak má vypadat aplikace, a skutečným vykreslením těchto prvků. Zde je využit Most pro poskytnutí rozhraní od React Native pro nativní API platformy. Namísto Reactu, kde je vykreslování do prohlížeče DOMu, React Native volí API rozhraní dané platformy. Například na platformě Androidu Most komunikuje s Java API rozhraní pro vykreslování nativních prvků. (18) (28)



Obrázek 17: Vykreslování na různé platformy za využití Virtuální DOMu

Jelikož React Native využívá nativního iOS a Android API, vývoj aplikace vypadá jako u nativní aplikace. Jak je zřejmé z Obrázek 17, pomocí této techniky, lze React Native využívat na kteroukoliv platformu. V případě, že vznikne nová mobilní platforma, je pouze potřeba vytvořit pouze komunikační Most, který s danou platformou komunikuje a předává jednotlivé informace o stavu chování. (18)

6.5.2 Syntax JSX

JSX je specifická syntaxe pro React rozšiřující standardy ECMAScriptu s použitím značkovacího jazyka podobnému XML. JSX je tvořeno z prvků a komponent. Prvky jsou ekvivalent pro klasické HTML elementy (např.) a komponenty odkazují na Reactové komponenty. (30)

Na následujícím Kód 3, můžeme vidět jednoduchý zápis v JavaScriptu bez využití syntaxe JSX. Vytváříme jednoduchý div, kterému nastavujeme text „Hello World“.

```
1 const helloWorld = () => {
2     return React.createElement("div", null, "Hello World");
3 }
```

Kód 3: Zápis bez využití JSX

Stejný příklad je zobrazen na Kód 3, kde je využit právě syntax JSX. Celkově se struktura podobá značkovacímu jazyku HTML, který zjednodušuje psaní pohledu za využití React technologie.

```
const helloWorld = () => {
  return <div>Hello World</div>
}
```

Kód 4: Zápis s využitím JSX

Na první pohled je vidět, že rozdíl přístupů není až tak velký. Nejvíce se však projeví při tvorbě komplexní logiky, která obsahuje více prvků a stylování. Pak se pomocí JSX syntaxe stává kód mnohem čitelnější, avšak na úkor plného využití jazyka JavaScript. (30)

6.5.3 React Native Komponenta

Jako React pro webové prohlížeče používá React Native architekturu založenou na komponentách a celý kód je tvořen uvnitř komponent. Komponenty jsou opakovaně použitelné objekty, které se používají k popisu nativních komponent, které se zobrazují. React Native komponenta má vždy „*render*“ metodu, která vykresluje vzhled. Dále obsahuje některé vlastnosti a vlastní stav chování. (31) (29)

Komponenty se mohou dělit na multiplatformní komponenty, nebo komponentu specifickou pro danou platformu. Například aplikace pro Android a iOS mohou využívat společně komponentu <View>, která se zobrazí jako UIView pro iOS a View pro Android. Avšak některé komponenty jsou dostupné pouze pro specifickou platformu, například <DatePickerIOS> zobrazí standardní výběr datumu pro iOS, zatímco na Androidu existuje <DatePickerAndroid> k dosažení nativní funkcionality výběru datumu. (31)

Vytvoření komponenty je potřeba proto, aby rozšiřovala třídy React.Component a měla metodu „render()“, která vrací JSX syntax. Nejjednodušší příklad komponenty poskytuje **Chyba! Nenalezen zdroj odkazů.**5.

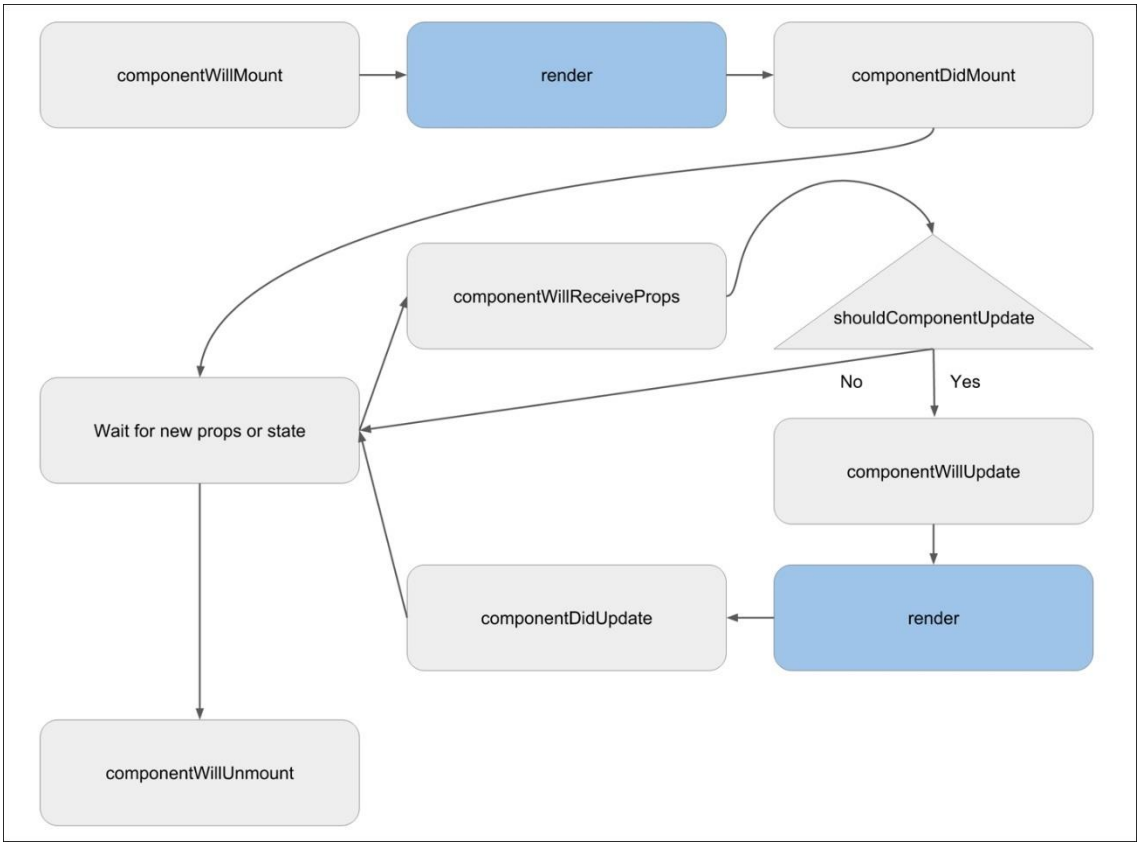
```
import { Component } from 'react'

class HelloWorldComponent extends Component {
    render() {
        <View>Hello World</View>
    }
}
```

Kód 5: React Native komponenta

React Native má různé komponenty podobné základním komponentám v jazyce HTML, které se používají při vývoji s Reactem pro webovou prezentaci nebo čistě v HTML. Pro základní a nejvíce používané elementy má React Native svoje protiklady. Například pro *div* React Native využívá klasickou *View* komponentu. (31)

6.5.4 Životní cyklus komponenty

Každá komponenta v React Native prochází určitými cykly ještě před samotným vykreslením, v průběhu vykreslení i potom, co byla odebrána. Na zmíněné události můžeme v React komponentě reagovat. Tyto kroky se v Reactu



Obrázek 18: Životní cyklus komponenty (33)

nazývají metody životního cyklu komponenty. Metody se nemusejí vynucovat, jsou automatiky volány u každé komponenty, která rozšiřuje *React.Components*. Tyto metody umožnují ovládat chování komponenty v celém průběhu cyklu, od vykreslení až po ukončení. Příkladem je zavolání asynchronního požadavku na server při samotném spuštění komponenty, volání se provede jednou a může poskytnout data, se kterými následně komponenta pracuje. (32) (33)

Na Obrázek 1823. vidíme diagram postupně jdoucích metod životního cyklu. Začneme metodou *componentWillMount*, u které lze zavolat událost ještě před samotným vykreslením komponenty. Následuje cyklus metody *render* následovanou *componentDidMount*. Jedná se o stav, ve kterém je komponenta vykreslena a připravena reagovat na jakoukoliv změnu. Při změně stavu se vyhodnocují metody *componentWillReceiveProps* a *componentWillUpdate*. Pokud dojde ke změně, komponenty se znovu překreslí podle nového stavu a zavolá se následně metoda „*componentDidUpdate*“. V případě že chceme komponenty odebrat z pohledu, zavolá se automaticky metoda *componentWillUnmount*. (33)

Veškeré zmíněné metody lze volat pouze na plnohodnotné třídové komponenty.

6.5.5 Dynamické chování komponenty

K dosažení reaktivity v komponentě je nutné definovat měnící se stavy, určující chování komponenty. U komponenty v Reactu tohoto lze docílit dvěma způsoby. Jedná se o takzvané „props“ a „state“. Každá má svůj specifický význam a způsob využití.

Využíváním props lze docílit větší znuvupoužitelnosti jednotlivých komponent. Mohou být použity referencí *„this.props“* uvnitř *„render“* metody dané komponenty a poté přijmutím při použití komponenty. Zjednodušeně se jedná o předávání stavu nebo dat do nižší úrovně komponenty. (34)

Props by se měly považovat za neměnné a nikdy by se neměly modifikovat přímo uvnitř komponenty. Následující **Chyba! Nenalezen zdroj odkazů.** ukazuje d emonstraci základního použití props v React Native komponentě. Jedná se o dvě základní komponenty. První je zobrazení výpisu lidí, které předává hodnotu *„name“* jako props do komponenty *People*. Druhá komponenta pouze zobrazí text předaný

z *prosp*. Pomocí props je možné definovat obecnou komponentu, která se chová stejně, jen s jinými daty. (34)

```
import { Component } from 'react'

class PeopleList extends Component {
    render() {
        <View>
          <People name="Jack" />
          <People name="John" />
        </View>
    }
}

class People extends Component {
    render() {
        return(
            <Text>{ props.name }</Text>
        )
    }
}
```

Kód 6: Demonstrace využití props

Zatímco *props* lze použít k zobrazení staticky neměnných dat, *state* je datový typ v Reactu, který se používá pro data, která se mění. Základní stav state je obecně definován v konstruktoru komponenty a dále lze přistupovat a měnit daný stav kdykoliv za využití funkce zvané setState. (35)

State je dobrý způsob ukládání jakéhokoliv uživatelského vstupu a může být použit jako možnost pro sledování jakýchkoliv asynchronních požadavků nebo událostí.

```
import { Component } from 'react'

class Clock extends Component {
    constructor() {
      this.state = {
        date: new Date()
      };
    }
    render() {
        <View>
          <Text>{ this.state.date.toLocaleTimeString() }</Text>
        </View>
    }
}
```

Kód 7: Demonstrace využití state

Představili jsme si základní lokální stavy komponenty. Existuje ještě jiný druh stavů, a tím je stav globální, ke kterému může přistoupit kterákoliv komponenta. Jedná se o abstrakci stavů do stejného uložiště, která se využívá zejména u větších aplikací. Příkladem může být Redux. Tento způsob je vysvětlen v sekci 6.5.8.

6.5.6 Interakce

Nesmí se opomenout samotná interakce aplikace s uživatelem. Základním způsobem interakce je u React Native vlastnost *onPress*. Vlastnost reagující na podnět, vyvolávající navěšenou metodu k určení následného chování aplikace.

Jako ukázku můžeme použít jednoduché počítadlo. Při každém kliknutí navýšíme číslo. Kód 8 ukazuje přímé použití interakce. (36)

```
1 import { Component } from 'react';
2 import {View, Button, Text} from 'react-native';
3
4 class Counter extends Component {
5     constructor(props) {
6       super(props);
7       this.state = {
8         count: 0
9       };
10    }
11
12    onPress = () => {
13        this.setState({
14          count: this.state.count++
15        })
16    }
17    render() {
18      return (
19        <View>
20            <Button onPress={this.onPress} title="Počitadlo"></Button>
21            <Text>{ this.state.count }</Text>
22        </View>
23      )
24    }
25 }
```

Kód 8: Demonstrace interakce v React Native

Dalším způsobem interakce je využití gesta na obrazovce. Jako gesto si lze představit potáhnutím prstem nad určitou položkou daným směrem. Systém reagující na gesta řídí životnost gesta v aplikaci. Dotek může projít několika fázemi,

jelikož aplikace určuje, jaký záměr dané gesto má. Aplikace potřebuje detekovat, jestli gesto byl tažný pohyb, dotek nebo reaguje na více gest najednou. K tomu slouží tento systém, který poskytuje komponentám možnost reagovat na tato gesta. (36)

Systém má vlastní metody, které dokáží detekovat začátek interakce, pohyb samotné interakce a samozřejmě umožňuje vývojáři reagovat vlastní událostí.

6.5.7 Stylování prvků

Součástí React Native je zjednodušená implementace CSS pro stylování objektu. Styly jsou deklarovány uvnitř JSX souboru, což umožnuje psát modulární[15] styly pro modulární komponentu, namísto globálních stylů v CSS.

Styly lze psát třemi různými způsoby. Prvním je *inline*, k danému elementu. Další možností je vytváření JavaScriptových objektů a následně reference elementu na daný objekt. Poslední možností je metoda *Stylesheet.create*. Doporučovaný způsob je stylování pomocí *Stylesheet.create*, která činí každou deklaraci stylu nezměnitelnou a zaručuje, že styly budou načítány pouze jednou během životního cyklu aplikace. (37)

Na kódu je ukázáno stylování v React Native.

```
1 import { Component } from 'react';
2 import {View, Text, StyleSheet} from 'react-native';
3
4 cosnt styles = StyleSheet.create({
5     red: {
6       color: 'red',
7     },
8 });
9
10 class TextComponent extends Component {
11     render( ) {
12       return (
13         <View>
14             <Text style={styles.red}>Červený text</Text>
15         </View>
16       )
17     }
18 }
```

Kód 9: Demonstrace použití stylů v React Native

[15] Modularita – přístup rozdělení systémů na menší samostatné části, které jsou využívány v různých částech systému.

V objektu *styles* je definovaný objekt *red* obarvující veškerý text červeně. Pro přiřazení stylu pro element stačí referovat přes atribut style a odkazovat na daný objekt v objektu *styles*. (37)

Mezi populární způsoby stylování u JavaScript aplikací patří používání *Styled-components*. Jedná se o zlehčující způsob stylování, kde css je psáno přímo s referencí, na který prvek mají vlastnosti přijít. Následně se modifikuje vzhled pomocí atributů na daném elementu. (37)

6.5.8 Redux

V předchozí kapitole jsme se zmínili o globálním uchovávání stavu aplikace, ke kterému může kterákoliv komponenta nezávisle přistupovat. K tomu slouží právě Redux, předvídatelný kontejner k uchovávání stavů pro JavaScriptové aplikace. Redux se řídí třemi hlavními principy.

Prvním principem je uchovávání všech stavů v jednom úložišti ve stromovém objektu. Tím docílí jednoduché manipulace s daty. Objekt může být tak snadno uchován na klientské straně bez žádného velkého úsilí, a tím se docílí lepšího uživatelského efektu. (38)

Druhým principem je pouze možnost číst uchovávané stavy. Znamená to, že jednotlivé stavy pro komponenty jsou určeny pouze pro čtení a nikoli k jejich modifikaci uvnitř komponenty. Pro modifikaci slouží v Reduxu *action*, které vyvolávají událost, při které se nastaví nová hodnota pro vybraný stav. To zaručí, že veškerá manipulace s daty zůstává na jednom místě. (39)

Posledním principem je provádění změn pomocí čistých funkcí neboli tzv. reducerů. Reducer je čistá funkce, která přijímá předchozí stav a akci, a vrátí modifikovaný stav. Je důležité si uvědomit, že návratová hodnota je nový objekt, nikoliv mutace předešlého. Reducer přijme objekt, modifikuje stav a následně vrací nový požadovaný stav. (39)

6.5.9 Testování aplikace

Občas pracujeme s logikou, která je komplexní a vyžaduje bezchybnou implementaci. V případě, že by nastala chyba, může to mít velký dopad pro aplikaci

a její správné fungování. Tento problém lze vyřešit pomocí psaní testů. U React aplikací lze provádět tyto druhy testů.

Jedny ze základních testů jsou jednotkové testy. Jednotkové testování je druh automatického testování zaměřeného na testování funkcionality částí systémů izolovaně. Testují se samotné funkce, React komponenty nebo celé třídy. Cílem jednotkových testů není samotná logika aplikace, ale pouze funkcionalita testovaného subjektu. (40)

```
1 const testArray = [1, 2];
2 const expectedArray = [2, 4];
3
4 test('test increment array', () => {
5     expect(multipleArray(testArray, 2)).toEqual(expectedArray);
6 })
```

Kód 10: Ukázka Jednotkového testu v Jest

Na testování použijeme JavaScriptový testovací nástroj Jest. Jest spravuje Facebook, tudíž kompatibilita s React a React Native je zaručená. V dokumentaci uvádí manuál pro testování přímo na React Native. Na obrázku však lze vidět jednoduchý test, porovnávající dvě pole. Funkce „multipleArray“ vynásobí pole dvěma, a následně se porovnává výsledek s očekávaným polem „expectedArray“. Pokud obě pole jsou stejná, test úspěšně prošel. (40)

Dalším stupněm testování jsou integrační testy. V integračních testech jsou jednotlivé části aplikace spojeny v jeden celek, a testuje se kompatibilita více prvků spolupracujících k dosažení požadovaného výsledku. Účelem integračních testů je zjištění případných chyb v rozhraní a interakcích mezi jednotlivými moduly a komponenty. Například každá funkce sama o sobě funguje správně, ale jako celek může vytvářet nechtěnou funkcionalitu. (40)

Posledním stupněm testování je testování pomocí snímků. Tímto způsobem lze testovat uživatelské rozhraní. Test probíhá uložením prvotní verze snímku, která je schválena. Následně probíhají testy, kde se porovnávají snímky. V případě že snímek selže, nemusíme činit obecný závěr, že test neprošel. Mohlo se jednat o úmyslné změnění vzhledu a mělo dojít k nesouladu. V tomto případě je správným postupem vytvořit a uchovat nový snímek a odstranit předchozí zastaralou verzi.

Veškeré testování je vhodné testovat automatizovaně. K tomu mohou sloužit automatizované nástroje, nebo verzovací systémy, které spouští tyto testy automaticky jako reakce na událost (událostí může být změna kódu). (40)

6.5.10 Ladění a hledání chyb

Poslední částí této kapitoly je ladění programu. Opomíjenou, ale důležitou součástí každého vývoje je potřeba umět ladit a případně vyhledat zdroj chyby v aplikaci.

Všechny chyby a varování v aplikaci se zobrazují uvnitř běžící aplikace. Jedná se o efektivní způsob, jak sdělit chybu na jednom místě.

Pokud se v naší aplikaci objeví chyba nebo nějaká nestandardní akce, framework ji vrátí do terminálu i obrazovky s vysvětlením o dané chybě. Chyba se zobrazí přes celou obrazovku s chybovou zprávou a stopou. React Native dokonce poskytuje způsob, jak se rychle dostat k problematickému kódu kliknutím na konkrétní řádek. (41)

Pro přístup ke skriptům JavaScriptu a zobrazování logů a ladění aplikace je nutné připojit běžící aplikaci do webového prohlížeče Chrome podporujícího Nástroje pro Vývojáře. V aplikaci React Native se musí povolit vzdálené ladění JavaScriptu a poté připojit k běžící aplikaci z webového prohlížeče. Tím dostáváme možnost ladit aplikaci stejným způsobem jako webovou stránku. Můžeme vkládat breakpointy, sledovat zpracování síťových požadavků a jejich odpovědi. (41)

6.6 Shrnutí

Tato kapitola pojednává o technologii React Native a její funkcionalitě. Vysvětleny jsou principy a z čeho je React Native složen, a jak funguje. Následující kapitola pojednává o praktickém použití technologie na reálném příkladu, kde se využívají jednotlivé prvky. Nejprve se zaměříme na specifikaci zadání a účel aplikace.

7 Specifikace zadání

V této části je specifikováno zadání, které se následně zpracuje jako praktický příklad. Určí se jednotlivé požadavky na aplikace a co všechno má splňovat. Následně se seznámíme s prostředím, ve kterém probíhá samotný vývoj.

7.1 Aplikační požadavky

Před samotnou implementací aplikace, je potřeba si definovat požadavky na aplikaci. Cílem aplikace je prakticky si vyzkoušet funkcionalitu React Native.

Aplikace jako samotná by měla běžet na více operačních systémech. To je hlavní myšlenka multiplatformního vývoje mobilní aplikace. Proto jako první požadavek je běh jak na iOS platformě, tak i na Android.

Dalším kritériem aplikace je využívání vestavěné funkcionality mobilního telefonu. Jde o otestování propojení React Native spolu s vestavěnou funkcionalitou mobilního zařízení. V případě, že bychom nevyužívali tento požadavek, celou aplikaci bychom mohli napsat jako webovou aplikaci zobrazovanou na mobilech.

Nedílnou součástí dnešní doby je dotazování se na připravené rozhraní pro získávání dat za běhu aplikace. Dalším požadavkem je tedy využití dotazování na API a následné zobrazení dat. Vyzkouší se ladění dotazování a zobrazení toku dat přes síť a reakce na možné situace. Jelikož bude probíhat velké množství dotazů, je nutné připravit optimalizaci. Výsledná data se uloží do aplikace k zamezení zbytečných duplikačních požadavků na síť a uživatel tak nemusí zbytečně posílat další požadavky. Následně aplikace bude schopna fungovat i bez připojení k internetu.

Dalším požadavkem je využívání externích knihoven rozšiřujících buď React Native funkcionalitu nebo danou logiku, kterou již někdo vymyslel a není potřeba to vymýšlet znovu. Zpravidla se jedná o neoficiální balíčky vyvíjené komunitou.

Aplikace by měla být rozšiřovatelná a lehce udržovatelná. Proto je potřeba psát aplikaci optimálně k těmto požadavkům.

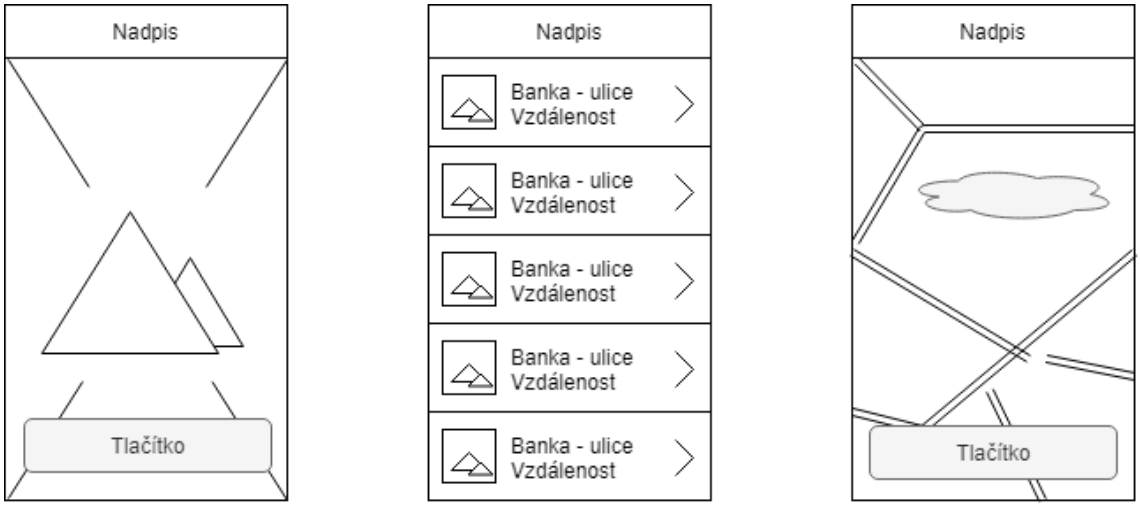
7.2 Případová studie

Jako případová studie pro vývoj mobilní aplikace psané v React Native, je vytvořena aplikace pro zobrazení nejbližších bankomatů podle aktuální pozice zařízení. Pracovní název je Banko, zkrácenina slova Bankomat.

Důvodem vzniku této aplikace je zjednodušit hledání bankomatů v cizím prostředí, kde uživatel nezná okolí. Uživateli má aplikace usnadnit nalezení nejbližšího bankomatu podle aktuální polohy.

Aplikace splňuje funkce jako získávání aktuální polohy zařízení, při kterém využívá vestavěné funkce jako GPS. Bankomaty a jejich lokality se získají ze zdrojů třetích stran pomocí dotazování do jejich připraveného rozhraní. Získaná data se budou uchovávat v zařízení, kde proběhne po určitém časovém intervalu aktualizace. To umožní optimalizaci v rychlosti a zmenší se počet dotazů, které sníží tok dat. Dalším požadavkem je implementace knihovny z třetí strany, kterou řeší zobrazení bankomatů na mapě, kde je ukázáno, jak lze cílit funkcionalitu pouze na konkrétní platformu. Aplikace řeší manipulaci s daty ve formě způsobu zobrazení a jejich seřazování podle vstupních údajů.

Aplikace je složená ze tří základních obrazovek. První obrazovka je domovská stránka, která bude sloužit jako rozcestník. Momentálně bude obsahovat pouze proklik do výpisu, ale jedná se o horizontální posouvací karty, kde mohou časem přibývat nové funkcionality.



Obrázek 19: Návrh obrazovek aplikace

Druhou obrazovkou je samotný výpis aplikace. Skládá se z dosažených dat seřazených od nejbližšího bankomatu. Každý záznam ve výpisu zobrazuje aktuální vzdálenost, pobočku bankomatu a ulici. Při prokliknutí záznamu se dostaneme do detailu bankomatu. Detail se skládá ze zobrazení vybraného bankomatu na mapě k lepšímu vyhledání bankomatu.

7.3 Příprava prostředí

Pro samotný vývoj aplikace je použito několik pomocných nástrojů. Některé z uvedených jsou potřebné pro samotný vývoj, jiné ho usnadňují a zpříjemňují.

Mezi hlavní nástroje, bez kterých nelze vyvíjet multiplatformní aplikace, patří vývojová prostředí neboli IDE. Napomáhají s přípravou samotného projektu a umožňují stáhnout potřebné balíčky na vývoj automaticky. K vývoji pro platformu iOS se používá prostředí XCode, jenž připraví veškeré potřebné knihovny na platformu a obsahuje samotný emulátor iPhonu pro testování aplikace. Android využívá svoje prostředí Android Studio, které je specificky navržené na vývoj Android mobilních aplikací. Dopomáhá stáhnout veškeré balíčky a umožnuje stáhnout různé verze operačního systému Android, stejně tak i s emulátorem konkrétních mobilních telefonů. Avšak pro samotnou práci je využit editor Visual Code Studio. Hlavním důvodem je samotná rychlost editoru oproti vývojovému prostředí a možnost osobního přizpůsobení editoru. Dalším důvodem je psaní multiplatformní aplikace v jazyce JavaScript, proto není potřeba využívat specifické IDE pro danou platformu.

Aplikace je tvořena na Macbook Pro 2016 s operačním systémem macOS High Sierra. Jelikož Apple je uzavřená komunita a nedovoluje vývoj aplikací pro platformu na jiných operačních systémech, lze na Macbooku testovat výslednou aplikaci v emulátoru.

Jako verzovací systém je využit GIT a kód tak lze snadno uchovávat na Githubu, který slouží jako veřejné úložiště zdrojového kódu.

Na vývoj je nutné mít v neposlední řadě připraveno JavaScriptové prostředí. Jedná se o Node.js, který umožňuje snadný vývoj JavaScriptových aplikací. Obzvlášť je využit správce JavaScriptových balíčků. Umožňuje jednoduše přidávat jednotlivé knihovny z repositáře a uchovávat jejich verze pro snadné spuštění projektu.

Aplikace je zadaná se všemi specifikacemi, které musí splňovat pro správnou demonstraci vývoje multiplatformní aplikace za využití React Native. Prostředí pro vývoj je definováno. Následující kapitola pojednává o samotné implementaci aplikace.

8 Vývoj aplikace Banko

Tato kapitola popisuje praktický vývoj mobilní aplikace zadané z případové studie. Čtenáři je ukázán způsob vývoje multiplatformní mobilní aplikace v technologii React Native.

Popis jednotlivých kapitol na sebe navazuje, a nelze přeskakovat jednotlivé části, aniž by aplikace mohla fungovat. Pro kompletní zdrojové kódy je k dispozici elektronická. Projekt obsahuje soubor *Readme.md*, ve kterém popisuje postup pro zprovoznění a spuštění projektu.

8.1 Příprava projektu

Než se začne s vývojem aplikace, je potřeba připravit vývojové prostředí. Prvotní úkol je zprovoznit základní „Hello World“[16] aplikaci a spustit ji v obou emulátorech (iOS a Android). Následně se vysvětlí struktura React Native aplikace a v jaké části se vyvíjí aplikace.

K chodu aplikace je potřeba nainstalovat Node.js, který lze stáhnout z oficiálního webu. Instalace probíhá intuitivně pomocí grafického rozhraní. Po dokončení instalace je důležité si zkontrolovat, zdali je Node.js dostupný z terminálu a můžeme využívat požadovaného rozhraní a správce JavaScriptových balíčku. Pro zjištění do terminálu spustíme následující příkazy. Pokud se Node.js nainstaloval správně, příkaz by měl vrátit aktuální nainstalovanou verzi.

```
➜  ~ node -v
v11.2.0
➜  ~ npm -v
6.7.0
```

Kód 11: Kontrola verze Node.js a npm

Dalšími nástroji jsou vývojářské prostředí podporující emulátory mobilních zařízení. Pro iPhone je potřeba nainstalovat XCode prostředí. Tento krok lze provést pouze na zařízení od společnosti Apple. XCode nainstalujeme jednoduše pomocí Apple Storu, kde se operační systém postará o stažení a instalaci. Po dokončení si otestujeme spuštění emulátoru pro iPhone. Stejný proces provedeme pro Android

[16] Hello World – malý program, pro první spuštění generující text na obrazovce.

Studio, kde na oficiálních stránkách stáhneme IDE a provedeme instalaci. U Android Studio je zprovoznění komplikovanější, proto je dobré postupovat podle návodu zveřejněného na oficiálních stránkách. Po dokončení těchto kroků jsou připravené emulátory na obě platformy. Nainstalujeme si poslední editor pro psaní aplikace. Jde pouze o osobní preference. Předchozí nástroje splní stejnou funkčnost.

Základní nástroje jsou připravené a je čas vytvořit samotný projekt. V příkazovém řádku se přesuneme do složky, kde chceme mít projekt uložený a pomocí npm stáhneme příkazové rozhraní pro React Native.

```
➜ projects npm install expo-cli -g
```

Kód 12: Instalace rozhraní Expo

Expo je nejjednodušší způsob, jak začít vytvářet nové aplikace pro React Native. Vygeneruje základní strukturu pro projekt bez jakékoliv další instalace nebo konfigurace. Po dokončení instalace Expa stačí zadat příkaz pro vytvoření projektu spolu s názvem složky, pod kterým je aplikace dostupná.

```
➜ projects expo init Banko
```

Kód 13: Vytvoření základní struktury aplikace

Jak je zmíněno, příkaz vytvoří složku pojmenovanou Banko spolu se všemi závislostmi. Pro spuštění projektu se stačí v příkazovém řádku přesunout do složky a spustit projekt pomocí vygenerovaného skriptu.

```
➜ projects cd Banko
➜ Banko git:(master) ✗ npm start
```

Kód 14: Příkazy pro spuštění aplikace

Příkaz spustí vývojový server, na kterém poběží aplikace. Dalším krokem je propojení aplikace pro oba emulátory na zobrazení „Hello World“ aplikace.

Aplikace je připravena na vývoj a běží v režimu sledování změn, kde při změně kódu se automaticky spustí sestavení aplikace a výsledný kód je vložen do běžící aplikace a dojde k překreslení zobrazované aplikace bez nutnosti vypnutí nebo manuální aktualizace mobilního zařízení. Umožnuje tak rychlejší a příjemnější vývoj aplikací.

8.2 Struktura aplikace

Na následujícím obrázku lze vidět vygenerovanou strukturu čistě aplikace.

```
App.js
__tests__
android
app.json
index.js
ios
node_modules
package-lock.json
package.json
```

Obrázek 20: Struktura React Native aplikace

Struktura aplikace je složena z konfiguračních a aplikačních souborů. Mezi konfigurační soubory lze zařadit *expo*, *babelrc*, *eslintrc*, *gitignore*, *watchmanconfig*. Dané soubory nastavují styl kompilace JavaScriptu, způsob syntaxe a jiné možné konfigurace týkající se celého projektu. Mezi důležité soubory konfigurační patří *package.json*. Uchovává v sobě seznam knihoven, na kterých je aplikace závislá pro svůj běh. Navíc obsahuje definované skripty, pro spuštění například aplikace, nebo spouštění testů, aniž bychom si museli pamatovat sekvenci příkazů. Po vygenerování se spustí příkaz pro stažení všech závislých knihoven. Knihovny jsou uchovávány ve složce *node_modules*. Po dokončení veškerého stahování se automaticky vygeneruje *package-lock.json* obsahující verze jednotlivých knihoven pro zpětnou kompatibilitu.

Nejdůležitějším souborem je však App.js. Tento soubor lze označit jako výchozí bod aplikace. Slouží jako spouštěč aplikace.

V tuto chvíli je projekt připraven a vysvětlena základní struktura aplikace. Následující část se již zabývá konkrétní implementací aplikace.

8.3 Vývoj aplikace

Tato sekce pojednává o samotném vývoji aplikace. Vývoj je psán postupně, jak aplikace vzniká od úvodní obrazovky, přes výpis, až do detailu s mapou. Není zde popsaná veškerá funckionalita a stylování. V případě zájmu je možné nahlédnout do zdrojového kódu.

8.3.1 Úvodní obrazovka a Navigace aplikace

Jako první je sestavena úvodní stránka. Začneme smazáním veškerého vygenerovaného obsahu v souboru App.js. Úvodní obrazovka je definovaná jako rozcestník pro vybrání požadované funkce. Rozcestník má být horizontální posouvání bloků karet, kde každá karta obsahuje specifickou ikonu, název funkce a její popis.

Ve struktuře aplikace vytvoříme složku *src*, v které je následně tvořen veškerý obsah pro aplikaci. Prvotně si vytvoříme konfigurační soubor uchovávající statická data pro aplikaci. Vytvoříme pole karet uchovávané v konstantě *homepageSlides*, kterou následně exportujeme pro možné použití z jiného souboru.

```
1 export const homepageSlider = [
2   {
3     title: 'Výpis bankomatů',
4     image: require('./../assets/homepage/atm.jpg'),
5     route: 'atmList',
6   }
7 ]
```

Kód 15: Reference na data pro úvodní stránku

Vytvoříme si nový adresář *screens* obsahující jednotlivé obrazovky. Jako první vytvoříme *homepage.js* pro úvodní stránku. Na úvodní stránce budeme chtít využívat externí knihovnu vytvářející *Carousel* efekt. Pro tuto specifikaci však vytvoříme samostatnou komponentu. Komponenta přijímá vykreslující prvky jako props, proto ji lze použít i v jiných částech aplikace. Do komponenty importujeme všechny potřebné knihovny a elementy nám poskytující knihovny „react-native“. Vytvořená třída MyCarousel obsahuje samotnou render metodu vykreslující svůj obsah. Dále je zde druhá metoda pro samotný prvek posílající dovnitř externí knihovny „react-native-snap-carousel“. Nesmíme zapomenout i na potřebné styly k zobrazení požadovaného efektu. Kompletní komponenta vypadá následovně.

```
import {Component} from 'react';
import {Text, View, ImageBackground, StyleSheet, TouchableOpacity} from 'react-native';
import Carousel from 'react-native-snap-carousel';

class CustomCarousel extends Component {
    _renderItem({item, index}) {
        return (
            <ImageBackground
              style={styles.carouselItem}
              source={item.sourceImage}
            >
              <View style={buttonContainer}>
              <TouchableOpacity
                style={[bubble, button]}
                onPress={() => this.props.navigation.navigate(item.route)}
              >
                <Text style={buttonText}>{item.title}</Text>
              </TouchableOpacity>
              </View>
            </ImageBackground>
        );
    }

    render () {
        const { entries, sliderWidth, itemWidth } = this.props;
        return (
            <Carousel
              ref={(c) => { this._carousel = c; }}
              data={entries}
              renderItem={this._renderItem.bind(this)}
              sliderWidth={sliderWidth}
              itemWidth={itemWidth}
            />
        );
    }
}
```

Kód 17: Vytvoření komponenty pro Carousel v souboru MyCarousel.js

MyCarousel importujeme do úvodní obrazovky spolu s definovanými daty. Statická data předáme komponentě *MyCarousel* a vznikne nám požadovaný efekt.

```
class HomeScreen extends React.Component {
  render() {
    return (
      <View style={{ flex: 1, alignItems: 'center', justifyContent: 'center' }}>
        <Carousel entries={homepageSlides} sliderWidth={globalWidth}
          itemWidth={globalWidth} navigation={this.props.navigation} />
      </View>
    );
  }
}
```

Kód 16: Zobrazení úvodní stránky

Úvodní stránka je vytvořena. Pro přechod do výpisu je potřeba vytvořit navigaci pro aplikace k možnému přechodu mezi jednotlivými stránkami.

8.3.2 Navigace aplikace

Pro nadefinování navigace aplikace je nejlepší využít již napsanou externí knihovnu, tvořící efekt možný pro mobilní aplikace. Na mobilních aplikacích lze využívat několik druhů navigací. Mezi nejvíce používané je postranní menu neboli Hamburger Menu. K často používaným patří i panelové karty pro přechod mezi obrazovkami. V naší aplikaci využijeme překrývání obrazovek nad sebou. Jedná se o populární způsob, který využívá například aplikace Instagram. Při prokliku nová stránka překryje stávající a zobrazí návratové tlačítko.

```
import { createStackNavigator } from 'react-navigation';

const AppNavigation = createStackNavigator(
  {
    home: {
      screen: Home,
      navigationOptions: () => ({
        title: screenTitles.home,
        headerLeft: null
      })
    },
    atmDetail: {
      screen: atmDetail,
      navigationOptions: () => ({
        title: screenTitles.atmDetail
      })
    },
    atmList: {
      screen: atmList,
      navigationOptions: () => ({
        title: screenTitles.atmList
      })
    }
  },
)
```

Kód 18: Definice navigace aplikace

Na **Chyba! Nenalezen zdroj odkazů.**je znázorněna demonstrace vytvoření n avigace pro aplikace Banko. Každá obrazovka má svoji identifikaci, podle které lze ke stránce přistupovat. Navigaci je potřeba aktivovat v kořenovém souboru *App.js*, jelikož určuje jaká obrazovka je v danou situaci aktivní.

```
1 import {Component} from 'react';
2 import AppNavigation from './src/screens';
3
4 export default class App extends Component {
5     render() {
6         return (
7             <AppNavigation>
8         )
9     }
10 }
```

Kód 19: Registrace a použití navigace pro aplikace

Navigace s úvodní stránkou je připravena. Dalším cílem je vytvořit stránku s výpisem bankomatů.

8.3.3 Výpis Bankomatů

V dané části je požadavek o vytvoření výpisu bankomatů seřazených od nejbližšího bankomatu podle aktuální polohy používaného zařízení. Před implementací si definujeme konkrétní požadavky, jak by daný proces měl fungovat. Vcelku si vytvoříme něco podobného jako je User Story[17] obsahující svoje akceptační kritéria pro splnění požadované funkcionality.

Požadavkem je vytvoření výpisu bankomatů. K tomu je daný seznam akceptačních kritérií.

- Když uživatel prohlíží výpis, má k dispozici aktuální seznam bankomatů získaných z API pobočky
- Každý záznam obsahuje logo pobočky, název, ulici a vzdálenost od zařízení
- Výpis bankomatů je seřazený od nejbližšího k aktuální poloze zařízení
- V případě přemístění zařízení jsou vzdálenosti automaticky přepočítávány
- Při kliknutí na záznam ve výpisu je uživatel přesměrovaný do detailu bankomatu
- Záznamy získané přes API lze uložit do mobilu, aby se při opětovném spuštění a přecházení mezi stránkami se nepouštěly duplicitní dotazy

Požadavky jsou definovány a následuje jejich samotná implementace.

[17] User Storie – způsob zadávání práce s přidanou hodnotou pro uživatele.

Jako data použijeme hlavně bankomaty společnosti Airbank. K získání přístupu je nutné se registrovat jako vývojář v jejich portálu a zaregistrovat samotnou aplikaci. Po dokončení registrace je vygenerován přihlašovací API klíč. Klíč umožňuje získat přístup k rozhraní banky. Z dokumentace výčtu rozhraní využijeme to, které jako návratovou hodnotu obsahuje všechny registrované bankomaty v České republice. Rozhraní neobsahuje žádné doplňující parametry, jenom posílá zpět bankomaty tak, jak jsou uložené v databázi. K dotazu se využije externí knihovna na protokolovou komunikaci *Axios*. Funguje jako klient a obsahuje širokou funkcionalitu. Požadavek je demonstrován následovně. (42)

```
export async function setAirbankAtms() {
    cosnt response = await axios({
        url: 'https://api.airbank.cz/openapi/public/v1/atm/own',
        method: 'get',
        headers: {
            'Content-type': 'application/json;charset=UTF-8',
            apiKey: airbankApiKey
        }
    });
    if (response.status === 200) {
        return response.data.data;
    }
}
```

Kód 20: Dotaz na získání bankomatů

Funkce má prefix *async*, to dává najevo, že funkce vrací *promise*. *Promise* funguje jako „příslib budoucí hodnoty“, kterou metoda jednou vrátí. Následně klíčovým slovem *await* je poručeno JavaScriptu, aby vyčkala, dokud se *promise* nedokončí. (42)

V dalším kroku zpracujeme data a ke každému záznamu přidáme záznam o aktuální vzdálenosti od zařízení. Pro zjištění aktuální polohy zařízení React Native nabízí možnost využít Geolokaci zařízení. Při vytváření aplikace pomocí Expa, je automaticky povoleno využívat Geolokaci na zařízení. Aby aplikace měla informace o aktuální poloze, je nutné získat souhlas od uživatele. Geolokace má vlastní metody *getCurrentPosition* k získání aktuální polohy.

```
getCurrentPosition() {
    navigator.geolocation.getCurrentPosition(
        position => {
            this.setState({
                currentPosition: {
                    lat: position.coords.latitude,
                    lon: position.coords.longitude
                }
            })
        },
        { enableHighAccuracy: true, timeout: 20000, maxiumAge: 1000 }
    )
}
```

Kód 21: Metoda pro získání aktuální polohy

Aktuální poloha je ukládána do *state currentPosition*, kterou využijeme pro spočítání vzdálenosti mezi jednotlivými bankomaty. K výpočtu vzdálenosti využijeme zeměpisné souřadnice. Je potřeba napsat funkci, která spočítá kilometrovou vzdálenost za využití zeměpisné délky a zeměpisné šířky. Použijeme takzvaný „haversine“ vzorec s převodem na kilometry, který vrátí skutečnou vzdálenost mezi body. Není přesná jako Vincentyho formula, ale splní v aplikaci svůj účel. (43)

```
function distanceBetweenPoints(lat1, lon1, lat2, lon2) {
    const R = 6371;
    const dLat = deg2rad(lat2 - lat1)
    const dLon = deg2rad(lon2 - lon1)
    const a = Math.sin(dlat / 2) * Math.sin(dLat / 2) +
        Math.con(deg2rad(lat1)) *
        Math.cos(deg2rad(lat2)) *
        Math.sin(dLon / 2) *
        Math.sin(dLon / 2)
    const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a))
    const d = R * c;
    return d;
}

function deg2rad(deg) {
    return deg * (Math.PI / 180)
}
```

Kód 22: Haversine formula pro výpočet vzdálenosti použitý v JavaScriptu (43)

Aplikace nevyžaduje vysokou přesnost výpočtu, vystačí si s orientační vzdáleností.

Momentálně máme data bankomatů obsahující zeměpisné souřadnice, používaného zařízení a pomocnou funkci na výpočet vzdálenosti. Při iteraci jednotlivých prvků vypočítáme aktuální vzdálenost mezi body a uložíme do nového pole, ve kterém jsou jednotlivé prvky uložený jako objekty s klíčem vzdálenosti.

```
setCurrentAtms() {
    items.reduce((atms, item) => {
      const distance = distanceBetweenPoints(
        this.state.currentPosition.lat,
        this.state.currentPosition.lot,
        item.location.latitude,
        item.location.longitude
      );
      let obj = {};
      obj[distance] = item
      array.push(obj)
      return atms;
    }, []);
}
```

Kód 23: Iterace bankomatu pro výpočet jednotlivých vzdáleností

Momentálně jsou data připravena s aktuální vzdáleností. Dalším úkolem je seřadit je podle nejmenší vzdálenosti. Protože se pracuje s velkým počtem záznamů, je potřeba vybrat vhodný řadící algoritmus. V této situaci je zvolen algoritmus *quicksort*. Algoritmus funguje na principu náhodného zvolení prvku (pivot) v poli. Následně se přeskakují prvky tak, aby na jedné straně byly prvky menší než pivot a druhé větší. U rozdělených prvků se provede stejná akce, pokud mají více než jeden prvek. Volba pivotu je velmi důležitá, jelikož určuje výkonnost samotného algoritmu. Jestli pivot je nejnižší prvek, nebo nejvyšší, nedojde k rozdělení polí a počet operací se zvýší. Oficiální algoritmus spočívá v řazení čistých dat, v našem případě je potřeba řadit objekty s klíčem. Implementace *quicksort* algoritmus je následující. (44)

```
export const quickSortAlgorithm = list => {
    if (list.length < 2) return list;
    let pivot = Number(Object.keys(list[0])[0])
    let pivotObject = list[0];
    let left = [];
    let right = [];
    for(let i = 1, total = list.length; i < total; i++) {
        if (Number(Object.keys(list[i])) < pivot) left.push(list[i])
        else right.push(list[i])
    }
    return [...quickSortAlgorithm(left), pivotObject, ...quickSortAlgorithm(right)];
}
```

Kód 24: Zjednodušená implementace quicksort algoritmu (44)

V daný moment jsou vytvořený funkce, které zpracovávají data pro další použití. Abychom zaručili funkcionalitu, je možné napsání testů pro vybrané funkce. V tuto chvíli lze napsat test na výpočet vzdálenosti a *quicksort* algoritmus pro řazení.

Každý test obsahuje vstupní data, popis funkce a očekáváný výsledek. Všechny testy jsou uchovávány ve složce __test__. Každý soubor se skládá z názvu souboru, následně „test“ pro označení souboru jako testovacího a příponou souboru (například *name.test.js*).

```
const pointA = {
    lat: '50.212909',
    lon: '15.824934'
}

const pointB = {
    lat: '50.204150',
    lon: '15.829489'
};
const expectedDistance = 1.03;

test('Distance between 2 points', () => {
    expect(distanceBetweenPoints(pointA, pointB)).toEqual(expectedDistance);
})
```

Kód 25: Test pro výpočet vzdálenosti mezi body

Na **Chyba! Nenalezen zdroj odkazů.** je definovaný jednoduchý test pro v ýpočet vzdálenosti. Máme připravené body a předem víme očekávanou vzdálenost. Vytvoříme test, kde funkce vrátí námi očekávanou vzdálenost. Pokud je výsledek správný, test projde. V případě že test neprojde, či test selže, je potřeba funkce

opravit. Test na výpočet vzdálenosti lze zařadit jako jednotkový test testující jednu konkrétní funkcionalitu.

Data bankomatů jsou připravená a zbývá je vykreslit do výpisu. K tomu využijeme nativní komponentu React Native, a to je *FlatList*. React Native nabízí dva hlavní způsoby vykreslování výpisu. *ScrollView* komponenta vykresluje všechny prvky najednou. Což umožňuje jednoduchou implementaci a použití. Naopak při větším počtu dat může nastat problém s výkonností, kdy se budeme snažit vykreslit větší množství dat najednou. Naopak *FlatList* přistupuje k vykreslování dat postupně. Jednotlivé prvky se načítají postupně podle toho, jak se mají zobrazovat na obrazovce. Pokud se prvek dostane ale mimo obrazovku, je odstraněn. Pomocí těchto technik *FlatList* šetří paměť a výpočetní čas a zvyšuje výkonnost celé aplikace. (45)

```
_renderItem = ({item}) => {
    return(
        <ListItem item={item} distance={Object.keys(item)[0]}></ListItem>
    )
}

render() {
    return (
        <FlatList
            data={this.state.items}
            renderItem={this._renderItem}
        >
        </FlatList>
    )
}
```

Kód 26: Zápis pro výpis bankomatů s použitím FlatList

Při použití *FlatList* posíláme do komponenty props data jako prvky, které chceme vypsat a následně *renderItem* props sloužící jako šablona. Šablona přijímá jako parametr jeden prvek z pole. Pomocí této techniky můžeme definovat jednoduše vzhled vykresleného prvku. V našem případě použijeme komponentu definovanou ve vlastním souboru. Komponenta využívá prvek z knihovny react-native-element připravený pro výpis.

Aktuálně je připravený výpis bankomatů, získaný z externího úložiště a seřazený podle nejbližší polohy. V dalším kroku je potřeba vytvořit samotný detail pro bankomat.

8.3.4 Zobrazení Detailu Bankomatu

Obrazovka detailu bankomatu se skládá z mapy ukazující pozici vybraného bankomatu. Detail obsahuje také akci pro vycentrování bankomatu v případě, že se ztratí z obrazovky vybraná plocha a nelze ji najít zpět.

K zobrazení mapy využijeme Map komponentu obsaženou přímo v Expo rozhraní. Komponenta používá aplikaci Apple Mapy v iOS zařízení nebo Google Mapy v případě Android zařízení. Expo využívá *react-native-maps* knihovnu. Nevyžaduje žádné nastavení v rámci aplikace vytvořené pomocí Expo. Samozřejmě při nasazení aplikace je potřeba nakonfigurovat jednotlivé mapy pro zařízení. Ale ještě nejdřív, než si zobrazíme mapu, musíme získat konkrétní informace o zobrazovaném bankomatu. (46)

Samotný detail je tvořen jako hrstka komponent bez potřebných informací. Stejně jak u webových aplikací, detail se skládá ze zobrazení konkrétního prvku z výpisu. U React Native nelze zadávat odkaz pro přechod do detailu. Místo toho využíváme data posílaná pomocí odkazu do detailu. Když ve výpisu klikneme na určitý prvek ve výpisu, přesměrujeme uživatele na detail spolu s údaji kliknutého prvku. Tím si získáme potřebné informace pro zobrazení na mapě. Abychom mohli přeposlat dané údaje využijeme další funkcionality *react-navigations* a to jsou props v přechodu.

```
1 onPress={() => {
2     this.props.navigation.navigate('atmDetail', {item: item})
3 }}
```

Kód 27: Předávání parametrů při přechodu do detailu

Každý prvek s sebou nese navěšenou interakci na kliknutí. Při kliknutí na prvek zavoláme metodu navigaci a vyvoláme akci *navigate* pro přesměrování na požadovanou obrazovku. Jako první argument vložíme klíč obrazovky, druhý argument je pak objekt dostupný z vyvolené obrazovky.

```
const { navigation} = this.props;
const item = navigation.getParam('item');
```

Kód 28: Uložení parametru předaný z navigace

Data jsou předána jako props uvnitř *navigation* prvku. Zavoláme metodu *getParam* s požadovaným argumentem odpovídající klíči, předaný při přechodu z výpisu. Jakmile máme potřebná data, lze konečně přejít k vykreslení mapy. K tomu tedy využijeme element *MavView*. Mapa vyžaduje pro správné vykreslení čtyři základní hodnoty – zeměpisná šířka, zeměpisná délka a přiblížení mapy pomocí *delta*.

```
class Map extends Component {
    state = {
        region: {
            latitude: this.props.item.location.latitude,
            longitude: this.props.item.location.longitude,
            latitudeDelta: 0.01,
            longitudeDelta: 0.008,
        }
    },
    render() {
        return(
            <View style={styles.container}>
                <MapView
                    ref={ref => this.map = ref}
                    style={styles.map}
                    initialRegion={this.state.region}
                >
                </MapView>
            </View>
        )
    }
}
```

Kód 29: Zobrazení mapy na vybraných souřadnicích

Po předání požadovaných parametrů do mapy, se bankomat vycentruje na mapě. Avšak na mapě není bankomat nijak viditelný. K tomu slouží takzvané

markery označující viditelně místo na mapě. Naštěstí Expo obsahuje i daný element. Stačí uvnitř mapy přidat nový element marker se souřadnicemi.

```
<MapView
    ref={ref => this.map = ref}
    style={styles.map}
    initialRegion={this.state.region}
>
    <MapView.Marker
        title={item.type}
        description={item.address.streetAddress}
        coordinate={item.location}
    />
</MapView>
```

Kód 30: Zobrazení markeru na mapě

Když máme připravenou mapu s vytyčeným bodem na bankomat, je potřeba přidat tlačítko. Funkce tlačítka je vycentrování zpět na prvopočáteční pozici při zobrazení detailu. Využívání funkcionality je využito pak při ztracení na mapě, kdy se při stisknutí tlačítko vycentruje původní pozice.

K implementaci použijeme React Native element *TouchableOpacity*. Při stisknutí zavoláme metodu, odkazující na mapu k zobrazení určité oblasti. Mapa k tomuto účelu využívá vlastní metodu *animateToRegion* obsahující parametr stejný jako samotná mapa – region.

```
centerMap() {
    this.map.animateToRegion(this.state.region)
}

<View style={buttonContainer}>
    <TouchableOpacity
          onPress={() => this.centerMap()}
          style={[bubble, button]}
    >
        <Text style={buttonText}>Vycentrovat</Text>
    </TouchableOpacity>
</View>
```

Kód 31: Ukázka implementace metody na vycentrování mapy

Chyba! Nenalezen zdroj odkazů.31 odkazuje na ukázku implementace. E lementy jsou umístěny uvnitř *render* metody na úrovni mapy. Pomocí stylů je tlačítko pozicování absolutně nad mapou v dolní části obrazovky.

Načítání obrazovky při přechodu však není uživatelsky přívětivé. Vykreslení mapy s určitými souřadnicemi není okamžité a může trvat několik vteřin. Abychom zlepšili uživatelský požitek z aplikace, implementujeme načítací stav aplikace. Vykreslíme při prvním načtení *perpetuum mobile* – nekonečně točící se kolečko. Poté co se mapa kompletně vykreslí, vyměníme obrazovku s načítáním za plně připravenou mapou.

K dosažení daného efektu je potřeba definovat proměnnou *isLoading* typu *Boolean* (může nabývat dvou možných stavů – pravda, nepravda). Jako prvotní hodnotu nastavíme pravda. Při zobrazování mapy vytvoříme podmínku na hodnotu *isLoading*. Pokud nabývá hodnoty pravda, označuje tak stav načítání mapy a je vykreslený prvek zobrazující načítání obrazovky.

```
1 if (this.state.loading) {
2     return <ActivityIndicator style={styles.spinner} size="large" />
3 } else {
4     return <Map item={item} onReady={this.setState({loading: false})} />
5 }
```

Kód 32: Efekt načítání mapy

Následně má mapa vlastní událost *onMapReady*, která se spustí poté co je mapa zcela načtena.

Jelikož naše komponenta je potomek rodičovské komponenty s logikou načítání, je potřeba poslat informaci rodiči o stavu Mapy. Na danou událost můžeme následně připojit vlastní aplikační kód

V neposlední řadě je potřeba doladit mapu. Při prohlížení vidíme pouze bankomat, ale chybí zobrazení uživatele na mapě. Abychom mohli lépe najít bankomat, je potřeba vědět kde se zrovna nacházíme. Mapa má však bohaté rozhraní a dostatečnou konfiguraci. Zobrazit uživatele je pak jednoduché, pomocí *showUserLocation*. Tím se při otevření mapy zobrazí dotaz na svolení se sdílením dat. Pokud uživatel odmítne, nelze zobrazovat aktuální pozici. S vykreslením

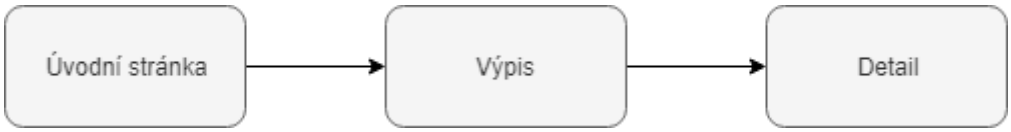
uživatele na mapě funguje i jeho aktualizace při pohybu, tvořící lepší efekt při vyhledávání bankomatu.

8.4 Řešení problémových situací

V této části jsou rozepsány jednotlivé situace, které mohou mít negativní efekt pro funkčnost aplikace buď nedbalostí nebo neznalostí. V článcích nebo návodech vždy narazíme na bezproblémové řešení, ve kterém autor popisuje, jak a co mu funguje. Nejsou však rozepsány části, kdy napsal nefunkční kód a jak přišel na opravu. Přičemž je důležité si uvědomit, proč nastal problém k tomu, aby se našlo správné řešení. Proto jsou zde shrnuty některé části, v kterých je vysvětlen problém a následné hledání správného řešení.

8.4.1 Návrh navigace aplikace

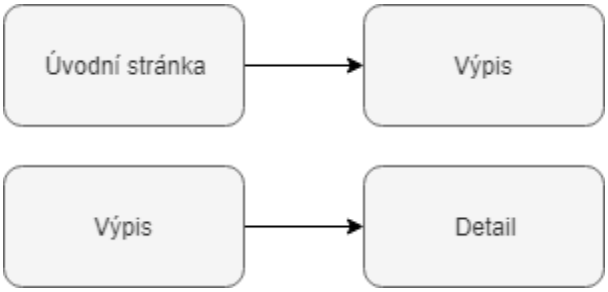
Na první pohled tvorba navigace pro celou aplikaci není složitá. Předem známe i jednotlivé přechody mezi obrazovkami. I přesto se může tato část stát problémovou. V naší aplikaci je situace, kdy máme jednoduchou strukturu navigace. Máme úvodní stránku, z které se dostaneme na výpis a následně z výpisu do detailu.



Obrázek 21: Navigace aplikace

(47)

Na první pohled se zdá řešení jednoduché. Podle dokumentace naší knihovny na tvorbu navigace lze zjistit, že pro tento případ funguje *createStackNavigation* funkce pro tvorbu navigace. Bez předchozích znalostí struktury mobilních aplikací je prvotní odhad vytvořit přechod z úvodní obrazovky do obrazovky výpisu. Následně postupovat vytvořením nové navigace z výpisu do detailu navigace. (47)



Obrázek 22: Špatný návrh navigace aplikace

Avšak při přechodu na detail nastane problém. Jsou aktuálně vidět dvě navigační lišty. První lišta je navigace mezi hlavní stránkou a výpisem ukazující prázdnou lištu. Druhá následně ukazovala to, co jsme požadovali – popis a tlačítko pro návrat. Pokoušet se skrývat menu pomocí stylování nebo API navigace nevyřešilo problém.

Řešením je vytvořit strukturu, kde jednotlivé obrazovky jsou na stejné úrovni. K jednotlivým přechodům využijeme metody, kde určíme, kam je uživatel přesměrován. Například nedovolíme přejít z hlavní stránky do detailu, i když je to možné. Navigace je i natolik chytrá, že tlačítko na předchozí obrazovku funguje i pro větší zanoření. Pokud dojdeme do detailu z úvodní stránky, tak zpětné tlačítko odkazuje na výpis a následně na úvodní obrazovku.

Tímto řešením odpadá duplikační navigační lišta a nemusíme vytvářet novou instanci *createStackNavigation*. Vše je ovládáno z jednoho místa, a tím se usnadní i následná rozšiřitelnost. Důležité je si promyslet při návrhu aplikace samotnou strukturu obrazovek a jak jsou provázány mezi sebou. Efektivním řešením je vytvořit si model aplikace pro lepší vizualizaci.

8.4.2 Odlišnost stylování mobilních aplikací

Další problémy, se kterými se může vývojář setkat je samotné stylování aplikace. Kdo není zvyklý stylovat pomocí CSS, může v této části vývoje narážet na nepříjemnosti. Celkový layout a pozicování prvků u React Native je tvořen za použití Flexboxu. Flexbox je moderní způsob, jak jednotlivé prvky zarovnávat jednoduše pomocí obalujícího elementu. Stal se tak moderní nástupcem oproti staršímu layout tvoření pomocí Floatových prvků. (37)

Prvotní problém nastal při tvorbě výpisu. Abychom mohli vyhledávat ve výpisu uvnitř elementu, musí element být roztažený na celou výšku obrazovky. V případě, že máme k tomu ukotvenou hlavičku nad výpisem, uvidíme poslední prvek useknutý o výšku hlavičky. Řešením je použít na obalující element výpisu hodnotu *flex:1*. Jedná se o zkratku pro vynucení roztáhnutí prvku do volného prostoru. Vynutíme roztáhnutí okna od hlavičky po spodní okraj obrazovky. Tím získáme viditelnost posledního prvku.

Stylování mobilních aplikací vychází ze stejného principu jako webové technologie. Využíváme samotné CSS pro stylování samotných prvků, jen jsou zapsány v jazyce JavaScript. Avšak to s sebou nese jistá omezení. Je potřeba si dávat pozor na efekty tvořené v mobilních aplikací. CSS v JavaScriptu je omezené na určité styly, obzvlášť u mobilních aplikací. Některé prvky nebo selektory nejsou podporovány. Příkladem mohou být stíny tvořené použitím *box-shadow*, nebo selektory *pseudo-element* pro elementy. Tímto je potřeba přizpůsobit samotný design aplikace. Většina vizuálních efektů se pak dá dosáhnout využitím obrázků. Bohužel tento způsob není až tak efektivní. Příkladem může být vytvoření stuhového efektu na kartičce. U webu bychom využili *pseudo-elementy* pro nastylováni daného efektu. U React Native aplikace tento vizuální vytvoříme za využití vložení obrázku, který umístíme absolutně na požadované místo.

Další výzvou požívání stylů u React Native aplikací je udržitelnost jednotného vzhledu. Je vhodné mít jednotné velikosti pro odsazování prvků, velikosti písma nebo samotné barvy. V případě, kdybychom dostali požadavek na změnu barvy jenom o menší odstín, museli bychom projít veškeré použití barev v celém projektu a nahrazovat barvu. Stejně tak to platí i u ostatních prvků. Řešením je vytvořit si soubor pro uchovávání těchto hodnot a sdílet je pro všechny případy. Ulehčí to práci se stylováním do budoucna a aplikace má tak lepší uživatelskou přívětivost. Další možnost, jak využívat stylování u JavaScriptových aplikací je knihovna *styled-components*. V případě, že aplikace má ambice se stát rozsáhlou, je vhodné už od začátku přemýšlet o použití pomocné knihovny na stylování.

8.4.3 Využívání lokálních souborů

Nejedná se tak o problém, jako o samotnou prevenci. Při používání vlastních souborů je důležité si dávat pozor na cestu k souboru. U vývoje si můžeme vystačit s relativní cestou. Soubor, nejčastěji obrázek, se nám zobrazí a vše se zdá v pořádku. Problém nastane až na hotové aplikaci. Při kompilaci aplikace se mohou cesty a umístění souborů měnit. Důvodem je sjednocení jednotlivých JavaScriptových souborů do balíčku. Kdybychom tedy v komponentě zanořené do nějaké složky potřebovali použít obrázek a využili složitou cestu k obrázku, po kompilaci se cesta může lišit.

Jednoduchým řešením je využívat metody *require()*. Metody přijímá jako argument relativní cestu k souboru s názvem. Zajistíme samotnou závislost obrázku na aplikaci a vytvoříme referenci na soubor.

Stejně jako při stylování, opakující se soubory je nejlepší vytěsnit do externího souboru. Soubor pak importujeme do potřebných komponent. V případě potřeby změny souboru stačí změnit pouze na jednom místě.

```
1 export const icons = {
2     airbank: require('../assets/imagegs/airbank.jpg'),
3     favicon: require('../assets/images/favicon.png'),
4 }
```

Kód 33: Uchovávání ikonek pro aplikaci

Názorný příklad můžeme použít objekt *icons*, uchovávající referenci na všechny ikonky v aplikaci. V případě potřeby nám stačí importovat objekt do potřebné komponenty a použít požadovanou ikonu. Uchováváme tak všechny ikony projektu na jednom místě.

8.4.4 Externí dotazování na API

Problém může nastat při samotném dotazováním na API. Momentálně pracujeme pouze s jednou pobočkou. Avšak i přitom je nutné přemýšlet na budoucí přidávání nových zdrojů. K tomu je důležité si navrhnout ukládání a zpracování jednotlivých dotazů na veřejné API.

Jako první bod je důležité samotné ukládání jednotlivých bankomatů. V našem případě se dotazujeme na více zdrojů. Přitom každý zdroj posílá zpět data ve svém vlastním formátu. Proto potřebujeme nějaký způsob, jak jednotlivá data sjednocovat a nevytvořit hromadu podmínek a složitou logiku pro zobrazení prostého názvu bankomatu.

Řešením tohoto problému je, že každá pobočka bude mít svoje vlastní mapování. Dotazování se ponechá stejné, ale při odpovědi zpracujeme data tak, aby odpovídala potřebnému formátu dat. Budeme takzvaně vytvářet model, kde si definujeme, jaké parametry potřebujeme zobrazovat a k tomu pro každou pobočku metodu na mapování dat.

```
1 airbankToAtm(atm) {
2     return {
3         name: 'Airbank',
4         icon: icons.airbank,
5         city: atm.address.city,
6         streetAddress: atm.address.streetAddress,
7         location: item.location
8     }
9 }
```

Kód 34: Uchovávání ikonek pro aplikaci

Při používání mapování výsledků, je nutno vracet data ve stejné struktuře. Pokaždé je potřeba mít název bankomatu. Každá pobočka má vlastní ikonku. Ve výpisu chceme stejně tak zobrazit město a ulici. Nakonec chceme vypočítávat vzdálenost a zobrazovat vše na mapě. K tomu je nutné mít souřadnice bankomatu. Tímto si zaručíme jednodušší pracování s daty a konzistentnost.

Dalším problémem je vyčkat na všechny dotazy do všech poboček a až následně poté vykonávat výpočet vzdáleností a řazení od nejbližšího bankomatu. Ve shrnutí je požadavkem vyslat několik požadavků najednou, kde u každého čekáme na odpověď. Při dosažení odpovědi se provede mapování a vrací se pole jednotlivých bankomatů. Po dokončení posledního požadavku, chceme všechny uložit do jednoho pole a následně vypočítat vzdálenost a seřadit od nejbližšího.

K řešení je nejlepší použít JavaScriptovou funkci *Promise.all*. *Promise.all* vyčkává na všechny dotazy na bankomaty, jakmile jsou dokončeny, vrací pole výsledků. (48)

```
async getAllAtms() {
    return Promise.all([getAirbankAtm, getCSOBAtm, ...]).then((values) => {
        return quickSortAlgorithm(getAtmsWithDistance(values));
    });
}
```

Kód 35: : Hromadné zpracování dotazů na API

Po dokončení *Promise.all* dostaneme veškeré požadované hodnoty, a lze použít algoritmus pro výpočet jednotlivých vzdáleností a následně seřadit bankomaty od nejbližšího. Celkově metoda *getAllAtms* vrací připravená data pro zobrazení do samotného výpisu.

8.4.5 Ukládání dat do mobilu

Každé spuštění aplikace momentálně spouští sekvenci dotazů na získávání bankomatů. Samo o sobě se data bankomatů nemění tak často. Jediné, co potřebujeme aktualizovat, jsou vzdálenosti vůči poloze zařízení. K tomu lze využít ukládání dat na mobilním zařízení.

Využijeme knihovnu redux-persist podporující React Native. Úkolem je při prvním spuštění aplikace uložit stažená data do mobilního zařízení a následně při každém spuštění nebo přechodu obrazovky se využívají již stažená data. Docílí se zrychlení aplikace a ušetření dat. Výhodou je i následné používání aplikace v off-line režimu.

Jak již bylo zmíněno, využijeme knihovnu redux-persist. Oproti standardním způsobům pro využívání localStorage použijeme AsyncStorage. AsyncStorage je jednoduché uložiště na principu klíč-hodnota používaná v aplikacích React Native. Uložená data jsou tak přístupná v celé aplikaci. Prvotně si vytvoříme konfigurační soubor. Předáme hodnoty uchovávané v Reduxu a ten lokálně uložíme. Abychom mohli využívat daný přístup, je potřeba data bankomatů uchovávat v Reduxu a posílat jej do samotného výpisu. Tím nebudeme muset zasahovat do logiky samotných komponent. Ty pouze budou číst samotná data z úložiště, která jsou reaktivní (pokud změníme nějaká data, automaticky se výsledek projeví bez vynucování překreslení). (49) (50) (51)

Při uchovávání dat na mobilním zařízení je nutné si dávat pozor na změnu struktury dat. V případě jakékoliv změny využívání a zobrazování nových dat nebo

změny struktury je potřeba připravit vynucené vymazání uložených dat. Kdybychom takto neučinili, při aktualizaci aplikace by předpokládala data s novou strukturou. Namísto toho by dostala stále uchovávaná data v zařízení a aplikace by pak nemusela pracovat dle očekávání.

V případě že bychom chtěli zachovat uchovávaná data, lze využívat *knihovnu redux-persist-migrate*. Pro změnu dat se vytvoří migrace, která převezme stávající data a z přenese je do požadované podoby a následně uloží. Migrace se spustí pouze jednou a nemusíme se o nic jiného starat. (49)

8.4.6 Ladění aplikace

Občas při vývoji nastává situace, kde aplikační logika nefunguje, jak bylo požadováno, i když náš kód je plně funkční a veškerá validace prošla bez problémů. V této situaci je nejspíše problém s předáváním jednotlivých dat v Reactu. K tomu nám pomocný debugger ani sledování http požadavků nepomůže. Potřebujeme sledovat aktuální stav uložených prvků uvnitř Reactu a jednotlivých komponent. Sledovat, jestli se při kliknutí pošle do komponenty požadovaná událost spolu se správnými daty.

Řešením je použití vývojářského nástroje „React Developer Tools“. Nástroj pro usnadnění ladění komponent psané v Reactu. Nástroj umožňuje právě integraci s React Native aplikacemi.

Využijeme možnost globální instalace balíčku a připojíme knihovnu pouze do našeho projektu.

```
1 npm install --save-dev react-devtools
```

Kód 36: Příkaz pro stažení nástroje pro ladění React aplikací

Následně přidáme nový skript do *package.json* ke spuštění nástroje. Tím se nám otevře samostatné nové okno podobné konzoli používané v moderních prohlížečích. Aplikace je psána v JavaScriptovém nástroji pro psaní desktopových aplikací. Po spuštění se zobrazí hierarchický výpis jednotlivých komponent. U každé komponenty lze sledovat její stavy a props. Výhodou je možnost přetěžovat a upravovat jednotlivá data v komponentách a sledovat následné chování. Při

sledování informací v komponentě, můžeme sledovat i aktuální styly připojené ke komponentě

8.4.7 Přenos aplikace do reálného zařízení

Pokud je aplikace funkční se všemi požadavky, je připravena pro použití na skutečném zařízení. Pro vydání aplikace do specifikovaných obchodů je potřeba splňovat licenční požadavky. Jednolité požadavky se ovšem liší pro každou platformu.

V případě iOS aplikace je proces podobný jako u nativních aplikací, avšak s několika kroky navíc. První krok se týká zabezpečení aplikace App Transport. Jedná o bezpečnostní funkcionalitu blokující veškeré požadavky posílané přes HTTP a nevyužívají HTTPS. Následně je potřeba si připravit konfiguraci schéma uvolnění aplikace. K tomu lze využít XCode zabudovaný formulář pro jednoduché sestavení schématu. Zablokuje aplikaci ve vývojovém režimu a následně sestaví veškeré potřebné JavaScripty lokálně. Umožní testování na skutečném zařízení. Poslední krok je samotné vypuštění aplikace do Apple obchodu. (52) (53)

Uvolnění aplikace u Androidu se liší v některých krocích. Stejně jako u iOS, většina kroků je podobná jako při vypuštění nativní aplikace. Velký rozdíl tvoří

```
1 BANKO_RELEASE_STORE_FILE=my-release-key.keystore
2 BANKO_RELEASE_KEY_ALIAS=my-key-alias
3 BANKO_RELEASE_STORE_PASSWORD=*****
4 BANKO_RELEASE_KEY_PASSWORD=*****
```

Kód 37: Příprava uvolnění Android aplikace

podepsaný APK[18] s certifikátem, než mohou být uvolněny do oficiálních Google obchodů. Pro vygenerování se využije *keytool*. Keytool vygeneruje osobní *keystore* soubor a ten by neměl být nikdy odeslán kamkoliv. Následně nastavíme Android gradle [19]proměnné pro vygenerovaný *keystore* soubor.

[18] APK – formát souboru balíčku používaný operačním systémem Android.
[19] Gradle – nástroj pro automatizované sestavování programu v jazyce Java

V neposlední řadě údaje předáme do konfiguračního souboru *build.gradle*. Zadáme heslo, předáme *keystore* soubor uložený jako globální proměnnou projektu. Nakonec pomocí příkazu vygenerujeme produkční APK soubor. Ten následně lze již nahrát na Google obchod.

Jelikož aplikace využívá nadstandardní funkce (mapy), je potřeba navíc nastavit samotnou konfiguraci pro mapy. V posledních letech Google vyžaduje vlastnit vygenerovaný API klíč pro využívání Google map. Klíč se dá získat na oficiálních stránkách a zaregistrováním aplikace. Následně je potřeba použít klíč v aplikaci. Přidáním do balíčku *android.package* registrujeme v konfiguraci API klíč.

```
"android": {
  "package": "ca.brentvatne.growlerprowler",
  "config": {
    "googleMaps": {
      "apiKey": "xxxxxxxxxxxxxxxxxxxxxxx"
    }
  }
}
```

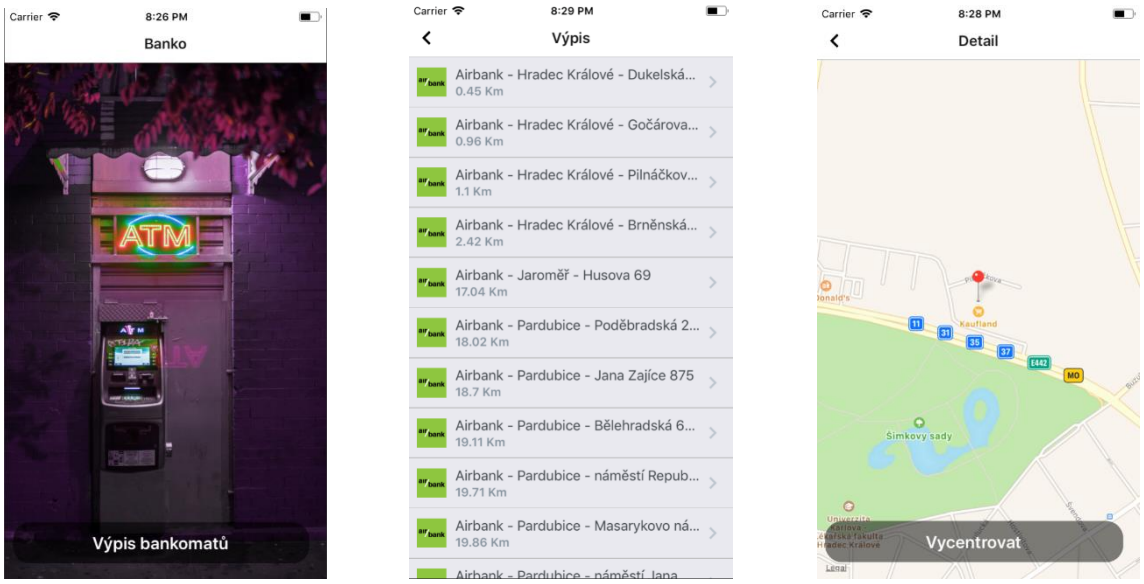
Kód 38: Registrace API klíče pro využívání Google map

Vygenerovaný otisk se vloží do *app.json* a uloží.

U iOS aplikací využívající Apple mapy funguje bez jakékoliv extra konfigurace. Pokud by byly vyžadovány Google Mapy, lze registrovat API klíč za využití *ios.config.googleMapsApiKey.*

8.5 Hotová aplikace

Vytvořená aplikace momentálně splňuje základní definované požadavky. Začíná úvodní stránkou, která je tvořena zatím pouze z výběru jedné funkce – výpis bankomatů. Při prokliku do výpisu se spustí stahování dat z API české banky AirBank. Po dostání jednotlivých bankomatů se provedlo zpracování dat – výpočet vzdálenosti a seřazení od nejmenší. Výpis je optimalizován na velké množství dat postupným načítáním dat při pohybu. Při výběru z výpisu se dostaneme na detail obsahující mapu. Na mapě je označena aktuální poloha uživatele spolu s vybraným bankomatem. K tomu je přidáno tlačítko pro vycentrování mapy zpět na bankomat.



Obrázek 23: Jednotlivé obrazovky aplikace Banko

Aplikace splňuje minimální požadovanou hodnotu, avšak má prostor pro budoucí přidání nové funkcionality.

8.6 Budoucí funkcionalita aplikace

I když aplikace je hotová, můžeme pořád zlepšovat její funkcionalitu. V této sekci si ukážeme, jak lze aplikaci vylepšit.

Jedna z možných funkcí je přidat možnost vyhledávání do filtrů. Ačkoliv jsou momentálně přidané bankomaty pouze z AirBank, mohou ostatní banky v budoucnu poskytovat svoji polohu bankomatů taky. K tomu lze implementovat výběr banky jako filtr, fulltextové vyhledávání nebo do určité vzdálenosti.

Mezi zajímavou funkcionalitu můžeme přidat do detailu bankomatu i ostatní nejbližší bankomaty, jen je nutné je vizuálně oddělit. Je zbytečné zobrazovat na mapě pouze jeden bankomat, když bychom mohli přidat i ostatní.

Aplikace je připravena na přidávání nových modulů. Tak byla i připravena úvodní stránka. Další plánovanou funkcí je převodník měn. Pomocí API lze získat aktuální kurz.

9 Shrnutí vývoje v React Native

V této části jsou rozebrány jednotlivé procesy vývoje a poznatky získané při vyvíjení aplikace za pomocí frameworku React Native. Před shrnutím je potřeba si jednotlivé části rozdělit podle kategorií. Výpis stanovených kategorií je následující:

- Využití frameworku
- Podpora nativních funkcí
- Použitá technologie
- Výsledný vzhled aplikace
- Složitost učení frameworku
- Komunita a podpora
- Očekávání

9.1 Využití frameworku

Existuje několik typu mobilních aplikací. Podle článku existuje více jak 33 typů aplikací na Google Play a 24 na Apple Storu. Mezi hlavní druhy aplikace lze kategorizovat 7 druhů. Jedná se o aplikace zaměřené na:

- Hry
- Nástroje
- Zábava
- Životní styl
- Zábava
- Produktivita
- Sociální sítě

React Native umožňuje vyvíjet všechny typy aplikací. Záleží pouze na schopnostech vývojáře a samotné komplexitě vyvíjené aplikace. Například lze vyvinout herní aplikaci, která není komplexní s určitým omezením. Avšak pro velké a složité hry a vysokou grafikou je určitě lepší zvolit vhodnější nástroj. Stejně tak komplexní uživatelské rozhraní s velkým počtem animací a zatíženým na grafiku. Framework umožňuje vytvářet výkonné aplikace, které se těžko dají rozeznat od

nativních. Framework lze tedy uplatnit u všech zmíněných kategoriích, avšak s omezením na herní aplikace. Aplikace zaměřené na nástroje využívající nativní funkcionalitu je potřeba doplnit potřebnými rozšířeními.

9.2 Podpora nativních funkcí

Při realizaci aplikace Banko se ukázalo, že potřebná nativní funkcionalita Geolokace je zabudována přímo v React Native a není potřeba doinstalovat rozšíření. Uplatnění funkcionality na projektu s sebou nenesla žádné problémy a využívání se stalo jednoduché a přímočaré. Dokumentace obsahovala veškeré potřebné informace a ukázkové kódy, jak funkcionalitu správně aplikovat.

React Native disponuje velkým počtem podporované nativní funkcionality. Pro některé funkcionalit existuje i více řešení vytvořené komunitou. Vývojář si tak může vybrat vhodné řešení poskytující požadovanou funkcionalitu.

9.3 Použitá technologie

Framework pro vývoj využívá webovou technologii React upravenou lehce upravenou pro vývoj mobilních aplikací.

React Native aplikuje pro svůj vývoj jazyk JavaScript. JavaScript posledních pár let prochází obrovským rozvojem. Každý rok se přidávají nové funckionality usnadňující vývoj a umožňující nové věci.

Moderní JavaScript přidal nové funkcionality a možnosti, jak zpracovávat asynchronní požadavky za posledních několik let. Až už je jedná o metodu „fetch“ nebo následné zpracovávání pomocí prototypu „then()“. V projektu se využila nová asynchronní funkce z ES 2017, a tou je „async await“. Pomocí funkce lze poslat požadavek a vyčkat na zpracování, než JavaScript bude pokračovat k dalšímu zpracování. Výsledkem je krátký a stručný zápis, který vytváří přehledný kód. U používání async await je potřena si dávat pozor na definování metody. Metoda musí obsahovat slovo async. Tím se označuje metoda jako asynchronní. K odchytávání JavaScript přidal novou funkcionalitu převzatou z jiných programovacích jazycích – „try catch“. Umožňuje odchytávat například chybové požadavky nebo případné zpracování dat.

Jak se postupně JavaScript rozvíjí, umožňuje frameworkům jako je React poskytovat nové funkce a jednodušší vývoj. Navíc u mobilních zařízení se nemusí řešit kompatibilita do takové míry jako u webových aplikací. Za použití transpilátorů poskytneme podporu. V případě webů musíme k tomu řešit podporu pro více druhů prohlížečů a operačních systémů. U mobilních zařízení se tento problém neřeší.

9.4 Výkon aplikace

React Native nabízí možnost vyvíjet výkonnostně porovnatelné aplikace jako nativní. Vývojáři je poskytnuto mnoho předem připravených komponent a funkcí pro dosáhnutí požadovaného výkonu aplikace.

I když React Native se snaží poskytovat vysoký výkon aplikacím, je potřeba se snažit o to samé při vývoji aplikace. Nedbalostí a neznalostí lze docílit zpomalení a využívání větších kapacit paměti, zatížit CPU nebo ztrátu vysokého FPS.

V projektu se snažíme zpracovávat bankomaty, kde jejich počet přesahuje 1000 záznamů. Jelikož každé API bank poskytuje data v jiném formátů je potřeba provést mapování do stejné podoby u všech bank – první iterace. Dále je potřeba znát jednotlivé vzdálenosti od aktuální pozice, tím potřebujeme pro každý bankomat vytvořit algoritmus na výpočet – druhá iterace. V neposlední řadě je potřeba jednotlivé záznamy seřadit od nejbližší bankomatu k aktuální poloze zařízení – třetí iterace. Pokud bychom neřešili žádnou optimalizaci ani výkon, všechny bankomaty (tisíce záznamů) bychom iterovaly vícekrát, kde při každé iteraci záznamů prováděli výpočet. Z tohoto hlediska je důležité si uvědomit i optimalizace na straně vývojáře a nespoléhat se, že technologie vyřeší všechny problémy.

Další optimalizace je tok dat. Při načtení je nutné stáhnout velký počet záznamů z různých zdrojů. Přitom se jednotlivé údaje o bankomatech nemění. Změna polohy bankomatů není denní záležitost. React Native tak nabízí možnost ukládat data na zařízení a předcházet možným duplicitním dotazům, ušetří tak přenos dat a urychlí aplikaci při opětovném spuštění. Framework tak nabízí využívat locakStorage, stejně jako webové aplikace. Pro lepší funkcionalitu a kontrole lze přidat i jednoduché databáze jako jsou SQLite nebo PouchDB pro React Native.

Výsledný výkon aplikace nezáleží pouze na zvolené technologii a na tom, co nabízejí vývojáři z Facebooku. Vytvořili framework, s možností dosahovat skoro až nativního výkonu. Je pouze na vývojáři, jestli bude při vývoji brát v úvahu i optimalizaci a zlepšení výkonu aplikace pro lepší uživatelský zážitek.

9.5 Výsledný vzhled aplikace

Pro vzhled React Native využívá CSS vlastnosti přebrané z webových technologií, akorát aplikované přes JavaScript. Ovšem ne všechny vlastnosti jsou podporované u mobilních zařízení. Například nelze používat pseudo elementy a jiné vlastnosti.

Další je si potřeba uvědomit, že veškeré sestavování layout (grid, uspořádání element) je pomocí CSS vlastnosti Flexbox. Flexbox je CSS modul pro tvorbu layoutu, který usnadňuje vytvářet responzivní strukturu layout bez používání floatu nebo pozicování. Lze skládat jednotlivé elementu podle potřeby. Flexbox se stal ihned populární u webových aplikací, kde si získal velký úspěch, u mobilních aplikace tomu není jinak.

9.6 Složitost učení frameworku

Framework se zpočátku zdát komplikovaný na naučení. V podstatě vývoj mobilních aplikací v React Native je jako vyvíjet JavaScriptové aplikace za využití React frameworku. Strukturu a způsob psaní se vývojáři snažili zachovat stejný, akorát je přidána podpora pro komponenty vytvořené pro mobilní zařízení.

Zprvu se React Native může zdát komplikovaný, avšak po naučení se pár základních elementů lze vytvářet základní vzhled a logiku aplikace. Framework sám o sobě není komplikovaný a nabízí pouze potřebné prvky uzpůsobené pro mobilní zařízení. Většina programování a logiky se vyvíjí v samotném jazyce JavaScript. Pokud vývojář zná základy jazyka JavaScript, lze si snadno a rychle zvyknout na syntax React Native a zaměřit se na učení React architektury a využívání funkcí framework nabízí.

Veškeré potřebné informace lze nalézt na oficiální dokumentaci frameworku. Dokumentace je přehledná a poskytuje ukázkové příklady, jak danou funckionalitu používat se všemi potřebnými atributy a metodami.

Komunita kolem Reactu a React Native je velmi aktivní. Vývojář tak může čerpat z mnoha článků, kde ostatní vývojáři sdílejí svoje vědomosti a možnosti řešení konkrétních problémů. Pokud vývojář nemůže nalézt řešení pro svůj problém, může napsat dotaz na fórum. „Stack Overflow“ je aktivním místem pro pokládání otázek, kde komunita je aktivní a lze získat odpovědi na položené otázky.

9.7 Očekávání frameworku

Od React Native se očekávala možnost vývoje multiplatformních aplikace za využití jazyka JavaScript. React Native měl být další framework s přístupem „run once, run anywhere“, neboli napiš jednou a využij kód všude pro všechny podporované platformy. Vzhledem k tomu, že Facebook na svém blogu jasně uvádí, že uznávají rozdíly mezi platformami, jejich cílem je přenést paradigma Reactu, který je velmi úspěšný na webu, na nativní aplikace se zachováním funkcionality pro kteroukoliv platformu. Místo toho lze u React Native aplikovat přístup „naučit jednou, aplikovat všude“.

Zklamáním u frameworku React Native je jeho stabilita. I když existuje již několik let, nevydal stabilní verzi. Místo toho pořád vyvíjí v nulové verzi, kde může při každé aktualizaci rozbít zpětnou kompatibilitu. Tento případ se stal při vývoji aplikace. Během implementace vyšla nová verze se změnou v API a přidala se podpora Expa do nativního buildu. Bohužel jednotlivé verze nebyli uzamknuté a při instalaci závislostí se stáhla aktuální a projekt bylo potřeba opravit k samotnému spuštění. Pro dlouhodobém vývoji u React Native je teda potřeba kontrolovat aktualizace a buď aktualizovat a pak sledovat návody k migraci nebo uzamknout jednotlivé verze balíčků a odmítnout tak nové funkce nebo opravy chyb.

9.8 Shrnutí

React Native umožnil vyvinout výkonnou aplikaci pro obě platformy. Příprava samotného vývojového prostředí neobnáší velké úsilí. Stačí jednou připravit a lze vyvíjet jednu aplikaci za druhou. React Native má velkou komunitu a návody, jak s frameworkem pracovat. Učící křivka frameworku je vysoká, kde po chvilce programování lze aplikovat stejný postup a vyvíjet rychle a efektivně se orientovat

v projektu. Framework poskytuje v základu výkonné jádro pro vývoj mobilních aplikací, ovšem je pouze na vývojáři, jak dokáže framework efektivně použít.

Nicméně po vývoji lze říct, že React Native není stabilní platforma. React Native z tohoto důvodu není vhodný pro vývoj dlouhodobě podporovaných aplikací. To však nevylučuje možnost, aby byl použit pro jiné druhy aplikace, které mají správný tým včetně správné úrovně znalostí jazyka JavaScript. React Native se zdá být vhodný pro jednorázové aplikace, které potřebují krátkou podporu. Tyto druhy aplikací mohou také těžit z poznatků a možnost využívat stále nejnovější funkcionality.

10 Závěr

V této diplomové práci byl vysvětlen pojem multiplatformní vývoj mobilních aplikací. Práce se zaměřuje na přiblížení čtenáře do problematiky vývoje mobilní aplikace a rozdílů mezi nativním přístupem a hybridním. Následuje srovnání vybraných alternativ pro vývoj mobilních aplikace. Výsledky slouží především pro usnadnění výběru vhodného nástroje.

Prvním cílem bylo vytvořit zmíněnou analýzu existujících frameworků. Každá vybraná technologie byla charakteristicky popsána stejným způsobem pro následné lepší porovnání. Podstatnou částí bylo srovnání jednotlivých frameworků. Z výsledků nelze přímo vyvodit, zda je nějaký z frameworků komplexně lepší než ty ostatní. Každý framework má svoje silné a slabé stránky, a je potřeba využít správnou technologii pro odpovídající případ. Ionic je snadno naučitelný, pomocí kterého lze rychle vyvíjet jednoduché mobilní aplikace s plnou podporou nativních funkcí. React Native nabízí podobné funkce s vyšším výkonem ale s nutnou předchozí znalostí JavaScriptu. Xamarin a Flutter jsou komplikovanější ale výkonné frameworky pro tvorbu bezpečných a grafických aplikací.

Dalším cílem bylo detailněji představit vybraný framework React Native a za jeho pomoci vyvinout aplikaci. Prvotně bylo důležité definovat jednotlivé požadavky na aplikaci, které musí splňovat. Následně je ukázána příprava prostředí a implementace aplikace, kde jsou popsány jednotlivé kroky vývoje. Po dokončení aplikace jsou vysvětlena jednotlivá řešení problémů, na která se při vývoji narazilo. Vždy je snaha o představení problému s odůvodněním a následným řešením.

V poslední části jde o samotné vyhodnocení vývoje aplikace za využití frameworku React Native. Ukazuje se, že bez předchozích znalostí JavaScriptu mohou být začátky komplikovanější, ale křivka učení zde zejména v začátcích velmi rychle a efektivně roste. Stejně tak výkon aplikace také závisí na samotném vývojáři a jeho optimálním využití frameworku. Spolu s velkou komunitou není problém nalézt již hotové řešení nebo dostat radu k vyřešení požadovaného problému. Framework však nemusí být stále stabilní a jednotlivé aktualizace mohou rozbít předchozí kompatibilitu. Proto je React Native vhodný využívat na projekty bez dlouhodobé údržby.

Samotný výběr možných technologií není úplně jednoznačný a vždy bude záležet i na jiných aspektech. Nejdůležitějším je požadavek na aplikaci. Má-li být aplikace bezpečná a výkonná, nebo zda je nutnost aplikaci dodat co nejdříve k představení klientovi. Stejně tak jsou podstatné znalosti a prostředí odkud vývojář přichází. Pokud vývojář má znalosti více z Javy nebo jiného objektově orientovaného programovacího jazyka, nebude mu dělat problém naučit se Flutter nebo Xamarin. Závěrem lze říct, že výběr závisí pouze na vývojáři, jestli mu framework vyhovuje a poskytuje potřebnou funkcionalitu.

11 Terminologický slovník

Termín	Význam
API	Application Programming Interface je rozhraní, které je vystaveno programátorovi pro komunikaci se softwarem.
DOM	Document object model označující – slouží jako API umožňující přístup nebo modifikaci obsahu na stránce.
CSS	Cascading Style Sheets slouží jako standard pro vytváření vizuálního formátování webu.
HTML	Značkovací jazyk skládající se elementů pro tvorbu webových stránek.
SDK	Sada nástrojů pro vývoj softwaru. Pomocí nástrojů lze vyvíjet aplikace, frameworky nebo počítačové systémy.
MVC	Softwarová architektura rozdělující aplikaci na tři nezávislé komponenty – datový model, řídící logiku a uživatelské rozraní
UI	UI neboli uživatelské rozhraní. Jedná se o sadu příkazů jejichž prostřednictvím uživatel komunikuje s programem.
UX	UX neboli uživatelská přívětivost. Určuje, jaký pocit má uživatel při používáním aplikace. Čímž lepší je uživatelská přívětivost, tím líp se koncovému uživateli pracuje s programem.
Front-end	Viditelná část webu / aplikace koncovým návštěvníkům.
JIT	Zkratka *Just In Time* (právě v čas) – metodika využívající techniky pro urychlení běhu programů.
Flexbox	Flexbox poskytuje efektivní uspořádání a zarovnání jednotlivých prvků v samotném kontejneru.
XML	Rozšiřující značkovací jazyk k vytváření nových značkovacích jazyků nebo serializaci dat.

12 Bibliografie

[1]. Hartmann, Gustavo. Cross-platform mobile development. [Online] Březen 2011. [Citace: 10. Září 2018.] https://wss.apan.org/jko/mole/Shared%20Documents/Cross-Platform%20Mobile%20Development.pdf.

[2]. What is an operating system? *Modern operating systems.* [Online] 8. Srpen 2018. http://stst.elia.pub.ro/news/SO/Modern%20Operating%20System%20-%20Tanenbaum.pdf.

[3]. Grifith, C. *Mobile App Development with Ionic: Cross-platform Apps.* California : O'Reilly Media., 2017.

[4]. Mobile Operating System Market Share Worldwide. *Statcounter.* [Online] 2. Srpen 2018. http://gs.statcounter.com/os-market-share/mobile/worldwide/2018.

[5]. Announcing the Android 1.0 SDK. *Android Developers Blog.* [Online] 5. Červen 2018. https://android-developers.googleblog.com/2008/09/announcing-android-10-sdk-release-1.html.

[6]. The Android Story. *Android.* [Online] 24. Červen 2018. https://www.android.com/history.

[7]. Google. The Android Source Code. *Android.* [Online] 8. Červen 2018. https://source.android.com/setup.

[8]. —. Platform Arcitecture. *Android.* [Online] 3. Srpen 2018. https://developer.android.com/guide/platform/.

[9]. Develop Android apps Language. *Android Authority.* [Online] 7. Srpen 2018. https://www.androidauthority.com/develop-android-apps-languages-learn-391008/.

[10]. The evolution of iOS. *CultofMac.* [Online] 8. Srpen 2018. https://www.cultofmac.com/488454/ios-evolution-iphone-os/.

[11]. iOS Architecture. *Intellipaat.* [Online] 8. Srpen 2018. https://intellipaat.com/tutorial/tutorial-ios-tutorial/ios-architecture/.

[12]. Apple Developer Program. *Apple.* [Online] 9. Srpen 2018. https://developer.apple.com/programs/.

[13]. Čarapina, M., Mekterović, I., Jagušt, T., Drljević, N., Baksa, J.,Kovačević, P., and Botički, I. Developing a multiplatform solution for mobile learning. *International Conference on Computers in Education.* [Online] 2015.

[14]. Raj, R., and Tolety, S.B. *A study on approaches to build cross-platform mobile.* India : INDICON, INDICON, 2012. IEEE.

[15]. A Touch-Optimized Web Framework. *jQuey Mobile.* [Online] jQuery. [Citace: 8. Říjen 2018.] https://jquerymobile.com.

[16]. Build amazing mobile apps powered by open web tech. *Adobe PhoneGap.* [Online] Adobe Systems inc. [Citace: 10. Září 2018.] https://phonegap.com/.

[17]. Dokumentace pro Xamarin. *Microsoft.* [Online] [Citace: 10. Září 2018.] https://docs.microsoft.com/cs-cz/xamarin/.

[18]. Yusuf, S. *Ionic Framework by Example.* Birmingham : Packt, 2016.

[19]. Introduction. *Cordova Apache.* [Online] [Citace: 2. Duben 2019.] https://cordova.apache.org/docs/en/latest/guide/overview/index.html.

[20]. Ionic Framework. *Ionic Docs.* [Online] [Citace: 2. Duben 2019.] https://ionicframework.com/docs.

[21]. About Mono. *Mono Project.* [Online] [Citace: 3. Duben 2019.] https://www.mono-project.com/docs/about-mono/.

[22]. Microsoft. *Xamarin Architecture.* [Online] [Citace: 4. Duben 2019.] https://docs.microsoft.com/en-us/xamarin/android/internals/architecture.

[23]. Xamarin iOS Architecture. *Microsoft.* [Online] [Citace: 4. Duben 2019.] https://docs.microsoft.com/cs-cz/xamarin/ios/internals/architecture.

[24]. Xamarin.Forms. *Microsoft.* [Online] [Citace: 4. Duben 2019.] https://docs.microsoft.com/cs-cz/xamarin/xamarin-forms/.

[25]. Eisenman, B. *Learning React Native.* Sebastopol, CA : O´Reilly, 2015. ISBN 978-1-4919-2900-1.

[26]. Facebook. React Native. [Online] 14. Červenec 2018. https://facebook.github.io/react-native/.

[27]. Google. Flutter. *Flutter - Beautiful native apps in record time.* [Online] [Citace: 11. Září 2018.] https://flutter.io/.

[28]. Willocx, M., Vossaert, J., and Naessens, V. A quantitative assessment. *Mobile Services.* [Online] 2015. [Citace: 10. Září 2018.] IEEE.

[29]. Leler, Wm. What's Revolutionary about Flutter. *Hackernoon.* [Online] 25. Srpen 2017. [Citace: 11. Září 2018.] https://hackernoon.com/whats-revolutionary-about-flutter-946915b09514.

[30]. ECMA. ECMA Standards. *Standard ECMA-262.* [Online] [Citace: 14. Září 2018.] https://www.ecma-international.org/publications/standards/Ecma-262.htm.

[31]. Mozilla. About JavaScript. *MDN .* [Online] Mozilla, 24. Červen 2018. https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript.

[32]. What is Babel. *Babel.* [Online] [Citace: 12. Září 2018.] https://babeljs.io/docs/en/.

[33]. React. [Online] Facebook. [Citace: 18. Prosinec 2018.] https://reactjs.org/.

[34]. What Is ReactJS and Why Should We Use It? *C#Corner.* [Online] 14. Listopad 2018. [Citace: 2018. 18 Prosinec.] https://www.c-sharpcorner.com/article/what-and-why-reactjs/.

[35]. Virtual DOM and Internal. *Reactjs.* [Online] Facebook. [Citace: 28. Září 2018.] https://reactjs.org/docs/faq-internals.html.

[36]. JSX Specification. *Facebook.* [Online] [Citace: 11. Září 2018.] https://facebook.github.io/jsx/.

[37]. Compoents and APIs. *React Native.* [Online] Facebook. [Citace: 20. Prosinec 2018.] https://facebook.github.io/react-native/docs/components-and-apis.html.

[38]. Understanding React - Component life-cycle. *Medium.* [Online] 6. Září 2017. [Citace: 20. Prosinec 2018.] https://medium.com/@baphemot/understanding-reactjs-component-life-cycle-823a640b3e8d.

[39]. The component lifecycle. *Packt.* [Online] [Citace: 21. Prosinec 2018.] https://subscription.packtpub.com/book/web_development/9781785885785/1/ch01lvl1sec16/the-component-lifecycle.

[40]. Props. *React Native.* [Online] Facebook. [Citace: 28. Prosinec 2018.] https://facebook.github.io/react-native/docs/props.

[41]. State. *React Native.* [Online] Facebook. [Citace: 28. Prosinec 2018.] https://facebook.github.io/react-native/docs/state.

[42]. Gesture Responder System. *React Native.* [Online] Facebook. [Citace: 29. Prosinec 2018.] https://facebook.github.io/react-native/docs/gesture-responder-system.

[43]. StyleSheet. *React Native.* [Online] Facebook. [Citace: 29. Prosinec 2018.] https://facebook.github.io/react-native/docs/stylesheet.

[44]. Gettings Started with Redux. *Redux.* [Online] [Citace: 3. Leden 2019.] https://redux.js.org/introduction/getting-started.

[45]. Docs. *Reat Redux.* [Online] 17. Srpen 2018. https://react-redux.js.org/.

[46]. Facebook. Jest. *Delightful JavaScript Testing.* [Online] 26. Srpen 2018. https://jestjs.io/en/.

[47]. Debugging. *React Native.* [Online] Facebook. [Citace: 6. Leden 2019.] https://facebook.github.io/react-native/docs/debugging.

[48]. Axios. *Promise based HTTP client for the browser and node.js.* [Online] 15. Srpen 2018. https://github.com/axios/axios.

[49]. Calculate distance, bearing and more between Latitude / Longitude points. *Movable Type Scripts.* [Online] [Citace: 2. Únor 2019.] https://www.movable-type.co.uk/scripts/latlong.html.

[50]. Dorantes, César Antón. Quicksort Algorithm. *Medium.* [Online] 8. Září 2018. https://medium.com/cesars-tech-insights/quicksort-17c5d24e7e5f.

[51]. FlatList. *React Native.* [Online] [Citace: 16. Únor 2019.] https://facebook.github.io/react-native/docs/flatlist.

[52]. Expo. *Expo Doc.* [Online] 25. Srpen 2018. https://docs.expo.io.

[53]. React Navigation. *Doc.* [Online] 9. Srpen 2018. https://reactnavigation.org/.

[54]. Promise.all(). *MDN.* [Online] Mozilla. [Citace: 2. Únor 2019.] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/all.

[55]. Redux Persist. *Github.* [Online] 26. Srpen 2018. https://github.com/rt2zz/redux-persist.

[56]. About SQLite. *SQLite.* [Online] 7. Srpen 2018. https://www.sqlite.org/index.html.

[57]. Window.localStorage. *Developer Mozilla.* [Online] Mozilla, 8. Srpen 2018. [Citace: 17. Září 2018.] https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage.

[58]. How to Deploy a React Native App for iOS, Android. *Dev.* [Online] 28. Listopad 2018. [Citace: 18. Únor 2019.] https://dev.to/iverve/how-to-deploy-react-native-app-546n.

[59]. Running On Device. *React Native.* [Online] Facebook. [Citace: 17. Únor 2019.] https://facebook.github.io/react-native/docs/running-on-device.

[60]. Github. *React Native Mapview component for iOS + Android.* [Online] 7. Červenec 2018. https://github.com/react-community/react-native-maps.

[61]. Kilián, Karel. Co je NFC a k čemu je dobré ho použít? *SvetAndroida.* [Online] 6. Červenec 2018. https://www.svetandroida.cz/co-je-nfc-k-cemu-je-dobre-ho-pouzit/.

[62]. About webkit. *Webkit.* [Online] 7. Srpen 2018. https://webkit.org/.

[63]. Bytecode. *Techopedia.* [Online] 7. Srpen 2018. https://www.techopedia.com/definition/3760/bytecode.

[64]. Android NDK. *Developer Android.* [Online] 7. Srpen 2018. https://developer.android.com/ndk/.

[65]. Boushehrinejadmoradi, N., Ganapathy, V., Nagarakatte, S., and Iftode, L. Testing cross-platform mobile app development frameworks. *International Conference on Automated Software.* [Online] Listopad 2015. [Citace: 10. Září 2018.] IEEE / ACM.

[66]. w3schools.com. XML Tutorial. *w3schools.* [Online] [Citace: 11. Září 2018.] https://www.w3schools.com/xml/.

[67]. What are the popular types and categories of apps. *ThinkMobiles.* [Online] [Citace: 6. Duben 2019.] https://thinkmobiles.com/blog/popular-types-of-apps/.

13 Seznam Tabulek a obrázků

Tabulky:

Obrázky:

Kódy:

Přílohy

Příloha 1: Zadání diplomové práce

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

Studijní program: Systémové inženýrství a informatika
Forma: Prezenční
Obor/komb.: Informační management (im2-p)

Podklad pro zadání DIPLOMOVÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Bc. Robotka Jakub	Dolní Libchavy 250, Libchavy - Dolní Libchavy	I14753

TÉMA ČESKY:

Multiplatformní vývoj mobilní aplikace

TÉMA ANGLICKY:

Cross-platform development of mobile application

VEDOUCÍ PRÁCE:

Ing. Jiří Štěpánek - KIT

ZÁSADY PRO VYPRACOVÁNÍ:

Cíl:
Cílem diplomové práce je seznámit čtenáře s pojmem hybridní mobilní aplikace. Jak se tento přístup liší od ostatních, pro jaké platformy se nejčastěji používá. Dalším cílem je vytvoření mobilní aplikace pomocí vybraných frameworků, jejich představení a základní analýza.

Osnova:
1. Úvod do mobilních aplikací
2. Multiplatformní vývoj
3. Nástroje pro vývoj hybridních aplikací
4. Vývoj pomocí React Native
5. Vývoj referenční aplikace
6. Shrnutí vývoje v React Native
7. Závěr

SEZNAM DOPORUČENÉ LITERATURY:

https://facebook.github.io/react-native/docs/getting-started
https://flutter.dev/docs
http://ionicframework.com/
https://cordova.apache.org/
A Step By Step Guide to Cross Platform Hybrid (HTML5) Apps for Android, BlackBerry, Firefox OS, iOS, and Windows Phone: Claudiu Militaru