Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informatiky a kvantitativních metod

Porovnání mobilních frameworků

Bakalářská práce

Autor: Ondřej Schneider

Studijní obor: Aplikovaná informatika

Vedoucí práce: doc. Ing. Filip Malý, Ph.D.

Hradec Králové srpen 2019

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne Ondřej Schneider

Poděkování:

Děkuji doc. Ing. Filipu Malému, Ph.D. za odborné vedení při zpracování této bakalářské práce.

Anotace:

Cílem bakalářské práce je porovnání frameworků pro vývoj multiplatformních mobilních aplikací z pohledu navržené vzorové aplikace. Teoretická část se nejprve zabývá mobilními aplikacemi a mobilními operačními systémy. Následuje seznámení s frameworky se zaměřením na frameworky pro vývoj multiplatformních mobilních aplikací a je představeno několik těchto frameworků v podobě základních informací. V praktické části je navržena a implementována vzorová aplikace ve dvou vybraných frameworcích. Na základě této aplikace jsou poté porovnány oba frameworky z hlediska uživatelského rozhraní, implementace funkcí a využití zdrojů či API třetích stran využívaných v aplikaci.

Annotation:

Title: Comparison of mobile frameworks

The aim of this Bachelor Thesis is to compare frameworks for development of cross-platform mobile applications from the point of view of designed sample application. The theoretical part firstly deals with mobile applications and mobile operating systems. The following is an introduction to frameworks focusing on frameworks for the development of cross-platform mobile applications and several of these frameworks are presented in the form of basic information. In the practical part is designed and implemented sample application in two selected frameworks. Based on this application, the two frameworks are then compared in terms of user interface, implementation of functions and use of third-party resources or APIs used in the application.

Obsah

1 Úvod

Frameworky jsou důležitou součástí vytváření všech možných druhů aplikací. Díky jejich schopnostem usnadnění práce a ušetření času developerům se stávají základním kamenem tvorby aplikací a jejich oblíbenost u vývojářů stoupá.

Tato práce se zaměřuje na mobilní aplikace a frameworky pro jejich tvorbu. Počet mobilních aplikací, díky narůstající popularitě chytrých telefonů, tabletů a dalších chytrých zařízení, neustále roste. Počet stažených mobilních aplikací uživateli skrze všechny platformy v roce 2018 přesáhl 200 biliónů. Toto číslo by se dle odhadů mělo zvýšit až na necelých 260 biliónů v roce 2022. (1) To vše vypovídá o důležitosti tohoto tématu.

Teoretická část bakalářské práce má za úkol seznámit čtenáře s oblastí mobilních aplikací – vysvětlení pojmu, trendy, a především jejich tvorba. Dále představit platformy, na kterých aplikace běží (především pak tři nejužívanější – *iOS, Android, Windows Mobile*). Osvětlit pojmy v oblasti frameworků, a to zejména těch, které se týkají tvorby mobilních aplikací. Představit čtenáři moderní trendy související s frameworky pro vývoj mobilních aplikací a několik vybraných hojně užívaných frameworků blíže specifikovat a poukázat na jejich přednosti či naopak slabé stránky (konkrétně *Xamarin, React Native, Ionic, Framework 7, NativeScript*).

Praktická část má za úkol porovnat dva frameworky pro tvorbu multiplatformních mobilních aplikací (*Xamarin* a *React Native*). Nejprve je implementována vzorová aplikace, na které je porovnání demonstrováno. Podoba a funkce této aplikace jsou součástí bakalářské práce. Dále jsou zmíněna vývojová prostředí, ve kterých byla aplikace vyvíjena a zařízení, na kterých byla testována. Následuje stěžejní kapitola, a to samotné porovnání frameworků. To je rozděleno do třech částí – z hlediska uživatelského rozhraní (vývoj v obou frameworcích a výsledná podoba aplikací v porovnání s nativním designem), z hlediska implementace funkcí (ukázky rozdílné implementace) a z hlediska využití zdrojů případně API třetích stran (správa knihoven, řešení povolení pro přístup ke zdroji, přehled možností frameworků).

2 Teoretická část

Teoretickou část bakalářské práce tvoří 3 kapitoly. První z nich se zabývá mobilními aplikacemi, druhá mobilními operačními systémy a poslední nejobsáhlejší kapitola frameworky se zaměřením na vývoj mobilních aplikací.

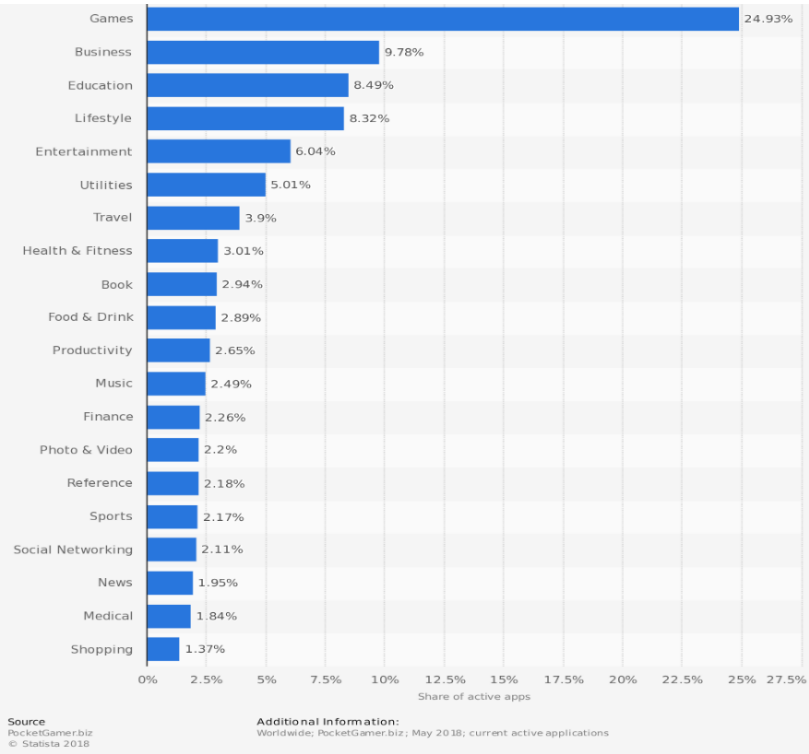
2.1 Mobilní aplikace

Mobilní aplikace je typ aplikačního softwaru vytvořený pro používání na mobilních zařízeních jako jsou mobilní telefony či tablety. Často jsou vytvářeny za účelem poskytnutí služeb, které jsou běžně k dispozici na laptopech a stolních počítačích, a to v takové podobě, aby byly plně kompatibilní na mobilních zařízeních. (2)

Složitost tvorby mobilních aplikací tkví v rozmanitosti mobilních zařízení – různá zařízení mají různá rozlišení a různé velikosti obrazovek, rozdílné ovládaní či dotykové funkce, uživatelské rozhraní a v neposlední řadě hardwarové vlastnosti jako typy procesorů, pamětí a podobně. S tímto problémem souvisí i trend *tvorby responzivních webů,* neboť je nežádoucí, aby část textu nebyla viditelná z důvodu překrytí jednotlivých částí webové stránky kvůli rozdílným rozlišením obrazovek různých zařízení – snaha webových developerů je taková, aby zobrazení webu bylo optimální na všech platformách. Mimochodem i na tuto problematiku existuje celá řada frameworků. (3)

2.1.1 Typy mobilních aplikací z pohledu uživatele

Mobilní nativní aplikace jsou poskytnuty běžnému uživateli skrze tzv. „*app store*“ – jedná se o jakýsi obchod nabízející širokou škálu aplikací různých kategorií. Každý operační systém má svůj vlastní obchod s aplikacemi. Dle serveru statista.com jsou nejpopulárnější kategorií na platformách Apple (po Androidu druhá nejužívanější platforma) mobilní hry s necelými 25 % ze všech aktivních aplikací. (4)



Obrázek 1 - Nejpopulárnější kategorie dle počtu dostupných mobilních aplikací na platformě Apple v květnu 2018, Zdroj: www.statista.com, 2018

2.1.2 Typy mobilních aplikací z pohledu vývojáře

Z pohledu vývojáře lze aplikace používané na mobilních zařízeních rozdělit do 3 kategorií – webové aplikace, hybridní a nativní aplikace. Tato kapitola popisuje základní charakteristiky, výhody a nevýhody či technologie u jednotlivých variant.

Webové aplikace

Webové aplikace jsou aplikace, jež jsou dostupné uživateli pouze skrze webové prohlížeče. Na rozdíl od webových stránek, které jsou čistě informační, nabízejí nějakou funkčnost. Jejich výhodami jsou hardwarová a paměťová nenáročnost, nižší náklady a obvykle menší časová náročnost pro vývojáře. (5) Webové aplikace jsou obvykle psány pomocí *JavaScriptu*, *HTML5* a *CSS* kaskádových stylů určených pro design stránky.

Nativní aplikace

Nativní aplikace tvoří většinu aplikací v mobilních zařízeních. Každá platforma vyžaduje vlastní vývoj aplikace a poskytuje k tomu vývojářům vhodné nástroje – vývojové prostředí (Apple – *Xcode*, Android – *Android Studio*), čímž vývojářům tvorbu zjednodušují. (5)

Největší výhodou nativních aplikací je jejich plná komptabilita se všemi funkcemi daného zařízení včetně dokonalé responzivity. Nevýhodou naopak náročnost tvorby, jak časová, tak finanční, a především velké množství kódu (5) – pokud má být aplikace určena pro více platforem, je nutné mít samostatnou implementaci pro každou platformu. Tento problém vedl ke snaze vytvořit framework, jenž by umožňoval vytvářet aplikace s jedním kusem kódu, které by byly schopné běžet na více platformách zároveň. (5) Příkladem takových to frameworků jsou *Xamarin* či *React Native.*

Hybridní aplikace

Hybridní aplikace jsou kombinací webových a nativních aplikací. Hybridní aplikace se tvoří stejně jako webové aplikace pomocí *HTML, CSS* a *JavaScriptu* a běží na zjednodušeném webovém prohlížeči zvaném *Webview*. Jedná se v podstatě o jakýsi „nativní obal“, který umožní z aplikace vytvořené pomocí webových technologií vytvořit nativní aplikaci. (4)

Tato technika značně ulehčuje vývoj mobilních aplikací – snadný přechod z jedné platformy na druhou, levné a nenáročné zpracování. Největší nevýhodou je nižší výkonnost těchto aplikací. (5) Mezi nejznámější frameworky v tvorbě hybridních aplikací se řadí *PhoneGap, Cordova* či *Framework 7*.

2.1.3 Srovnání mobilních a webových aplikací

Pro vývojáře je vždy otázka, čeho chce svojí aplikací vlastně docílit. Výhod mobilních aplikací je několik. Často jsou schopny fungovat v offline režimu, mají vlastní ikonu na ploše telefonu (uživatel nemusí danou aplikaci vyhledávat na webovém prohlížeči, čímž si ušetří čas) a umožňují zasílat notifikace uživateli. Další jejich důležitou výhodou je umístění v app storech, a tím možnost získání nových potenciálních uživatelů, kteří pak mohou aplikace

sdílet a business rozvíjet – ve výsledku lze říci, že mobilní aplikace mají jistou přidanou hodnotu a bývají více výdělečné. (5)

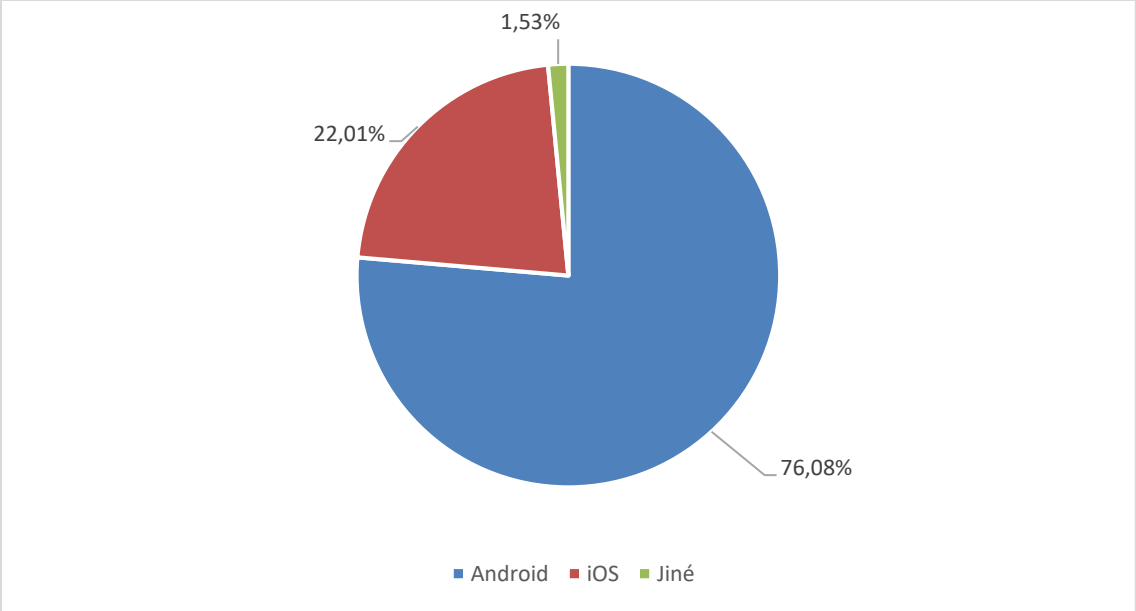
Jejich nevýhodou je však nemožná funkčnost stejné aplikace na různých platformách – ačkoliv se zdají být některé nativní aplikace na různých operačních systémech stejné, jsou vytvářeny zvlášť (5). Tento problém se snaží řešit frameworky pro tvorbu multiplatformních aplikací (viz kapitola *2.3.2 Frameworky pro vývoj multiplatformních aplikací*).

2.1.4 Development „mobile first“

Rychlý vývoj mobilních aplikací a jejich mohutný růst na trhu přinesl nové trendy pro vývojáře aplikací. Jedním z nich je tzv. „*mobile first*“ (3). Jedná se o nový postup tvorby aplikací. V podstatě je to další alternativa k zajištění *responzivity*. V běžném původním přístupu vývojář zpracuje daný software prvně na počítači a až poté řeší zpracování na mobilních zařízeních. Díky vzrůstajícímu vlivu mobilních aplikací se dnes často vyplácí tento postup otočit a nejdříve vytvořit mobilní verzi aplikace a až poté adaptovat software na obsáhlejší specifikace jako jsou počítače. (3)

2.2 Operační systémy

Stejně tak jako u počítačů, i mobilní zařízení potřebují svůj operační systém, tedy software, který zajistí chod zařízení a vytvoří vhodné prostředí pro fungovaní všech ostatních aplikací. V mobilním světě funguje hned několik operačních systému, tři nejznámější a nejpoužívanější jsou *Android OS, iOS* a *Windows Mobile*. (6) Na Obrázku 2 je vyobrazen podíl mobilních operačních systémů na trhu v roce 2019 – do kategorie „Jiné“ je zařazen i Windows Mobile s procentuálním zastoupením 0,2 %.



Obrázek 2 - Podíl na trhu mobilních operačních systémů v roce 2019, Zdroj: http://gs.statcounter.com, 2019.

2.2.1 Android OS

Mobilní operační systém Android je free and open software poskytovaný společností Google. Obsahuje jak samotný operační systém, tak základní klíčové aplikace pro mobilní zařízení. Názvy jednotlivých verzí Androidu se inspirují názvy dezertů (Cupcake, Donut, Ice Cream Sandwich atd.). Android poskytuje vývojářům na této platformě vývojové prostředí *Android Studio*. Co se týče programovacích jazyků, které se pojí s tvorbou nativních aplikací na Androidu, je to jednoznačně *Java*. (6)

2.2.2 iPhone OS / iOS

Operační systém iPhone OS od společnosti Apple byl původně určen pouze pro zařízení iPhone. Nyní se systém označuje jako iOS a slouží hned na několika zařízeních (iPhone, iPod, iPad atd.). Na rozdíl od platformy Android však pouze na zařízeních od společnosti Apple, neboť Apple neposkytuje licenci na iOS ostatním výrobcům. Pro vývojáře nativních aplikací na této platformě je poskytováno vývojové prostředí *Xcode* a tvoří se v jazycích *Objective-C* či *Swift*. (6)

2.2.3 Windows Mobile

Windows mobile je mobilní operační systém od společnosti Microsoft. Je založen na jádru Windows CE 5.2 a obsahuje základní sadu aplikací vyvinutých pomocí rozhraní Microsoft Windows API. Nativními programovacími jazyky této platformy jsou *Visual C++* a *C#*. (6)

2.3 Frameworky pro vývoj mobilních aplikací

Prvně je třeba osvětlit pojem framework, jeho význam pro vývojáře a rozšíření v různých oblastech IT.

2.3.1 Framework

Pojem framework označuje jakousi kostru či strukturu, jež pomáhá v řešení určitého problému. Ve světě IT to znamená určitou programovou strukturu, která vývojářům napomáhá ve vývoji a tvorbě aplikací. (7) Frameworky obsahují *knihovny* (sdílené a znovupoužitelné části kódu), *kompilátory* (překladače programovacího jazyka do strojového kódu), *API* (rozhraní pro programátory) a různé podpůrné programy a nástroje. (8)

Frameworky se skládají ze dvou částí – tzv. „*frozen spots*“ a „*hot spots*“. *Frozen spots* obsahují základní strukturu daného frameworku, to znamená jeho komponenty a vztahy mezi nimi. *Hot spots* pak reprezentují vlastní kód vývojáře psaný za účelem tvorby dané aplikace. (9)

Obecně platí, že frameworky značně ulehčují práci vývojářům. Díky své struktuře umožnují vývojáři soustředit se pouze na požadavky daného projektu nikoliv na opakující se detaily při vývoji (např. při tvorbě webové aplikace se může plně věnovat funkcionalitě, a ne řešit bezchybnou navigaci mezi položkami v menu stránky). Další výhodou je, že jakmile se developer seznámí s frameworkem, jeho produktivita v tvorbě značně stoupá, neboť se spousta kroků stále opakuje. (8)

Existuje značné množství typů různých frameworků. Od Ajax frameworků, přes velice populární frameworky pro tvorbu webových aplikací – zde stojí za zmínku populární architektura *MVC* (10) (oddělení kódu modelu aplikace od pohledů či uživatelského rozhraní, propojené pomocí tzv. controlleru, který řídí veškerou aplikační logiku), manažerské frameworky, multimediální, a především pro tuto práci důležité – frameworky pro tvorbu mobilních aplikací. (8)

Frameworků pro tvorbu mobilních aplikací je ve světě IT značné množství. Podobně tak jako mobilní aplikace i nástroje na jejich tvorbu můžeme rozdělit do 3 skupin – frameworky pro tvorbu nativních aplikací, pro tvorbu multiplatformních aplikací a pro tvorbu mobilních

webových aplikací. (11) Tato práce se dále zaměří na frameworky pro tvorbu multiplatformních mobilních aplikací a přiblíží některé z nich.

2.3.2 Frameworky pro vývoj multiplatformních mobilních aplikací

Jedná se o skupinu frameworků s moderním přístupem tvorby aplikací, kdy jedna aplikace psaná jedním kódem může běžet na několika platformách zároveň. Skupinu můžeme rozdělit do dvou podskupin, které jsou si dost podobné, přesto se mezi nimi dají najít rozdíly. A to frameworky pro tvorbu hybridních aplikací a frameworky pro tvorbu multiplatformních nativních aplikací. Snaha obou těchto podskupin je vytvořit takovou aplikaci, aby se svými možnostmi a kompatibilitou vyrovnala klasickým nativním aplikacím, které jsou specifické pro každý operační systém, a to pomocí jedné jediné aplikace. (12)

Co se týče frameworků pro tvorbu hybridních aplikací, jejich podstatou je využití webových technologií jako jsou *HTML5, CSS* a *JavaScript*. (5) Pomocí nástrojů jako jsou *Cordova* či *PhoneGap* dojde k vytvoření jakéhosi mostu k nativním technologiím daného zařízení. Aplikace pak běží v prostředí *WebView* (13) (jakýsi miniaturní webový prohlížeč), ale využívá nativní uživatelské rozhraní (Více o hybridních aplikacích v kapitole *Hybridní aplikace*). (12)

Frameworky pro tvorbu multiplatformních nativních aplikací fungují na podobné bázi. Aplikace se vytváří pomocí jedné technologie či programovacího jazyka a jednotlivé prvky jsou namapovány na nativní prvky daného zařízení. Komunikace mezi nativními a vytvořenými částmi probíhá pomocí k tomu určených nástrojů – spojovacích mostů. (12) Výsledkem je plně kompatibilní nativní aplikace s využitím všech nativních funkcí. Výhodami těchto aplikací oproti hybridním bývají právě větší možnosti využití nativních funkcí na daném zařízení. (5)

2.3.3 Přehled frameworků pro tvorbu multiplatformních mobilních aplikací

Tato kapitola představuje přehled několika vybraných multiplatformních frameworků a to konkrétně: *Xamarin, React Native, Ionic, Framework 7* a *NativeScript*. První odstavec vždy

představuje obecné informace o frameworku, následuje stručná stavba frameworku, nejdůležitější ovládací prvky, vývojové prostředí, případně nástroje pro tvorbu UI a poslední odstavec je shrnutím daného frameworku (silné či slabé stránky atd.).

Xamarin

Xamarin byl založený v roce 2011 a od roku 2016 je vlastněný společností Microsoft. Jedná se o framework určený pro tvorbu multiplatformních nativních aplikací, a to pro všechny tři hlavní platformy – *iOS*, *Android* a *Windows Mobile*. Framework je založený na programovacím jazyku *C#* a frameworku *.NET*. Vývojovým prostředím (IDE) je *Microsoft Visual Studio,* a to jak pro Microsoft, tak pro Mac, takže ačkoliv je Xamarin vlastněn Microsoftem, i vývojáři pracující na platformě Apple mohou tento framework využít. (14)

Co se týče vývoje pro jednotlivé platformy, nejproblematičtější je vývoj pro iOS, pokud tedy není aplikace vyvíjena na platformě Applu. Existují sice již nástroje, které umožní debugging a kompilaci kódu, ale pro finální sestavení aplikace a její umístění na App store je vždy potřeba vlastnit Mac (s Xcode IDE, které je určené pro tvorbu nativních iOS aplikací), popřípadě se k němu virtuálně připojit přes internet. Vývoj pro Android vyžaduje pouze stažení aktuálních Java a Android SDK. Jelikož je Xamarin od Microsoftu, Windows Mobile v podstatě nevyžaduje kromě Visual Studia nic. (14)

Jádrem Xamarinu je open-source projekt *Mono*, který se skládá z kompilátoru, .NET systému pro *Garbage Collector* (systém, který shromažďuje objekty, na které již neexistuje reference) a základních knihoven. (15) Xamarin poskytuje vývojáři nativní funkce jednotlivých operačních systémů, a to v podobě SDK (sad vývojových nástrojů) daných operačních systémů, ty pak představují *namespaces* v projektu, a tudíž jsou dostupné pomocí referencí v C#. Xamarin tedy poskytuje znovupoužitelné a multiplatformní C# knihovny i nativní knihovny daných platforem. (14)

K tvorbě *frontendu*, tedy uživatelského rozhraní, se používají nativní ovládací prvky, jež jsou k nerozeznání od aplikací psaných v nativních jazycích jako Objective-C nebo Java. Developer má možnost buď využít již definované prvky nebo vytvořit své vlastní. Každá platforma má svůj vlastní systém pro práci s uživatelským rozhraním (iOS Designer pro iOS, Android UI designer pro Android, Microsoft má UI designer přímo ve Visual Studiu,

popřípadě lze použít Blend). K tvorbě designu se používá značkovací jazyk *XAML* odvozený od dobře známého XML. (14)

Xamarin se řadí mezi vůbec nejoblíbenější a nejpoužívanější frameworky pro tvorbu multiplatformních aplikací. Počet developerů od roku 2011 přesáhl 1 milion a stále stoupá. Mezi další nástroje sloužící k podpoře Xamarinu patří například *Xamarin Test Cloud* (sloužící k testování) nebo *Xamarin Profiler* (pro monitoring paměti). (15)

Pro tvorbu uživatelského rozhraní lze též využít framework *Xamarin.Forms*. Na rozdíl od běžného Xamarinu umožňuje ještě větší sdílení kódu. Obsahuje přes 30 komponent, které jsou během chodu aplikace mapovány na nativní komponenty. Jeho využití je především tam, kde se upřednostňuje sdílený kód před důrazem na libovolná uživatelská rozhraní. (16)

React Native

Dalším frameworkem pro multiplatformní tvorbu nativních mobilních aplikací je React Native od společnosti Facebook. Framework byl založen v roce 2013 a od roku 2015 jsou jeho knihovny volně dostupné na platformě *GitHub*. (17) Je určen k tvorbě aplikací pro dvě nejpoužívanější platformy, a to *iOS* a *Android*. Stěžejním programovacím jazykem je *JavaScript*. (18)

React Native vychází z frameworku, respektive knihovny *React*, což je JavaScriptová knihovna pro tvorbu uživatelského rozhraní webu či webových aplikací. Stejně jako React i React Native se skládá z jednotlivých komponent, v tomto případě jsou však ty webové nahrazeny nativními mobilními. React Native používá stejné prvky jako nativní prvky operačních systémů Android a iOS, tudíž podobně jako u Xamarinu výsledná aplikace se velmi blíží tradičním nativním aplikacím. (19)

Tvorba aplikace a jejího UI tedy spočívá ve skládání komponent, které mají určité parametry – *vlastnosti* a nachází se v různých *stavech*. Díky definování stavů lze poměrně snadno vytvářet interaktivní UI, neboť React může efektivně aktualizovat a vykreslovat správné komponenty při změně dat. Pomocí funkce *render* se pak vstupní data přemění na výsledný pohled. Komponenty v Reactu jsou psány pomocí *JSX*, což je rozšíření syntaxe jazyku JavaScript, velice podobné značkovacímu HTML jazyku. (18)

Vývojových prostředí (IDE) pro tvorbu React Native aplikací je poměrně velké množství. Mezi nejznámější open-source IDE patří *Nuclide*, *Atom* či *Visual Studio Code* od Microsoftu. Všechny tyto editory jsou dostupné na všech hlavních operačních systémech – Windows, Mac, Linux. Dalším známým open-source editorem je *Deco IDE*, které je však již určeno pouze pro Mac a iOS. Co se týče IDE s placenou licencí, pak stojí za zmínku *WebStorm*, taktéž dostupný pro všechny tři platformy. (20)

React Native se pyšní především rychlou a poměrně snadnou tvorbou a zároveň dobrou výkonností výsledné aplikace při porovnání s nativními aplikacemi. Od roku 2013 velmi rychle stoupla popularita a rozrostla se komunita dalších a dalších vývojářů, čemuž napomáhá i podpora Facebooku. (19)

Ionic

Ionic je volně dostupný framework pro vývoj hybridních aplikací (*viz kapitola hybridní aplikace*), ačkoliv díky používaným technologiím se poměrně dost blíží nativním aplikacím. Framework byl vytvořen v roce 2013 společností Drifty. Tvorba aplikací spočívá ve využití tradičních webových technologií jako jsou *HTML*, *CSS* a *JavaScript*. V tomto frameworku lze vytvářet aplikace pro iOS, Android i Windows. (21)

Ionic se zaměřuje především na vývoj front-endu, tedy uživatelského rozhraní aplikací, a snaží se ho zejména zjednodušit a zefektivnit. V podstatě se dá říci, že kombinuje prvky ostatních frameworků pro tvorbu hybridních aplikací jako jsou *Cordova* či *PhoneGap* a vytváří nástroj pro tvorbu UI tak, aby se co nejvíce přiblížil UI nativních aplikací, čímž zmenšil mezeru mezi tvorbou v HTML5 a tvorbou nativních aplikací. (22) Pomocníkem pro tvorbu UI je nástroj *Ionic Creator*, který funguje na velmi prostém principu *drag and drop* (táhni a pusť), kde vývojář jednoduše vybere komponentu, vezme ji a umístí na požadované místo. (21)

Základním kamenem frameworku je *Angular* (23), což je open-source framework pro tvorbu front-endu všech možných typů aplikací, založený na programovacím jazyku *TypeScript* (24), jenž je jakousi nadstavbou jazyka JavaScript, používaný při velkém množství JavaScriptového kódu. Ionic předkládá developerům designové vzory nativních aplikací jednotlivých platforem, a to v podobě zabalených TypeScriptových pluginů (zásuvných

modulů) od Cordovy. Především však tyto vzory umožňuje modifikovat, popřípadě vytvářet zcela nové, avšak kompatibilní, a to vše pomocí jednoduchého CSS a JavaScriptu. (21)

K tvorbě Ionic aplikace je zapotřebí mít vývojové prostředí (IDE) podporující JavaScript 6 a TypeScript. Mezi doporučené se, stejně tak jako u React Native, řadí *Visual Studio Code*, *Atom* či placený *WebStorm*. Pakliže má výsledná aplikace směřovat na daný app store některé z mobilních platforem je potřeba ji prvně kompilovat ve frameworku *Cordova* či *PhoneGap*. (21)

Dalším tak trochu speciálním produktem, na který se Ionic zaměřuje, je tzv. *Progresivní webová aplikace*. Jedná se v podstatě o webové aplikace běžící tradičně v prohlížeči, které však díky novým web API (rozhraní pro programování aplikací pro webový prohlížeč nebo server) a technologiím dokážou vykonávat některé funkce, které jsou specifické pro nativní aplikace (např. zasílání notifikací nebo zobrazení na ploše). (25)

Největší předností frameworku Ionic je tvorba UI pomocí webových technologií, neboť HTML5 přináší zcela nové možnosti ve vytváření designových prvků a pro webové developery se tím otevírají možnosti tvorby mobilních aplikací bez nutnosti učit se nativní jazyky. (22)

NativeScript

NativeScript od společnosti Progress se řadí spíše do skupiny frameworků pro tvorbu multiplatformních nativních aplikací podobně jako zmínění Xamarin a React Native. Na trhu se objevil poprvé v roce 2014. Specializuje se na tvorbu aplikací především pro Android a iOS, popřípadě i UWP (26) (Univerzální platforma Windows – stejná aplikace je spustitelná na jakémkoliv zařízení s operačním systémem Windows 10). Základním vývojovým nástrojem je programovací jazyk *JavaScript*, případně odvozený *TypeScript*. (27)

Na rozdíl od spousty frameworků pro tvorbu multiplatformních aplikací, NativeScript nevytváří žádný obal kolem nativních funkcí, ale díky svému běhovému prostředí (v angličtině zkráceně *runtime*) je schopný komunikovat s nativními prvky přímo pomocí JavaScriptu. V podstatě nalezne nativní API, na kterou se adresuje, přeloží požadavek psaný v JavaScriptu do nativního jazyka (Java pro Android, Objective-C či Swift pro iOS) a zavolá nativní API vracející požadovaný výsledek. O to, aby se developer nemusel starat o nativní

kód a mohl pracovat pouze s JavaScriptem se starají modulová jádra, jež jsou další součástí frameworku NativeScript a slouží k abstrakci spojení s nativním kódem. (28)

NativeScript může odkazovat na všechny možné nativní API podporovaných platforem, včetně senzorů a knihoven třetích stran psaných v jazycích Java, Objective-C nebo na .NET platformě. Uživatelské rozhraní tvoří tradičně nativní UI komponenty a události jsou zpracovávány nativními obsluhovateli (např. View.OnClickListener), které jsou deklarované v jazyku JavaScript. (28) Pro tvorbu UI lze využít i k tomu určené frameworky *Angular*, o kterém již byla řeč, či *Vue.js*, což je další open-source front-end framework založený na JavaScriptu. (27) NativeScript podporuje i tvorbu v kaskádových stylech a architekturu *MVVM* (28) – kdy je oddělen front-end a back-end, tedy tvorba UI od modelu či aplikační logiky.

Vývoj aplikace může probíhat v podstatě v jakémkoliv IDE podporující tvorbu mobilních aplikací. Přesto nejlepší volbou je *Visual Studio Code*, neboť NativeScript pro něj připravil obsáhlé rozšíření s rozsáhlým Intellisense (inteligentní nápověda pro psaní kódu), interaktivním debugging systémem a efektivní integrací s emulátory pro testování aplikací. Pro tvorbu aplikací pro iOS a Android lze použít také *WebStorm*, pro něhož též existuje NativeScript plugin. (29)

Díky odlišné technologii tvorby multiplatformních aplikací se NativeScript aplikace pyšní velice dobrou výkonností, v porovnání s nativními aplikacemi je ztráta výkonu způsobena běhovým prostředím frameworku pouhých 10 %. (28) Updaty vychází pravidelně každý 6.-8. týden a komunita je, jak na serveru Stack Overflow (pravděpodobně největší komunitní fórum pro vývojáře), tak na službě GitHub, velice rozsáhlá. (27)

Framework 7

Jedná se o framework vytvořený společností iDangero.us v roce 2016, jehož autorem je Vladimir Kharlampidi. Slouží k tvorbě webových a mobilních hybridních aplikací s podporou nativních prvků iOS a Android. K tvorbě slouží pouze webové technologie především *HTML* a *CSS*, částečně také *JavaScript*, případně *jQuery*, jakožto JavaScriptová knihovna. (30)

Framework 7 se zaměřuje na front-end aplikací. V podstatě se jedná o open-source set knihoven a návrhů různých řešení UI aplikací, které pomocí jednoduchého HTML a CSS

vývojář může všemožně upravovat dle svých vlastních představ. Jak již bylo zmíněno, framework nabízí podporu pro platformy iOS a Android v podobě velkého množství UI elementů pro obě platformy, a to opravdu do veliké míry. Přístupné jsou funkce typické pro nativní aplikace jako například dynamické navigační lišty nebo realistické „swipe back“ akce, tedy přejetí zpět na předešlý pohled pomocí tažení obrazovky. Framework také nabízí velkou míru svobody, jelikož jednotlivé prvky lze velice snadno modifikovat, neboť jsou ukládány a rozděleny do malých souborů s příponou *.less*. (30)

Framework 7 se neopírá o žádný jiný framework či nějakou jeho součást, nýbrž pouze o svůj vlastní DOM systém (Document Object Model – rozhraní, které umožňuje přistupovat, updatovat a měnit obsah a strukturu XML a HTML dokumentů). (31) Tento systém je nazvaný *DOM 7* a je založený na principech klasické jQuery knihovny, jako jsou nejznámější a nejpoužívanější jQuery metody nebo možnost jejich řetězení. Účelem DOM 7 systému je co nejjednodušší použití. Zároveň však lze tento framework využívat i v kooperaci s jinými dobře známými a již zmíněnými frameworky pro tvorbu UI jako *React.js, Vue.js* či *Angular*. (30)

Od tvůrců Frameworku 7 není doporučeno žádné vývojové prostředí, ale mělo by být možné využít v podstatě jakékoliv s podporou HTML, CSS a JavaScriptu. (30)

Závěrem lze říci, že Framework 7 přináší na trh jednu z nejjednodušších variant tvorby UI hybridních aplikací. Poskytuje velké množství ukázkových aplikací, které lze různě upravovat, i poměrně obsáhlou dokumentaci s velkým množstvím nativních UI komponent. Další zajímavou výhodou je spolupráce s platformou *vi*, díky níž lze vytvářet reklamní videa propagující aplikace umístěné v app storech. Slabou stránkou je menší komunita, a především nedostatečná podpora od tvůrců frameworku při řešení problémů. (30)

3 Praktická část

Praktická část bakalářské práce se zaobírá porovnáním dvou ze třech nejpoužívanějších frameworků pro vývoj multiplatformních aplikací – Xamarinu a React Nativu.

Za tímto účelem byla navržena vzorová aplikace, na které je porovnání vývoje demonstrováno. Jako cílový operační systém, pro který byla aplikace vyvíjena a následně testována, byl zvolen Android, a to z několika důvodů: Zaprvé se jedná o celosvětově nejpoužívanější operační systém pro mobilní zařízení (viz kapitola *2.2 Operační systémy*), zadruhé vývoj a testování může být u některých operačních systému komplikovanější – nutnost pracovat s macOS při vývoji pro iOS v Xamarinu (viz kapitola *Xamarin*) a v neposlední řadě by se značně komplikovalo samotné porovnání (především vzhledem k množství funkcionalit vzorové aplikace).

Nejprve byl navržen design aplikace, a to v Android studiu, tedy frameworku pro vývoj nativních aplikací pro operační systém Android. Následně byla vzorová aplikace implementována v obou frameworcích. Následovalo testování aplikace jak na fyzickém zařízení – *Huawei FIG-LX1 (verze systému Android – 8.0.0)*, tak v Android emulátoru – zde bylo vybráno virtuální zařízení *Nexus 5 (verze systému Android – 8.1.0)*.

První kapitola praktické části se věnuje vzorové aplikaci, jejímu popisu a funkcím. Dále se práce věnuje zvoleným nástrojům pro vývoj vzorové aplikace a zařízením, na kterých byla aplikace testována. Poté již následuje samotné porovnání frameworků z pohledu vývoje vzorové aplikace, které je rozděleno do třech částí – porovnání z hlediska uživatelského rozhraní, implementace funkcí a využití zdrojů testovaných mobilních zařízení.

3.1 Vzorová aplikace

Vzorová aplikace byla navržena tak, aby její funkce byly co nejrozmanitější, tedy aby využívala co nejvíce zdrojů mobilního zařízení, obsahovala alespoň jednu složitější GUI nativní komponentu a pracovala s různými API třetích stran. Zároveň byla snaha o to, aby aplikace byla smysluplná a měla své využití na trhu.

3.1.1 Popis aplikace

Aplikace představuje jakousi smart home aplikaci, respektive nabízí několik funkcí pro manipulaci se smart zařízeními typu smart žárovek či smart LED pásek. Implementovaná aplikace se abstrahuje od propojení se smart zařízeními, avšak nabízí jednotlivé funkce s možností aktivace či deaktivace příslušné funkce a zobrazuje údaje, které se k příslušné funkci vážou (například hodnota ambientního osvětlení u funkce světelného senzoru atd.). Aplikace je tedy vytvořena tak, aby bylo zřejmé, zda daná funkcionalita funguje přesně tak, jak má a jednotlivé obrazy jsou tomu tak přizpůsobeny (v reálném využití by se konkrétní hodnoty ze senzorů nezobrazovaly, stejně tak gps funkce, viz kapitola *3.1.2 Funkce aplikace*, a s tím spojená ukázka Google Maps API by neumožňovala neustále měnit zvolenou polohu domova). Nicméně i přes kladený důraz na demonstraci využití funkcí a zdrojů mobilního zařízení, obsahuje aplikace reálně použitelné atributy pro ukládaní informací, sloužící ke komunikaci se smart zařízením (zpravidla Booleovské hodnoty true/false pro rozsvícení, změnu barvy atd.).

3.1.2 Funkce aplikace

Tato kapitola se věnuje veškerým funkcím vzorové aplikace. Součástí popisu každé funkce je vizuální podoba funkce vytvořená v Android Studiu představující nativní uživatelské rozhraní pro Android. Po zapnutí aplikace je uživatel přesměrován na hlavní stránku, jež obsahuje carousel se screeny všech funkcí aplikace. Následně si v menu může vybrat příslušnou funkci. Aplikace je rozdělena na čtyři hlavní funkcionality.

První funkcí je funkce snímání ambientního osvětlení. Ta funguje tak, že jakmile je aktivována prostřednictvím switch buttonu, spustí se světelný sensor mobilního zařízení a na obrazovku se vypíše údaj ambientního osvětlení dle dokumentace (32) v jednotkách lux. Pomocí switch buttonu lze funkci také deaktivovat. Funkce má v praxi sloužit k regulaci intenzity osvětlení, které je produkováno smart zařízením (čím vyšší je hodnota světla

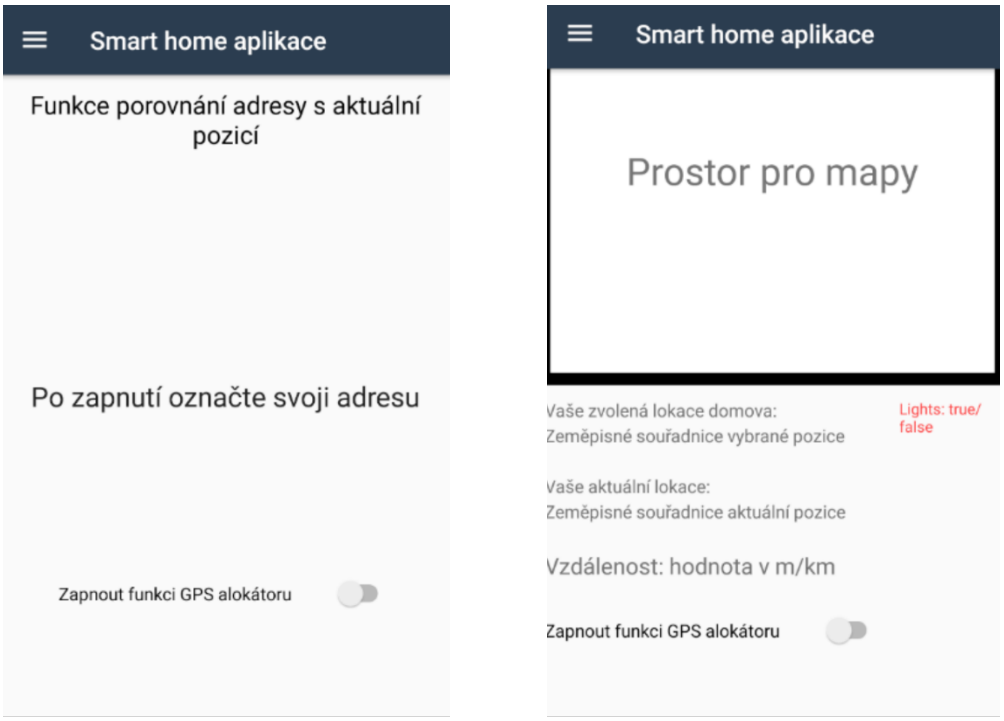
snímaná senzorem mobilního zařízení tím více je snižována intenzita osvětlení a naopak) Součástí této funkcionality je i ukázka práce s relačním databázovým systémem SQLite, jenž je vystavěn přímo v operačním systému Android a je ideální volbu pro ukládání dat aplikace. Do databáze se ukládá jak hodnota světla, tak zda je funkce zapnutá či nikoliv. (viz Obrázek 3)

Obrázek 3 - vizuální podoba funkce snímání ambientního osvětlení

Druhou funkcí je snímání polohy zařízení, a to prostřednictvím senzoru, kterému se říká gyroskop. Gyroskop poskytuje informace o poloze zařízení v podobě úhlové rychlosti náklonů okolo os X, Y a Z mobilního zařízení. Hodnota rotace je kladná v protisměru hodinových ručiček a po směru naopak záporná. Hodnoty jsou udávány v radiánech za sekundu. (32) Funkce má sloužit k přepínání barev u smart zařízení při prudkém pohybu mobilního zařízení po ose X (pokud hodnota na ose X přesáhne hodnotu 1 nebo -1 dochází k aktivaci změny barvy – změna hodnoty na true). Aktivace snímání polohy je obdobná jako v případě světelného senzoru. Po aktivaci se zobrazí údaje z gyroskopu a zda má dojít ke změně barvy u smart zařízení či nikoliv. (viz Obrázek 4)

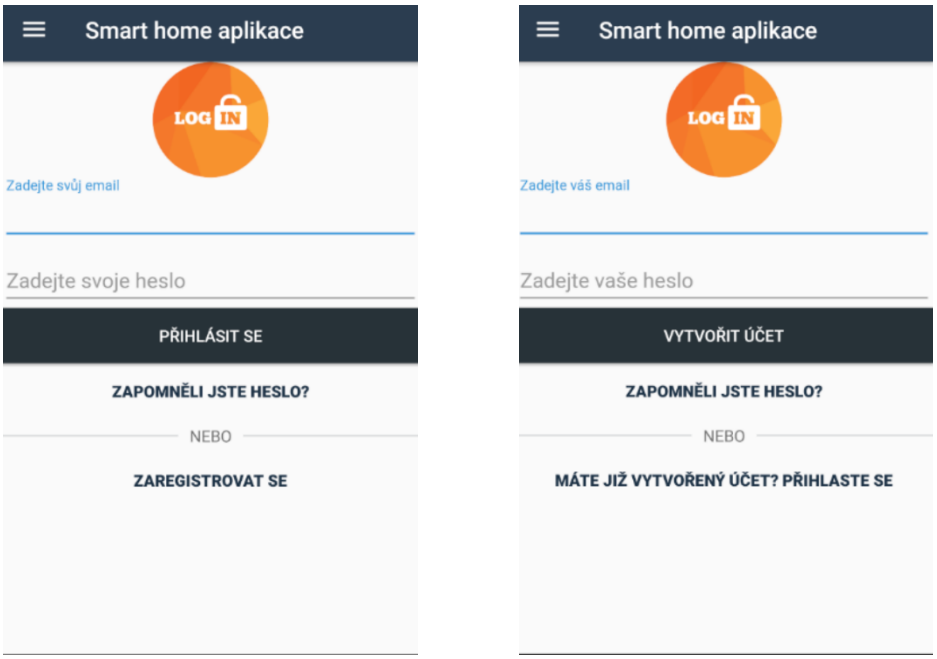
Obrázek 4 - Vizuální podoba funkce snímání polohy zařízení

Třetí funkcí je porovnání aktuální pozice s pozicí domova. Tato funkce kombinuje dvě API, a to Google Maps API pro zobrazení map a následnou práci s ní a Location API pro zaměření pozice mobilního zařízení pro Android. Dále jsou použity různé knihovny pro výpočet vzdálenosti – GeoCoordinate u Xamarinu a geolib u React Nativu. Účelem funkce je porovnání aktuální pozice zařízení se zvolenou pozicí na mapě, jenž má představovat lokaci domova uživatele. Pokud je vzdálenost mezi těmito pozicemi menší než 100 metrů, smart zařízení je aktivováno, a naopak pokud je vzdálenost větší než 100 metrů, je zařízení deaktivováno. Jakmile uživatel aktivuje funkci, zobrazí se mu v horní polovině obrazovky mapa s vyznačeným místem aktuální pozice prostřednictvím markeru (toto místo je zároveň středem mapy). Uživatel poté může na mapě označit další bod, představující jeho zvolenou pozici domova. Následně dojde k výpočtu vzdálenosti mezi těmito body a nastavení atributu stavu světla (vypnuto/zapnuto). Všechny tyto údaje se uživateli zobrazují ve spodní polovině obrazovky, tedy souřadnice aktuální a vybrané pozice, vzdálenost v metrech a hodnota atributu stavu světla. (viz Obrázek 5)

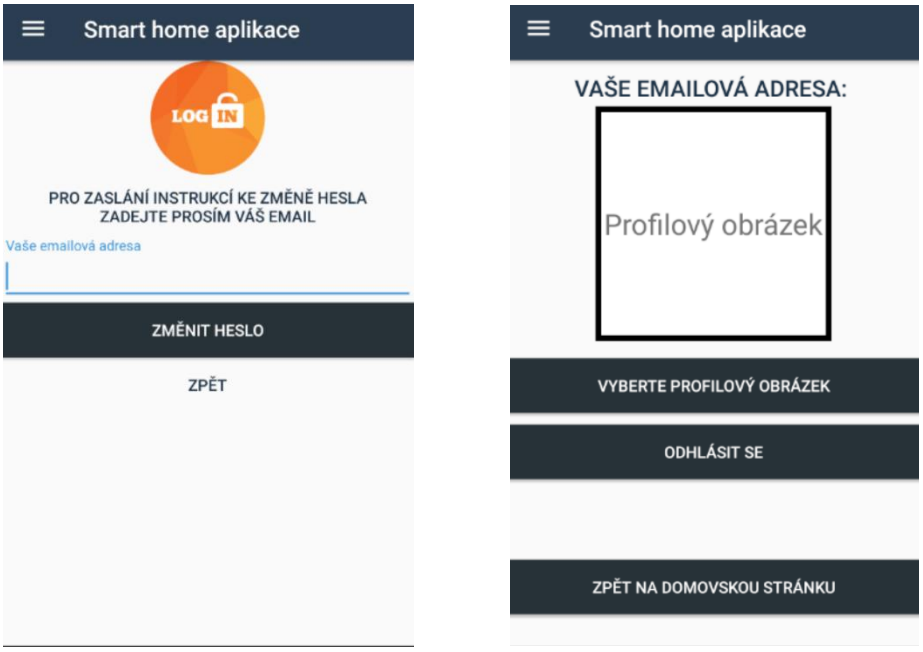


Obrázek 5 - vizuální podobna funkce porovnání adresy s aktuální pozicí před zapnutím (vlevo) a po zapnutí (vpravo)

Poslední funkce je komplexnější. Jedná se o autentifikace s využitím služby Firebase, konkrétně Firebase Auth, jež je určena pro autentifikaci. Dále je demonstrována práce s uložištěm, konkrétně s galerií, a s fotoaparátem. Autentifikace je do aplikace přidána z toho důvodu, aby uživatel mohl po přihlášení využít své nastavení na různých zařízeních. Funkce představuje uživatelův profil, tedy jestliže uživatel není přihlášený je po kliknutí na funkci profilu přesměrován na přihlašovací obrazovku. Zde má několik možností. Krom přihlášení se může odkázat na registraci a založit si zde nový účet. Uživatel se registruje a přihlašuje prostřednictvím emailové adresy a hesla. (viz Obrázek 6) Další možností je zaslání zapomenutého hesla na zadaný email – na emailu se pak objeví instrukce o změně hesla k příslušnému uživatelskému účtu. V případě že je uživatel přihlášený, je přesměrován na svoji profilovou stránku, kde má možnost vybrat si profilový obrázek z galerie či pořídit nový snímek za využití fotoaparátu (obrázek je možné volit vícekrát, neboť se přepisuje a zůstává tak pouze aktuální). Po úspěšném vybrání je příslušný obrázek přidán k aktuálně přihlášenému uživateli přímo do Firebase Auth, tudíž i po odhlášení či novém spuštění na jakémkoliv zařízení mu zvolený obrázek zůstává. Poslední možností je samotné odhlášení uživatele. (viz Obrázek 7)



Obrázek 6 - přihlašovací stránka (vlevo) a stránka pro registraci nového uživatele (vpravo)



Obrázek 7 - vizuální podoba funkce pro zapomenuté heslo (vlevo) a profil uživatele (vpravo)

3.2 Zvolená vývojová prostředí

Pro vývoj aplikace v Xamarinu bylo zvoleno *Microsoft Visual Studio*, jakožto jediná možná volba při vývoji na operačním systému Windows. U React Nativu si lze vybrat v podstatě jakékoliv IDE. Zvoleno bylo vývojové prostředí *Intellij IDEA* od společnosti JetBrains.

3.2.1 Microsoft Visual Studio

Visual Studio je vývojové prostředí od společnosti Microsoft, poskytující sadu nástrojů pro vývoj různých druhů aplikací na různých platformách, a to například Microsoft, Windows Mobile, Microsoft Silverlight a především .NET. Právě na platformě .NET je možné využít nástrojů pro vývoj nativních mobilních multiplatformních aplikací – tato sada nese název *Visual Studio Tools for Xamarin*. (33) Visual Studio nabízí nástroje pro vývoj, ladění – debugging, testování, spolupráci a rozšíření. Pro vývoj nabízí chytrý *IntelliSense* – nástroj pro zobrazení informací o částech kódu, kdy tyto informace se zobrazují přímo při psaní kódu. Dále *Code cleanup* pro formátování, případně opravení chyb před tím, než se kód dostane do fáze kontroly před samotnou kompilací. A v neposlední řadě *Refactoring*, jenž umožňuje přejmenovat proměnné, měnit pořadí parametrů metod či vyčlenit část kódu metody do metody nové. (34) Pro tvorbu uživatelského rozhraní je možné krom editoru využít také *XAML Designer*, kde je možné naformátovat jednotlivé elementy GUI (viz kapitola *Vývoj UI v Xamarinu*). Co se týče ladění, Virtual Studio nabízí debug pro různé programovací jazyky, a to bez ohledu na to, kde aplikace běží – tedy i na emulátoru Androidu (33), což je ideální pro účely této práce. Pro testování výkonu aplikace či hledání chyb na úrovni hardwaru daného zařízení jsou k dispozici *Diagnostic tools* (33), kde je k dispozici využití paměti a procesoru, dále registry či zpětný překlad kompilace. Silnou stránkou Visual Studio je možné rozšíření prostřednictvím *Virtual Studio SDK* (34). Virtual Studio SDK umožňuje rozšíření nástrojů a funkcí Virtual Studia v podobě modulů - tzv. *VSPackages*. Nejnovější verzí IDE je Visual Studio 2019. K bakalářské práci bylo používáno Visual Studio 2017.

3.2.2 Intellij IDEA

Stejně jako Visual Studio tak i Intellij IDEA je vývojové prostředí, které poskytuje širokou škálu nástrojů pro různé typy aplikací na různých platformách. Součástí IDE jsou nástroje pro vývoj v řadě frameworků (např. Spring, Angular či React), z frameworků pro

tvorbu mobilních a hybridních aplikací pak nativní Android Studio, Ionic či PhoneGap. Jednotlivé části zdrojového kódu jsou indexovány a dochází k analýze dat, díky čemuž je automatická komplementace a refaktoring kódu mnohem efektivnější (na úkor větší výpočetní náročnosti při zpracování kódu). (35) Editor detekuje i duplikace a potenciální možné chyby přímo při psaní kódu. Ačkoliv je Intellij IDEA určen převážně pro programovací jazyk Java, obsahuje IntelliSense i pro některé další jazyky včetně JavaScriptu (vhodné pro práci v React Native). Intellij nabízí také nástroje pro většinu frameworků určených k testování (JUnit, TestNG atd.) a podporuje většinu nástrojů pro sestavení aplikace (35) (pro tuto práci je důležitý *Gradle* pro sestavení aplikací na operačním systému Android). Součástí IDE je systémová řádka, tudíž při psaní příkazů není nutné IDE opouštět (v React Nativu velmi častá záležitost). I přes velké množství nástrojů integrovaných přímo v IDE, je možné přidávat další rozšíření, a to prostřednictvím Intellij IDEA Ultimate Plugins, kde se nachází mimo jiné plugin pro Node.js, bez něhož se při vývoji aplikace v React Nativu nelze obejít, neboť prostřednictvím systému balíčků *npm* umožní nainstalovat rozhraní příkazové řádky React Nativu tzv. *React Native CLI,* což je základní nástroj pro práci v tomto frameworku. Intellij IDEA je k dispozici ve dvou verzích – základní free verze nazvaná Community a rozšířená placená verze Ultimate (například s podporou JavaScriptu a většiny frameworků). Pro vývoj vzorové aplikace ve frameworku React Native byla využita rozšířená verze Ultimate se školní licencí.

3.3 Testovací zařízení

Pro testovaní vzorové aplikace byla použita dvě fyzická zařízení (starší Huawei SCL-L21 a novější Huawei FIG-LX1) a jedno virtuální zařízení (Nexus 5) zprostředkované pomocí Android emulátoru. V Tabulce 1 je přehled všech testovacích zařízení s jejich základními parametry a informacemi důležitými pro samotné testování.

Tabulka 1 - Charakteristika testovacích zařízení

Typ zařízení	Značka	Model	Procesor	Verze Android OS	Rozlišení	Podpora senzorů	Podpora gps
Fyzické	Huawei	SCL-L21	32 bit	5.1.1 (API 22)	1280x720	Pouze světelný, nikoliv gyroskop	Ano
Fyzické	Huawei	FIG-LX1	64 bit	8.0.0 (API 26)	2160x1080	Ano	Ano
Virtuální	Nexus	Nexus 5	32 bit	8.1.0 (API 27)	1920x1080	Ano	Ano

Jak lze vyčíst z Tabulky 1 u zařízení Huawei SCL-L21 bylo v průběhu testování zjištěno a následně ověřeno (36), že nepodporuje gyroskopický senzor (jako pohybový senzor využívá pouze akcelometr a kompas). U zařízení Huawei FIG-LX1 nastal problém při testování aplikace vytvořené v React Nativu, neboť zařízení má 64bitový procesor na což nebyla aplikace konfigurována, respektive některé služby Google požadovali konfiguraci pro 64bitovou verzi, která nebyla k dispozici. Problém byl vyřešen úpravou souboru build.gradle, tedy konfiguračního souboru sestavení aplikace pro operační systém Android. Do souboru pod položku defaultConfig byl přidán následující kód:

```
ndk {
    abiFilters "armeabi-v7a", "x86"
}
```

Kód 1 - React Native – nastavení konfiguračního souboru při testování na 64bit zařízení.

Kód 1 způsobil, že službám, které požadovali 64bitovou verzi nativního kódu byl předán soubor pro 32bitovou verzi. V pozdějších verzích React Nativu je tento problém již

vyřešen. U virtuálního zařízení bylo potřeba doinstalovat a následně aktualizovat Google Play Store a to kvůli službám využívající Google Play Services (Google Maps a Firebase). K dalším problémům při testování již nedošlo.

3.3.1 Android emulátor

Jako Android emulátor pro testování aplikace byl využit emulátor nativního Android Studia. Emulátor slouží k simulaci reálného fyzického mobilního zařízení, které je nahrazeno virtuálním zařízením, jež se chová obdobně jako toto fyzické zařízení. Pro použití emulátoru je nutné mít nainstalované Android Studio, kde lze prostřednictvím manažeru SDK nástrojů nainstalovat emulátor. Jelikož je emulátor poměrně náročný na výkon zařízení, na kterém běží, je vhodné využít hardwarové akcelerace. Ta je použita, pokud je na daném zařízení k dispozici a pokud je u virtuálního zařízení použit system image založený na architektuře x86. Každé virtuální zařízení je označeno jako *Android virtual device (AVD)* a je specifikováno typem zařízení a verzí operačního systému – ta představuje tzv. *system image* (lze použít různé system image u různých zařízení). Typem zařízení může být mobilní telefon, tablet, chytré hodinky nebo chytrá televize. Hardwarová specifikace je dána zvoleným typem zařízení, nicméně některé parametry lze modifikovat (například rozlišení obrazovky nebo aktivní senzory zařízení). Možnosti testování jsou v podstatě totožné s fyzickým zařízením. Lze měnit gps polohu zařízení, hodnoty snímané senzory (u gyroskopického senzoru pomocí naklánění zařízení) či pracovat s uložištěm. Po instalaci Google Play lze i pracovat se službami využívajících Google Play Services (v případě vzorové aplikace Google Maps a Firebase).

Android virtual devices (AVDs) jsou spravována nástrojem, který se nazývá *AVD Manager*. Ten slouží k vytváření, editaci, spouštění a mazání virtuálních zařízení. React Native spouští aplikace v emulátoru přes AVD Manager od Android Studia. Je možné k němu přistupovat buď pomocí příkazové řádky nebo v grafické podobě prostřednictvím Android Studia. Xamarin má k dispozici vlastní nástroj pro správu virtuálních zařízení tzv. *Android Device Manager*. Přes drobné změny v uživatelském rozhraní je práce s ním totožná jako v AVD Manageru Android Studia.

3.4 Porovnání frameworků na základě vzorové aplikace

V této kapitole jsou porovnány frameworky React Native a Xamarin z pohledu vývoje vzorové aplikace. Porovnání je rozděleno na tři části: Porovnání z hlediska uživatelského rozhraní, implementace funkcí aplikace a využití zdrojů a API třetích stran a práce s nimi.

3.4.1 Porovnání z hlediska uživatelského rozhraní

Vývoj uživatelského rozhraní (UI) je u obou frameworků značně odlišný. Nicméně oba frameworky obsahují velké množství základních nativních komponent a oba též umožňují nativní komponenty vytvářet a upravovat.

Vývoj UI v Xamarinu

Při vývoji uživatelského rozhraní v Xamarinu má vývojář dvě možnosti – pracovat s knihovnou *Xamarin.Android* a vyvíjet tedy UI čistě s nativními prvky operačního systému Android (tato volba je vhodná v případě, kdy v aplikaci je využíváno velké množství nativních UI prvků nebo aplikace využívá velké množství nativních API), nebo s knihovnou *Xamarin.Forms*, která umožňuje vytvářet vlastní komponenty, které se nativním komponentám do značné míry podobají a jsou kompatibilní na obou operačních systémech. (vhodná v případě kdy aplikace není komplikovaná a nepotřebuje využívat velké množství nativních prvků). Jednoznačnou výhodou Xamarin.Forms je použitelnost jednoho kódu UI na více platformách. Obě varianty mají své výhody a nevýhody, nicméně jako standart se považuje nativní Xamarin, tedy Xamarin.IOS pro IOS a Xamarin.Android pro Android. Tato možnost je méně paměťově náročná, více kompatibilní se zařízeními a vykazuje lepší výkonnost, neboť přistupuje přímo k nativním SDK. (37) Vzorová aplikace byla vyvíjena v Xamarin.Android z důvodu využívaní velkého množství nativního API a z důvodu, že je tento přístup považován jako standartní.

Při vytvoření nové aplikace v Xamarinu se vytvoří adresářová struktura podobná struktuře v Androidu Studiu. Součástí této struktury je adresář *Resources*, který obsahuje veškeré stylování, ikony, obrázky, tedy vše, co vytváří v aplikaci design. Součástí tohoto adresáře je složka *layouts*, ve které se nachází všechny zobrazované stránky. Každá zobrazovaná stránka, respektive součást zobrazované stránky, se vytváří zvlášť v souborech s příponou *.xaml*. K tvorbě UI se v Xamarinu používá značkovací jazyk *XAML* (Extensible

Application Markup Language), odvozený od známého XML. Na následujícím obrázku je definována zobrazovaná stránka představují hlavní stránku aplikace.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/textHome"
        android:layout_centerHorizontal="true"
        android:text="Vítejte v smart home aplikaci"
        android:textSize="20dp"
        android:textColor="@color/colorPrimaryDark"
        android:layout_margin="20dp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

    <RelativeLayout
        android:layout_below="@+id/textHome"
        android:background="@android:color/black"
        android:layout_width="335dp"
        android:layout_height="425dp"
        android:layout_centerHorizontal="true"
        android:padding="10dp"
        >

    <android.support.v4.view.ViewPager
        android:layout_below="@+id/textHome"
        android:id="@+id/viewPager"
        android:layout_width="330dp"
        android:layout_height="420dp"
        android:layout_centerHorizontal="true"
    />
    </RelativeLayout>
```

Obrázek 8 - Definice layoutu hlavní stránky aplikace v XAML

Každá stránka má definované rozložení komponent, které se v ni nachází, a to v podobě elementu *Layout* – k dispozici je celkem 8 typů layoutů. (38) Layouty se do sebe mohou různě vnořovat. Následně se do layoutu přidávají nativní komponenty Androidu. Každá komponenta má své atributy – z Obrázku 8 lze vidět, že se používají nativní atributy prostřednictvím knihovny Xamarin.Android. Definice takové komponenty je pak totožná s definicí komponenty v Android Studiu. Atributy formují design a umístění komponenty v layoutu. Některé často využívané hodnoty atributů jsou uloženy v dalších složkách v adresáři Resources a přístupné pomocí symbolu „@“. Tento přístup je také totožný s tvorbou v Android Studiu. Aby bylo možné s komponentou pracovat v aplikační logice, je nutné definovat její id (klasická forma zápisu id je: „@+id/názevId“). Ke komponentě je přistupováno následovně:

```
// vytvoření instance pro komponentu
private ViewPager viewPager;

// inicializace instance
viewPager = View.FindViewById<ViewPager>(Resource.Id.viewPager);
```

```
// práce s komponentou prostřednictvím instance
viewPager.CurrentItem = 0;
```
Kód 2 - Xamarin – ukázka přístupu ke GUI komponentě

Ke komponentám se přistupuje pomocí metody *FindViewById<typ komponenty>(id komponenty)* – pomocí instance *viewPager* nyní lze s komponentou pracovat. Pokud se nacházíme ve Fragmentu a nikoliv Aktivitě[1] je potřeba metodu volat na kořenovém pohledu (pohledu aktivity, jejíž je součástí) příslušného fragmentu. Každá komponenta má tedy definovanou svoji třídu s příslušnými metodami a vlastnostmi k její obsluze. Obecně lze říci, že tvorba UI a přístup ke komponentám je velmi podobný s tvorbou a přístupem v Android Studiu.

Pro uživatele preferující tvorbu designu v grafické formě je k dispozici *Xamarin.Android Designer*. V Designeru je možné zvolit zařízení s různým rozlišením obrazovky, na němž vývojář chce UI testovat. Dále tento nástroj obsahuje panel nástrojů, který umožňuje tvorbu UI v podobě přesouvání komponent z panelu. Panel nástrojů obsahuje seznam všech nativních komponent. Designer je možné využít i při vývoji UI prostřednictvím kódu, neboť lze vidět aktuální stav vytvářeného obrazu.

Vývoj UI v React Nativu

Jak již bylo zmíněno (kapitola *React Native*), React Native je založený na frameworku React, který je rozšířený o nativní prvky mobilních platforem, jež jsou součástí knihovny *react-native*. V React Nativu není oddělené uživatelské rozhraní od aplikační logiky tak jako v Xamarinu, nýbrž společně tvoří React komponenty. Neřeší se tak žádné propojení mezi vrstvami aplikační logiky a uživatelského rozhraní.

Základem při vývoji v React Native je tedy tzv. React komponenta, ze kterých je celá aplikace vytvořena. Existují dva typy – *Class Component* a *Functional* (někdy také *Stateless*) *Component*. (39) Jednodušší variantou je Functional Component, která slouží pouze k vykreslení určité částí UI. Není schopna udržovat žádný vnitřní stav ani předávat informace. Pouze informace přijímá od Class komponenty a vykresluje. Class komponenta je klasická forma komponenty, která dědí od React.Component, což ji poskytuje možnosti k řízení svého životního cyklu. Tento typ komponenty totiž má svůj vnitřní stav tzv. *state*, který se v průběhu života komponenty mění. Ke komunikaci mezi komponenty jakožto předávání

[1] Stránky aplikace vytvořené v Xamarinu jsou tvořeny buď Aktivitami nebo Fragmenty. Ve vzorové aplikaci je použit klasický vzor, kdy každá stránka je tvořena fragmentem, které jsou součástí jedné hlavní aktivity.

informací se používají tzv. *props*. K řízení změn v průběhu života komponenty slouží několik metod, respektive funkcí, z nichž dvě nejdůležitější jsou: *componentDidMount()* (volaná při načtení komponenty) a *componentWillUnmount()* (volaná naopak při zrušení zobrazování komponenty). Nicméně stěžejní a jedinou povinnou metodou, kterou musí React komponenta obsahovat je metoda *render()*. Ta slouží k samotnému vykreslení obsahu příslušné komponenty. K zápisu obsahu vykreslení se používá syntaxe podobná HTML nazvaná *JSX*. Veškeré uživatelské rozhraní je tedy definováno v metodě render() a to pomocí rozšířené syntaxe jazyka JavaScript JSX. V Kódu 3 je ukázka definice UI pro hlavní stránku vzorové aplikace.

```
   render() {
        return (
            <View>
                <View style={styles.container}>
                <Text style={styles.welcome}>Vítejte v smart home aplikaci</Text>
                    <View>
                    <Carousel width={330} height={420} delay={2000}
                              indicatorSize={20} indicatorText="*">
                        <View style={styles.contentContainer}>
                            <Image source={{ uri: 'asset:/lightscreen.png'}}/>
                        </View>
                        <View style={styles.contentContainer}>
                            <Image source={{ uri: 'asset:/gyroscopescreen.png'}}/>
                        </View>
                        <View style={styles.contentContainer}>
                            <Image source={{ uri: 'asset:/gpsscreen.png'}}/>
                        </View>
                    </Carousel>
                    </View>
                </View>
            </View>
        );
    }
}
```

Kód 3 - Definice layoutu hlavní stránky aplikace v JSX

Z Kódu 3 je patrné, že k vykreslení jsou použity nativní GUI komponenty, zahrnuté v knihovně react-native. Rozvržení layoutu a stylizace komponent je praktikována prostřednictvím props (zpravidla nazvaná jako *style*). U složitější stylizace je vhodné využít konstrukce *Stylesheet.create* k definici stylů. V komponentě se lze poté na příslušnou definici odkázat prostřednictvím props style. Ukázka stylizace využitím Stylesheet.create je uvedena v Kódu 4.

```
const styles = StyleSheet.create({
    container: {
        marginTop: 10,
        justifyContent: 'center',
        alignItems: 'center',
    },
    welcome: {
        fontSize: 20,
        textAlign: 'center',
```

```
        margin: 10,
        color: 'black'
    },
    contentContainer: {
        borderWidth: 5,
        borderColor: 'black',
        flex: 1,
        width: 330,
        height: 420
    }
});
```
Kód 4 - Stylizace layoutu pomocí Stylesheet

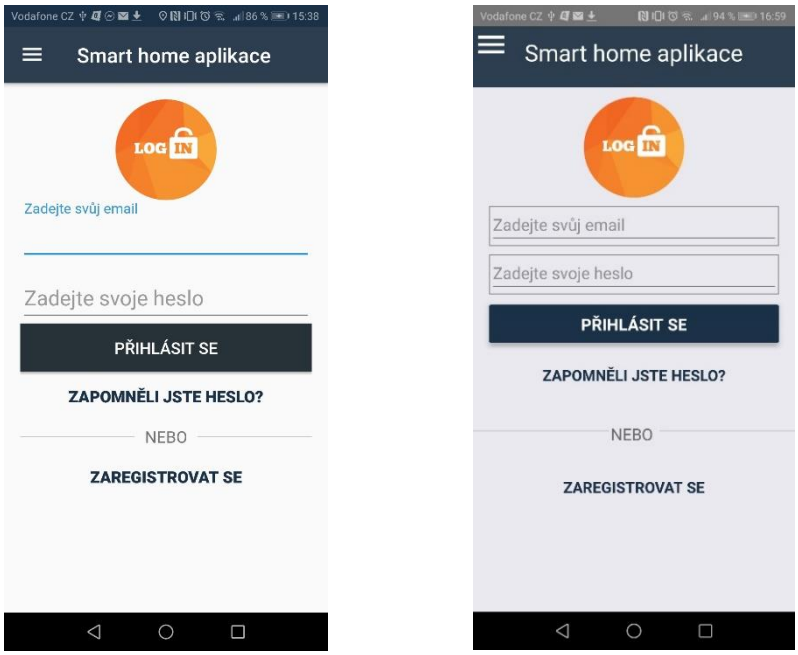
Zápis většiny atributů a jejich hodnot je velmi často shodný se zápisem v CSS. Definice rozměrů kontejnerů layoutu může být buď fixní (pomocí definice šířky a výšky) nebo flexibilní, a to pomocí atributu flex, který určuje, jak velkou část obrazovky bude daný kontejner zabírat. Atribut flex také určuje rozvržení stránky, respektive jak budou komponenty v kontejneru uspořádané. Pro testování designu stránky není k dispozici žádný designer tak jako v případě Xamarinu. Je tedy nutné provádět testování přímo na testovacím zařízení. Součástí každé komponenty jsou také props, pomocí nichž lze reagovat na různé události a slouží tak k práci s danou komponentou. Na ukázce Kódu 5 je představena práce s nativní komponentou *Switch*, jež je součástí každého screenu aplikace, neboť pomocí této komponenty se funkce aktivují či deaktivují.

```
<Switch onValueChange={this.toggleSwitch} value={this.state.switchValue}></Switch>
```
Kód 5 - React Native – ukázka práce s komponentou

Ke změně hodnoty komponenty Switch se používá props *onValueChange* a k zobrazení hodnoty state *switchValue*. Tato konstrukce přístupu ke komponentě, kdy k nastavení hodnot se používá příslušná state a ke změnám hodnot příslušná props, je základní konstrukcí při práci v React Native.

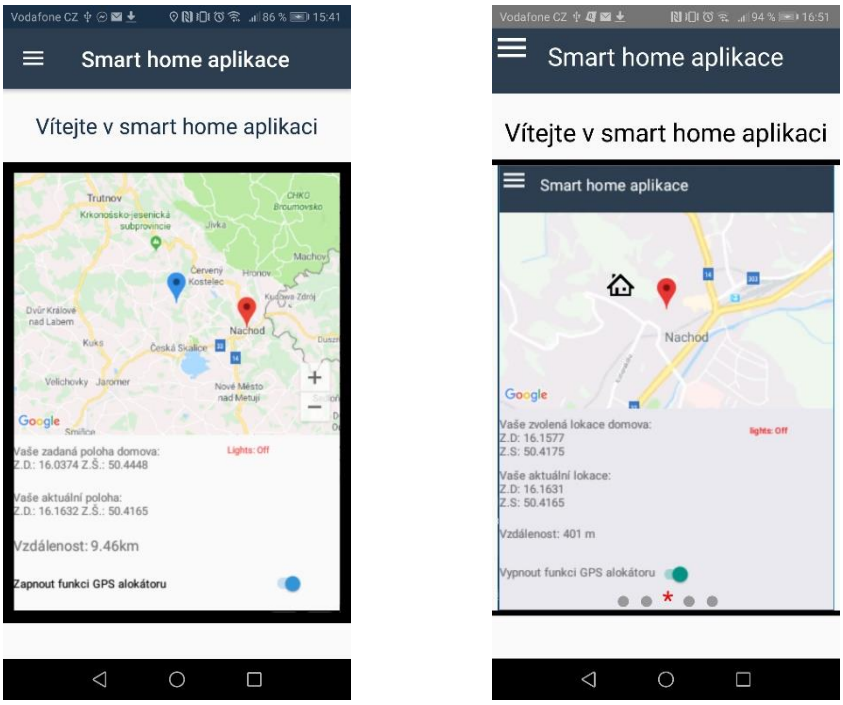
Porovnání designu ve vzorové aplikaci

Co se týče výsledného vzhledu jednotlivých stránek vzorové aplikace v porovnání s designem vytvořeném v nativním Android Studiu, jsou základní nativní komponenty obou frameworků (tedy jak součástí knihovny Xamarin.Android, tak knihovny react-native) v podstatě k nerozeznání od komponent, jež jsou součástí Android Studia. Jako příklad je vybrána přihlašovací stránka aplikace (Obrázek 9).



Obrázek 9 - porovnání vzhledu přihlašovací stránky vytvořené v Xamarinu (vlevo) a React Nativu (vpravo)

Změny v designu (Obrázek 9) u obou frameworků jsou minimální. Drobné změny lze pozorovat například u inputů textu v přihlašovacím formuláři, kde u Xamarinu je vzhled naprosto totožný jako v Android Studio (viz Obrázek 6), a to díky tomu, že definovat UI v Xamarinu je možné přímo pomocí atributů totožných s atributy v Android Studiu (viz kapitola *Vývoj UI v Xamarinu*). Tato možnost při definici UI v React Nativu není. U složitějších GUI komponent (v případě vzorové aplikace je testován carousel) jsou již rozdíly patrné (Obrázek 10).



Obrázek 10 - Porovnání vzhledu hlavní stránky vytvořené v Xamarinu (vlevo) a React Nativu (vpravo)

Z tohoto srovnání na Obrázku 10 si lze povšimnout viditelných změn. Součástí carouselu vytvořeného v Xamarinu mimo jiné není indikátor stránek. Knihovna Xamarin.Android totiž nepodporuje implementaci carouselu. Je sice možné použít nativní carousel, nicméně dokumentace na implementaci funkcí je psaná v Javě, jakožto nativním jazyku pro Android, dokumentace pro C# chybí. Naopak při implementaci v React Nativu je k dispozici knihovna, upravující nativní carousel pro Android Studio do jazyka JavaScript. Je tak možné využít všechny funkce, které jsou k dispozici pro nativní carousel ve zjednodušené formě.

3.4.2 Porovnání z hlediska implementace funkcí

Stejně tak jako vývoj UI je i samotná implementace jednotlivých funkcí u obou frameworků odlišná. Tato kapitola se zaměřuje na porovnání implementace stěžejních funkcionalit (práce s různými Android API či API třetích stran) v obou frameworcích.

Práce se senzory

Součástí vzorové aplikace jsou dvě funkce, které využívají práci se senzory – pracuje se světelným a pohybovým senzorem, konkrétně s gyroskopem. Všechny základní senzory mobilního zařízení s operačním systémem Android jsou spravovány nativní třídou *SensorManager*, jež je součástí knihovny android.hardware. A právě s touto třídou lze pracovat v obou frameworcích prostřednictvím upravených verzí pro každý framework.

V případě Xamarinu se jedná o stejnojmennou třídu SensorManager. V Kódu 6 je znázorněno, jak získat hodnoty z gyroskopického senzoru v Xamarinu.

```
// instance SensorManageru
private SensorManager sensorManager;

// inicializace instance SensorManageru
sensorManager = (SensorManager)this.Activity.GetSystemService(Context.SensorService);
// výběr senzoru pro event listener
Sensor sen = sensorManager.GetDefaultSensor(SensorType.Gyroscope);
// zaregistrování event listeneru
sensorManager.RegisterListener(this, sen, Android.Hardware.SensorDelay.Game);

// stěžejní metoda reagující na změny senzoru
public void OnSensorChanged(SensorEvent e) {
    // výběr senzoru
    e.Sensor = sensorManager.GetDefaultSensor(SensorType.Gyroscope);
        // pokud je funkce aktivována
        if (gyroscopeActivated) {
           // vypsání hodnot ze senzoru do textové komponenty
           textViewGyroscopeValue.Text = string.Format("x={0:f}, y={1:f}, z={2:f}",
                  e.Values[0], e.Values[1], e.Values[2]);
           lightColor.compute(e.Values[0], e.Values[1], e.Values[2]);
        }
}
```

Kód 6 - Xamarin – implementace gyroskopického senzoru

Pro manipulaci se senzorem je nutné implementovat rozhraní *ISensorEventListener*, jehož součástí je klíčová metoda *OnSensorChanged(SensorEvent)*, která je volána vždy při změně snímané hodnoty senzorem a ve které je možné hodnoty ze senzoru získat za využití parametru typu *SensorEvent*. K samotným hodnotám se přistupuje pomocí vlastnosti *Values*. Tato vlastnost představuje pole, kde každá položka obsahuje hodnotu vrácenou ze senzoru – význam hodnoty určuje zvolený typ senzoru. Tedy v případě gyroskopu (32) je hodnotou na nultém indexu v poli úhlová rychlost náklonu zařízení kolem osy x, na prvním indexu kolem osy y a na druhém indexu kolem osy z.

U implementace v React Nativu je k dispozici knihovna *NativeModules* (18), která slouží k manipulaci s nativními API obou platforem, které nemají své ekvivalenty pro React Native. V případě Androidu má tak knihovna strukturu stejnou jako nativní knihovny pro Android. V Kódu 7 je naznačeno, jak získat hodnoty ze senzoru za využití SensorManageru, jež je součástí knihovny NativeModules.

```
// instance SensorManageru
    var mSensorManager = require('NativeModules').SensorManager;
    // pokud je funkce aktivována
    if(!this.state.switchValue) {
        // přidání event listeneru prostřednictvím DeviceEventEmitter
        DeviceEventEmitter.addListener('Gyroscope', function (data) {
            console.log(data);
            // uložení dat z gyroskopu
            this.setState({x_axis : data.x, y_axis : data.y, z_axis : data.z});
```

```
        }.bind(this));
        // aktivace gyroskopu
        mSensorManager.startGyroscope(100);
    }
};
```
Kód 7 - React Native – Implementace gyroskopického senzoru

K naslouchání událostí nativních modulů (v tomto případě změn senzoru) se používá *DeviceEventEmitter*, kterému je možné přidat event listener požadovaného typu (v tomto případě gyroskopický senzor). Z parametru návratové funkce lze poté získat hodnoty snímané senzorem. Vzhledem k tomu, že se opět využívá nativní SensorManager, struktura získaných dat ze senzoru je stejná jako v případě Xamarinu. Senzor se aktivuje pomocí metody *startGyroscope()*, která v parametru přijímá časovou prodlevu mezi změnami senzoru v milisekundách (40).

Práce s SQLite databází

Ve vzorové aplikaci je ukázka práce s databází demonstrována u funkce snímání světelného světlení. Do databáze je ukládána aktuální hodnota snímaného osvětlení, případně zda je funkce aktivní.

Pro práci s SQLite databází v Xamarinu se využívá ORM knihovna *sqlite-net-pcl*. Tabulku v databázi představuje modelová třída. Nejprve je tedy nutné si vytvořit modelovou třídu, s vhodnou strukturou pro ukládání dat. Modelová třída obsahuje tři atributy, respektive vlastnosti – primární klíč *Id, Enabled* pro informaci, zda je funkce aktivována či ne a *QuantityOfLightInLuxUnits* pro uložení hodnoty světla. K definování tabulky (název, význam sloupců atd.) slouží anotace, které se přidávají k elementům jazyka C#. Kód modelové třídy je obsahem Kódu 8.

```
    [Table("AmbientLight")]
    public class LightIntensity
    {
       [PrimaryKey, AutoIncrement, Column("Id")]
       public int Id { get; set; }
       public bool Enabled { get; set; }
       public float QuantityOfLightInLuxUnits { get; set; }

    }
```
Kód 8 - Xamarin – ukázka modelové třídy reprezentující tabulku v SQLite

Následně se vytvoří pomocná statická třída, obsahující metody pro komunikaci s databází – připojení k databázi a CRUD operace nad daty. Většina operací s databází má svoji definovanou metodu, kterou lze volat na objektu třídy *SQLiteConnection*. Jestliže je požadována složitější operace je možné napsat libovolný SQL příkaz pomocí metody *Query()*.

V Kódu 9 je představeno připojení k databázi a ukázka využití metody Query() pro update kompletního řádku v tabulce.

```
// umístění souboru databáze
string folder =
System.Environment.GetFolderPath(System.Environment.SpecialFolder.Personal);

// připojení k databázi
using (var connection = new SQLiteConnection(System.IO.Path.Combine(folder,
"App.db"))) {

// operace Update pomocí metody Query
connection.Query<LightIntensity>("UPDATE AmbientLight set
Enabled=?,QuantityOfLightInLuxUnits=? Where Id=?",
 ambientLight.Enabled, ambientLight.QuantityOfLightInLuxUnits, ambientLight.Id);
}
```

Kód 9 - Xamarin – ukázka metody pro update databáze

Kód 9 je tělem metody s názvem *UpdateTableAmbientLight()* s parametrem instance modelové třídy LightIntensity (viz Kód 8). Jestliže ve fragmentu pro funkci snímání ambientního osvětlení je potřeba aktualizovat údaje v databázi, použije se Kód 10.

```
DataHelperForAmbientLight.UpdateTableAmbientLight(ambientLight);
```

Kód 10 - Xamarin – zavolání metody pro update tabulky z fragmentu pro funkci snímání osvětlení

V React Nativu se k implementaci SQLite databáze používá knihovna *react-native-sqlite-storage*. Implementace je velmi jednoduchá – stačí importovat metodu *openDatabase()*, přijímající v parametru objekt nastavení připojení s jediným povinným atributem a to názvem souboru databáze (umístění souboru je v případě implementace pro Android fixní). Pomocí této metody se vytvoří objekt databáze, prostřednictvím něhož lze provádět transakce v databázi za využití metody transaction(). Součástí metody transaction() je funkce, ve které lze napsat libovolný SQL příkaz. Metoda pro provádění SQL příkazů se nazývá příznačně *executeSql()* a její součástí je návratová funkce, jenž obsahuje informace o prováděném příkazu (mimo jiné hodnoty a počet řádků, na kterých byl příkaz proveden). V případě SQL dotazu obsahují tyto informace výsledek v podobě výstupu z databáze. V ukázce Kódu 11 je příklad práce s SQLite databází v React Nativu v podobě vytvoření připojení a následném updatu databáze při změně hodnoty osvětlení.

```
// vytvoření připojení k databázi
var db = openDatabase({ name: 'UserDatabase.db'});

// funkce pro update tabulky
 db.transaction(function (tx) {
     tx.executeSql(
         'UPDATE table_light set light_amount=? where light_id=?',
         [data.light, data.id],
         // návratová funkce s informacemi o provedeném SQL příkazu
         (tx, results) => {
```

```
            console.log('ResultsUp', results.rowsAffected);
        });
});
```

Kód 11 - React Native – ukázka práce s databází při updatu tabulky

Práce s Google Maps API, aktuální pozicí a výpočtem vzdálenosti mezi pozicemi

Součástí funkce porovnání aktuální pozice s pozicí domova je práce s Google Maps API (načtení mapy a práce s markery), zjištění aktuální pozice zařízení a výpočet vzdálenosti mezi aktuální a zvolenou pozicí.

Pro přístup ke Google Maps API je nutné si vytvořit projekt na *Google Cloud Platform*, který sdružuje všechny důležité API od Googlu. Ve vytvořeném projektu je potřeba aktivovat Google Maps API pro Android a vygenerovat unikátní API klíč, jež se poté přidá do hlavního konfiguračního souboru aplikace *AndroidManifest.xml* a díky němuž lze s API komunikovat.

Pro práci s mapou v Xamarinu slouží knihovna *Xamarin.GooglePlayServicesMaps.Maps*. Knihovna obsahuje třídy a rozhraní ekvivalentní k nativnímu Google Maps API pro Android (pouze lehce pozměněné názvy a převedené konstrukce z Javy do C#). K zobrazení mapy je potřeba kontejner *MapFragment*, do kterého se mapa vkládá, a samotný objekt map v podobě instance třídy *GoogleMap*. Pro práci s mapou je klíčové rozhraní *IOnMapReadyCallback*, jehož součástí je jediná metoda *OnMapReady()*, ve které se nachází veškerá implementace práce s mapou. Pro přidání markeru na zvolenou pozici domova do mapy je použito rozhraní *IOnMapClickListener* s klíčovou metodou *OnMapClick()*. Marker je přidán také na aktuální pozici uživatele. Pro definici markeru je použita třída Marker s velkým množstvím atributů pro úpravu vzhledu a chování markeru. Kód 12 zahrnuje implementaci načtení a zobrazení mapy a přidání markeru na zvolenou pozici uživatelem za pomoci metody *OnMapClick()*.

```
// Instance mapFragmentu pro umístění map
private MapFragment mapFragment;
// instance Google Map
private GoogleMap googleMap;
// inicializace mapFragmentu a umístění Google Map do fragmentu
mapFragment = (MapFragment)FragmentManager.FindFragmentById(Resource.Id.map);
if (mapFragment != null)
{
    mapFragment.GetMapAsync(this);
}

// metoda zavolaná při načtení map
public void OnMapReady(GoogleMap googleMap)
{
    this.googleMap = googleMap;
    // zaregistrování listeneru pro kliknutí na mapu
    googleMap.SetOnMapClickListener(this);
}
```

```
// metoda volaná při kliknutí na mapu
public void OnMapClick(LatLng point)
{
     // uložení pozice při kliknutí do pozice domova
     this.homeLocation = point;
     // přidání nového markeru na označenou pozici a nastavení jeho atributů
     marker = googleMap.AddMarker(new MarkerOptions()
     .SetPosition(point)
     .SetIcon(Android.Gms.Maps.Model.BitmapDescriptorFactory
     .DefaultMarker(Android.Gms.Maps.Model.BitmapDescriptorFactory.HueAzure))
     .SetTitle("Můj domov"));
}
```

Kód 12 - Xamarin – ukázka práce s Google Maps API

React Native nabízí pro práci s Google Maps API knihovnu *react-native-maps*. Pro implementaci map není potřeba mít instanci na mapu (v některých případech ano – například při animaci kamery), dostačující je kontejner pro umístění mapy – komponenta *MapView*. Veškeré nastavení je součástí právě této komponenty. Pro interakci s mapou lze využít props *onPress* a v ní definovat obslužnou funkci. Pro vytvoření a nastavení vlastností markeru slouží komponenta MapMarker. V Kódu 13 je představena ukázka inicializace mapy a přidání markeru při kliknutí uživatele.

```
// definice mapy
<MapView
    // odkaz na mapu
    ref={(ref) => { this.mapView = ref }}
    style={styles.map}
    // nastavení počáteční lokace
    region={{
        latitude: this.state.currentPosition.lat,
        longitude: this.state.currentPosition.lng,
        latitudeDelta: 0.015,
        longitudeDelta: 0.0121,
    }}
    onPress={this.mapPressed}>
// definice markeru
<MapMarker title="Můj domov"
           // umístění markeru
           coordinate={{latitude:this.state.homePosition.lat, longitude:
this.state.homePosition.lng}}>

// funkce při kliknutí na mapu
mapPressed = (event) => {
    this.setState({
        // uložení pozice při kliknutí do domovské pozice
        homePosition: {lat: event.nativeEvent.coordinate.latitude, lng:
event.nativeEvent.coordinate.longitude},
        // atribut řídící zda se má zobrazit marker
        mapClicked: true
    });
```

Kód 13 - React Native – ukázka práce s Google Maps API

Pro zjištění aktuální pozice zařízení je nutné mít povolení od uživatele zařízení (viz kapitola *Přidání povolení*). V Xamarinu je využito *Google Location Services API*, které je

součástí služeb Google Play Services a v React Native *android.location API*. Dle dokumentace pro Android (32) je přesnější a doporučené *Google Location Services API*.

Pro spojení se službami Google Play Services je nejjednodušším spojovatelem objekt *GoogleApiClient*. Při implementaci v Xamarinu se nejdříve zažádá o povolení uživatele. Následně se vytvoří a nakonfiguruje objekt spojení *GoogleApiClient* a vytvoří se objekt *LocationRequest*, který se využije při zaslání požadavku na zjištění aktuální pozice. Po vytvoření klíčových objektů se implementuje metoda *OnConnected()*, která je součástí listeneru pro objekt *GoogleApiClient IConnectionCallbacks*, a v této metodě se požádá o zasílání aktuální polohy při změně pozice zařízení. K tomu slouží metoda *RequestLocationUpdates()* obsahující v parametrech objekty *GoogleApiClient* a *LocationRequest*. Posledním krokem je samotné uložení a zobrazení aktuální pozice, což je zprostředkováno díky metodě *OnLocationChanged()*, která je součástí rozhraní *ILocationListener* a je volána při změně polohy zařízení. Při opuštění stránky je služba zastavena pomocí metody *RemoveLocationUpdates()*. Vzhledem ke složitosti implementace je v následující Kódu 14 ukázána pouze metoda pro aktivaci zasílání aktuální polohy a metoda pro naslouchání změny aktuální polohy.

```
// klíčové objekty pro spojení a komunikaci
private GoogleApiClient mGoogleApiClient;
private Location mLastLocation;

// metoda při úspěšném spojení se službou Google Location Services API
public void OnConnected(Bundle connectionHint)
{
    // zobrazení aktuální pozice na mapě
    DisplayLocation();
    // pokud je povolena aktualizace aktuální pozice
    if(mRequestingLocationUpdates)
    {
        // aktualizuj aktuální pozici
        LocationServices.FusedLocationApi.
        RequestLocationUpdates(mGoogleApiClient, mLocationRequest, this);
    }
}

// metoda pro uložení a zobrazení aktuální pozice
public void OnLocationChanged(Location location)
{
    mLastLocation = location;
    DisplayLocation();
}
```

Kód 14 - Xamarin – ukázka implementace získání aktuální polohy zařízení

Pro implementaci *android.location API* v React Nativu je nutné nainstalovat knihovnu *react-native-geolocation*. V kódu není třeba žádného importu. Pro přistup k API slouží objekt *navigator* a k jednotlivým metodám se přistupuje podle následujícího vzoru:

navigator.geolocation.názevMetody(). Součástí každé metody jsou návratové funkce při úspěšném a neúspěšném provedení metody a nastavení konfigurace zjištění polohy (míra přesnosti či časová prodleva). Obsahem Kódu 15 je nastavení konfigurace a metoda pro zjištění aktuální pozice při načtení komponenty, včetně návratové funkce při úspěšném provedení.

```
// nastavení konfigurace
let geoOptions = {
    enableHighAccuracy: true,
    timeOut: 2000000,
};

// zjištění pozice při načtení
navigator.geolocation.getCurrentPosition( this.geoInitialPosSuccess,
    this.geoFailure,
    geoOptions);

// návratová funkce při úspěšném získání počáteční pozice
geoInitialPosSuccess = (position) => {
    this.setState({
        currentPosition: {lat: position.coords.latitude,lng:position.coords.longitude},
    })
};
```

Kód 15 - React Native – ukázka implementace získání aktuální polohy zařízení

Výpočet vzdálenosti mezi aktuální pozicí a vybranou domovskou pozicí je prováděn vždy při kliknutí na mapu (při změně zvolené pozice domova). V Xamarinu je pro výpočet použito nativní Android API, přístupné prostřednictvím knihovny *System.Device.Location.GeoCoordinate* – konkrétně metoda *GetDistanceTo()*, která je schopna vypočítat vzdálenost mezi zeměpisnými souřadnicemi dvou bodů v metrech. V React Nativu je výpočet zprostředkován komunitní knihovnou poskytující základní geoprostorové operace – *geolib*. Pro samotný výpočet vzdálenosti mezi dvěma body slouží metoda *getDistance()*, kde v parametru je možné navíc nastavit přesnost výpočtu na metry. Výsledná hodnota je opět v metrech. Při porovnání vrácených výsledků obou metod se hodnoty lišily minimálně (v řádu centimetrů až milimetrů).

Práce s Firebase Authentication API a přístup k uložišti

Služba Firebase, konkrétně její součást Authentication, je v aplikaci využita pro systém autorizace. Vzorová aplikace využívá funkce registrace, přihlášení a zapomenutého hesla. Pro přístup k API lze využít existující projekt na platformě Google Cloud Service, případně si založit nový. Následně je na základě několika zadaných údajů (povinný je pouze název aplikace a *Android package name*, obsažený v souboru *AndroidManifest.xml*) vygenerován soubor *google-services.json*, který zastupuje API klíč a umožňuje tedy přístup k API. Poté je potřeba stáhnout SDK pro příslušnou platformu – knihovny *Xamarin.Firebase* pro Xamarin a

react-native-firebase pro React Native. Posledním krokem je synchronizace aplikace se službou Firebase.

Implementace funkcí je krátká, pouze v případě Xamarinu je navíc nutné inicializovat instanci *FirebaseAuth* za pomoci instance *FirebaseApp*, která slouží k inicializaci služby a je nutné ji vytvářet při každém spuštění aplikace. Instance obsahuje API klíč a Id aplikace, oba údaje jsou k dispozici ve zmiňovaném souboru konfigurace služby *google-services.json*. (viz Kód 16). V React Native se k instanci *FirebaseAuth* přistupuje přes objekt *firebase* – není nutná žádná inicializace a konfigurace služby (viz Kód 17). Na instanci *FirebaseAuth* jsou volány metody služby Firebase Authentication. Metody jsou v obou frameworcích totožné. V Kódu 16 a 17 jsou příklady implementace funkce pro přihlášení uživatele v obou frameworcích.

```
// inicializace služby Firebase Authentication
private void InitFirebaseAuth()
{
    // konfigurace inicializace
    var options = new FirebaseOptions.Builder()
        .SetApplicationId("1:505549548041:android:493c7b35ca18a06a")
        .SetApiKey("AIzaSyBPVgzrAbK3Ong-3wOCHyxz6BEAkphQh-E")
        .Build();
        if (app == null)
        {
            app = FirebaseApp.InitializeApp(this, options, "App 2");
        }
        // inicializace klíčové instance FirebaseAuth pro používání funkcí služby
        auth = FirebaseAuth.GetInstance(app);
}

// metoda pro přihlášení uživatele
private void LoginUser(string email, string password)
{
    auth.SignInWithEmailAndPassword(email, password)
        .AddOnCompleteListener(this);
}
```

Kód 16 - Xamarin – ukázka přihlašovací funkce pomocí Firebase Authentication

```
// funkce přihlášení uživatele
handleLogin = () => {
    const { email, password } = this.state;
    firebase.auth().signInWithEmailAndPassword(email, password)
        .then(() => this.props.navigation.navigate('DashBoardScreen'))
        .catch(error => this.setState({errorMessage: error.message}))
};
```

Kód 17 - React Native – ukázka přihlašovací funkce pomocí Firebase Authentication

Přístup k uložišti, respektive ke galerii obrázků, je ve vzorové aplikaci součástí funkce uživatelova profilu. Uživatel zde může vybrat obrázek z různých sekcí galerie či pořídit nový za použití fotoaparátu. Poté se mu obrázek uloží k jeho profilu. Jak implementace, tak UI nástrojů, použitých pro přístup do uložiště u obou frameworků je značně odlišné. Nicméně

oba nástroje umožňují použití fotoaparátu pro pořízení nového obrázku. Pro přidání obrázku k profilu je využita možnost, která je součástí Firebase Authentication API a to upravit profil uživatele a rozšířit ho o profilový obrázek – metoda *updateProfile()* (k tomu stačí pouze URI obrázku, nicméně pro zobrazení v aplikaci je nutné URI převést na bitmap).

V Xamarinu není nutná žádná rozšiřující knihovna. Stačí využít objekt *Intent* (standartní objekt ke startu nové aktivity) a rozhraní *IOnCompleteListener* k zachycení výsledku aktivity. Objekt *Intent* totiž obsahuje akci, která umožní vybrat a vrátit libovolný dokument pouze pomocí zadání konstanty *ActionOpenDocument*. Co má být obsahem vrácených dat je definováno v kategorii objektu *Intent*. Typ vráceného média (jaké soubory uživatel může vybrat) se nastavuje v metodě *setType()*. Metoda *StartActivityForResult()* pak odstartuje přístup do uložiště. Metoda přijímá nakonfigurovaný objekt *Intent* a libovolně zvolený kód požadavku.[2] Pro zachycení vrácených dat je implementována metoda *OnActivityResult()*. Příklad implementace je v Kódu 18.

```
// přístup k uložišti
Intent intent = new Intent(Intent.ActionOpenDocument); // definice aktivity
intent.AddCategory(Intent.CategoryOpenable); // volba návratová hodnoty - stačí URI
obrázku
intent.SetType("image/*"); // chceme vybrat pouze obrázky
StartActivityForResult(intent, GALLERY_INTENT_CALLED); // zahájení aktivity

// metoda pro získání výsledků
public override void OnActivityResult(int requestCode, Result resultCode, Intent data)
{
if (resultCode == Result.Ok) {
     // uložení dat
     originalUri = data.Data;
    }
}
// přidání obrázku k profilu pomocí metody UpdateProfile()
....
```

Kód 18 - Xamarin – ukázka přístupu k uložišti

V React Nativu neexistuje přímá možnost přístupu k uložišti pomocí nativních prvků. Je nutné rozšířit projekt o knihovnu *react-native-image-picker*. Ta obsahuje klíčový objekt *ImagePicker*, který umožňuje vybrat médium z uložiště prostřednictvím nativního UI. Vše řídí metoda *showImagePicker()*, jež přijímá objekt konfigurace a návratovou funkci s vrácenými daty. Prostřednictvím těchto dat je možné řídit chování po úspěšném či neúspěšném výběru média a v případě úspěšného provedení samotná data ukládat. Objekt konfigurace umožňuje nastavovat například šířku, výšku či kvalitu zobrazovaných obrázků,

[2] Tento postup implementace platí pouze pro verze Androidu vyšší než Android API 19 (KitKat). Pro verze nižší je implementace pozměněná. I tato varianta je součástí vzorové aplikace.

typ vráceného média nebo možnosti uložení při vyfocení nového obrázku. Ukázka implementace je v Kódu 19.

```
// konfigurace nástroje pro výběr obrázku
const options = {
    quality: 1.0,
    maxWidth: 500,
    maxHeight: 500,
    path: 'images',
    storageOptions: {
        skipBackup: true
    }
};
// funkce pro přístup do uložiště
ImagePicker.showImagePicker(options, (response => {
    console.log('Response: ' + response);
    if (response.ok) {
        const source = { uri: response.uri };
        this.setState({
            imgSource: source,
        });
    }
}));
```

Kód 19 - React Native – ukázka přístupu k uložišti

Implementace carouselu

React Native nabízí pro implementaci carouselu komunitou vytvořenou knihovnu *react-native-carousel-view*. Ta se stará o veškerou implementaci – stačí tedy pouze importovat komponentu *Carousel*, nastavit její vlastnosti v props a přidat obrázky. Xamarin.Android nenabízí žádnou knihovnu pro přímou implementaci carouselu. Přesto si lze takovou komponentu vytvořit, a to skloubením nativního kontejneru pro vložení obrázků s možností manuálního přepínání mezi obrázky *ViewPager* a časovače pro automatické přepínání (třída *Timer*). O přepínání obrázků se stará abstraktní třída *PageAdapter*, jejíž metody je potřeba implementovat (jedná se o definici, formát a celkový počet obrázků a aktuálně zobrazovaný obrázek). Pro automatické přepínání mezi obrázky je klíčová vlastnost *CurrentItem*, pomocí níž lze měnit aktuálně zobrazovaný obrázek, v tomto případě po určitém časovém úseku určeném v časovači.

3.4.3 Porovnání z hlediska využití zdrojů a API třetích stran

K tomu, aby bylo možné pracovat s některými zdroji mobilního zařízení je nutné pracovat s externími knihovnami. Proces přidání a správa knihoven, včetně kontroly závislostí mezi knihovnami je předmětem podkapitoly *Přidání a správa knihoven*. Součástí práce se zdroji jsou od verze operačního systému Android 4.4 KitKat (API 19) také nutná povolení od uživatele. Těm se věnuje podkapitola *Přidání povolení*. Poslední podkapitola *Možnosti*

frameworků srovnává možnosti využití zdrojů a API třetích stran testovaných ve vzorové aplikaci u obou frameworků.

Přidání a správa knihoven

V Xamarinu je veškerá správa knihoven v podobě balíčků řízena nástrojem *NuGet* (41). Systém funguje tak, že tvůrci knihovny zabalí svůj kód do balíčku a ten umístí na server *nuget.org*, odkud si ho mohou uživatele Xamarinu stáhnout a následně ho použít. Při vyhledávání balíčků je k dispozici náhled s výpisem všech závislostí daného balíčku, možností zvolit si verzi ke stažení či odkaz na dokumentaci. Největší předností nástroje je však správa závislostí. Už při stahování knihovny se kontrolují veškeré jeho závislé knihovny a jejich verze a porovnávají se s knihovnami již nainstalovanými v projektu. Pokud dojde k nesrovnalostem, vývojář je upozorněn a jsou mu vypsány všechny balíčky a jejich verze, které je nutné stáhnout, aby požadovaná knihovna byla plně funkční.

V React Nativu je postup instalace externích knihoven složitější. K instalaci balíčků je využit registr knihoven *npm*, který je k dispozici po instalaci *Node.js* a je ovládán pomocí příkazové řádky. Pro použití npm příkazů je nutné se nacházet v adresáři React Native projektu. Pro instalaci je použit příkaz: *npm install <název_balíčku>*. Po instalaci je nutné knihovnu nalinkovat do projektu, tak aby se s ní dalo pracovat v kódu. To je možné buď automaticky pomocí příkazu *npm link <název_balíčku>* nebo manuálně. Při manuálním postupu je knihovna přidána do hlavní aktivity Android aplikace a do konfiguračních souborů *settings.gradle* a *build.gradle*, které slouží pro inicializaci Android aplikace. Při přidávání balíčků není nikterak kontrolována závislost s ostatním již nainstalovanými knihovnami. Vývojář se o rozporu mezi verzemi použitých knihoven dozví až při testování aplikace.[3] Přehled všech závislostí mezi knihovnami je možné vypsat pomocí příkazu: */gradlew app:dependencies*.

Přidání povolení

U většiny zdrojů mobilního zařízení je nutné pro jejich využívání požádat uživatele zařízení o povolení k přístupu. Při vývoji aplikace pro Android se tato povolení přidávají do souboru *AndroidManifest.xml* (stejné u obou frameworků). Seznam všech zdrojů, které pro přístup vyžadují povolení je v dokumentaci (32). Ve vzorové aplikaci se jednalo o přístup

[3] Při vývoji vzorové aplikace byl rozpor především mezi knihovnami Firebase API a Google Maps API a to proto, že obě využívají služby Google Play Services, nicméně každá jinou verzi. Všechny problémy byly vyřešeny, především díky React Native komunitě na webové službě GitHub.

k aktuální pozici, uložišti a fotoaparátu zařízení. Následující kód obsahuje přidání povolení pro přístup k aktuální pozici v *AndroidManifest.xml*.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
android:versionCode="1" android:versionName="1.0" package="App2.App2"
android:installLocation="auto">
. . .
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
. . .
</manifest>
```

Kód 20 - Xamarin – ukázka přidání povolení pro přístup k aktuální pozici

Povolení *ACCESS_FINE_LOCATION* slouží k zaměření co nejpřesnější polohy za využití GPS, zatímco *ACCESS_COARSE_LOCATION* využívá pouze Wi-Fi a mobilní data pro určení přibližné polohy (42). Jakmile je povolení přidáno, uživatel je automaticky dotázán při prvním spuštění aplikace o povolení ke zdroji. V Xamarinu je navíc nutné, aby zažádání o povolení bylo přímo součástí implementace práce s příslušným zdrojem. Veškerá implementace práce se zdrojem následuje až za implementací přidání povolení, neboť bez ní není možné se zdrojem pracovat. Obsahem Kódu 21 je implementace kontroly povolení pro přístup k aktuální poloze a následné zažádání o povolení, pokud již nebylo přiděleno.

```
// kontrola přidělení povolení k aktuální poloze
if (ActivityCompat.CheckSelfPermission(this,
Manifest.Permission.AccessFineLocation) != Permission.Granted
&& ActivityCompat.CheckSelfPermission(this,
Manifest.Permission.AccessCoarseLocation) != Permission.Granted)
{
// pokud nejsou povolení přidělena – je o ně zažádáno
ActivityCompat.RequestPermissions(this, new string[]{
               Manifest.Permission.AccessCoarseLocation,
               Manifest.Permission.AccessFineLocation},
 MY_PERMISSION_REQUEST_CODE);
}
```

Kód 21 - Xamarin – ukázka implementace kontroly a zažádání o povolení k aktuální poloze

Možnosti frameworků

V následující tabulce jsou srovnány možnosti obou frameworků z pohledu využití všech zdrojů a API třetích stran, které byly implementovány ve vzorové aplikaci. Možnosti frameworku jsou rozděleny do 3 kritérií: Jestliže je SDK pro práci se zdrojem přímo součástí frameworku, je zdroj označen jako „součástí frameworku“; Pokud je zdroj ovládán pomocí přidané externí knihovny je označen jako „externí knihovna“; Pokud framework vůbec nepodporuje danou funkcionalitu je zdroj označen jako „není podporováno“.

Tabulka 2 - přehled možností frameworků z pohledu práce se zdroji a API třetích stran

Zdroj	Senzory	Aktuální poloha	Přístup k uložišti	Google Maps API	Firebase API	SQLite API	Carousel
Xamarin	Součástí frameworku	Externí knihovna	Součástí frameworku	Externí knihovna	Externí knihovna	Externí knihovna	Není podporováno
React Native	Externí knihovna	Externí knihovna	Externí knihovna	Externí knihovna	Externí knihovna	Externí knihovna	Externí knihovna

Z Tabulky 2 vyplývá, že v React Nativu je možné veškeré funkcionality testované ve vzorové aplikaci realizovat, a to za využití externích knihoven. V Xamarinu (Xamarin.Android) není možnost přímé implementace carouselu. Práce se senzory a uložištěm je přímo součástí SDK integrovaných ve frameworku. Pro zjištění aktuální polohy a práci s API třetích stran jsou použity externí knihovny.

4 Shrnutí výsledků

Xamarin a React Native jakožto dva vybrané frameworky pro vývoj multiplatformních mobilních aplikací byly porovnány z pohledu vzorové aplikace implementované v obou frameworcích. Porovnání proběhlo na základě třech aspektů – uživatelské rozhraní, implementace funkcí a využití zdrojů zařízení případně API třetích stran.

Pro vývoj uživatelského rozhraní s využitím základních nativních komponent se zdá být efektivnějším a rychlejším frameworkem Xamarin, a to především díky možnosti využití grafického designeru, který umožňuje okamžitou kontrolu designu již při psaní kódu a díky možnosti implementace za pomoci nativního Android kódu (je možné využít obsáhle dokumentace Androidu). Naproti tomu React Native nabízí širokou škálu knihoven pro vývoj složitějších GUI komponent typu carouselu, pro který nativní Xamarin nemá podporu vůbec. Stejně tak samotná koncepce React Nativu, kdy se celá aplikace skládá z nativních GUI komponent a aplikační logika představuje pouze vlastnosti těchto komponent je velmi výhodná při vývoji aplikace se složitější grafikou za využití mnoha nativních prvků s možností velké interakce s uživatelem.

Tato koncepce vývoje aplikace je velmi výhodná i při samotné implementaci funkcionalit, neboť z implementace funkcí vzorové aplikace je patrné, že kód je zpravidla kratší a jednodušší v React Nativu. Rozdíly mezi návratovými hodnotami z funkcí byly minimální, a to z toho důvodu, že oba frameworky zpravidla využívají stejné nativní Android API, pouze v případě zjištění aktuální polohy Xamarin využívá, dle dokumentace pro Android, novější a doporučenou knihovnu. I zde však byly rozdíly v řádu centimetrů až milimetrů, tedy zanedbatelné.

Pro správu knihoven a s tím spojeném řešení závislostí mezi knihovnami je v Xamarinu využit nástroj NuGet, který uživatele odstíní od řešení problémů se závislostmi mezi externími knihovnami při spouštění či testování aplikace. V React Nativu žádný podobný nástroj neexistuje, problémy se závislostmi se tak řeší manuálně až při testování aplikace. Při řešení povolení ke zdrojům je v Xamarinu nutné zakomponovat do implementace dotaz na povolení funkce. V případě React Nativu stačí pouze přidat povolení a dotaz proběhne automaticky. Oba frameworky vykázali výborné výsledky z pohledu jejich možností pro implementaci veškerých funkcí vzorové aplikace, neboť jediné, co nebylo možné implementovat byl carousel v Xamarinu. Výhodou Xamarinu však je, že některé funkce pro

práci s nativním prvky Androidu jsou přímo součástí frameworku a není tak nutné instalovat a přidávat rozšiřující externí knihovny.

5 Závěr

Cílem práce bylo porovnání dvou vybraných frameworků pro vývoj multiplatformních aplikací z pohledu vzorové aplikace implementované v obou frameworcích. Jako porovnávané frameworky byly vybrány Xamarin a React Native. Následně byla navržena vhodná vzorová aplikace využívající různé zdroje a funkce mobilního zařízení. Byly vybrány vhodné nástroje pro vývoj aplikace v obou frameworcích a zařízení pro testování (jak fyzická, tak virtuální). Jako cílový operační systém byl zvolen Android. Následně byla aplikace implementována a oba frameworky byly porovnány. Toto porovnání vychází z implementovaných funkcionalit vzorové aplikace a je rozděleno do tří kategorií: Porovnání z hlediska uživatelského rozhraní, porovnání z hlediska implementace funkcí a porovnání z hlediska využití zdrojů mobilního zařízení a služeb API třetích stran.

Oba frameworky umí vytvořit uživatelské rozhraní s užitím nativním GUI prvků, tudíž výsledný dojem aplikace z pohledu designu je srovnatelný s aplikací vytvořenou v nativním Android Studiu. Jestliže se klade důraz na jednodušší design s využitím základních nativních komponent je vhodnější vývoj UI v Xamarinu. Pakliže má být součástí UI větší množství složitějších komponent za využití velké interakce s uživatelem vývoj je efektivnější v React Nativu. Oba frameworky umožnili implementaci všech funkcí vzorové aplikace, a to s minimálními rozdíly ve vrácených hodnotách. Implementace jednotlivých funkcionalit byla velmi odlišná, nicméně v naprosté většině byl kód vytvořený v React Nativu kratší a přímočařejší. Při srovnání práce se zdroji dominoval Xamarin ve správě externích knihoven a řešení závislostí mezi nimi. Naopak při přidání povolení pro práci se zdroji vynikal React Native. Nejpodstatnějším zjištěním však bylo, že oba frameworky umožnili práci se všemi zdroji zařízení, se kterými vzorová aplikace pracovala, ať už v podobě externích knihoven či přímo integrovaných knihoven ve frameworku.

Jak pomocí Xamarinu, tak pomocí React Nativu je tedy možné vytvořit komplexní mobilní aplikaci využívající různorodé nativní prvky zařízení s nativním designem a dobrou výkonností, která je srovnatelná s aplikací vytvořenou v nativním Android Studio. Výhodou takovéto technologie je možnost sdílení velké části kódu při vývoji aplikace pro různé platformy.

6 Seznam zdrojů

1. www.statista.com. *Statista.* [Online] 5. 2018. [Citace: 13. 8. 2018.] https://www.statista.com/statistics/271644/worldwide-free-and-paid-mobile-app-store-downloads/.

2. www.techopedia.com. *Techopedia.* [Online] 2018. [Citace: 13. 8. 2018.] https://www.techopedia.com/definition/2953/mobile-application-mobile-app.

3. Viswanathan, Priya. www.lifewire.com. *Lifewire.* [Online] 13. 11. 2017. [Citace: 13. 8. 2018.] https://www.lifewire.com/what-is-a-mobile-application-2373354.

4. www.thinkmobiles.com. *ThinkMobiles.* [Online] 5. 12. 2016. [Citace: 14. 8. 2018.] https://thinkmobiles.com/blog/popular-types-of-apps/.

5. www.mobiloud.com. *MobiLoud.* [Online] [Citace: 14. 8. 2018.] https://www.mobiloud.com/blog/native-web-or-hybrid-apps/.

6. Beal, Vangie. www.webopedia.com. *webopedia.* [Online] 7. 8. 2018. [Citace: 15. 8. 2018.] https://www.webopedia.com/DidYouKnow/Hardware_Software/mobile-operating-systems-mobile-os-explained.html.

7. www.techterms.com. *TechTerms.* [Online] 7. 3. 2013. [Citace: 15. 8. 2018.] https://techterms.com/definition/framework.

8. Wodehouse, Carey. www.upwork.com. *upwork.* [Online] [Citace: 15. 8. 2018.] https://www.upwork.com/hiring/development/understanding-software-frameworks/.

9. www.chipkidz.wordpress.com. *Chipkidz.* [Online] 6. 8. 2009. [Citace: 18. 8. 2018.] https://chipkidz.wordpress.com/2009/08/06/frozen-spots-hot-spots/.

10. Goodrich, Glen. www.sitepoint.com. *sitepoint.* [Online] 25. 10. 2017. [Citace: 20. 8. 2018.] https://www.sitepoint.com/model-view-controller-mvc-architecture-rails/.

11. www.g2crowd.com. *G2CROWD.* [Online] 2018. [Citace: 15. 8. 2018.] https://www.g2crowd.com/categories/mobile-development-frameworks.

12. Gažo, František. www.medium.com. *INLOOPX.* [Online] 15. 6. 2018. [Citace: 16. 8. 2018.] https://medium.com/inloopx/native-or-cross-platform-mobile-app-f113aa7ab581.

13. Looper, Jen. www.developer.telerik.com. *Telerik Developer Network.* [Online] 9. 11. 2015. [Citace: 16. 8. 2018.] https://developer.telerik.com/featured/what-is-a-webview/.

14. Burns, Amy, Umbaugh, Brad a Dunn, Craig. www.docs.microsoft.com. *Microsoft.* [Online] 23. 3. 2017. [Citace: 16. 8. 2018.] https://docs.microsoft.com/en-us/xamarin/cross-platform/app-fundamentals/building-cross-platform-applications/understanding-the-xamarin-mobile-platform.

15. Mádr, Vojtěch. www.eman.cz. *eman.* [Online] 29. 2. 2016. [Citace: 16. 8. 2018.] https://www.eman.cz/blog/xamarin-predstavujeme-nastroj-pro-multiplatformni-vyvoj-mobilnich-aplikaci-dil-1/.

16. Fischer, Roman. www.skeleton.cz. *skeleton.* [Online] 1. 3. 2017. [Citace: 16. 8. 2018.] https://www.skeleton.cz/uvod-do-xamarin-forms.

17. Sekulić, Robert. www.medium.com. *Medium.* [Online] 3. 10. 2016. [Citace: 18. 8. 2018.] https://medium.com/react-native-development/a-brief-history-of-react-native-aae11f4ca39.

18. www.facebook.github.io. *React Native.* [Online] 2018. [Citace: 18. 8. 2018.] https://facebook.github.io/react-native/.

19. Official, Thinkwik. www.medium.com. *Medium.* [Online] 31. 1. 2018. [Citace: 18. 8. 2018.] https://medium.com/@thinkwik/react-native-what-is-it-and-why-is-it-used-b132c3581df.

20. Basrai, Murtaza. www.icicletech.com. *icicle technologies.* [Online] 9. 9. 2016. [Citace: 18. 8. 2018.] https://www.icicletech.com/blog/top-10-editors-for-react-native.

21. www.ionicframework.com. *Ionic docs.* [Online] [Citace: 19. 8. 2018.] https://ionicframework.com/docs/.

22. Bradley, Adam. www.blog.ionicframework.com. *Ionic blog.* [Online] 28. 10. 2013. [Citace: 19. 8. 2018.] https://blog.ionicframework.com/where-does-the-ionic-framework-fit-in/.

23. www.angular.io. *Angular docs.* [Online] Google, 2019. [Citace: 20. 8. 2018.] https://angular.io/docs.

24. www.typescriptlang.org. *TypeScript.* [Online] Microsoft, 2019. [Citace: 22. 8. 2018.] https://www.typescriptlang.org/.

25. Lynch, Max. www.blog.ionicframework.com. *Ionic blog.* [Online] 18. 5. 2016. [Citace: 19. 8. 2018.] https://blog.ionicframework.com/what-is-a-progressive-web-app/.

26. www.docs.microsoft.com. *Universal Windows Platform Documentation.* [Online] Microsoft. [Citace: 22. 8 2018.] https://docs.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide.

27. www.nativescript.org. *NativeScript.* [Online] 2018. [Citace: 20. 8. 2018.] https://www.nativescript.org/faq.

28. Avram, Abel. www.infoq.com. *InfoQ.* [Online] 6. 3. 2015. [Citace: 20. 8. 2018.] https://www.infoq.com/news/2015/03/nativescript.

29. www.nativescript.nl. *NativeScript NL.* [Online] [Citace: 20. 8. 2018.] https://nativescript.nl/tools/nativescript-ides/.

30. www.framework7.io. *Framework 7.* [Online] 2016. [Citace: 21. 8. 2018.] https://framework7.io/.

31. www.w3schools.com. *w3schools.* [Online] [Citace: 21. 8. 2018.] https://www.w3schools.com/js/js_htmldom.asp.

32. www.developer.android.com. *Google Developers Documentation.* [Online] [Citace: 15. 7. 2019.] https://developer.android.com/reference.

33. www.visualstudio.microsoft.com. *Visual Studio 2019.* [Online] 2019. [Citace: 17. 7. 2019.] https://visualstudio.microsoft.com.

34. www.docs.microsoft.com. *Dokumentace k sadě Visual Studio.* [Online] 2019. [Citace: 17. 7. 2019.] https://docs.microsoft.com/cs-cz/visualstudio/?view=vs-2019.

35. www.jetbrains.com. *Intellij IDEA: The Java IDE for Professional Developers by JetBrains.* [Online] JetBrains s.r.o, 2019. [Citace: 17. 7. 2019.] https://www.jetbrains.com/idea/.

36. www.gsmarena.com. *gsmarena.* [Online] 2019. [Citace: 17. 7. 2019.] https://www.gsmarena.com/huawei_y6-7440.php.

37. www.applikeysolutions.com. *Applikey solutions.* [Online] 29. 6. 2018. [Citace: 18. 7. 2019.]

38. www.docs.microsoft.com. *Xamarin Documentation.* [Online] Microsoft. [Citace: 18. 7. 2019.] https://docs.microsoft.com/en-us/xamarin/.

39. Korolyov, Ven. www.itnext.io. *ITNEXT.* [Online] LINKIT, 14. 3. 2018. [Citace: 21. 7. 2019.] https://itnext.io/react-component-class-vs-stateless-component-e3797c7d23ab.

40. Kevin, Primicerio. www.github.com. *GitHub.* [Online] [Citace: 22. 7. 2019.] https://github.com/kprimice/react-native-sensor-manager.

41. docs.microsoft.com. *NuGet Documentation.* [Online] Microsoft, 24. 5. 2019. [Citace: 25. 7. 2019.] https://docs.microsoft.com/en-us/nuget/what-is-nuget.

42. www.developers.google.com. *Google Maps Platform Documentation.* [Online] Google. [Citace: 27. 7. 2019.] https://developers.google.com/maps/documentation/.

7 Seznam obrázků

8 Seznam kódů

9 Přílohy

Příloha 1 - Zadání bakalářské práce

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2018/2019

Studijní program: Aplikovaná informatika
Forma: Prezenční
Obor/komb.: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Schneider Ondřej	Pražská 1552, Náchod - Staré Město nad Metují	I1600605

TÉMA ČESKY:

Porovnání mobilních frameworků

TÉMA ANGLICKY:

Comparison of mobile frameworks

VEDOUCÍ PRÁCE:

doc. Ing. Filip Malý, Ph.D. - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Zadání: Seznámit se s oblastí mobilních aplikací a mobilních operačních systému. Představit několik frameworků pro tvorbu mobilních aplikací a porovnat přístupy ve vývoji v těchto frameworcích. Navrhnout vzorovou aplikaci demonstrující práci s mobilními aplikacemi. Vzorovou aplikaci implementovat ve vybraných (alespoň dvou) frameworcích. Porovnat práci s vybranými frameworky a jejich možnosti z pohledu vývoje v rámci vzorové aplikace.

Osnova:
1) Úvod
2) Mobilní aplikace a operační systémy
3) Přehled a porovnání frameworků pro tvorbu mobilních aplikací
4) Implementace vzorové aplikace
5) Porovnání vývoje aplikace ve vybraných frameworcích
6) Výsledky, doporučení, závěr
7) Literatura

Cíl práce: Vytvořit vzorovou mobilní aplikaci a následně na ni demonstrovat porovnání frameworků pro tvorbu mobilních aplikací z hlediska její implementace, finálního vzhledu a možností frameworků při vývoji.

SEZNAM DOPORUČENÉ LITERATURY:

Podpis studenta: ..... **Datum:** 18. 3. 2019

Podpis vedoucího práce: ..... **Datum:** 18. 3. 2019

(c) IS/STAG , Portál - Podklad kvalifikační práce , I1600605 , 18.03.2019 02:40