České vysoké učení technické v Praze
Fakulta elektrotechnická

Bakalářská práce

Virtuální naučné stezky v React Native

Šimon Maňour

Praha 2019

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Maňour** Jméno: **Šimon** Osobní číslo: **465818**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Studijní obor: **Software**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Virtuální naučné stezky v React Native

Název bakalářské práce anglicky:

Virtual natural trails in React Native

Pokyny pro vypracování:

) Seznamte se s aktuálním stavem projektu EduARd (rozšířená realita pro školy).
2) Navrhněte a implementujte klientskou aplikaci schopnou zobrazovat výuková data v návaznosti na editační systém.
3) Aplikace bude umožňovat stažení si dat pro danou naučnou stezku předem (pro použití v offline modu venku). Aplikace si bude pamatovat svůj stav i při zavření programu a bude funkční i když bude tzv. na pozadí. Implementujte všechny typy úkolů a jejich varianty, které systém EduARd podporuje.
4) V návrhu aplikace postupujte podle metodiky UCD (User Center Design).
5) Ověřte funkcionalitu a stabilitu aplikace na systému Android i iOS.

Seznam doporučené literatury:

[1] T. Lowdermilk, User-Centered Design, O'Reilly Media, 2013
[2] B. EisenmanLearning React Native: Building Native Mobile Apps with JavaScript, O'Reilly Media, 2016
[3] B. Fling, Mobile Design and Development, O'Reilly Media, 2009

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. David Sedláček, Ph.D., katedra počítačové grafiky a interakce FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **22.01.2019** Termín odevzdání bakalářské práce: **24.05.2019**

Platnost zadání bakalářské práce: **20.09.2020**

Ing. David Sedláček, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání

Podpis studenta

MAŇOUR, Šimon, *Virtuální naučné stezky v React Native*.
Praha: České vysoké učení technické v Praze, Fakulta elektrotechnická
Bakalářská práce
Vedoucí práce: Ing. David Sedláček, Ph.D.

Abstrakt

Tato bakalářská práce pojednává o tvorbě multiplatformní mobilní aplikace v React Native pro off-line procházení virtuálních naučných stezek za pomoci geolokace. První část se věnuje rozboru požadavků a analýzou aktuálního způsobu popisu stezek pomocí *XML*. Dále je pak popsán návrh aplikace a různé případy užití. V implementaci je proveden rozbor jednotlivých technologií a knihoven, které byly během vývoje použity.

Klíčová slova

React-Native, multiplatformní aplikace, virtuální naučné stezky, Expo

Abstract

This bachelor thesis describes the process of creation of multiplatform mobile application in React Native for off-line visiting of virtual educational trails using geolocation. The first part analyzes the requirements for the application and current way of defining the trails using *XML*. Then the application design and various use cases are described. The implementation analyzes the individual technologies and libraries used during the development.

Keywords

React-Native, multiplatform application, virtual educational trails, Expo

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci na téma „Virtuální naučné stezky v *React Native*“ vypracoval samostatně s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu literatury na konci práce.

V Praze dne ……. …….

(podpis autora)

Poděkování

Děkuji vedoucímu práce Ing. Davidu Sedláčkovi, Ph.D. za možnost podílet se na projektu EduARd a cenné rady týkající se implementace a tvorby mé bakalářské práce.

V Praze dne ……. …….

(podpis autora)

Obsah

1 Úvod

1.1 Motivace

Většina z nás přestala vnímat mobilní telefony pouze jako zařízení umožňující volat, či posílat zprávy. Namísto toho pro nás telefon představuje způsob, jak prohlížet web, používat různé komunikační kanály nebo jak si usnadnit cestování pomocí navigačních systémů. Mimo jiné také plní roli oblíbeného multimediálního zařízení umožňujícího přehrávání hudby, sledování filmů, či hraní her. Tyto funkcionality dohromady umožňují uživateli získávat informace v reálném čase, a to téměř o čemkoli.

Vzhledem k jejich dostupnosti, se elektronická zařízení stala naprosto samozřejmou součástí každodenního života. Dnes běžně prodávané chytré telefony jsou vybaveny vícejádrovými procesory a několika gigabyty operační paměti, a proto spíše připomínají malé počítače, které lze schovat v kapse a nosit s sebou všude kam jdeme. Této skutečnosti se snaží využít mnohé základní i střední školy ke zlepšení kvality výuky. Často tedy nakupují učební pomůcky jako jsou interaktivní tabule, tablety, či počítače do svých tříd. Roste tedy poptávka nejen po papírových učebnicích, ale i po jejich elektronických variantách, kterých je momentálně na trhu nedostatek.

Jednou z nevýhod rozšíření elektronických pomůcek je jejich podpora pasivního trávení času. Mnoho dětí dnes trpí nedostatkem pohybu a roste míra dětské obezity. Proto se nabízí myšlenka přesunout výuku ze školních lavic ven. K tomu lze právě využít elektronické pomůcky jakou jsou tablety či mobilní telefony. Tato zařízení jsou jednak skladnější než klasické učebnice a jednak nabízejí možnost pohodlného stahování studijních materiálů. Další výhodou těchto digitálních učebnic může být to, že si je kantor může sám upravovat dle svých požadavků, což u tištěných verzí dost dobře nejde.

2 Analýza problému

2.1 Cíle projektu

Cílem projektu EduARd se tedy stalo vytvořit sadu nástrojů, která by umožnila alespoň do určité míry přesunout výuku mimo školní lavice. Tyto nástroje by měly umožnit učiteli tvorbu virtuálních naučných stezek, které mohou sloužit jako alternativa běžných učebnic. Pokud by student chtěl tyto učebnice prohlížet, musel by danou naučnou stezku sám projít a navštívit všechny body zájmu. To by mělo pomáhat řešit problém nedostatečně aktivního způsobu života žáků a zároveň nabídnout způsob výuky interaktivní a zábavnou formou. Takovýto koncept aplikace by pak umožnil tvorbu nejen interaktivních studijních materiálů, ale zároveň by mohl být použit například pro hry či soutěže, kde by se žáci rozdělili do týmů, ve kterých by měli řešit nějaký problém nebo hádanku, přičemž by učitel schoval na různých místech indicie, které by vedly k jeho řešení.

Mezi potřebné nástroje patří server, který bude komunikovat s klientskými zařízeními, webovou aplikaci pro tvorbu obsahu a mobilní aplikace pro prohlížení tohoto obsahu. Autor této práce se zabýval vývojem multiplatformní aplikace v React-Native pro prohlížení 2D obsahu.

2.2 Virtuální naučené stezky

Virtuální naučnou stezku lze vnímat jako sadu několika bodů zájmu vytvořených právě učitelem. Každý z těchto bodů je určen svou geografickou lokací. Ke každému z nich pak učitel může doplnit krátkou prezentaci, která bude obsahovat různé relevantní informace pro jeho studenty. Důležitou součástí mohou být i obrázky, které poslouží jako vizuální pomůcka k obsaženému textu.

Prezentaci lze dále doplnit různými kvízy, které učitel v rámci instituce sám sestaví a které mohou otestovat získané znalosti studentů. Pedagog může vybírat z různých typů otázek, jako jsou výběr z odpovědi, seřazování různých hodnot anebo obyčejné doplnění odpovědi.

Každá stezka také může obsahovat úvodní obrazovku obsahující popisek dané stezky. Tímto způsobem lze uživateli sdělit téma stezky nebo za jakým účelem byla vytvořena. Další informací může být to, jaká místa uživatel v rámci průchodu stezky navštíví, případně jakých nových znalostí díky ní nabyde. Na úvodní obrazovce také může být údaj o tom, kdo danou stezku vytvořil, případně kdo její tvorbu financoval.

2.3 Popis mobilní aplikace pro studenty

Uživatelé aplikace – tedy studenti – pak budou mít k dispozici mobilní aplikaci, kterou mohou spustit na svém telefonu či tabletu. Uživatel si v této aplikaci může zobrazit všechny dostupné stezky, které si může stáhnout. Výběrem jedné z nich pak dojde ke stažení veškerých potřebných dat přímo do úložiště zařízení. Takto stažené stezky aplikace nabídne uživateli k procházení. Jednu z nich si pak může vybrat a zařízení ji zobrazí na mapě. Poté může uživatel navštěvovat jednotlivé body zájmu, prohlížet prezentace a vyplňovat otázky. Nepotřebné stezky lze ze zařízení smazat.

2.3.1 Stahovaní dat

Stahovaná data v sobě zahrnují nejen podklady vytvořené učitelem, ale i podrobnou mapu okolí jednotlivých bodů zájmu. Toto umožní používání aplikace bez neustálého přístupu k internetu, což je určitě výhodné pro uživatele, kteří mají omezený datový limit, nebo kteří vůbec nemají možnost mobilního připojení k internetu (např. tablety).

2.3.2 Prohlížení stezek

Ve svém zařízení mohou mít studenti stažených několik stezek a vždy si mohou vybrat jednu jako aktivní. Body zájmu této zvolené stezky se pak promítnou uživateli do mapy přímo v aplikaci. Ta bude zároveň pomocí GPS sledovat polohu uživatele, kterou zobrazí na mapě kvůli snazší orientaci. Pokud se uživatel přiblíží dostatečně blízko k nějakému bodu zájmu, systém tuto událost detekuje opět pomocí GPS souřadnic a nabídne uživateli prohlížení prezentace spojené s daným bodem. Pokud je tato prezentace zároveň zakončena kvízem, aplikace umožní kvíz vyplnit a odpovědi studenta sama vyhodnotí. Cílem studenta tedy bude navštívit všechny uvedené body zájmu, přečíst si informace týkající se daného místa a následně správně odpovědět na všechny otázky.

2.3.3 Mazání dat

Z důvodu šetření místa v úložišti zařízení je možno již splněné nebo nepoužívané stezky a jejich data z paměti smazat. Mezi tato data především patří mapy, obrázky a soubory obsahující definice stezek. Nicméně informace o již dosažených bodech a vypracovaných kvízech zůstanou uloženy, a je možné smazanou stezku kdykoli stáhnout znovu a dokončit.

2.3.4 Multiplatformnost

Vzhledem k existenci více mobilních platforem je pak potřeba, aby byla aplikace spustitelná na co největším množství mobilních zařízení. V tomto případě byl měla být podpora pro dva nejčastěji užívané operační systémy, tedy iOS a Android.

2.4 Analýza existujících řešení

Jedním z dostupných produktů na českém trhu je aplikace Geofun (1), která se podobně jako tento projekt zabývá zobrazováním virtuálních naučných stezek uživateli. Je dostupná na telefony typu Android i iOS, rovněž také podporuje off-line stahování materiálů. Jedna z jejích hlavních funkcí je, že si uživatelé mohou sami vytvářet a sdílet vlastní naučné stezky.

Rozdílem Geofun oproti této aplikaci je právě její cílení na širokou veřejnost. Projekt EduARd se především zaměřuje na školy a vzdělávací zařízení, přičemž vytvořené materiály budou dostupné pouze studentům dané instituce, a pro přístup k nim bude vyžadována autorizace.

Dále pak existují naučné stezky s informačními tabulkami, na kterých je natištěn QR kód, jehož přečtením je uživatel typicky přenesen na webovou stránku s informacemi. Hlavní nevýhodou tohoto řešení je fakt, že vyznačení bodu zájmu by vyžadovalo nalepení QR kódu na daný objekt, což je často neproveditelné, případně nezákonné. Tuto formu naučných stezek používá například národní park České Švýcarsko. (2)

2.5 Komunikace se serverem

V rámci projektu EduARd probíhá také vývoj serveru, se kterým by měla mobilní aplikace komunikovat. Na implementaci serveru se autor práce nepodílel. Posíláním požadavků na koncový bod */api/book* bude aplikace získávat seznam dostupných stezek. Pro stažení informací o stezce s identifikátorem *x* je potřeba zavolat koncový bod */api/book/x*. Během stahování poskytne server aplikaci definice stezky a obrazové materiály. Mapy tento server poskytovat nebude.

Oproti serveru se aplikace autentizuje pomocí dříve vygenerovaného klíče. Vytvoření tohoto klíče zajišťuje koncový bod */api/Auth/ApiKey*

2.6 Zadaná definice stezek ve formátu XML

Každá virtuální naučná stezka je definovaná souborem ve formátu XML. Struktura tohoto souboru byla definována již dříve v rámci jiného projektu. Tím pádem může soubor obsahovat prvky či atributy, se kterými aplikace nepracuje. XML dokument je opět poskytován serverem, konkrétně koncovým bodem */api/Books/Download/{fileName}*.

Pro stažení map je potřeba dokument přečíst a vyhledat v něm GPS lokace všech bodů zájmu. Taktéž z něj budou čerpána data pro zobrazování prezentací a vykreslování kvízů.

2.6.1 Prvek Questionset

Kořenový prvek se nazývá *questionset*. Tento prvek reprezentuje celou virtuální naučnou stezku. Obsahuje atribut *maxTries*, jehož hodnota je celé kladné číslo. Tak je určen počet pokusů, které má uživatel na vypracování každého kvízu.

2.6.2 Prvek Task

Element *Task* zastupuje jeden bod zájmu. Má atributy *name* a nepovinný atribut *title*. *Name* obsahuje identifikátor bodu zájmu a je určen spíše pro programátora. *Title* pak určuje jméno daného bodu, které je prezentováno uživateli.

Mezi potomky prvku *Task* patří prvek *location*. Ten se větví na další dva elementy – *latitude* a *longitude*. Ty už se dále nevětví, ale mají hodnotu GPS souřadnic. Společně

tedy jednoznačně určují polohu daného bodu zájmu na mapě. Pokud mají *latitude* a *longitude* hodnotu rovnou nule, znamená to, že se jedná o úvodní obrazovku a není potřeba je vykreslovat na mapě.

2.6.3 Prvek Questionslide

Mezi potomky elementu *Task* dále patří libovolný počet prvků typu *questionslide*. Tak se označuje element, který reprezentuje jednu obrazovku prezentace daného bodu zájmu. V dokumentu XML jsou seřazeny v pořadí, v jakém si je bude uživatel prohlížet na mobilním zařízení.

Questionslide má také několik atributů, nicméně jejich význam či hodnota nejsou ještě ustálené a aplikace s nimi prozatím nepracuje. Mezi potomky tohoto elementu patří také *description* a *images*, které se už dále nevětví. Prvek *description* obsahuje text, který má být zobrazen na daném snímku. Hodnota *images* má tvar identifikátoru obrázku sloužícího jako doprovodný podklad k textu.

Dalším potomkem pak může být element *questions*. Jeho přítomnost ukazuje, že daný snímek obsahuje kvíz. Každý prvek typu *questions* obsahuje alespoň jeden element *question,* který reprezentuje jednu kvízovou otázku. Atribut *type* pak programátorovi říká, o jaký typ otázky se jedná.

Některé prvky *questionslide* žádné potomky nemají.

2.6.4 Otázka typu Sort

Otázka typu Sort označuje seznam, který musí uživatel seřadit. Obsahuje element *description,* v němž je uloženo zadání otázky. Dalšími potomky jsou pak prvky typu *variant*. Každá varianta zde reprezentuje jednu položku seznamu, který musí uživatel seřadit.

V dokumentu XML jsou vypsány varianty za sebou již v tom správném pořadí. S tím souvisí atribut *shuffle*, který označuje, zda se má seznam při prvním zobrazení zamíchat. Je tedy žádoucí, aby měl hodnotu *true*. Kdyby tomu tak nebylo, tak při spuštění kvízu budou možnosti již ve správném pořadí.

2.6.5 Otázka typu Singlechoice

Atribut *type* s hodnotou *singlechoice* označuje otázku s výběrem odpovědi. Znění otázky se nachází uvnitř elementu *description*. Otázka dále obsahuje několik prvků *variant,* označujících jednotlivé možné odpovědi. Identifikátory případných doprovodných obrázků jsou obsaženy v prvcích *images.*

Každý element *variant* obsahuje atribut *valid*. Pokud má hodnotu *true,* jedná se o správnou odpověď, v opačném případě se jedná o chybnou. Hodnotu *true* má vždy jedna varianta, vzhledem k tomu, že otázka má právě jednu správnou odpověď.

Atribut *shuffle* určuje, zda je před zobrazením potřeba jednotlivé možnosti zamíchat.

2.6.6 Otázka typu Intervalquestion

Během vyplňování otázky *intervalquestion,* uživatel vybírá jednu číselnou hodnotu z určeného intervalu. Element *description* opět obsahuje zadání otázky, *variant* obsahuje správnou odpověď.

Otázka dále obsahuje jeden až dva prvky typu *interval*, které mají atribut *valid,* nabývající hodnot *true* a *false*. *Interval* se také větví na dva potomky, elementy *start,* obsahující začátek intervalu a *end,* který označuje jeho konec. Pokud má *valid* hodnotu *false,* jedná se o interval odpovědí, ze kterých bude moci uživatel vybírat. Hodnota *true* znamená, že odpovědi z tohoto intervalu se počítají jako správné. V případě, že interval s hodnotou *true* atributu *valid* chybí, uživatel musí odpovědět přesně.

Poslední potomek nese jméno *step.* Ten určuje, jak daleko od sebe budou čísla z intervalu hodnot, která budou nabídnuta uživateli jako možné odpovědi.

2.6.7 Otázka typu Togglebuttonsgrid

Jedná se o otázku, která zahrnuje výčet tvrzení. U každého z nich uživatel vybere jednu z nabízených možností, tak aby byl výrok pravdivý. Element *description* obsahuje zadání otázky. Jednotlivé možné odpovědi jsou definovány pomocí prvků *option.* Obsahem těchto prvků je vždy jedna z odpovědí. Dále má každý atribut *id,* který slouží jako jeho unikátní identifikátor.

Dalšími potomky jsou elementy typu *variant*. Každý z nich reprezentuje jedno

tvrzení, na které je potřeba odpovědět. Správná odpověď je určena identifikátorem možnosti, který je uložen v atributu *option.*

2.6.8 Otázka typu Numberquestion

V této otázce se po uživateli žádá, aby sám vyplnil odpověď, kterou je celé číslo. Zadání otázky se nachází uvnitř prvku *description.* Dalším potomkem je *variant,* který obsahuje správnou odpověď.

2.6.9 Otázka typu Filltext

Filltext se strukturou velmi podobá *numberquestion*. Opět je uživatel vyzván, aby sám doplnil odpověď. V tomto případě se ale jedná o libovolný text. Znění otázky formuluje *description.* Dalšími potomky je jeden či více elementů *variant*, které obsahují všechny přijatelné odpovědi. To reflektuje fakt, že se jedná o poměrně abstraktní formu kvízu a správných odpovědí může být více. Otázka má dále atribut *caseSensitive*, který určuje, zda se má při validaci brát zřetel na velká a malá písmena, či nikoli.

2.6.10 Otázka typu Multichoice

Zadání *multichoice* otázky se strukturou podobá typu *singlechoice*. Její znění je uloženo v elementu *description*, a jednotlivé odpovědi jsou reprezentovány potomky typu *variant.* Každý z nich představuje jednu možnou odpověď. Pokud má atribut *valid* hodnotu *true,* jedná se o správnou odpověď, v případě *false* se jedná o odpověď chybnou.

Počet správných a chybných odpovědí není přesně určen. Tím pádem mohou být správně všechny, ale také žádná. Atribut *shuffle* u elementu *question* určuje, zda je nutné před zobrazením kvízu uživateli otázky zamíchat.

2.6.11 Otázka typu Dragtoline

Otázka *dragtoline* se skládá ze dvou seznamů obsahujících různé výrazy. Tyto seznamy mají stejný počet složek. Úkolem uživatele se pak stává přiřadit k výrazům z prvního seznamu související výrazy z toho druhého.

Zadání otázky se nachází v XML prvku *description.* Dalším potomkem otázky *dragtoline* je prvek *options*. Ten se větví na určitý počet elementů typu *option*. Každý *option* zastupuje jednu položku prvního seznamu. Také obsahuje atribut *id*, který slouží

jako jeho identifikátor.

Poslední typ potomků se nazývá *variant*, který reprezentuje složky druhého seznamu. Tyto prvky rovněž obsahují atribut *option*, který nese hodnotu identifikátoru některého z *option,* ke kterému patří.

Pokud je atribut *shuffle* u otázky nastaven na *true,* je potřeba před zobrazením položky *variant* náhodně zamíchat.

2.6.12 Otázka typu Dragtomiddle

Typ otázky *dragtomiddle* strukturou připomíná *dragtoline*. Skládá se ze dvou seznamů, přičemž první z nich obsahuje obrázky, druhý je tvořen textovými popisky. Uživatel má pak na starost přiřadit ke každému obrázku správný popisek.

Znění otázky je obsaženo uvnitř elementu *description*. Dále se otázka větví na prvek *options*, a určité množství prvků typu *variant*. *Options* pak obsahuje stejné množství elementů *option*.

Jediným potomkem *option* je *image*, ve kterém je uložen identifikátor obrázku, jenž má být zobrazen. Každý *option* má unikátní atribut *id*. Položky *variant* obsahují text, který je potřeba přiřadit k obrázkům. K jakému obrázku popisek patří, lze zjistit z atributu *option,* který nese hodnotu *id* správného obrázku.

Atribut *shuffle* ukazuje, zda je nutno zobrazovat možnosti v náhodném pořadí.

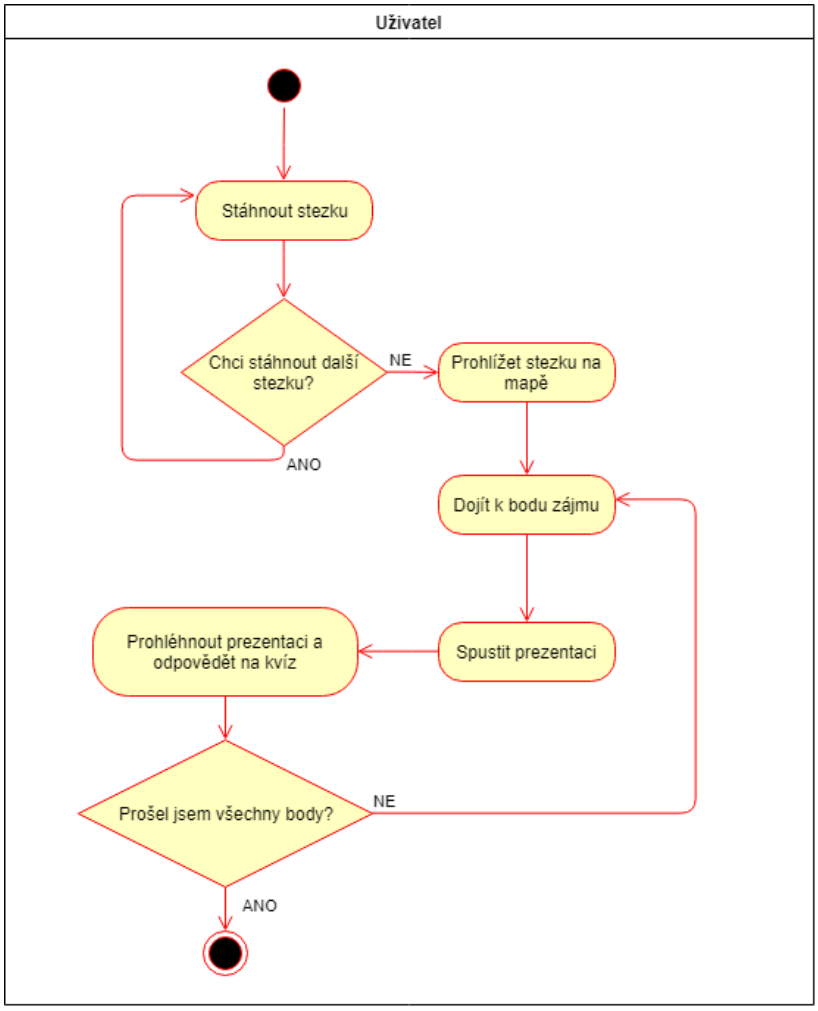
3 Návrh aplikace

Tato kapitola se bude zabývat návrhem jednotlivých scén v aplikaci a možnými případy užití. Vzhledem k velkému množství možných klientských zařízeních používajících různé operační systémy, zaměřili jsme se na jednoduchost návrhu a uchopitelnost ze strany uživatelů.

V rámci vývoje aplikace nejprve vznikl jednoduchý návrh, který popisuje některé scény a jejich funkcionality. Některé z nich nicméně byly v období implementace upraveny z důvodu lepšího ovládání, jiné vznikaly až přímo během implementace, vzhledem k tomu, že docházelo k úpravám XML definic.

Vzhledem k velkému množství typů kvízových otázek, podstatná část návrhu se zabývá právě popisem jejích zamýšleného vzhledu a chování. Každá otázka totiž vyžaduje vlastní obrazovku a jejich struktura je často velmi odlišná. Tím pádem obrazovky s otázkami tvoří více než polovinu celkového počtu scén.

Obrázek 1 popisuje průchod aplikací ze strany uživatele a byly podle něho modelovány jednotlivé scény.



Obrázek 1 - Diagram aktivit uživatele

3.1 Hlavní scéna

Poté, co uživatel spustí aplikaci, zobrazí se mu úvodní obrazovka. Ta bude obsahovat v záhlaví název aplikace a její tělo se bude skládat ze dvou tlačítek. První bude mít popisek „Moje stezky“, druhé „Správce stezek“. Stisknutím jednoho z nich bude uživatel přenesen na další scénu.

Pokud by uživatel chtěl prohlížet některou ze stažených stezek, může kliknout na tlačítko „Moje stezky“. „Správce stezek“ umožňuje získat nové stezky z *API*, případně mazat ty nepotřebné. Obrázek 2 ilustruje možný vzhled úvodní scény.

Obrázek 2 - Návrh úvodní scény

3.2 Správce stezek

Tato scéna slouží uživateli ke stahování a mazání virtuálních naučných stezek. Při jejím spuštění se systém pokusí stáhnout seznam dostupných stezek z API, pokud to není možné, např. zařízení nemá přístup k internetu, bude uživatel varován, že seznam nebylo možno aktualizovat.

Po úspěšném načtení informací o dostupných stezkách aplikace zobrazí uživateli dva seznamy. První bude obsahovat stezky již stažené v zařízení, druhý stezky dostupné, které staženy nejsou. Uživatel může kliknout na libovolnou stezku z některého seznamu, tímto se přesune na další aktivitu obsahující informace o dané stezce. Návrh této scény zachycuje Obrázek 3.

Obrázek 3 - Návrh správce stezek

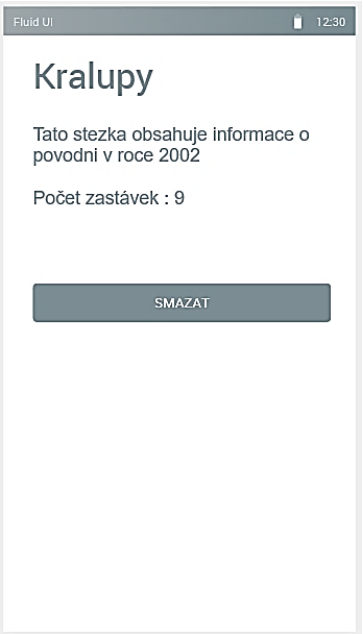
3.3 Správce konkrétní stezky

Aktivita správce stezek je vždy vyvolána s argumentem obsahujícím identifikátor konkrétní stezky. Při jejím spuštění se systém pokusí pomocí API stáhnout informace o dané stezce. Poté, co je stahování dokončeno, zobrazí se uživateli detaily této stezky. Mezi ně může patřit stručný popis stezky, případně počet bodů zájmu. Poté může uživatel stisknout tlačítko „Stáhnout“, čímž zahájí stahování map a jiných obrazových materiálů do úložiště zařízení. Uživateli se zobrazí informace o stavu stahování: množství souborů, které je ještě potřeba stáhnout, případně zpráva o chybě.

Pokud je stezka již stažena v zařízení, může uživatel stisknout tlačítko „Odstranit“, čímž dojde ke smazání všech dat, které jsou vázány na tuto stezku.

Jestliže jakýkoli požadavek není splněn, je o této události uživatel informován pomocí notifikace.

Stisknutím zpětné šipky se uživatel přesune zpět na okno „Správce stezek“. Zde může opět stáhnout či odstranit další stezku anebo se opětovným stisknutím tlačítka „zpět“ vrátit na úvodní obrazovku. Vzhled této obrazovky popisuje Obrázek 4.



Obrázek 4 - Návrh správce stezky

3.4 Moje stezky

Hlavním prvkem této scény je seznam, obsahující jména všech dostupných stezek, jejichž data byla stažena na úložiště zařízení. Vybráním jedné z položek tohoto seznamu se uživatel přesune na další aktivitu sloužící k procházení zvolené stezky.

Pokud má stezka definovanou úvodní obrazovku ještě před zobrazením mapy je uživateli nabídnuta krátká prezentace obsahující snímky této obrazovky. Po jejím zhlédnutí pak může uživatel přejít k procházení mapy. Původní návrh je zachycen na Obrázek 5.

Obrázek 5 - Návrh scény "Moje Stezky"

3.5 Prohlížení map

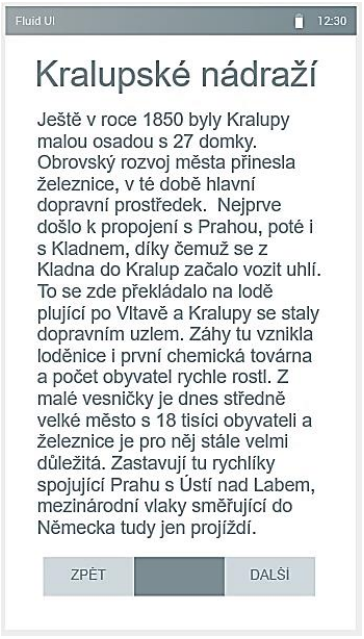
Scéna mapy se spouští s argumentem obsahujícím informace o konkrétní stezce. Obsahuje v záhlaví název této stezky. Zbytek scény zabírá mapa, na které se stezka zobrazuje jako sada bodů zájmu. Dále na je na mapě také vidět poloha uživatele.

Pokud uživatel klikne na nějaký bod zájmu a nachází se dostatečně blízko, spustí se prezentace svázaná s tímto bodem. Pokud je příliš daleko, objeví se pouze stručný popisek daného bodu spolu s informací, že musí vzdálenost od něj zredukovat. Návrh této obrazovky se nachází na Obrázek 5

Obrázek 6 - Návrh prohlížeče map

3.6 Prezentace bodu zájmu

Tato scéna se opět spouští s argumentem obsahujícím informaci o stezce a bodu zájmu v rámci dané stezky. Prezentace je složena z několika obrazovek. K přepínání mezi nimi slouží tlačítka „Předchozí“ a „Další“ v dolní části displaye. Pokud se uživatel nachází na první obrazovce, tlačítko „Předchozí“ bude neaktivní. To samé platí pro tlačítko „Další“, pokud se uživatel nachází na poslední obrazovce. Každá obrazovka obsahuje informace k souvisejícímu bodu zájmu, případně doprovodné obrázky. V záhlaví scény zároveň obsahuje název bodu zájmu a informaci, kterou obrazovku z kolika možných uživatel právě prohlíží. Návrh vzhledu této scény popisuje Obrázek 7.



Obrázek 7 - Návrh scény prezentace

3.7 Scény s kvízem

Obrazovka také může obsahovat kvíz. Kvízů může být několik druhů, ale určité vlastnosti budou mít společné. Obrazovka vždy bude v horní části obsahovat popisek kvízu a zadání otázky, pod kterým bude už kvíz samotný. Ve spodní části se bude nacházet tlačítko „Potvrdit“, které uživatel může stisknout pro validaci odpovědí. Pokud odpověděl chybně a má ještě více než jeden pokus, dostane pouze zprávu o tom, že odpověď není zcela správně. Pokud odpověděl správně nebo nemá žádné volné pokusy, aplikace odpověď zkontroluje a zobrazí uživateli správnou odpověď.

3.7.1 Kvíz Sort

Tělo otázky typu *sort* se skládá z jednoho vertikálně orientovaného seznamu dlaždic. Pokud uživatel jednu z nich stiskne, bude pak moci měnit pořadí položek seznamu tím, že ji přetáhne na jiné místo.

Při validaci systém obarví položky na správném indexu zeleně, ostatním nastaví barvu na červenou. Dále ke každé dlaždici doplní číslo, které reprezentuje správný index dané dlaždice.

3.7.2 Kvíz Singlechoice

Zde se jedná o obyčejný výběr z odpovědi, přičemž právě jedna odpověď je správná. Tělo otázky tvoří vertikální list tlačítek s popisky. Pokud uživatel jedno z nich stiskne, toto tlačítko se pak označí jako zvolené. Všechna ostatní se stanou neaktivními.

Pokud odpověděl uživatel správně, během validace bude jeho odpověď označena zelenou barvou. Pokud odpověděl chybně, zvolená odpověď se obarví červeně a správná zeleně.

3.7.3 Kvíz Intervalquestion

Zde uživatel vybírá konkrétní číslo z určeného intervalu. Systém zobrazí uživateli pod zadáním otázky číselné kolo obsahující možné odpovědi. Uživatel vybere tažením nahoru a dolů odpověď, o které si myslí, že je správná.

Po stisknutí tlačítka pro validaci, systém zobrazí textové okénko obsahující dvě informace: zvolenou hodnotu a správnou hodnotu. Pokud se uživatelova odpověď nachází v tolerovaném intervalu, je okno obarveno zeleně, v opačném případě se jeho pozadí nastaví na červenou.

3.7.4 Kvíz Togglebuttongrid

Obrazovka obsahující tuto otázku bude mít vzhled tabulky. Uvnitř první buňky každého řádku se bude nacházet text reprezentující jedno tvrzení, které je nutno klasifikovat. Ostatní buňky budou potom mít vzhled zaškrtávacích tlačítek, přičemž uživatel vždy jedno stisknutím označí.

Po validaci se tlačítka deaktivují a systém vyhodnotí správnost odpovědí. Pokud

uživatel odpověděl správně, tlačítko se obarví zeleně, v případě nesprávné odpovědi se zvolené tlačítko obarví červeně a správná varianta dostane zelenou barvu.

3.7.5 Kvíz Nuberquestion

V případě otázky *nuberquestion* je uživatel vyzván, aby sám doplnil číselnou odpověď. Klíčovým prvkem se stává textové pole, se zastupující hodnotou „Zadejte číslo“, do kterého uživatel vyplní svou odpověď.

Po ověření správnosti odpovědi, systém zobrazí vyplněné číslo a zároveň správnou odpověď. Pokud jsou si tyto dvě hodnoty rovny, je pozadí textu obarveno zeleně, jinak nabývá červené barvy.

3.7.6 Kvíz Filltext

Scéna vzhledově odpovídá *numberquestion*. Rovněž obsahuje textové pole s instrukcí „Zadejte odpověď“, do kterého může uživatel napsat svoji odpověď.

Ve chvíli, kdy uživatel svou odpověď potvrdí, systém zobrazí nejprve zadaný textový řetězec, ale také seznam všech přijatelných odpovědí. Pokud se alespoň jedna z nich shoduje se zadaným řetězcem, obarví se textová oblast zeleně. V případě nesprávné odpovědi se oblast obarví červeně.

3.7.7 Kvíz Multichoice

Tento kvíz má vzhled výběru z odpovědi, nicméně s variabilním počtem pravdivých a nepravdivých odpovědí. Tělo otázky pak tvoří vertikální seznam zaškrtávacích tlačítek s popisky. Uživatel může odpovědi, o kterých si myslí že jsou správně, označit stisknutím příslušného tlačítka. Opětovným stisknutím se pak výběr zruší.

Poté, co proběhne validace, správně zvolená pole jsou vybarvena zeleně, ta chybná se vybarví červeně. Za chybu se považuje, pokud je odpověď správně a nebyla zaškrtnuta, nebo pokud odpověď zaškrtnuta je i když správně není.

3.7.8 Kvíz Dragtoline

Smyslem tohoto cvičení je přiřadit k sobě korespondující prvky dvou seznamů. Původně vzniklo několik návrhů na jeho vzhled. V prvním případě se jednalo o dva vertikální listy

vedle sebe, přičemž by uživatel měnil pořadí prvků tažením. Zde ale bylo problematické nastavovat výšku jednotlivých položek tak, aby se listy nacházely přesně vedle sebe.

Další návrh byl založen na spojování pomocí barev. Jeho myšlenka spočívala v tom, že by každý prvek levého listu byl jinak obarven. Prvky patřící k sobě by pak uživatel spojoval přiřazením stejné barvy. Toto by však mohlo být nepřehledné v případě delších listů a zároveň problematické pro uživatele se zhoršenou schopností vnímaní barev.

Pro zatím zvolenou možností se staly dva seznamy tlačítek, přičemž uživatel stisknutím označuje položky pravého seznamu. Kliknutím na prvek levého seznamu dojde z přiřazení označené položky k tomuto prvku. Cílem uživatele se stává, aby se tlačítka obsahující korespondující výrazy nacházela vedle sebe.

Po provedení validace není již možno s prvky manipulovat. Ty, které jsou přiřazeny správně se obarví zeleně, chybné červeně.

3.7.9 Kvíz Dragtomiddle

Kvíz se strukturou i zadáním podobá d*ragtoline* a jejich návrh byl vytvářen společně. I zde je potřeba přiřazovat položky jednoho seznamu ke druhému. V tomto případě jsou prvky levého listu obrázky. K obrázkům pak uživatel přiřazuje textové popisky z pravého seznamu.

Kliknutím na popisek dojde k jeho označení modrou barvou, který je pak možno druhým kliknutím přiřadit k obrázku. Tímto dojde k prohození dvou položek seznamu vpravo.

Po kontrole se celý řádek obarví zeleně, pokud k sobě dané prvky patří, nebo červeně, pokud jsou přiřazeny chybně.

4 Popis implementace

4.1 Volba technologie

Úkolem autora bylo implementovat mobilní aplikaci pro správu a prohlížení stezek. Prvním problémem, který bylo potřeba vyřešit se stala otázka multiplatformnosti. Tu je možné řešit dvěma způsoby. Buďto psát dvě oddělené aplikace, jednu pro iOS, druhou pro Android, nebo zvolit takovou technologii, která umožňuje tvorbu programů pro obě platformy zároveň. Bylo rozhodnuto pro druhou variantu a použitou knihovnou se stal *React-Native* (3), který je i součástí zadání.

4.2 Node.js

Základním stavebním kamenem pro vývoj v *React-Native* je *Node.js* (4). Jedná se o knihovnu, rozšiřující funkcionalitu skriptovacího jazyka *JavaScript,* který původně sloužil především jako nástroj, jak vložit logiku do jinak statických webových stránek a učinit je interaktivními. Tímto způsobem byl interpretován webovým prohlížečem a vykonával se na straně uživatele.

Node.js umožňuje vývojáři psát plnohodnotné aplikace čistě pomocí *JavaScriptu*. Ten se překládá na rychlejší strojový kód pomocí *V8 Engine,* prostředí napsaného v C++, zajišťujícího běh celé aplikace. (5)

4.2.1 Asychnronní operace

Další velkou výhodou je podpora asynchronních operací. To umožňuje jakýkoli blok kódu, jehož vykonání by mohlo být časově náročné, učinit neblokujícím. Jedná se zpravidla o operace obsahující čtení nebo zápis na disk, přístup k databázi, nebo posílání http požadavků. Při volání asynchronní operace je vytvořen objekt typu *Promise (slib),* který spravuje stav dané operace. Díky tomu je možno vykonávat další příkazy, mezi tím, než je daný požadavek obsloužen. Tímto způsobem je pak možno zpracovávat několik požadavků najednou, bez zbytečného čekání.

Asynchronní operace se v *JavaScriptu* značí klíčovým slovem *async* a lze na ni

počkat pomocí příkazu *await*. Vyvíjená mobilní aplikace využívá asynchronních operací velmi často, a to především pro manipulaci s daty, síťovou komunikaci a geolokační služby.

4.2.2 Node Package Manager

Takto se nazývá správce balíčků (modulů) pro *Node.js,* ačkoli většinou se používá pouze zkratka *npm* (6). Pomocí něho lze spravovat knihovny pro použití v daném projektu. Názvy potřebných modulů jsou vždy uloženy v souboru *package.json,* jejich implementace pak ve složce *node_modules.*

Všechny použité knihovny, včetně balíčků *React-Native* a *Expo* byly instalovány pomocí tohoto nástroje.

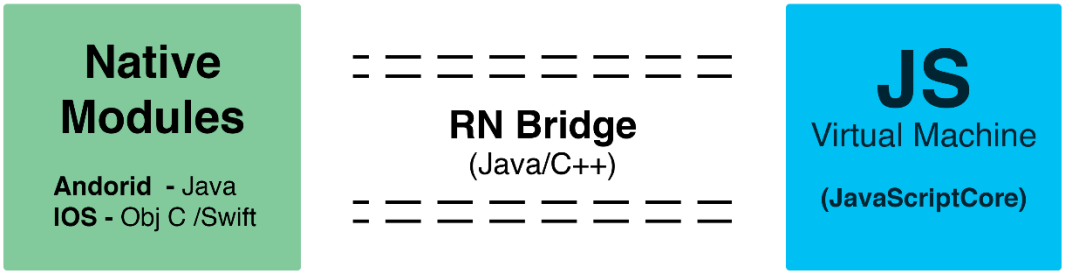
4.3 React-Native

React-Native se nazývá sada *JavaScriptových* nástrojů pro tvorbu nativních mobilních aplikací. Je založen na knihovně *React* od společnosti Facebook, která slouží ke tvorbě uživatelských rozhraní pro webové aplikace.

Hlavním důvodem, proč o aplikacích hovoříme jako nativních, je způsob vykreslování komponent. Komponenty používané v *JavaScriptu* uvnitř volají přímo aplikační rozhraní dané mobilní platformy. *React-Native* pak slouží jako most mezi naší aplikací v *JavaScriptu* a vykreslováním nativních komponent psaných v programovacím jazyce *Java* pro platformu Android a *Objective-C* pro iOS.

Tímto se *React-Native* odlišuje od dalších multiplatformních knihoven jako je *Apache Cordova* nebo *Ionic*, jelikož k zobrazování uživatelského rozhraní využívá vlastní komponenty dané mobilní platformy namísto vykreslování pomocí webového prohlížeče. (7)

Části *JavaScriptu*, které není možné převést na nativní objekty dané platformy jsou vykonávány pomocí virtuálního stroje, který používá *JavaScriptCore*. (8) Jedná se o interpret, využívaný prohlížečem Safari. Do aplikací pro zařízení typu Android, která nemají Safari ve své základní výbavě se *JavaScriptCore* přidává během kompilace. Komunikaci mezi nativním vláknem a virtuálním strojem zajišťuje *React-Native Bridge* (Obrázek 8).



Obrázek 8 - Struktura aplikace v React-Native (9)

4.4 Nedostatky obyčejného React-Native

Ačkoli je React-Native efektivní nástroj pro tvorbu mobilních aplikací, v rámci vývoje byly nalezeny některé nedostatky. V prvním případě se jednalo a silnou závislost na externích knihovnách, které nejsou vždy velmi dobře zdokumentovány a jejich instalace občas bývá poměrně náročná. V druhém případě tím byl fakt, že není možné kompilovat aplikaci pro iOS z jiného stroje, než je Mac.

Tyto dva problémy se staly hlavním důvodem, proč bylo potřeba vyhledat jiné vývojové prostředí, než je *react-native-cli*.

4.4.1 Použití externích knihoven

Ačkoli *React-Native* obsahuje mnoho prvků využívajících nativní mobilní komponenty, stále existují funkcionality, které základní knihovna nepodporuje. Mezi ty hlavní, bez kterých by tato aplikace nemohla fungovat, patří přístup na úložiště zařízení a zobrazování map. Další užitečná rozhraní pak mohou umožňovat animace komponent nebo ovládání pomocí gest.

Tyto funkce bývají typicky implementovány jako součást knihoven třetích stran. Některé z nich lze jednoduše nainstalovat jediným příkazem v terminálu pomocí *npm*. Nicméně ty, které využívají nějakou nativní funkci či komponentu je potřeba tzv. *linkovat*. (10)

Linkování znamená, že je pro správné načtení komponenty potřeba upravit konfigurační soubory jednotlivých platforem. Pro iOS je nutné přidat závislosti do souboru *ios/Podfile*. V rámci zprovoznění knihovny na zařízení Android je potřeba upravit soubory *gradle*, *AndroidManifest.xml* a přidat balíček v *MainApplication.java*.

Nicméně v některých případech jsou návody zastaralé nebo nekompatibilní s novými verzemi *Gradle* a je potřeba hledat postup v jiných zdrojích.

4.4.2 Nemožnost kompilace pro iOS

Základní *React-Native* pro operační systém Windows nebo Linux standardně nepodporuje kompilaci aplikací na zařízení iOS. To by znamenalo, že jakékoli testování aplikace by mohlo probíhat pouze na telefonech či emulátorech s operačním systémem Android a tím pádem by nešlo prohlásit, že aplikace je skutečně multiplatformní.

Jedním z řešení může být například projekt *Expo*. Díky jeho vývojovému serveru je možné testovat jak lokálně v emulátoru, tak i na fyzických zařízeních. Tímto způsobem pak lze spustit aplikaci i na fyzickém zařízení iOS, což základní React-Native neumožňuje. (11)

4.5 Spravované vývojové prostředí Expo

Expo je společný název pro dva programy. V prvé řadě se jedná o mobilní aplikaci, která slouží ke spouštění a testování našeho projektu. Lze ji zdarma stáhnout z *Google Play* nebo *AppStore,* v závislosti na použitém mobilním zařízení. Pomocí ní lze pak spustit jakýkoli *Expo* projekt buďto zadáním adresy, nebo naskenováním QR kódu (12).

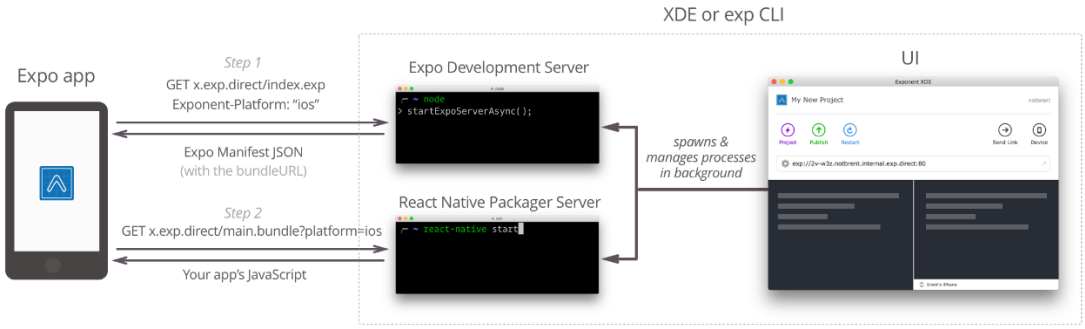
V dalším případě můžeme hovořit o rozhraní pro příkazový řádek, které se instaluje na počítač pomocí *npm*. Nový projekt pak lze vytvořit příkazem *expo init*, který vytvoří projekt v nové složce a stáhne do něj všechny potřebné moduly. Spustit ho můžeme zadáním *expo start*. Tento příkaz spustí vývojový server, který slouží ke komunikaci s aplikací Expo na mobilním zařízení. Server také slouží ke kompilaci naší aplikace a nahrávání všech nezbytných souborů na mobilní zařízení. Zároveň slouží jako konzole pro zobrazování výpisů z aplikace pomocí *console.log*.

V poslední řadě *Expo* obsahuje sadu nástrojů rozšiřujících základní *React-Native*, ve formě *npm* modulu. Tato knihovna obsahuje balíčky pro práci se souborovým systémem zařízení a přístup k mnoha funkcionalitám zařízení, jako jsou mapy, navigace, či fotoaparát. Dále také podporuje vytváření procesů na pozadí a posílání notifikací. Velkým bonusem je pak fakt, že se zpravidla jedná o nativní moduly, které jsou poskytovány třetími stranami. Tyto moduly je pak často nutné *linkovat*, což *Expo* řeší

předem za nás.

Možnou nevýhodou programu *Expo* může být skutečnost, že některé nativní komponenty ještě nejsou podporovány. Pokud by tyto komponenty vyžadovaly *linkování*, tak je v našem projektu nelze použít. Přesto je však projekt *Expo* aktualizován téměř každý měsíc a počet podporovaných knihoven stabilně stoupá.

Vzhledem k tomu, že všechny nezbytné knihovny *Expo* podporuje, bylo toto prostředí použito pro vývoj mobilní aplikace, především kvůli možnosti testování i na platformě iOS. Obrázek 9 ilustruje průběh vývoje pomocí nástroje *Expo*.



Obrázek 9 - Průběh vývoje ve spravovaném prostředí (13)

4.6 Komponenty v prostředí React

Vizuální prvky aplikace v *React-Native* jsou složeny z komponent. Ty můžeme vytvářet v kódu pomocí syntaxe *JSX*, což je zkratka pro *JavaScript Syntax Transformers*. Tato speciální syntaxe, kombinující *JavaScript* a značkovací jazyky, nám umožňuje vytvářet komponenty pomocí *tagů*, podobně jako v *XML*. Stejně jako v případě XML je lze dále větvit pomocí vytváření potomků nebo nastavovat jejich atributy. (14)

Atributem společným pro většinu komponent je *style*, který umožňuje měnit vzhled jejich vzhled. Ten může být definován přímo při zakládání komponenty, anebo může být předáván v rámci dříve založené proměnné. Vždy se ale jedná o obyčejný *JavaScriptový* objekt, který obsahuje páry typu „klíč a hodnota“. Jednotlivé položky svými názvy sice připomínají kaskádní styly, nicméně zde se styl vždy vztahuje pouze k jednomu objektu a není možno jednotlivé styly dědit nebo definovat závislosti na předcích.

4.6.1 Životní cyklus komponent

Komponenty v prostředí *React* mohou být buďto třídy nebo funkce. Funkcionální komponenty bývají jednodušší, jelikož nejsou schopny udržet informace o svém stavu a pracují pouze s daty, která jim byla předána jako argumenty volání funkce.

Komponenty definované třídou musí dědit ze třídy *React.Component*. Tyto komponenty pak implementují funkci *render*, která slouží k vykreslování uživatelského rozhraní. Data komponenty jsou pak uložena v objektu *state*. Pokud by došlo ke změně stavu, je nutno data přenastavit voláním funkce *this.setState*. Ta spočítá rozdíl předchozího stavu oproti novému a zavolá znovu *render*. Tím dojde k překreslení uživatelského rozhraní tak, aby odpovídalo novému stavu.

Mezi další používané funkce patří *componentWillMount*, která se volá tehdy, když je komponenta poprvé vytvořena. Jejím antagonistou je *componentWillUnmount*. Ta se spustí ve chvíli, kdy komponenta přestane být aktivní. (15)

4.6.2 Použité komponenty knihovny React-Native

Některé komponenty jsou implementovány již v modulu *React-Native*. Základní z nich se nazývá *View*. Jedná se o komponentu, která slouží k definování hierarchie potomků. V rámci atributu *style* můžeme definovat například její rozměry, pozici, barvu pozadí či vzhled okrajů. Podstatným parametrem je také *flex-direction*, který určuje, zda se budou potomci komponenty *View* řadit pod sebe nebo za sebe. Vnořováním několika těchto komponent pak lze vytvořit strukturu celé scény.

V případě komponenty *ScrollView* se jedná o rozšíření *View*, s tím rozdílem, že pokud by se její obsah neměl vejít na display zařízení, umožní uživateli obsah posouvat pomocí tažení prstem.

TouchableOppacity rozšiřuje *View* tím, že podporuje interakci s uživatelem. Do jejího atributu *onPress* lze uložit referenci na funkci, která se zavolá pokaždé, když uživatel stiskne prostor zabíraný touto komponentou.

Text je jednoduchá komponenta, která se nemůže větvit. Jediným jejím obsahem může být textový řetězec, který se zobrazí uživateli. Jeho vzhled je možné upravit pomocí

atributu *style*.

Jakožto indikátor načítání dat slouží komponenta *Spinner,* která má tvar točícího se kolečka.

4.6.3 Vlastní komponenty

Každá programátorem definovaná komponenta se vždy musí skládat z komponent nativních. Mezi použité vlastní komponenty patří *Button,* která má funkci tlačítka a je složena z *TouchableOppacity* s kulatými okraji, mající v sobě komponentu *Text.*

Card a *CardSection* rozšiřují *View* přidáním odsazení a elevace v rámci stylu, tím pádem okno vypadá jako by se vznášelo nad pozadím.

4.7 Implementace potřebné funckionality

Zde se budeme zabývat implementací funkcí nezbytných pro chod mobilní aplikace. Mezi ně patří navigace v rámci jednotlivých scén, zobrazování map, stahování a ukládání dat pro off-line použití, komunikace s uživatelem, zobrazovaní prezentací, geolokační služby a ukládání navštívených bodů zájmu.

4.7.1 Navigace mezi scénami

Základní *React-Native* neumožňuje navigaci mezi scénami, tím pádem bylo potřeba sáhnout po externí knihovně jménem *react-native-router-flux* (16). Její rozhraní nám dává k dispozici několik komponent. Kořenová komponenta se nazývá *Router* a obsahuje potomky typu *Scene*.

Každá scéna má následující atributy. *Key* obsahuje unikátní identifikátor scény, *component* obsahuje ukazatel na komponentu, která bude obsahem dané scény. Její titulek se pak nachází v atributu *title.* Úvodní scéna, která má být zobrazena při startu aplikace nese atribut *initial.*

Knihovna pro každou scénu vytvoří záhlaví, v jehož středu se nachází titulek a vlevo šipka pro návrat zpět. Spuštění scény definované parametrem *key* lze vyvolat zavoláním stejnojmenné metody objektu *Actions.*

4.7.2 Knihovna pro mapy

Jedním z problémů, které bylo třeba vyřešit, byla volba vhodné knihovny pro zobrazování map pomocí *React-Native*. Pro tyto účely se zdály být použitelné dva balíčky. Prvním z nich je projekt *React-Native Maps* (17) od firmy Airbnb, druhým pak *React-Native MapboxGL* (18). Z těchto dvou knihoven byla zvolena ta první, vzhledem k tomu, že je lépe zdokumentovaná, ale především z toho důvodu, že ji využívá pro svou implementaci map právě framework *Expo*.

Není potřeba nic instalovat, stačí v projektu importovat komponentu *MapView* z balíčku *Expo*. Pro zobrazování off-line obsahu lze použít *UrlTile*, která umožní načítat soubory z nějaké adresy. Při načítání map z úložiště zařízení je potřeba postupovat stejně jako při načítání z webu. Na místo adresy URL je nezbytné využít URI adresu souborů s předponou $file://$.

K zobrazování pozice bodů zájmu či polohy uživatele je použita komponenta *Marker,* která zobrazí ukazatel na souřadnicích definovaných v atributu *coordinate.*

4.7.3 Použití dlaždic pro zobrazovaní map

Důležitou součásti aplikace je stažení map pro off-line použití. Služba *OpenStreetMap* (19) naše požadavky splňovala. Mapy od tohoto poskytovatele jsou složeny z takzvaných dlaždic. Každá dlaždice je definovaná třemi čísly: X, Y a $zoom$. X určuje horizontální polohu dlaždice, Y polohu vertikální, $zoom$ je číslo z intervalu $[1; 19]$, stanovující hodnotu přiblížení. Tyto dlaždice je možno zdarma stahovat z několika dostupných serverů. URL adresa jednotlivých dlaždic mívá tvar $server/zoom/X/Y.png$ (20).

Nicméně vzhledem k tomu, že GPS používá pro určení lokace zeměpisné souřadnice, tedy zeměpisnou šířku (*latitude*) a zeměpisnou délku (*longitude*), je potřeba tyto souřadnice nejprve převést na hodnoty Xa Y. Pro tento převod nám mohou posloužit Rovnice 1 a Rovnice 2:

$$X = \left\lfloor \frac{lon + 180}{360} \times 2^{zoom} \right\rfloor$$

Rovnice 1 – Výpočet horizontální souřadnice

$$Y = \left\lfloor \left(1 - \frac{\ln\left(tan\left(lat \times \frac{\pi}{180} \right) + \frac{1}{cos\left(lat \times \frac{\pi}{180} \right)} \right)}{\pi} \right) \times 2^{zoom-1} \right\rfloor$$

Rovnice 2 – výpočet vertikální souřadnice

Pokud bychom chtěli získat souřadnice všech dlaždic regionu definovaného parametry lon_{min}, lon_{max}, lat_{min}, lat_{max}, lze postupovat následujícím způsobem: Nejprve tato čísla převedeme pomocí vzorců (1) a (2) na souřadnice X_{min} , X_{max}, Y_{min}, Y_{max}. Provedením kartézského součinu množin $\{X_{min} \dots X_{max}\}$ a $\{Y_{min} \dots Y_{max}\}$ získáme souřadnice (X, Y) všech dlaždic daného regionu. Opakováním této procedury pro různé hodnoty $zoom$ lze vygenerovat množinu URL adres všech souborů potřebných ke stažení, aby bylo možné tento region zobrazit na mapě.

4.7.4 Posílání *http* požadavků

Na posílání požadavků a komunikací s API slouží knihovna *axios* (21). Její výhodou oproti metodě *fetch* obsažené v *JavaScriptu* je například automatický převod odpovědi na objekt anebo dekompozice práce s odpovědí do dvou bloků.

Volání API vytváříme pomocí funkce *get,* práce s odpovědí v případě očekávaného průběhu volání je umístěna do bloku *then*. Pokud požadavek skončí s chybou, je na ni reagováno pomocí bloku *catch.*

4.7.5 Stahování off-line obsahu

Ve chvíli, kdy je vygenerován seznam URL adres, ze kterých je potřeba získat dlaždice, lze přejít k samotnému stahování obsahu. Vzhledem k tomu, že manipulace se souborovým systémem zařízení není součástí nativních komponent *React-Native,* bylo opět potřeba použít externí knihovnu. V tomto případě se jednalo o modul *frameworku Expo* s názvem *FileSystem* (22), který umožňuje práci se soubory a složkami v úložišti zařízení. Jednou z výhod tohoto modulu je, že veškeré operace jsou asynchronní a vždy vrací *Promise*. To nám umožnuje pokaždé vykonávat několik těchto operací najednou, což vede k výraznému zrychlení aplikace oproti případné sekvenční verzi.

Soubory související s danou stezkou jsou vždy ukládány do oddělené složky

s názvem *book#id,* kde *#id* je unikátní identifikátor dané stezky. Složku lze vytvořit pomocí funkce *makeDirectoryAsync,* mazání včetně obsahu umožňuje *deleteAsync.*

Knihovna také umožňuje stahování souborů přímo na úložiště zařízení. Zde jsme však narazili na zásadní chybu, znemožňující tuto funkci používat. Tato chyba spočívá v tom, že pokud je vytvořen požadavek (*request*) na stažení souboru, tak i po úspěšném stažení dat a jejich zapsání na úložiště, nedojde k uzavření daného požadavku. Zmíněný problém by nebyl tak závažný, pokud by docházelo pouze ke stahování malého množství souborů. Během stahování map se ale vytváří takovýchto požadavků desítky až stovky. Neuzavřené požadavky zůstávají v paměti zařízení a po určitém čase způsobí pád aplikace.

Z těchto důvodů není přímé stahování souborů použitelné pro naši aplikaci, dokud nedojde k opravení chyb. Naštěstí je možné tento problém obejít. Řešením je použít na tvorbu požadavku knihovnu *axios*. Pomocí ní získáme data z webového zdroje pomocí *http* metody *GET* a uložíme je do proměnné typu *string*.

Data v operační paměti zařízení zapíšeme na úložiště pomocí metody *writeAsStringAsync,* které předáme textový řetězec, jenž je třeba zapsat. Pro zápis obrázku je před zápisem ještě provedena konverze na formát *base64.* Navzdory těmto operacím se soubory, se zmíněné řešení jeví jako dostatečně rychlé.

Načítání obsahu souboru do paměti se provádí pomocí funkce *readAsStringAsync.*

4.7.6 Dekódování definice stezek

Pomocí *FileSystem* je možné načíst obsah XML souboru do paměti jako *string*. Následně, pokud je potřeba pracovat s obsaženými daty, je nutné ho převést na *JavaScriptový* objekt. Pro tento převod byla použita knihovna s názvem *React-Native-xml2js* (23). Tento parser vždy ze všech *tagů* stejného typu, které sdílí rodiče, vytvoří seřazený seznam objektů, na něž se jednotlivé *tagy* převedly.

Každý XML element se převádí na seznam z důvodu bezpečnosti, jelikož nelze jednoznačně určit, zda se u osamocených položek jedná o proměnnou nebo o jednoprvkový seznam.

4.7.7 Zprávy pro uživatele

Aplikace implementuje tři způsoby předávání zpráv uživateli. Prvním typem je *Toast*, což označuje bublinu obsahující krátkou zprávu pro uživatele, která sama po několika sekundách zmizí. Jelikož iOS standardně nepodporuje tuto funckionalitu, je použita externí knihovna *react-native-easy-toast* (24). Pomocí její komponenty *Toast* lze posílat zprávy podobné těm nativním z platformy Android.

Další typ zprávy je modální okno. Pro jeho implementaci je použita třída *Alert* (25) z modulu *React-Native*. Kromě textu zprávy lze ještě definovat tlačítka pro odpověď ze strany uživatele. Na každé tlačítko lze navěsit funkci, která je vyvolána poté, co jej uživatel stiskne.

Posledním typem jsou *Push* notifikace, k jejichž vytváření poskytuje rozhraní *Expo* (26). Jejich hlavní výhodou je nezávislost na jakékoli komponentě a možnost posílání, i když se aplikace nachází na pozadí.

4.7.8 Persistetní ukládání informací

Některé informace je potřeba zachovat i poté, co je aplikace vypnuta. To naštěstí umožňuje sám *React-Native* pomocí knihovny *AsyncStorage* (27). Jedná se o databázi typu „klíč-hodnota“, implementovanou pomocí asynchronních operací. Pár lze nastavit funkcí *setItem,* přijímající datový typ *string.* Pro uložení složitějších objektů je tedy vhodné provést serializaci, například do formátu *json.* Načítání z databáze zajišťuje metoda *getItem.*

4.7.9 Další vizuální komponenty

Některé komponenty uživatelského rozhraní nejsou součástí knihovny *React-Native,* a proto bylo potřeba použít externí moduly. *React-Native-Elements* (28) je první z nich. Komponenta *Overlay* má funkci modálního *View,* které lze dynamicky zobrazit a skrýt pomocí atributu *isVisible. CheckBox* rozšiřuje *TouchableOppacity* a má vzhled zaškrtávacího tlačítka s popiskem, jehož styl lze změnit na *Radio Button* pomocí atributu *checkedIcon. ListItem* je použit pro vykreslování některých seznamů, především díky možnosti doplnit nápis o ikony.

Mezi další knihovny patří *react-native-tab-view* (29), který umožňuje scénu dělit

na záložky a přepínat mezi nimi. *React-native-draggable-flatlist* (30) přináší list, jehož prvky může uživatel přesouvat tažením, *React-native-wheel-scroll-picker* (31) nám umožňuje používat číselník známý z iOS i na platformě Android.

4.7.10 Geolokační služby

Pro sledování uživatelovy GPS lokace se používá třída *Location* (32) z knihovny Expo. Metoda *watchPositionAsync* začne sledovat geografickou pozici a při její změně zavolá funkci zadanou jako argument, které novou pozici předá. Tuto metodu je možné zavolat z komponenty, jejíž stav lze poté měnit.

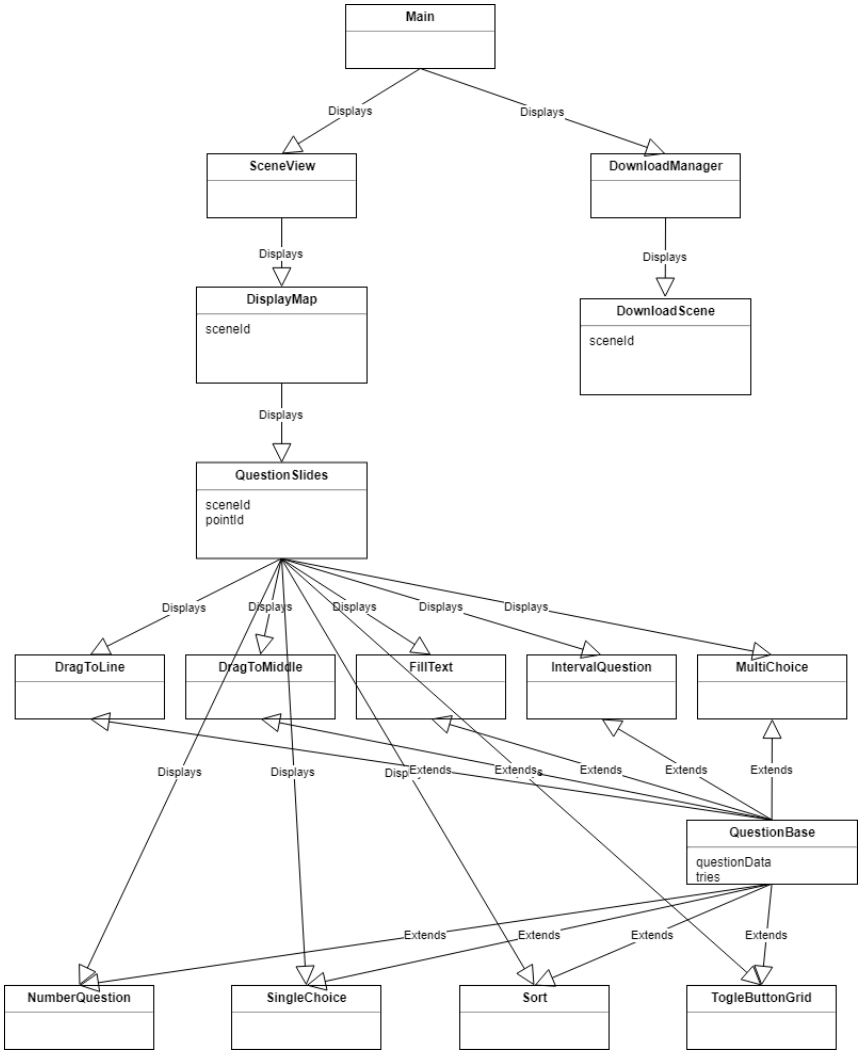
StartLocationUpdatesAsync slouží pro situace, kdy je aplikace na pozadí. Nesmí být vázaná na žádnou komponentu, a proto komunikace se zbytkem aplikace musí být zajištěna například pomocí *AsyncStorage*. Sledování lokace lze zastavit funkcí *stopLocationUpdatesAsync*.

4.8 Implementace scén

Posledním krokem se stala implementace jednotlivých scén, přičemž každá z nich je oddělená komponenta registrovaná pomocí třídy *Router*. Zde se budeme zabývat tvorbou grafické i funkční stránky jednotlivých scén. Rozdělení scén do tříd a jejich jednotlivé vztahy popisuje Obrázek 10.

Během implementace bylo dbáno na různé konvence pro design. Jednou z nich mohla být snaha o sdělení stavu aplikace pomocí barvy komponent. Tedy šedé tlačítko je neaktivní, modré je aktivní nebo určení správné a chybné odpovědi pomocí zelené a červené barvy. Dále pokud systém vykonává nějaký časově náročný úkon nebo dojde k neočekáváné chybě, je o tomto stavu uživatel vždy informován pomocí zpráv, modálních oken či jednoduchých popisků (33).

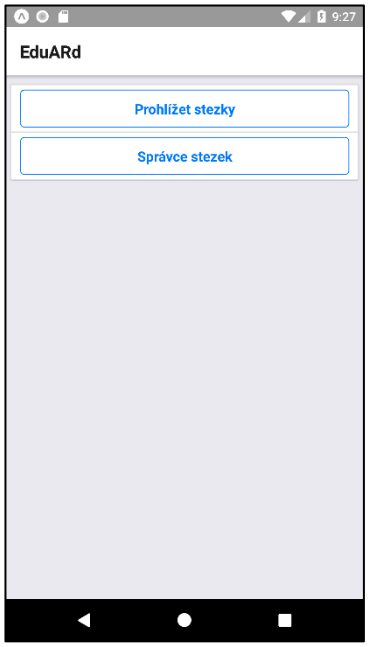
Některé scény zobrazují textová data získaná z *API*. Vzhledem k tomu, že v době tvorbě práce bylo *API* rovněž ve stavu vývoje, některá data obsahovala *HTML tagy* či různé nesmyslné textové řetězce. V produkční verzi se nicméně počítá s tím, že z *API* bude chodit pouze obyčejný text.



Obrázek 10 - Diagram tříd použitých pro grafické rozhraní

4.8.1 Hlavní scéna

Rozložení hlavní scény je jednoduché. Skládá se z jedné *Card* obsahující dvě *CardSection*. Uvnitř každé z nich se nachází jeden *Button (*Obrázek 11*)*, jehož *onPress* funkce vyvolá spuštění další scény pomocí třídy *Actions*.

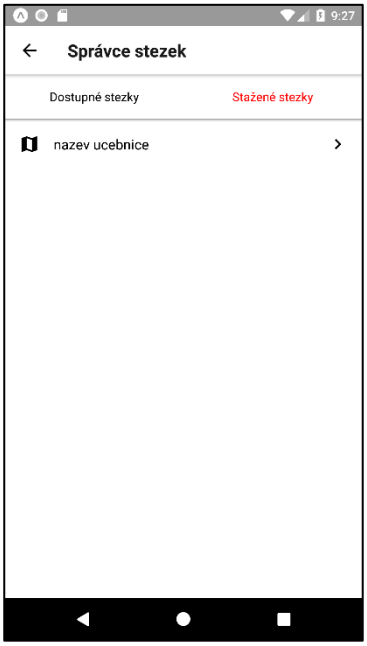


Obrázek 11 - Ukázka implementace hlavní scény

4.8.2 Správce stezek

Scéna je rozdělena pomocí *TabView* na dvě záložky (viz Obrázek 12). Po zavolání *componentWillMount* systém načte z *AsyncStorage* seznam obsahující stažené stezky a stáhne z informace o dostupných stezkách z API pomocí *axios*. Dostupné stezky se vykreslí jako seznam komponent *ListItem*. Ty, které nejsou stažené se zobrazí v první záložce, ostatní pak ve druhé. *OnPress* funkce každé položky zavolá *Actions* a spustí scénu *Správce konktrétní stezky*. Informace o stezce se této scéně předávají jako argument.

Pokud není požadavek úspěšný, je o tom uživatel informován modálním oknem *Alert*, po jehož zavření dojde k návratu na předchozí scénu zavoláním funkce *Actions.pop*.



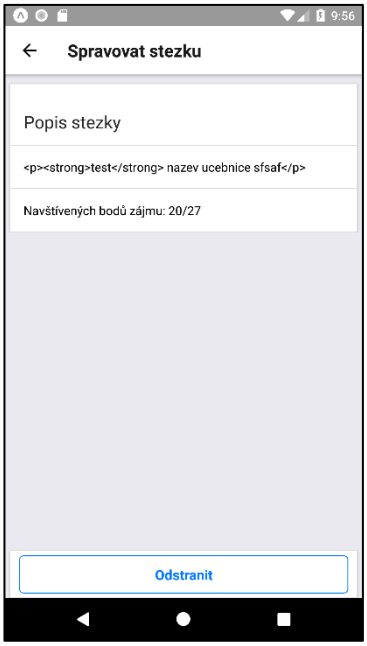
Obrázek 12 - Ukázka implementace správce stezek

4.8.3 Správce konkrétní stezky

Tělo scény tvoří *Card* s *CardSection,* obsahující info o stezce získané z API. V zápatí se nachází *Button. ComponentWillMount* přečte, zda byla stezka již stažena (viz Obrázek 13).

Pokud stažena nebyla, dojde ke stažení *XML* definice stezky pomocí *axios* a uložení přes *FileSystem*. Poté se *Button* aktivuje, a spuštěním *onPress* funkce dojde k zahájení stahování map. Uživateli je zobrazen *Overlay* s textovou informací o počtu stažených souborů. Systém mezitím dekóduje *XML,* získá z něj seznam všech GPS bodů a vytvoří *set URL* adres dlaždic, které je nutno stáhnout. Poté začne tvořit požadavky na stažení každé z nich, kterých může být najednou aktivních maximálně třicet. Po stažení všech nezbytných souborů dojde k načtení seznamu dostupných stezek z *AsyncStorage* a přidání dané stezky do tohoto seznamu.

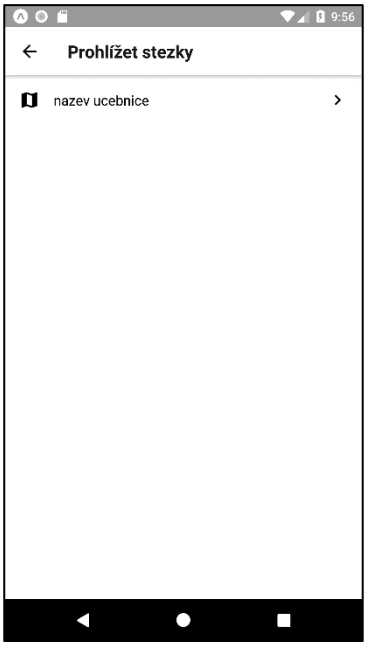
Pokud je stezka již stažena, je nastavena *onPress* funkce tlačítka na operaci mazání. Jejím spuštěním *FileSystem* odstraní složku dané stezky a vymaže ji ze seznamu těch stažených v *AsyncStorage*.



Obrázek 13 - Ukázka implementace správce stezky

4.8.4 Moje stezky

V *componentWillMount* sytém získá seznam stažených stezek z *AsyncStorage*, které následně zobrazí jako seznam položek *ListItem* (viz Obrázek 14). V *onPress* metodě každého z nich dojde k načtení a dekódovaní *XML* definice a vytvoření nové scény. Pokud má první bod zájmu atribut *intro*, je spuštěna scéna prezentace, které je předán daný bod a příznak *intro*. V opačném případě je spuštěna scéna mapy a argumentem je celé info o stezce.



Obrázek 14 - Ukázka implementace scény "Moje Stezky"

4.8.5 Mapa stezky

Kořenovou komponentou této scény je *MapView* s atributem *mapType* rovným *none*, což zamezí on-line stahování map. *MapView* se dále větví na *UrlTile* s cestou k souborům stezky, která byla předána jako argument scény.

ComponentWillMount načte *XML* definici stezky a převede na objekt. Následně z *AsyncStorage* přečte informace o dosažených bodech zájmu. Body zájmu zobrazí na mapě, jakožto list komponent *Marker* a u dosažených nastaví *pinColor* na zelenou (viz Obrázek 15). Všechny body zájmu se dočasně uloží na *AsyncStorage*, aby s nimi bylo možné pracovat, i když je aplikace na pozadí.

Dále dojde k zahájení dvou procesů sledování uživatelovy pozice. První pracuje na popředí a je svázán se scénou. Ve chvíli, kdy je načtena uživatelova pozice, je

zobrazena na mapě ve formě modré značky *Marker*. Pokaždé, když dojde ke změně GPS lokace, systém přepočítá vzdálenost od každého bodu zájmu, a těm, které nebyly navštíveny a jsou blíže než třicet metrů, nastaví barvu na žlutou. Pokud je barva bodu žlutá nebo zelená, *onPress* metoda spustí pomocí *Actions* scénu prezentace pro daný bod.

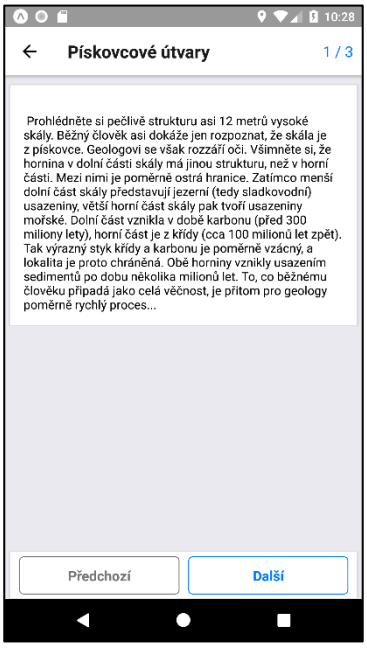
Druhý proces pracuje na pozadí a není svázaný s žádnou komponentou. Pracuje pouze s *AsyncStorage*. Při aktualizaci pozice rovněž spočítá vzdálenosti od bodů zájmu, a jména dostupných pošle ve formě *Push* notifikace.

V rámci *componentWillUnmount* dojde k deaktivaci obou procesů pro sledování GPS pozice.

Obrázek 15 - Ukázka implementace Mapy stezky

4.8.6 Prezentace

Tělo scény zabírá *Card* obsahující *Text*, případně *Image*, pokud prezentaci doprovázejí obrázky. V zápatí se vyskytují dvě tlačítka *Button* vedle sebe, která slouží k přepínání mezi snímky (Obrázek 16). Při spuštění scény dojde k filtraci snímků předaných scéně jako argument, a jsou zachovány jen ty, mající nějaký obsah. Scéna také uchovává index snímku, který zrovna zobrazuje. Při posouvání indexu vždy proběhne kontrola, zda neobsahuje nová obrazovka kvíz. Pokud ano, tak dojde k inicializaci objektu otázky a následnému volání jeho metody *render*. Pokud je otázka již navštívena, je načtena z *AsyncStorage*, v opačném případě se vytvoří zcela nový objekt.



Obrázek 16 - Ukázka implementace scény prezentace

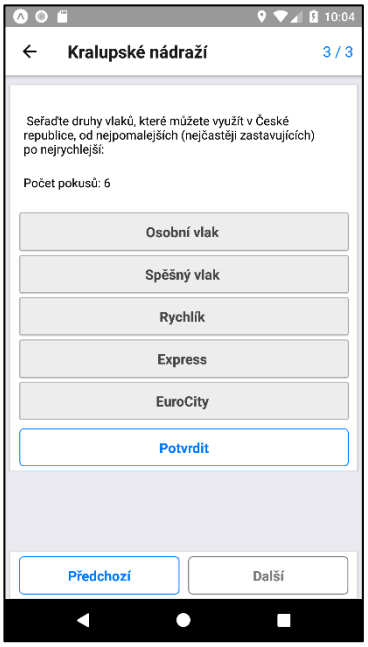
4.9 Implementace otázek

Každá otázka je samostatná třída dědící ze základní třídy *QuestionBase*, která obsahuje vlastnosti společné pro všechny otázky. Definuje funkci pro aktualizaci scény obsahující otázku nebo počítadlo pokusů. Dále poslední komponentou každé otázky je tlačítko sloužící k validaci. Ve funkci *onPress* je zjištěna správnost odpovědi. Pokud je odpověď chybná, dojde k snížení počtu pokusů. Pokud je odpověď správně nebo není dostatek pokusů, kvíz bude ohodnocen a uzamkne se pomocí příznaku *validated*. Správnost odpovědí je vždy určena hodnotou atributu *backgroundColor* na buďto zelenou, nebo

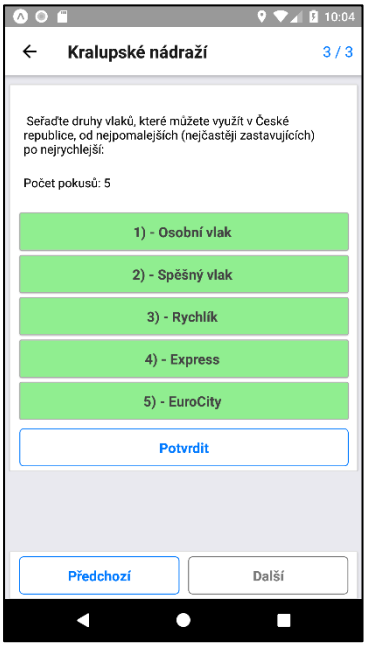
červenou. Každá změna stavu otázky se také zapisuje do *AsyncStorage*, aby se informace o vyplnění ukládaly i po zavření scény.

4.9.1 Kvíz sort

Sort je implementována pomocí *DragableFlatList*, který obsahuje jednotlivé možnosti ve formě *TouchalbeOppacity* (Obrázek 17). Po validaci se promění na obyčejný neměnný seznam položek, u kterých je nastaven *backgroundColor*, v závislosti na tom, jestli je prvek na správném indexu (Obrázek 18).



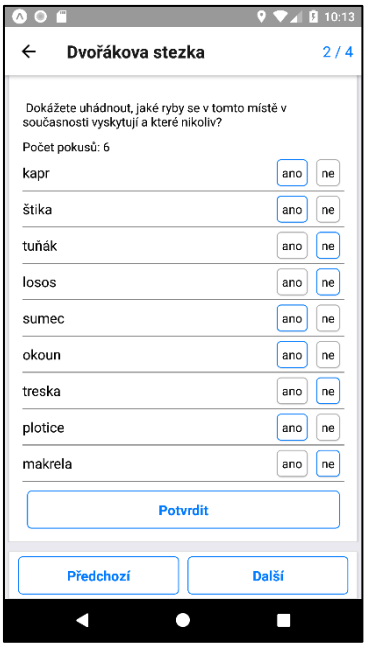
Obrázek 17 - Ukázka kvízu sort



Obrázek 18 - Ukázka vyhodnocení kvízu sort

4.9.2 Kvíz togglebuttongrid

Tělo otázky *togglebuttongrid* tvoří seznam horizontálně orientovaných *View*. Jeho prvním elementem je *Text*, za kterým následuje seznam *TouchableOppacity* pro každou možnou odpověď (Obrázek 19). Třída si pamatuje indexy zvolených položek, kterým nastaví pozadí na modro. Po validaci je jednotlivým položkám nastavena barva pozadí v závislosti na správnosti odpovědi (Obrázek 20).

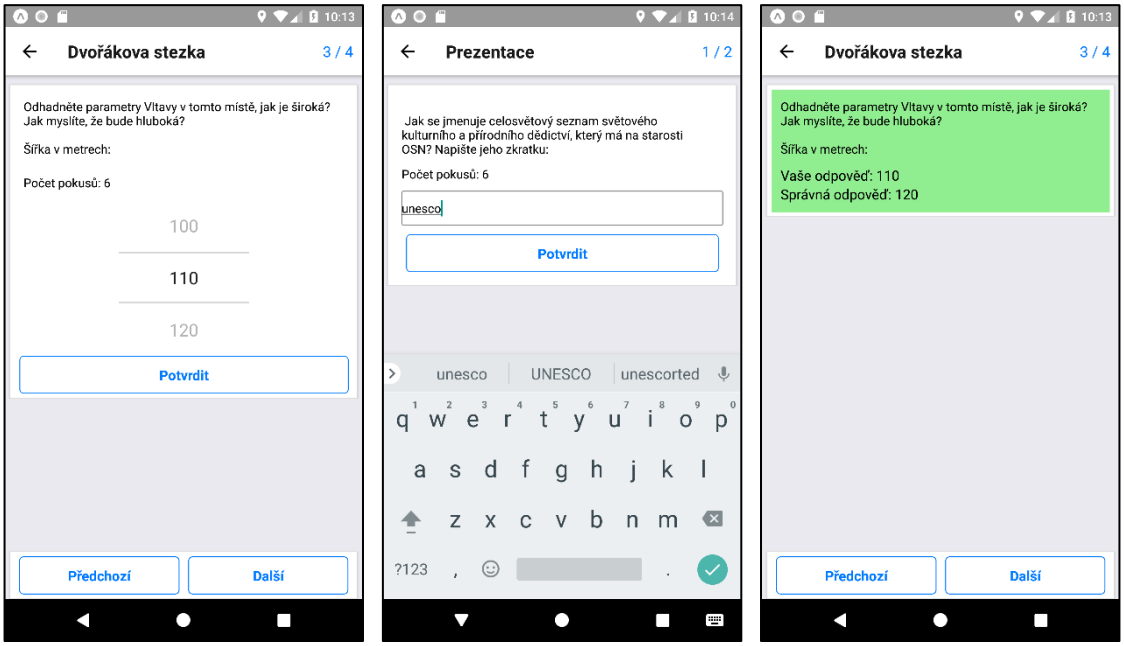


Obrázek 19 - Ukázka kvízu togglebuttongrid

Obrázek 20 - Ukázka vyhodnocení kvízu togglebuttongrid

4.9.3 Kvízy se zadáním odpovědi

Intervalquestion je tvořen pouze číselníkem *ScrollPicker*, jehož rozpětí je nastaveno na interval definovaný v *XML* předpisu. *Numberquestion* a *filltext* tvoří obyčejný *TextInput*. Po validaci je nahrazen *View*, obsahujícím pole *Text* s informací o zvolené odpovědi a správné odpovědi. Pozadí tohoto *View* je určeno tím, zda je uživatelova odpověď v intervalu tolerance (Obrázek 21).

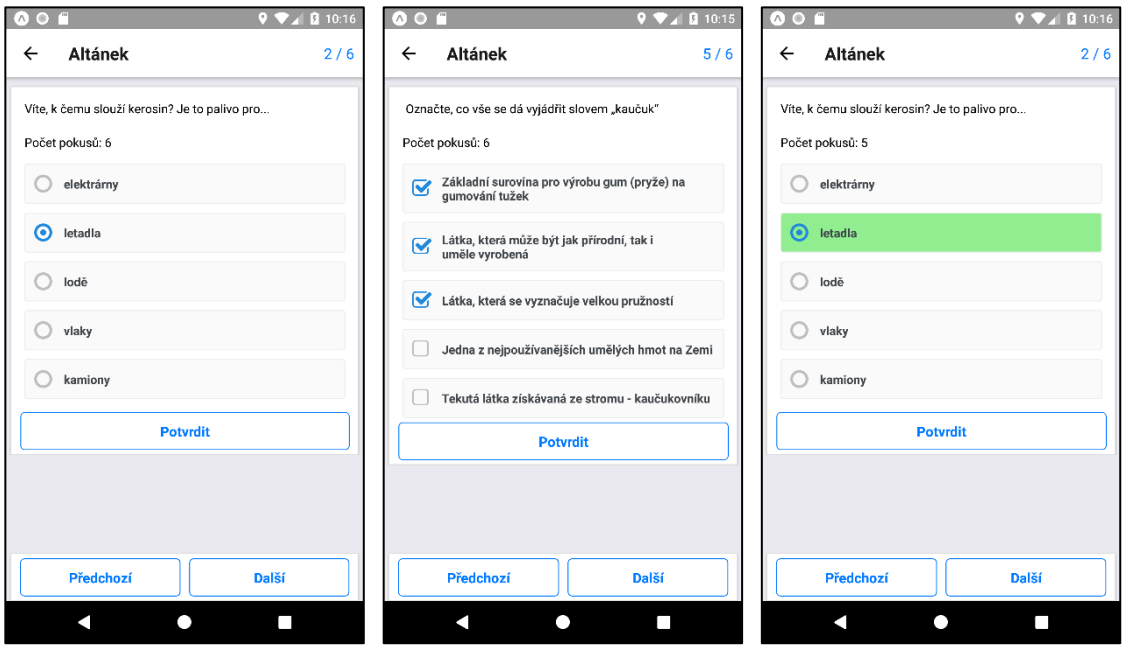


Obrázek 21 - Ukázka kvízů se zadáním hodnoty. Zleva ukázka číselníku, formulář pro zadání odpovědi a obrazovka vyhodnocení

4.9.4 Kvízy s výběrem odpovědi

Singlechoice i *multichoice* jsou implementovány jako list komponent *CheckBox*. V případě typu *singlechoice* je vzhled změněn tak, aby připomínaly *radio button*. Třída nese informaci o tom, zda jsou jednotlivé položky zvoleny a nastavuje podle toho jejich atribut *checked*.

Po validaci je u jednotlivých položek nastavena barva pozadí v závislosti na tom, jestli byly vybrány a jestli se jedná o správnou odpověď či nikoli (Obrázek 22).



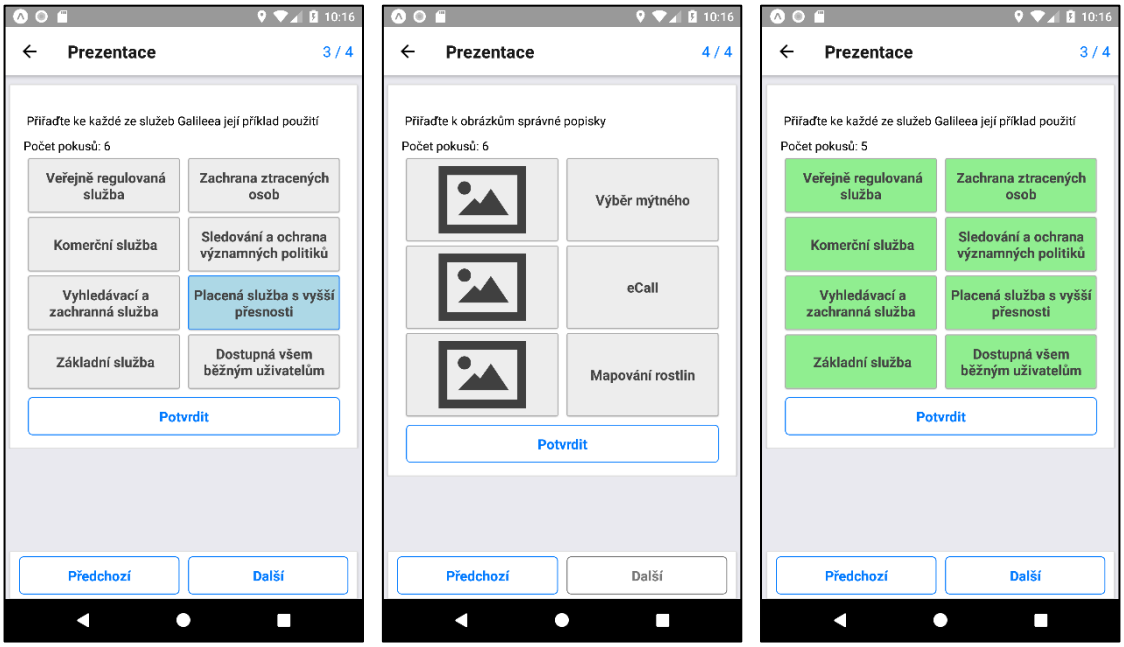
Obrázek 22 - Ukázka výběru z odpovědi. Zleva singlechoice, multichoice a evaluace

4.9.5 Přirazovací kvízy

Mezi tuto kategorii se řadí kvízy *dragtoline* a *dragtomiddle*. Jejich struktura je velmi podobná. Vždy je hlavní komponentou seznam horizontálních komponent *View*, které se dělí na dva prvky typu *TouchableOppacity*. Obsahem toho prvního je v případě *dragtoline Text*, v opačném případě *Image*. Druhý prvek je pak vždy tvořen blokem textu. Místo skutečných fotografií je zatím používán zástupce, jelikož v době tvorby práce *API* ještě nebylo schopno poskytovat obrázky ke stažení.

Stisknutím pravého tlačítka dojde k zapamatování jeho indexu a nastavení jeho barvy na modrou. Stisknutím levého tlačítka se pak prohodí pravý prvek na indexu tohoto tlačítka s pravým prvkem na zvoleném indexu. Tato změna je animovaná pomocí třídy *LayoutAnimation* z knihovny *React-Native*.

Po validaci se vždy obarví celý řádek v závislosti na tom, zda k sobě levá a pravá položka patří, tedy zda atribut *option* pravé složky se rovná atributu *id* složky levé (Obrázek 23).



Obrázek 23 - Ukázka přiřazovacích kvízů. Zleva dragtoline, dragtomiddle a příklad evaluace

5 Testování

Aplikace byla testována na třech zařízeních. Prvním byl emulovaný telefon Nexus 6, další byla fyzická zařízení Archos Sense S5 a Apple iPhone 7. Pro testovací účely je možné nastavit konstantu *debug* v souboru *constants.js* na *true*, čímž dojde k odemčení prezentací, které je pak možné prohlížet, i když se uživatel nenachází dostatečně blízko.

5.1 Vykreslování map přes sebe

Na zařízeních iOS docházelo k chybě ve scéně s mapou. Při zobrazování dlaždic totiž docházelo k jejich načítání z úložiště zařízení, ale pod nimi se ještě vykreslovaly nativní mapy iOS, které byly stahovány z webu. Po nastavení atributu *provider* komponenty *MapView* na hodnotu *google*, tento jev ustal.

5.2 Posouvání komponenty *FlatList*

Při řazení seznamu v rámci kvízu *Sort* na telefonu iPhone docházelo k posouvání nejen jednotlivých položek, ale i celého seznamu, což velmi ztěžovalo práci s kvízem. Řešením se stalo nastavení atributu *bounces* komponenty *DraggableFlatList* na *false*.

Dalším nedostatkem byla téměř sekundová prodleva mezi stisknutím položky seznamu a zahájení posouvání. Zmíněnou prodlevu bylo nutno snížit nastavením hodnoty atributu *delayLongPress* u položek seznamu na 50 milisekund.

6 Závěr

Cílem práce bylo implementovat funkční prototyp multiplatformní geolokační aplikace pro virtuální naučné stezky s možností stahování a prohlížení map a zobrazováním prezentací a kvízů. Na začátku bylo potřeba seznámit se s celkovým stavem projektu EduARd. Byl analyzován současný formát *XML* definic virtuálních naučných stezek a proběhl rozbor požadavků na mobilní aplikaci.

Dalším krokem byl návrh grafické podoby aplikace a ilustrace jednotlivých případů užití. Byly navrženy jednotlivé scény a jejich interakce s uživatelem. Také proběhl prvotní návrh vzhledu a způsobu vyplňování jednotlivých kvízových otázek.

V rámci implementace byly vyhledány jednotlivé knihovny, které by zajistily požadované funkcionality definované v zadání práce. Poté došlo k výběru potřebných komponent, pomocí nichž byly implementovány všechny scény.

Posledním část práce se věnovala testování aplikace, během kterého bylo ověřeno splnění všech zadaných funkčních požadavků, případně napraveny nalezené nedostatky. Jednalo se o implementaci prototypu aplikace, který bude v budoucnu dále rozpracován.

7 Reference

1. ***Geofun - výletní hry s mobilem.*** **[Online] https://www.geofun.cz/.**

2. První virtuální stezka v Českém Švýcarsku. ***NP České Švýcarsko.*** **[Online] http://www.npcs.cz/prvni-virtualni-stezka-v-ceskem-svycarsku.**

3. ***React-Native.*** **[Online] 2019. https://facebook.github.io/react-native/.**

4. ***Node.js.*** **[Online] https://nodejs.org/en/.**

5. Kadishay, Yotam. JavaScript V8 Engine Explained. ***Hacker Noon.*** **[Online] https://hackernoon.com/javascript-v8-engine-explained-3f940148d4ef.**

6. ***npm.*** **[Online] https://www.npmjs.com/.**

7. Eisenman, Bonnie. ***Learning React Native.*** **Sebastopol : O'Reilly Media, 2016.**

8. JavaScript Environment. ***React Native.*** **[Online] https://facebook.github.io/react-native/docs/javascript-environment.**

9. ***React Native Made Easy.*** **[Online] https://www.reactnative.guide/3-react-native-internals/3.1-react-native-internals.html.**

10. Linking. ***React Native.*** **[Online] https://facebook.github.io/react-native/docs/linking.**

11. Running on Device. ***React Native.*** **[Online] https://facebook.github.io/react-native/docs/running-on-device.**

12. ***Expo.*** **[Online] 2019. https://docs.expo.io/versions/latest/.**

13. How Expo Works. ***Expo.*** **[Online] https://docs.expo.io/versions/latest/workflow/how-expo-works/.**

14. Learn the Basics. ***React Native.*** **[Online] https://facebook.github.io/react-native/docs/tutorial.**

15. React.Component. ***React.*** **[Online] https://reactjs.org/docs/react-component.html.**

16. React-native-router-flux. ***github.*** **[Online] https://github.com/aksonov/react-native-router-flux.**

17. React-Native-Maps. ***github.*** **[Online] https://github.com/react-native-community/react-native-maps.**

18. React-Native-Mapbox-GL. ***github.*** **[Online] https://github.com/mapbox/react-**

native-mapbox-gl.

19. *OpenStreetMap.* [Online] 2019. https://www.openstreetmap.org/.

20. *OSM Tiles.* [Online] https://c.tile.openstreetmap.org/7/63/42.png.

21. Axios. *github.* [Online] https://github.com/qiangmao/axios#readme.

22. FileSytem. *Expo Documentation.* [Online] https://docs.expo.io/versions/v32.0.0/sdk/filesystem/.

23. react-native-xml2js. *npmjs.* [Online] https://www.npmjs.com/package/react-native-xml2js.

24. react-native-easy-toast. *Github.* [Online] https://github.com/crazycodeboy/react-native-easy-toast.

25. Alert. *Expo Documentation.* [Online] https://docs.expo.io/versions/latest/react-native/alert/.

26. Notifications. *Expo Documentation.* [Online] https://docs.expo.io/versions/v32.0.0/sdk/notifications/.

27. AsyncStorage. *Expo Documentation.* [Online] https://docs.expo.io/versions/latest/react-native/asyncstorage/.

28. React Native Elements. *React Native Training.* [Online] https://react-native-training.github.io/react-native-elements/docs/overview.html.

29. React Native Tab View. *npmjs.* [Online] https://react-native-training.github.io/react-native-elements/docs/overview.html.

30. React Native Draggable FlatList. *npmjs.* [Online] https://www.npmjs.com/package/react-native-draggable-flatlist.

31. react-native-wheel-scroll-picker. *npmjs.* [Online] https://www.npmjs.com/package/react-native-wheel-scroll-picker#react-native-wheel-scroll-picker.

32. Location. *Expo Documentation.* [Online] https://docs.expo.io/versions/v32.0.0/sdk/location/.

33. Lowdermilk, Travis. *User Centered Design.* Sebastopol : O'Reilly Media, 2013.

8 Seznam obrázků a rovnic

9 Přílohy

K této práci je rovněž přiloženo DVD obsahující následující položky:

1. Složka *source* obsahující zdrojové kódy aplikace
2. Soubor *build.apk*, tedy zkompilovanou aplikaci pro Android
3. Soubor *README.md*, který obsahuje návod, jak aplikaci spustit