Pedagogická fakulta
Faculty of Education

Jihočeská univerzita v Českých Budějovicích
University of South Bohemia in České Budějovice

JIHOČESKÁ UNIVERZITA v ČESKÝCH BUDĚJOVICÍCH

Pedagogická fakulta

Katedra informatiky

Vývoj multiplatformních aplikací v React Native s využitím analýzy obrazu

Cross-platform development in React Native with image analysis

Bakalářská práce

Vypracoval: Martin Pongrácz

Vedoucí práce: Ing. Jan Jára, Ph.D.

České Budějovice 2018

JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH
Fakulta pedagogická
Akademický rok: **2016/2017**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Martin PONGRÁCZ**
Osobní číslo:	**P15673**
Studijní program:	**B7507 Specializace v pedagogice**
Studijní obor:	**Informační technologie a e-learning**
Název tématu:	**Vývoj multiplatformních aplikací v React Native s využitím analýzy obrazu.**
Zadávající katedra:	**Katedra informatiky**

Zásady pro vypracování:

Bakalářská práce se bude zabývat problematikou vývoje multi-platformních aplikací za pomocí frameworku React Native. Vysvětlí základy používání tohoto frameworku, porovná ho s jinými technologiemi a osvětlí, v čem spočívají jeho výhody. Dále se bude práce zabývat analýzou obrazu a portování hotové aplikace na operační systémy iOS a Windows 10. Studentům může sloužit jako studijní materiál pro práci s frameworkem React Native a postupem řešení problematiky analýzy obrazu.

V praktické části bakalant vytvoří jednoduchou aplikaci demonstrující základní funkce frameworku; příručku popisující, jak vytvořit jednoduchou aplikaci a rozsáhlejší aplikaci pro využití ve včelařství, která obsahuje funkci na rozpoznání včelí matky na plástvi.

Cíle:

1. Vyzkoumat možnosti řešení analýzy obrazu - hledání včelí matky.
2. Vytvořit aplikaci na použití ve včelařství - NFC a QR identifikace úlu; editace, správa a analýza získaných údajů
3. Vytvořit příručku pro zájemce o studium ve frameworku React Native

Rozsah grafických prací:	**CD ROM**
Rozsah pracovní zprávy:	**40**
Forma zpracování bakalářské práce:	**tištěná**

Seznam odborné literatury:

1. **React Native for Android: How we built the first cross-platform React Native app. [online]. Dostupný z WWW: https://code.facebook.com/posts/1189117404435352/react-native-for-android-how-we-built-the-first-cross-platform-react-native-app/**
2. **React Native, A framework for building native apps using React [online]. Dostupný z WWW: https://facebook.github.io/react-native/**
3. **React Native, Getting started [online]. Dostupný z WWW: http://facebook.github.io/react-native/docs/getting-started.html**

Vedoucí bakalářské práce:	**Ing. Jan Jára, Ph.D.** Katedra informatiky
Datum zadání bakalářské práce:	**24. dubna 2017**
Termín odevzdání bakalářské práce:	**30. dubna 2018**

Mgr. Michal Vančura, Ph.D.
děkan

L.S.

doc. PaedDr. Jiří Vaníček, Ph.D.
vedoucí katedry

V Českých Budějovicích dne 24. dubna 2017

Prohlášení

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích dne 7. 7. 2018

Podpis studenta

Abstrakt

Práce pojednává o tvorbě mobilních aplikací ve frameworku React Native. Popisuje základní funkce frameworku a výhody proti klasickému nativnímu a hybridnímu vývoji. V teoretické části se práce věnuje problematice nativního vývoje, porovnáním současných technologií a postupů v této oblasti. Dále se v teoretické části zabývá představením frameworku React Native včetně jeho instalace, teoretického fungování a správnými postupy pro dosažení optimálního běhu vytvořené aplikace. Dalším tématem teoretické části je analýza obrazu, kde je vysvětleno jak probíhá proces rozpoznávání objektů a učení algoritmu na vyhledávání požadovaného objektu. Praktická část obsahuje tvorbu aplikace na operační systém Android ve frameworku React Native, která bude obsahovat identifikaci úlu pomocí NFC nebo QR kódu, a rozpoznávání včelí matky na plástvi pomocí knihoven OpenCV spolu s Tracking.js, která bude následně přeportována na jiné operační systémy.

Klíčová slova

Mobilní aplikace, React Native, OpenCV, analýza obrazu

Abstract

This bachelor thesis is about mobile application development in the React Native framework. It describes basic functions of the framework and its pros compared to classic-native and hybrid development. The theoretical part of the thesis describes the process and theory of native development and comparation of current technologies and practices in this field. Another topic of the theoretical part is the introduction of the React Native framework including its installation process, theoretical functioning and correct programming procedures which help achieve the creation of optimally running applications. The next topic describes image processing, object detection and the process of training algorithms to find specific objects. The practical part includes the creation of an application using the React Native framework for Android OS, which can identify a bee hive using NFC or QR codes, and detection of a bee queen on a bee hive using the OpenCV and Tracking.js libraries, which will be subsequently ported to other operating systems.

Keywords

Mobile applications, React Native, OpenCV, image analysis

Poděkování

Touto cestou bych rád poděkoval panu Ing. Janu Járovi, Ph.D. za jeho odborné vedení při zpracování mé bakalářské práce. Dále bych rád poděkoval své rodině, která mi umožnila studium na vysoké škole.

Obsah

1 Úvod ... 11

1.1 Vlastnosti frameworku ... 11

2 Porovnání s hybridním a nativním vývojem ... 13

2.1 Platformy a jazyky ... 15

2.2 Debugování ... 15

2.3 Pluginy a komunita ... 17

2.4 Rychlost ... 17

3 Instalace frameworku ... 18

3.1 Prerekvizity ... 18

3.2 Emulátor ... 21

3.3 Fyzické zařízení namísto emulátoru ... 23

3.4 Instalace React Native a první aplikace ... 23

4 Vývoj aplikací ve frameworku ... 25

4.1 Syntaxe ... 25

4.2 Arrow functions ... 25

4.3 Proměnné ... 26

4.4 Moduly ... 26

4.5 JSX ... 27

4.6 Komponenty ... 28

4.6.1 Props ... 28

4.6.2 Vlastní komponenty ... 29

4.6.3 Styly ... 31

4.6.4 Dynamické styly ... 32

4.6.5 States ... 33

4.6.6 Text input ... 35

4.6.7 Seznamy ... 36

4.7 Layout ... 37

4.7.1 Flex ... 37

4.8 Struktura aplikace ... 41

4.9 Navigation ... 41
4.9.1 Stack navigator ... 42
4.9.2 Tab navigator ... 45
4.9.3 Drawer navigator ... 45
4.9.4 Skládání navigátorů ... 45
4.9.5 Posílání parametrů ... 46
4.10 Storage ... 47
4.11 Adresářová struktura ... 49
5 Debugování aplikace ... 52
5.1 Chybová a varovná okna ... 52
5.1.1 Yellow Box ... 52
5.1.2 Red Box ... 52
5.2 Konzole ... 52
5.3 Inspector ... 53
5.4 Chrome ... 53
5.5 Reload ... 53
5.6 Reactotron ... 54
6 Portování ... 55
7 OpenCV ... 56
7.1 Kaskádový klasifikátor ... 56
8 Praktická část ... 58
8.1 Skenování QR kódů ... 58
8.2 Čtení NFC štítků ... 60
8.3 Databáze ... 60
8.4 Detekce objektů ... 62
8.4.1 Node.js ... 62
8.4.2 Tracking.js ... 62
8.5 Trénink klasifikátoru ... 64
8.5.1 Příprava ... 65
8.5.2 Trénink ... 67

9 Závěr 69
Seznam použité literatury a zdrojů 70
Seznam obrázků 72
Seznam ukázek kódu 73

1 Úvod

React native je JavaScriptový framework[1] pro vývoj mobilních aplikací. Dal by se označit jako kombinace hybridního a nativního vývoje. Kombinuje totiž kvality obou přístupů. Vývoj je rapidní, protože aplikace je skládána pomocí předpřipravených komponentů, které jsou také optimalizovány jak na Android OS, tak na iOS. Tím je zaručena rychlost a spolehlivost spojována s nativním vývojem. Tomuto přístupu by se dalo vytknout, že neumožňuje práci s tolika komponenty jako klasický nativní vývoj. S uplynutým časem a novými verzemi takovýchto základních komponentů přibývá a o většinu chybějících komponentů se postará komunita.

Z hybridního vývoje je zachována také jednoduchá portabilita. Za jeho vývojem stojí společnost Facebook, která ho představila v roce 2015 na F8 – Facebook Developer Conference. Pomocí tohoto frameworku již byla vytvořena spousta významných aplikací, jako třeba mobilní aplikace společností Facebook, Instagram nebo Skype. Na letošní F8 tento projekt podpořil také Microsoft a Samsung s cílem rozšířit podporu na Windows a Tizen OS. Dá se tedy očekávat, že se React Native rozšíří na všechny platformy včetně chytrých televizí a palubních počítačů v automobilech.

1.1 Vlastnosti frameworku

„React Native je JavaScriptový framework, který využívá syntax JavaScriptu, běží na JavaScript VM[2] jádru mobilních zařízení a využívá React Platform k vytvoření mostu mezi JavaScriptovou syntaxí a hostitelskou platformou.“[1] *„Kód psaný v JavaScriptu je následně přeložen do nativních programovacích jazyků, tedy Java pro Android a Objective-C pro iOS. Další důležitou vlastností, kterou React Native přebral z klasického frameworku React, je koncept manipulace s DOM.“*[1] *„DOM neboli Document Object Model je platforma a jazykově neutrální rozhraní, které umožňuje programům a scriptům dynamicky přistoupit a upravit obsah, strukturu a styl dokumentu.“*[2]

Jelikož je ale úprava DOM náročná na čas a prostředky, React Native pracuje s konceptem virtual DOM. *„Pro každý DOM object existuje odpovídající „virtual DOM object“. Virtual DOM object je reprezentace DOM objektu, něco jako lehká kopie. Virtual DOM object má stejné vlastnosti jako reálný DOM object. Chybí mu ale schopnosti toho reálného, zejména*

[1] Framework – podpůrný prvek pro vývoj aplikací
[2] JavaScript VM – program nebo překladač vykonávající JavaScriptový kód

přímo měnit to, co se děje na obrazovce. Manipulace DOM objektů je ale pomalá. Manipulace virtual DOM je mnohem rychlejší, protože se na obrazovce nic nevykresluje. O úpravě *virtual DOM lze uvažovat jako o úpravě nákresu místo pohybu místnostmi v opravdovém domě.“*[3] *„React Native usnadňuje věci tím, že místo renderování celé stránky eviduje změny udělané na DOMu stránky, která je uložená v paměti a potom využívá minimum výkonu na to, aby vykreslila jenom tyto malé změny.“*[1]

Změny v DOM by měly být principiálně vázány na změnu dat. React Native toto pravidlo nastavuje životním cyklem komponentu, který je pevně definovaný. Hlavní charakteristikou tohoto životního cyklu je nové vykreslení komponentu na základě změny dat neboli stavů, které ho definují. Existuje-li tedy komponent jehož součástí je stav obsahující text, a tento text je změněn, tak je obnoven pouze tento konkrétní komponent.

2 Porovnání s hybridním a nativním vývojem

React Native je podle jeho charakteristik možné zařadit mezi další frameworky určené pro vývoj hybridních aplikací, což jsou *„webové stránky vložené do webview a vytvořené pomocí HTML5, CSS a Javascriptu. Vykonávají stejný kód nezávisle na platformě na které běží. Pomocí nástrojů jako PhoneGap[3] a Cordova[4] mohou využívat nativní funkce jako je například GPS nebo kamera.“*[4] Hlavním rozdílem oproti klasickým hybridním aplikacím je, že uživatelské rozhraní v React Native je při kompilaci přeloženo do nativních komponentů pro danou platformu. Javascriptový kód je skutečně vykonáván v jeho původní podobě a komunikuje s přeloženými komponenty pomocí takzvaného mostu neboli bridge.

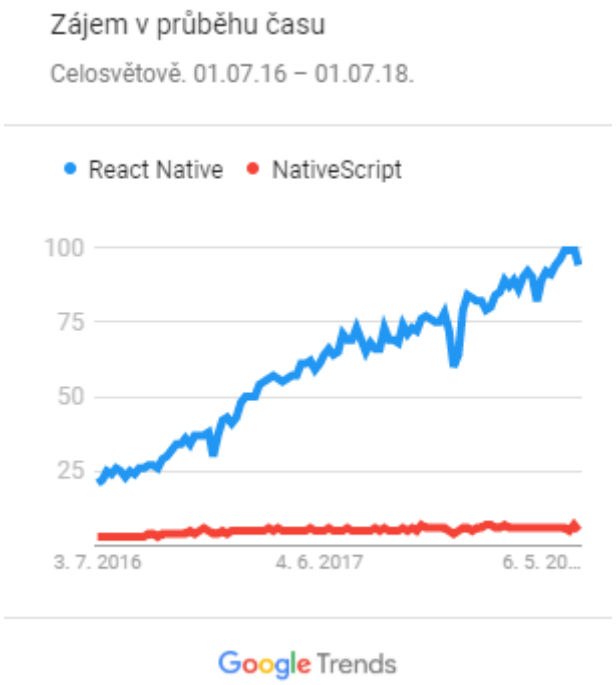
Iluze nativnosti React Native pochází z jeho interakce s hardware zařízení. Jelikož jsou komponenty překládány do jejich nativních protějšků, tak je dosaženo podobných rychlostí v interakci s uživatelským rozhraním. Tyto komponenty jsou tvořeny tak, aby stejnou implementací byly zasaženy všechny platformy, které React Native podporuje. Stejným přístupem je dosažena i možnost využití většiny periferií zařízení. Ve skutečnosti tedy uživatel ovládá nativní funkce operačního systému na kterém je aplikace spuštěna. Dalo by se tedy říci, že vývoj v React Native je hybridní, ale výsledná aplikace je spíše podobná těm nativním, což jsou aplikace *„vyvíjeny v jazyku potřebném na cílenou platformu, tedy Objective-C nebo Swift pro iOS a Java pro Android a ostatní. Napsaný kód nelze sdílet mezi platformami a jeho chování se liší. Má přímý přístup ke všem funkcím nabízením danou platformou bez jakýchkoliv omezení.“*[4]

React Native tedy nemusí, stejně jako při hybridním vývoji, rozlišovat mezi operačními systémy na které je cílen. Toto je znatelná výhoda, protože existují veliké rozdíly ve vývoji aplikací na různé platformy. Kromě rozdílů v programovacích jazycích, ve kterých se programují nativní aplikace na tyto platformy, existují i rozdíly v tom, jak se pracuje s API[5], jako například práce s GPS nebo tvorba animací. Díky tomu, že React Native tyto rozdíly řeší při definici komponentů, je v procesu nasazení aplikace šetřen čas i peníze. Neplatí to ale ve všech případech. Některé nativní vlastnosti nejsou v React Native zatím dostatečně podporované a je nutné využít nativního kódu, který lze pomocí React Native jednoduše implementovat. V tu chvíli je ale nutné mít dostatečné znalosti z nativního vývoje. Doporučil

[3] PhoneGap – framework pro tvorbu hybridních mobilních aplikací
[4] Cordova – framework pro tvorbu hybridních mobilních aplikací
[5] API – rozhraní pro programování aplikací

bych tedy využít tento framework na středně velké projekty, kde je žádoucí, aby byla aplikace rychle nasazena a fungovala svižně. Velké projekty využívající funkce specifické pro určitou platformu, jako například iMessage nebo složité animace, mohou být v React Native komplikovanější a časově více náročné než klasický nativní vývoj.

Za největšího protivníka frameworku React Native by se dal považovat NativeScript. Oba frameworky se snaží přistupovat k vývoji mobilních aplikací tak, že využívají zároveň výhody z hybridního i nativního vývoje. Rozdílné jsou však frameworky, na kterých jsou založeny. React Native využívá principů a jádra webového frameworku React, zatímco NativeScript je založen na frameworku Angular. Je tedy logické si vybrat framework, se kterým má uživatel předchozí zkušenosti.



Obrázek 2.1: Zájem v průběhu času mezi React Native a NativeScript

Výkonostně jsou na tom oba frameworky podobně, protože pracují s nativními funkcemi. React Native je zaměřen více na rychlost uživatelského rozhraní, dokáže využívat více jader procesoru a obecně mají aplikace menší velikost. Mezi výhody NativeScript patří tým výhradně profesionálních vývojářů a lepší výkon při práci s daty i nativními funkcemi.

2.1 Platformy a jazyky

Jelikož hlavní výhodou React Native je multiplatformnost, tak musí být aplikace na všechny operační systémy vyvíjeny jednotným způsobem. Konkrétně jsou použity dva typy syntaxe – JavaScript verze ES6 a JSX, což je krok přidávající XML syntaxi k JavaScriptu. Tato XML syntaxe je použita při definici obsahu komponentů a je podobná vývoji webových stránek v HTML. Princip stylování je také velmi podobný CSS z webových stránek. Syntaxe je uzpůsobena tak, aby připomínala CSS v rámci omezení způsobenými syntaxí JavaScriptu. Práce v React Native je tedy oproti hybridnímu vývoji o něco složitější, protože je nutná znalost JavaScriptu a základů OOP[6].

[6] OOP – objektově orientované programování

```
<div class="container">
      <p>Hello World!</p>
</div>
<div style="padding: 30px; background-color: white; text-align: center;
color: black; ">
      <p>Hello World!</p>
</div>
<style>
      .container {
          padding: 30px;
          background-color: white;
          text-align: center;
          color: black;
      }
</style>
```

Ukázky kódu 2.1: HTML – Porovnání s React Native

```
<Text>Hello World!</Text>
</View>
<View style={{padding: 30, backgroundColor: "white"; textAlign: "center",
color: "black"}}>
      <Text>Hello World!</Text>
</View>
var styles = StyleSheet.create({
      container: {
      padding: 30,
      backgroundColor: "white",
      textAlign: "center",
      color: "black";
}});
```

Ukázky kódu 2.2: React Native – Porovnání s HTML

2.2 Debugování

Testování v React Native je velmi svižné a jednoduché. Není nutné rekompilovat aplikaci po každé změně. Je také možné aktivovat funkci Live Reload, která načte aplikaci při každé změně kódu, nebo funkci Hot Reload, která při načtení zachová aktuální stav aplikace. Toto je značné zrychlení oproti nativnímu vývoji, kde je nutné zdrojové soubory aplikace vždy znovu

kompilovat. Pro potřebu debugování[7] je k dispozici kromě klasických varovných a chybných oken i debugger v prohlížeči Chrome. Jedná se o klasické vývojářské nástroje používané při vývoji webových stránek, které jdou jednoduše propojit s React Native. Toto umožňuje přístup ke konzoli a debugování JavaScriptových souborů. Pro debugování komponentů slouží Inspector, opět podobný tomu z webových vývojářských nástrojů. Umožňuje prohlížet umístění a velikosti elementů na obrazovce. Debugovací nástroje jsou velmi podobné těm z webu jak rychlostí, tak dostupnými nástroji.

2.3 Pluginy a komunita

React Native je veřejný projekt. Chyby ve frameworku jsou opravovány s pomocí komunity stejně jako přidávání nových funkcí. *„Přes 650 lidí předložilo změny kódu do kódové databáze React Native. Z těchto 5800 změn bylo 30 % z nich uděláno lidmi, kteří nepracují ve Facebooku. V únoru 2016 bylo poprvé více než 50 % změn od externích přispěvatelů … Mnoho z nich bylo vysoce kvalitních a zavádělo široce používané funkce.“*[5] Dá se tedy předpokládat, že rozšiřující balíčky umožňující přístup k různým funkcím, které nejsou v základním frameworku podporované, budou kvalitní. Tyto balíčky jsou dostupné z repozitáře NPM. Je možné také využít rozšíření, které umožňuje používat funkce z frameworku Cordova. React Native má tedy výhodu oproti hybridnímu vývoji v tom, že má k dispozici více funkcí a nástrojů.

2.4 Rychlost

React Native a nativní aplikace psaná ve Swift na iOS mají téměř identickou spotřebu CPU a GPU. Spotřeba paměti se v testech liší. Obecně by se dalo konstatovat, že nativní aplikace má výhody ve využití paměti při operacích, které spouštějí jiné aplikace nebo funkce operačního systému. Jelikož jsou prvky uživatelského rozhraní postaveny na těch nativních, tak existuje pouze malý rozdíl v rychlosti a ovladatelnosti uživatelského rozhraní. V každém případě jde o znatelný rozdíl oproti hybridním aplikacím.[6][7]

[7] Debugování – testování nebo ladění

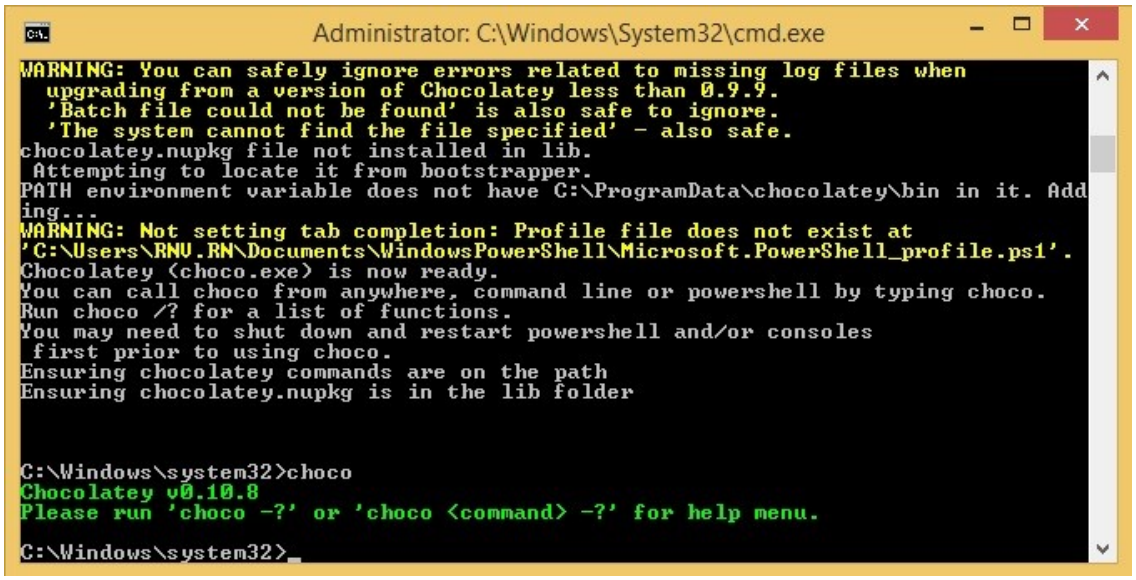
3 Instalace frameworku

V této kapitole bude popsán postup instalace frameworku React Native s cílem vytvořit aplikaci na operační systém Android –- na operačním systému Windows. Instalace je aktuální k datu 9.9.2017. Časem může dojít ke změnám v postupu instalace. Aktuální instalace by měla být vždy dostupná na adrese https://facebook.github.io/react-native/docs/getting-started.html.[8]

3.1 Prerekvizity

Prerekvizitami pro instalaci React Native je systém Node.js. Programové rozhraní Python verze 2 a vývojové rozhraní JDK verze 8. Všechny tyto balíky mohou být nainstalovány pomocí instalátoru Chocholatey. Chocolatey je možné nainstalovat pomocí následujícího příkazu zadaného do příkazového řádku spuštěného v administrátorském režimu.

```
@"%SystemRoot%\System32\WindowsPowerShell\v1.0\powershell.exe" -NoProfile -
InputFormat None -ExecutionPolicy Bypass -Command "iex ((New-Object
System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))
" && SET "PATH=%PATH%;%ALLUSERSPROFILE%\chocolatey\bin"
```

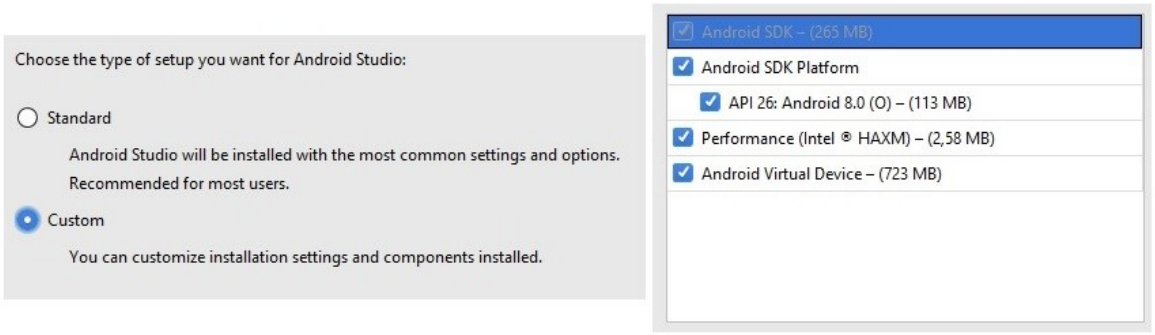


Obrázek 3.1: Úspěšná instalace programu Chocolatey

Zda byla instalace úspěšná lze ověřit zadáním příkazu `choco`. Poté lze nainstalovat Node.js, Python 2 a JDK 8 pomocí následujícího příkazu.

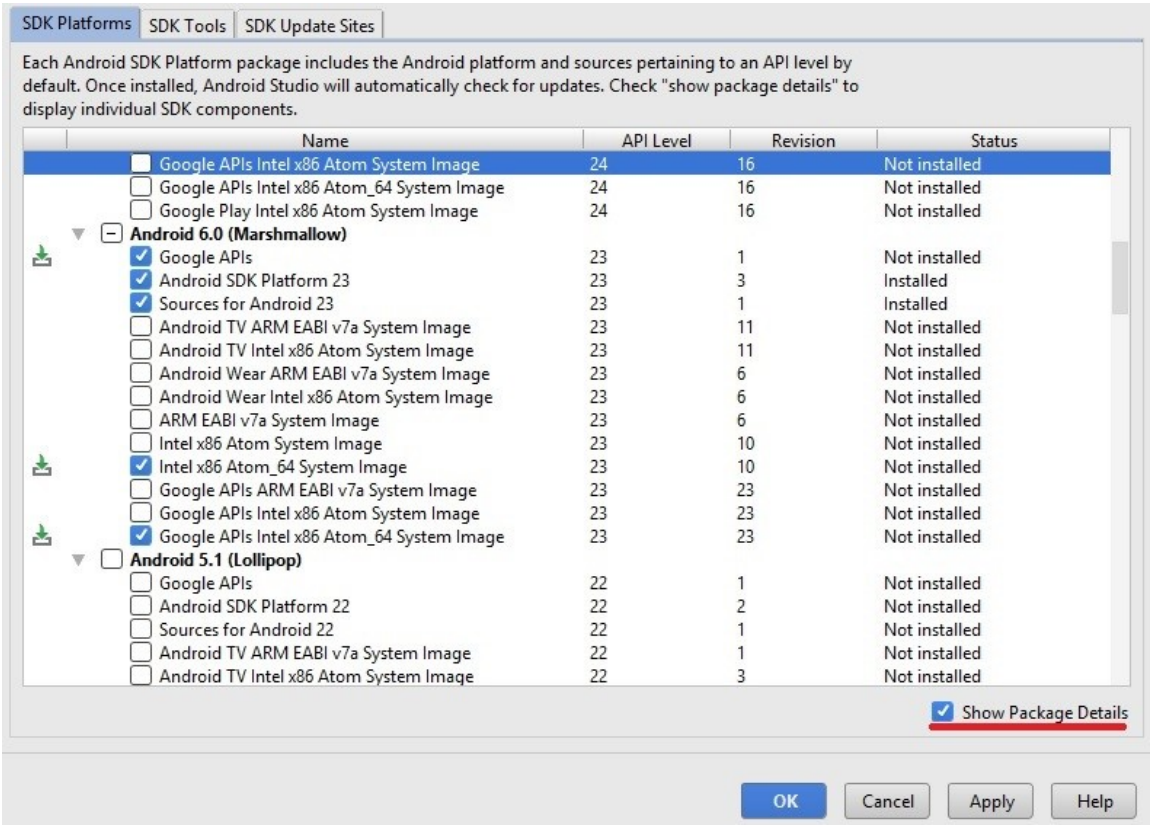
```
choco install -y nodejs.install python2 jdk8
```

Parametr `-y` příkazu install znamená potvrzení vyskakovacích oken, což je ve většině případů potvrzení licence. Pokud instalace proběhla v pořádku, tak je čas nainstalovat Android Studio. Tento program může sloužit jako IDE – vývojové prostředí, ale jeho instalace je důležitá z jiného důvodu. Slouží totiž jako grafické rozhraní pro stahování balíčků do Android SDK a nastavení emulátoru. Během instalace je důležité zvolit Custom Installation a zvolit instalaci následujících doplňků – Android SDK, Android SDK Platform, Performance (Intel ® HAXM), Android Virtual Device.

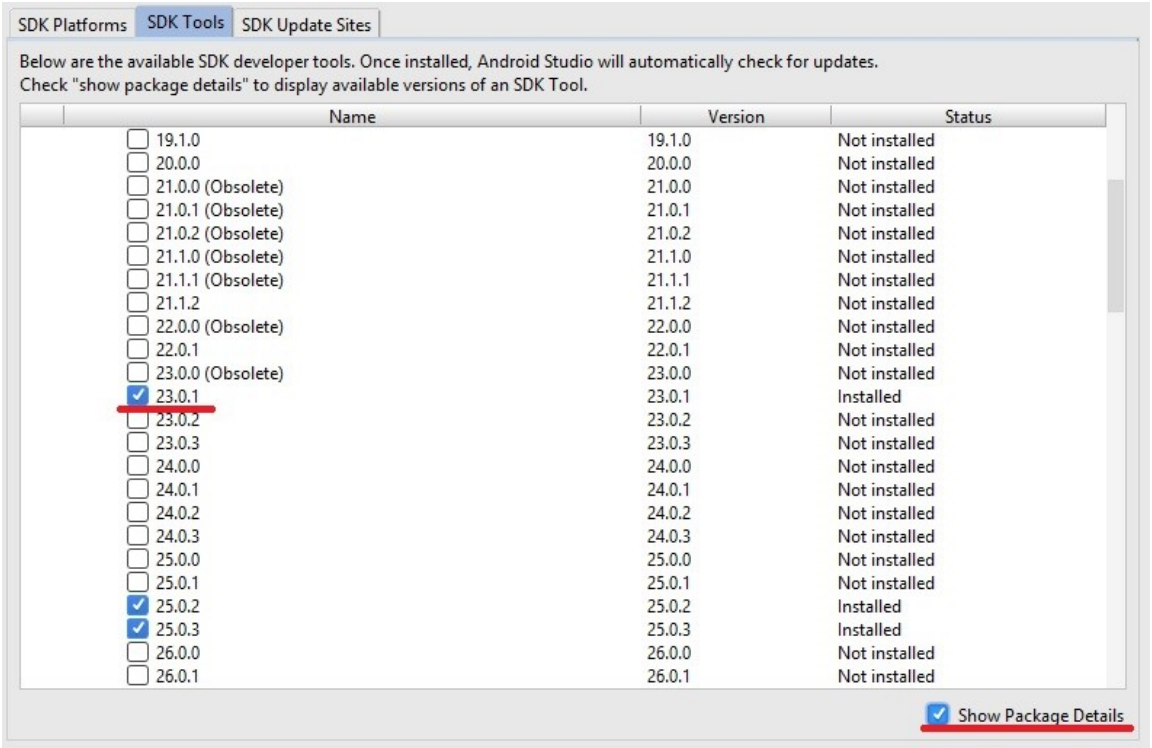


Obrázek 3.2: Správné nastavení instalace Android Studia

Po dokončení instalace a nastavení prostředí je nutné nainstalovat balíčky do Android SDK. Konkrétně se jedná o balíčky Google APIs, Android SDK Platform 23, Intel x86 Atom_64 System Image a Google APIs Intel x86 Atom_64 System Image. Okno pro instalaci balíčků se v aktuální verzi Android Studia nachází pod Tools → Android → SDK Manager. Aby byly tyto balíčky viditelné, je nutné zvolit možnost Show Package Details. Další důležitou součástí je nainstalovat Android SDK Build-Tool verze 23.0.1. Opět je nutné zvolit možnost Show Package Details.



Obrázek 3.3: Instalace povinných balíčků Android SDK



Obrázek 3.4: Instalace povinných nástrojů Android SDK

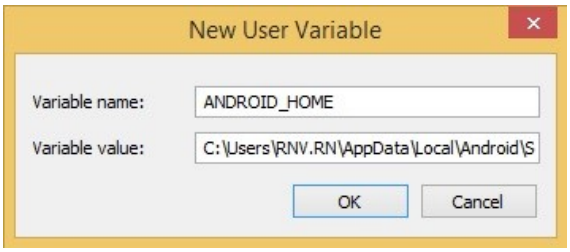
Posledním krokem je přidání proměnné ANDROID_HOME do uživatelských proměnných prostředí, aby mohly nástroje React Native kompilovat aplikace do nativního kódu. Toto okno lze vyvolat zadáním následujícího příkazu do okna Spustit.

```
rundll32 sysdm.cpl,EditEnvironmentVariables
```

Zde stačí přidat novou uživatelskou proměnnou. Jméno proměnné je tedy ANDROID_HOME a hodnota je cesta ke složce obsahující Android SDK. Výchozí cesta je

```
C:\Users\Uzivatelske_jmeno\AppData\Local\Android\Sdk
```

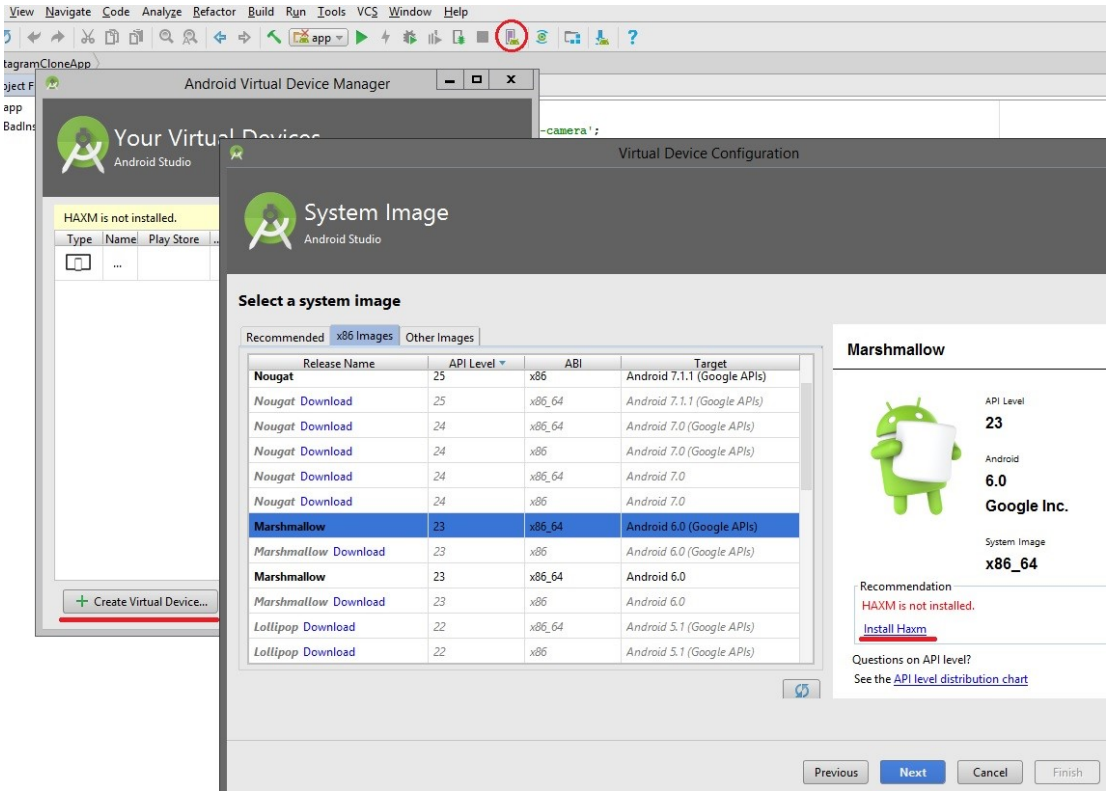
Cestu ke složce lze zjistit v Android Studiu pod nabídkou Tools → Android → SDK Manager.



Obrázek 3.5: Přidání uživatelské proměnné ANDROID_HOME

3.2 Emulátor

Nyní je nutné nastavit emulátor. Do nabídky emulátorů je možné se dostat z nabídky Tools → Android → AVD Manager, nebo kliknutím na ikonu zvýrazněnou na obrázku níže. Nový emulátor je vytvořen pomocí tlačítka Create Virtual Device. Typ zařízení není důležitý, protože se liší jenom v rozměrech a rozlišení displeje. Je doporučeno zvolit verzi operačního systému Android 6.0 Marshmallow s rozhraním x86_64. Pokud není nainstalovaný balík Intel ® HAXM, tak je nutné tvorbu emulátoru přerušit a tento balík nainstalovat, protože bez něj emulátor nebude fungovat.



Obrázek 3.6: Instalace emulátoru

Po vytvoření emulátoru je možné tento emulátor používat i mimo Android Studio. Tato funkce je užitečná zejména když Android Studio nevyužíváte jako vývojové prostředí. Prvním krokem je najít seznam vytvořených emulátorů. K tomuto poslouží funkce avdmanager, která se nachází ve složce s Android SDK.

```
..\Android\sdk\tools\bin
```

V této složce je nutné spustit příkazový řádek a následujícím příkazem vyvolat seznam vytvořených emulátorů.

```
avdmanager list avd
```

Důležitou informací je jméno emulátoru, který chceme spustit. Toto jméno je nutné zadat jako vstup programu emulator, jenž se nachází ve složce tools. Spuštění emulátoru má následující sytaxi. Proces je možné dále zjednodušit tvorbou dávkového souboru.

```
emulator -avd jmeno_emulatoru
```

Obrázek 3.7: Spouštění emulátoru a tvorba dávkového souboru.

3.3 Fyzické zařízení namísto emulátoru

Kromě emulátoru lze použít i fyzické zařízení. Tento proces probíhá stejně jako při klasickém vývoji aplikací na Android OS. Stačí zařízení připojit pomocí USB a povolit Ladění USB. Jestli byl telefon úspěšně rozpoznán je možné zkontrolovat zadáním následujícího příkazu do příkazového řádku. Pokud se zařízení v seznamu nachází, tak by mělo být použitelné při spouštění aplikace.

```
adb devices
```

3.4 Instalace React Native a první aplikace

React Native lze nyní nainstalovat pomocí balíčkovacího manageru NPM zadáním následujícího příkazu do příkazového řádku.

```
npm install -g react-native-cli
```

NPM lze vyvolat příkazem npm, protože se při instalaci sám přidal do PATH. Atribut –g znamená, že se balíček nainstaluje globálně spíše než lokálně. React Native byl také při instalaci přidán do proměnné PATH, takže je nyní dostupný pod zkratkou react-native. Nový projekt lze vytvořit zadáním následujícího příkazu do příkazového řádku spuštěném v administrátorském režimu.

```
react-native init Test
```

Toto vytvoří projekt v momentálně otevřené složce, který je již připraven ke spuštění. Pokud tvorba projektu proběhne v pořádku, tak je tento testovací projekt připraven ke spuštění. V první řadě je třeba spustit emulátor, nebo připojit fyzické zařízení, na které bude aplikace nainstalována. V příkazovém řádku stačí otevřít složku s projektem a použít následující příkaz. První instalace může trvat několik minut.

```
react-native run-android
```

Tato první aplikace je nyní připravena na úpravu. V libovolném textovém editoru nebo vývojovém prostředí stačí upravit soubor index.android.js nacházející se ve složce s projektem. Po úpravě stačí v emulátoru dvakrát zmáčknout písmeno R aby se projevily změny v kódu. Pokud aplikace na zařízení spadne, ale React Native Packager stále běží, tak je možné aplikaci rychle opět spustit z menu nainstalovaných aplikací na zařízení.

4 Vývoj aplikací ve frameworku

Tato kapitola popisuje nutné znalosti a základní principy potřebné k vývoji aplikací ve frameworku React Native. Principy vývoje jsou přizpůsobeny převážně pro webové vývojáře. Hlavním předpokladem, ale zároveň také výhodou, je JavaScript verze ES6, který se pomalu stává standardem ve webových prohlížečích a nabízí proti starším verzím několik výhod. Systém komponentů je záměrně velmi podobný komponentům z HTML a stylování probíhá stejným způsobem jako v CSS. I pokud vývojáři není známý webový framework React na kterém je React Native postaven, tak by měl být schopen rychle začít s vývojem mobilních aplikací s pouhou znalostí tvorby webových stránek.

4.1 Syntaxe

React Native využívá JavaScriptovou syntaxi ES2015. Jedná se o oficiální standard vydaný v červnu 2015. ES2015, také známý pod zkratkou ES6 je prvním velkým vylepšením od vydání ES5 v roce 2009 a přináší moderní praktiky a funkce z C# a Java 8. I přesto že se jedná o standard z roku 2015, tak dnes není běžně používaný z toho důvodu, že není plně podporovaný ve všech webových prohlížečích. Nejdůležitějšími a často používanými funkcemi ES6 v React Native jsou šipkové funkce, nové typy proměnných a moduly.

4.2 Arrow functions

Šipky neboli Arrows je alternativní zápis funkcí v ES6. Kromě toho, že zkracuje syntaxi má jeden velký rozdíl oproti standardnímu zápisu. Jeho hodnota `this` pochází od rodiče a uvnitř funkce nevytváří další rámec pro hodnotu `this`. Tento zápis je doporučeno používat ve všech případech kromě těch, kde je funkce použita jako konstruktor, generátor nebo kde je žádoucí využít nový rámec `this` klasického zápisu funkce.

```
1 //ES5
2 let secti = function(a, b) {
3 return a + b;
4 };
5 //ES6
6 let secti = (a, b) => { return a + b; }
```

Ukázky kódu 4.1: Porovnání šipkového a normálního zápisu funkce

Pokud má funkce jeden argument, tak ho není nutné uzavřít do závorek. Závorky se uvádějí pouze v případě, že má funkce žádný, nebo více než jeden argument.

```
1 let log = () => { consloe.log(this); }
2 let log = text => { console.log(text); }
```

Ukázky kódu 4.2: Využití argumentů v šipkových funkcích

4.3 Proměnné

Proměnné typu `let` a `const` jsou nové v ES6 a mají za úkol nahradit proměnnou typu `var`. Obě tyto nové proměnné se liší tím, že jejich rámec je blokový namísto rámce celé funkce, kde je proměnná deklarována. Toto má za výsledek větší přehlednost kódu, protože pokud je žádoucí aby proměnná využívala rámec celé funkce nebo třídy, tak stačí tuto proměnnou umístit na začátek.

Tím se vyjasňuje zamýšlený účel této proměnné jak při vývoji, tak při dodatečných modifikacích kódu. Rozdíl mezi proměnnou typu `let` a `const` je, že proměnná typu `const` je konstantní. Nejde tedy měnit její obsah a používá se tehdy, kdy předpokládáme, že se proměnná nebude měnit. Jediné, co se na proměnné typu `const` může měnit jsou vlastnosti objektu. Proměnná typu `var` byla nahrazena proměnnou typu `let` jako lokální proměnná.

4.4 Moduly

Další novinkou v ES6 je implementace modulů. *„Javascript měl moduly dlouhou dobu, byly však implementovány skrze knihovny a nebyly zabudovány přímo do samotného jazyku. ES6 je poprvé co má JavaScript zabudované moduly. Moduly v ES6 jsou uloženy v souborech. Existuje právě jeden modul na soubor a jeden soubor na modul. Jsou dva způsoby jak exportovat věci z modulu. Tyto dva způsoby mohou být kombinovány, ale většinou je lepší je používat samostatně.“*[9]

Prvním takovýmto způsobem je více pojmenovaných exportů. Při importování se názvy dávají do složených závorek a jsou oddělené čárkou. Pokud tímto způsobem importujeme pouze jednu funkci, třídu nebo proměnnou, tak se závorky nepoužívají. Překladač totiž vyhodí chybu, že nemůže uvedený soubor nalézt.

```
1 //komponenty.js
2 export class MujKomponent extends React.Component { ... }
3 export const stylyKomponentu = StyleSheet.create({ ... })
4 //index.js
5 import { MujKomponent, stylyKomponentu } from ./komponenty';
```

Ukázky kódu 4.3: Exportování a importování modulů

Další možností je využití znaku `*` namísto názvů, čímž se importují všechny exporty ze souboru. Zde je nutné takovýto modul pojmenovat za pomocí klíčového slova `as`. Přistupuje se k nim potom jako ke vlastnostem.

```
1 import * as komponenty from ./komponenty';
2 <komponenty.MujKomponent />
```

Ukázky kódu 4.4: Jmenné importování modulu

Posledním způsobem je `default` export. Takovýto export může být v souboru pouze jeden a není třeba uvádět jeho jméno při deklaraci ani při importu. K jeho identifikaci stačí pouze název souboru ve kterém se nachází.

```
1 //MujKomponent.js
2 export default class extends React.Component { ... }
3 //index.js
4 import MujKomponent from './MujKomponent';
```

Ukázky kódu 4.5: Default export

4.5 JSX

JSX je syntax pro zapojení XML do JavaScriptu, jejíž úkolem je definovat komponenty a to, jak mají vypadat. Je možné této funkce nevyužít a spoléhat se pouze na JavaScript, ale tento zápis je mnohem jednodušší a přehlednější.

```
1 React.createElement(View, {style: styles.container},
2 React.createElement(Text,{style: styles.welcome},"Text"))
3
4 <View style={styles.container}>
5       <Text style={styles.welcome}>Text</Text>
6 </View>
```

Ukázky kódu 4.6: Porovnání mezi deklarací komponentu v JSX a JavaScriptu

4.6 Komponenty

Komponenty v React Native slouží ke skládání uživatelského rozhraní. Jde o předpřipravené prvky jako je například text, tlačítko nebo textové pole, které univerzálně fungují na všech podporovaných platformách. Kromě těchto základních komponentů lze definovat vlastní komponenty, nebo importovat komponenty tvořené komunitou. Kromě univerzálních komponentů existují i komponenty nativní, které jsou exkluzivní pro svoji platformu, protože není možné je spolehlivě implementovat na platformy ostatní. Podobně jako v HTML se komponenty dělí na ty s párovými a nepárovými tagy. Komponenty s párovými tagy přijímají potomky buď ve formě dalších komponentů anebo textu. Před tím než lze komponent použít je třeba ho importovat z modulu `react-native`.

```
<View>
      <Text>Hello World!</Text>
      <Image source={require('./img/obrazek.jpg')}/>
</View>
```

Ukázky kódu 4.7: Párový a nepárový komponent

```
import {
      AppRegistry,
      Text,
      View,
      Image,
} from 'react-native';
```

Ukázky kódu 4.8: Importování React Native komponentu

4.6.1 Props

Většina komponentů má k dispozici vlastnosti neboli `props`, jako například `style`, `onPress` nebo `source`, které umožňují manipulaci nebo práci s handlery[8] daného komponentu. Každý zabudovaný komponent má k dispozici jiné vlastnosti. Tyto vlastnosti lze nalézt v dokumentaci React Native.

[8] Handler – funkce propojená s událostí

```
<View style={{backgroundColor: 'red'}}>
      <Image source={require('./img/obrazek.jpg')}/>
      <Button onPress={() => {console.log("Klik");}} title="Tlačítko" />
</View>
```

Ukázky kódu 4.9: Vlastnosti komponentů

4.6.2 Vlastní komponenty

Tvorba vlastních komponentů slouží pro přehlednost a znovupoužitelnost komponentů. Tyto vlastní komponenty lze tvořit pomocí skládání zabudovaných komponentů a definováním vlastních `props`.

```
//LoginName.js
import React from 'react';
import {
      AppRegistry,
      Text,
      View,
} from 'react-native';
export default class LoginName extends React.Component {
      render() {
          return (
               <View style={{padding: 15, backgroundColor:  'skyblue'}}>
                    <Text style={{fontSize: 30, textAlign:
                    'center'}}>Přihlášen jako:
                    {this.props.loginName}</Text>
               </View>
          );
      }
}
//index.android.js
import LoginName from './app/components/LoginName';
class Test extends Component {
      render() {
          return (
               <LoginName loginName={"Administrator"} />
          );
      }
}
```

Ukázky kódu 4.10: Tvorba vlastního komponentu

Obrázek 4.1: Reálné zobrazení vlastního komponentu

Je také možné nastavit základní parametry komponentu pomocí `defaultProps`. To způsobí, že i když komponent při vytvoření žádné parametry nedostane, tak bude vytvořen s parametry základními.

```
1 export default class LoginName extends React.Component { ... }
2 LoginName.defaultProps = {
3       loginName: 'Anonym'
4 };
```

Ukázky kódu 4.11: Nastavení základních vlastností komponentu

Pro vytvoření komponentu s párovými tagy stačí využít `props.children`. Toto umístí jakýkoliv obsah uvnitř tagů na místo tohoto příkazu. Pokud je účelem komponentu obalit text, tak je nutné při tvorbě komponentu umístit příkaz `this.props.children` do komponentu `<Text>`, jinak překladač vrátí chybu. Pokud vytvářený komponent má za účel

obalit jiné komponenty, jako například komponent `<View>`, tak není žádný speciální postup nutný.

```
1  //deklarace komponentu
2  export default class H1 extends React.Component {
3        render() {
4            return (
5                 <View>
6                      <Text style={{fontSize: 30, fontWeight:
7                      'bold'}}>{this.props.children}</Text>
8                 </View>
9            );
10       }
11 }
12 //využití komponentu
13 <H1>Nadpis 1</H1>
```

Ukázky kódu 4.12: Tvorba vlastního párového komponentu

4.6.3 Styly

Stylování probíhá velmi podobně jako v HTML s několika malými změnami v syntaxi. Každý prvek má k dispozici jiné parametry při stylování, ale je možné vždy jakýkoliv komponent obalit komponentem `<View>`.

```
1  <View style={{padding: 15, backgroundColor: 'skyblue'}}>
2        <Text style={{fontSize: 30, textAlign: ' center'}}>Text</Text>
3  </View>
```

Ukázky kódu 4.13: Inline stylování komponentu

Styly lze také deklarovat pro vícenásobné použití. Tato metoda je doporučena z důvodu přehlednosti a optimalizace. Přesunou se tím z funkce `render()` což způsobí, že nebudou renderovány[9] při každém načtení. Tyto deklarované styly je také možné importovat a exportovat jako jiné proměnné.

[9] Renderování - vykreslování

```
1  //styles.js
2  export const styly = StyleSheet.create({
3        container: {
4        justifyContent: 'center',
5        backgroundColor: 'skyblue',
6  },
7  h1: {
8        fontSize: 25,
9        margin: 10,
10       fontWeight: 'bold',
11 }});
12 //index.android.js
13 import styly from './app/styles/styles;
14 class Test extends Component {
15       render() {
16           return (
17                <View style={styly.container}>
18                <Text style={styly.h1 }>Nadpis 1</Text>
19       </View>
20 );}}
```

Ukázky kódu 4.14: Stylování pomocí proměnné

4.6.4 Dynamické styly

Jedním ze způsobů, jak dosáhnout dynamických stylů je s využitím `props`. Toto umožňuje ovlivňovat vzhled individuálních komponentů, i když jejich základní vzhled je založený na společné šabloně. K tomuto slouží funkce flatten třídy StyleSheet. Tato funkce spojuje několik stylů do jednoho. To umožňuje spojit definovaný styl komponentu se stylem vytvořeným pomocí props. Přidáním proměnné this.props.style do funkce flatten lze přidat do kombinace i in-line styly[10]. Pokud je žádoucí ovlivňovat styly podle vstupu od uživatele, tak je nutné využití stavů.

[10] In-line styly – styly zapsané v deklaraci komponentu

```
export default class Box extends React.Component {
      render() {
          let bgcolorstyle = StyleSheet.create({
          boxcolor: {
          backgroundColor: this.props.bgcolor,
          }
      })
      let combinedstyle = StyleSheet.flatten([boxstyle.box,
      bgcolorstyle.boxcolor, this.props.style]);
      return (
          <View style={combinedstyle}>
               {this.props.children}
          </View>
      );}}
      const boxstyle = StyleSheet.create({
          box: {
          width: 50,
          height: 50,
      },
});
```

Ukázky kódu 4.15: Kombinování stylů

4.6.5 States

Komponenty využívají states neboli stavy pro data měnící se v průběhu času. Konkrétně by tedy měly být stavy využity pouze v případě, kdy aplikace přijme vstup od uživatele, nebo se obsah mění na základě časovače. Při práci se stavy je *„běžnou praktikou vytvoření několika bezstavových komponentů, které pouze vykreslují data a mají nad sebou jeden komponent, který stavů využívá a posílá tyto stavy svým potomkům pomocí props. Tento stavový komponent zapouzdřuje veškerou interakční logiku, zatímco se ty bez stavové starají o renderování dat deklarativní cestou.“*[10] Tím je myšleno, že stavy jsou měněny ve funkci render, kde se nachází importované komponenty. Při deklaraci komponentu je nastavován pouze základní stav. Prvním krokem k použití stavů je incializace. Ta by měla být provedena v konstruktoru komponentu. Kvůli jednoduchosti je doporučeno používat jako hodnotu stavů booleanovské hodnoty.

```
class Container extends Component {
      constructor() {
          super();
          this.state = {viditelny: true};
}}
```

Ukázky kódu 4.16: Inicializace stavu

Stavy mají k dispozici několik funkcí a proměnných umožňujících jednoduchou manipulaci. Funkci `this.setState({ klic: hodnota })` sloužící ke změně stavu, proměnnou `this.state.klic` sloužící k získání aktuálního stavu a proměnnou `prevState.klic` k získání předchozího stavu. Proměnná `this.state.klic` obsahuje aktuální hodnotu stavu a slouží pro předání stavu k dalším operacím, jako je například uložení tohoto stavu do `props` komponentu. Funkce `setState()` by měla být jediným způsobem jak bude manipulováno se stavem. Po změně stavu dojde k automatickému vykreslení celého komponentu a jeho potomků, takže není nutné se starat o aktualizaci manuálně. Toto vykreslení nemusí nutně proběhnout okamžitě. React Native tyto požadavky může zdržet a provést několik aktualizací současně. Dále je doporučeno provádět více změn stavů najednou, aby nedocházelo ke zbytečným vykreslením. Požadavky na změnu stavu by měly být, pokud je to možno, řetězeny do jediného příkazu `setState()`.

```
this.setState({ klic: hodnota, klic: hodnota })
render() {
      let hodnotaTextu = this.state.viditelny ? "Text" : " ";
      return (
          <View>
               <Text>{hodnotaTextu}</Text>
               <Button title={"viditelny: " + this.state.viditelny}
               onPress={() => {     this.setState({ viditelny:
               !this.state.viditelny }) }} />
          </View>
);}}
```

Ukázky kódu 4.17: Využití stavů ke změně komponentu

Obrázek 4.2: Reálné zobrazení změny stavu komponentu

„Proměnná `prevState` slouží k získání předchozího stavu. Neměla by být měněna přímo. Místo toho by měla být reprezentována vytvořením nového objektu založeném na vstupu z `prevState` a `props`.“[11] Tato proměnná nachází využití při změnách stavu pomocí setState(). Je u ní totiž zaručeno, že je aktuální.

```
1 <View>
2       <Button title={""+this.state.hodnota} onPress={() => {
3          this.setState((prevState) => {
4               return { hodnota: prevState.hodnota + 1 }
5       })}} />
6 </View>
```

Ukázky kódu 4.18: Využití vlastnosti prevState

4.6.6 Text input

Textový vstup je jedním z nejdůležitějších konceptů, kterými může uživatel s aplikací komunikovat. K tomuto slouží komponent `TextInput`. Ten zprostředkovává textové pole, které po interakci s ním otevře nativní klávesnici. K dispozici má dvě hlavní vlastnosti a to `onChangeText` a `onSubmitEditing`. Vlastnost `onChangeText` je spuštěna při každé změně textového pole a `onSubmitEditing` při zmáčknutí tlačítka potvrzení. Text je ukládán ve stavu, protože se průběžně mění.

```
1  class Test extends React.Component {
2        constructor(props) {
3            super(props);
4            this.state = {text: ''};
5        }
6        render() {
7            return (
8                 <View>
9                      <TextInput onChangeText={(text) =>
10                     this.setState({text})} />
11                     <Text>{this.state.text}</Text>
12                </View>
13           );
14       }
15 }
```

Ukázky kódu 4.19: Využití komponentu TextInput

4.6.7 Seznamy

Komponenty `FlatList` a `SectionList` slouží k vykreslování dynamicky generovaných seznamů z dat umístěných v poli. Tyto komponenty se hodí převážně pro data u kterých může dojít ke změně. Data komponentu jsou definovány vlastností `data` a formát buňky je definován parametrem `renderItem`. Rozdílem mezi komponentem `FlatList` a `SectionList` je, že `SectionList` umožňuje seznam strukturovat pomocí nadpisů nebo kategorií. Každá položka musí mít přiřazený klíč, neboli `key`, který ji jednoznačně identifikuje. Překladač v opačném případě vypíše varování. Pokud tedy objekty v seznamu který je použit jako data mají pouze jeden údaj, tak tento údaj musí mít klíč s názvem `key` i v případě, že tento klíč není použit ve vlastnosti `renderItem`.

```
1  class Test extends React.Component {
2        render() {
3        let polozky = [
4            {key: 1, polozka: 'Polozka1'},
5            {key: 2, polozka: 'Polozka2'},
6            {key: 3, polozka: 'Polozka3'},
7            {key: 4, polozka: 'Polozka4'},
8        ];
9        return (
10           <View>
11                <FlatList data={polozky} renderItem={({item}) =>
12                <Text>{item.polozka}</Text>} />
13           </View>
14           );
15       }
16 }
```

Ukázky kódu 4.20: Implementace seznamu pomocí komponentu FlatList

4.7 Layout

Uspořádání komponentů v React Native funguje podobně jako v CSS ale s několika rozdíly. Pokud není ve stylech řečeno jinak, tak komponent zabírá celou šířku obrazovky a jeho výška je určena výškou obsahu. Většina komponentů po vykreslení vytvoří nový řádek. Výška a šířka komponentů může být specifikována stylovacími vlastnostmi `width` a `height`. Tyto hodnoty jsou bezjednotkové a určují velikost nezávislou na hustotě a rozlišení displeje. Měly by být tedy použity pro komponenty, jejichž velikost má být vždy stejná a nezávislá na zařízení. Dále lze rozvržení komponentů ovládat vlastnostmi `left`, `right`, `top` a `bottom`, které ovládají odsazení kraje komponentu. Velikost může být určena jak v jednotkách, tak v procentech. Rozvržení komponentů je také možné ovlivnit parametrem margin, který určuje vnější okraj komponentu nebo parametrem padding, který určuje vnitřní okraj komponentu.

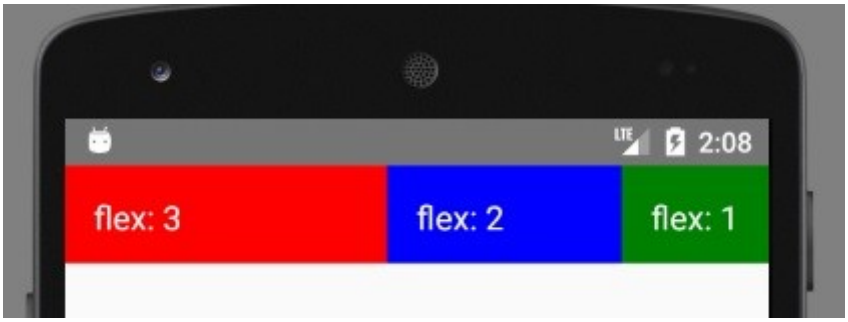
4.7.1 Flex

Flex je systém podobný flexboxu z CSS. Velikost komponentů je určena s ohledem na ostatní potomky a dostupné místo. Pro použití tohoto uspořádání je nutné u rodičovského komponentu pevně určit velikost, nebo nastavit hodnotu `flex`, jinak bude nastavena velikost

na 0. To má za důsledek, že se potomci nezobrazí, protože tato velikost nemůže být rozdělena mezi potomky. Číslo u parametru flex určuje velikost oproti ostatním potomkům.

```
<View style={{ height: 50, flexDirection: 'row'}}>
      <Box style={{ flex: 3 }}></Box>
      <Box style={{ flex: 2 }}></Box>
      <Box style={{ flex: 1 }}></Box>
</View>
```

Ukázky kódu 4.21: Velikosti v rozložení flex

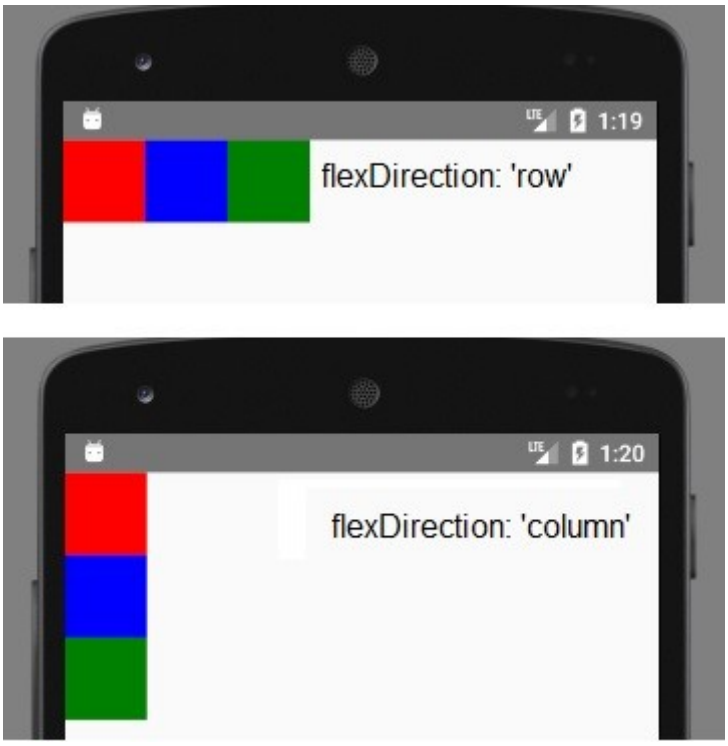


Obrázek 4.3: Zobrazení velikostí v rozložení flex

Rozmístění potomků je řízeno pomocí vlastností `flexDirection`, `justifyContent` a `alignItems`. `flexDirection` určuje primární osu podél které budou prvky rozmístěny na základě hodnot `row` – do řádku, nebo `column` – do sloupce.

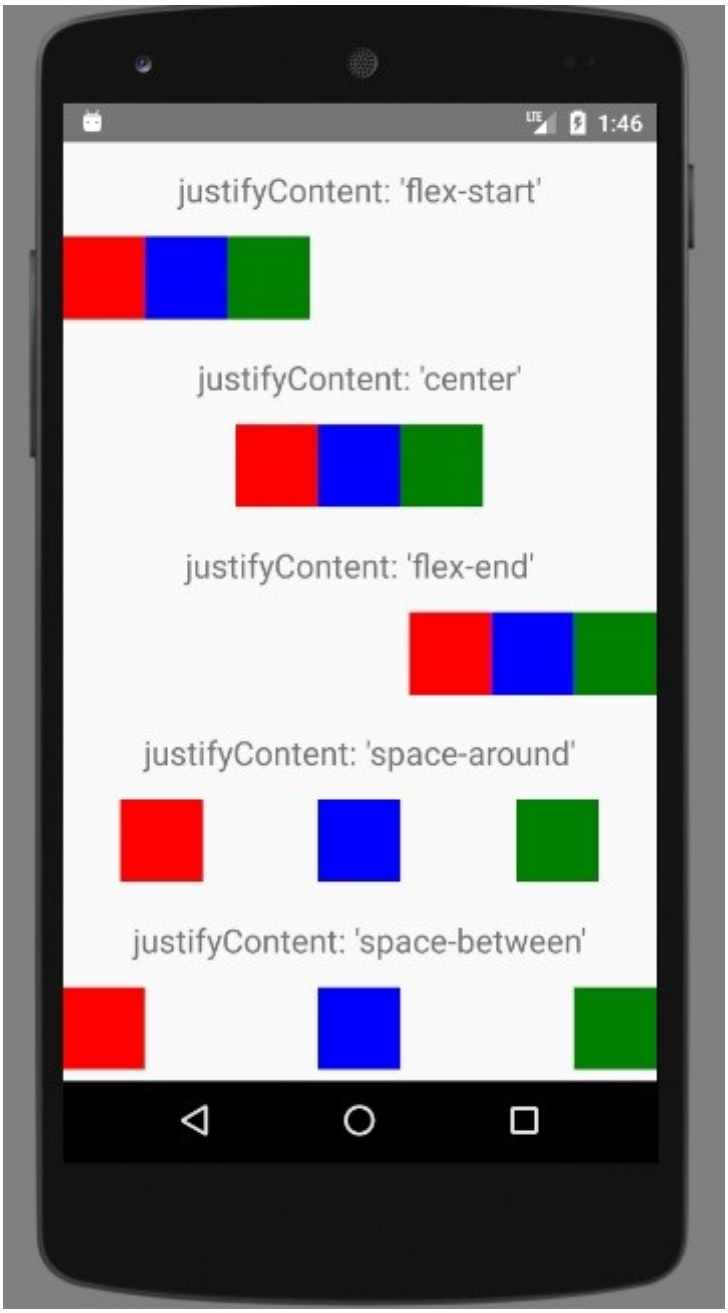
```
<View style={{flex: 1, flexDirection: 'row'}}>
      <Box bgcolor={'red'}></Box>
      <Box bgcolor={'blue'}></Box>
      <Box bgcolor={'green'}></Box>
</View>
```

Ukázky kódu 4.22: Využití parametru flexDirection



Obrázek 4.4: Ukázka využití parametru flexDirection

Vlastnost `justifyContent` určuje rozmístění prvků na primární ose. Prvky mohou být různě rozmístěny po primární ose s tím, že jsou rozmístěny rovnoměrně pokud není jinak určeno. Potomci mohou být všichni umístěni na začátek – `flex-start`, konec – `flex-end` nebo doprostřed – `center` rodičovského komponentu. Mohou také mít rovnoměrné mezery mezi sebou – `space-between` nebo kolem sebe – `space-around`.



Obrázek 4.5: Ukázka využití parametru justifyContent

Parametrem `alignItems` lze určit sekundární osu, podle které se budou komponenty řadit. Pokud je tedy hlavní osu určuje parametr `flexDirection: 'row'`, tak parametr `alignItems` určuje pohyb po ose X a pokud hlavní osu určuje parametr `flexDirection: 'column'`, tak určuje pohyb po ose Y. K dispozici jsou hodnoty `flex-start`, `center`, `flex-end` a `stretch`. Hodnota `stretch` roztáhne potomkovské komponenty tak, aby zabíraly celý dostupný prostor. Nefunguje však v tom

případě, kdy je šířka pevně definována parametrem `width` nebo `height` s ohledem na hodnotu sekundární osy.

4.8 Struktura aplikace

Všechny aplikace v React Native mají základní strukturu, kterou je nutné dodržet. Jelikož třída aplikace rozšiřuje třídu `Component`, tak je nutné importovat třídu `Component` z modulu `react`. Dále je nutné importovat moduly `AppRegistry` a `View`. *„AppRegistry je JavaScriptový vstupní bod pro spouštění všech aplikací v React Native. Kořenové komponenty aplikace by měly být registrovány pomocí AppRegistry.registerComponent. Poté může nativní systém načíst celý balík aplikace a spustit ji, když je připravena zavoláním AppRegistry.runApplication.“*[12] Komponent `View` je nutné importovat, protože všechny komponenty nacházející se v návratové funkci funkce `render` musejí být zabaleny v kořenovém komponentu `<View>`. Nakonec je nutné registrovat třídu aplikace do `AppRegistry`, aby bylo možné aplikaci spustit.

```
1  import React, { Component } from 'react';
2  import {
3        AppRegistry,
4        View,
5  } from 'react-native';
6
7  class Aplikace extends Component {
8        render() {
9            return (
10                <View></View>
11           );
12       }
13 }
14 AppRegistry.registerComponent('Aplikace', () => Aplikace);
```

Ukázky kódu 4.23: Základní struktura aplikace

4.9 Navigation

Pro implementaci několika obrazovek a přepínání mezi nimi je nutné importovat modul react-navigation. Tento modul se stará o vzhled, funkčnost a přechody mezi obrazovkami. Modul je možné stáhnout z repozitáře NPM pomocí příkazu `npm install --save react-`

`navigation`. Modul umožňuje strukturovat kód do několika obrazovek, mezi kterými následně přepíná třemi různými typy navigátorů. Pokud je aplikace cílená nebo speciálně upravena pro iOS, tak je možné využít komponentu `NavigatorIOS`. Tento navigátor, jak již název napovídá, je podporovaný pouze na iOS. Nabízí více nativních vlastností, jako například vzhled a animace, ale jeho funkčnost a obsluha je oproti klasickému navigátoru rozdílná.

4.9.1 Stack navigator

`StackNavigator` je základním typem navigace, která se vyznačuje navigační lištou v základním nastavení na horní straně obrazovky. Tato navigační lišta obsahuje titulky obrazovek a navigační šipku, která slouží k navigaci o krok zpátky. Navigační lištu je možné vypnout změnou parametrů, což je obzvlášť užitečné v případě, že navigace využívá několik složených StackNavigatorů. Parametry také umožňují měnit vzhled této lišty. Mezi obrazovkami se přepíná pomocí funkcí, které odkazují na příslušnou obrazovku. Tyto funkce se vkládají obecně do vlastnosti `onPress` u tlačítek. Prvním krokem k implementaci StackNavigatoru je import modulu.

```
import { StackNavigator } from 'react-navigation'
```

Dále je nutné obrazovku deklarovat. Tato třída nahrazuje hlavní třídu aplikace a je tedy v ní umístěn veškerý obsah titulní stránky. Hlavní třída aplikace se mění na proměnnou s cestami navigátoru.

```
1  class HlavniObrazovka extends React.Component {
2        static navigationOptions = {
3           title: 'Vítejte',
4        };
5        render() {
6           return (
7                <Text>Vítejte na hlavní obrazovce</Text>
8           );
9        }
10 }
```

Ukázky kódu 4.24: Deklarace obrazovky komponentu StackNavigator

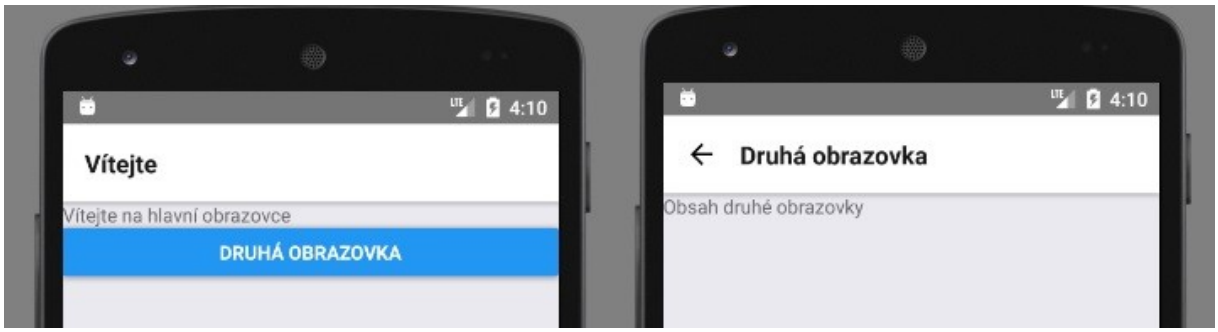
```
1 const Test = StackNavigator ({
2       Hlavni: { screen: HlavniObrazovka },
3 });
```

Ukázky kódu 4.25: Deklarace komponentu StackNavigator

Přidání nové obrazovky probíhá stejně jako přidání té hlavní. V tomto bodě začíná být rozumné nové obrazovky umisťovat do samostatných souborů a importovat je pro lepší přehlednost. Hlavní změnou v následujícím kódu je přidání tlačítka, které umožňuje navigaci na druhou obrazovku. K tomuto slouží funkce `navigate`, která musí být inicializována ve funkci `render`. Parametry navigace, jako například titulek nebo barva navigátoru, jsou měněny v `navigationOptions`. Každý navigátor má k dispozici jiné možnosti. Ty jsou popsány v dokumentaci react-navigation.

```
class HlavniObrazovka extends React.Component {
      static navigationOptions = {
          title: 'Vítejte',
      };
      render() {
          const { navigate } = this.props.navigation;
          return (
               <View>
                    <Text>Vítejte na hlavní obrazovce</Text>
                    <Button title={'Druhá obrazovka'} onPress={() =>
                    navigate('Druha')} />
               </View>
          );
      }
}
class DruhaObrazovka extends React.Component {
      static navigationOptions = {
          title: 'Druhá obrazovka',
      };
      render() {
          return (
               <Text>Obsah druhé obrazovky</Text>
          );
      }
}
const Test = StackNavigator({
      Hlavni: { screen: HlavniObrazovka },
      Druha: { screen: DruhaObrazovka },
});
```

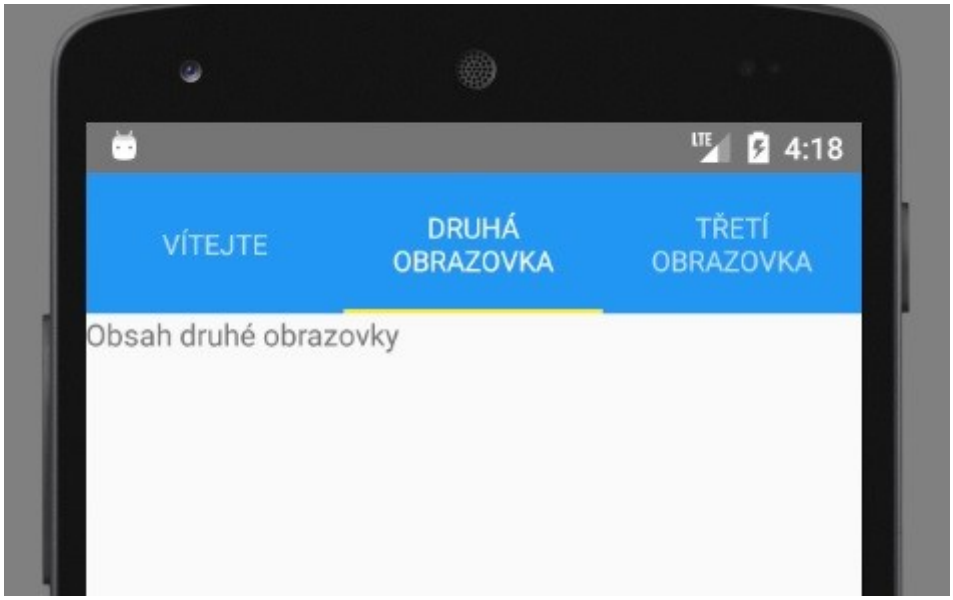
Ukázky kódu 4.26: StackNavigator se dvěma obrazovkami



Obrázek 4.6: Ukázka komponentu StackNavigator

4.9.2 Tab navigator

`TabNavigator` je rozdílný oproti `StackNavigator` v tom, že je možné vidět všechny obrazovky umístěné v navigátoru najednou. Navigační lišta obsahuje záložky s titulky obrazovek. Stejně jako u ostatních navigátorů jsou k dispozici parametry pro stylování navigační lišty. Implementace TabNavigatoru probíhá stejně jako u StackNavigatoru. Stačí importovat příslušný modul a v proměnné aplikace změnit typ navigátoru. Není zde však už nutné tlačítko pro změnu obrazovky, protože už je obsažené v samotné navigaci.



Obrázek 4.7: Ukázka komponentu TabNavigator

4.9.3 Drawer navigator

Navigace v DrawerNavigatoru je dostupná z takzvaného šuplíku, který se nachází v základním nastavení na levé straně obrazovky. Tento šuplík je vysouván pomocí gesta přejetím ze strany obrazovky. Pomocí dostupných funkcí lze však tuto pozici změnit na pravou stranu, změnit položky v šuplíku nebo tento šuplík nastylovat. Je také možné přidat funkce pro otevírání a zavírání šuplíku, pokud není základní ovládání dostačující.

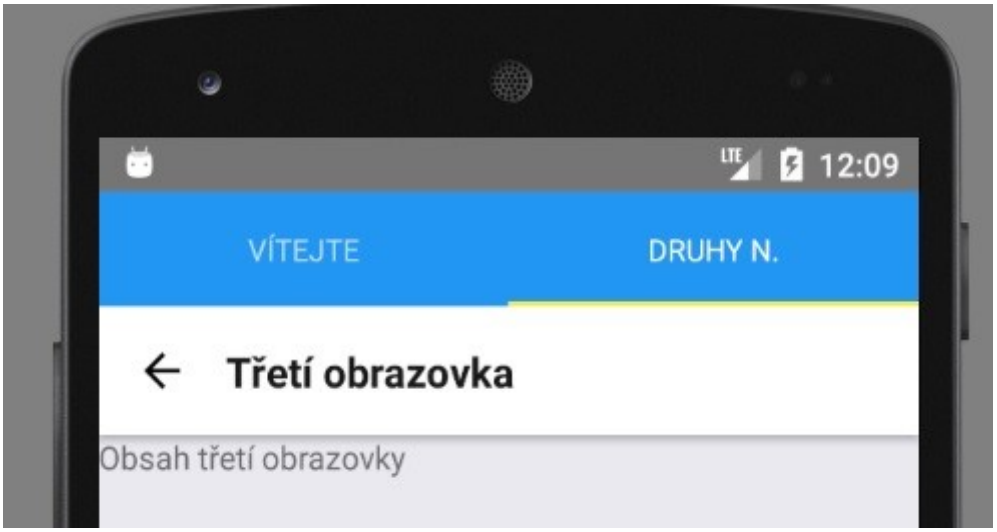
4.9.4 Skládání navigátorů

Předchozí navigátory umožňují tvorbu jednoúrovňové navigace. Toto však u složitějších aplikací nestačí a je nutné tyto navigátory skládat. Skládání navigátorů je jednoduché a je provedeno tím, že místo odkazu na obrazovku je použit odkaz na další navigátor. Tento

postup není doporučený u několika TabNavigatorů, protože podpora gest při navigaci u dvou TabNavigatorů současně není plně funkční.

```
const DruhyNavigator = StackNavigator({
      Druha: { screen: DruhaObrazovka },
      Treti: { screen: TretiObrazovka },
});
DruhyNavigator.navigationOptions = {
      title: 'Druhy N.',
};
const Test = TabNavigator({
      Hlavni: { screen: HlavniObrazovka },
      DruhyNavigator: { screen: DruhyNavigator },
});
```

Ukázky kódu 4.27: Skládání navigátorů



Obrázek 4.8: Ukázka složených navigátorů

4.9.5 Posílání parametrů

Posílání parametrů v cestě navigátoru pomáhá přestoupit od statických cest k cestám dynamickým. Umožňuje totiž přeposlat proměnné a objekty v cestě, čímž dynamicky mění obsah otevřeného okna. Je tedy možné při přechodu poslat například jméno uživatele, objekt s daty nebo styly, a využít je na otevřené obrazovce. Šablona obrazovky se tedy nemění a obsah je generován na základě přeposlaných parametrů.

```
render() {
      const { navigate } = this.props.navigation;
      let uziv = { jmeno: "Petr", prijmeni: "Novak", id: 2 };
      return (
          <View>
               <Text>Vítejte na hlavní obrazovce</Text>
               <Button
                    onPress={() => navigate('Profil', { uzivatel: uziv
                    title={uziv.jmeno + " " + uziv.prijmeni} />
          </View>
})}
class Profil extends React.Component {
      static navigationOptions = ({ navigation }) => ({
          title: navigation.state.params.uzivatel.jmeno + " " +
          navigation.state.params.uzivatel.prijmeni,});
      render() {
          const { params } = this.props.navigation.state;
          return (
               <View>
                    <Text>Jmeno: {params.uzivatel.jmeno}</Text>
                    <Text>Prijmeni: {params.uzivatel.prijmeni}</Text>
                    <Text>ID: {params.uzivatel.id}</Text>
               </View>
);}}
```

Ukázky kódu 4.28: Odesílání parametrů na obrazovku Profil

4.10 Storage

Základním způsobem jak v React Native ukládat data jsou stavy. Ty však nepřetrvávají při restartu aplikace. Pro paměť, která přetrvá i při restartu, je nutné využít funkci `AsyncStorage`. Tato funkce ukládá asynchroně a nezašifrovaně data na principu klíč – hodnota. Hodnoty musí být řetězcové, takže pro ukládání objektů je nutné data převést pomocí `JSON.stringify()` a při vybrání hodnoty z úložiště `JSON.parse()`. Každá metoda vrací skutečné chybové objekty, takže je možné využít bloky `try...catch`. Jak je již z názvu patrné, tak se jedná o úložiště asynchronní. To se projevuje tím, že je možné využít takzvaných slibů – `Promise`, které umožňují asynchronní práci s daty.

Jak již bylo zmíněno dříve v kapitole věnující se debugování, tak základní debugovací nástroje neumožňují prohlížet data uložené v AsyncStorage. K tomuto lze využít aplikaci

Reactotron, která kromě jiných nástrojů umožňuje také v grafickém rozhraní prohlížet obsah AsyncStorage. `AsyncStorage` je užitečný pro méně robustní aplikace, protože úložiště nelze strukturovat. To lze vyřešit pomocí různých modulů, jako například react-native-db-models nebo react-native-store, které staví na `AsyncStorage` s tím, že zprostředkovávají klasický databázový model. Pro skutečné databázové řešení lze využít knihovny Realm, služby Firebase nebo PouchDB.

```
async function _ulozitData(text) {
      try {
          await AsyncStorage.setItem('klic',    JSON.stringify(text));
      } catch (error) {
          console.log(error);
}}
async function _nacistData(klic) {
      try {
          const data = await AsyncStorage.getItem(klic);
          data = JSON.parse(data);
          if (data !== null) {
               return data;
          }} catch (error) {
               console.log(error);
}}
```

Ukázky kódu 4.29: Obsluha modulu AsyncStorage

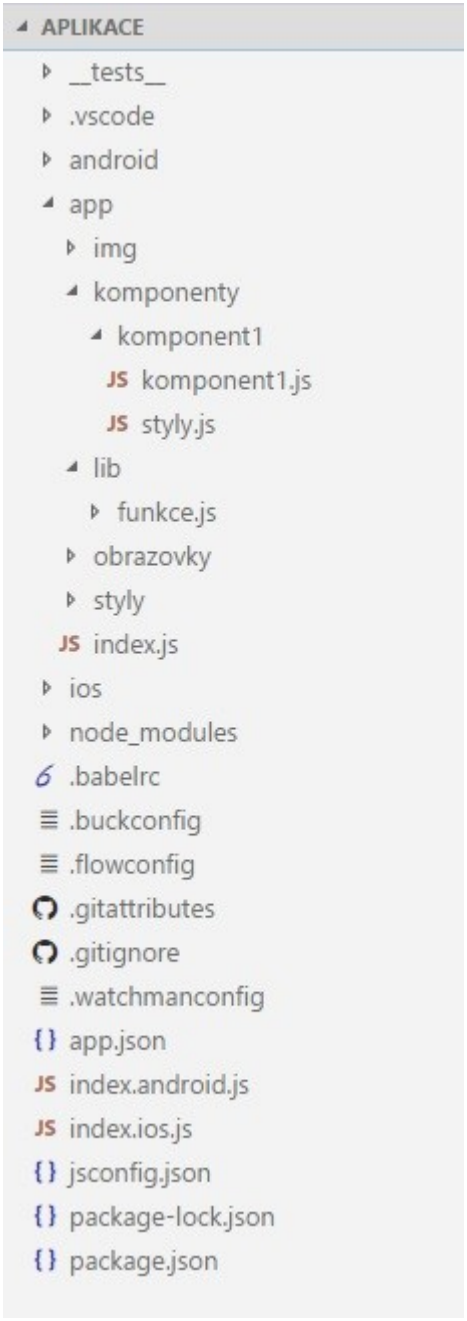
V tomto upraveném příkladu z předchozí kapitoly o komponentu `TextInput` je stav text uložen a načítán, takže jeho hodnota přetrvá i restart aplikace. Funkce na ukládání a načítání dat jsou asynchronní což znamená, že ostatní kód nečeká, než se funkce dokončí a vrátí data. Z tohoto důvodu není použita ve funkci `componentWillMount`, protože tato funkce je provedena před prvním vykreslením komponentu a není asynchronní. Funkci `componentWillMount` je možné deklarovat jako asynchronní, ale tím by bylo zdrženo první vykreslení komponentu, což obecně není žádoucí. Asynchronní funkce `componentWillMount` může být řešena přidáním stavu načítání, který je nastaven při načítání dat na `true` a po načtení na `false`. Změnou stavu se komponent znovu vykreslí a data, na která se čekalo, budou aktualizována.

4.11 Adresářová struktura

React Native nevyžaduje striktně dodrženou adresářovou strukturu. Soubory `index.android.js` a `index.ios.js` slouží jako vstupní body pro aplikaci a je v nich aplikace registrována pomocí `AppRegistry`. Pokud nejsou využívány komponenty podporované pouze jednou platformou, tak je doporučeno vytvořit jeden soubor, například `index.js`, který bude následně importován do souborů `index.android.js` a `index.ios.js` pro jednodušší správu kódu. Složky android a ios obsahují nativní kód.

Tyto složky by obvykle byly otevírány pomocí Android Studia nebo Xcode IDE. Cílem adresářové struktury je maximalizovat přehlednost, znovupoužitelnost a sdílení kódu mezi platformami. U návrhu komponentů je důležité upřednostňovat řízení pomocí vlastností a ne pomocí stavů. Stavy by měly být řízeny v hlavním spouštěcím souboru.

Komponenty a obrazovky by měly mít vlastní složky. U stylů záleží na velikosti aplikace a složitosti komponentů. Konfigurace jako globální styly a barvy by měly být importovány z jednoho souboru, aby bylo jednoduché je změnit. Styly lokální pro komponent je doporučeno podle složitosti buď s umístit do stejného souboru, kde je deklarován komponent, nebo skladovat v samostatném souboru u komponentu. To ale může zvýšit úroveň adresářů. Často používané funkce mají také dedikovaný soubor nebo složku pro lepší znovupoužitelnost a přehlednost.



Obrázek 4.9: Ukázka doporučené adresářové struktury

5 Debugování aplikace

Debugování aplikací v React Native by se dalo rozdělit do dvou kategorií. První kategorií je debugování aplikace na zařízení, kde je aplikace nainstalována. Tento typ debugování je charakteristický okamžitou odezvou a není nutné ho jakkoliv vyvolat. Druhou kategorií je debugování pomocí webových nástrojů, nebo nástrojů které ty webové připomínají.

5.1 Chybová a varovná okna

Informační okna zobrazující se na zařízení, ze kterého je aplikace spouštěna slouží k okamžité zpětné vazbě při chybě v aplikaci. Jedná se o dvě okna, kde prvním oknem je okno varovné a druhým okno chybové. Obě okna jsou zobrazovány pouze ve vývojových verzích aplikace a v živé verzi jsou vypnuty.

5.1.1 Yellow Box

Varovná okna se vyznačují žlutou barvou a bývají zobrazována po chybách v syntaxi a obecně chybách, které nezpůsobují pád aplikace. Tyto okna lze skrýt ve spuštěné aplikaci nebo pomocí příkazu `console.disableYellowBox = true;`. Specifické varování lze vypnout příkazem `Console.ignoredYellowBox = ['Warning: ...'];` nebo je možné okno vyvolat pomocí příkazu `console.warn()`.

5.1.2 Red Box

Chybové okno se vyznačuje červenou barvou a vypisuje příčinu chyby způsobující pád aplikace. Okno obsahuje odkaz na chybový soubor, řádek a znak. Po kliknutí na adresu souboru je tento soubor otevřen v přednastaveném textovém editoru. Chybové okno nejde na rozdíl od varovného okna zavřít ani přeskočit. Lze ho však uměle vyvolat pomocí příkazu `console.error()`.

5.2 Konzole

Konzole slouží jako alternativní výstup pro veškeré grafické varování a chyby. Obsahuje detailnější popis chyb a varování, popisuje proces spouštění a překladu aplikace a uživatelské výstupy z příkazu `console.log()`.

5.3 Inspector

Inspector je debugovací nástroj podobný vývojářským nástrojům z webových prohlížečů. Umožňuje prohlížet seznam všech komponentů na aktuální obrazovce a zobrazuje jejich parametry jako je velikost, odsazení a styly. Další záložky obsahují informace o výkonu, síťové komunikaci a dotykové interaktivní oblasti. Tento inspector se nachází v menu vývojářských nástrojů, které lze otevřít pomocí zkratky ctrl + m.

5.4 Chrome

Dalším způsobem jak debugovat React Native je využití vývojářských nástrojů v prohlížeči Chrome. Tato funkce je dostupná z vývojářského menu pod tlačítkem Debug JS remotely, které následně otevře záložku na adrese http://localhost:8081/debugger-ui. Zde jsou k dispozici dva důležité nástroje. Jedním je konzole, která kromě diagnostického nástroje slouží k vypisování proměnných, stavů a textu příkazem `console.log()`. Druhým důležitým nástrojem je záložka sources, která zobrazuje zachycené výjimky a k nim patřičný soubor a řádek na kterém se výjimka vyskytla. Je doporučeno využít funkce „Pause On Caught Exceptions“, která pozastaví aplikaci při zachycené výjimce. Nejdůležitější funkcí těchto nástrojů je možnost využití takzvaných breakpointů. Breakpoint lze umístit do JavaScriptového kódu pomocí příkazu `debugger;`. Tento příkaz zastaví výkon kódu v požadovaném místě, kde je dále možné prohlížet hodnoty proměnných.

5.5 Reload

Pro zobrazení změn napsaných v kódu aplikace není třeba aplikaci znovu kompilovat. Stačí použít funkci Reload, která je dostupná z vývojářského menu, nebo zmáčknutím dvakrát písmene R v Android emulátoru. Tím je aplikace téměř okamžitě znovu načtena bez potřeby opětovné kompilace. Nadstavbami této funkce je Live Reload, který automaticky znovu načte aplikaci při změnách v kódu aplikace a Hot Reload, který stejně jako Live Reload načítá aplikaci automaticky s tím, že zachovává aktuální stav, ve kterém se momentálně aplikace nachází.

5.6 Reactotron

Jelikož žádný z předchozích debugovacích nástrojů neposkytuje informace o interním úložišti, tak je nutné obrátit se na nástroje tvořené komunitou. Jedním z takovýchto nástrojů je Reactotron. Kromě již zmíněné funkce zobrazení interního úložiště aplikace umí tento program zobrazovat například stavy aplikace nebo API požadavky a odpovědi.

6 Portování

O portování aplikace na jiné operační systémy je postaráno při kompilaci zdrojového kódu. Ten samý kód je použit na všechny platformy a záleží jen na tom, na které zařízení je momentálně aplikace instalována, nebo pro kterou platformu momentálně generujeme instalační balíček. Dále je možné mezi platformami rozlišovat použitím jiného kódu v souborech index.ios.js a index.android.js. Pokud se kód v těchto souborech liší, například použitím jiných modulů, tak budou aplikace na obou operačních systémech odlišné. Posledním způsobem, jak narušit zabudovanou multiplatformnost je implementace nativního kódu.

7 OpenCV

„OpenCV (Open Source Computer Vision Library) je open-source softwarová knihovna pro počítačové vidění a strojové učení. OpenCV bylo vytvořeno, aby poskytovalo společnou infrastrukturu pro aplikace využívající počítačové vidění a aby akcelerovalo strojové vnímání v komerčních produktech. Tím že jde o BSD[11] licencovaný produkt, tak OpenCV umožňuje podnikatelům jednoduše využít a modifikovat kód.“[13]

„Tato knihovna obsahuje více než 2500 optimalizovaných algoritmů mezi které patří například detekce obličejů, identifikace objektů, sledování pohybujících se objektů nebo hledání podobných obrázků z databáze. Tím že je napsána v optimalizovaném C/C++, tak může využít výhod více jádrového processingu a za pomocí OpenCL lze využít také hardwarové akcelerace. OpenCV má uživatelskou komunitu čítající více než 47 000 lidí a je využita například v interaktivním umění, inspekci dolů, sešíváním map na webu nebo pokročilé robotice. Mezi společnosti, které využívají OpenCV patří například Google, Yahoo, Microsoft, Intel nebo IBM.“[14] Tato práce je zaměřená konkrétně na využití kaskádového klasifikátoru k detekci včelí matky na plástvi. Tento vytrénovaný algoritmus bude následně importován do mobilní aplikace za pomocí modulu Tracking.js.

7.1 Kaskádový klasifikátor

„Kaskádový klasifikátor je konkrétní příklad soustředěného učení založený na zřetězení několika klasifikátorů. Využívá všech informací sesbíraných z výstupu daného klasifikátoru jako dodatečné informace pro další klasifikátor v kaskádě.“[15] *„Práce s posílenou kaskádou slabých klasifikátorů zahrnuje dvě hlavní fáze: tréninkovou fázi a fázi detekční. Detekční fáze je provedena pomocí modelů založených na Haar nebo LBP modelech.“*[16] *„Detekce objektů pomocí Haar na rysech založené kaskádě klasifikátorů je efektivní metoda pro detekci objektů navržena Paul Viola a Michael Jones v jejich práci, "Rapid Object Detection using a Boosted Cascade of Simple Features“ v roce 2001. Jde o přístup založený na strojovém učení, kde je funkce kaskády trénována ze spousty pozitivních a negativních vzorků.“*[17] *„Pozitivní vzorky jsou obrázky obsahující objekty, které mají být detekovány a negativní vzorky jsou obrázky vše co nemá být detekováno. Sada negativních vzorků musí být připravena manuálně, kdežto sada pozitivních vzorků je vytvořena pomocí programu*

[11] BSD licence – licence pro svobodný software

opencv_createsamples.“[16] Tento program je užitečný v případě, že detekovaný objekt je, nebo má podobné vlastnosti jako například logo firmy neboli je neměnný. Program opencv_createsamples pomůže vytvořit sadu obrázků kde je vzorový objekt rotován, kombinován s pozadími a je různě manipulováno s jeho světlostí. Tímto způsobem lze dojít k sadě pozitivních obrázků která vyhovuje požadavkům tréninku. Jedná-li se o objekt který může nabývat několika až mnoha různých podob, jako je například jakýkoliv automobil, tak je nutné připravit sadu pozitivních snímků ručně, ideálně z fotografií z reálného prostředí. Program opencv_createsamples v tomto případě poslouží k generování souboru ve formátu .vec, který obsahuje všechny pozitivní snímky v černobílé podobě.

8 Praktická část

Aplikace tvořená v praktické části práce má za úkol evidovat zásahy do včelích úlů, které jsou identifikovány buď pomocí QR kódů, nebo pomocí NFC. Tato evidence má formu seznamu, do kterého jsou přidávány položky pod identifikací generovanou daty z QR kódu nebo NFC. Položka v seznamu má parametry odpovídající požadavkům včelaře. Tyto parametry mají různé datové typy a názvy a je možno je upravit v nastavení aplikace. Obsah těchto parametrů lze u jednotlivých položek upravit po klepnutí na položku v seznamu. Další funkcí aplikace je detekce včelí matky pomocí kamery buď během doby, kdy je spuštěn fotoaparát, nebo z vyfoceného snímku. Tato funkce by měla být jednoduše dostupná ze všech obrazovek aplikace.

8.1 Skenování QR kódů

Detekce QR kódu je v aplikaci zajištěna pomocí modulu react-native-camera. Tento modul obsahuje hotový komponent, který obsahuje připravenou vlastnost `onBarCodeRead(Obj)`. Tato vlastnost je spuštěna pokud se v záběru kamery nachází jakýkoliv čárový nebo jiný podobný kód. Jednou z vlastností vráceného objektu je vlastnost `bounds`. Tato vlastnost obsahuje souřadnice detekovaného čárového kódu. Na těchto souřadnicích může být následně vykreslen obdélník zvýrazňující tuto oblast. Další vlastností vráceného objektu je vlastnost `data`. Tato vlastnost obsahuje text, který byl z čárového kódu přečten. V aplikaci, kterou se tato práce zabývá jsou tato data odeslána na obrazovku s úly a na jehož základě je následně otevřena již existující, nebo vytvořena nová položka. Výsledná implementace kamery je znázorněna v následujícím kódu.

```
<Camera
      style={{ flex: 1 }}
      ref={(cam) => {
          this.camera = cam;
      }}
      aspect={Camera.constants.Aspect.fill}
      onBarCodeRead={(barcodeObj) => {
          if (barcodeObj.data) {
               let parsed = JSON.parse(barcodeObj.bounds)
               this.setState({ bounds: parsed, opacity: 1,        data:
               barcodeObj.data });
          }
          else {
               this.setState({ opacity: 0 });
          }
      }
}>
      <View style={{
          position: "absolute",
          borderColor: 'red',
          borderWidth: 3,
          opacity: this.state.opacity,
          height: (this.state.bounds[0][1] -    this.state.bounds[1][1]) /
          PixelRatio.get(),
          width: (this.state.bounds[2][0] -     this.state.bounds[1][0])
          / PixelRatio.get(),
          top: (this.state.bounds[1][1] / PixelRatio.get()) -50,
          left: this.state.bounds[0][0] / PixelRatio.get(),
      }}>
      </View>
</Camera>
```

Ukázky kódu 8.1: Skenování čárových kódů

Výpočet souřadnic, na které má být obdélník vykreslen je důležitá, protože souřadnice ve vlastnosti `bounds` neodpovídají objektu na kameře. Modul `PixelRatio` je použit, aby bylo zaručeno, že souřadnice budou odpovídat na každém zařízení.

8.2 Čtení NFC štítků

Implementace čtečky NFC štítků, které stejně jako QR kód slouží v aplikaci k identifikaci úlu je zajištěna modulem react-native-nfc-manager. Tento modul nabízí multiplatformní podporu pro většinu potřebných funkcí NFC jako je například čtení, zápis a zjištění stavu NFC. Konkrétně nás v aplikaci zajímají funkce `isSupported()`, `isEnabled()`, `registerTagEvent(function)` a `unregisterTagEvent()`. Všechny tyto funkce lze využít po importování NFC manažeru z nainstalovaného modulu. Funkce `isSupported()` a `isEnabled()` slouží k ověření, zda je NFC přítomné na zařízení a jestli běží. Je na vývojáři, jak s těmito informacemi naloží. V aplikaci spojené s touto prací jsou tyto informace zobrazeny a je úkolem uživatele, aby funkci NFC zapnul.

Komplexnějším řešením by mohlo být uživatele přesměrovat do nastavení NFC pomocí funkce `goToNfcSetting()`, ale tato funkce je dostupná pouze pro zařízení s Android OS. Nejdůležitější funkcí je funkce `registerTagEvent(function)`, která spouští proces hledání NFC štítku. Po objevení štítku je zavolána funkce v argumentu, kde by standardně měl být nastaven stav obsahující data ze štítku. Data jsou v aplikaci využity, stejně jako data z QR kódu, jako název úlu, který je odeslán na obrazovku s úly a na jehož základě je následně otevřena již existující, nebo vytvořena nová položka.

8.3 Databáze

Databáze v zařízení slouží k uchovávání záznamů zásahů do včelích úlů. Tato databáze je zajištěna kombinací dvou databázových systémů – CouchDB na serveru a PouchDB na zařízení. Tento databázový systém je určený pro javascriptové webové a mobilní aplikace. Jeho důležitou vlastností je to, že funguje na principu „Offline First Data Sync“, což znamená že synchronizuje data, která jsou uchovávána v zařízení se serverem, když je to vhodné. Toto je obzvlášť důležité pro aplikaci kterou se tato práce zabývá, neboť se jedná o aplikaci, která bude využívána v řídce obydlených oblastech. Struktura dat v databázi je jednoduchá. Jedná se o řetězce ve formátu JSON.

Řádky, neboli v tomto databázovém systému dokumenty, mají dva povinné atributy. Atribut `_id` je originální identifikátor dokumentu a atribut `_rev` je generovaný klíč, který se mění při změně dokumentu. CouchDB, která se nachází na serveru lze nainstalovat pomocí příkazu `sudo apt-get install couchdb` na operačním systému Linux, nebo z instalátorů na operačních systémech Windows a macOS z adresy

https://couchdb.apache.org/#download.[18] Po úspěšné instalaci lze ověřit běh serveru na adrese localhost:5984. Tato stránka by měla zobrazit základní informace o serveru. Do administrace serveru lze přejít odkazem localhost:5984/_utils/. Jelikož PouchDB a CouchDB spolu komunikují v základu pouze v rámci stejné sítě, tak je nutné CouchDB nakonfigurovat tak, aby přijímala spojení ze všech rozhraní. V administraci serveru je nutné nastavit bind address na 0.0.0.0 a nainstalovat CORS. CORS je webová technologie, která umožňuje webovým stránkám přijímat informace z jiné domény. PouchDB má tuto funkci v základu povolenou, ale u CouchDB je nutné tuto funkci povolit. Stačí spustit příkaz `npm install -g add-cors-to-couchdb` a poté tento modul spustit příkazem `add-cors-to-couchdb`. PouchDB lze nainstalovat do React Native standardně pomocí NPM příkazem `npm install pouchdb`. Po importování modulu PouchDB příkazem `import PouchDB from 'pouchdb-react-native'` je nutné inicializovat lokální databázi příkazem `const localDB = new PouchDB('localDB')`. Poté je možné získat všechny dokumenty následující funkcí.

```
localDB.allDocs({ include_docs: true, limit: null })
      .then(result => {
         const items = result.rows.map(row => row.doc)
         const ds = new ListView.DataSource({ rowHasChanged:     (r1, r2)
         => r1.id !== r2.id })
         this.setState({
         "dataSource": ds.cloneWithRows(items),
         "count": items.length,
         "items": items,
      })
})
```

Ukázky kódu 8.2: Získání dokumentů z databáze

Vložení dokumentu do databáze je možné provést funkcí `localDB.put(doc)`, kde `doc` je řetězec JSON objektu obsahující parametr `_id`. Synchronizace mezi databázemi je možná po inicializaci venkovní databáze příkazem `const remoteDb = new PouchDB('http://127.0.0.1/jmeno', { ajax: { cache: false } })`. Pokud databáze na serveru neexistuje, v tomto případě databáze jmeno, tak je vytvořena. Samotnou synchronizaci lze provést funkcí `PouchDB.sync(localDB, remoteDb, { live: true, retry: true })`.

8.4 Detekce objektů

K využití analýzy obrazu v aplikaci na React Native je nejdříve zapotřebí sehnat vytrénovaný detekční klasifikátor. V době, kdy je tato práce psána je k dispozici jediné řešení jak implementovat „computer vision“ neboli strojové vidění do webového frameworku jako je React Native. Jedná se o kombinaci knihovny OpenCV k vytrénování detekčního klasifikátoru a Node.js nebo Tracking.js pro zapojení tohoto klasifikátoru do aplikace. Obě řešení mají své výhody a nevýhody. Výhodou použití Node.js je, že je možné využít většinu funkcí OpenCV, jako jsou například morfologické operace na obrázcích nebo algoritmy pro detekci geometrických vzorců.

8.4.1 Node.js

Prvním způsobem je serverové řešení, kdy je nutné zajistit Node.js server na který budou odesílány požadavky na detekci a následně bude zpět odeslán výsledek. Konkrétně se v našem případě jedná o obrázek, na kterém bude po detekci nakreslen obrazec zvýrazňující detekovaný objekt. Node.js server může být umístěn jak na síti internet, tak na zařízení. V současné době existuje několik knihoven umožňujících spuštění Node.js serveru na zařízeních s Android OS. Jedná se například o Node.js for Mobile Apps, J2V8 nebo Termux. Většina knihoven umožňuje běžnou instalaci, ale vhodným způsobem by bylo tuto instalaci provést tak, aby o ní uživatel nevěděl a byla součástí instalace aplikace samotné.

8.4.2 Tracking.js

Druhým řešením pro implementaci detekčního klasifikátoru je Tracking.js. Jedná se o webový framework, který není připravený pro React Native takže je nutné vytvořit webovou stránku, která bude pomocí webview otevřena v aplikaci. Tato webová stránka zajišťuje vstupní obrázek, spuštění javascriptového kódu pro detekci objektů a vytvoření výstupního obrázku. Nejjednodušším způsobem pro nahrání vstupního obrázku je využití HTML elementu `<input>` s parametrem `type="file"`.

Bohužel v současné době nativní komponent WebView, který je součástí React Native, nepodporuje tento HTML element na platformě Android OS takže je nutné využít modul react-native-webview-android. Jedinou prací tohoto komponentu bude otevření webové stránky, která může, ale nemusí být přítomna v paměti zařízení. U tohoto komponentu nás zajímají dvě vlastnosti. Vlastnost `url` definující venkovní adresu, která bude v komponentu otevřena a vlastnost `source`, která určuje zdroj souboru. Právě vlastnost `source` nás bude

zajímat, jelikož naším cílem je otevřít webovou stránku uloženou na zařízení. Tato webová stránka se musí nacházet v adresáři `android\app\src\main\assets`, aby jí bylo možné otevřít zkratkovou cestou `file:///android_asset/stranka.html`. Celý komponent by ve výsledku měl mít, kromě obecných náležitostí, následující formu.

```
<WebViewAndroid
      ...
      source={{uri: "file:///android_asset/stranka.html"}}
/>
```

Ukázky kódu 8.3: Načtení lokálně uložené webové stránky

Samotná webová stránka obsahuje kromě vstupu pro obrázek implementaci knihovny Tracking.js. Součástí knihovny jsou klasifikátory pro detekci tváře rozdělené do tří částí – celá tvář, oči a pusa. Hlavními funkcemi knihovny je datový typ `ObjectTracker` a funkce `tracking`. `ObjectTracker` definuje který klasifikátor bude použit a funkce `tracking(element, ObjectTracker)` určuje zdroj obrázku a `ObjectTracker` který bude použit. Po načtení obrázku do stránky je nutné spustit funkci, která inicializuje ObjectTracker následujícím způsobem.

```
var tracker = new tracking.ObjectTracker(['face', 'eye', 'mouth']);
```

Ukázky kódu 8.4: Inicializace trackeru

Klasifikátory `['face', 'eye', 'mouth']` jsou součástí knihovny a pro použití vlastního klasifikátoru, neboli v případě této knihovny trackeru, je nutné tento tracker definovat. Nejjednoduší cestou k tomuto cíli je přepsat jeden z již přítomných trackerů. Je možné vytvořit i úplně nový tracker, ale toto řešení slouží jen ke zlepšení přehlednosti projektu a není potřebné pokud není ve stránce využito méně než tří trackerů. V dokumentaci Tracking.js je tento proces popsán, ale poměrně nekompletně. V každém případě je nutné přeložit klasifikátor vytrénovaný v OpenCV ve formátu XML do interního formátu v JavaScriptu používaném v Tracking.js. Tento proces je zautomatizován díky projektu na adrese https://github.com/cirocosta/gulp-converter-tjs.[19]

Poté co je inicializován ObjectTracker je možné nastavit parametry algoritmu detekce. Tyto parametry vycházejí z frameworku Viola–Jones object detection a určují vlastnosti okna a postupu při detekci. Nastavení těchto parametrů záleží od klasifikátoru. Neexistuje jedno správné nastavení a ke zjištění optimálních hodnot je nutné experimentovat. Konkrétně

v příkladové webové stránce Tracking.js je nastaven parametr `stepSize` pomocí funkce `tracker.setStepSize(1.7)`. Tento parametr určuje velikost posunu detekčního okna při neúspěšné detekci v předchozím kroku. Dále je možné nastavit parametr `scaleFactor` funkcí `setScaleFactor(scaleFactor)` určující násobek velikosti pro zvětšení detekčního okna, `initialScale` určující počáteční velikost detekčního okna nebo `edgesDensity` určující hustotu hran v bloku při které bude rozhodnuto, jestli bude nebo nebude blok přeskočen. Všechny tyto parametry mají i základní hodnoty a nemusejí být měněny. V tomto bodě je vše připraveno ke spuštění detekce. Následující kus kódu spouští detekci s již určeným trackerem a vykresluje zvýrazňující obdélník do elementu který obaluje obrázek `#img`.

```
tracking.track('#img', tracker);
tracker.on('track', function (event) {
      event.data.forEach(function (rect) {
            window.plot(rect.x, rect.y, rect.width, rect.height);
      });
});

window.plot = function (x, y, w, h) {
      var rect = document.createElement('div');
      document.querySelector('.container').appendChild(rect);
      rect.classList.add('rect');
      rect.style.width = w + 'px';
      rect.style.height = h + 'px';
      rect.style.left = (img.offsetLeft + x) + 'px';
      rect.style.top = (img.offsetTop + y) + 'px';
};
```

Ukázky kódu 8.5: Trackování a vykreslování zvýrazňujícího obdélníku

8.5 Trénink klasifikátoru

Trénink klasifikátoru proběhne za pomocí knihovny OpenCV verze 2.4. Číslo verze je u této knihovny důležité, protože se mezi verzemi mění obsažené programy a syntaxe příkazů. Nutnou přípravou kromě stažení knihovny samotné je kompilace zdrojových kódů do spustitelných souborů a také přidání adresáře `opencv\build\x64\vc12\bin` do systémové proměnné PATH pro usnadnění práce.

8.5.1 Příprava

K vytrénování klasifikátoru je třeba vlastnit sadu pozitivních a negativních obrázků. Tyto obrázky mohou být generovány kombinací objektu a pozadí, nebo vyfoceny ručně z reálného prostředí. Kvalita přípravy těchto snímků určuje výslednou kvalitu klasifikátoru a je tedy důležité připravit velké množství, ideálně stovky až tisíce, pozitivních i negativních obrázků. Co se týká vlastností obrázků, tak by měly mít stejnou velikost, může být použito formátu .jpg nebo .png a pozitivní obrázky mohou být barevné, neboť z nich bude generován soubor, který barevné kanály vymaže.

Negativní obrázky musí být černobílé, konkrétně v tom smyslu, že neobsahují barevné kanály. V každém případě bude použit program opencv_createsamples. Tento program připravuje sadu obrázků připravenou k použití při tréninku. Důležitým vstupním parametrem je parametr `-info`. Tento parametr určuje cestu k souboru ve formátu .info, který obsahuje informace o pozitivních obrázcích. Konkrétně jde o cestu k obrázku, počet objektů, které se na něm nachází, souřadnice objektu a velikost. Tento soubor, pojmenovaný například pozitivni.info musí mít formát `/opencv/trenink/pos/pos-1.png 1 0 0 72 100`.

Cesta k souboru by měla být absolutní a soubor by se měl nacházet na stejném disku jako adresář s OpenCV. První číslo, tedy číslo 1, určuje počet objektů nacházejících se na obrázku. Další dvě čísla, tedy 0 a 0, určují souřadnice objektu. V tomto případě se jedná o dvě nuly proto, že obrázek obsahuje pouze tento objekt, zajímá nás tedy celý obrázek. Další dvě čísla určují šířku a výšku celého obrázku. Tento formát obrázku, kde celý obrázek zabírá detekovaný objekt je doporučeným způsobem pro optimální výsledky tréninku. Dále by velikost obou stran obrázku neměla překročit 100 pixelů, protože větší obrázky znatelně prodlužují dobu tréninku.

Obrázek 8.1: Ukázka pozitivního obrázku

Dalším důvodem, proč dodržovat tento formát pozitivních obrázku je ten, že generování souboru .info zabere mnohem více času, pokud se na něm nachází více než jeden objekt, nebo

není ořízнut na přesnou velikost. Ve zkratce je při generování .info souboru vhodné dodržovat stejný formát pro všechny obrázky. K získání všech obrázků v adresáři je možné využít PowerShell příkazu `Get-ChildItem ./pos -Name > pozitivni.txt`. Následně je možné přidat ostatní parametry pomocí funkce nahradit v libovolném textovém editoru. V generovaném souboru je důležité změnit formátování na UTF-8, jinak ho program opencv_createsamples nepřijme. Druhým důležitým parametrem programu opencv_createsamples je parametr `-vec`. Tento parametr určuje cestu k souboru, který bude vygenerován. Tento soubor obsahuje informace o pozitivních obrázcích a je generován přímo ze souboru v parametru `-info`. Dalším parametrem je parametr `-num`. Tento parametr určuje počet položek v souboru, který generujeme. Toto číslo by mělo přímo odpovídat počtu pozitivních snímků, ale může být nižší. Poslední parametry, které jsou potřebné jsou parametry `-w` a `-h`, které určují velikosti obrázků. Výsledný příkaz, kterým generujeme soubor .vec vypadá takto.

```
opencv_createsamples -info pozitivni.info -num 1000 -w 72 -h 100 -vec
pozitivni.vec
```

Po spuštění v příkazovém řádku nebo PowerShell by měl být úspěšně vygenerován soubor pozitivni.vec. Tento soubor bude dále využit při tréninku. Při generování souboru je možné využít dalších parametrů, které ovlivňují způsob jakým se generují snímky v souboru. Základním nastavením je pouze přeměna na černobílé snímky, ale je možné využít rotace, kombinování pozitivních snímků s pozadím, nebo náhodné invertování barev. Rotace je nastavena parametry `-maxxangle, -maxyangle, -maxzangle` s tím, že hodnota musí být v radiánech. Parametr `-bg` určuje cestu k souboru s cestami k obrázkům, které budou použity jako pozadí při generování. Soubor obsahuje pouze cesty k obrázkům a žádné další parametry. Tyto alternativní parametry slouží k rozšíření pestrosti mezi pozitivními snímky. Při tomto generování je však použito stejné množství pozitivních snímků a nemusí tím pádem nutně zlepšovat výsledky tréninku. Tímto je připraven soubor .vec, který je potřebný jako vstup do programu opencv_traincascade. Soubor .vec je možné prohlédnout následujícím příkazem.

```
opencv_createsamples -w 72 -h 100 -vec pozitivni.vec
```

Tím se lze ujistit, že jsou obrázky vygenerovány správně. Dalším nutným souborem je .txt soubor obsahující cesty k negativním obrázkům v následujícím formátu.

```
e:/opencv/trenink/neg/neg-1.jpg
```

Opět obsahuje absolutní cesty, ale program opencv_traincascade je náročnější na jejich kompletnost a je tedy vhodné přidat i adresu disku, na kterém se soubory nachází. Dále je důležité soubor převést do formátu UTF-8. Tímto by měl být soubor připraven tak, aby nedošlo ke zbytečným chybám.

8.5.2 Trénink

Program opencv_traincascade má několik různých povinných parametrů. Parametr `-vec` určuje cestu k souboru .vec, který jsme generovali. Parametr `-bg` určuje cestu k souboru obsahující cesty k negativním snímkům. Parametr `-data` určuje cestu k adresáři, do kterého budou zapisovány výsledky tréninku. V tomto adresáři lze následně najít výsledek tréninku a výsledky jednotlivých fází tréninku. Fáze tréninku jsou uchovávány z toho důvodu, že trénink může být časově náročný a je možné v tréninku pokračovat, pokud byl přerušen. Pokud má trénink začínat od začátku, tak musí být adresář prázdný. Parametry `-numPos` a `-numNeg` určují množství obrázků použitých při tréninku. Je doporučeno použít tři čtvrtiny všech pozitivních snímků pro optimální výsledek.

Parametr `-numStages` určuje počet fází tréninku. Pro vyzkoušení parametrů je možné využít například dvou fází. Výsledný klasifikátor ale může být nespolehlivý. Obecně je doporučeno vyzkoušet trénink na třech fázích, poté přejít na 5 fází. Po úspěšném tréninku na pěti fázích je možné přejít na 8 fází. Pokud nedošlo ke změně, tak tréninková sada dosáhla svého maximálního potenciálu. Další zvyšování počtu fází může vést k přetrénování. Kvalita klasifikátoru je určena údajem acceptanceRatio. Toto číslo by mělo být ideálně kolem 0.0000412. Pokud dosahuje hodnot podobných 4.8789e-05, tak je klasifkátor přetrénovaný.

Dále je nutné nastavit parametry `-w` a `-h`, které určují velikosti vstupních obrázků. Posledním parametrem, který je volitelný je parametr `-featureType`. Tento parametr určuje metodu, která bude při tréninku použita. K dispozici jsou dvě metody. Metoda LBP je rychlejší a metoda HAAR, což je základní nastavení programu, je pomalejší ale spolehlivější. Výsledný příkaz pro spuštění tréninku vypadá následně.

```
Opencv_traincascade -data data -vec pozitivni.vec -bg negativni.txt -numPos
750 -numNeg 100 -numStages 3 -w 72 -h 100 -featureType LBP
```

Program obsahuje další parametry umožňující měnit parametry tréninku jako je například `-minHitRate`, `-maxFalseAlarmRate`, `-weightTrimRate`, `-maxDepth` a `-`

`maxWeakCount`. Tyto parametry záleží od vlastností každé tréninkové sady a pro optimální výsledek je potřeba s těmito nastaveními experimentovat. Výsledek tréninku je dále možné vizualizovat pomocí programu `opencv_visualisation`. Tento program vyžaduje dva vstupní parametry. Parametr `--image` vyžaduje cestu k obrázku, na kterém bude vizualizace provedena, a Parametr `--model` vyžadující cestu k vytrénovanému klasifikátoru, který se nachází ve složce s daty z předchozího tréninku s názvem model.xml.

9 Závěr

Cílem práce bylo zjistit jakým způsobem je možné implementovat analýzu obrazu v multiplatformní aplikaci postavené na frameworku React Native. Nejedná se o funkci, která je součástí frameworku, nebo je v současné době implementovatelná pomocí modulu tvořeném veřejností. Jediným multiplatformním způsobem je využití knihovny OpenCV ve spojení s webovým řešením za pomocí knihovny Tracking.js. Tento přístup má několik nevýhod. Jde o hybridní řešení za pomoci komponentu webview, který otevírá webovou stránku uloženou v zařízení. I přesto, že není nutné tento soubor nahrávat ze sítě internet, tak je rychlost detekce silně závislá na výkonu zařízení. Je však možné implementovat jakýkoliv klasifikátor vytrénovaný pomocí OpenCV.

Konkrétní klasifikátor, který měl být v praktické části použit, je klasifikátor určený pro detekci včelí matky na úlu. Již před zahájením procesu tréninku bylo zřejmé, že vytrénovaný klasifikátor bude značně nespolehlivý. Jelikož HAAR a LBP model, který je použit v knihovně OpenCV vyhledává rysy na základě světlosti, tak nebylo možné vytrénovat klasifkátor, který by dokázal rozeznat jakoukoliv včelu od pozadí, na kterém se nachází. Pokud by u včel bylo zaručeno pozadí, které je kontrastní vůči samotné včele, tak by byla detekce možná. Nejednalo by se však o prostředí, které je pro včely přirozené.

Seznam použité literatury a zdrojů

[1] SHAUN, Eduo. *What is React Native? How does it work?* [online]. 2017 [cit. 2017-07-15]. Dostupné z: https://medium.com/@eduoshaun/what-is-react-native-how-does-it-work-e745ca1fae7b

[2] *JavaScript HTML DOM* [online]. 2017 [cit. 2017-07-18]. Dostupné z: https://w3schools.com/js/js_htmldom.asp

[3] *React: The Virtual DOM* [online]. 2017 [cit. 2017-07-19]. Dostupné z: https://codecademy.com/articles/react-virtual-dom

[4] NAGPAL, Narendra. *React Native vs Hybrid for mobile apps: Which Is Really Better?* [online]. 2017 [cit. 2017-07-19]. Dostupné z: https://ipragmatech.com/react-native-vs-hybrid-mobile-app

[5] KONICEK, Martin. *React Native: a year in review* [online]. 2016 [cit. 2017-07-23]. Dostupné z: https://code.facebook.com/posts/597378980427792/react-native-a-year-in-review/

[6] CALDERAIO, John A. *Comparing the Performance between Native iOS (Swift) and React-Native* [online]. 2017 [cit. 2017-07-26]. Dostupné z: https://medium.com/the-react-native-log/comparing-the-performance-between-native-ios-swift-and-react-native-7b5490d363e2

[7] CHRZANOWSKA, Natalia. *We've rewritten a Swift app in React Native. Was it worth it? - Case study* [online]. 2017 [cit. 2017-07-26]. Dostupné z: https://netguru.co/blog/react-native-vs-swift-performance-development-comparison/

[8] *React Native: Getting Started* [online]. 2017 [cit. 2017-08-1]. Dostupné z: https://facebook.github.io/react-native/docs/getting-started.html

[9] *JavaScript ES6 – Modules* [online]. 2017 [cit. 2017-08-4]. Dostupné z: http://exploringjs.com/es6/ch_modules.html

[10] SHKUT, Konstantin. *State in React Native Components* [online]. 2016 [cit. 2017-08-5]. Dostupné z: http://rationalappdev.com/state-in-react-native-components

[11] *React.Component* [online]. 2017 [cit. 2017-08-5]. Dostupné z: https://facebook.github.io/react/docs/react-component.html

[12] *React Native – AppRegistry* [online]. 2017 [cit. 2017-08-6]. Dostupné z: https://facebook.github.io/react-native/docs/appregistry.html

[13] *OpenCV – About* [online]. 2017 [cit. 2017-08-14]. Dostupné z: http://opencv.org/about.html

[14] *OpenCV* [online]. 2017 [cit. 2017-08-14]. Dostupné z: http://opencv.org/

[15] WIKIPEDIA, *Cascading classifiers* [online]. 2017 [cit. 2017-08-15]. Dostupné z: https://en.wikipedia.org/wiki/Cascading_classifiers

[16] PUTTEMANS, Steven. *OpenCV - Cascade Classifier Training* [online]. 2017 [cit. 2017-08-17]. Dostupné z: http://docs.opencv.org/trunk/dc/d88/tutorial_traincascade.html

[17] OpenCV - *Face Detection using Haar Cascades* [online]. 2018 [cit. 2018-06-24]. Dostupné z: http://docs.opencv.org/trunk/d7/d8b/tutorial_py_face_detection.html

[18] *CouchDB download* [online]. 2018 [cit. 2018-06-25]. Dostupné z: https://couchdb.apache.org/#download

[19] COSTA, Ciro S. *gulp-converter-tjs* [online]. 2018 [cit. 2018-06-28]. Dostupné z: https://github.com/cirocosta/gulp-converter-tjs

Seznam obrázků

Obrázek 2.1: Zájem v průběhu času mezi React Native a NativeScript 14
Obrázek 3.1: Úspěšná instalace programu Chocolatey 18
Obrázek 3.2: Správné nastavení instalace Android Studia 19
Obrázek 3.3: Instalace povinných balíčků Android SDK 20
Obrázek 3.4: Instalace povinných nástrojů Android SDK 20
Obrázek 3.5: Přidání uživatelské proměnné ANDROID_HOME 21
Obrázek 3.6: Instalace emulátoru 22
Obrázek 3.7: Spouštění emulátoru a tvorba dávkového souboru. 23
Obrázek 4.1: Reálné zobrazení vlastního komponentu 30
Obrázek 4.2: Reálné zobrazení změny stavu komponentu 35
Obrázek 4.3: Zobrazení velikostí v rozložení flex 38
Obrázek 4.4: Ukázka využití parametru flexDirection 39
Obrázek 4.5: Ukázka využití parametru justifyContent 40
Obrázek 4.6: Ukázka komponentu StackNavigator 44
Obrázek 4.7: Ukázka komponentu TabNavigator 45
Obrázek 4.8: Ukázka složených navigátorů 46
Obrázek 4.9: Ukázka doporučené adresářové struktury 51
Obrázek 8.1: Ukázka pozitivního obrázku 65

Seznam ukázek kódu

Ukázky kódu 2.1: HTML – Porovnání s React Native 16
Ukázky kódu 2.2: React Native – Porovnání s HTML 16
Ukázky kódu 4.1: Porovnání šipkového a normálního zápisu funkce 25
Ukázky kódu 4.2: Využití argumentů v šipkových funkcích 26
Ukázky kódu 4.3: Exportování a importování modulů 27
Ukázky kódu 4.4: Jmenné importování modulu 27
Ukázky kódu 4.5: Default export 27
Ukázky kódu 4.6: Porovnání mezi deklarací komponentu v JSX a JavaScriptu 27
Ukázky kódu 4.7: Párový a nepárový komponent 28
Ukázky kódu 4.8: Importování React Native komponentu 28
Ukázky kódu 4.9: Vlastnosti komponentů 29
Ukázky kódu 4.10: Tvorba vlastního komponentu 29
Ukázky kódu 4.11: Nastavení základních vlastností komponentu 30
Ukázky kódu 4.12: Tvorba vlastního párového komponentu 31
Ukázky kódu 4.13: Inline stylování komponentu 31
Ukázky kódu 4.14: Stylování pomocí proměnné 32
Ukázky kódu 4.15: Kombinování stylů 33
Ukázky kódu 4.16: Inicializace stavu 34
Ukázky kódu 4.17: Využití stavů ke změně komponentu 34
Ukázky kódu 4.18: Využití vlastnosti prevState 35
Ukázky kódu 4.19: Využití komponentu TextInput 36
Ukázky kódu 4.20: Implementace seznamu pomocí komponentu FlatList 37
Ukázky kódu 4.21: Velikosti v rozložení flex 38
Ukázky kódu 4.22: Využití parametru flexDirection 38
Ukázky kódu 4.23: Základní struktura aplikace 41
Ukázky kódu 4.24: Deklarace obrazovky komponentu StackNavigator 42
Ukázky kódu 4.25: Deklarace komponentu StackNavigator 43
Ukázky kódu 4.26: StackNavigator se dvěma obrazovkami 44
Ukázky kódu 4.27: Skládání navigátorů 46
Ukázky kódu 4.28: Odesílání parametrů na obrazovku Profil 47
Ukázky kódu 4.29: Obsluha modulu AsyncStorage 49

Ukázky kódu 8.1: Skenování čárových kódů 59
Ukázky kódu 8.2: Získání dokumentů z databáze 61
Ukázky kódu 8.3: Načtení lokálně uložené webové stránky 63
Ukázky kódu 8.4: Inicializace trackeru 63
Ukázky kódu 8.5: Trackování a vykreslování zvýrazňujícího obdélníku 64