TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Arvutiteaduse instituut

ITI40LT
Mikk Kärner 135051IAPB

REACT NATIVE RAAMISTIKU KASUTAMINE MOBIILIRAKENDUSE UNISPOTTER NÄITEL

Bakalaurusetöö

Juhendaja: Roger Kerse
Tehnikateaduse magister
Lektor

Tallinn 2016

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Mikk Kärner

19.05.2016

Annotatsioon

Käesoleva lõputöö eesmärgiks on React Native raamistikuga tutvumine ning selle rakendamine. Rakendamise osa seisneb mobiilirakenduse UNIspotter arendamises antud raamistikus iOS platvormile, koos testimisega Androidi platvormil. Töö käigus kirjeldab autor UNIspotteri rakenduse algseisu, React Native raamistiku tööpõhimõtteid, kasutatud tehnoloogiaid koos kommentaaridega ning annab ülevaate arendusprotsessist.

Töö tulemusel valmib nimetatud rakenduse põhifunktsionaalsus React Native raamistikus iOS platvormile. Autor testib kirjutatud koodibaasi samuti Androidi platvormil, millest järeldub, et suurt osa koodist on võimalik taaskasutada. Loodud rakendus osutub jõudluselt ning kasutusmugavuselt paremaks, kui sama rakenduse esialgne Ionic raamistikus loodud versioon, mille tõttu on autori hinnangul otstarbekas jätkata nimetatud rakenduse arendamist kasutades React Native raamistikku.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 35 leheküljel, 6 peatükki, 13 joonist, 0 tabelit.

Abstract

The usage of React Native's framework in the example of UNIspotter

The purpose of this thesis is to learn the principles of React Native's framework and to apply that knowledge at building UNIspotter's mobile application. The main programming is done for the iOS platform, but some testing is also done on Android to ensure that the written code is reusable. Reusability and the same developing principles on both platforms are the main purposes of using React Native's framework.

Prior to this thesis, the UNIspotter application is implemented using the Ionic framework, which enables developers to build applications simultaneously on both platforms using only technologies supported by the WebView component – HTML, CSS, JavaScript. Due to these restrictions, the UNIspotter application has been experiencing some performance issues. However, React Native does not have such restrictions, but also allows the programmer to build applications on both platforms efficiently, while using a lot of the same codebase.

In this thesis the author describes the prior state of the UNIspotter application, the principles of React Native's framework and used technologies with comments, following a summary of the development process. This work is mainly centered on the technologies used by the author. Though this is specific to the development of one application, it represents the main state of mobile development using React Native.

The application developed during this thesis shows that it is possible to create an application that uses React Native's framework and uses mainly the same base code on both iOS and Android. Also the application performs better than the prior Ionic version of the application, which is an indicator that it is reasonable to continue developing using React Native.

The thesis is in Estonian and contains 35 pages of text, 6 chapters, 13 figures, 0 tables.

Lühendite ja mõistete sõnastik

AJAX	Asynchronous JavaScript and XML, asünkroonse serveripäringu tehnika
API	Application programming interface, rakendusliides ehk programmiliides
Callback funktsioon	Asünkroonse toimingu lõppedes väljakutsutav funktsioon
CSS	Cascading Style Sheets, kaskaadlaadistik
Emulaator	Seadme virtuaalne koopia, rakenduse testimiseks
Flexbox	CSS3 kujunduse mudel
HTML	HyperText Markup Language, hüperteksti märgistuskeel
IDE	Integrated development environment, integreeritud arenduskeskkond
Live Reload	Koodimuudatustele reageeriv kohene uuesti laadimine
MongoDB	Dokumendipõhine andmebaasisüsteem
MVC	Model–view–controller, mudel–vaade–kontroller
NPM	Node Package Manager, Node.js'i pakihaldussüsteem
ODM	Object-Document Mapper, objekt-dokument tõlgendaja
Plugin	Pistikprogramm, lisavõimalusi pakkuv tarkvaramoodul
Põlis	*Native*, platvormi-spetsiifiline
SDK	Software development kit, tarkvaraarenduskomplekt
Silumine	*Debugging*, rakenduses vigade otsimine ja parandamine
Simulaator	Seadme imiteerija, rakenduse testimiseks
SPM	Swift Package Manager, Swifti pakihaldussüsteem
Tag	XML element
Täisdupleks	Süsteem mis võimaldab üheaegset suhtlust mõlemas suunas
UI lõim	Rakenduse kasutajaliidest haldav lõim, ka *Main* lõim
WebSocket	TCP põhine täisdupleks suhtlusprotokoll
WebView	Vaade, mis võimaldab kuvada HTML lehte
Wrapper	Mähis, alamrutiini väljakutsuv rutiin

Sisukord

Jooniste loetelu

1 Sissejuhatus

Käesoleva lõputöö ülesanne seisneb React Native raamistiku tundmaõppimises ning selle rakendamises. Nimetatud raamistiku näol on tegemist vahendiga mis peaks lihtsustama kaasaegset mobiilirakenduste loomise protsessi, eemaldades vajaduse kirjutada ulatuslikku platvormi-spetsiifilist koodi. Antud lõputöö kirjutamise hetkel toetab React Native kahte mobiiliplatvormi – iOS ja Android – pakkudes võimalust kasutada mõlemal platvormil arendamisel suuresti ühtset koodibaasi.

Lõputöö eesmärgiks on kasutada nimetatud raamistikku, et realiseerida mobiilirakenduse UNIspotter baasfunktsionaalsus. UNIspotter on antud lõputöö kirjutamise hetkel Ionic raamistikus loodud rakendus, mille põhiülesanne on tutvustada Austria ülikoolide poolt pakutavaid erialaprogramme. Nimetatud rakenduse arendamine kuulub autori igapäevatöö ülesannete hulka.

Põhjus taasluua UNIspotteri mobiilirakendus tuleneb asjaolust, et funktsionaalsuse kasvades on sellel tekkinud märgatavad jõudlusprobleemid. Põhiliseks probleemide tekitajaks võib pidada Ionic raamistiku poolt püstitatud piiranguid, mis ei võimalda kasutada kõiki platvormi poolt pakutavaid võimalusi. Sellest tulenevalt langes otsus proovida React Native raamistikku, kuna see võimaldab kasutada kogu platvormi poolt pakutavat funktsionaalsust, samal ajal eemaldades vajaduse kirjutada ekstensiivset põliskoodi.

Käesolev lõputöö tutvustab mobiilirakenduse arendusprotsessi kasutades React Native raamistikku. Arenduse rõhk on suunatud iOS'i platvormile, kuid lisaks sellele testib autor loodud rakenduse mõningaid osi ka Androidi platvormil, veendumaks et mõlema platvormi peal on võimalik kasutada sama koodibaasi. Töö alguses tutvustab autor UNIspotteri rakendust, React Native raamistikku ning selle põhimõtteid, sellele järgneb autori poolt kasutatud tehnoloogiate kirjeldus koos kommentaaridega ning viimases osas sisaldub arenduskäigu kokkuvõte.

2 UNIspotter

UNIspotter on süsteem, mis ühendab erinevate ülikoolide poolt pakutavad erialad ühte kohta – mobiilirakendusse. Antud lõputöö kirjutamise hetkel on nimetatud süsteem suunatud Austria turule, ühendades sealseid ülikoole ning tuues kokku nende poolt üle 2000 pakutava erialaprogrammi. Rakenduse kasutajate sihtgrupp on peamiselt keskkooli lõpetanud abituriendid ning magistriõppe eriala otsivad tudengid.

UNIspotteri mobiilirakendus on realiseeritud kasutades Ionic ja Cordova raamistikke. Sellest tulenevalt on kogu rakendus kirjutatud veebiarenduskeeltes – HTML, CSS, JavaScript. Ionic vastutab rakenduse vaadete ja loogika eest, kasutades mudel–vaade–kontroller (MVC) tüüpi arhitektuuri ning AngularJS raamistikku. Cordova põhiline eesmärk on Ionicuga loodud rakenduse jooksutamine reaalsel mobiiliplatvormil (iOS, Android), milleks ta kasutab platvormidesse sisseehitatud WebView tüüpi komponenti.

Süsteemi serveripool on realiseeritud Node.js keskkonnale, kasutades Express.js raamistikku ja WebSocket protokolli rakendusega suhtlemiseks. Andmebaasina on kasutusel dokumendipõhine MongoDB ning JavaScripti ja andmebaasi vahelise suhtluse eest vastutab Mongoose ODM raamistik.

Tulenevalt võimalusest kasutada platvormi põliskomponentidest üksnes WebView'd, on UNIspotteri rakenduse kasvades tekkinud märgatavad jõudlusprobleemid, mis põhjustavad kasutajaliidese hangumise või rakenduse sulgumise. Rakendus sisaldab endas mitmeid elemente, mis töötaksid märksa kiiremini kasutades põliskomponente. Näiteks näpuga edasi-tagasi lohistatavad vaated ning Google Maps kaardivaade, mille jaoks saab ära kasutada platvormi sisseehitatud žestide tunnustamise meetodeid ning Google Maps'i poolt pakutavat kaardivaate põlismoodulit. Rakenduse Ionicus loodud versioonis on nimetatud elemendid realiseeritud üksnes veebiarenduskeeltes.

Nimetatud jõudlusprobleemide lahendamiseks on antud lõputöö eesmärgiks tutvuda React Native raamistikuga ning luua UNIspotteri baasfunktsionaalsus antud raamistikus.

3 React Native raamistik

React Native raamistik võimaldab kirjutada mobiilirakendusi iOS'i ja Androidi platvormidele, kasutades mõlemal platvormil ühiseid arenduspõhimõtteid. Ühtne arenduskogemus luuakse kasutades JavaScripti ja React raamistiku põhimõtteid [1] . React raamistik oli algselt loodud veebirakenduste arendamiseks, kasutades komponendipõhist arhitektuuri, kuid laienes 2015. aastal ka mobiiliplatvormidele [2] .

React Native'i eesmärk on tõhus mobiilirakenduste arendus mõlemal platvormil. Selle siht on "*Õpi üks kord, kasuta igal pool*" tüüpi arenduse soodustamine [1] , [2] . Võrdluseks võib tuua HTML5 (kaasa arvatud Ionic) tüüpi rakenduste põhimõtte "*Kirjuta üks kord, kasuta igal pool*", kus luuakse rakendusena veebileht, mis kuvatakse välja vastava mobiiliplatvormi WebView elemendis.

3.1 Komponendid

React raamistikus defineeritakse iga vaate komponent eraldi klassina, mis laiendab *React.Component* ülemklassi. Antud klassil on kohustuslik *render* meetod, mis tagastab vaate visuaalse osa. Tavalise React raamistiku korral tagastatakse *render* meetodis HTML elementidest koosnev vaade, React Native'i puhul luuakse aga vaade põliskomponentidest [1] .

Põliskomponendid on vastava platvormi vaadete loomiseks kasutatavad elemendid. Näiteks kõige tavalisem vaate element iOS'i platvormil on *UIView* ning samaväärne element Androidil on *View*. React Native seob eelnevalt nimetatud põliselemendid üheks React Native'i komponendiks – *View*'ks. Joonis 1 kujutab View elemendi loomist, mille tagajärjel kuvatakse iOS'i platvormil *UIView* ning Androidi platvormil *View*. Selline lähenemine võimaldab ühe platvormi jaoks kirjutatud koodi taaskasutada teise peal.

```
class ExampleComponent extends React.Component {
    render() {
        return (
            <View></View>
        );
    }
}
```

Joonis 1. Vaate loomine React Native raamistikus

Komponentide ühendamine, loomaks keerulisemaid vaateid, toimub sarnaselt HTML'ile, kus ühe elemendi *tag*'ide vahele asetakse teine element. Joonis 2 kujutab olukorda kus ühendatakse üks tavaline vaade ning üks tekstivaade. Antud näite korral loob React Native raamistik iOS'i platvormil *UIView*, mille sees on üks *UITextView*.

```
class ExampleComponent extends React.Component {
    render() {
        return (
            <View>
                <Text>Hello, World!</Text>
            </View>
        );
    }
}
```

Joonis 2. Komponentide ühendamine React Native raamistikus

3.2 Kujundamine

Vaadete kujundamine React Native raamistikus sarnaneb veebilehe kujundamisele CSS'iga. Stiili kirjeldamiseks kasutatakse sarnaseid termineid, kuid väikeste erinevustega. Erinevatele kujundusomadustele viitavaid märksõnu on kohandatud, töötamaks JavaScripti keeles. Näiteks taustavärvi deklareerimiseks kasutatakse viitajat *backgroundColor*, mitte CSS'ile omast *background-color*. Elementide kujundamise aluseks kasutatakse CSS'ist tuntud *flexbox*'i põhimõtteid, mis võimaldab, ilma suurema lisatööta, luua erinevate ekraanisuurustega kohanduvaid vaateid [3] .

Kujunduste jaoks luuakse eraldi *React.StyleSheet* tüüpi objekt, mille sees kirjeldatakse JavaScripti objekti kujul ära elementide stiilid [3] . Selleks, et panna vaate elemendile viitama konkreetne stiil, kirjutatakse see atribuudina andud elemendi külge, sarnaselt sellele, kuidas pannakse HTML elemendile külge *class* atribuut. Joonis 3 kujutab stiili

loomist ja selle sidumist vaate elemendiga, kusjuures *styles* on *React.StyleSheet* tüüpi objekt, mille sees on defineeritud *viewStyle* stiil.

```
class ExampleComponent extends React.Component {
    render() {
        return (
            <View style={styles.viewStyle}></View>
        );
    }
}

const styles = React.StyleSheet.create({
    viewStyle: {
        width: 10,
        height: 10,
        backgroundColor: "#ff0000"
    }
});
```

Joonis 3. Stiili loomine ja rakendamine React Native raamistikus

Stiili objekt defineeritakse kasutades *React.StyleSheet.create* meetodit ning asetatakse samasse JavaScripti faili, kus on antud komponendi kood, millele stiil vastab. Seda tüüpi defineerimine tõlgendab kirjutatud stiilid ümber numbriteks ja hoiab neid raamistiku siseses tabelis, mis garanteerib, et stiilid on rakenduse jooksutamise käigus muutumatud ning nad laetakse ainult üks kord, mitte iga olekumuutuse järel uuesti [3] .

3.3 JavaScript ja põlisplatvorm

React Native kasutab Androidi ja iOS'i platvormidel JavaScripti jooksutamiseks *JavaScriptCore* mootorit. Seda juhul kui rakendusega pole ühendatud Chrome'i silumist, mille korral jookseb kogu JavaScript Chrome'i enda V8 mootori peal ning suhtlus rakendusega käib üle WebSocketi ühenduse [4] (täpsemalt peatükis 4.2.3).

Kogu JavaScripti kood jookseb põliskoodist eraldatult JavaScripti lõimel ning suhtlus põlisplatvormiga toimub asünkroonselt. Suhtlus ja vaadete uuendused saadetakse põliskoodile tsüklites (*batches*), mis tähendab, et sõnumid pannakse mingiks hetkeks ootele ning iga iteratsiooni käigus saadetakse mitu käsku korraga [5] .

Kirjeldatud tsükliline suhtlus võib aga tekitada näilisi jõudlusprobleeme ning vaate hangumist. Suure osa uute seadmete ekraanide uuendussagedus on 60 korda sekundis, sellest tulenevalt peab ühe tsükli pikkus jääma alla 16,67 ms. Kui JavaScripti lõim ei

suuda selle aja sees platvormile vaate uuendust saata, jätab platvorm ühe kaadri vahele. Jättes vahele rohkem kui 100 ms jagu kaadreid, annab see kasutajale tunda [5] . Seetõttu tuleb arendajal arvestada, et JavaScripti lõimel on tehtavad animatsioonid tihti ebasujuvad. Lahenduseks nähakse kõikide JavaScriptis tehtavate animatsioonide teisaldamist UI lõimele.

UI lõimele animatsioonide teisaldamine tähendaks, et JavaScripti lõim saab tegeleda rakenduse loogika töötlemisega, samal ajal kui UI lõim kannab hoolt vastavate kaadrite kuvamise eest [5] . Nimetatud lahendus pole aga antud lõputöö kirjutamise hetkel React Natve raamistikus implementeeritud.

Oma rakenduse animatsioonide optimeerimiseks ja hangumise vähendamiseks on arendajal võimalik kasutada *InteractionManager.runAfterInteractions* meetodit koos *callback* parameetriga, mille *callback* funktsioon pannase tööle, kui hetkel käimasolev animatsioon (näiteks navigeerimine ühelt lehelt teisele) on lõppenud [5] . Pärast animatsiooni lõppemist saab arendaja teha muud vajalikud vaate uuendused, mis animatsiooni ajal oleks põhjustanud vaate hangumise.

3.4 Põlismoodulid

Arenduse käigus võib tekkida vajadus kasutada platvormi-spetsiifilisi funktsioone, mille jaoks pole veel vastavat React Native'i moodulit loodud. Näiteks antud lõputöö kirjutamise hetkel (RN versioon 0.24) puudub arendajal võimalus pääseda JavaScriptist ligi platvormi poolt pakutavatele kalendrihaldus meetoditele. Põlisfunktsioonidele lisaks, võib arendajal olla soov taaskasutada juba olemasolevat Java või Swifti koodi, ilma et seda tuleks JavaScriptis uuesti implementeerida [6] , [7] .

React Native raamistik võimaldab arendajal vajaduse korral kirjutada põliskoodi (Java, Objective-C, Swift) ning teha seda nähtavaks JavaScriptile. React Native käsitleb põlismoodulina Objective-C klassi, mis implementeerib *RCTBridgeModule* protokolli või Java klassi mis laiendab *ReactContextBaseJavaModule* ülemklassi. Loodud moodul ja kõik selle juurde kuuluvad meetodid muutuvad JavaScriptis kättesaadavaks läbi spetsiaalse *React.NativeModules* mooduli [6] , [7] .

Kuna suhtlus JavaScripti ja põlisplatvormi vahel toimub asünkroonselt, siis ei saa põlismeetodi väljakutse vastust tagastada tavalise *return*'iga, vaid tuleb kasutada

callback funktsioone. React Native toetab *callback*'ina nii tavalisi funktsioone, kui ka ECMAScript 2015 standardis kirjeldatud *promise* tüüpi funktsioone [6] , [7] .

Põlismoodulid saavad JavaScripti saata sündmusi ka ilma, et neid oleks spetsiaalselt välja kutsutud. Platvormi poole peal tuleb sündmus anda ette *RCTDeviceEventEmitter* (Android) või RCTEventDispatcher (iOS) moodulile, mis omakorda väljastab selle vastavalt *React.DeviceEventEmitter*'i või *React.NativeAppEventEmitter*'i kaudu JavaScripti koodi [6] , [7] .

3.5 Põlisvaated

React Native pakub omalt poolt mähiseid (*wrapper*) paljudele (32-le, RN 0.24) platvormi komponentidele, nagu näiteks *ScrollView* ja *TextInput*. Raamistiku poolt pakutud kompnendid on piisavad, et kirjutada valmis lihtne rakendus, kuid keerulisemate süsteemide puhul tekib vajadus kasutada selliseid komponente mille jaoks vastav mähis puudub. Selleks võimaldab React Native luua arendajal vaateid põliskoodis ning neid JavaScriptis välja kutsuda [1] , [8] , [9] .

Näiteks pakub React Native omalt poolt piiratud funktsionaalsusega kaardivaate komponenti, mis kuvab iOS'i platvormil Apple Maps tüüpi kaardi ning Androidil Google Maps kaardi. Selleks, et saada mõlemal platvormil kasutada täisfunktsionaalset Google Mapsi, tuleb see põliskoodis luua ning hiljem JavaScripti eksportida (täpsemalt peatükis 4.3.2).

Põlisvaate eksportimiseks JavaScripti, peab iOS'ile loodav vaade laiendama *RCTViewManager* klassi ning implementeerima *-(UIView *)view* meetodit [9] . Androidi vaade peab laiendama *SimpleViewManager* klassi ning implementeerima *createViewInstance* ja *getName meetodeid* [8] . JavaScriptis pääseb loodud komponendile ligi *React.requireNativeComponent* meetodi kaudu ning selle kuvamine oma rakenduse vaates toimub sama moodi nagu on kirjeldatud peatükis 3.1.

4 Autori poolt kasutatud tehnoloogiad koos kommentaaridega

Antud lõputöö käigus valmis mobiilirakendus iOS'i platvormile kasutades React Native raamistikku. Rakenduse loomiseks kasutas autor erinevaid programmeerimiskeeli, tehnoloogiaid ja tööriistu. Järgnevalt kirjeldab autor kasutatud tehnoloogiaid, lisades neile omapoolsed kommentaarid ja põhjendused.

4.1 Keeled

Autor kasutas arendusprotsessi käigus peamiselt kolme programmeerimiskeelt – JavaScript, Objective-C ning Swift. Suurem töö toimus JavaScriptis, mille hulka kuulus vaadete ja rakenduse loogika kirjuamine. Objective-C ja Swift leidsid kasutust selleks, et luua iOS'i platvormile põlismooduleid. Järgnevalt kirjeldab autor nimetatud keeli lähemalt.

4.1.1 JavaScript

Suurem osa arendusprotsessist toimus kasutades JavaScripti põhjusel, et React Native raamistik on oma olemuselt JavaScripti raamistik. JavaScripti koodi jooksutamine toimus kas JavaScriptCore või V8 mootoris. Erinevate mootorite kasutamise tingis asjaolu, et React Native jooksutab erinevates arendusrežiimides koodi erinevates keskkondades. Arendust erinevate mootorite kasutamine ei mõjutanud, kuid tuli arvestada asjaoluga, et erinevates keskkondades ei ole sama koodi järjepidevus garanteeritud.

Lisaks kasutas autor JavaScripti serveri poolel. Arendatud rakenduse server jookseb Linuxi keskkonnas mille peale on paigaldatud Node.js server ja MongoDB andmebaas. Kuna nimetatud tehnoloogiad jäävad antud lõputöö skoobist välja, ei selgita autor neid pikemalt.

Autori hinnangul oli võimalus, kasutada peaaegu igas arendusetapis ühte programmeerimiskeelt, väga mugav ning soodustas arendusprotsessi kiirust. Samas teadvustab autor asjaolu, et JavaScripti mootoreid on võrdlemisi palju ning kõik need ei toeta samasid standardeid, mis võib tekitada segadust. Eriti andis tunda serveri ja

rakenduse mootorite erinevus, kus osad rakenduse poolel kasutatavad mugavused ei töötanud serveris.

Üldjuhul on JavaScripti süntaks autori arvates kergesti loetav ning mugav kirjutada. Selle poolt pakutav dünaamiline tüüpimine teeb arendamise kohati lihtsamaks, eemaldades andmetüüpide defineerimise vajaduse, kuid suuremate süsteemide puhul soodustab see vigade tekkimist.

Üheks JavaScripti positiivseks küljeks, autori hinnangul, on tema laiendatavus. Kasutades *npm* (Node Package Manager) pakihaldusmehhanismi, saab JavaScripti koodi laiendada teiste arendajate poolt loodud lisamoodulitega. Teisest küljest toob väliste moodulite kasutamine kaasa lisa sõltuvuse tundmatust koodist, mille töökindlus ei ole alati garanteeritud.

4.1.2 Objective-C

Objective-C kasutamise tingis asjaolu, et kogu React Native raamistiku iOS'i platvormi pool on sisemiselt kirjutatud Objective-C keeles. Sellest tulenevalt ei olnud antud keele kasutamine otseselt autori valik, vaid paratamatus.

Selleks, et kasutada raamistiku pakutavat võimalust luua põlismooduleid ja -vaateid, tuli kasutada platvormi-spetsiifilist koodi. iOS'i platvormile arendamiseks saab kasutada kahte programmeerimiskeelt – Objective-C ja Swift [10] , [12] . Objective-C'd kasutas autor peamiselt selleks, et luua sild (*bridge*) JavaScripti ning põliskoodi ja -vaadete vahele. Selleks tuli implementeerida vastavalt *RCTBridgeModule* ja *RCTViewManager* Objective-C protokollid.

Autori hinnangul on Objective-C, võrreldes JavaScripti ja Swiftiga, võrdlemisi keeruline keel. Selle süntaks on, peamiselt Javaga kokku puutunud arendajale, harjumatu, mis teeb koodist arusaamise ja selle kirjutamise raskeks. Sisemiselt on Objecive-C keel tavaline C keel, mille tõttu on ta sama võimekas kui C, kuid tema süntaks lähtub SmallTalk keele süntaksist [11] .

Vaatamata React Native'i ja Objective-C tihedale seotusele, oli siiski võimalik teha suur osa platvormi-spetsiifilisest programmeerimisest keeles Swift, mis on autori hinnangul lihtsamini loetav ja mugavam kirjutada.

4.1.3 Swift

Sarnaselt Objective-C'le, on Swift Apple'i platvormidel kasutatav objektorienteeritud programmeerimiskeel. See on loodud töötama Objective-C's kirjutatud Apple'i platvormide siseste programmiliidestega ning samuti suure hulga muu Apple'i toodete jaoks kirjutatud Objective-C koodiga [12] .

Tänu võimalusele siduda Swifti koodi ja Objective-C koodi, sai autor teha suure osa vajalikust põlisplatvormi poolsest arendusest Swiftis. Selleks, et teha mõni Swifti klass või meetod nähtavaks Objective-C koodile tuli sellele Swiftis lisada vastav annotatsioon (vt. Joonis 4). Selline lähenemine võimaldas kogu mooduli loogika kirjutada Swiftis ning kasutada Objective-C'd üksnes loodud mooduli eksportimiseks JavaScripti koodi.

```
3
4  @objc(CalendarManager)
5  class CalendarManager: NSObject {
6
7    @objc func addEvent(name: String, details: NSDictionary, callback: (NSObject) -> ()) -> Void {
8
```

Joonis 4. Swifti ja Objective-C koodi sidumine

Autor eelistab Swifti Objective-C'le, kuna see on lihtsamini loetav kui Objective-C ning nõuab vähem kohustuslikku koodi (*boilerplate* koodi), et töödata. Swifti süntaks on sarnane JavaScripti süntaksile, mis tegi kahe koodi koos kirjutamise, ja ka arendusprotsessi üldiselt, lihtsamaks. Swifti negatiivseks küljeks on autori hinnangul asjaolu, et tegemist on võrdlemisi noore keelega, mis pidevalt muutub ja uueneb. Selle tõttu tuleb arvestada olukordadega, kus olemasolev kood vajab muutmist, kuna mõni element on keelest kaotatud või ümber muudetud. Õnneks aitab nimetatud muutustega Xcode arenduskeskkond, mida kirjeldab autor peatükis 4.2.2.

4.2 Tööriistad

Arenduse käigus kasutas autor kolme tüüpi tööriistu – IDE ehk arenduskeskkond, JavaScripti konsool ning virtuaalsed seadmed, nagu iOS'i simulaatorid ning Androidi emulaatorid. Arenduskeskkondadena kasutas autor WebStromi, Xcode'i ning Android Studiot. Nimetatud kahe viimase IDE'ga tulid kaasa rakenduse jooksutamiseks hädavajalikud virtuaalsed seadmed. Sellele lisaks kasutas autor ka Google Chrome'i pakutavaid arendajatööriistu, mille JavaScripti konsool mängis rakenduse testimisel juhtivat rolli. Järgnevalt kommenteerib autor kasutatud tööriistu detailsemalt.

4.2.1 WebStorm

WebStorm on peamiselt veebiarenduseks mõeldud arenduskeskkond (IDE) [13] . Autor otsustas kasutada arendusel WebStromi, kuna see põhineb Intellij IDEA IDE'l, millega oli autor juba tuttav ning mille tõttu puudus vajadus uue keskkonna eraldi tundmaõppimiseks.

Erinevalt Java arenduseks mõeldud Intellij IDEA'st, toetab WebStorm põhjalikult JavaScripti ning pakub mugavaks arendamiseks lisavõimalusi, nagu koodi soovitused, koodi lõpetamine ning refaktoreerimine. Samuti kasutas autor arenduse käigus pidevalt WebStromi sisseehitatud käsurida, mis kaotas vajaduse eraldi käsurearakenduse ning lisaakna järele.

WebStormi asemel kaalus autor ka Nuclide'i kasutamist, mis on spetsiaalselt loodud Reacti projektide arendamiseks. Autor otsustas Nuclide'i mitte kasutada, kuna tegemist on antud lõputöö kirjutamise hetkel väga uue tarkvaraga, mis töötas autorile kohati harjumatult ning millel puudus osa autori poolt nõutavast funktsionaalsusest.

WebStormi puuduseks toob autor React Native raamistiku otsese toe puudumise. WebStrom võimaldab küll erinevate pistikprogrammide (*plugins*) installeerimise, kuid React Native'i jaoks puudub eraldi selline lisa. Selle tõttu tuli ette olukordi, kus WebStrom ei suutnud interpreteerida kirjutatud koodi ning värvis seda ebareeglipäraselt ja esitas veateateid. Tulevikus plaanib autor siiski üle minna Reacti spetsiifilisele arenduskeskkonnale – Nuclide.

4.2.2 Xcode ja iOS'i simulaator

Xcode on arenduskeskkond, mis on mõeldud Apple'i platvormidele tarkvara arendamiseks. See toetab nii Swifti kui ka Objective-C koodi ning tunneb iOS'i arenduseks vajalikke Cocoa ja Cocoa Touch raamistikke [14] . Xcode on ainuke arenduskeskkond mis võimaldab arendada iOS'i platvormile, seetõttu oli selle kasutamine autorile kohustuslik.

Xcode'i kasutas autor peamiselt põliskoodi kirjutamiseks ja rakenduse jooksutamiseks. Jooksutamisel toimus kaks sündmust – läks tööle *React Packager* käsurea programm ning iOS'i simulaator (või rakendus füüsilise seadme peal). Nimetatud käsurea programm kuulas JavaScripti koodis tehtavaid muudatusi ning serveeris neid pidevalt

ette jooksvale rakendusele. Tehtud koodimuudatus kajastus kohe jooksvas rakenduses (*Live Reload*), seda aga ainult JavaScripti koodi muutudes. Põliskoodis muudatusi tehes pidi rakenduse uuesti kompileerima ja jooksutama.

Lisaks pakub Xcode võimalust näha jooksva rakenduse konsooli, kuhu programm väljastab koodist tulnud logi, hoiatusi ja veateateid. Tegemist on iOS'i spetsiifilise (Swifti ja Objective-C) konsooliga, mille tõttu leidis see vähem kasutust kui peatükis 4.2.3 kirjeldatav Google Chrome'i JavaScripti konsool.

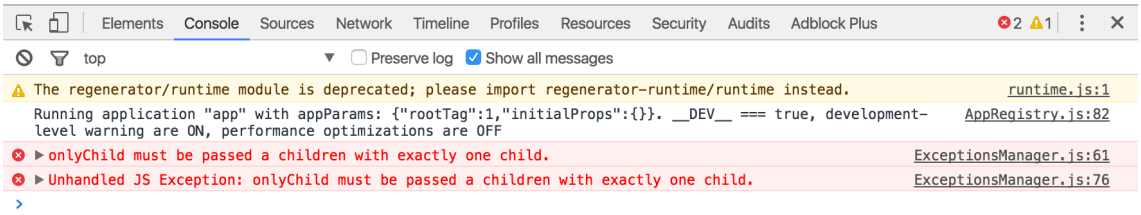
Rakenduse jooksutamiseks ja testimiseks on Xcode'i sisse ehitatud erinevad iOS'i seadmete simulaatorid. Arendajal on jooksutamise hetkel võimalik valida millise iOS'i versiooni ja seadme peal ta tahab oma rakendust testida. Autor kasutas arenduse käigus peamiselt 4" ja 4,7" ekraanidega iPhone'i simulaatoreid ning 9,7" tahvelarvuti simulaatorit.

Autori hinnangul on Xcode'i üldine kasutusmugavus hea, kuid üheks märgatavaks puuduseks on refaktoreerimise puudumine. Samuti peab esile tõstma Xcode'i sisseehitatud väga kiireid simulaatoreid, mis koos React Native'i poolt pakutava *Live Reload* funktsionaalsusega vähendavad koodi kompileerimisega tulenevat friktsiooni ning kiirendavad märgatavalt arendusprotsessi.

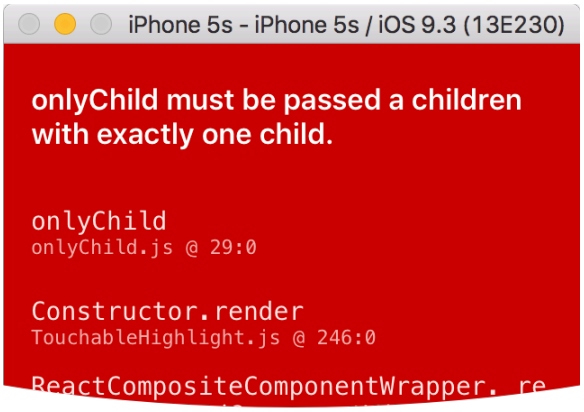
4.2.3 Google Chrome'i arendaja tööriistad

Google Chrome'i kasutamise tingis React Native raamistiku pakutav võimalus, kasutada rakenduse silumisel Google Chrome'i arendaja tööriistu, peamiselt JavaScripti konsooli. Rakendust jooksutavas simulaatoris või seadmes on võimalik sisse lülitada *Debug In Chrome* režiim, mis avab Chrome'i akna aadressil *localhost:8081/debugger-ui*, mille arendaja tööriistad kajastavad jooksva rakenduse seisu.

Chrome'i konsoolis kuvatakse kõik programmikoodist tulevad sõnumid. Nende hulka kuuluvad kogu JavaScriptist väljastatud logi, erinevad hoiatussõnumid ning veateated. JavaScripti koodi väljundile lisaks, kuvab Chrome'i konsool ka põliskoodist saadetavaid veateateid, kuid mitte logi ja hoiatussõnumeid. Kõik veateated näidatakse, lisaks konsoolile, välja ka rakendust jooksutava seadme ekraanil.



Joonis 5. Veateate kuvamine Chrome'i konsoolis



Joonis 6. Veateate kuvamine seadme peal

Kui rakendus on *Debug In Chrome* režiimis, jookseb kogu rakenduse JavaScript Chrome'i brauseri JavaScripti mootoris – V8. Suhtlus JavaScripti ja põliskoodi vahel toimub sellisel juhul üle WebSocketi ühenduse. *Debug In Chrome* režiimi välja lülitades, kasutab React Native JavaScripti jooksutamiseks, iOS'i ja Androidi seadmetel, JavaScriptCore mootorit [4] . Selline keskkondade erinevus arenduse käigus tunda ei andnud, kuid arendaja pidi olema teadlik sellega kaasneda võivatest anomaaliatest.

Autor hindab võimalust kasutada arenduse käigus Chrome'i JavaScripti konsooli väga kasulikuks. Tegemist on JavaScripti programmide arendamisel hädavajaliku tööriistaga, mis annab arendajale tagasisidet programmi seisust ning teavitab tekkinud ja potentsiaalsetest vigadest. Chrome'i arendajatööriistad olid autoril arenduse käigus alati avatud.

4.2.4 Android Studio ja Androidi emulaator

Android Studio on Androidi platvormile rakenduste loomiseks mõeldud arenduskeskkond, mis põhineb sarnaselt WebStormile Intellij IDEA IDE'l [15] . Tekstiredaktorile lisaks pakub nimetatud arenduskeskkond Androidi platvormi jaoks

vajaminevate tööriistade ja teekide haldamise süsteemi (*SDK Manager*) ning erinevate Androidi seadmete emulaatoreid.

Antud lõputöö käigus kasutas autor Android Studiot, et paigaldada arvutisse rakenduse jooksutamiseks vajalikud teegid ja tööriistad – peamiselt Androidi versiooni 6 poolt nõutavad vahendid. Autor installeeris Android Studio kaudu ka ühe Androidi seadme emulaatori (LG Nexus 5), mille peal toimus rakenduse jooksutamine ja testimine. Arenduskeskkonna poolt pakutavat tekstiredaktorit autor ei kasutanud, kuna põlismoodulite kirjutamine Androidi platvormile jäi antud lõputöö skoobist välja.

Android Studio poolt pakutavate emulaatorite asemel kaalus autor ka Genymotion tarkvara poolt pakutavaid emulaatoreid. Genymotioni emulaatorid on ajalooliselt olnud Androidi arenduse juures populaarsed, kuna nende kiirus ja jõudlus on olnud märgatavalt parem võrreldes Android Studio emulaatoritega. Sellele vaatamata otsustas autor kasutada Android Studio emulaatoreid, kuna vahetult enne antud lõputöö kirjutamise algust välja tulnud Android Studio uue versiooniga, kaasnes pakutavate emulaatorite märgatav jõudluse paranemine.

Olles testinud rakendusi mõlema programmi poolt pakutavate emulaatoritega, eelistab autor Android Studiot, kuna selle poolt pakutavad emulaatorid on jõudluselt võrreldavad Genymotioni omadega ning kohati näiliselt kiiremad. Samuti eemaldab see järjekordse lisaprogrammi vajaduse.

4.3 Tehnoloogiad

React Native'le lisaks leidsid arendusel kasutust ka teised tehnoloogiad. Üheks mahukamaks osutus Google Maps SDK, mille abil realiseeris autor rakendusele kaardivaate ning asukoha otsingu ja selekteerimise. Nimetatud ja teiste väliste teekide integreerimisel kasutas autor CocoaPods sõltuvuste haldamiste vahendit. Antud peatükis tuleb lisaks juttu React raamistiku rakendustele omasest Flux tüüpi arhitektuurist ning ühest selle spetsiaalsest implementatsioonist – Redux. Samuti kirjeldab autor Socket.IO teeki, mille abil toimus WebSocketi ühendus rakenduse serveriga.

4.3.1 CocoaPods

CocoaPods on sõltuvuste haldamise süsteem Swifti ja Objective-C Cocoa raamistiku (Apple'i platvormide peamine programmiliides) projektide jaoks [16] . Sarnaselt *npm*'ile JavaScriptis ja *gradle*'le Javas, võimaldab CocoaPods importida lihtsal viisil oma projekti väliseid mooduleid. CocoaPodsile lisaks on Swifti projektide jaoks olemas ametlik *spm* (Swift Package Manager), kuid see on antud lõputöö kirjutamise hetkel alles oma arengu algstaadiumis ning toetab üksnes tavalisi Swifti mooduleid. CocoaPodsi eelis *spm*'i ees on see, et ta toetab Cocoa raamistikuga (iOS ja OS X) projekte ning on juba laialdaselt kasutusel.

Autori peamine otsus kasutada CocoaPodsi, ning mitte integreerida väliseid mooduleid käsitsi, tulenes asjaolust, et Google Maps'i SDK integreerimine oma iOS'i projekti käib ametlikult läbi CocoaPodsi ning selle käsitsi integreerimine oleks võrdlemisi suur töö. CocoaPodsiga kaasneb aga teatud hulk lisafaile ning olukord, kus ametlikku projektifaili ei tohi enam kasutada. Selle asemel luuakse uus projektifaili, mis sisaldab endas kõiki CocoaPodsiga installeeritud mooduleid [16] . Seda peab autor üheks CocoaPodsi puuduseks, ning näeb ette olukordi kus antud situatsioon võib tekitada segadust.

Olles tuttav väliste moodulite käsitsi integreerimisega, hindab autor CocoaPodsi kui väga kasulikku tööriista, mis teeb arendaja töö mugavamaks. See laseb arendajal keskenduda rohkem programmi kirjutamisele, eemaldades ebavajalikud keerukused, mis kaasnevad väliste moodulite käsitsi integreerimisel. Samas eelistaks autor kasutada Swifti sisseehitatud *spm*'i, mis kaotaks CocoaPodsi ja sellega kaasnevate lisafailide vajaduse, kuid *spm* ei toeta, antud lõputöö kirjutamise hetkel, iOS'i projekte.

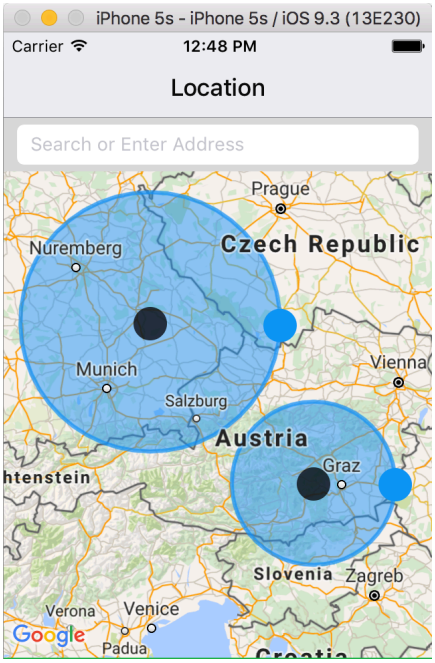
4.3.2 Google Maps iOS SDK

Üks suurem väline moodul, mida autor antud lõputöö käigus kasutas, oli Google Maps'i iOS SDK. Põhjuseks oli asjaolu, et loodav rakendus pidi sisaldama kaardivaadet, mis võimaldaks otsida, kuvada ja selekteerida erinevaid piirkondi. Autor otsustas Google Maps'i kasuks, kuna tal oli sellega juba eelnev kogemus ning Google Maps pakub rohkem funktsionaalsust, kui iOS'i sisseehitatud Apple Maps.

Käsitletavas SDK's leidis kasutust kolm põhilist komponenti – kaardivaade, ring, otsing. Kaardivaate jaoks kasutas autor *GMSMapView* elementi mis tagastas vaikimisi

Google Maps'i kaardi. Kaardvaatele anti ette kaamera keskpunkti koordinaadid ning suumingu tase.

Asukoha selekteerimiseks kasutas autor *GMSCircle* komponenti, mis joonistas kaardivaate peale etteantud raadiuse ja keskpunktiga ringi, mille käest oli hiljem võimalik küsida samasid parameetreid, et nende põhjal arvutada ala, mis jäi ringi alla. Sellele lisaks implementeeris autor funktsionaalsuse, mis võimaldas loodud ringe liigutada ning nende raadiust muuta. Liigutamiseks asetas autor ringi keskpunkti ümmarguse *UIVew* tüüpi vaate, millest näpuga "kinni võttes" sai ringi kaardil lohistada. Raadiuse muutmiseks asetas autor samasuguse ümmarguse vaate ringi piiri peale, mille lohistamine muutis samal ajal ringi raadiust (vt. Joonis 7).



Joonis 7. Mikk Kärneri poolt loodud piirkonna selekteerimise komponent

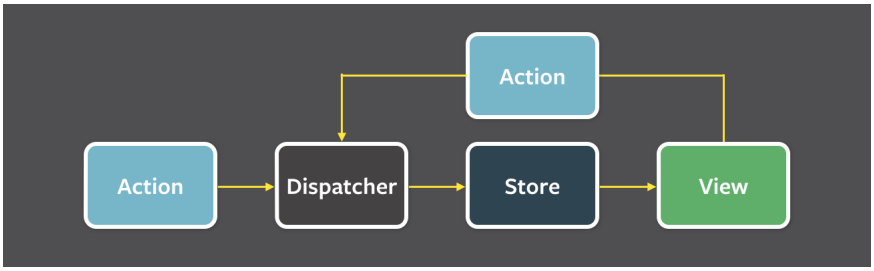
Asukoha otsingu jaoks kasutas autor *GMSPlacesClient*'it, mis võimaldas teha erinevat tüüpi päringuid Google Maps'i API pihta. Peamised päringu tüübid, mida autor kasutas, olid *Autocomplete Query* asukoha otsimiseks nime järgi ning *Look Up Place ID* asukoha koordinaatide saamiseks.

Kogu kirjeldatud töö antud SDK'ga toimus põlispoole peal Swiftis ning loodud komonendid eksporditi JavaScripti, kasutades React Native'i *RCTViewManager* ja *RCTBridgeModule* protokolle.

Google Maps'i iOS SDK pakub palju võimalusi erinevat tüüpi funktsionaalsusega kaardivaadete loomiseks, kuid selle dokumentatsioon on autori hinnangul kesine. Tegemist on Objective-C keeles kirjutatud SDK'ga, kuid sellele vaatamata saab seda kasutada Swiftis. SDK'd kirjeldaval veebilehelt leiab näiteid nii Objective-C, kui ka Swifti kohta, kuid dokumentatsioon on rangelt Objective-C põhine [17] .

4.3.3 Flux ja Redux

Flux on arhitektuuritüüp, mida kasutavad peamiselt React raamistikus kirjutatud süsteemid. See koosneb kolmest komponendist – dispetšer (*dispatcher*), andmete hoidlad (*store*) ja vaade (*view*). Sündmuste liikumine toimub alati ühes suunas: dispetšer, hoidla, vaade. Iga andmete muudatus algab sündmusega (*action*), mis saab alguse kas vaates toimunud sündmusest (näiteks kasutaja vajutas nuppu) või väljastpoolt (näiteks serverist) saabunud sõnumist. Sündmus saadetakse alati dispetšerisse, mis kutsub välja andmete muudatuse hoidlas, mis omakorda põhjustab vaate uuendamise [20] .



Joonis 8. Flux arhitektuuri struktuur

Erinevalt MVC tüüpi arhitektuurist, puudub Flux'is vaadet juhtiv kontroller tüüpi komponent. Selle asemel on spetsiaalsed kontroller–vaade tüüpi React'i komponendid, mis on ennast registreerinud kuulama muudatusi teatud andmete hoidlatele. Tuleb siiski täpsustada, et iga React'i komponent ei ole kontroller–vaade tüüpi komponent, vaid ainult need, mis kuulavad andmete muudatusi [20] .

Flux tüüpi arhitektuuri implementeerimiseks kasutas autor, Flux'i põhimõtteid kasutavat kuid mõningate erinevustega, Redux'it. Redux eemaldab Flux'ist dispetšeri ning vajaduse (ja võimaluse) mitme andmehoidla järele. Selle asemel on Redux'is *reducer* tüüpi komponent, mille kaudu toimub rakenduse hetkeseisundi (*store*) muutmine. *Reducer* funktsioon võtab sisendiks hetkeseisundi ning käsu nimetuse, mille põhjal ta

tagastab uue seisundi [21] . Redux'i lähenemine Flux arhitektuurile on autori hinnangul hea, kuna see vähendab rakenduse keerukust, kasutades ainult ühte andmehoidlat, kuid siiski võimaldab hoida rakenduse erinevate osade loogikat lahus, pakkudes võimalust luua iga osa jaoks eraldi *reducer*.

Autori hinnangul sobib suurte süsteemide loomiseks Redux tüüpi arhitektuur paremini, kui MVC arhitektuur. Põhjusel, et MVC rakendustes võib ette tulla olukordi kus erinevate vaadete mudelid sõltuvad üksteisest, või nad sisaldavad samasid andmeid. Ühe vaate mudeli muutmine peab välja kutsuma ka teiste mudelite muutmise, mis võib omakorda kutsuda välja uusi muutusi. Reduxi puhul puudub selline muudatuste ülekandmine, kuna rakenduse olek (andmete hetkeseis) on kogu rakenduse piires alati ühine.

4.3.4 Socket.IO

Loodud rakenduse ja serveripoole suhtlus toimus kasutades WebSocket protokolli. WebSocketi eelis AJAX–HTTP tüüpi päringute ees, on asjaolu, et ühendus kliendi ja serveri vahel jääb kogu sessiooni vältel avatuks. Tegemist on täisdupleks tüüpi ühendusega, mille tõttu saavad sõnumid liikuda mõlemas suunas samaaegselt [19] . Selline lähenemine eemaldab vajaduse korduvate klient–server suunas päringute järele, mille eesmärk oleks serveris toimunud muutustest teadasaamine. Selle asemel saab server teavitada klienti muudatustest ise, kasutades juba olemasolevat WebSocketi ühendust.

Rakenduse loomise hetkel oli WebSocketi implementeerimiseks serveripoolel kasutusel Socket.IO programmiliides. Sellest tulenevalt kasutas autor React Native'i poolel sama lahendust. WebSocket protokolli implementeerimisele lisaks, pakub Socket.IO tagavara protokolle (AJAX, JSONP), juhul kui WebSocketi ühendus pole saadaval või peaks nurjuma [18] .

Socket.IO integreerimine React Native'i projekti toimus läbi Node.js pakihaldussüsteemi – npm. Selleks tuli paigaldada lisamoodul nimega *socket.io-client* ning nimetatud moodul importida JavaScripti koodi, lisaks tuli defineerida serveri aadress, kus jookseb *socket.io* serverirakendus (vt. Joonis 9).

```
24  import io from 'socket.io-client/socket.io';
25
26  const socket = io('http://localhost:3000', {
27      transports: ['websocket']
28  });
29
30  socket.on('connect', () => {
31      console.log('connected!');
32  });
```

Joonis 9. WebSocketi ühenduse loomine ning sündmusele reageerimine

Autori hinnangul oli WebSocket protokolli implementeerimine arendatavas rakenduses võrdlemisi mugav. Vajaminev koodihulk on võrreldes AJAX päringute programmeerimisega märgatavalt väiksem ning kontseptuaalselt lihtsam. Samuti vähendab antud protokoll ebavajalikke päringuid serverile, võimaldades serveril rakendusele ise sõnumeid saata.

5 Arenduskäik

Antud peatükis annab autor ülevaate lõputöö käigus valminud mobiilirakenduse arenduskäigust. Järgnevast järeldub, et üheks arenduse etapiks osutus ka React Native raamistikuga tutvumine, kuhu hulka kuulus eraldi näiterakenduste loomine. Lisaks võtab autor kokku peamised arendusetapid, nagu vaadete ja loogika programmeerimine ning rakenduse testimine. Testimise juurde kuulus ka olemasoleva iOS'i platvormile kirjutatud koodi jooksutamine Androidi platvormil.

5.1 React Native raamistikuga tutvumine

Esimese etapina tegeles autor React Native raamistiku kohta info otsimisega ning sellega tutvumisega. Peamisteks allikateks osutusid raamistiku dokumentatsioon, selle kohta kirjutatud blogipostitused (nii raamistiku autorite kui ka teiste huviliste poolt) ning videod. Videote näol oli üldjuhul tegemist konverentsidel tehtud React Native'i esitluste salvestistega ning raamistikku tutvustavate õppevideotega.

Raamistikuga tutvumise hulka kuulus ka väikeste näiteprogrammide loomine, mis kinnistas raamistiku tööpõhimõtteid ning andis aimu loodava rakenduse struktuurist. Näiteprogrammide hulk piirdus kahega, kuna uue projekti ülesseadmine nõudis võrdlemisi palju kohustuslikku koodi, mille pidevas korrutamises ei näinud autor vajadust.

5.2 Vaadete loomine

Teise etapina alustas autor rakenduste vaadete loomisega. Iga vaate jaoks lõi autor projektistruktuuri eraldi kausta, kuhu sisse asetas kõik antud vaate juurde kuuluvad komponendid ja loogikat sisaldavad koodifailid. Vaadete kujundamisel jäljendas autor juba olemasolevat Ionic raamistikus loodud UNIspotteri rakendust, mis lihtsustas märgatavalt arendusprotsessi.

Vaatekomponendid, mis leidsid kasutust mitme koha peal, tõstis autor eraldi failidesse, reeglina konkreetse vaate all olevasse komponentide kausta või projekti juurkausta, kui

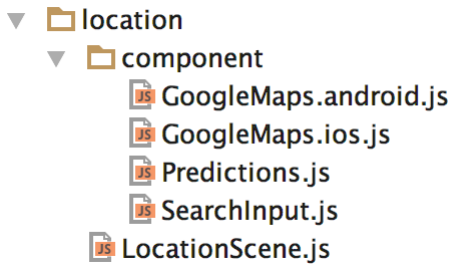
oli tegemist universaalsete komponentidega mida kasutati mitme vaate poolt. Peamised universaalsed komponendid olid nupud ja ümmargused profiilipildi vaated.

Samuti tõstis autor programmikoodist lahku kõik universaalselt kasutatavad värvikoodid, et tagada kasutajaliidese järjepidevus üle kogu rakenduse. Värvid ning kõik teised rakenduse piires kasutust leidnud konstandid asetati *constants* kausta, kus oli iga konstandi tüübi jaoks eraldi JavaScripti fail.

```
1  export const PRIMARY_GREEN = '#2AAF4C';
2  export const PRIMARY_GREEN_DARK = '#1e7b35';
3  export const TEXT_COLOR = '#595959';
4  export const SUBHEADER_COLOR = '#a6a6a6';
5  export const CELL_BORDER_COLOR = '#cccccc';
```

Joonis 10. Lõige Colors.js konstantide failist

Lisaks kasutas autor võimalust luua mõlema platvormi jaoks eraldi komponendifailid. Lisades failinime lõppu *.android.js* või *.ios.js*, oskab React Native eristada vastavalt Androidile ja iOS'ile kirjuatud koodi. Peamiselt leidis see kasutus iseloodud põlisvaadete (näiteks Google Maps kaardivaate) kasutamisel.



Joonis 11. Kaardivaate komponendi struktuur koos eraldi Androidi ja iOS'i failidega

Vaadete loomist React Native raamistikuga hindab autor väga mugavaks. Esiteks, sama koodi kasutamine mõlemal platvormil vähendab märgatavalt vajamineva programmeerimise hulka. Teiseks on kujundamisel kasutatav, HTML ja CSS tüüpi, notatsioon selgesti mõistetav ning lihtsasti muudetav, kuna kogu vaate struktuur ja stiil on defineeritud ühes failis. Põlisvaadete loomise ja kasutamise võimalus annab arendajale lisaks vabad käed loomaks oma rakenduse spetsiifilisi vajadusi katvaid komponente, mis soodustab rakenduse kvaliteeti.

5.3 Rakenduse loogika kirjutamine

Vaate kujundamisega samaaegselt toimus ka seda juhtiva loogika kirjutamine, põhjusel et React raamistikus asetsevad vaate kujundus ning selle loogika samas failis. Sellele vaatamata jaotus loogika ka teistesse failidesse. Nimelt reageeris vaatega toimuvatele otsestele sündmustele kood komponendi enda failis, kuid sellele vastav serveripäring või rakenduse olekumuutus (läbi Redux'i) olid eraldi failides. Suur osa rakenduse üldise oleku loogikast paiknes *reducer* tüüpi komponentides, mida on detailsemalt kirjeldatud peatükis 4.3.3.

```
4   const initialState = {};
5
6   export default (state = initialState, action) => {
7
8       switch (action.type) {
9           case OPEN_PROGRAM_DETAIL:
10              const props = {
11                  programId: action.id
12              };
13              Actions.program(props);
14              return state;
15          default:
16              return state;
17      }
18
19  }
```

Joonis 12. Lõige nimekirjavaate *reducer* failist

JavaScriptis kirjutatud loogikale lisaks, paiknes osa programmikoodi iOS'i platvormi poolel Swiftis. Peamine Swiftis kirjutatud loogika puudutas kalendrirakendusega suhtlemist ning Google Maps kaardi juhtimist. Suur osa põliskoodi kirjutamise keerukusest seisnes autori hinnangul selle suhtluses JavaScriptiga. Nimelt tuli loodud Swifti kood panna esiteks suhtlema Objective-C koodiga ning seejärel alles JavaScriptiga (vt. Joonis 13).

Joonis 13. Swifti (üleval), Objective-C (keskel) ja JavaScripti (all) ühendamine

React Native'i poolt pakutav võimalus, teha suur osa programmeerimisest JavaScriptis, on autori hinnangul väga kasulik. See hoiab kokku arenduse aega ning soodustab arendaja efektiivsust, eemaldades vajaduse implementeerida rakenduse äriloogikat topelt mõlemal platvormil. Vaatamata sellele ei piira raamistik võimalust kirjutada soovi korral põliskoodi, tehes loodava rakenduse jõudluselt ja funktsionaalsuselt võrreldavaks puhta põlisrakendusega (ilma abiraamistiketa loodud rakendusega).

5.4 Suhtlus serveriga

Rakenduse algusetappides, vaadete ja loogika loomisel, kasutas autor andmebaasist kopeeritud testandmeid. Kuna andmebaasi näol on tegemist dokumendipõhise MongoDB'ga, oli võimalus sealt eksporditud JSON formaadis andmed salvestada otse koodi JavaScripti objektina. See võimaldas lihtsasti jäljendada tulevikus serverist tulevaid reaalseid andmeid.

Serverist reaalsete andmete pärimise realiseeris autor kasutades WebSocket protokolli ning juba olemasolevat serveriga suhtlemise liidest, mis oli loodud varasemalt arendades sama rakenduse Ionic raamistiku versiooni. Sellest tulenevalt puudus autoril vajadus tegeleda igasuguse serveri ja andmebaasi poolse arendamisega, tuli üksnes integreerida WebSocketit implementeeriv Socket.IO programmiliides.

5.5 Rakenduse jooksutamine ja testimine

Rakenduse jooksutamine toimus peamiselt Xcode'i arenduskeskkonna juurde kuuluvates iOS'i simulaatorites, kasutades selle poolt pakutavaid 4", 4,7" ja 9,7" virtuaalseid seadmeid. Simulaatoritele lisaks testis autor loodud rakendust ka füüsilisel iOS seadmel – iPhone 6. Füüsilise seadme peal testimiseks tuli antud seade ühendada samasse WiFi võrku kus paiknes arendusserverit jooksutav arvuti. Sellisel juhul oli võimalik kasutada samasid silumisfunktsioone, mis toimisid simulaatorite peal, nagu rakenduse jooksev uuendamine (*Live Reload*) ning Chrome'i arendaja tööriistad.

Arvestades, et arendatav rakendus peab tulevikus töötama ka Androidi platvormil, testis autor mõnda selle vaadet (peamiselt nimekirjavaade ja detailivaade) Androidi emulaatoril. Emulaatorina kasutas autor Android Studio poolt pakutavat virtuaalset seadet – LG Nexus 5.

Rakendust Androidi platvormil testides tunnistas autor asjaolu, et suurt osa loodud koodist on võimalik taaskasutada. Peamised komponendid mida ei olnud võimalik taaskasutada olid iOS'i platvormile spetsiifiliselt loodud moodulid, nagu näiteks Google Maps kaardivaade ning kalendrirakendusega suhtlev moodul. Esines ka vaateid, mida oli võimalik taaskasutada, kuid mille jaoks pakub React Native Androidi peale sobivamat lahendus. Näiteks pakub raamistik Androidi jaoks spetsiaalset nuppude loomise komponenti, milles sisaldub platvormile omane animatsioon, luues antud platvormil parema kasutuskogemuse.

6 Kokkuvõte

Käesoleva lõputöö eesmärgiks oli tutvuda React Native raamistikuga ning kasutada seda mobiilirakenduse UNIspotter loomisel. Peamiselt toimus arendus iOS'i platvormil. Autor testis rakendust lisaks Androidil veendumaks, et kirjutatud koodi saab kasutada mõlema platvormi juures. Autor kirjeldas arenduse käigus kasutust leidnud erinevaid tööriistu ja tehnoloogiaid ning nendega seonduvaid eeliseid ja probleeme.

Töö tulemusena valmis UNIspotteri rakenduse baasfunktsionaalsus React Native raamistikus. Rakenduse arendamise käigus kasutas autor ära raamistiku poolt pakutavaid lisavõimalusi, millest olulisemateks peab ta põlismoodulite ja -vaadete loomist. Kõige keerulisemaks ülesandeks osutus kaardivaate programmeerimine, mis nõudis ulatuslikku põliskoodi kirjutamist ning selle suhtluse koordineerimist JavaScriptiga, mille käigus puutus autor kokku ka Objective-C ja Swift keeltes programmeerimisega.

Töö järeldusena väidab autor, et React Native raamistiku kasutamine UNIspotteri realiseerimisel õigustas ennast, kuna sellega kaasnes märgatav jõudluse paranemine, väljendudes sujuvamates animatsioonides ning rakenduse üleüldise kasutuskogemuse paranemises. Võrreldes varasema Ionic raamistikus realiseeritud rakendusega, pakub loodud React Native'i rakendus, tänu võimalusele kasutada põliskomponente, lisaks platvormi juurde kuuluvaid spetsiifilisi tunnusjooni ning käitumist.

Rakenduse kasutuskogemuse paranemisele lisaks, hindab autor arendusprotsessi märkimisväärset mugavust. Olles varasemalt kokku puutunud iOS'i ja Androidi platvormidele rakenduste arendamisega, toob autor välja, et React Native raamistik lihtsustab märgatavalt arendaja tööd, luues ühtse arenduskogemuse mõlemal platvormil ning võimaldades taaskasutada juba kirjutatud koodi. Autori hinnangul on otstarbekas jätkata UNIspotteri rakenduse arendamist React Native raamistikus ka tulevikus.

Kasutatud kirjandus

[1] React Native | A framework for building native apps using React. [WWW] https://facebook.github.io/react-native/ (23.04.2016). (React Native raamistiku koduleht)

[2] React.js Conf 2015 Keynote - Introducing React Native. [WWW] https://youtu.be/KVZ-P-ZI6W4 (23.04.2016). (React Native raamistiku avalikustamise salvestis)

[3] Style – React Native. [WWW] https://facebook.github.io/react-native/docs/style.html (23.04.2016). (Kujundamise dokumentatsioon)

[4] JavaScript Environment – React Native. [WWW] https://facebook.github.io/react-native/docs/javascript-environment.html (24.04.2016). (JavaScripti keskkonna dokumentatsioon)

[5] Performance – React Native. [WWW] https://facebook.github.io/react-native/docs/performance.html (24.06.2016). (Jõudluse dokumentatsioon)

[6] Native Modules Android – React Native. [WWW] https://facebook.github.io/react-native/docs/native-modules-android.html (24.04.2016). (Androidi põlismoodulite dokumentatsioon)

[7] Native Modules iOS – React Native. https://facebook.github.io/react-native/docs/native-modules-ios.html (24.04.2016). (iOS'i põlismoodulite dokumentatsioon)

[8] Native UI Components Android – React Native. [WWW] https://facebook.github.io/react-native/docs/native-components-android.html (24.04.2016). (Androidi põlisvaadete dokumentatsioon)

[9] Native UI Components iOS – React Native. [WWW] https://facebook.github.io/react-native/docs/native-components-ios.html (24.04.2016). (iOS'i põlisvaadete dokumentatsioon)

[10] About Objective-C. [WWW] https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html (30.04.2016). (Objective-C tutvustus)

[11] Examining Objective-C | Dr Dobb's. [WWW] http://www.drdobbs.com/examining-objective-c/184402055 (30.04.2016). (Artikkel veebist Objective-C kohta)

[12] The Swift Programming Language (Swift 2.2): About Swift. [WWW] https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/index.html (30.04.2016). (Swifti tutvustus)

[13] Meet WebStorm. [WWW] https://www.jetbrains.com/help/webstorm/2016.1/meet-webstorm.html (30.04.2016). (WebStormi arenduskeskkonna tutvustus)

[14] Xcode - IDE - Apple Develop. [WWW] https://developer.apple.com/xcode/ide/ (30.04.2016). (Xcode'i arenduskeskkonna tutvustus)

[15] Meet Android Studio | Android Studio. [WWW] https://developer.android.com/studio/intro/index.html (14.05.2016). (Android Studio arenduskeskkonna tutvustus)

[16] CocoaPods.org. [WWW] https://cocoapods.org/about (14.05.2016). (CocoaPodsi tutvustav veebileht)

[17] Class List | Google Maps SDK for iOS | Google Developers. [WWW] https://developers.google.com/maps/documentation/ios-sdk/reference/ (14.05.2016). (Google Maps iOS SDK dokumentatsioon)

[18] WebSocket and Socket.IO. [WWW] https://davidwalsh.name/websocket (14.05.2016). (Artikkel veebist WebSocket ja Socket.IO tehnoloogia kohta)

[19] HTML5 WebSocket - A Quantum Leap in Scalability for the Web. [WWW] http://www.websocket.org/quantum.html (14.05.2016). (Artikkel veebist WebSocket tehnoloogia kohta)

[20] Flux | Application Architecture for Building User Interface. [WWW] https://facebook.github.io/flux/docs/overview.html (14.05.2016). (Flux arhitektuuri ülevaade)

[21] Read Me | Redux. [WWW] http://redux.js.org/ (14.05.2016). (Redux arhitektuuri ülevaade)