APLIKACIJE IN INFORMACIJSKI SISTEMI

OSEBNI ASISTENT LIFEY

Mentor: Marko Kastelic Avtor: Tom Šabanov, G 4. A

Ljubljana, april 2017

Zahvala

Zahvaljujem se mentorju Marku Kastelicu za mentorstvo in svetovanje. Zahvaljujem pa se tudi Jaku Mušiču za pomoč pri oblikovanju aplikacije.

Povzetek in ključne besede

Cilj projektne naloge je ustvariti osnovnega osebnega asistenta, ta pomaga uporabniku organizirati svoj vsakdanjik, ki ga bom lahko z lahkoto sam nadgrajeval v prihodnosti. V seminarski nalogi so predstavljene in na kratko opisane vse tehnologije, s katerimi sem se ob načrtovanju aplikacije srečal, ter tehnologije, ki sem jih na koncu izbral za svojo projektno nalogo. Izbral sem: React Native, Redux, Redux Saga in MongoDB. Predstavljena je celotna povezava med temi tehnologijami oziroma pretok podatkov, priprava izvajalnega okolja za React Native, načrt izdelave aplikacije skupaj s podatkovno strukturo aplikacije in predstavitev najpomembnejših elementov in modulov aplikacije, ki sem jih moral spisati za popolno delovanje aplikacije.

Ključne besede:

React Native, Redux, Redux Saga, MongoDB, Osebni asistent, Javascript, Avtohtona aplikacije, Hibridna aplikacija, Spletna aplikacija

Summary and keywords

The project objective is to create a basic personal assistant that helps the user to organize their daily routine, which I could easily upgrade in the future. In the paper are presented and briefly described all technologies, which I met in the planning stage of the application, and technologies I chose. I chose: React Native, Redux, Redux Saga and MongoDB. The paper presents the entire connection between these technologies (or the data flow), the preparation for the execution environment for React Native, the planning of the application with the information structure of it and the presentation of the most important elements and modules of the application, which I wrote myself.

Keywords:

React Native, Redux, Redux Saga, MongoDB, Personal assistant, Javascript, Native application, Hybrid application, Web application

Kazalo vsebine

[1 Uvod 7](#_Toc480230713)

[2 Pregled in izbira tehnologij 8](#_Toc480230714)

[2.1 Spletne aplikacije 8](#_Toc480230715)

[2.1.1 HTML5 8](#_Toc480230716)

[2.1.2 Prednosti 8](#_Toc480230717)

[2.1.3 Slabosti 9](#_Toc480230718)

[2.2 Avtohtone aplikacije 9](#_Toc480230719)

[2.2.1 Prednosti 9](#_Toc480230720)

[2.2.2 Slabosti 9](#_Toc480230721)

[2.3 Hibridne aplikacije 9](#_Toc480230722)

[2.3.1 Prednosti 10](#_Toc480230723)

[2.3.2 Slabosti 10](#_Toc480230724)

[2.4 Odločitev 10](#_Toc480230725)

[2.4.1 Osnovne značilnosti React Native 11](#_Toc480230726)

[2.4.2 Redux in Redux Saga 14](#_Toc480230727)

[2.4.3 MongoDB 16](#_Toc480230728)

[2.4.4 Pretok podatkov 16](#_Toc480230729)

[2.4.5 Priprava izvajalnega okolja 18](#_Toc480230730)

[3 Izdelava aplikacije – Lifey 20](#_Toc480230731)

[3.1 Načrt aplikacije 20](#_Toc480230732)

[3.1.1 Podatki aplikacije 21](#_Toc480230733)

[3.2 Moduli in ključni deli aplikacije 22](#_Toc480230734)

[3.2.1 Index.js (Index.android.js) 22](#_Toc480230735)

[3.2.2 App 23](#_Toc480230736)

[3.2.3 Main 24](#_Toc480230737)

[3.2.4 Event in Reminder 28](#_Toc480230738)

[3.2.5 Day in PreviewDay 30](#_Toc480230739)

[3.2.6 API.js, Utils.js in Avdio modul 31](#_Toc480230740)

[3.3 Končni izdelek 33](#_Toc480230741)

[4 Sklepne ugotovitve 37](#_Toc480230742)

[5 Literatura in viri 38](#_Toc480230743)

[6 Priloge 39](#_Toc480230744)

[7 Izjava o avtorstvu 40](#_Toc480230745)

Kazalo slik

[Slika 1: Primer deklarativnih komponent 12](#_Toc480230746)

[Slika 2: Primer uporabe komponente spisane v avtohtoni kodi 13](#_Toc480230747)

[Slika 3: Primer uporabe state 14](#_Toc480230748)

[Slika 4:Primer uporabe props 14](#_Toc480230749)

[Slika 5: Primer ustvarjalca akcije 15](#_Toc480230750)

[Slika 6: Primer zmanjševalca 16](#_Toc480230751)

[Slika 7: Primer trgovine 16](#_Toc480230752)

[Slika 8: Primer generator funkcije 17](#_Toc480230753)

[Slika 9: Diagram pretoka podatkov skozi aplikacijo 18](#_Toc480230754)

[Slika 10: Inštalacija Node.s ter python2 z Chocolatey 19](#_Toc480230755)

[Slika 11: Inštalacija znakovnega uporabniškega vmesnika za React Native 19](#_Toc480230756)

[Slika 12: Potrebne stvari za React Native 20](#_Toc480230757)

[Slika 13: Ustvaritev spremenljivke ANDROID\_HOME 20](#_Toc480230758)

[Slika 14: Ustvaritev projekta in zagon strežnika 21](#_Toc480230759)

[Slika 15: Začetni ekran aplikacije 21](#_Toc480230760)

[Slika 16: Stanje aplikacije 23](#_Toc480230761)

[Slika 17: Registracija aplikacije 24](#_Toc480230762)

[Slika 18: renderScene metoda 24](#_Toc480230763)

[Slika 19: Diagram modula App 25](#_Toc480230764)

[Slika 20: Diagram modula Main 25](#_Toc480230765)

[Slika 21: Začetno stanje zmanjševalca 26](#_Toc480230766)

[Slika 22: Komponenta Swiper 27](#_Toc480230767)

[Slika 23: FloatingActionButton komponenta 27](#_Toc480230768)

[Slika 24: PopupDialog 28](#_Toc480230769)

[Slika 25: AndroidBackButton komponenta 28](#_Toc480230770)

[Slika 26: Potrebni podatki in metode za modul Event/Reminder 29](#_Toc480230771)

[Slika 27: Začetno stanje 30](#_Toc480230772)

[Slika 28: Saga module Event 30](#_Toc480230773)

[Slika 29: Dan, dogodki in opomniki 31](#_Toc480230774)

[Slika 30: Diagram modula Day 32](#_Toc480230775)

[Slika 31: Objekti v utils.js 32](#_Toc480230776)

[Slika 32: Primer dveh metod v modulu Audio 33](#_Toc480230777)

[Slika 33: Primer uporabe MediaRecorder 34](#_Toc480230778)

[Slika 34: Odprtje aplikacije 34](#_Toc480230779)

[Slika 35: Menu dogodka/opomnika 35](#_Toc480230780)

[Slika 36: Primer zvočnih posnetkov 35](#_Toc480230781)

[Slika 37: Koledar in predogled dneva 36](#_Toc480230782)

[Slika 38: Dodajanje dogodka 36](#_Toc480230783)

[Slika 39: Primer obvestila 37](#_Toc480230784)

# Uvod

Danes se na trgu pojavljajo številne aplikacije s katerimi si lahko uporabniki poskušamo urediti življenje tako, da imamo vse podatke o bodočih dogodkih ali o opomnikih priročno v eni aplikaciji. Takšnemu tipu aplikacije pravimo osebni asistent, in je namenjena vsem, ki si poskušajo bolje organizirati svoje življenje.

Noben osebni asistent, ki sem ga do sedaj preizkusil, ni ustrezal mojim zahtevam. Želim imeti aplikacijo, ki mi bo omogočala vnašanje bodočih dogodkov, opomnikov, šolskega urnika in ocene šolskih predmetov ter imela pregled mojega vsakdanjega dneva. Zato sem se pri projektni nalogi odločil za izdelavo aplikacije, ki bi mi prav to omogočala. Ker vem, da so aplikacije takega tipa lahko zelo zahtevne za izdelavo, sem se odločil, da bo moj glavni cilj projektne naloge, da ustvarim osnovnega osebnega asistenta, ki ga bom lahko skozi čas sam nadgrajeval, da bo ustrezal mojim potrebam.

Moj osebni asistent omogoča vnašanje opomnikov in dogodkov z zmožnostjo obveščanja ob danem času teh dogodkov ali opomnikov, hkrati pa bo tudi omogočal zaenkrat le dnevni pregled opomnikov in dogodkov. Za dani osebni asistent sem potreboval izdelati podroben načrt, strukturo same aplikacije ter definirati vsak modul aplikacije, hkrati pa tudi definirati podatkovno strukturo, da bom lahko čim lažje dodajal bodoče značilnosti aplikacije.

Izbira tehnologij, opredelitev izvajalnega okolja so del moje naloge in kot taka opisana v naslednjih izločenih poglavjih.

# Pregled in izbira tehnologij

Nove tehnologije, ki naj bi omogočile preprosto ustvarjanje aplikacij, se pojavljajo vsak teden. Vsaka ima svoje prednosti in slabosti. Nekatere so mlade, prepočasne, vsebujejo številne hrošče, a so načeloma dokaj enostavne za uporabljati, z njimi pa se celo da veliko hitreje ustvariti aplikacijo, nekatere pa so stare, a zanesljive z ogromno skupnostjo.

Načeloma poznamo tri vrste tehnologij oziroma tri vrste mobilnih aplikacij:

## Spletne aplikacije

Spletne aplikacije pravzaprav niso »prave« aplikacije, temveč so spletne strani, ki le izgledajo kot avtohtone (ang. native) aplikacije, a implementirane na čisto drugačen način. Poganja jih brskalnik in so v večini primerov spisane kar v HTML5. Uporabniki dostopajo do takih aplikacij enako kot bi dostopali do spletne strani: usmerijo se na nek URL naslov in imajo potem opcijo »inštalacije« aplikacije na domači zaslon z ustvaritvijo zaznamka. Take vrste aplikacije so postale zares popularne ko je izšel HTML5.

### HTML5

Oznaka HTML označuje Hypertext Markup Language, ki je osnova za celoten splet. HTML koda, ki je sestavljena iz niza oznak, brskalniku pove, kako naj določeno besedilo, sliko ali video, prikaže. HTML5 je peta in trenutna verzija HTML standarda.

HTML5 je bil objavljen oktobra 2014 z namenom izboljšati jezik s podporo za najnovejše multimedije z lahko berljivo in konsistentno kodo, ki jo razumejo ne le ljudje, temveč tudi brskalniki.

Dve glavni lastnosti HTML5 sta izboljšana integracija z grafiko in opcija geolociranja. V sklopu grafike sta dva dodatka Canvas (platna) in SVL (Scalable Vector Graphics), ki omogočata predvajanje grafike neposredno iz kode. Do sedaj je bilo za predvajanje takšnih funkcij potrebno uporabiti Adobe Flash predvajalnik. Geolokacija pa omogoča, da sistem že samostojno prepozna lokacijo brskalnika.

HTML5 razširja, izboljšuje ter racionalizira sintakso, ki je na voljo za dokumente, ter uvaja aplikacijske programske vmesnike (API) za kompleksne spletne aplikacije.

### Prednosti

Največja prednost spletnih aplikacij je njihova fleksibilnost. Na katerikoli platformi, ali na pametnem telefonu ali na tabličnem računalniku, lahko kodo le prekopiramo in z malo ali pa celo brez kakršnihkoli prilagoditev za določeni operacijski sistem ustvarimo aplikacijo za drugo platformo, zato je njihov celotni proces izdelave veliko hitrejši kot pa pri avtohtonih aplikacijah, vendar pa so vseeno nekatere funkcije avtohtonih aplikacij vseeno nedosegljive, kot so bogat uporabniški vmesnik in izkušnja ( ang. UI/UX ).

### Slabosti

Definitivno so pri spletnih aplikacijah prisotne ogromne omejitve. Uporabnikova izkušnja je veliko slabša zaradi počasnega delovanja aplikacije kot pa pri avtohtonih aplikacijah, so nekonsistentne, niso optimizirane za naprave, trgovina jih ne more odkriti, so veliko manj stabilne, prav tako pa ne omogočajo toliko funkcij brez povezave interneta kot pa avtohtone aplikacije.

## Avtohtone aplikacije

Avtohtone aplikacije živijo na telefonu in so dostopne preko ikon na domačem zaslonu. Te aplikacije se inštalirajo preko trgovine kot je Google Play ali pa App Store. Avtohtone aplikacije so razvite za točno specifično platformo, in lahko zato v celoti izkoristijo vse funkcije naprave. Lahko uporabljajo kamero, GPS, kompas, seznam stikov, sistem obveščanja in vse ostale lastnosti telefona.

### Prednosti

Njihova največja prednost je zmožnost dostopa vseh funkcij posamezne platforme ter seveda boljša učinkovitost same aplikacije. Prav tako pa so vse aplikacije, ki jih z lahkoto lahko dobimo preko trgovine, potrjene s strani trgovine, zato smo lahko uporabniki popolnoma prepričani, da je aplikacija stoodstotno varna.

### Slabosti

Avtohtone aplikacije so v večini primerov dražje za izdelati. To še posebej velja, če bi radi, da je aplikacija kompatibilna z različnimi mobilnimi napravami in platformami. Prav tako je cena vzdrževanja in posodabljanja veliko večja za avtohtone aplikacije, še posebej, če aplikacija podpira več kot eno mobilno platformo. Njihova največja slabost je pa relativno dolg proces izdelave malo bolj kompleksnih aplikacij,

## Hibridne aplikacije

Hibridne aplikacije so delno avtohtone in delno spletne aplikacije. Inštalira se jih preko trgovine, tako kot avtohtone aplikacije, a zgrajene so s kombinacijo spletnih tehnologij kot so HTML, CSS in Javascript. Hibridne aplikacije so pravzaprav gostovane znotraj avtohtone aplikacije, ki uporablja WebView mobilne platforme (na WebView lahko pomislite kot okno brskalnika v celozaslonskem načinu). To jim omogoča dostop do vseh zmogljivosti naprave, kot so raznorazni merilniki, fotoaparat, stiki in še več. To so sposobnosti, do katerih znotraj mobilnih brskalnikov ne moremo dostopati.

Danes, večina mobilnih aplikacij izkorišča Apache Cordova, platformo ki ponuja konsistenten JavaScript API za dostop sposobnosti telefona skozi uporabo vtičnikov, ki so zgrajeni z avtohtono kodo.

### Prednosti

Hibridne aplikacije zagotavljajo razvijalcem ponovno uporabo svojih obstoječih znanj razvoja spletnih strani. Predvsem so hibridne aplikacije privlačne za podjetja. Zakaj zaposliti več različnih razvijalcev za vsako platformo posebej, če pa lahko najamemo le enega in tako ciljamo na vse platforme preko HTML,CSS in JavaScript tehnologij? Podjetje ima tako lahko enostavno eno ekipo, ki lahko cilja na več različnih platform z enim setom tehnologij.

### Slabosti

Čeprav omogočajo hibridne aplikacije razvijalcem ciljati na več kot eno platformo, ima vsaka platforma neke izjeme ko pridemo do WebView. Mogoče želi celo razvijalec delati z nekaterimi unikatnimi sposobnostmi platforme. V takih primerih mora biti izkoriščena kombinacija vtičnikov in specifične kode za platformo da izkoristimo vso njeno zmogljivost.

Prav tako so nekatere funkcionalnosti, kot so zajem medijev, še vedno nepodprte v raznih mobilnih brskalnikih, ali pa so nekateri vtičniki zastareli in nezanesljivi, zato se zaradi takih omejitev razvijalci obrnejo na avtohtone aplikacije.

## Odločitev

Ker so lahko hibridne aplikacije ter spletne aplikacije izredno počasne oziroma ne zagotavljajo vseh zmožnosti kot jih ponujajo avtohtone aplikacije, sem se odločil da bom ustvaril avtohtono aplikacijo s pomočjo tehnologije React Native, ki jo Facebook razvija že od leta 2015.

React Native nam omogoča grajenje mobilnih aplikacij z uporabo Javascripta. Uporablja pa enak princip grajenja aplikacije kot React, Javascript knjižnice, kar nam omogoča sestavljanje bogatega mobilnega uporabniškega vmesnika iz deklarativnih komponent, ki naredijo našo kodo veliko bolj predvidljivo in lažjo za odkrivanje napak.

Poleg React Native sem moral izbrati še bazo, v kateri bom hranil vse ustvarjene podatke v aplikaciji, ter arhitekturo toka podatkov in načina hranjenja stanja aplikacije. Zaradi same hitrosti vseh poizvedb, sem izbral za bazo MongoDB, čeprav je baza SQL veliko bolj podprta, Redux Sago in Redux pa kot način hranjenje stanja aplikacije ter kot arhitekturo toka podatkov.

Slika : Primer deklarativnih komponent

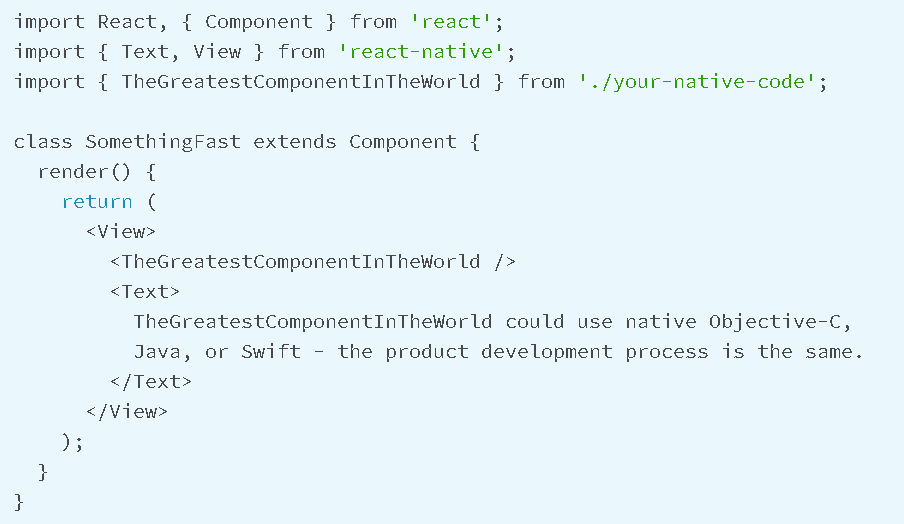


### Osnovne značilnosti React Native

Kljub temu, da lahko celotno aplikacijo zgradimo le z Javascriptom, z React Nativom ne delamo mobilne spletne aplikacije ali pa hibridne aplikacije. Gradimo pravo mobilno aplikacijo ki ni prav nič drugačna od aplikacije zgrajene z Objective-C ali pa Javo in se v zmogljivosti lahko s to popolnoma primerjajo, saj se React Native prevede direktno v avtohtono kodo platforme iOS ali pa Android. React Native uporablja enake temeljne gradnike kot navadne iOS in Android aplikacije. Te gradnike le sestavimo skupaj z Javascript jezikom in React knjižnico.

React Native nam omogoča veliko hitrejše grajenje aplikacij. Namesto, da moramo aplikacijo ob vsaki malenkostni spremembi, kot je recimo sprememba postavitve aplikacije, vsakič znova prevesti, lahko osvežimo aplikacijo instantno ob vsaki spremembi kode (kar je izjemen plus za tiste, ki mogoče uporabljajo starejše računalnike, ki niso dovolj zmogljivi, da bi ustvarjali aplikacijo s standardni orodji za izdelavo mobilnih aplikacij kot je npr. Android Studio), kar je zelo podobno izdelavi spletnih strani. Omogoča nam pa tudi, da se lahko preprosto spustimo v avtohtono kodo, če želimo kakšen aspekt aplikacije optimizirati, zato je zelo lahko narediti aplikacijo delno v React Native, delno pa kar direktno v avtohtoni kodi.

Slika : Primer uporabe komponente spisane v avtohtoni kodi



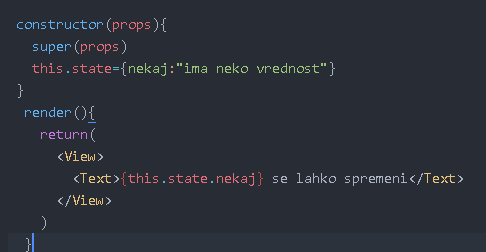
Še ena fantastična značilnost React Native je, da se ista generična koda lahko popolnoma uporabi in pri iOS in Android verziji aplikacije. React Native omogoča zaznavanje platforme v kodi, zato je izjemno preprosto spisati komponente, ki so enake v funkcionalnosti, a so adaptirane za svoj operacijski sistem. To pomeni da lahko uporabimo enake Javascript datoteke za Android in iOS verzijo aplikacije, z občasnimi pogojnimi stavki znotraj datotek.

Obstajata tudi dva tipa podatkov, ki kontrolirata komponento: state in props.

#### State

Za podatke, ki se bodo spremenili, moramo uporabiti state oziroma stanje. Stanje moramo inicializirati v konstruktorju komponente in potem klicati metodo setState, ko ga želimo spremeniti, in vsakič ko se stanje spremeni, se tudi celotna komponenta ponovno upodobi, zato moramo paziti, kje in kdaj bomo uporabili state.

V večji aplikaciji se state uporablja le za podatke, ki nimajo nobenega vpliva na preostali del aplikacije, saj se večinoma uporablja vsebnik stanja kot je Redux, da kontroliramo tok podatkov

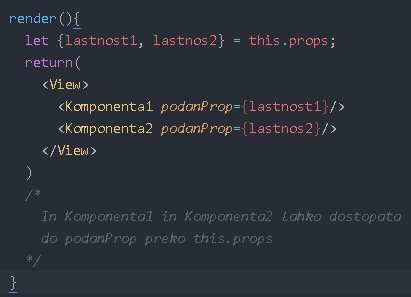
. 

Slika : Primer uporabe state

#### Props

Večina komponent v React Native se da prilagoditi ob njihovi kreaciji z različnimi parametri. Te parametri se imenujejo props, in dostopamo lahko do njih v funkciji render, kjer imamo vse gradnike aplikacije. Tako lahko uporabimo eno in isto komponento večkrat, le da podamo vsakič različne lastnosti. Props nastavi nadrejena komponenta, hkrati pa se jih ne da spreminjati v samem življenjskem ciklu komponente

Slika :Primer uporabe props



#### Življenjski cikel komponente

Vsaka komponenta ima več metod pri svojem življenjskem ciklu, ki jih lahko zaženemo ob določenih časih v procesu. Metode z »will« se kličejo preden se nekaj zgodi, in metode z »did« se kličejo po nekem dogodku.

Pri namestitvi komponente imamo naslednje štiri metode:

* constructor()
* componentWillMount()
* render()
* componentDidMount()

Pri posodobitvi pa naslednjih pet metod:

* componentWillReceiveProps()
* shouldComponentUpdate()
* componentWillUpdate()
* render()
* componentDidUpdate()

### Redux in Redux Saga

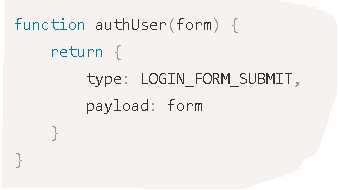
Redux je arhitektura toka podatkov aplikacije. Ta se večinoma uporablja za upravljanje stanja celotne aplikacije v enem nespremenljivem stanju objekta, ki se ga ne da direktno spremeniti. Ko se nek podatek spremeni, se ustvari kar nov objekt ( z uporabo akcij in zmanjševalcev).

Redux ima tri sestavne dele, ki morajo biti prisotne v vsaki aplikaciji, ki ga uporablja.

#### Akcije

Akcije podajo trgovini podatke o nekem dogodku, ki ga je sprožil sistem ali pa uporabnik.. Vsak dogodek mora imeti lastnost »type« oziroma tip dogodka (recimo: »PRIJAVA\_UPORABNIKA«), zato da trgovina ve, kako bo to akcijo procesirala. Akcije ustvarjamo kar z ustvarjalci akcij, ki so le navadne funkcije, ki vrnejo akcijo, odpošiljamo jih pa trgovini z metodo dispatch, ki ji podamo ustvarjeno akcijo.

Slika : Primer ustvarjalca akcije



#### Zmanjševalci

Zmanjševalec je funkcija, ki sprejme trenutno stanje aplikacije ter akcijo in nato vrne novo posodobljeno stanje aplikacije, ki je odvisno od tipa dogodka, zato uporabljamo vedno pri zmanjševalcih tudi pogojne stavke. Zmanjševalci opravljajo večino dela pri Redux aplikacijah.

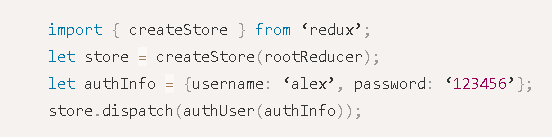
Slika : Primer zmanjševalca



#### Trgovina

Trgovina je objekt, ki drži celotno stanje aplikacije in hkrati tudi zagotavlja nekaj metod, s katerimi si lahko pomagamo dostopati do stanja, odpošiljati akcije ter registrirati poslušalce akcij. Celotno stanje aplikacije predstavlja ena sama trgovina.

Slika : Primer trgovine



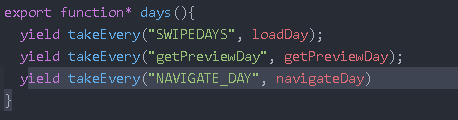
#### Redux Saga

Redux Saga je knjižnica, ki olajša stranske učinke ( kot so razne asinhrone poizvedbe v bazo) aplikacije pri odpošiljanju akcij v trgovino.

Saga deluje kot ločena nit v naši aplikaciji, ki je odgovorna samo za stranske učinke. Ta nit se lahko zažene, pavzira ali pa prekliče iz same aplikacije s pomočjo normalnih redux akcij. Ima tudi polni dostop do stanja aplikacije ter lahko tudi odpošilja redux akcije.

Uporablja generator funkcije, značilnost sodobnega Javascript jezika, za lahko berljivost asinhronega toka operacij, saj izgleda funkcija kot navadna sinhrona Javascript koda.

Slika : Primer generator funkcije



### MongoDB

MongoDB je dokumentno orientiran odportokodni sistem za upravljanje podatkovne baze, narejen pa je v programskem jeziku C++. Dokumenti v MongoDB niso vezani na določeno shemo, zato lahko podatki v določeni zbirki (ang. collection) vsebujejo različne atribute. Zaradi tega ima MongoDB dobro fleksibilnost - informacije o določenem izdelku je možno shraniti v eno zbirko, medtem ko je pri relacijskih podatkovnih bazah pogosta praksa, da so podatki shranjeni v več različnih tabelah.

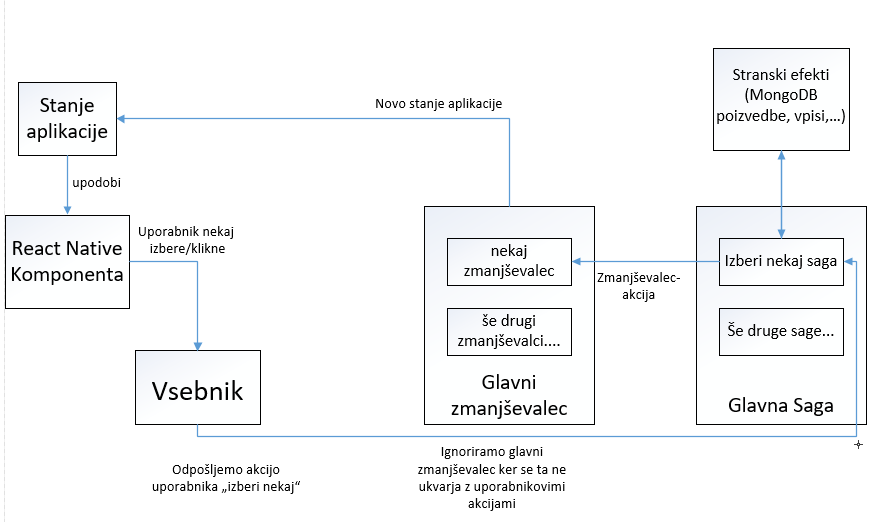
MongoDB shranjuje dokumente v zbirke, ki ne zahtevajo posebne sheme, zato se lahko dokumenti v zbirki med seboj zelo razlikujejo. To je koristno predvsem v začetnih fazah izdelave aplikacij, ko se sestava dokumentov pogosto spreminja.

Dokumenti se vnašajo v MongoDB v obliki ključ : vrednost. Dokumenti v MongoDB so tipa BSON ( ang. Binary JSON), torej so podobni JSON (ang. JavaScript Object Notation) objektom v binarni obliki, kar pomeni, da so namenjeni za računalniško branje. Vrednost ključev lahko vsebuje druge dokumente, tabele in tabele dokumentov.

### Pretok podatkov

Sedaj, ko imam izbrane vse elemente oziroma tehnologije aplikacije, lahko povežem React Native, MongoDB ter Redux Saga. Pretok podatkov skozi aplikacijo je predstavljen v sledečem diagramu:

Slika : Diagram pretoka podatkov skozi aplikacijo



Vse redux akcije sem razdelil v dve različni kategoriji:

1. Sistemske in uporabniške akcije
2. Akcije zmanjševalca

#### Sistemske in uporabniške akcije

Take akcije bodo React Native vsebniki odposlali glede na nek dogodek v aplikaciji (npr. uporabnik nekaj vnese v neko polje, komponenta se želi posodobiti itd…). Sistemskih in uporabniških akcij ne bo nikoli sam zmanjševalec obdelal, ampak bodo delovale kot vnos za sage.

#### Akcije zmanjševalca

Vse akcije zmanjševalca odpošlje saga. Samo te akcije bodo dejansko spremenile stanje aplikacije, hkrati pa ima vsaka akcija zmanjševalca samo eno odgovornost in zato bo točno eden zmanjševalec to akcijo procesiral oziroma obdelal.

#### Posledice takšnega pretoka podatkov

S takšno strukturo, kjer imamo akcije razdeljene v dva tipa, bomo imeli zelo preprost pretok podatkov. Pred to strukturo se ni dalo vedeti, ali naj odpošljemo dve različni akciji (recimo »poizvediPodatke« in »spremeniBarvo«), ali naj odpošljem samo eno akcijo, ki jo bosta procesirala dva zmanjševalca. Obe imata nekaj slabosti. V prvem primeru moramo odposlati dve akciji z logiko, ki se bo lahko čez čas spremenila, v drugem primeru pa enostavno ne moremo vedeti kaj se bo točno spremenilo v stanju aplikacije.

S takšnim pretokom podatkov odpošljemo samo eno uporabniško akcijo v vsebniku komponente (recimo »izberiProdukt«) in nobeno ali pa več akcij zmanjševalca v naši sagi, ki bodo spremenila stanje aplikacije.

Za vsako uporabniško akcijo moramo ustvariti eno sago. Tako lahko takoj vidi vsak programer, kateri dogodki se lahko zgodijo v aplikaciji in kakšne bodo njihove posledice.

### Priprava izvajalnega okolja

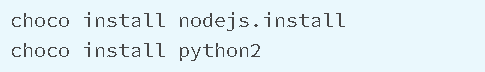
Za vsak operacijski sistem je priprava okolja za razvijanje React Native aplikacije drugačen. Ker sem razvijal aplikacijo za sistem Android na operacijskem sistemu Windows, bom opisoval pripravo okolja za ta dva sistema.

#### Inštalacija odvisnosti

Potrebovali bomo Node.js, znakovni uporabniški vmesnik za React Native ter Android Studio

Naprej potrebujemo inštalirati Node.js ter python2. Ta dva jezika lahko vsakega posebej inštaliramo tako, da gremo na njune spletne strani in ju ročno sami inštaliramo, ali pa inštaliramo Chocolatey, popularen manager za pakete za Windows, ter potem preko tega inštaliramo oba, Node.js ter python2.

Slika : Inštalacija Node.s ter python2 z Chocolatey



Skupaj z Node.js pa pride npm – Node Package Manager ( spet še en manager za pakete), ki nam omogoča inštalacijo znakovnega uporabniškega vmesnika za React Native (ang. Command line interface).

Slika : Inštalacija znakovnega uporabniškega vmesnika za React Native

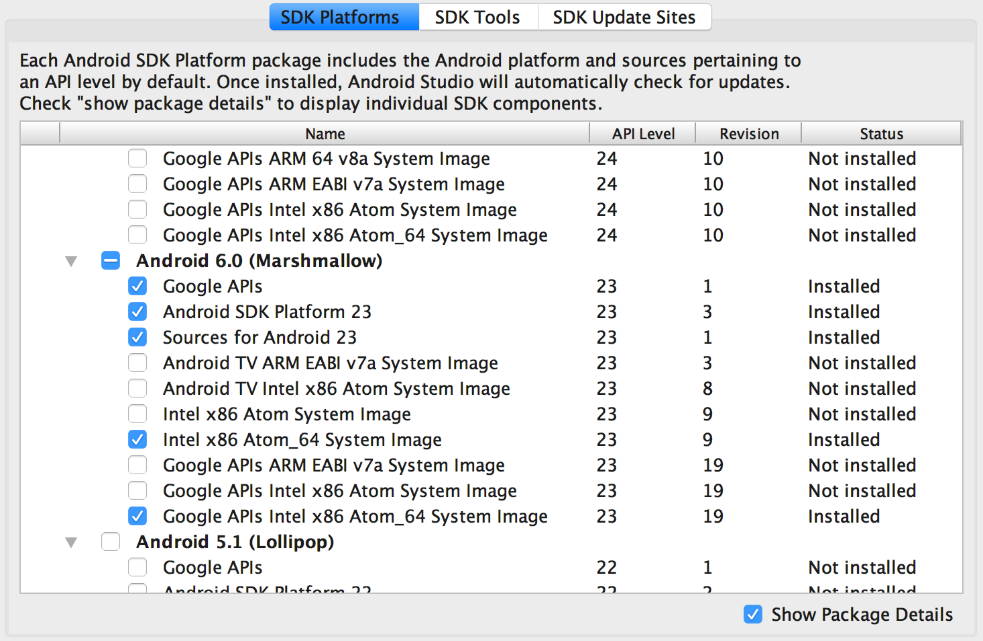
4

Naslednja stvar ki jo potrebujemo nastaviti, je naše Android okolje. Naprej seveda prenesemo in inštaliramo Android Studio, ki nam zagotavlja Android SDK in AVD (emulator), ki so potrebni za zaganjanje in testiranje naše React Native aplikacije.

Android Studio inštalira privzeto najnovejši Android SDK – Software Development Kit, a React Native zahteva Android 6.0 (Marshmallow) SDK. Da ga inštaliramo, moramo enostavno v SDK managerju inštalirati naslednje stvari:

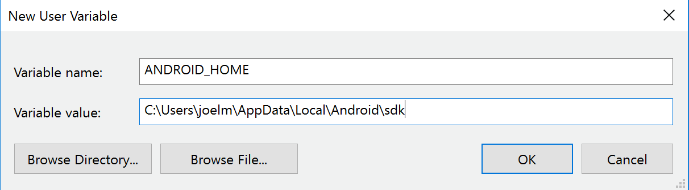
* Google API-je
* Android SDK Platform 23
* Intel x86 Atom\_64 System Image
* Google API-je x86 Atom\_64 System Image

Slika : Potrebne stvari za React Native



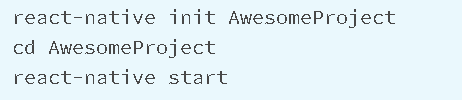
Znakovni uporabniški vmesnik za React Native prav tako potrebuje vzpostavitev ANDROID\_HOME spremenljivke okolja. Po ustvaritvi te spremenljivko moramo samo resetirati ukazno vrstico (ang. Command Prompt).

Slika : Ustvaritev spremenljivke ANDROID\_HOME



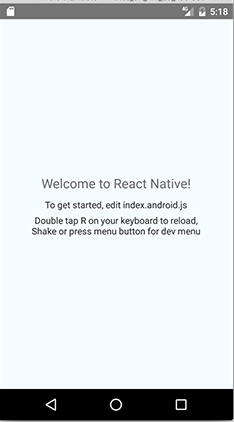
React Native aplikacijo se lahko tudi zaganja preko emulatorja, a zaradi zastarele opreme tega sploh poskušal nisem, zato sem jo zaganjal kar direktno na svoj Android telefon. Da zaženemo React Native aplikacijo na pravi napravi jo moramo povezati z računalnikom preko USB kabla ter omogočiti USB odpravljanje napak na telefonu. Nato moramo v ukazni vrstici izvršiti naslednje ukaze:

Slika : Ustvaritev projekta in zagon strežnika



*react-native init ime\_projekta* le ustvari oziroma inicializira projekt, *react-native start* pa zažene strežnik, na katerem bodo pozneje tudi vidne vse napake v aplikaciji. Zadnji ukaz, ki sledi prejšnjim trem, pa *je react-native run-android*, ki bo naložil aplikacijo na naš telefon in jo zagnal. Če je vse pravilno nastavljeno, bomo našo aplikacijo na telefonih videli takole:

Slika : Začetni ekran aplikacije



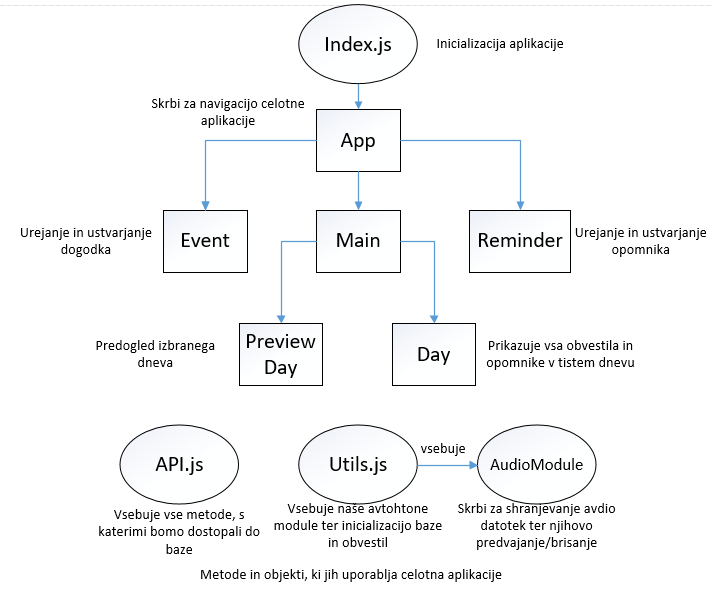
# Izdelava aplikacije – Lifey

Aplikacija Lifey bo omogočala vnašanje opomnikov in dogodkov, vsakemu opomniku in dogodku pa bomo lahko tudi dodali avdio zapiske, hkrati pa bo se bodo prikazovala tudi obvestila o dogodkih in opomnikih tistega dne. Da bo aplikacija delovala tako tekoče kot ostale avtohtone aplikacije, narejene z recimo Javo ali Swift jezikom, moramo biti zaradi arhitekture Redux zelo previdni kako bomo strukturirali aplikacijo, saj želimo, da se aplikacija čim manjkrat ponovno upodablja.

## Načrt aplikacije

Aplikacijo je potrebno razdeliti na module. Vsak modul, ki bo povezan s trgovino, bo imel svoj zmanjševalec ter svojo sago. Ker nas skrbi prekomerno ponovno upodabljanje aplikacije bomo povezali komponente, ki so čim nižji v drevesu, na trgovino, saj je to najbolj efektivna metoda, da se bo po vsaki spremembi posodobilo le tisto, kar se potrebuje in nič drugega. Prav tako pa bo celotna aplikacija uporabljala veliko podobnih oziroma celo istih objektov in metod za svojo delovanje, zato je načrt aplikacije sledeč:

Slika 7: DIAGRAM APLIKACIJE

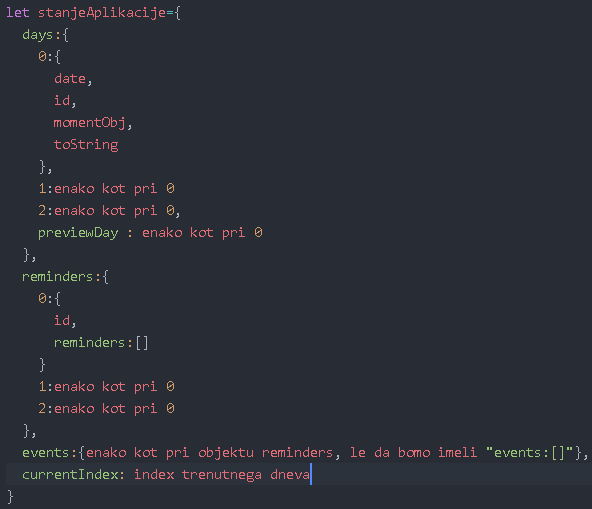


Moduli, ki bodo povezani na trgovino, so : Main,Event, Reminder in Day. Ker so Event, Reminder in Day na dnu drevesa, ne bodo pri ponovnih upodabljanjih povzročali nobenih problemov pri učinkovitosti aplikacije. Problem lahko le nastane pri modulu Main, ker vsebuje tudi module Preview Day in Day, a problem se da rešiti zelo enostavne z uporabo metode componentWillUpdate v življenjskem ciklu obeh teh komponent. Ta metoda mora vrniti neko logično vrednost, zato lahko enostavno le primerjamo prejšnje podane podatke (props) komponente s tistimi, ki ji bodo dane (nov props). Modul Main je potrebno zato sestaviti tako, da ko se bo ponovno upodobil, ne bo spreminjal lastnosti props, ki jo bo komponentam podajal.

### Podatki aplikacije

Potrebujem tudi definirati vse podatke, ki se bodo pretakali skozi aplikacijo ter v bazo in iz nje. Stanje aplikacije bo izgledalo tako kot kaže naslednja slika:

Slika : Stanje aplikacije



Stanje vsebuje objekte days,reminders, events in currentIndex.

Objekt Days vsebuje še štiri objekte, ki vsebujejo vse informacije o dnevu na tem indeksu. Dneve, poimenovane od 0-2, bo potrebovala vsaka Day komponenta oziroma Day komponenta bo potrebovala tisti dan z indeksom, ki je bil komponenti dan. Vsak dan ima še lastnosti date, id, momentObj (neka variacija Date objekta), in pa toString. PreviewDay lastnost bo potrebovala le komponenta PreviewDay.

Objekta reminders in events sta skorajda identična. Vsebujeta objekte, poimenovane od 0-2, ki imajo lastnosti id ter tabelo objektov opomnikov/dogodkov. Te objekti imajo sledeče lasnosti : about,date,time (ali »duration« pri dogodku) in pa recordings, ki je enostavna tabela zapisov zvočnih datotek. Ta dva objekta bodo potrebovali trije moduli: Day, Event in Reminder.

Seveda bodo podatki bili oblikovani drugače v bazi, kot pa v stanju aplikacije. Ker imamo bazo MongoDB, ki je dokumentno orientirana baza, bo vsak dan svoj dokument s ključem \_id, ki bo kar datum dneva.

Vsak tak dan bo imel le dve tabeli : events in reminders, ki bodo imeli znotraj objekte dogodkov in opomnikov, enako kot pri stanju aplikacije.

## Moduli in ključni deli aplikacije

### Index.js (Index.android.js)

Index.js datoteka ne deluje kot modul, temveč kot vstopna točka v aplikacijo. Njena naloga je inicializacija same aplikacije ter trgovine, hkrati pa poskrbi, da se naložijo v aplikacijo današnji, včerajšnji in jutrišnji dan s klicem metode initializeDays v modulu API.

Naprej se nastavi redux trgovina. Importirati moramo nekaj metod za samo ustvaritev redux trgovine ter uporabo vmesne opreme, Redux Saga, na trgovini in pa glavni zmanjševalec, ki ga sestavljajo trije drugi zmanjševalci (za module Day,Event in Reminder). Ta vmesna oprema bo, po inicializaciji trgovine, zagnala glavno sago. Naša glavna saga vsebuje tri različne sage, za Day, Event ter Reminder modul.

Nato pa se še importira glavni modul, App, ter se nato registrira aplikacijo s komponento, ki jo sestavljata trgovina in modul App.

Slika : Registracija aplikacije

registracija aplikacije

### App

App modul skrbi za celotno navigacijo aplikacije. To stori enostavno z React Native komponento Navigator. Ta obravnava vse tranzicije scen v naši aplikaciji. Implementiran je v Javascript jeziku, na voljo pa je na iOS in Android sistemu (a če bi delal aplikacijo za iOS, bi uporabil NavigatorIOS, ker je implementiran avtohtono).

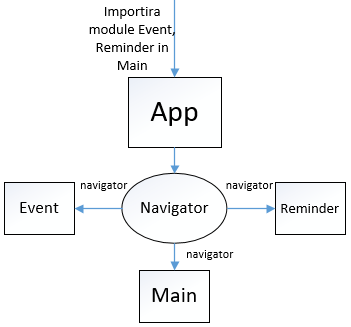
Za nastavitev Navigator komponente mu potrebujemo podati eden ali več objektov oziroma poti. Prav tako mu tudi zagotovimo renderScene funkcijo, ki bo upodobila vsako našo sceno(Main, Event in Reminder, te moramo tudi seveda importirati) za vsak objekt poti. Vsaka pot mora vsebovati tudi »title« lastnost oziroma ime poti, z podano lastnostjo »initialRoute« Navigator komponenti pa povemo, katera bo prva scena, ki se nam bo prikazala ob odprtju aplikacije.

Slika : renderScene metoda



Pri renderScene metodi pa moramo tudi vsem scenam oziroma potem podati objekt »navigator«, s katerim se bodo lahko vračali nazaj, ali pa se usmerili na katerokoli drugo pot (poleg navigator objekta poda tudi vse podatke, ki bi jih modul želel mogoče imeti).

Slika : Diagram modula App

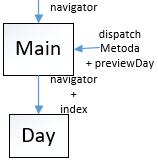


### Main

Modul Main je prva povezana komponenta na trgovino. Poleg objekta navigator, ki ga poda vsem trem dnevom (poda jim tudi unikaten index), dobi tudi možnost odpošiljanja treh različnih akcij v trgovino (oziroma v sago) ter eno vrednost, ki ji modul sledi za spremembe, poimenovana previewDay, ki ga nato tudi podamo modulu previewDay.

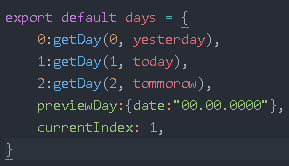
Modul vsebuje komponento Swiper, v kateri s moduli Day, PopupDialog, CalendarPicker in PreviewDay komponento.

Slika : Diagram modula Main



#### Module.js

Module.js je datoteka, ki vsebuje skoraj vso logiko modula Main. Vsebuje tri ustvarjalce akcij (swipeDays, getPreviewDay in navigate), vsebuje pa tudi svojega zmanjševalca, ki bo spremenil stanje aplikacije v primeru treh različnih dogodkov (pri posodobitvi enega dneva, pri inicializaciji dneh ter pri posodobitvi previewDay



Slika : Začetno stanje zmanjševalca

Metodi getDay podamo indeks ter Date objekt, vrne nam pa objekt s sledečimi lastnostmi:

* id
* momentObj
* toString
* date

V začetnem stanju imamo tudi tri dneve(včeraj:0, danes: 1, jutri: 2), poimenovane kar z indeksi (katere mi podamo dnevom). Na te indekse se sklicuje skoraj da celotna aplikacija. Dnevi iščejo podatke v trgovini glede na indeks, ki jim je dan, stanje aplikacije se posodablja glede na indeks dneva, kjer naj se posodobi itd…

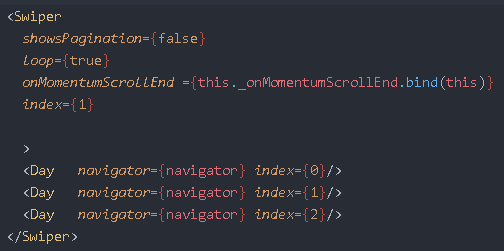
#### Saga

Saga sprejema tri različne dogodke: SWIPEDAYS, getPreviewDay in pa NAVIGATE\_DAY.

Ko sprejme dogodek SWIPEDAYS, pogleda naprej v bazo če že obstaja dan z podanim datumom, in če ne obstaja, ga enostavno ustvari in vrne nazaj sagi, ta pa potem odpošlje tri akcije. Eno odpošlje zmanjševalcu za dneve, drugi dve pa odpošlje zmanjševalcu za dogodke in opomnike. To je potrebno storiti, saj so te trije moduli popolnoma ločeni in neodvisni med seboj.

#### Swiper komponenta

Main modul vsebuje komponento Swiper, kateri podamo podrejene komponente oziroma module Day skupaj z njihovimi indeksi. V modulu Main držimo tudi vrednost trenutnega indeksa, ta se spreminja ob spreminjanju izbranega dneva če podrsamo naprej ali nazaj. Komponenta Swiper zahteva za svojo delovanje metodo onMomentumScroll, kjer bomo tudi ugotovili, ali je uporabnik podrsal naprej, ali nazaj, ter nato odposlali akcijo swipeDays, kjer bomo generirali nov dan. Ob vsakem drsanju naprej ali nazaj bomo le generirali naslednji ali prejšnji dan, da se ostala dva dneva ne bosta ponovno upodobila in bo aplikacija tako delovala zares tekoče, saj se bo ponovno upodabljala le tista stvar, ki se more.

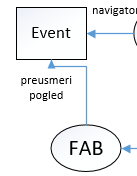


Slika : Komponenta Swiper

#### Komponenta FloatingActionButton

Ta komponenta služi ustvarjanju dogodkov in opomnikov na izbrani dan. Na dotik gumba se nam prikažeta dve možnosti, ena za dodajanje dogodka, druga pa za dodajanje opomnika. Ko izberemo, kaj želimo dodati, se s pomočjo objekta »navigator« usmerimo na željeno pot, kjer bomo lahko potem urejali dogodek ali opomnik, ki ga želimo dodati.

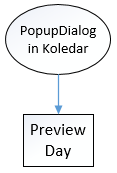
Slika : FloatingActionButton komponenta



#### Komponente PopupDialog, CalendarPicker in PreviewDay

Komponenta PopupDialog nam bo ob pritisku na gumb, ki je lociran v komponenti Header (ta služi samo za enostavno prikazovanje menija, ne vsebuje nobene logike), prikazal na ekran koledar, kjer se nam bo, ob dotiku na poljubni datum, prikazala komponenta PreviewDay.

Slika : PopupDialog

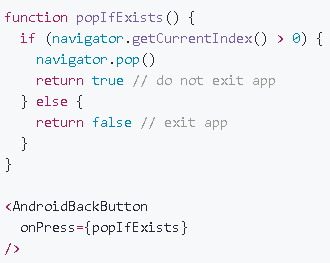


Ob samem kliku na datum bomo odposlali akcijo getPreviewDay, ki jo bo procesirala saga, kjer se bo izvedla poizvedba v bazo ter nato spreminjanje stanja aplikacije oziroma sprememba lastnosti previewDay, ki bo dobila vrednost dneva, ki je bil pridobljen iz baze. Ko se bo ta lastnost spremenila, se bo komponenta Main ponovno upodobila, saj v njej sledimo spremembam previewDay, in se nato tisti dan prikazal v komponenti Preview Day.

#### Komponenta AndroidBackButton

S to komponento lahko kontroliramo vedenje aplikacije ob pritisku gumba za nazaj na telefonu. Z funkcijo onPress lahko definiramo obnašanje aplikacije, ko uporabnik klikne gumb za nazaj, in če onPress funkcija vrne neresnično vrednost, potem bo pritisk na gumb povzročil zaprtje aplikacije. Če pa vrne resnično vrednost, potem pritisk na ta gumb ne naredi ničesar.

V našem primeru se v funkciji onPress preveri, na kateri poti navigator objekta smo, in potem glede na rezultat gre aplikacija na prejšnjo sceno, ali pa se celo zapre.



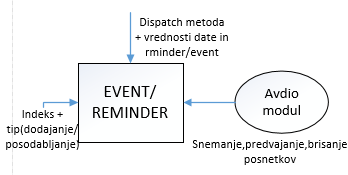
Slika : AndroidBackButton komponenta

### Event in Reminder

Kakšna je pravzaprav razlika med dogodkom in opomnikom? Opomnik sem definiral tako, da je to neka stvar, ki ima stanje (torej dokončano ali nedokončano), in se lahko, če je nedokončano, pojavlja tudi na naslednjem dnevu, medtem ko je dogodek enkraten oziroma ima neko trajanje. Tudi pri opomniku lahko definiramo, kdaj naj nas na tisti dan aplikacija o njem obvesti, a pri dogodku lahko definiramo koliko prej nas lahko aplikacija obvesti o nekem dogodku.

Oba modula vključujeta le eno komponento, Form, ter identične metode, ki jih importiramo v modul iz trgovine ter avdio modula, ki sem ga spisal v Java jeziku, hkrati pa pri navigaciji iz modula Main dobita tudi podatke, ki jih potrebujeta za svojo delovanje, namreč dobita indeks dneva, kateremu bomo dodali opomnik/dogodek ter vrednost i, ki določa, ali se gre za dodajanje ali posodabljanje dogodka/opomnika, saj ne želim ustvarjati dodatnih komponent (ena variacija torej posodabljanje, druga pa dodajanje).

Slika : Potrebni podatki in metode za modul Event/Reminder



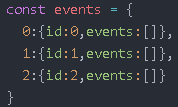
#### Module.js

Oba modula, Reminder in Event, imata tudi identično datoteko module.js, kjer imata zmanjševalca, začetno stanje zmanjševalca, ter tri ustvarjalce akcij (dodajanje, posodabljanje, brisanje). Zmanjševalec spremeni stanje glede na sledeče tipe dogodkov:

1. UPDATE\_EVENTS
2. ADDEVENT
3. EDITEVENT
4. DELETEEVENT
5. INITIALIZEEVENT

Začetno stanje je spet podobno stanju zmanjševalca pri modulu Main.

Slika : Začetno stanje



Tudi pri teh dveh modulih se razlikujejo dogodki/opomniki po indeksih.

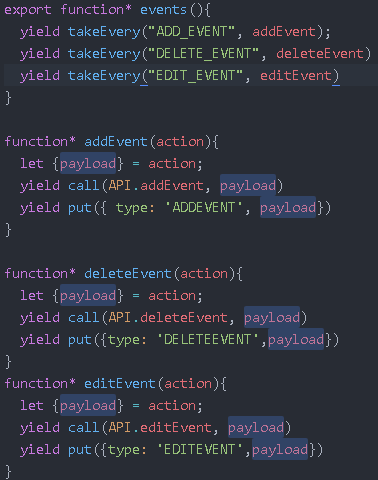
#### Saga.js

Saga modulov Reminder in Event sprejema tri tipe uporabniških akcij:

1. ADD\_EVENT (za dodajanje dogodka/opomnika)
2. EDIT\_EVENT (za spreminjanje že obstoječega dogodka/opomnika)
3. DELETE\_EVENT (za brisanje dogodka/opomnika)

Pri vsaki obdelavi akcije se kliče primerna metoda iz modula API, ki bo poskrbela, da se bodo podatki iz baze zbrisali/dodali/spremenili.

Slika : Saga module Event



#### Komponenta Form

Komponenta Form je pri obeh modulih skorajda identična. Od modula Event/Reminder dobi potrebne podatke (celoten dogodek/opomnik, navigator, metode za predvajanje, brisanje in shranjevanje ter ustvarjanje zvočnih posnetkov, ter metode, s katerimi bomo odposlali akcije v sago, ter vrednost, na podlagi katere se bo komponenta odločila, ali gre za dodajanje ali spreminjanje že obstoječega dogodka/opomnika).

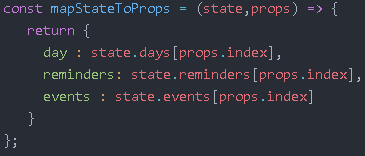
Ta komponenta se ob uporabi večkrat upodobi, saj so vsi podatki, ki jih bomo poslali v akciji v sago, prisotni v state, kar pomeni, da vsakič ko karkoli spremenimo (vnesemo kaj v tekst polje, itd...), se bo izvedla metoda setState in komponenta se bo ponovno upodobila. A ta operacija nima skoraj da nikakršnega vpliva na samo učinkovitost, saj je ta zelo zelo zanemarljiv.

### Day in PreviewDay

Modula Day in PreviewDay si delita z modulom Main datoteko module.js ter saga.js. Modul Day ima dve komponenti, Header in Body, katera vsebina je identična modulu PreviewDay. V Header komponenti se prikazuje le datum dneva, v Body komponenti pa se prikazujejo vsi dogodki in opomniki, ki imajo lastnost completed nastavljeno na »false«. Naprej se upodobijo vsi dogodki, šele nato pa opomniki, ki jih komponenta dobi od modula Day. Pri opomnikih, ki se naložijo, ima uporabnik možnost nastavljanja lastnosti completed opomnika na »true«, kar bo modul Day in komponento Body še enkrat upodobilo, saj se je tako stanje aplikacije spremenilo. Hkrati pa ima pri dogodkih in opomnikih možnost brisanja in spreminjanja.

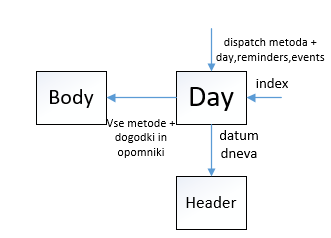
Z indeksom, ki mu ga poda modul Main, lahko modul Day sledi spremembam trem vrednostim iz stanja aplikacije: day, reminders in pa events.

Slika : Dan, dogodki in opomniki



Komponenti Body pa tudi poda vse tri metode, s katerimi bo uporabnik upravljal ko bo želel opomnik nastaviti na »končano«, ali pa zbrisati/spremeniti dogodek/opomnik.

Slika : Diagram modula Day



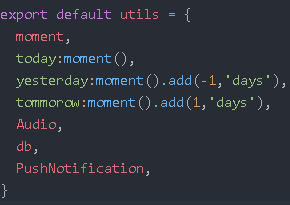
Glavna razlika med modulom Day in PreviewDay je le v tem, da PreviewDay ne vsebuje Header komponente, temveč v svoji render metodi vsebuje le skrajšano verzijo komponente Body, kjer se bodo prikazali vsi opomniki in dogodki tistega dne, le na manjše ter brez kakršnihkoli možnosti spreminjanja, brisanja ali dodajanja novih dogodkov ali opomnikov.

### API.js, Utils.js in Avdio modul

Te tri stvari so uporabljene konstantno skozi celotno aplikacijo. API.js vsebuje vse metode, s katerimi dostopamo do baze, hkrati pa tudi vsebuje objekt PushNotification, ki je uporabljen pri vseh metodah, kjer se spremeni čas dogodka ali pa opomnika.

PushNotification nam zagotavlja nastavljanje notifikacij aplikacije. API.js pa ga dobi iz datoteke Utils.js, kjer so zbrani vsi objekti, ki jih bo aplikacija potrebovala za delovanje.

Slika : Objekti v utils.js



Vsakič, ko bomo dodali nek opomnik ali dogodek, bomo lahko z metodo PushNotification.localNotificationSchedule nastavili notifikacijo za določen čas. Ta notifikacija se nam bo na telefonu prikazala tudi če je aplikacija v ozadju, ali pa je zaprta.

Pri metodi initializeDays, ki se kliče že v vstopni točki aplikacije (index.android.js), se ustvarijo vse notifikacije današnjega dne, zato nam ni treba ustvarjati notifikacij za dva tedna vnaprej, ker se ob dodajanju opomnika ali pa dogodka lahko enostavno preveri, ali je datum dogodka/opomnika enak današnjemu datumu.

Za snemanje, ustvarjanje, predvajanje in brisanje zvočnih datotek sem pa spisal preprost avtohton modul (torej v Javi) ter ga importu v React Native.

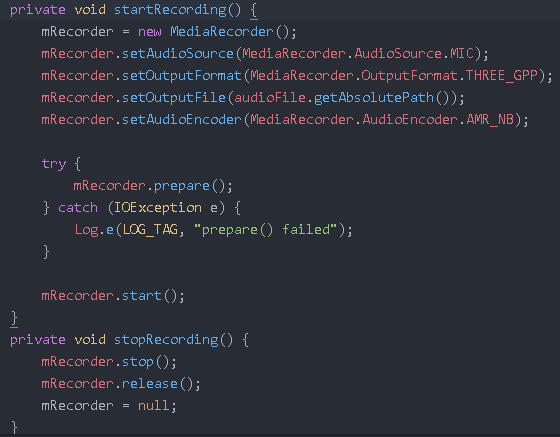
Slika : Primer dveh metod v modulu Audio



Avdio modul potrebuje za izvedbo vseh svojih operacij ( ki sem jih prej naštel), podano datum dneva in številko. Ta številka je zaporedna številka ustvarjene zvočne datoteke za opomnik in dogodek, tako da končno ime datoteke lahko izgleda takole: 11061998-1.3gp.

Za snemanje zvoka uporablja modul Audio objekt MediaRecorder, za predvajanje zvočne datoteke pa MediaPlayer.

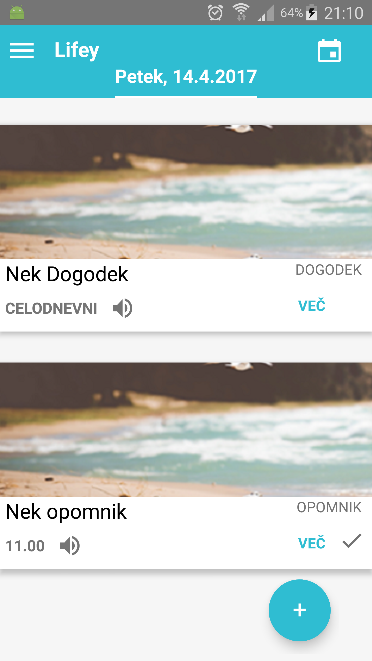
Slika : Primer uporabe MediaRecorder



## Končni izdelek

Ob odprtju aplikacije se nam naprej prikaže današnji dan skupaj z vsemi opomniki in dogodki tistega dne.

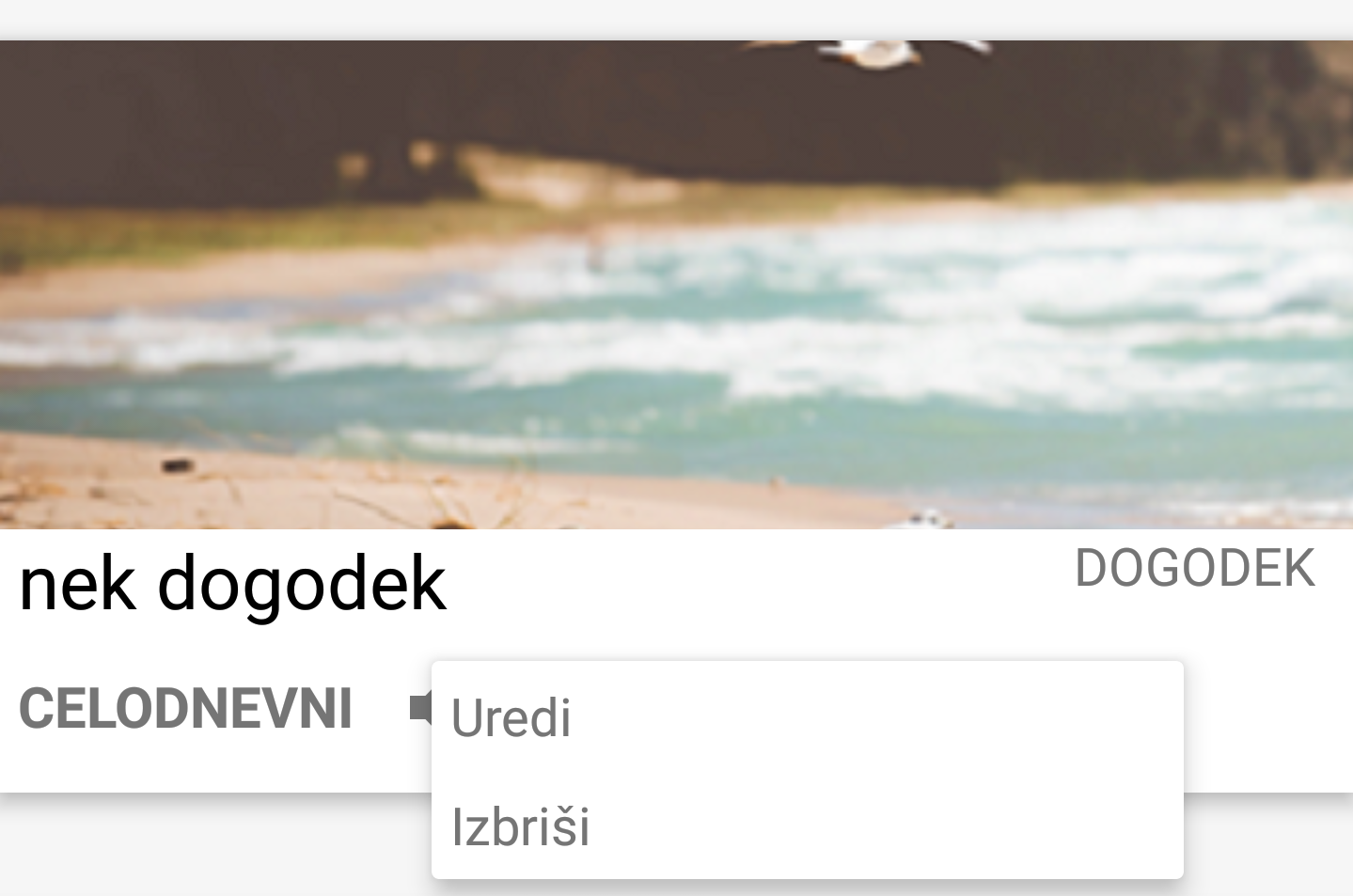
Slika : Odprtje aplikacije



Ob pritisku na več, se nam prikaže menu, kjer lahko izberemo ali uredi, ali pa izbriši.

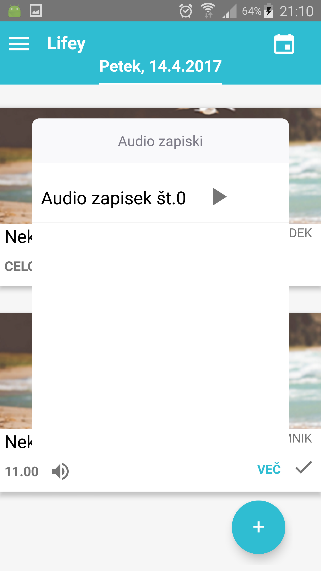
Vsak dogodek in opomnik imata ime levo zgoraj, spodaj imata trajanje oziroma uro, desno zgoraj pa imata tip (torej dogodek ali opomnik).

Slika : Menu dogodka/opomnika



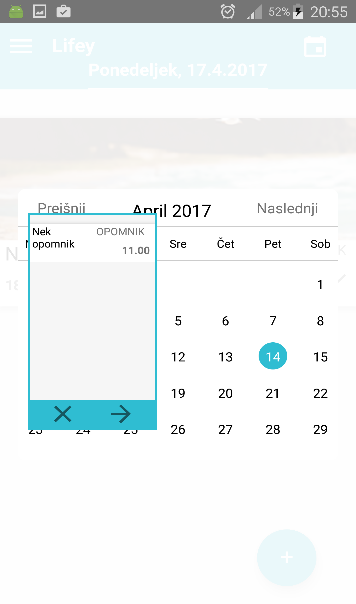
Če ima dogodek ali opomnik tudi zvočne posnetke, je zraven trajanje ikona za glasnost. Ko kliknemo na to ikono se nam odprejo vsi zvočni posnetki, na katere lahko poslušamo.

Slika : Primer zvočnih posnetkov



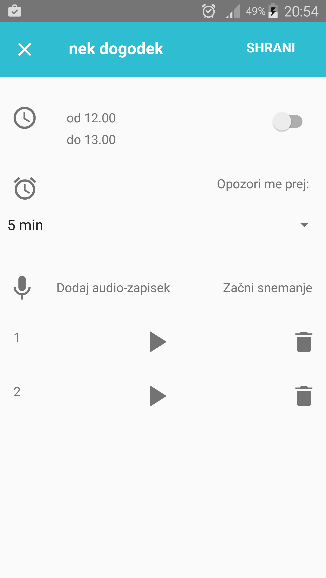
Zgoraj desno v aplikaciji imamo ikono za koledar, na katero lahko kliknemo, nato pa se nam bo odprl koledar, kjer se nam bo ob izbiri nekega datuma prikazal tudi predogled tega dne. Ob pritisku na »X« bo predogled izginil, ob pritisku na puščico pa se bomo premaknili na tisti dan.

Slika : Koledar in predogled dneva



Dodajanje opomnika in dogodka je skoraj da identično, samo da imamo pri dogodku tudi možnost definirati trajanje (od-do) ter koliko prej naj nas aplikacija opozori na ta dogodek, medtem ko imamo pri opomniku samo en čas.

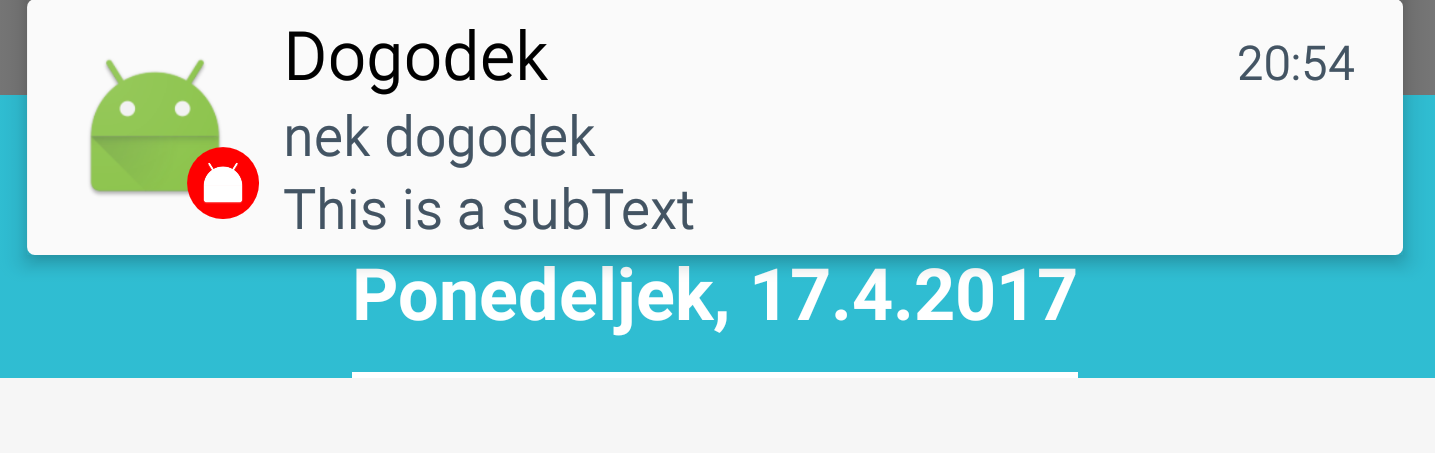
Slika : Dodajanje dogodka



Na klik začni snemanje lahko tudi dodajamo zvočne posnetke, na klik na koš za smeti pa izbrišemo ta zvočni posnetek. Celoten pogled pri urejanju/shranjevanju je identičen, le da imamo pri gumbu za shranit enkrat napisano »POSODOBI«, enkrat pa »SHRANI«.

Prejemamo tudi vsa obvestila za dogodke in opomnike, ne glede na to, ali je aplikacija v ozadju ali je celo zaprta. Če nismo v aplikaciji nas lahko klik na obvestilo popelje nazaj v aplikacijo.

Slika : Primer obvestila



# Sklepne ugotovitve

S tem projektom sem se naučil veliko novih tehnologij, kot so Redux, Redux Saga, MongoDB in seveda React Native. Naučil sem se veliko o samem planiranju aplikacije ter kako strukturirati aplikacije, da se da še tudi v bodočnosti razumeti kodo.

Ugotovil sem, da se da s pomočjo React Native tehnologije ustvariti aplikacijo, ki je primerljiva v učinkovitosti z avtohtono aplikacijo. Le treba je biti previden pri strukturiranju aplikacije, da ne pride do nepotrebnih ponovnih upodobitev scen.

Aplikaciji umanjka še ogromno lastnosti, ki jih mislim še v prihodnosti dodati. Prva stvar, na katero pomislim, je tedenski in mesečni pregled dogodkov in opomnikov, a tam se zna pravzaprav zelo zaplesti, kar se tiče učinkovitosti aplikacije, saj nočem imeti posebej sceno prav za tedenski pregled in mesečni pregled, ker bi to pomenilo ponovno upodabljanje skorajda celotne aplikacije ob vsakem skoku na mesečni, tedenski ali pa dnevni pregled dogodkov in opomnikov.

Mislim, da zna React Native tehnologija popolnoma prevzeti svet ustvarjanja aplikacij v naslednjih letih. Spraviti mobilno aplikacijo na druge platforme, kot so iOS in Windows, ne predstavlja več takšnega problema kot je to predstavljalo pet let nazaj. Ta tehnologija je popolnoma primerna za podjetja, ki želijo v čim krajšem času dati na trg aplikacijo, ki bo delovala tako dobro kot navadna avtohtona aplikacija, ter za posameznike, ki v svojem prostem času ustvarjajo aplikacije in želijo, da so te aplikacije dostopne na vseh platformah.

# Literatura in viri

React Native, pridobljeno z: <https://facebook.github.io/react-native/> dne 1. 5. 2017

Redux, pridobljeno z: [https://github.com/reactjs/redux](https://github.com/reactjs/redux%20dne%201.%205.%202017) dne 1.5.2017

Redux dokumentacija, pridobljeno z: http://sparks.gogo.co.nz/ch340.html dne 1.5.2017

Razumevanje Redux, pridobljeno z: <http://www.youhavetolearncomputers.com/blog/2015/9/15/a-conceptual-overview-of-redux-or-how-i-fell-in-love-with-a-javascript-state-container> dne 1.5.2017

10 napotkov za boljšo Redux arhitekturo, pridobljeno z: <https://medium.com/javascript-scene/10-tips-for-better-redux-architecture-69250425af44> dne 1.9.2017

React Native Swiper, pridobljeno z: <https://github.com/leecade/react-native-swiper> dne 1. 8. 2017

MongoDB, pridobljeno z: [https://github.com/mongodb/mongo](https://github.com/mongodb/mongo%20%20) dne 2.2.2017

Redux Saga, pridobljeno z: <https://github.com/redux-saga/redux-saga> dne 2. 2. 2017

MongoDB za React Native, pridobljeno z: <https://github.com/smartdemocracy/react-native-local-mongodb> dne 3. 1. 2017

Nastavitev PushNotifications za React Native, pridobljeno z: <https://medium.com/differential/how-to-setup-push-notifications-in-react-native-ios-android-30ea0131355e> dne 3. 10. 2017

PushNotifications, pridobljeno z: <https://github.com/zo0r/react-native-push-notification> dne 3. 10. 2017

CalendarPicker , pridobljeno z: <https://github.com/stephy/CalendarPicker> dne 3.12. 2017

# Priloge

Ker je v temu projektu prisotna ogromna količina kode (sam API.js obsega okrog 350 vrstic kode), bom priložil le povezavo do projekta na githubu.

Povezava: <https://github.com/xmisd/Lifey>

# Izjava o avtorstvu

Izjavljam, da je seminarska naloga v celoti moje avtorsko delo, ki sem ga izdelal samostojno s pomočjo navedene literature in pod vodstvom mentorja.

23. 06. 2017

Tom Šabanov