



Samo Mulej Bratec

**Razvoj heterogenih mobilnih aplikacij v razvojnem  
okolju RAD Studio 10**

Diplomsko delo

Maribor, september 2017



# **Razvoj heterogenih mobilnih aplikacij v razvojnem okolju RAD**

## **Studio 10**

### **Diplomsko delo**

Študent: Samo Mulej Bratec

Študijski program: Univerzitetni študijski program, Informatika in tehnologije  
komuniciranja

Smer: Informacijski sistemi

Mentor: doc. dr. Domen Verber

Maribor, september 2017



Univerza v Mariboru

Fakulteta za elektrotehniko,  
računalništvo in informatiko  
Smetanova ulica 17  
2000 Maribor, Slovenija

**FERI**

Številka: E1078132

Datum in kraj: 29. 09. 2016, Maribor

Na osnovi 330. člena Statuta Univerze v Mariboru (Ur. l. RS, št. 44/2015)  
izdajam

### SKLEP O DIPLOMSKEM DELU

1. **Samu Muleju Bratcu**, študentu univerzitetnega študijskega programa **INFORMATIKA IN TEHNOLOGIJE KOMUNICIRANJA**, smer **Informacijski sistemi**, se dovoljuje izdelati diplomsko delo.
2. **MENTOR:** doc. dr. Domen Verber
3. **Naslov diplomskega dela:**  
**RAZVOJ HETEROGENIH MOBILNIH APLIKACIJ V RAZVOJNEM OKOLJU RAD STUDIO 10**
4. **Naslov diplomskega dela v angleškem jeziku:**  
**HETEROGENOUS MOBILE APPLICATION DEVELOPMENT IN RAD STUDIO 10 DEVELOPMENT ENVIRONMENT**
5. Diplomsko delo je potrebno izdelati skladno z "Navodili za izdelavo diplomskega dela". Skladno s 7. členom *Pravilnika o postopku priprave in zagovora diplomskega dela na dodiplomskem študiju* je bilo odobreno podaljšanje roka za oddajo diplomskega dela do 30. 09. 2017. Diplomsko delo študent-ka odda v treh izvodih (dva trdo vezana izvoda in en v spiralo vezan izvod) ter en izvod elektronske verzije v referatu za študentske zadeve.

Pravni pouk: Zoper ta sklep je možna pritožba na senat članice v roku 3 delovnih dni.

Dekan:

red. prof. dr. Borut Žalik



Obvestiti:

- kandidata,
- mentorja,
- odložiti v arhiv.

## **ZAHVALA**

Zahvaljujem se mentorju doc. dr. Domnu Verberju za pomoč in vodenje pri opravljanju diplomskega dela.

Posebna zahvala velja staršem in starim staršem, ki so mi omogočili študij, dekletu Vesni, ki me je spodbujala v času študija, ter prof. Olgi Bratec Veleski za lektoriranje diplomskega dela.

## **Razvoj heterogenih mobilnih aplikacij v razvojnem okolju RAD Studio**

**Ključne besede:** RAD Studio, Android Studio, Xcode, heterogene aplikacije, domorodne aplikacije

**UDK:**

### **Povzetek**

*V diplomskem delu smo raziskovali problem razvoja heterogenih aplikacij za pametne telefone. Obravnavali smo več različnih mobilnih naprav, ki uporabljajo različne operacijske sisteme. Za doseg večjega števila uporabnikov je potrebno podpreti čim več le-teh. Aplikacije se lahko razvijajo v domorodnih okoljih ali okoljih za navzkrižne platforme. Zanimalo nas je kvaliteta razvojnega okolja RAD Studio, če lahko nadomesti razvoj domorodnih aplikacij. Da bi to ugotovili, smo izdelali prototipno aplikacijo tako v domorodnih okoljih (Android Studio in Xcode) ter v RAD Studiu in jih na koncu primerjali med seboj. RAD Studio nudi manj udobja od domorodnih okolij.*

## **Heterogenous mobile application development in RAD Studio 10 development environment**

**Key words:** RAD Studio, Android Studio, Xcode, heterogeneity, heterogenous applications, native applications, cross platform

**UDK:**

### **Abstract**

*In the B.S. thesis we research the problem of development of heterogeneous applications for smart phones. We considered several different mobile devices that used different operational systems. Therefore one must produce applications for different operational system in order to reach several users. Applications can be developed in native environments or in cross platforms. We were interested in the quality of the development environment RAD Studio, whether it can replace development of native applications. To find this out we generated prototype application both in native environments (Android Studio and Xcode) and in RAD Studio and compared them at the end. RAD Studio offered less comfort than the native environments.*

# KAZALO

<b>1. UVOD .....</b>	<b>1</b>
1.1 Področje .....	1
1.2 Namen.....	1
1.3 Sestava diplomskega dela .....	1
<b>2. HETEROGENI RAZVOJ APLIKACIJ .....</b>	<b>3</b>
2.1 Heterogenost mobilnih aplikacij .....	3
2.1.1 Android .....	3
2.1.2 Windows Phone .....	3
2.1.3 iOS .....	4
2.2 Navzkrižna platforma .....	4
<b>3. PRIMERJAVA RAZVOJNEGA OKOLJA RAD STUDIO Z DRUGIMI KONKURENČNIMI OKOLJI.....</b>	<b>5</b>
3.1 Apache Cordova .....	5
3.2 Xamarin .....	5
3.3 Appcelerator Titanium .....	6
3.4 RAD Studio .....	6
3.5 Primerjava heterogenih razvojnih orodij .....	7
<b>4. NAČRT IN IZDELAVA TESTNE APLIKACIJE.....</b>	<b>8</b>
4.1 Zasnova testne aplikacije .....	8
4.2 Izdelava aplikacije v Android studio.....	8
4.2.1 Facebook povezava .....	10
4.2.2 Prikaz podatkov na geografski karti .....	13
4.2.3 Določanje položaja uporabnika .....	15
4.2.4 Lokalno shranjevanje podatkov .....	16
4.2.5 Uporaba JSON-a in povezava s strežnikom.....	18
4.3 Izdelava aplikacije v xCode.....	20
4.3.1 Facebook povezava .....	21
4.3.2 Prikaz podatkov na geografski karti .....	23



4.3.3	Določanje položaja uporabnika .....	25
4.3.4	Lokalno shranjevanje podatkov .....	27
4.3.5	Uporaba JSON-a in povezava s strežnikom.....	29
4.4	Izdelava testne aplikacije v Rad studio .....	31
4.4.1	Prijava.....	33
4.4.2	Prikaz podatkov na geografski karti.....	35
4.4.3	Določanje položaja uporabnika .....	39
4.4.4	Lokalno shranjevanje podatkov .....	41
4.4.5	Uporaba JSON-a in povezava s strežnikom.....	47
4.5	Primerjava okolja RAD Studio z domorodnimi okolji.....	51
<b>5.</b>	<b>SKLEP .....</b>	<b>53</b>
<b>6.</b>	<b>VIRI .....</b>	<b>54</b>

## KAZALO SLIK

Slika 4.1: MVC in Android Studio .....	9
Slika 4.2: Grafična izdelava uporabniškega vmesnika v Android Studiu.....	10
Slika 4.3: Metoda onCreate.....	10
Slika 4.4: Facebook prijava v Android Studiu .....	11
Slika 4.5: Metoda za Facebook povezavo v Android Studiu .....	11
Slika 4.6: Shranjevanje uporabnikovih podatkov v Android Studiu.....	12
Slika 4.7: Shranjevanje uporabnikovih prijateljev v Android Studiu.....	12
Slika 4.8: Izbira dogodka na zemljevidu v Android Studiu .....	13
Slika 4.9: Prikaz dogodkov na zemljevidu v Android Studiu .....	13
Slika 4.10: Inicializacija GoogleApiClient.....	14
Slika 4.11: Metoda za spreminjanja lokacije ob kliku po zemljevidu v Android Studiu.....	14
Slika 4.12: Metoda za prikaz dogodkov v Android Studio .....	14
Slika 4.13: Delovanje GPS v Android Studiu.....	15
Slika 4.14: Prikaz zemljevida v Android Studiu .....	15
Slika 4.15: Metoda za pridobivanje trenutne lokacije v Android Studiu .....	16
Slika 4.16: Prikaz dogodkov v Android Studiu.....	16
Slika 4.17: Podatkovna baza v Android Studiu.....	17
Slika 4.18: Kreiranje tabel v Android Studiu .....	17
Slika 4.19: Odpiranje in zapiranje povezave s podatkovno bazo v Android Studiu.....	17
Slika 4.20: Metoda za shranjevanje uporabnika v Android Studiu .....	18
Slika 4.21: Metoda za pridobivanje uporabnika v Android Studiu .....	18
Slika 4.22: Metoda za shranjevanje podatkov na strežnik v Android Studiu.....	19
Slika 4.23: Metoda za branje podatkov iz strežnika v Android Studiu .....	19
Slika 4.24: Razred Parameter .....	20
Slika 4.25: Prikaz izgleda v Storyboardu .....	20
Slika 4.26: Metoda viewDidLoad.....	21
Slika 4.27: Prikaz Facebook prijave v Xcodu .....	21
Slika 4.28: Metoda za prijavo s Facebookom v Xcodu .....	22
Slika 4.29: Metoda za pridobivanje osebnih podatkov v Xcodu .....	22
Slika 4.30: Metoda za pridobivanje prijateljev v Xcodu.....	22
Slika 4.31: Izbira dogodka na zemljevidu v Xcode .....	23
Slika 4.32: Prikaz dogodkov na zemljevidu v Xcode.....	23
Slika 4.33: Metoda za različne barve označevalcev .....	24
Slika 4.34: Razred ColorPointAnnotation.....	24
Slika 4.35: Metoda za spreminjanje lokacije ob kliku po zemljevidu v Xcodu.....	25
Slika 4.36: Metoda za prikaz dogodkov na zemljevidu v Xcodu .....	25
Slika 4.37: Prikaz delovanja GPS v Xcode .....	25
Slika 4.38: Prikaz zemljevida v Xcode.....	25
Slika 4.39: Metoda za pridobivanje trenutne lokacije v Xcodu .....	26
Slika 4.40: Metoda, ki pridobi podatke o lokaciji iz koordinat v Xcodu .....	26
Slika 4.41: Prikaz dogodkov v Xcodu .....	27
Slika 4.42: Izdelava podatkovnega modela v Xcodu .....	27
Slika 4.43: Metoda za shranjevanje dogodkov v Xcodu.....	28
Slika 4.44: Metoda za branje dogodkov v Xcode .....	28
Slika 4.45: Pretvorba v JSON obliko v Xcodu.....	29

Slika 4.46: Metoda za komuniciranje s strežnikom v Xcodu .....	29
Slika 4.47: Metoda za pridobivanje podatkov iz strežnika v Xcodu .....	30
Slika 4.48: Metoda za shranjevanje dogodka na strežnik v Xcodu .....	30
Slika 4.49: Postavitev projekta v RAD Studiu .....	31
Slika 4.50: Videz razvijalnega okolja RAD Studio .....	32
Slika 4.51: Dodajanje komponent in dogodkov v RAD Studiu .....	32
Slika 4.52: Prijava v RAD Studiu .....	33
Slika 4.53: Registracija v RAD Studiu .....	33
Slika 4.54: Razred User .....	34
Slika 4.55: Vmesnik User .....	34
Slika 4.56: Metoda za preverjanje prijave uporabnika v RAD Studiu .....	34
Slika 4.57: Metoda za uspešno prijavo v RAD Studiu .....	34
Slika 4.58: Metoda za registracijo v .....	35
Slika 4.59: Metoda za validacijo registracije v RAD Studiu .....	35
Slika 4.60: Izbira dogodka na zemljevidu v RAD Studiu .....	36
Slika 4.61: Prikaz dogodkov na zemljevidu v RAD Studiu .....	36
Slika 4.62: Metoda za dodajanje dogodkov na zemljevid v RAD Studiu .....	36
Slika 4.63: Metoda, ki pretvori naslov v koordinate v RAD Studiu .....	37
Slika 4.64: Metoda, ki prejme kot parametre koordinate pretvorjene iz naslova v RAD Studiu .....	37
Slika 4.65: Metoda, ki doda označevalec za posamezni dogodek na zemljevid v RAD Studiu .....	37
Slika 4.66: Metoda, ki se izvede ob kliku na označevalec v RAD Studiu .....	38
Slika 4.67: Metoda, ki prikaže dialog za dogajanje dogodkov in ustrezno vnese podatke o naslovu .....	38
Slika 4.68: Prikaz delovanja GPS v RAD Studiu .....	39
Slika 4.69: Prikaz zemljevida v RAD Studiu .....	39
Slika 4.70: Metoda za dodajanje začetnega označevalca na zemljevid v RAD Studiu .....	40
Slika 4.71: Metoda za centriranje zemljevida v RAD Studiu .....	40
Slika 4.72: Metoda, ki se proži ob kliku na gumb "trenutna lokacija" v RAD Studiu .....	40
Slika 4.73: Metoda, ki vnese podatke v ustrezna polja pri dodajanju dogodkov v RAD Studiu .....	41
Slika 4.74: Prikaz dogodkov v RAD Studiu .....	42
Slika 4.75: Urejanje sloga v RAD Studiu .....	43
Slika 4.76: Sprememba ozadja za ListView v RAD Studiu .....	43
Slika 4.77: Nastavitev lastnosti za lokalno podatkovno bazo vRAD Studio .....	43
Slika 4.78: Urejanje povezave z SQLite v RAD Studiu .....	44
Slika 4.79: Metoda, ki poveže podatkovno bazo z s3db datoteko v RAD Studiu .....	44
Slika 4.80: Kreiranje tabel v RAD Studiu .....	44
Slika 4.81: Metoda, ki ustvari tabele v RAD Studiu .....	45
Slika 4.82: Izgled Forme ta podatkovno bazo v RAD Studiu .....	45
Slika 4.83: SQL stavek za dodajanje dogodka v podatkovno bazo v RAD Studiu .....	45
Slika 4.84: Metoda za shranjevanje dogodkov v lokalno podatkovno bazo v RAD Studiu .....	46
Slika 4.85: SQL stavek za branje podatkov iz podatkovne baze v RAD Studiu .....	46
Slika 4.86: Metoda za branje podatkov iz lokalne podatkovne baze v RAD Studiu .....	47
Slika 4.87: Nastavitve zahteve v RAD Studiu .....	47
Slika 4.88: Metoda, ki pošlje zahtevo na strežnik v RAD Studiu .....	48

Slika 4.89: Metoda, ki prebere vse dogodke iz strežnika v RAD Studiu .....	48
Slika 4.90: Metoda, ki izloči vsebino znotraj podanih znakov .....	49
Slika 4.91: Metoda, ki razdeli niz na več delov glede na podan znak .....	49
Slika 4.92: Razred Parameter .....	50
Slika 4.93: Metoda, ki pretvori niz v razred Parameter .....	50
Slika 4.94: Metoda za pretvorbo dogodka iz JSON oblike v objekt .....	51

## KAZALO TABEL

Tabela 3.1: Primerjava karakteristik pri razvojnih okoljih.....	7
-----------------------------------------------------------------	---

## **SEZNAM UPORABLJENIH KRATIC**

SQL - strukturirani povpraševalni jezik za delo s podatkovnimi bazami (angl. Structured Query Language)

IT – informacijska tehnologija (information technology)

HTML – označevalni jezik (angl. HyperText Markup Language)

CSS – stilska podloga (angl. Cascading Style Sheets)

JSON – sintaksa za izmenjavo podatkov (angl. JavaScript Object Notation)

GPS – globalni sistem pozicioniranja (angl. Global Positioning System)

# 1. UVOD

## 1.1 Področje

Razvoj aplikacij za mobilne telefone je od prvih dni razvoja na enobarvnih zaslonih zelo napredoval. Danes so na voljo prefinjene funkcije in veliko število platform za razvoj nove programske opreme. [2] Tako morajo razvijalci za doseg večjega trga podpirati različne operacijske sisteme, ki delujejo povsem drugače. Za reševanje tega problema je tako nastalo vedno več okolij, namenjenih razvoju, kjer lahko aplikacija, izdelana v enem okolju in enem jeziku, deluje na različnih operacijskih sistemih. Zaradi tega smo se odločili testirati RAD Studio, eno izmed takšnih okolij.

## 1.2 Namen

Pred začetkom razvoja mobilnih aplikacij si je potrebno izbrati primerno okolje (domorodna okolja ali okolje za navzkrižne platforme). Ker za pametne telefone obstajajo različni operacijski sistemi, lahko razvoj v domorodnih okoljih vzame več časa ter je potrebno tudi znanje različnih jezikov. Razvoj v okolju za navzkrižne platforme zahteva poznavanje le enega jezika, kar je odvisno od okolja, kateri jezik le-to uporablja. Naš namen je bil, da ugotovimo, če lahko z okoljem RAD Studio izdelamo enako kvalitetne aplikacije (stabilne, optimizirane, dobra uporabniška izkušnja) ter porabimo manj časa kot v domorodnih okoljih (Android Studio in Xcode). Prav tako nas je tudi zanimala kvaliteta razvojnega okolja, če deluje hitro in nam ne povzroča odvečnih težav.

## 1.3 Sestava diplomskega dela

V prvem delu diplomskega dela smo obrazložili, kaj pomeni heterogenost, na kratko opisali prevladujoče operacijske sisteme za pametne telefone ter naštetili, katera orodja se uporabljajo za razvoj navzkrižnih platform. Nato smo si bolj podrobno pogledali razvijalno okolje RAD Studia ter ugotovili, kaj le-ta vsebuje. Za ugotovitev kvalitete okolja RAD Studio smo izdelali prototipno aplikacijo BunchUp v orodjih Android Studio ter Xcode, domorodnih razvojnih okoljih za Android in iOS, ter nato še enako aplikacijo v okolju RAD Studiu in med

seboj primerjali razvoj aplikacije. Zaradi težav pri povezovanju aplikacije, izdelane v RAD Studiu z iPhone mobilno napravo, nam ni uspelo prikazati zaslonskih mask na njem. RAD Studio namreč ne podpira nalaganje aplikacij na iOS simulator za aplikacije, izdelane v C++ jeziku. Prav tako ima napako, da ni možno naložiti zadnje verzije SDK (Software Development Kit), kar je posledično pripeljalo do tega, da aplikacije prav tako ni bilo mogoče naložiti na mobilno napravo, ker za to Apple zahteva najnovejšo verzijo.



## 2. HETEROGENI RAZVOJ APLIKACIJ

### 2.1 Heterogenost mobilnih aplikacij

Razvoj aplikacij za mobilne telefone je od prvih dni razvoja na enobarvnih zaslonih storil velik korak. Danes so na voljo prefinjene funkcije in veliko število platform za razvoj nove programske opreme. Tako strojna kot tudi programska oprema na mobilnih napravah se je v zadnjih letih znatno izboljšala kot tudi računalniška moč. Novi pametni telefoni so v bistvu majhni in zmogljivi računalniki s sodobnimi zmožnosti omrežja naprav, ki pomenijo, da ostanejo ljudje povezani z omrežjem in tako s svetom ves čas. Trenutno na trgu najpopularnejši operacijski sistemi za pametne telefone so Android, Windows Phone in iOS. [2]

#### 2.1.1 Android

Android je Google izdal leta 2007 kot odprtokodni mobilni operacijski sistem, ki temelji na jedru Linuxa in uporablja jezik Javo. Android tako vključuje mobilni operacijski sistem z razvojnim okoljem, virtualni stroj za zagon aplikacij ter tudi vmesno programsko opremo med kodo in operacijskim sistemom. Android olajša uporabo 2d in 3d grafičnih knjižnic, prilagojene ter vgrajene SQL motorje za vztrajno shranjevanje in napredne omrežne zmogljivosti, kot so 3G, 4G in WLAN. Razvojna orodja za Android so Android Studio in Eclipse. [2]

#### 2.1.2 Windows Phone

V preteklosti se je operacijski sistem Windows Phone, ki ga je ustvaril Microsoft, imenoval Windows Mobile, vendar se je leta 2007 preimenoval. Windows Phone je operacijski sistem za pametne telefone in se uporablja na napravah na dotik ter ponuja funkcionalnosti, kot so omrežna povezava, senzorji in integracija kamere. Obstajata dva jezika, ki se lahko uporabita za pisanje programov Visual Basic .NET in C#. Programi, ustvarjeni za Windows Phone, so pakirani v datoteke XAP, kar je programski paket Silverlight. Razvojno orodje za Windows Phone je Visual Studio. [2]

### 2.1.3 iOS

iOS je operacijski sistem za več naprav Apple, med katerimi je najpomembnejši iPhone. Applov iPhone je bil izdelan leta 2007 in je spremenil trg pametnih telefonov. Vključil je velik zaslon na dotik in, vsaj za tisti čas, impresivno specifikacijo strojne opreme. Aplikacije za iOS so napisane v Objective-C ali Swiftu. Medtem ko sta Java in C# v sintaksi zelo podobna, sta Objective-C in Swift precej drugačna. Razvoj za iOS zahteva računalnik, v katerem je nameščen operacijski sistem Mac OS. Aplikacija, ki se običajno uporablja za iOS aplikacij, je XCode. [2]

## 2.2 Navzkrižna platforma

Navzkrižna platforma je izdelek ali sistem, ki lahko deluje na več različnih platformah ali delovnih okoljih. Različni sistemi navzkrižnih platform vključujejo strojne in programske sisteme ter sisteme z ločenimi gradnjami za vsako platformo, pa tudi druge širše sisteme, ki so zasnovani tako, da delujejo na enak način na več platformah. [5]

Vsaka naprava in operacijski sistem ima svoj programski vmesnik za obravnavanje aplikacij. Upravljanje le-teh na različne načine lahko pomaga IT sistemom učinkovito delovati v različnih okoljih. V mnogih primerih operacije med platformami vključujejo ne le delo z vmesniki aplikacijskega programiranja, temveč tudi z vsemi potrebnimi licenčnimi zahtevami. Odprtokodna programska oprema in operacijski sistemi so zmanjšali uporabo tradicionalne licenčne pogodbe o programski opremi, vendar so številni vrhunski operacijski sistemi in druga okolja še vedno na voljo v okviru tradicionalnih licenc. [5]

Za razvoj navzkrižnih aplikacij se uporabljajo številna okolja, nekatera izmed njih so:

- Apache Cordova;
- Xamarin;
- Appcelator Titanium;
- RAD Studio.

## 3. PRIMERJAVA RAZVOJNEGA OKOLJA RAD STUDIO Z DRUGIMI KONKURENČNIMI OKOLJI

### 3.1 Apache Cordova

Apache Cordova je brezplačen, odprtokodno razvojno okolje za izdelavo navzkrižnih aplikacij. Podpira razvoj aplikacij za Android, iOS in Windows Phone ter uporablja tehnologije HTML5, CSS3 in JavaScript. Za boljšo podporo mobilnih aplikacij implementira zbirko API-jev, ki omogočajo na primer delo s kamero ter upravljanje s stiki. Apache Cordovo sestavljajo naslednje komponente:

- Izvirna koda za vsako od podprtih mobilnih platform;
- Upravljanje vtičnikov;
- Uporabo domorodnih SDK-jev;
- Testiranje aplikacij na različnih emulatorjih za mobilne naprave [8].

### 3.2 Xamarin

Xamarin Studio je prilagojena različica IDE MonoDevelop, ki se lahko uporablja za razvijanje mobilnih aplikacij Android, iOS in Windows Phone. Xamarin Studio je na voljo v operacijskih sistemih OS X in Windows, za razvoj aplikacij uporablja jezik C# ter ima naslednje funkcije:

- Avtomatsko zaključevanje kode;
- Sintaktično poudarjanje;
- Nadzor kode;
- Navodila za pisanje kode;
- Integrirano razhroščevanje za mobilne aplikacije, ki se izvajajo v emulatorjih ali na napravah;
- Podpora za nadzor različic Git in Subversion. [7]

### 3.3 Appcelerator Titanium

Titanium je izdelek podjetja Appcelerator, ki omogoča ustvarjanje mobilnih aplikacij za Android, iOS in BlackBerry v programskem jeziku JavaScript. Podpora za BlackBerry sicer obstaja, vendar ni tako močna kot za Android in iOS. Čeprav Titanium uporablja JavaScript, se za razvoj aplikacij ne uporablja HTML5 in CSS3. Aplikacije se razvija v razvojnem okolju Titanium Studio, ki vsebuje naslednje lastnosti:

- Avtomatsko preverjanje sintakse;
- Avtomatsko zaključevanje kode;
- Podpora razhroščevanja na posamezni napravi
- Izdelava izvirne kode za podprte mobilne naprave
- Podpora za nadzor različic Subversion, Git, Mercurial [9]

### 3.4 RAD Studio

Integrirano razvojno okolje RAD Studio podpira celoten življenjski cikel razvoja, z namenom zagotoviti eno izvorno kodno bazo, ki se jo preprosto prevede in na novo naloži na različne platforme. Omogoča razvijanje aplikacij za platformi iOS in Android ter izbiro jezika C++ ali Pascal (Delphi) in HTML5 za prikaz.

RAD Studio uporablja sodobne prakse objektnega programiranja, razhroščevanje za posamezno platformo posebej in omogoča podporo za nadzor različic Subversion, Git in Mercurial. V razvojnem okolju RAD Studio je na voljo široka paleta orodij in komponent iz skupnosti Delphi in C++, katere se lahko uporabijo pri razvoju mobilnih aplikacij.

Razvoj mobilnih aplikacij je hiter, saj s pomočjo krmilnih elementov za uporabniški vmesnik in z uporabo oblikovanja za več naprav hkrati dosežemo podporo na različnih napravah. Aplikacije, izdelane z RAD Studio, so izvirne v posamezni platformi. Na obeh platformah (iOS in Android) so podprte vse jezikovne funkcije, od najpreprostejših zank in spremenljivk do sodobne generike, anonimnih metod, vzporedno programskih knjižic.

### 3.5 Primerjava heterogenih razvojnih orodij

Za ugotovitev izbire najboljšega razvojnega okolja smo le-te primerjali med seboj glede na različne karakteristike:

- Ceno;
- Programski jezik, ki ga posamezno razvojno okolje uporablja;
- Podporo za mobilne platforme;
- Podporo za nadzor različic.

**Tabela 3.1: Primerjava karakteristik pri razvojnih okoljih**

	Apache Cordova	Xamarin	Appcelerator Titanium	RAD Studio
Cena	0 €	0 €	0 €	> 2000 €
Programski jezik	HTML5, CSS3, JavaScript	C#	JavaScript	Object Pascal, C++
Podpora za mobilne platforme	Android, iOS, Windows Phone	Android, iOS, Windows Phone	Android, iOS, BlackBerry	Android, iOS
Podpora za nadzor različic	Subversion, Git	Subversion, Git	Subversion, Git, Mercurial	Subversion, Git, Mercurial

Ugotovili smo, da je okolje RAD Studio cenovno neugodno v primerjavi z ostalimi okolji. Primerjava cen je sicer za skraćeno različico, kar pomeni, da je za širšo cena še višja. Prav tako ne podpira mobilne platforme Windows Phone kot jih preostala konkurenčna okolja. Glede izbire jezika je odvisno od samega posameznega razvijalca, kateri jezik mu je bolj všeč. Med izbiro Apache Cordovo, Xamarin ter Appcelerator Titanium bi se odločili za Xamarin, saj nam jezik C# bolj leži, medtem ko so ostale karakteristike enake (Tabela 4.1).

## 4. NAČRT IN IZDELAVA TESTNE APLIKACIJE

Da bi ugotovili, ali je razvoj aplikacij boljši ali enako kvaliteten za mobilne platforme z orodjem RAD Studio 10 kot z orodji, ki so specifična za posamezno platformo, smo izdelali prototipno identično aplikacijo v RAD Studio in v domorodnih orodjih za Android in iOS.

### 4.1 Zasnova testne aplikacije

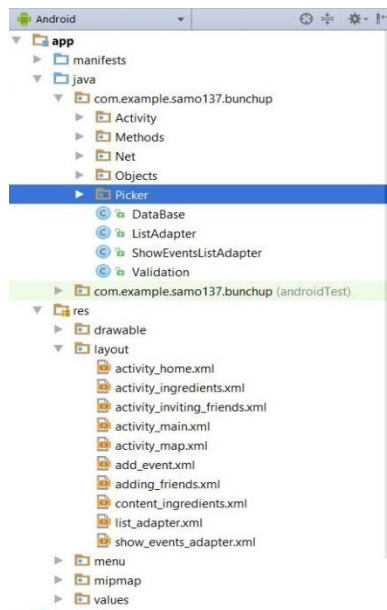
Za prototipno aplikacijo smo izdelali aplikacijo BunchUp, ki bo delovala s komuniciranjem preko Facebooka. Za uporabo le-te bo potrebna povezava uporabnika s Facebookom. Idejna zasnova aplikacije je komuniciranje prijateljev med seboj preko dodajanja različnih dogodkov. Uporabnik lahko doda posamezen dogodek, katerega nato vidijo njegovi prijatelji.

Za ugotavljanje kvalitete razvojnega orodja RAD Studio v primerjavi z domorodnimi orodji za mobilne naprave smo v aplikaciji uporabili naslednje gradnike:

- Facebook povezavo (preko katere pridobivamo podatke od uporabnika in njegove prijatelje);
- Prikaz podatkov na geografski karti (kjer so prikazani dogodki, ki jih je dodal uporabnik ter njegovi prijatelji).
- GPS (pridobivanje uporabnikove lokacije);
- Lokalno shranjevanje podatkov na naprave;
- JSON knjižnico (Podatki se pošiljajo na strežnik in iz strežnika v JSON obliki);

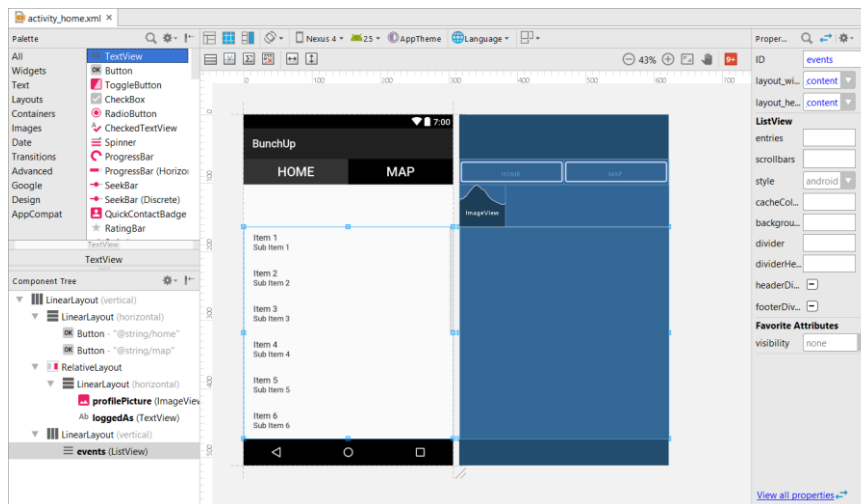
### 4.2 Izdelava aplikacije v Android studio

Izdelava aplikacije v domorodnem orodju za Android ni predstavljal večje ovire, saj je razvojno orodje Android Studio zelo pregledno in dobro organizirano. Uporablja tehnologijo MVC (Model View Controller), zaradi česar ima logiko, povsem ločeno od samega prikaza (Slika 4.1). Za logiko se uporabljajo Java razredi, medtem ko za sam prikaz XML.



**Slika 4.1: MVC in Android Studio**

Na začetku je bilo potrebno izdelati grafični vmesnik. Grafični vmesnik v Android Studioju je možno izdelati grafično (pri čemer povlečemo in izpustimo komponento in se na podlagi tega samodejno generira XML datoteka ali pa direktno pišemo XML značke (Slika 4.2). Za razpored komponent se lahko uporabi LinearLayout ali RelativeLayout. Razlika med njima je, da se LinearLayout lahko prilagaja glede na velikost zaslona (določimo lahko, glede na toali naj so komponente horizontalne ali vertikalne, težnost posameznih komponent, ki se nato razširijo čez celotni zaslon), medtem kot pri RelativeLayout ostanejo komponente vedno na istih pozicijah in se njihova velikost ne spreminja. Zaradi tega je v večini primerov bolje uporabiti LinearLayout, saj so velikosti zaslonov lahko zelo različne, mi pa želimo podpirati čim več različnih mobilnih naprav.



**Slika 4.2: Grafična izdelava uporabniškega vmesnika v Android Studio**

Po izdelavi grafičnega vmesnika je bila potrebna povezava s krmilnikom. Vsak ločen uporabniški vmesnik ima svoj krmilnik. Ko ustvarimo nov Activity, nam Android Studio avtomatsko ustvari XML datoteko in Java krmilnik ter ju poveže med seboj. Lahko to tudi naredimo ročno, tako da ustvarimo nov razred Java, ki podeduje lastnosti od razreda Activity, in nato ustvarimo onCreate metodo, v kateri povemo, katero XML datoteko bo krmilnik uporabljal (Slika 4.3),

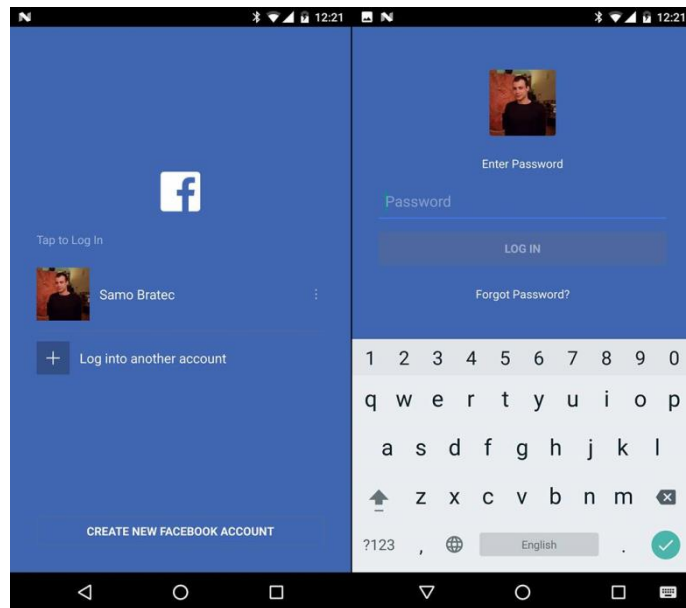
```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_home);
}
```

**Slika 4.3: Metoda onCreate**

#### 4.2.1 Facebook povezava

Za uporabo aplikacije je najprej potrebna prijava, saj je njen namen komuniciranje s prijatelji. Prijavo smo naredili s pomočjo Facebooka, saj je ta način za naš primer enostavnejši, Facebook namreč že vsebuje podatke o uporabniku in njegove prijatelje. Ob prvem zagonu aplikacije se tako avtomatsko prikaže prijava (Slika 4.4). Naprava si to prijavo zapomni, tako da ob vsakem naslednjem zagonu več le-ta ni potrebna.





**Slika 4.4: Facebook prijava v Android Studiu**

Za delovanje takšne prijave je v Android Studiu najprej potrebno inicializirati Facebook SDK (Software Development Kit). Če je bil le-ta uspešno inicializiran, je nato potrebno na LoginManager registrirati povratne metode. Te metode so: onSuccess (prijava je bila uspešna), onCancel (uporabnik je prekinil prijavo), onError (prišlo je do kakršnekoli napake) (Slika 4.5).

```

public void facebookLogin(){
    final Context context = this;
    if (InternetConnection.isNetworkAvailable(this)) {
        facebookSDKInitialize();
        if (FacebookSdk.isInitialized()) {
            List<String> permissions = Arrays.asList("public_profile", "user_friends");
            LoginManager.getInstance().loginWithReadPermissions(this, permissions);
            callbackManager = CallbackManager.Factory.create();
            final Activity activity = this;
            LoginManager.getInstance().registerCallback(callbackManager, new FacebookCallback<LoginResult>() {
                @Override
                public void onSuccess(LoginResult loginResult) {
                    Toast.makeText(context, "Prijava je bila uspešna!", Toast.LENGTH_SHORT).show();
                }
                @Override
                public void onCancel() {
                    Toast.makeText(context, "Preklicali ste prijavo!", Toast.LENGTH_SHORT).show();
                }
                @Override
                public void onError(FacebookException e) {
                    Toast.makeText(context, "Prišlo je do napake, poskusite ponovno!", Toast.LENGTH_SHORT).show();
                }
            });
        }
    }
    else{
        Toast.makeText(context, "Nimate internetne povezave!", Toast.LENGTH_SHORT).show();
    }
}

```

**Slika 4.5: Metoda za Facebook povezavo v Android Studiu**

Ob uspešni prijavi je seveda potrebno shraniti podatke o uporabniku. Za shranjevanje osnovnih podatkov, kot so id, ime in slika, smo implementirali nit, katera nam shrani poleg podatkov tudi sliko uporabnikovega profila na napravo. Za shranjevanje uporabnikovih podatkov smo uporabili SharedPreferences, s pomočjo katerih se podatki shranijo lokalno na napravo in ostanejo shranjeni, dokler jih uporabnik ne izbriše. Uporabili smo tudi Gson knjižnico, katera nam omogoča, da lahko objekt pretvorimo v Json obliko tipa String. To je pomembno, ker v SharedPreferences ni mogoče shranjevati objektov. Ko so podatki shranjeni, se prav tako zažene nov Activity (Slika 4.6).

```

@Override
public void onSuccess(LoginResult loginResult) {
    final Profile actualProfile = Profile.getCurrentProfile();
    final SharedPreferences settings = getSharedPreferences("PREF_NAME", Context.MODE_PRIVATE);
    final SharedPreferences.Editor editor = settings.edit();
    final Gson gson = new Gson();
    Thread thread = new Thread(new Runnable() {
        @Override
        public void run() {
            Bitmap profilePicture = drawableFromUri(String.valueOf(actualProfile.getProfilePictureUri(200, 200)));
            User user = null;
            try {
                user = new User(actualProfile.getId(), actualProfile.getName(), saveToInternalStorage(profilePicture));
            } catch (IOException e) {
                e.printStackTrace();
            }
            String userJson = gson.toJson(user);
            editor.putString("user", userJson);
            editor.commit();
        }
    });
    thread.start();
    Intent intent = new Intent(MainActivity.this, Map.class);
    startActivity(intent);
    overridePendingTransition(0, 0);
    activity.finish();
}

```

Slika 4.6: Shranjevanje uporabnikovih podatkov v Android Studiu

Za pravilno delovanje aplikacije je bilo potrebno tudi shranjevanje vseh prijateljev. Za dosego le-tega smo uporabili GraphRequest, preko katerega smo od Facebook-a pridobili vse uporabnikove prijatelje, katere smo nato, prav tako kot prejšnje podatke, shranili v SharedPreferences (Slika 4.7).

```

new GraphRequest(AccessToken.getCurrentAccessToken(), "/me/friends", null, HttpMethod.GET,
new GraphRequest.Callback() {
    public void onCompleted(GraphResponse response) {
        try {
            JSONArray allFriends = response.getJSONObject().getJSONArray("data");
            ArrayList<User> users = new ArrayList<User>();
            for (int i = 0; i < allFriends.length(); i++) {
                User user = new User(allFriends.getJSONObject(i).getString("id"),
                    allFriends.getJSONObject(i).getString("name"));
                user.setInvite(false);
                users.add(user);
            }
            Friends friends = new Friends(users);
            editor.putString("friends", gson.toJson(friends));
            editor.commit();
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }
}).executeAsync();

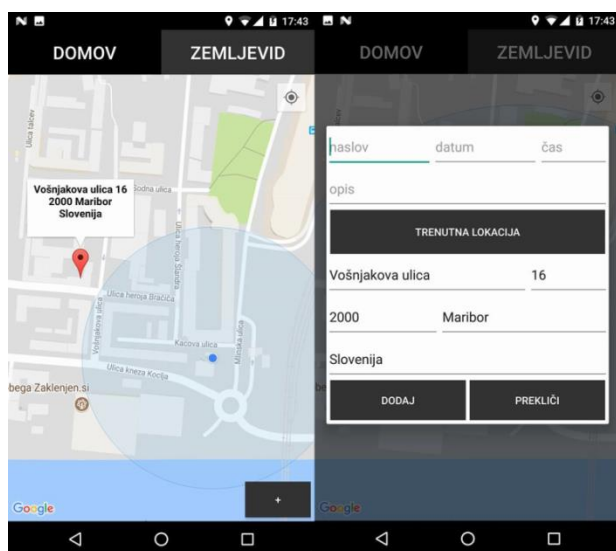
```

Slika 4.7: Shranjevanje uporabnikovih prijateljev v Android Studiu

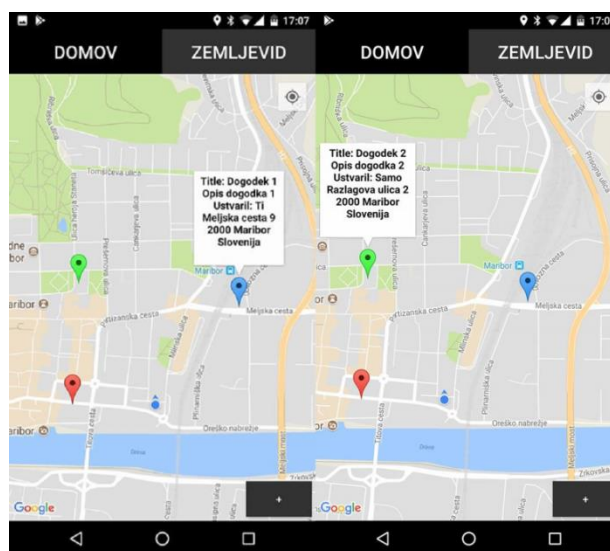
#### 4.2.2 Prikaz podatkov na geografski karti

Za prikaz podatkov na geografski karti smo uporabili API od Googla. Ta nam omogoča prikazati celotni zemljevid Sveta znotraj aplikacije z njihovimi obstoječimi podatki. Zemljevid pomaga uporabnikom, da lahko samo s klikanjem po njem izberejo lokacijo dogodka, ki ga želijo ustvariti, in se ta ob kliku na dodajanje dogodka samodejno vpiše v ustrezna polja (Slika 4.8).

Da bi naredili uporabo zemljevida še bolj praktično, se na njem prikazujejo tudi vsi obstoječi dogodki, ki so aktualni za uporabnika (uporabnikovi in prijateljevi dogodki). Za doseg boljše preglednosti se le-ti prikazujejo v različnih barvah: modra (uporabnikovi dogodki), zelena (prijateljevi dogodki). Ob kliku na dogodek se prav tako izpišejo podatki o njem in lokacija. Na tak način lahko uporabniki hitro in enostavno vidijo lokacije dogodkov, na katere so povabljeni (Slika 4.9).



Slika 4.8: Izbira dogodka na zemljevidu v Android Studiu



Slika 4.9: Prikaz dogodkov na zemljevidu v Android Studiu

Implementacija tega v Android Studiu je bila preprosta. Za uporabo API-ja je bilo najprej potrebno inicializirati GoogleApiClient, kar smo naredili v metodi onCreate, ki se izvede ob samem zagonu Activitya (Slika 4.10). Ostale funkcionalnosti, kot so spreminjanje lokacije ob kliku po zemljevidu (Slika 4.11) in prikaz dogodkov (Slika 4.12), smo implementirali v metodi onMapReady in se izvede takrat, ko je zemljevid pripravljen za uporabo.

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_map);
    mGoogleApiClient = new GoogleApiClient
        .Builder(this)
        .addApi(Places.GEO_DATA_API)
        .addApi(Places.PLACE_DETECTION_API)
        .addConnectionCallbacks(this)
        .addOnConnectionFailedListener(this)
        .build();
}

```

Slika 4.10: Inicializacija GoogleApiClient

```

map.setOnMapClickListener(new GoogleMap.OnMapClickListener() {
    @Override
    public void onMapClick(LatLng point) {
        if (InternetConnection.isNetworkAvailable(context)) {
            String title = location.getStreet() + " " + location.getStreetNumber()+"\n"+location.getPostalCode()+" "
                +location.getCity()+"\n"+location.getCountry();
            if (marker[0] != null) {
                marker[0].remove();
            }
            marker[0] = map.addMarker(new MarkerOptions().position(point).title(title));
            marker[0].showInfoWindow();
        }
    }
});

```

Slika 4.11: Metoda za spreminjanja lokacije ob kliku po zemljevidu v Android Studio

```

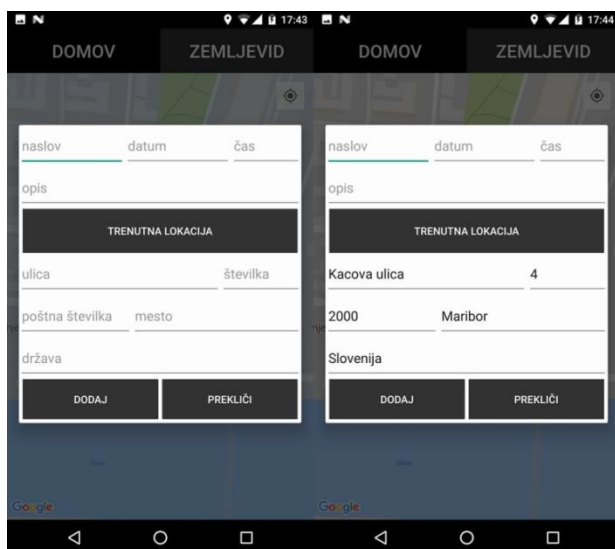
public void setEvents(GoogleMap map) {
    DataBase database=new DataBase(this);
    ArrayList<Event>events=database.getAllEvents();
    Marker[] marker=new Marker[events.size()];
    for (int i=0;i<events.size();i++){
        LatLng coordinates=null;
        try {
            String locationName=events.get(i).getLocation().getStreet()+" "+events.get(i).getLocation().getStreetNumber()+" "
                +events.get(i).getLocation().getPostalCode()+" "+events.get(i).getLocation().getCity()+" "+
                events.get(i).getLocation().getCountry();
            coordinates=getLatLng(locationName);
        } catch (IOException e) {
            e.printStackTrace();
        }
        BitmapDescriptor bitmapDescriptor;
        Gson gson = new Gson();
        User user = gson.fromJson(settings.getString("user", ""), User.class);
        if (events.get(i).getUser().getId().equals(user.getId())) {
            events.get(i).getUser().setName("Ti");
            bitmapDescriptor = BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_AZURE);
        }
        else{
            bitmapDescriptor = BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_GREEN);
        }
        String title="Title: "+events.get(i).getTitle()+"\n"+events.get(i).getDescription()+"\nUstvaril: "+
            events.get(i).getUser().getName()+"\n"+events.get(i).getLocation().getStreet()+" "+
            events.get(i).getLocation().getStreetNumber()+"\n"+events.get(i).getLocation().getPostalCode()
            +" "+events.get(i).getLocation().getCity()+"\n"+events.get(i).getLocation().getCountry();
        marker[i] = map.addMarker(new MarkerOptions().position(coordinates).icon(bitmapDescriptor).title(title));
        marker[i].showInfoWindow();
    }
}

```

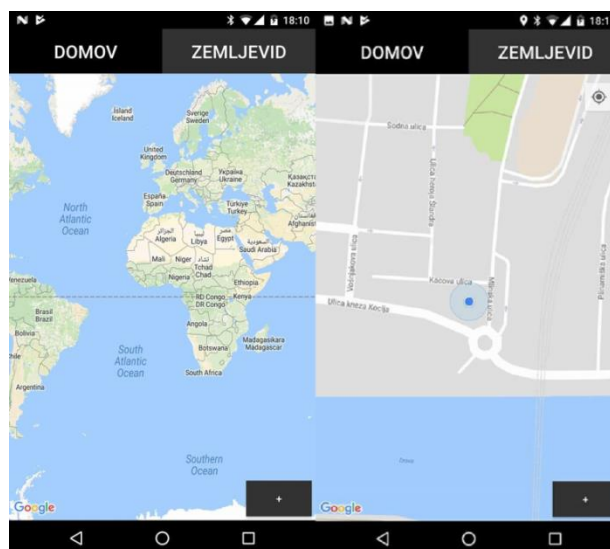
Slika 4.12: Metoda za prikaz dogodkov v Android Studio

### 4.2.3 Določanje položaja uporabnika

GPS smo uporabili s podobnim razlogom kot GoogleMap, da bi uporabniku poenostavili dodajanje dogodkov. Tako uporabnik, če želi dodati dogodek na lokaciji, kjer se trenutno dogaja, lahko enostavno klikne na gumb »TRENUTNA LOKACIJA« in spodaj se bodo izpolnili podatki v ustrezna polja (Slika 4.13). Prav tako se s pomočjo GPSa ob samem zagonu zemljevid centrira na uporabnikovo trenutno lokacijo, in tako mu ni treba iskati svojega mesta (Slika 4.14).



Slika 4.13: Delovanje GPS v Android Studiu



Slika 4.14: Prikaz zemljevida v Android Studiu

Pridobivanje trenutne lokacije deluje preko GPSTrackera, ki vsebuje zemljepisno širino in dolžino. Nato lahko s pomočjo Geocoderja, na podlagi koordinat, pridobimo podatke o lokaciji (naslov, kraj, država) (Slika 4.15).

```

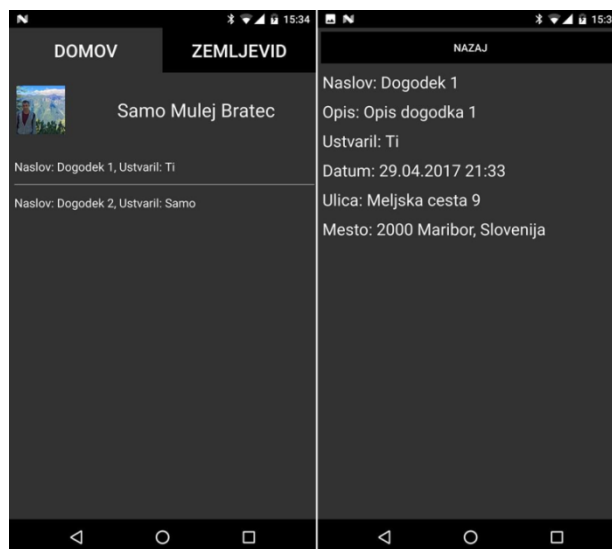
public void currentLocation(View view, Dialog dialog) {
    GPSTracker gps = new GPSTracker(Map.this);
    double latitude = gps.getLatitude();
    double longitude = gps.getLongitude();
    try {
        Geocoder geocoder = new Geocoder(getApplicationContext(), Locale.getDefault());
        List<Address> addresses = geocoder.getFromLocation(latitude, longitude, 1);
        if (addresses.size() > 0) {
            Address address = addresses.get(0);
            location = new Location(address.getThoroughfare(), "", address.getLocality(),
                address.getPostalCode(), address.getCountryName());
        }
        EditText street = (EditText) dialog.findViewById(R.id.street);
        EditText streetNumber = (EditText) dialog.findViewById(R.id.streetNumber);
        EditText city = (EditText) dialog.findViewById(R.id.city);
        EditText postalCode = (EditText) dialog.findViewById(R.id.postalCode);
        EditText country = (EditText) dialog.findViewById(R.id.country);
        street.setText(location.getStreet());
        streetNumber.setText(location.getStreetNumber());
        city.setText(location.getCity());
        postalCode.setText(location.getPostalCode());
        country.setText(location.getCountry());
    } catch (IOException e) {
        Toast.makeText(this, "Ni mogoče pridobiti trenutne lokacije!",
            Toast.LENGTH_SHORT).show();
    }
}

```

Slika 4.15: Metoda za pridobivanje trenutne lokacije v Android Studiu

#### 4.2.4 Lokalno shranjevanje podatkov

Za shranjevanje podatkov na napravi smo v Android Studiu uporabili SQLite. Namen tega je, da se vsi uporabnikovi podatki, naloženi iz strežnika, prenesejo v lokalno podatkovno bazo in se lahko aplikacija uporablja tudi brez internetne povezave, saj se dela z že naloženimi podatki. Hkrati pa tudi optimiziramo samo aplikacijo, saj ni potrebno ves čas dostopati do strežnika (vedno se naložijo samo novi podatki). Seveda prikaz zemljevida ne more delovati brez internetne povezave, zato smo izdelali še en Activity, na katerem so v seznamu prikazani osnovni podatki o dogodku (naslov in kdo ga je ustvaril); ob kliku na posamezni dogodek se prikažejo še preostali podatki (Slika 4.16).



Slika 4.16: Prikaz dogodkov v Android Studiu



Izdelava podatkovne baze z SQLite je precej podobna MySQL. Najprej je potrebno določiti ime podatkovne baze, tabel in njihovih atributov (Slika 4.17) ter nato ustvariti tabele in jih povezati med seboj s pomočjo tujih ključev (Slika 4.18).

```
private final String DB_NAME = "events";
private final int DB_VERSION = 2;
private final String EVENT = "event";
private final String EVENT_ID = "id";
private final String EVENT_REAL_ID = "real_id";
private final String EVENT_TITLE = "title";
private final String EVENT_DESCRIPTION = "description";
private final String EVENT_TIME = "time";
private final String USER = "user";
private final String USER_ID = "id";
private final String FACEBOOK_USER_ID = "facebook_id";
private final String USER_NAME = "name";
private final String USER_EVENT_ID = "user_id";
private final String LOCATION = "location";
private final String LOCATION_ID = "id";
private final String LOCATION_STREET = "street";
private final String LOCATION_STREET_NUMBER = "street_number";
private final String LOCATION_CITY = "city";
private final String LOCATION_POSTAL_CODE = "postal_code";
private final String LOCATION_COUNTRY = "country";
private final String LOCATION_EVENT_ID = "location_id";
private final String FRIENDS = "friends";
private final String USER_FRIENDS_ID = "friends_id";
private final String EVENT_FRIENDS_ID = "event_id";
private final String FRIENDS_INVITE = "invite";
```

**Slika 4.17: Podatkovna baza v Android Studiu**

```
String table="create table " + USER + " ("
+ USER_ID + " integer primary key autoincrement, "+FACEBOOK_USER_ID+" text,"
+ USER_NAME + " text)";
db.execSQL(table);
table="create table " + LOCATION + " ("
+ LOCATION_ID + " integer primary key autoincrement, "+LOCATION_STREET+" text,"
+ LOCATION_STREET_NUMBER + " text," + LOCATION_CITY+ " text," +
LOCATION_POSTAL_CODE+" text,"+LOCATION_COUNTRY+" text)";
db.execSQL(table);
table="create table " + FRIENDS + " ("
+ FRIENDS_ID + " integer primary key autoincrement, "+FRIENDS_INVITE+" integer," +
"+USER_FRIENDS_ID+" integer, "+EVENT_FRIENDS_ID+" integer," +
"FOREIGN KEY("+USER_FRIENDS_ID+") REFERENCES artist("+USER_ID+")" +
", FOREIGN KEY("+EVENT_FRIENDS_ID+") REFERENCES artist("+EVENT_ID+)");
db.execSQL(table);
table="create table " + EVENT + " ("
+ EVENT_ID + " integer primary key autoincrement, "+EVENT_REAL_ID+" integer,"
+ EVENT_TITLE + " text," + EVENT_DESCRIPTION+ " text," +
EVENT_TIME+" integer, "+USER_EVENT_ID+" integer, "+LOCATION_EVENT_ID+" integer," +
"FOREIGN KEY("+USER_EVENT_ID+) REFERENCES artist("+USER_ID+")" +
", FOREIGN KEY("+LOCATION_EVENT_ID+) REFERENCES artist("+LOCATION_ID+)");
db.execSQL(table);
```

**Slika 4.18: Kreiranje tabel v Android Studiu**

Pred začetkom dela s podatki je potrebno odpreti povezavo s podatkovno bazo in jo zapreti na koncu (Slika 4.19).

```
private void close(){
    this.dbManager.close();
}
private void open(){
    try {
        this.dbManager = new DBmanager(context);
        db = this.dbManager.getWritableDatabase();
    } catch (SQLException e) {
        Log.e("DBHelper ", "Error opening db "+e.getMessage());
    }
}
```

**Slika 4.19: Odpiranje in zapiranje povezave s podatkovno bazo v Android Studiu**

Ko to storimo, lahko začnemo delati operacije s tabelami (shranjevanje, branje, brisanje, posodabljanje podatkov). S pomočjo `ContentValues`, v katerega damo ime stolpca in vrednost, lahko v tabelo shranimo podatke (Slika 4.20).

```
private long saveUser(User user) {
    ContentValues initialValues = new ContentValues();
    initialValues.put(FACEBOOK_USER_ID, user.getId());
    initialValues.put(USER_NAME, user.getName());
    long userId = db.insert(USER, null, initialValues);
    return userId;
}
```

**Slika 4.20: Metoda za shranjevanje uporabnika v Android Studiu**

Za branje podatkov iz tabele je potrebno ustvariti SQL stavek, kjer nato pridobimo želene podatke s pomočjo `Cursor`a (Slika 4.21).

```
private User getUser(int userId) throws SQLException {
    User user = new User();
    Cursor cur = db.rawQuery("SELECT * from "+USER+" where "+USER_ID+"="+userId, new String [] {});
    if (cur.moveToFirst()) {
        do {
            String facebookUserId = cur.getString(cur.getColumnIndex(FACEBOOK_USER_ID));
            String name = cur.getString(cur.getColumnIndex(USER_NAME));
            user = new User(facebookUserId, name);
        } while (cur.moveToNext());
    }
    return user;
}
```

**Slika 4.21: Metoda za pridobivanje uporabnika v Android Studiu**

Kot vidimo, je uporaba `SQLite`a v `Android Studiu` zelo preprosta, saj imamo pregledno strukturo in lahko na enostaven način delamo s podatki.

#### 4.2.5 Uporaba JSON-a in povezava s strežnikom

Vsi dogodki, ki jih katerikoli uporabnik doda, so shranjeni na strežniku, vendar se uporabniku naložijo samo njegovi aktualni (dogodki, ki jih je sam dodal, ter dogodki prijateljev). Celotno shranjevanje na strežniku je pomembno, saj v nasprotnem primeru lahko uporabnik izgubi podatke, ki so shranjeni v lokalni podatkovni bazi (lahko ročno počisti podatke na napravi ali ob samem izbrisu aplikacije). Na strežniku so narejene skripte v PHP-ju, do katerih dostopamo v `Android Studiu` s pomočjo `URLConnection`a. Podatki se prenašajo v `JSON`



obliki, saj lahko tako enostavno pošiljamo in beremo podatke tako na strežniku kot tudi na napravi. Za komunikacijo med strežnikom in napravo smo implementirali dve univerzalni metodi. Namen prve metode je shraniti podatke na strežnik (npr. shranjevanje dogodkov) (Slika 4.22); namen druge metode je brati podatke s strežnika (npr. branje dogodkov) (Slika 4.23).

```
public static void saveData (List<Parameter> parameters, String data) {
    try {
        URL url = new URL(Jlink+""+data);
        HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
        urlConnection.setRequestMethod("POST");
        urlConnection.setDoOutput(true);
        urlConnection.setDoInput(true);
        Uri.Builder builder = createUriBuilder(parameters);
        String query = builder.build().getEncodedQuery();
        OutputStream os = urlConnection.getOutputStream();
        BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(os, "UTF-8"));
        writer.write(query);
        writer.flush();
        urlConnection.connect();
        os.close();
        urlConnection.getInputStream();
        writer.close();
        urlConnection.disconnect();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

**Slika 4.22: Metoda za shranjevanje podatkov na strežnik v Android Studiu**

```
public static JSONArray readData (List<Parameter> parameters, String data) {
    try {
        URL url = new URL(Jlink+""+data);
        HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
        urlConnection.setRequestMethod("POST");
        urlConnection.setDoOutput(true);
        urlConnection.setDoInput(true);
        Uri.Builder builder = createUriBuilder(parameters);
        String query = builder.build().getEncodedQuery();
        OutputStream os = urlConnection.getOutputStream();
        BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(os, "UTF-8"));
        writer.write(query);
        writer.flush();
        urlConnection.connect();
        os.close();
        InputStream is = urlConnection.getInputStream();
        try {
            BufferedReader rd = new BufferedReader(new InputStreamReader(is, Charset.forName("UTF-8")));
            String jsonText = readAll(rd);
            JSONArray json = new JSONArray(jsonText);
            return json;
        } catch (JSONException e) {
            e.printStackTrace();
        } finally {
            is.close();
        }
        writer.close();
        is.close();
        urlConnection.disconnect();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return null;
}
```

**Slika 4.23: Metoda za branje podatkov s strežnika v Android Studiu**

Podatke bi lahko pošiljali preko metode GET ali POST. Mi smo se odločili za metodo POST, saj je ta metoda za komunikacijo s strežnikom varnejša. Univerzalnost metod smo dosegli z

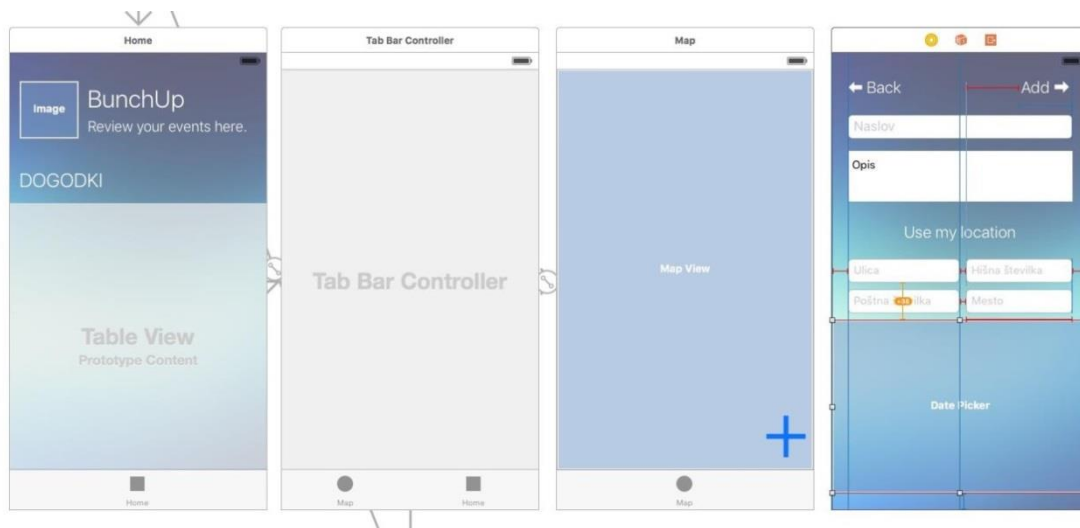
razredom Parameter, ki vsebuje lastnosti ključ in vrednost tega ključa (Slika 4.24). Metodi prejmeta seznam parametrov, katere nato pošljemo strežniku.

```
public class Parameter {  
    private String key;  
    private String value;  
    public Parameter(String key, String value) {  
        this.key = key;  
        this.value = value;  
    }  
    public String getKey() { return key; }  
  
    public void setKey(String key) { this.key = key; }  
  
    public String getValue() { return value; }  
  
    public void setValue(String value) { this.value = value; }  
}
```

Slika 4.24: Razred Parameter

### 4.3 Izdelava aplikacije v xCode

Razvoj iOS aplikacije v orodju Xcode je povzročal večje težave kot Android aplikacije v Android Studio, saj le-ta uporablja svoj jezik Swift, ki nam je manj znan. Sama struktura je podobna kot pri Android Studiu, saj uporablja prav tako MVC tehnologijo. Razlika je v tem, da ima pri Android Studio vsak krmilnik svojo XML datoteko, medtem ko pri Xcodu lahko naredimo Storyboard, v katerem so vsi pogledi združeni v eno datoteko (Slika 4.25).



Slika 4.25: Prikaz izgleda v Storyboardu

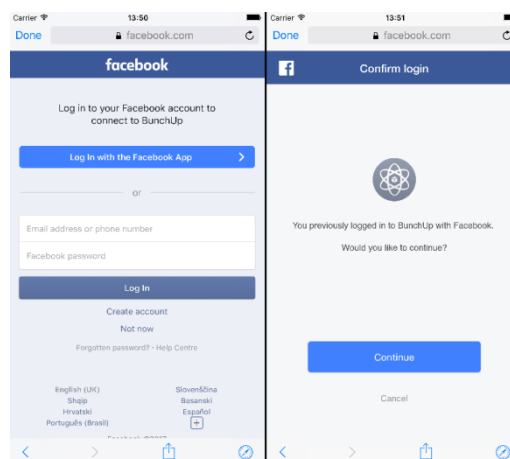
Prav tako kot v Android Studiu lahko za izdelavo grafičnega vmesnika pišemo XML značke, vendar je takšen način pri Xcodu manj pregleden (slabše strukturiran). Pri zagonu Activitya se najprej izvede metoda `viewDidLoad` (Slika 4.26).

```
override func viewDidLoad() {  
    super.viewDidLoad()  
}
```

**Slika 4.26: Metoda `viewDidLoad`**

### 4.3.1 Facebook povezava

V Xcodu smo prav tako naredili povezavo s Facebookom, kjer se je potrebno prijaviti pred začetkom uporabe aplikacije, z namenom pridobiti podatke uporabnika. Najprej je treba vnesti pravilne podatke (elektronski naslov in geslo) in nato še potrditi prijavo preko aplikacije (Slika 4.27).



**Slika 4.27: Prikaz Facebook prijave v Xcodu**

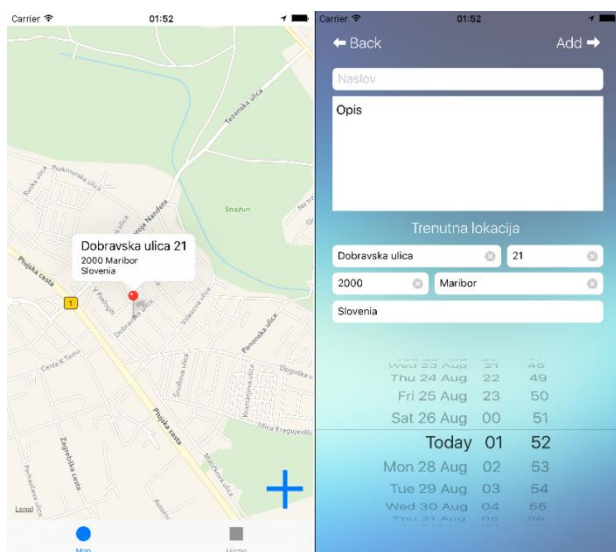
Če je bilo vse uspešno, se uporabnika preusmeri v notranjost aplikacije. Princip je podoben kot pri Android Studiu, razlika pa je v sintaksi (Slika 4.28).



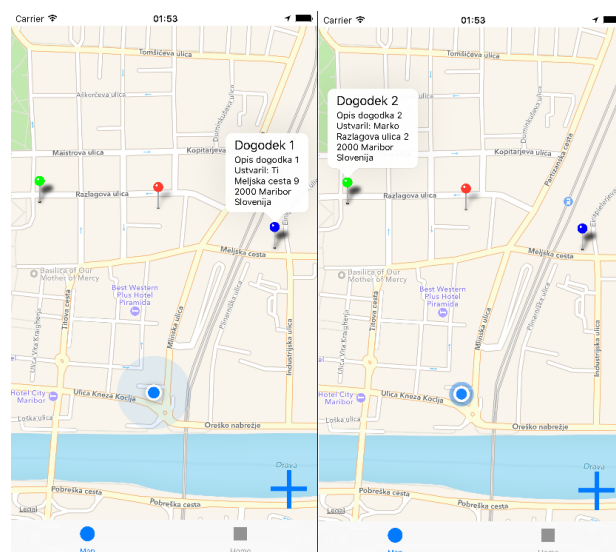
### 4.3.2 Prikaz podatkov na geografski karti

Za prikaz podatkov na geografski karti smo v okolju Xcode uporabili AppleMap, saj je ta način v Xcodu bolje podprt. Na tak način je smo dosegli enako funkcionalnost kot pri Android Studiu. Uporabnik lahko po zemljevidu s klikom izbere lokacijo, ki se nato avtomatsko izpolni v ustrezna polja pri dodajanju dogodkov (Slika 4.31).

Prikaz dogodkov je na zemljevidu v različnih barvah (modra – uporabnikovi dogodki, zelena – prijateljevi dogodki) in ob kliku na posamezni dogodek se izpišejo njegovi podatki (Slika 4.32).



Slika 4.31: Izbira dogodka na zemljevidu v Xcode



Slika 4.32: Prikaz dogodkov na zemljevidu v Xcode

Implementacija prikaza dogodkov je bila malo drugačna. Ker za MKPointAnnotation ne moremo spremeniti privzete barve, smo implementirali metodo, v kateri le-tega pretvorimo v MKPinAnnotationView, kateremu nato spremenimo barvo. Ta metoda nam prav tako omogoči, da ima lahko podnaslov označevalca več vrstic (Slika 4.33).

```

func mapView(mapView: MKMapView, viewForAnnotation annotation: MKAnnotation) -> MKAnnotationView? {
    if annotation is MKUserLocation {
        return nil
    }

    let reuseId = "pin"
    var pinView = mapView.dequeueReusableAnnotationViewWithIdentifier(reuseId) as? MKPinAnnotationView
    if pinView == nil {
        pinView = MKPinAnnotationView(annotation: annotation, reuseIdentifier: reuseId)
        pinView?.canShowCallout = true
        if let colorPointAnnotation = annotation as? ColorPointAnnotation {
            pinView?.pinTintColor = colorPointAnnotation.pinColor
        }
    }
    else {
        pinView?.annotation = annotation
    }
    let subtitleView = UILabel()
    subtitleView.font = subtitleView.font.fontWithSize(12)
    subtitleView.numberOfLines = 0
    subtitleView.text = annotation.subtitle!
    pinView!.detailCalloutAccessoryView = subtitleView
    return pinView
}

```

**Slika 4.33: Metoda za različne barve označevalcev**

Ta metoda se uporabi, ko dodamo označevalca na zemljevid. Za lepšo in preglednejšo kodo smo ustvarili razred `ColorPointAnnotation`, ki podeduje vse lastnosti razreda `MKPointAnnotation` in vsebuje barvo označevalca (Slika 4.34).

```

class ColorPointAnnotation: MKPointAnnotation {
    var pinColor: UIColor
    init(pinColor: UIColor) {
        self.pinColor = pinColor
        super.init()
    }
}

```

**Slika 4.34: Razred ColorPointAnnotation**

Nato smo naredili metodo, ki doda označevalec ob kliku na zemljevid (ko ustvarimo nov označevalec, je potrebno prejšnjega izbrisati) (Slika 4.35), in metodo, ki doda vse uporabnikove in prijateljeve dogodke (Slika 4.36). Razlika med njima je v podatkih, ki se nato prikažejo na označevalcu. Pri prikazu lokacije so potrebni le podatki o njej, medtem ko pri prikazu dogodka potrebujemo še naslov in opis dogodka.



```

func addAnnotation(street: Street, location: CLLocationCoordinate2D){
    let annotation = MKPointAnnotation()
    annotation.coordinate = location
    var ann_title:String!
    var ann_subtitle:String!
    if let street = street.street{
        ann_title = street
    }
    if let streetNumber = street.streetNumber{
        ann_title = ann_title + " " + streetNumber
    }
    if let postalCode = street.city!.postalCode{
        ann_subtitle = postalCode
    }
    if let city = street.city!.city{
        ann_subtitle = ann_subtitle + " " + city
    }
    if let country = street.city!.country{
        ann_subtitle = ann_subtitle + "\n" + country
    }
    annotation.title = ann_title
    annotation.subtitle = ann_subtitle
    if addedAnnotation != nil{
        map.removeAnnotation(addedAnnotation!)
    }
    map.addAnnotation(annotation)
    map.selectAnnotation(annotation, animated: true)
    markerSet = true
    addedAnnotation = annotation
    markedLocation = street
}

```

Slika 4.35: Metoda za spreminjanje lokacije ob kliku po zemljevidu v Xcodu

```

func addEventAnnotation(event: Event, location: CLLocationCoordinate2D){
    var annotation = MKPointAnnotation()
    var ann = event.desc! + "\n"
    ann += "Ustvaril: "
    if defaults!.stringForKey("userid") == event.user?.userid {
        annotation = ColorPointAnnotation(pinColor: UIColor.blueColor())
        ann += "Ti"
    } else {
        annotation = ColorPointAnnotation(pinColor: UIColor.greenColor())
        ann += event.user!.userName!
    }
    ann += "\n"
    annotation.coordinate = location
    let streetObject = event.street!
    annotation.title = event.title
    if let street = streetObject.street{
        ann += street + " "
        if let streetNumber = streetObject.streetNumber{
            ann += streetNumber + "\n"
        }
    }
    if let postalCode = streetObject.city!.postalCode{
        ann += postalCode + " "
    }
    if let city = streetObject.city!.city{
        ann += city + "\n"
    }
    if let country = streetObject.city!.country{
        ann += country
    }
    annotation.subtitle = ann
    map.addAnnotation(annotation)
}

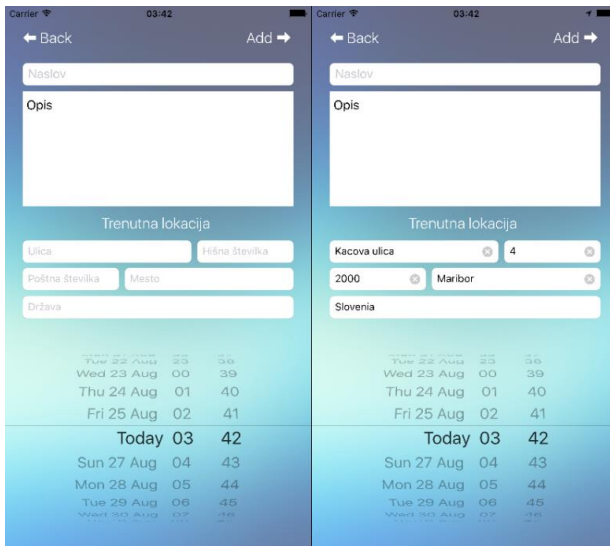
```

Slika 4.36: Metoda za prikaz dogodkov na zemljevidu v Xcodu

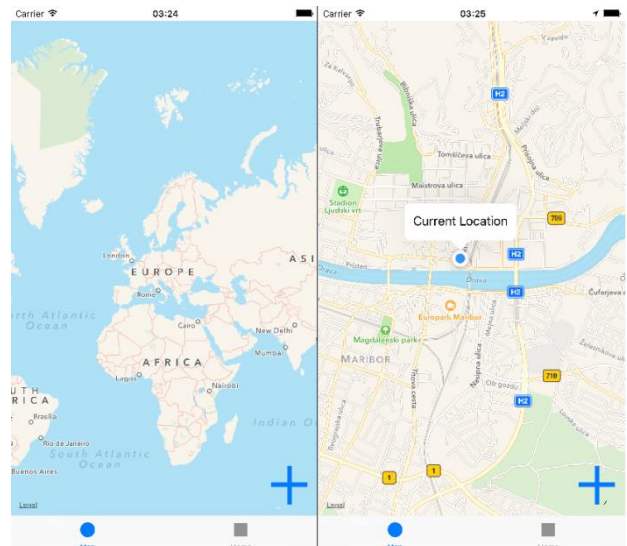
### 4.3.3 Določanje položaja uporabnika

Delovanje GPSa je enako kot pri Android Studiu, se pravi:

- uporabniku se ob kliku na gumb »trenutna lokacija« vnesejo podatki o njegovi lokaciji v ustrezna polja (Slika 4.37);
- zemljevid se ob prikazu centrira na uporabnikovo lokacijo (Slika 4.38).



Slika 4.37: Prikaz delovanja GPS v Xcode



Slika 4.38: Prikaz zemljevida v Xcode

Kot lahko vidite, je delovanje identično kot v Android Studiu. Pridobivanje trenutne lokacije smo implementirali z OneShotLocationManager, ki nam vrne koordinate lokacij (Slika 4.39).

```

var manager: OneShotLocationManager?
@IBAction func useCurrentLocation(sender: AnyObject) {
    manager = OneShotLocationManager()
    manager!.fetchWithCompletion {location, error in
        if let loc = location {
            self.dataService.reverseGeocode(loc, completion: { (street) -> Void in
                if let eventStreet = street{
                    if let eventCity = eventStreet.city!.city{
                        self.eventCity.text = eventCity
                    }
                    if let postalCode = eventStreet.city!.postalCode{
                        self.eventPostalCode.text = postalCode
                    }
                    if let eventStreet = eventStreet.street{
                        self.eventStreet.text = eventStreet
                    }
                    if let eventStreetNumber = eventStreet.streetNumber{
                        self.eventStreetNumber.text = eventStreetNumber
                    }
                    if let eventCountry = eventStreet.city!.country{
                        self.eventCountry.text = eventCountry
                    }
                }
            })
        }
    } else if let err = error {
        print(err.localizedDescription)
    }
    self.manager = nil
}
}

```

**Slika 4.39: Metoda za pridobivanje trenutne lokacije v Xcodu**

Nato je bilo potrebno iz teh koordinat dobiti razumljive podatke za uporabnika (ulica, hišna številka, mesto, poštna številka in država) (Slika 4.44Slika 4.40), katere smo nato izpolnili v ustrezna polja pri dodajanju dogodkov.

```

func reverseGeocode(location: CLLocation, completion: (street: Street?) -> Void){
    let streetObject:Street = Street()
    streetObject.city = City()
    let geoCoder = CLGeocoder()
    let location = CLLocation(latitude: location.coordinate.latitude, longitude: location.coordinate.longitude)
    geoCoder.reverseGeocodeLocation(location){
        (placemarks, error) -> Void in
        let placeArray = placemarks as [CLPlacemark]!
        var placeMark: CLPlacemark!
        placeMark = placeArray?[0]
        if let street = placeMark.addressDictionary?["Thoroughfare"] as? String
        {
            streetObject.street = street
        }
        else{
            if let street = placeMark.addressDictionary?["Name"] as? String{
                streetObject.street = street
            }
        }
        if let street_number = placeMark.addressDictionary?["SubThoroughfare"] as? String
        {
            streetObject.streetNumber = street_number
        }
        if let city = placeMark.addressDictionary?["City"] as? String
        {
            streetObject.city!.city = city
        }
        if let zip = placeMark.addressDictionary?["ZIP"] as? String
        {
            streetObject.city?.postalCode = zip
        }
        if let country = placeMark.addressDictionary?["Country"] as? String
        {
            streetObject.city!.country = country
        }
        completion(street: streetObject)
    }
}
}

```

**Slika 4.40: Metoda, ki pridobi podatke o lokaciji iz koordinat v Xcodu**



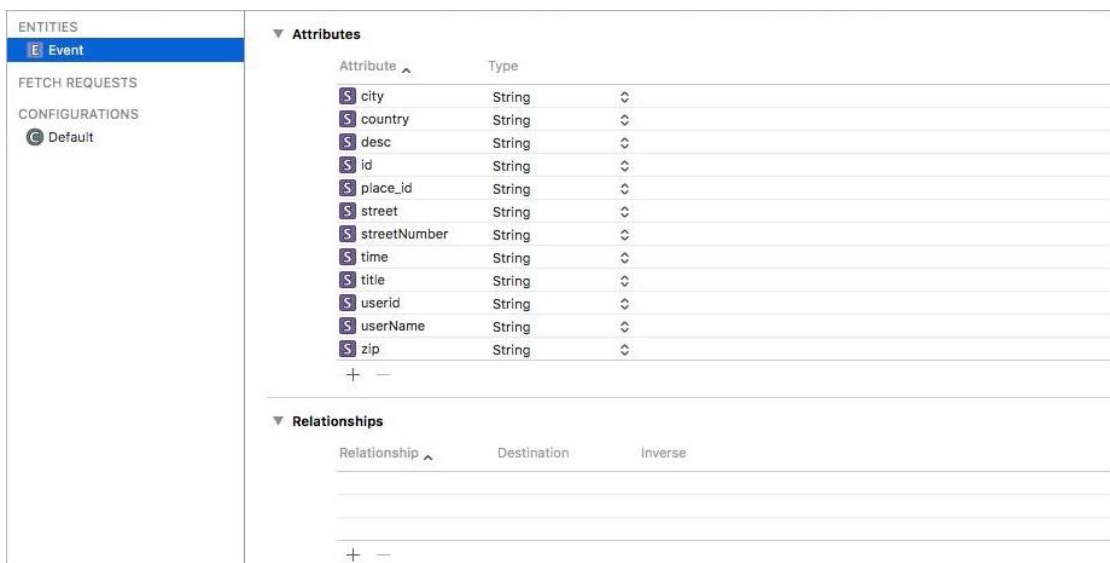
#### 4.3.4 Lokalno shranjevanje podatkov

V Xcodu smo naredili še en ločen prikaz vseh dogodkov, kjer so prikazani osnovni podatki o teh in ob kliku na njega se prikažejo vsi podatki (Slika 4.41). Te podatke črpamo iz lokalne podatkovne baze, zato za prikaz ni potrebna internetna povezava.



Slika 4.41: Prikaz dogodkov v Xcodu

Za lokalno podatkovno bazo je bilo najprej potrebno izdelati podatkovni model (Slika 4.42V podatkovni model smo nato dodali entiteto in vse potrebne atribute (podatke, ki jih želimo shranjevati) in določili njihove tipe.



Slika 4.42: Izdelava podatkovnega modela v Xcodu

Ko je bil podatkovni model ustvarjen, smo začeli delati operacije na njem (shranjevanje in branje podatkov). Naredili smo metodo, ki nam shrani vse podatke o dogodku v podatkovno bazo (Slika 4.43).

```
func saveEvent(e: Event) {
    let appDelegate = UIApplication.sharedApplication().delegate as! AppDelegate

    let managedContext = appDelegate.managedObjectContext
    let entity = NSEntityDescription.entityForName("Event", inManagedObjectContext:managedContext)
    let event = NSManagedObject(entity: entity!, insertIntoManagedObjectContext: managedContext)
    event.setValue(e.user!.userid, forKey: "userid")
    event.setValue(e.user!.userName, forKey: "userName")
    event.setValue(e.id, forKey: "id")
    event.setValue(e.desc, forKey: "desc")
    event.setValue(e.street!.street, forKey: "street")
    event.setValue(e.street!.streetNumber, forKey: "streetNumber")
    event.setValue(e.street!.city!.city, forKey: "city")
    event.setValue(e.street!.city!.postalCode, forKey: "zip")
    event.setValue(e.street!.city!.country, forKey: "country")
    event.setValue(e.title, forKey: "title")
    event.setValue(e.time, forKey: "time")
    do {
        try managedContext.save()
    } catch let error as NSError {
        print("Prišlo je do napake pri shranjevanju podatkov \(error), \(error.userInfo)")
    }
}
```

Slika 4.43: Metoda za shranjevanje dogodkov v Xcodu

Uspešno shranjene podatke smo lahko kadarkoli prebrali iz podatkovne baze, zato smo ustvarili še eno metodo, ki prebere vse podatke o določenem dogodku (Slika 4.44).

```
func setupEvents(){
    let userid = defaults!.stringForKey("userid")
    loadEvents(userid!)
    let coreEvents = CoreDataService.fetchEvents()
    var fetchedEvents: Array<Event> = []
    if let events = coreEvents{
        for coreEvent in events{
            let event:Event = Event()
            event.user = User()
            if let time = coreEvent.valueForKey("time"){
                event.time = time as? String
            }
            if let userid = coreEvent.valueForKey("userid"){
                event.user!.userid = userid as? String
            }
            if let userName = coreEvent.valueForKey("userName"){
                event.user!.userName = userName as? String
            }

            if let title = coreEvent.valueForKey("title"){
                event.title = title as? String
            }

            if let desc = coreEvent.valueForKey("desc"){
                event.desc = desc as? String
            }
            if let id = coreEvent.valueForKey("id"){
                event.id = id as? String
            }
            let streetObject = Street()
            let cityObject = City()
            if let zip = coreEvent.valueForKey("zip"){
                cityObject.postalCode = zip as? String
            }
            if let street = coreEvent.valueForKey("street"){
                streetObject.street = street as? String
            }
            if let streetNumber = coreEvent.valueForKey("streetNumber"){
                streetObject.streetNumber = streetNumber as? String
            }
            if let city = coreEvent.valueForKey("city"){
                cityObject.city = city as? String
            }
            if let country = coreEvent.valueForKey("country"){
                cityObject.country = country as? String
            }
            streetObject.city = cityObject
            event.street = streetObject
            fetchedEvents.append(event)
        }
    }
    self.events = fetchedEvents
    self.eventsTableView.reloadData()
}
```

Slika 4.44: Metoda za branje dogodkov v Xcode

Kot lahko vidite, je delo z lokalno podatkovno bazo v Xcode pregledno in enostavno ter ne zahteva veliko kode za različne operacije.

#### 4.3.5 Uporaba JSON-a in povezava s strežnikom

Da bi dosegli trajno shranjevanje podatkov, le-te shranjujemo na strežnik. Podatke tako kot v Android Studiu prav tako pošiljamo v JSON obliki, saj jih strežnik tako zna najlažje prebrati. Za razliko od dela v Android Studiu, kjer smo uporabili Gson knjižico za pretvorbo v JSON, je bilo v Xcode potrebno ustvariti lastno metodo, ki objekt spremeni v JSON (Slika 4.45). To metodo je bilo potrebno implementirati v vsakem objektu, katerega smo želeli poslati na strežnik.

```
func toJSON() -> Dictionary<String, AnyObject> {
    return [
        "title": self.title!,
        "description": self.desc!,
        "eventDate": self.eventDate!,
        "eventTime": self.eventTime!,
        "street": self.street!.toJSON()
    ]
}
```

Slika 4.45: Pretvorba v JSON obliko v Xcodu

Ko smo implementirali pretvorbo v JSON v vseh razredih, smo lahko naredili povezavo s strežnikom. Najprej smo ustvarili metodo, ki komunicira s strežnikom (pošilja podatke na strežnik in prav tako dobi podatke s njega) (Slika 4.46).

```
func networkFunction(function: String, query: String, completion: (feedback: NSData?) -> Void) {
    let url = "http://www.issadev.com/bunchup/\(function).php?" + query
    print("URL: \(url)")

    let task = NSURLSession.sharedSession().dataTaskWithURL(NSURL(string: url)!, completionHandler: { (responseData, response, error) -> Void in
        if error == nil {
            completion(feedback: responseData)
        } else {
            print(error)
            completion(feedback: nil)
        }
    })
    task.resume()
}
```

Slika 4.46: Metoda za komuniciranje s strežnikom v Xcodu

Za pravilno delovanje je bilo potrebno implementirati še dve metodi:

- prva vrne vse aktualne dogodke s strežnika (Slika 4.47);
- druga shrani dodan dogodek na strežnik (Slika 4.48).

```

dataService.networkFunction("GetEvents", query: "last_update_time=\(last_update_time!)") { (feedback) -> Void in
    let json = JSON(data: feedback!)
    for (_, obj) in json {
        if let title = obj["title"].string{
            let event = Event()
            event.street = Street()
            event.street?.city = City()
            event.title = title
            event.user = User()
            if let desc = obj["description"].string{
                event.desc = desc
            }
            if let street = obj["street"]["street"].string{
                event.street!.street = street
            }
            if let number = obj["street"]["streetNumber"].string{
                event.street!.streetNumber = number
            }
            if let city = obj["street"]["city"]["city"].string{
                event.street!.city!.city = city
            }
            if let zip = obj["street"]["city"]["postalCode"].string{
                event.street!.city!.postalCode = zip
            }
            if let country = obj["street"]["city"]["country"].string{
                event.street!.city!.country = country
            }
            if let userid = obj["user"]["id"].string{
                event.user?.userid = userid
            }
            if let userName = obj["user"]["name"].string{
                event.user!.userName = userName
            }
            event.id = obj["id"].string!
            event.time = obj["eventDate"].string! + " " + obj["eventTime"].string!
            var exists = false
            if let coreEvents = self.coreDataService.fetchEvents(){
                for coreEvent in coreEvents{
                    if coreEvent.valueForKey("id")! as? String == event.id{
                        exists = true
                    }
                }
            }
            if !exists{
                self.coreDataService.saveEvent(event)
            }
        }
    }
}
}
}

```

Slika 4.47: Metoda za pridobivanje podatkov iz strežnika v Xcodu

```

func addEvent(event: Event, time: Int) -> Bool{
    let defaults = NSUserDefaults.standardUserDefaults()
    let userid:String?

    if let id = defaults.stringForKey("userid") {
        userid = id
    }
    else{
        userid = "0"
    }
    let date = NSDate()
    let calendar = NSCalendar.currentCalendar()
    let components = calendar.components([.Day, .Month, .Year, .Hour, .Minute], fromDate: date)
    let updatedDate = URLEncode("\(components.year)-\(components.month)-\(components.day) \(components.hour):\(components.minute)")
    do {
        let JSONPayload: NSData = try NSJSONSerialization.dataWithJSONObject(event.toJSON(), options: [])
        let jsonString = NSString(data: JSONPayload, encoding: NSUTF8StringEncoding) as! String
        let bodyData = "event=\(jsonString)&updatedDate=\(updatedDate)"
        let url1 = NSURL(string: "http://www.issadev.com/bunchup/SaveEvent")
        let request = NSMutableURLRequest(URL:url1!)
        request.HTTPMethod = "POST"
        request.HTTPBody = bodyData.dataUsingEncoding(NSUTF8StringEncoding);
        NSURLConnection.sendAsynchronousRequest(request, queue: NSOperationQueue.mainQueue())
        {
            (response, data, error) in
            let json = JSON(data: data!)
            event.id = json.stringValue
            event.time = String(time)
            event.user!.userid = userid
            self.coreDataService.saveEvent(event)
        }
    } catch {
        let err = error as NSError
        NSLog("\(err.localizedDescription)")
    }
    return true
}
}

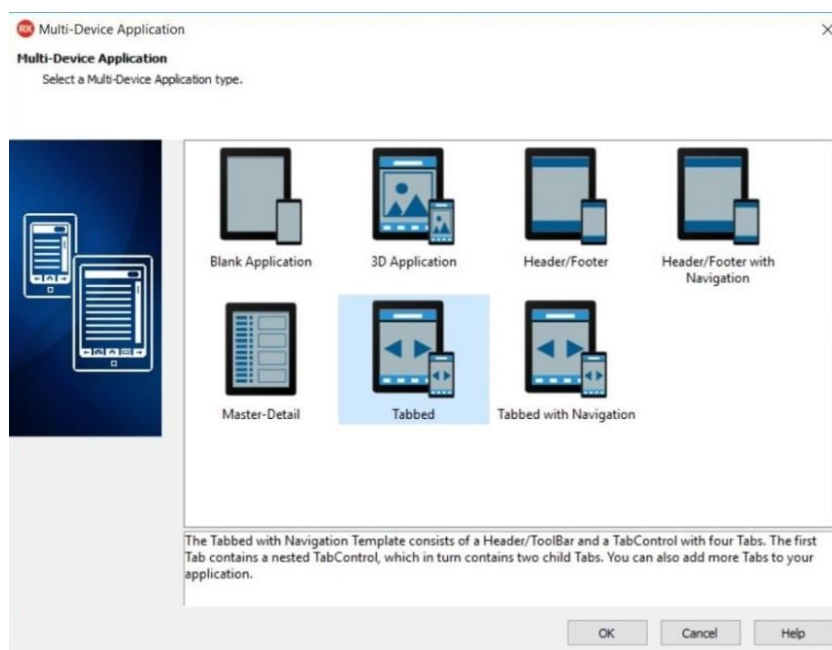
```

Slika 4.48: Metoda za shranjevanje dogodka na strežnik v Xcodu

Implementacija le-tega je bila prav tako preprosta kot v Android Studiu.

#### 4.4 Izdelava testne aplikacije v Rad studio

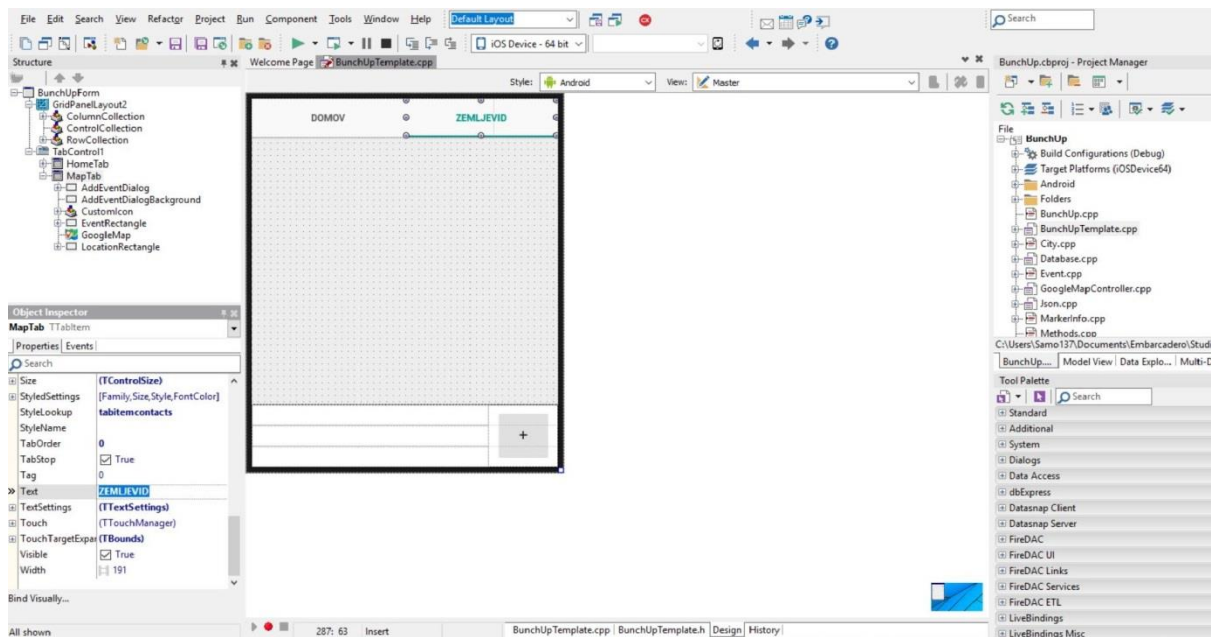
Za začetek izdelave aplikacije v okolju RAD Studio smo si morali izbrati jezik za implementacijo. Izbrali smo si C++, saj nam je ta bolj poznan od Pascala. Nato smo ustvarili projekt. Pri ustanovitvi projekta smo lahko zbirali med različnimi tipi aplikacije. Pravilna izbira nam olajša delo, saj se začetno ogrodje za aplikacijo naredi samo. Izbrali smo Aplikacijo z zavihki (Tabbed), saj smo želeli aplikacijo narediti podobno kot v Android Studiu in XCodu, da bi najbolje ugotovili razlike med njimi (Slika 4.49).



Slika 4.49: Postavitev projekta v RAD Studiu

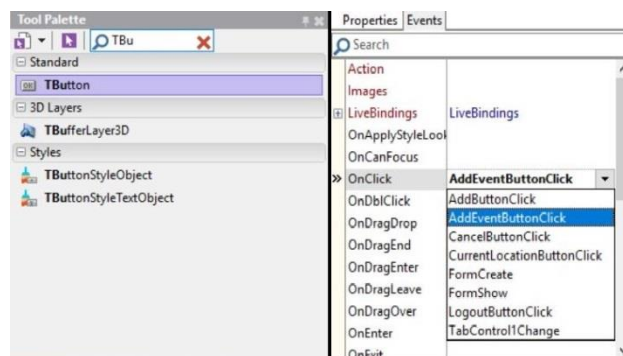
Uporaba razvijalnega okolja RAD Studio je videti preprosta, saj imamo na desni strani datoteke projekta ter komponente, ki jih lahko uporabimo, na levi pa komponente uporabniškega vmesnika ter nastavitve videza le-tega. Medtem ko v Android Studiu in XCodu imamo izbiro, če bomo uporabniški vmesnik izdelali grafično ali z XML, pri RAD Studiu lahko le-tega izdelamo samo grafično, je izdelava preprosta (Slika 4.50).





Slika 4.50: Videz razvijalnega okolja RAD Studio

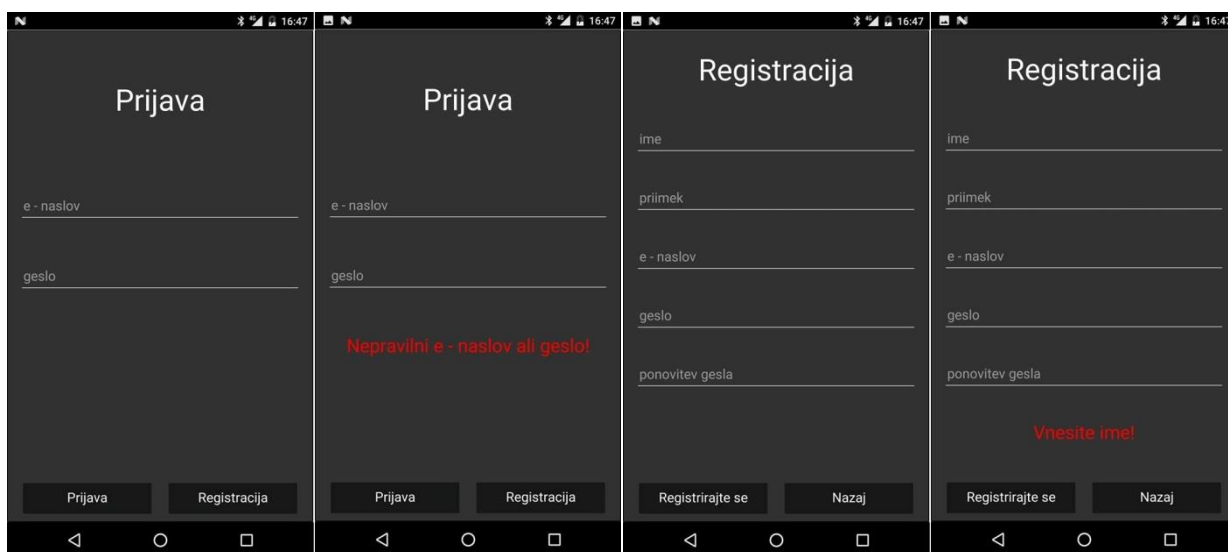
Posamezna datoteka je sestavljena iz oblikovanja (.fmx), vmesnika (.h) in C++ kode (.cpp). Vsako novo ustvarjeno metodo je potrebno implementirati v razredu in jo prav tako dodati v vmesnik. RAD Studio vsebuje veliko komponent, ki jih lahko najdemo z iskanjem in jih nato vstavimo v naš uporabniški vmesnik (Slika 4.51). Na komponente v uporabniškem vmesniku lahko dodajamo dogodke, ki se sprožijo ob ustrezni situaciji. Na primer OnClick dogodek se sproži ob kliku na komponento. Potrebno je izbrati metodo, ki se sproži ob dogodku (Slika 4.51).



Slika 4.51: Dodajanje komponent in dogodkov v RAD Studiu

#### 4.4.1 Prijava

V razvijalnem okolju RAD Studio z uporabo jezika C++ povezava s Facebookom ni dobro podprta, zato smo implementirali lastno prijavo. Za uporabnike je ta način zahtevnejši, saj je potrebno za prijavo vpisati e-naslov in geslo ter predhodno ustvariti račun. Pri prijavi s Facebookom to ni potrebno, saj so uporabniki že registrirani. Ustvarili smo okno s prijavo, kjer se ob nepravem vpisu podatkov izpiše napaka (Slika 4.52). To okno se zažene ob prvem zagonu aplikacije. Ob uspešni prijavi se ta shrani in zato ob vsakem naslednjem zagonu prijava ni potrebna. Če uporabnik še ni ustvaril računa, to stori v okviru registracije (Slika 4.53). Pri registraciji se mu prav tako ob vsakem nepravilnem vnosu izpiše napaka.



Slika 4.52: Prijava v RAD Studiu

Slika 4.53: Registracija v RAD Studiu

Za implementacijo prijave smo najprej ustvarili razred (Slika 4.54) in vmesnik (Slika 4.55) User, ki vsebujeta vse potrebne podatke o uporabniku. V vmesniku smo določili uporabljene spremenljivke, konstruktorje in Get metode, potem pa smo jih v razredu implementirali. Razred vsebuje lastnosti ime, priimek, elektronski naslov in geslo. To so podatki, ki jih o uporabniku potrebujemo.

```

#pragma hdrstop
#include "User.h"
#include "Methods.h"
//-----
#pragma package (smart_init)
CUser::CUser()
{
    this->Email = "";
    this->Password = "";
    this->Login = false;
}
CUser::CUser(String Name, String Surname, String Email, String Password)
{
    this->Name = Name;
    this->Surname = Surname;
    this->Email = Email;
    this->Password = Password;
}
String CUser::GetName()
{
    return this->Name;
}
String CUser::GetSurname()
{
    return this->Surname;
}
String CUser::GetEmail()
{
    return this->Email;
}
String CUser::GetPassword()
{
    return this->Password;
}

```

Slika 4.54: Razred User

```

#ifndef UserH
#define UserH
#include <string>
#include <System.JSON.hpp>
class CUser
{
private:
    String Name;
    String Surname;
    String Email;
    String Password;
public:
    CUser();
    CUser(String Name, String Surname, String Email, String Password);
    String GetName();
    String GetSurname();
    String GetEmail();
    String GetPassword();
    String GetRepeatPassword();
};
//-----
#endif

```

Slika 4.55: Vmesnik User

Nato smo implementirali metodo, ki se sproži ob kliku na prijavo (Slika 4.56). Njen namen je preverjanje pravilnosti uporabniškega vnosa. Le-to deluje preko strežnika. Pri uspešni prijavi se podatki o uporabniku shranijo v napravo in se uporabnika preusmeri v notranjost aplikacije. Odpiranje novega okna v RAD Studiu dosežemo z metodo Show() (Slika 4.57).

```

void __fastcall TLoginForm::LoginButtonClick(TObject *Sender)
{
    TJsonForm *JsonForm = new TJsonForm(this);
    String email = EmailEdit->Text;
    String password = PasswordEdit->Text;
    CUser *User = new CUser(email, password);
    JsonForm->CheckLogin(User, this);
}

```

Slika 4.56: Metoda za preverjanje prijave uporabnika v RAD Studiu

```

void __fastcall TLoginForm::Login(CUser User)
{
    if (User.IsLogin()) {
        TDatabaseForm *DatabaseForm = new TDatabaseForm(this);
        DatabaseForm->SaveUser(User);
        LoginForm->Hide();
        BunchUpForm->Show();
    } else {
        ErrorText->Text = "Nepravilni e - naslov ali geslo!";
    }
}

```

Slika 4.57: Metoda za uspešno prijavo v RAD Studiu

Za registracijo smo implementirali metodo, ki shrani uporabnika na strežniku (Slika 4.58). Pri registraciji preverjanje podatkov ne poteka na strežniku tako kot pri prijavi, saj še podatki o uporabniku niso naloženi nanj. Preverja se, ali so vsa polja izpolnjena, ali je elektronski naslov pravilno vpisan (vsebuje '@' in '.') ter ali se gesli ujemata (Slika 4.59).



```

void __fastcall TRegistrationForm::RegisterButtonClick(TObject *Sender)
{
    String Name = NameEdit->Text;
    String Surname = SurnameEdit->Text;
    String Email = EmailEdit->Text;
    String password = PasswordEdit->Text;
    String repeatPassword = RepeatPasswordEdit->Text;
    CUser *User = new CUser(Name, Surname, Email, password, repeatPassword);
    String Error = this->Validate(*User);
    if (Error == "") {
        TJsonForm *JsonForm = new TJsonForm(this);
        JsonForm->SaveUser(*User);
        RegistrationForm->Hide();
        LoginForm->Show();
    } else {
        ErrorText->Text = Error;
    }
}

```

**Slika 4.58: Metoda za registracijo v RAD Studiu**

```

String __fastcall TRegistrationForm::Validate(CUser User)
{
    String error = "";
    if (User.GetName() == "") {
        error = "Vnesite ime!";
    } else if (User.GetSurname() == "") {
        error = "Vnesite priimek!";
    } else if (User.GetEmail() == "") {
        error = "Vnesite e - naslov!";
    } else if (!this->IsValidEmail(User.GetEmail())) {
        error = "Nepravilni e - naslov!";
    } else if (User.GetPassword() == "") {
        error = "Vnesite geslo!";
    } else if (User.GetPassword() != User.GetRepeatPassword()) {
        error = "Gesli se neujemata!";
    }
    return error;
}

```

**Slika 4.59: Metoda za validacijo registracije v RAD Studiu**

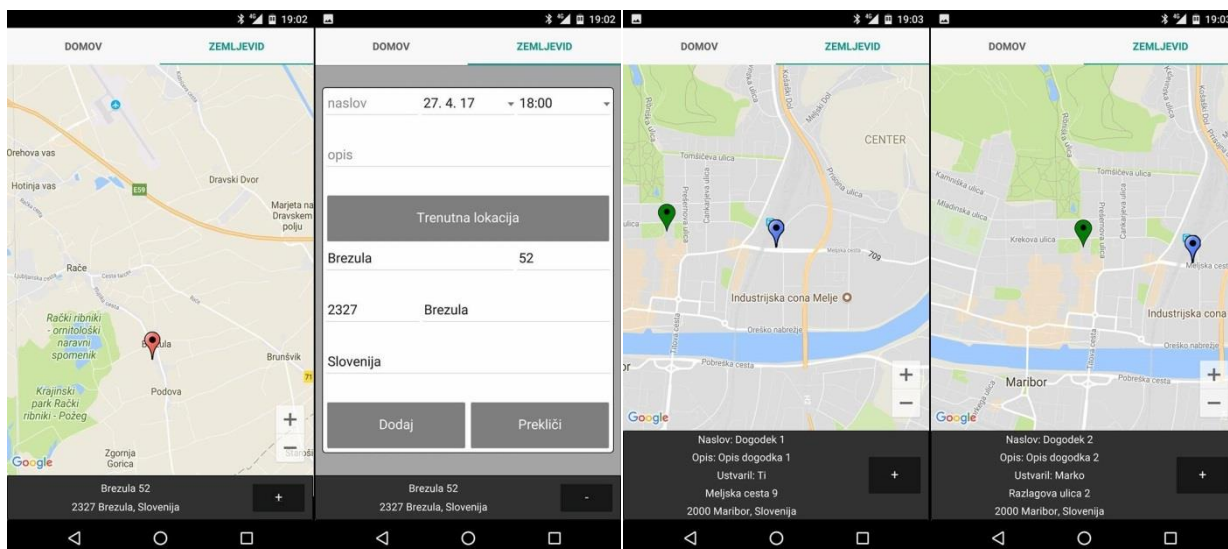
Implementacija prijave v RAD Studiu je sicer preprosta, vendar ni za uporabnike tako enostavna kot v Android Studiu in XCodu. Tam smo lahko enostavno implementirali Facebook povezavo, a implementacija tega je težavnejša.

#### 4.4.2 Prikaz podatkov na geografski karti

Prikaz podatkov na geografski karti v razvijalnem okolju RAD Studio je sicer preprost, vendar se oteži pri dodajanju različnih označevalcev. V Android Studiu in XCodu smo lahko spremenili barvo označevalca, medtem ko smo v RAD Studiu za dosego tega morali vstaviti slike označevalcev v različnih barvah. Zaradi tega se opazi razlika v obliki označevalca med modrim in zelenim označevalcem. Tako kot v prejšnjih dveh orodij so v modri barvi prikazani uporabnikovi dogodki, a v zeleni prijateljevi (Slika 4.61).

Prav tako so nastale težave pri pisanju vsebine v označevalec, zato smo naredili prikaz podatkov pod zemljevidom. Ker je ob kliku na dogodek potrebno prikazati več podatkov, se kvadrataček za prikaz podatkov poveča (Slika 4.60 in Slika 4.61).

Podobno kot v Android Studiu in XCodu nam je uspelo implementirati prekopiranje podatkov o označevalcu v ustrezna vnosna polja pri dodajanju dogodkov, vendar smo naleteli na težavo pri samem dialogu dodajanja dogodka. Gumbi na samem dialogu so definirani črne barve, vendar na njihovo prikazano barvo vpliva ozadje dialogov, kar je v našem primeru bela barva in so zaradi tega prikazani v svetlejši barvi (Slika 4.60).



Slika 4.60: Izbira dogodka na zemljevidu v RAD Studiu

Slika 4.61: Prikaz dogodkov na zemljevidu v RAD Studiu

RAD Studio vsebuje komponento TMapView, katero je potrebno za uporabo povleči v uporabniški vmesnik. Za prikaz GoogleMap je potrebno vnesti API ključ v nastavitve projekta, katerega dobimo od Googla in s tem je prikaz zemljevida v aplikaciji narejen.

Za začetek je bilo potrebno implementirati dodajanje dogodkov na zemljevid. Za ta namen smo implementirali metodo, katera za vsak dogodek posebej doda označevalec na zemljevid (Slika 4.62).

```
void __fastcall TBunchUpForm::AddEvents()
{
    TDatabaseForm *DatabaseForm = new TDatabaseForm(this);
    DynamicArray<CEvent> Events = DatabaseForm->GetEvents();
    for (int i = 0; i < Events.Length; i++) {
        CEvent Event = Events.operator[](i);
        CCurrentEvent = Event;
        CStreet Street = Event.GetStreet();
        CUser User = Event.GetUser();
        CCity City = Street.GetCity();
        TCivicAddress *Address = new TCivicAddress();
        Address->Thoroughfare = Street.GetStreet();
        Address->SubThoroughfare = Street.GetStreetNumber();
        Address->PostalCode = City.GetPostalCode();
        Address->Locality = City.GetCity();
        Address->CountryName = City.GetCountry();
        GoogleMapControllerForm->SetGeocoder(Address, false, User);
    }
    GoogleMapControllerForm->SetGpsActive();
}
```

Slika 4.62: Metoda za dodajanje dogodkov na zemljevid v RAD Studiu

Ker smo za prikaz dogodkov na zemljevidu potrebovali koordinate naslova, v dogodkih pa so shranjeni naslovi (ulica, hišna številka, kraj, poštna številka in država), smo morali implementirati metodi, ki naslov pretvorita v koordinate. Prva metoda prejme kot

parameter naslov (Slika 4.63) ter je v njej potrebno določiti drugo metodo, ki prejme kot parameter koordinate (Slika 4.64). To pretvorbo nam omogoča Geocoder.

```
void __fastcall TGoogleMapControllerForm::SetGeocoder(TCivicAddress* const Address,
    bool CenterPosition, CUser UserData)
{
    Center = CenterPosition;
    User = UserData;
    if (Geocoder == NULL) {
        Geocoder = (TGeocoder*)new TGeocoderClass(TGeocoder::Current);
    }
    if (Geocoder != NULL && Geocoder->OnGeocode == NULL) {
        Geocoder->OnGeocode = OnGeocodeEvent;
    }
    if (Geocoder != NULL && Geocoder->OnGeocode != NULL) {
        Geocoder->Geocode(Address);
    }
}
```

Slika 4.63: Metoda, ki pretvori naslov v koordinate v RAD Studiu

```
void __fastcall TGoogleMapControllerForm::OnGeocodeEvent(DynamicArray<TLocationCoord2D> Coords)
{
    if (Coords.get_length() > 0) {
        TLocationCoord2D Location = Coords.operator[] (0);
        TMapCoordinate Position = TMapCoordinate::Create(Location.Latitude, Location.Longitude);
        BunchUpForm->AddEventMarker(Position, Center, User);
    }
}
```

Slika 4.64: Metoda, ki prejme kot parametre koordinate, pretvorjene iz naslova v RAD Studiu

Ko smo pridobili lokacije iz naslova, smo lahko klicali metodo, ki nam doda označevalec na zemljevid (Slika 4.65). V tej metodi se tudi določi, ali se bo prikazal zeleni ali modri označevalec.

```
void __fastcall TBunchUpForm::AddEventMarker(TMapCoordinate Position, bool centerPosition, CUser User)
{
    TMapMarkerDescriptor MarkerDescriptor = TMapMarkerDescriptor::Create(Position, "");
    TDatabaseForm *Database = new TDatabaseForm(this);
    CUser LoginUser = Database->GetLoginUser();
    if (User.GetEmail() == LoginUser.GetEmail()) {
        MarkerDescriptor.Icon = BitmapBlueMarker;
    } else {
        MarkerDescriptor.Icon = BitmapGreenMarker;
    }
    MarkerDescriptor.Visible = true;
    CMarkerInfo *MarkerInfo = new CMarkerInfo(MarkerDescriptor, CurrentEvent);
    MarkersInfo->set_length(MarkersInfo->get_length() + 1);
    MarkersInfo[0][MarkersInfo->get_length() - 1] = *MarkerInfo;
    GoogleMap->AddMarker(MarkerDescriptor);
    if (centerPosition) {
        GoogleMap->Location = Position;
    }
}
```

Slika 4.65: Metoda, ki doda označevalec za posamezni dogodek na zemljevid v RAD Studiu

Ko smo naredili prikaz vseh dogodkov na zemljevidu, smo se lotili prikaza podatkov ob kliku na posamezni označevalec. Tako smo ustvarili metodo, ki preverja, ali gre za klik na dogodek ali za samo klik na zemljevidu, posledično se prikažejo ustrezni podatki (Slika 4.66).

```
void __fastcall TBunchUpForm::GoogleMapMarkerClick(TMapMarker *Marker)
{
    ActiveMarker = Marker;
    if (Marker->Descriptor.Icon == BitmapRedMarker) {
        EventRectangle->Visible = false;
        LocationRectangle->Visible = true;
        GoogleMapControllerForm->SetGeocoder(Marker->Descriptor.Position);
    } else {
        if (CurrentMarker != NULL) {
            CurrentMarker->Remove();
        }
        GoogleMap->Margins->Bottom = 120;
        EventRectangle->Visible = true;
        LocationRectangle->Visible = false;
        CEvent Event = Methods::GetEventFromMarker(Marker, MarkersInfo);
        CStreet Street = Event.GetStreet();
        CCity City = Street.GetCity();
        CUser User = Event.GetUser();
        EventTitleText->Text = "Naslov: " + Event.GetTitle();
        EventDescriptionText->Text = "Opis: " + Event.GetDescription();
        TDatabaseForm *Database = new TDatabaseForm(this);
        CUser LoginUser = Database->GetLoginUser();
        if (User.GetEmail() == LoginUser.GetEmail()) {
            EventCreatedByText->Text = "Ustvaril: Ti";
        } else {
            EventCreatedByText->Text = "Ustvaril: " + User.GetName();
        }
        EventStreetText->Text = Street.GetStreet() + " " + Street.GetStreetNumber();
        EventCityText->Text = City.GetPostalCode() + " " + City.GetCity() + ", " + City.GetCountry();
    }
}
```

Slika 4.66: Metoda, ki se izvede ob kliku na označevalec v RAD Studiu

Za dodajanje dogodkov smo naredili metodo, ki naredi le-tega vidnega, hkrati pa povzroči nevidnost zemljevida. To je bilo potrebno zato, ker je v RAD Studiu prikaz zemljevida vedno v ospredju in ni možno prikazati ostalih komponent na njem. Metoda prav tako pridobi vse podatke o naslovu iz trenutnega označevalca in jih vnese v ustrezna polja (Slika 4.67).

```
void __fastcall TBunchUpForm::ShowAddEventDialog()
{
    if (CurrentMarker == NULL || CurrentMarker != ActiveMarker) {
        CStreet Street = Methods::GetStreetFromMarker(ActiveMarker, MarkersInfo);
        CCity City = Street.GetCity();
        StreetEdit->Text = Street.GetStreet();
        StreetNumberEdit->Text = Street.GetStreetNumber();
        PostalCodeEdit->Text = City.GetPostalCode();
        CityEdit->Text = City.GetCity();
        CountryEdit->Text = City.GetCountry();
    } else {
        GoogleMapControllerForm->SetGeocoderAddEvent(CurrentMarker->Descriptor.Position);
    }
    AddEventDialog->Margins->Bottom = GoogleMap->Margins->Bottom + 30;
    AddEventButton->Text = "-";
    AddEventDialogBackground->Visible = true;
    AddEventDialog->Visible = true;
    GoogleMap->Visible = false;
}
```

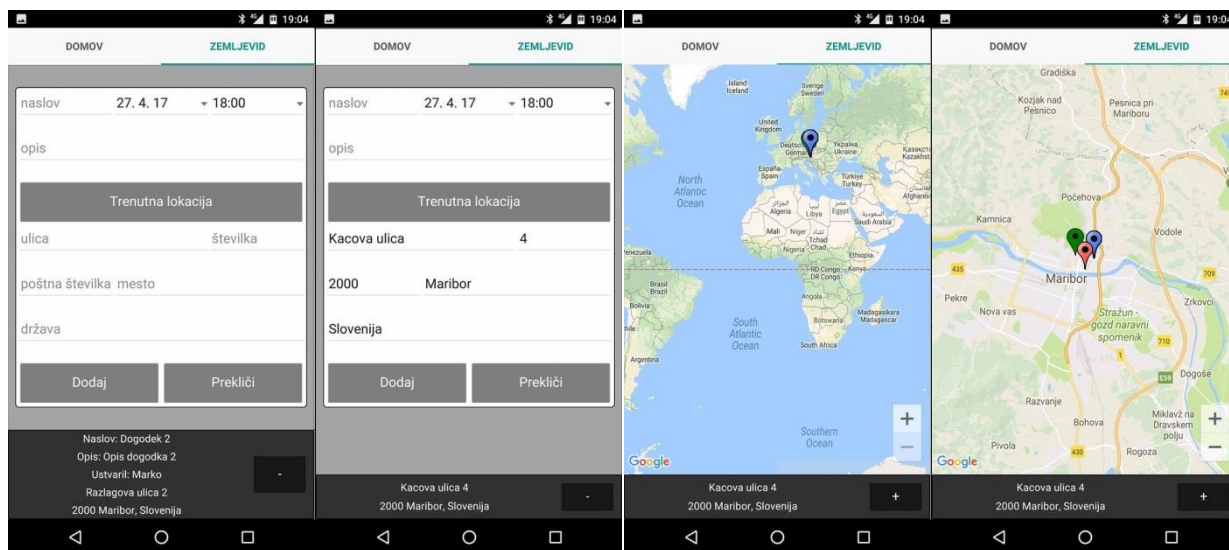
Slika 4.67: Metoda, ki prikaže dialog za dogajanje dogodkov in ustrezno vnese podatke o naslovu

Za implementacijo GoogleMap v RAD Studiu smo potrebovali več časa kot pri Android Studiu in Xcodu in tudi kvaliteta je v RAD Studiu slabša, saj smo naleteli na precej težav.

#### 4.4.3 Določanje položaja uporabnika

Delovanje GPS-a nam je v RAD Studiu uspelo približati Android Studiu in Xcodu, z razliko, da ni tako optimizirano. Namreč, zgodi se lahko, da kdaj porabi več časa in zaradi tega se lahko zaustavi delovanje aplikacije. Pri dodajanju dogodkov ima uporabnik možnost klikniti na gumb »trenutna lokacija« in se nato v ustrezna polja vnesejo podatki (Slika 4.68).

Prav tako deluje GPS ob prikazu zemljevida tako, da se centrirna na uporabnikovo trenutno lokacijo. Implementacijo tega v Android Studiu in Xcodu je možno ustvariti tako, da se na uporabnikovi trenutni lokaciji prikaže moder krog, medtem ko pri RAD Studiu to ni mogoče. Zaradi tega smo na uporabnikovi trenutni lokaciji prikazali rdeči označevalec in pod zemljevidom izpisali podatki o njej (Slika 4.69).



Slika 4.68: Prikaz delovanja GPS v RAD Studiu Slika 4.69: Prikaz zemljevida v RAD Studiu

Za doseg prikaza označevalca na trenutni lokaciji smo implementirali metodo, ki ga doda na zemljevid (Slika 4.70). Metoda prejme koordinate trenutne lokacije, ki jih pridobimo s pomočjo Geocoder-ja.



```

void __fastcall TBunchUpForm::AddMarker(TMapCoordinate Position)
{
    if (CurrentMarker != NULL) {
        CurrentMarker->Remove();
    }
    TMapMarkerDescriptor MarkerDescriptor = TMapMarkerDescriptor::Create(Position, "");
    MarkerDescriptor.Icon = BitmapRedMarker;
    MarkerDescriptor.Visible = true;
    CurrentMarker = GoogleMap->AddMarker(MarkerDescriptor);
    ActiveMarker = CurrentMarker;
}

```

**Slika 4.70: Metoda za dodajanje začetnega označevalca na zemljevid v RAD Studiu**

Implementirali smo še eno metodo, ki centrira zemljevid na trenutno lokacijo in ga prav tako približa (Slika 4.71).

```

void __fastcall TBunchUpForm::ZoomGoogleMap(TMapCoordinate Position)
{
    GoogleMap->Location = Position;
    GoogleMap->Zoom = 12;
}

```

**Slika 4.71: Metoda za centriranje zemljevida v RAD Studiu**

Prav tako je bila podobna implementacija za vnos trenutne lokacije v ustrezna polja pri dodajanju dogodkov. Naredili smo metodo, ki se proži ob kliku na gumb »trenutna lokacija« (Slika 4.72).

```

void __fastcall TBunchUpForm::CurrentLocationButtonClick(TObject *Sender)
{
    GoogleMapControllerForm->GPS->Active = true;
    GoogleMapControllerForm->AddEventCurrentLocation = true;
    GoogleMapControllerForm->SetGeocoderAddEvent(ActiveMarker->Descriptor.Position);
}

```

**Slika 4.72: Metoda, ki se proži ob kliku na gumb "trenutna lokacija" v RAD Studiu**

Podobno kot smo naredili že pri prejšnjem poglavju s pomočjo Geocoder-ja, pretvorimo koordinate v naslov. Ko smo pridobili naslov, smo ga vnesli v ustrezna polja (Slika 4.73).

```

void __fastcall TBunchUpForm::SetAddEventInfo(TCivicAddress *Address)
{
    LocationRectangle->Visible = true;
    EventRectangle->Visible = false;
    GoogleMap->Margins->Bottom = 60;
    if (Address->Thoroughfare != NULL && !Address->Thoroughfare.IsEmpty()
        && Address->PostalCode != NULL && !Address->PostalCode.IsEmpty()
        && Address->Locality != NULL && !Address->Locality.IsEmpty()
        && Address->CountryName != NULL && !Address->CountryName.IsEmpty()) {

        StreetEdit->Text = Address->Thoroughfare;
        StreetNumberEdit->Text = Address->SubThoroughfare;
        CityEdit->Text = Address->Locality;
        PostalCodeEdit->Text = Address->PostalCode;
        CountryEdit->Text = Address->CountryName;
    }
}

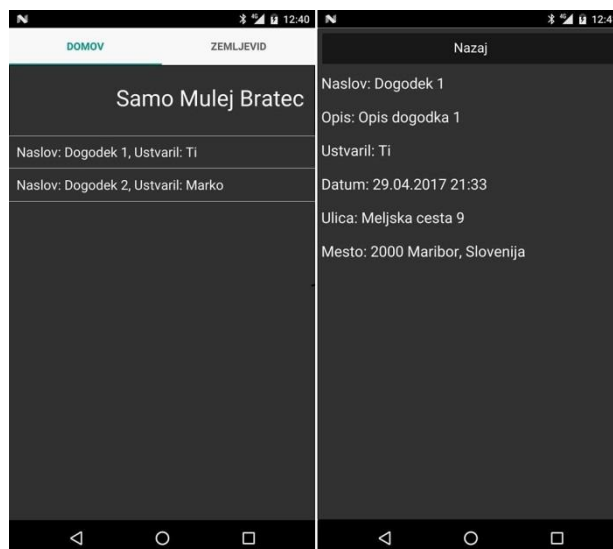
```

**Slika 4.73: Metoda, ki vnese podatke v ustrezna polja pri dodajanju dogodkov v RAD Studiu**

Implementacija tega ni povzročala večjih težav, smo pa imeli dodatno delo z dodajanjem lastnega označevalca, ker nismo mogli uporabiti v RAD Studiu modrega kroga za prikaz trenutne lokacije. Prav tako je način, kot ga omogočata Android Studio in Xcode, lepši za uporabnike.

#### 4.4.4 Lokalno shranjevanje podatkov

Za delovanje aplikacije brez internetne povezave smo naredili še zavihek Domov, kjer so brez uporabe GoogleMap prikazani vsi aktualni dogodki za uporabnika. Tako kot v Android Studiu in Xcodu so tudi tukaj prikazani le naslov o dogodku in kdo ga je ustvaril zaradi hitrega pregleda. Ob kliku na posamezni dogodek se nam odpre novo okno, na katerem so nato prikazani vsi podatki za izbrani dogodek (naslov, opis, datum, ulica, mesto in kdo ga je ustvaril) (Slika 4.74).

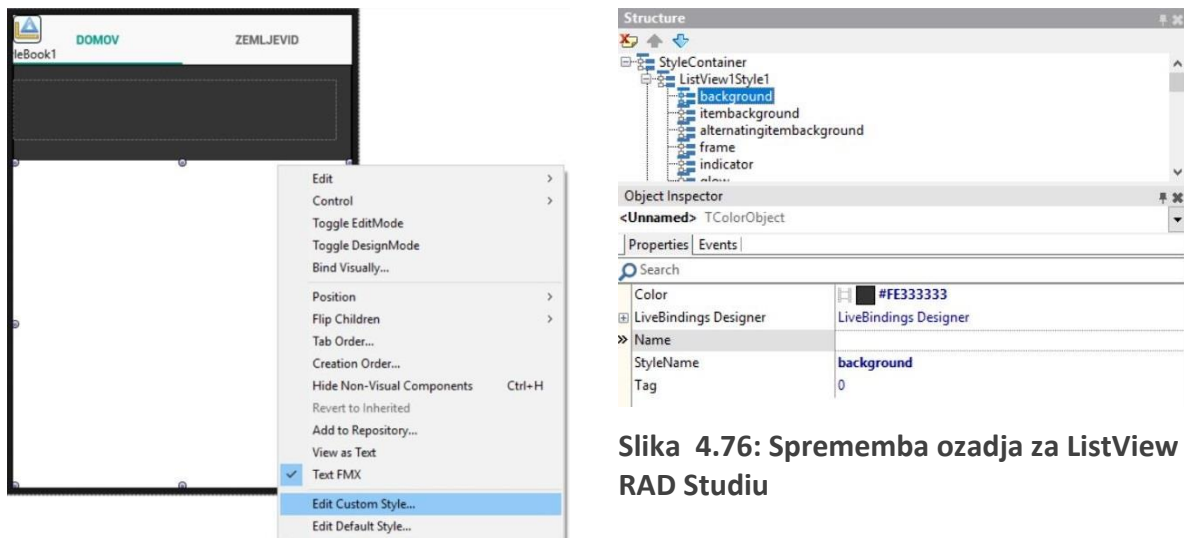


**Slika 4.74: Prikaz dogodkov v RAD Studiu**

Za prikaz vseh dogodkov smo uporabili ListView, kajti z njim lahko dosežemo prikaz neomejenega števila dogodkov. Če je dogodkov več, kot jih je lahko prikazanih na zaslonu, se ti pomaknejo v ozadje. Do njih dostopamo tako, da s pomočjo prsta premikamo dogodke navzdol ali navzgor; zato se prikazujejo dogodki v pravem vrstnem redu, kot smo to implementirali.

Nekaj težav smo imeli pri spremembi ozadja za ListView (privzeto nastavljena barva je bela), saj v RAD Studio komponenta sama po sebi nima lastnosti pod nastavitvami. Ugotovili smo, da lahko to dosežemo s spremembo sloga. Ob kliku z desno miško na ListView se nam pojavi možnost »Edit Custom Style« (Slika 4.75). Ob kliku na to možnost pridemo v oblikovanje sloga, kjer v sami strukturi lahko najdemo naš ListView in spremenimo njegovo ozadje (Slika 4.76).

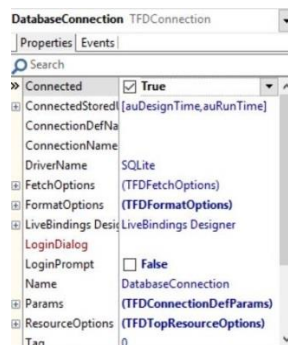




Slika 4.76: Sprememba ozadja za ListView v RAD Studiu

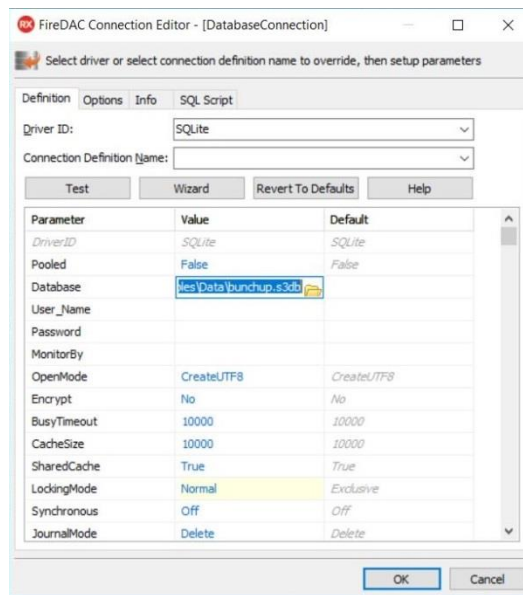
Slika 4.75: Urejanje sloga v RAD Studiu

Za implementacijo shranjevanja podatkov smo lahko v RAD Studiu tako kot v Android Studiu uporabili SQLite. Najprej je bilo potrebno ustvariti povezavo z lokalno podatkovno bazo. To smo naredili z uporabo komponente FDConnection, kjer smo v nastavitvah lastnosti izmed Driveri izbrali SQLite (Slika 4.77).



Slika 4.77: Nastavitve lastnosti za lokalno podatkovno bazo v RAD Studiu

Za delovanje povezave je bilo potrebno ustvariti s3db datoteko, kjer se nato podatki shranjujejo. Ko smo datoteko ustvarili, smo jo še morali nastaviti v urejevalniku povezave s podatkovno bazo (Slika 4.78) ter za Android in iOS napravo implementirati metodo, ki se proži pred vzpostavljeno povezavo s podatkovno bazo. V njej smo implementirali spremembo lokacije te datoteke, saj je lokacija na mobilnih napravah drugačna (Slika 4.79).



Slika 4.78: Urejanje povezave z SQLite v RAD Studiu

```
void __fastcall TDatabaseForm::DatabaseConnectionBeforeConnect(TObject *Sender)
{
    #if defined(_PLAT_IOS) || defined(_PLAT_ANDROID)
        DatabaseConnection->Params->Values["Database"] =
            System::Iutils::TPath::Combine(System::Iutils::TPath::GetDocumentsPath
            (), "bunchup.s3db");
    #endif
}
```

Slika 4.79: Metoda, ki poveže podatkovno bazo z s3db datoteko v RAD Studiu

V RAD Studiu lahko SQL stavke pišemo s pomočjo komponente FDQuery. Pred začetkom uporabe podatkovne baze je bilo potrebno ustvariti tabele (Slika 4.80), nad katerimi lahko nato počnemo operacije. Da se te tabele zares ustvarijo, smo implementirali metodo, ki se izvede takoj po vzpostavitvi povezave s podatkovno bazo ter izvrši SQL (Slika 4.81).

```
SQL Command Parameters Macros Options
CREATE TABLE IF NOT EXISTS city(id_city integer primary key autoincrement,city TEXT NOT NULL,
postal_code TEXT NOT NULL, country TEXT NOT NULL);
CREATE TABLE IF NOT EXISTS street(id_street integer primary key autoincrement,street TEXT NOT NULL,
street_number TEXT NOT NULL, city_id integer NOT NULL);
CREATE TABLE IF NOT EXISTS user(id_user integer primary key autoincrement,name TEXT NOT NULL,surname TEXT NOT NULL,
email TEXT NOT NULL,password TEXT NOT NULL,login BOOL NOT NULL);
CREATE TABLE IF NOT EXISTS event (id_event integer primary key autoincrement,title TEXT NOT NULL,description TEXT NOT NULL,
event_date date NOT NULL,event_time time NOT NULL,street_id integer NOT NULL,user_id integer NOT NULL);
CREATE TABLE IF NOT EXISTS last_updated_date (last_updated_date_id integer primary key autoincrement,last_updated_date DATE_TIME NOT NULL);
```

Slika 4.80: Kreiranje tabel v RAD Studiu

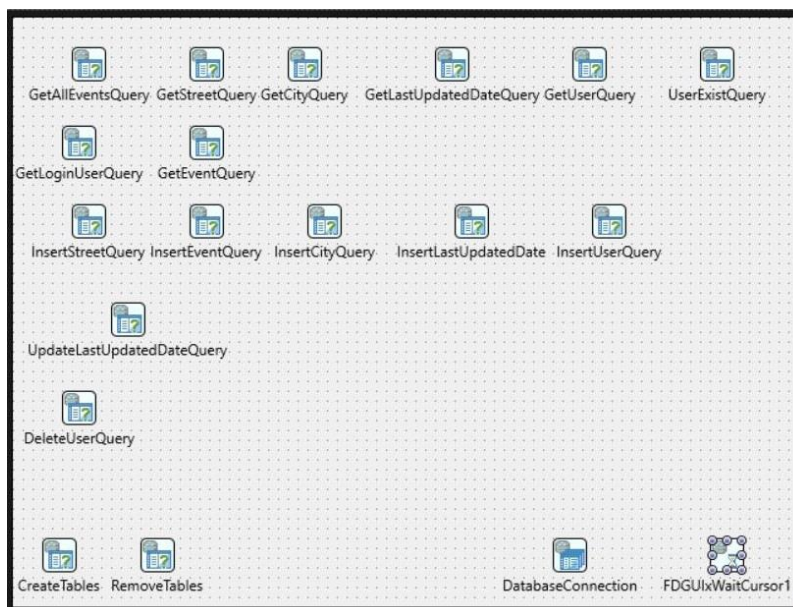
```

void __fastcall TDatabaseForm::DatabaseConnectionAfterConnect(TObject *Sender)
{
    CreateTables->ExecSQL();
}

```

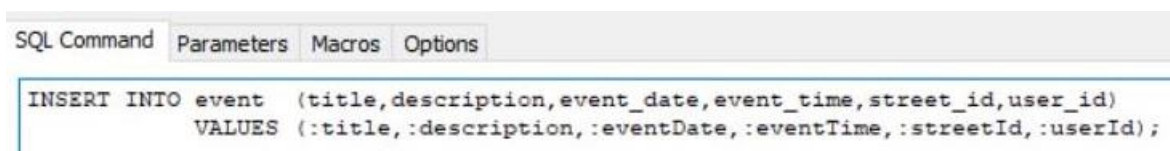
**Slika 4.81: Metoda, ki ustvari tabele v RAD Studiu**

Vse komponente FDQuery, ki vsebujejo SQL stavke za različne poizvedbe, je v RAD Studiu potrebno povleči v Formo (Slika 4.82).



**Slika 4.82: Videz Forme za podatkovno bazo v RAD Studiu**

V komponente FDQuery pišemo SQL stavke. Za uporabo parametrov, vrednosti katerih lahko dodamo v kodi pred izvršitvijo SQL stavka, napišemo pred imenom spremenljivke »:« (na primer :title) (Slika 4.83).



**Slika 4.83: SQL stavek za dodajanje dogodka v podatkovno bazo v RAD Studiu**

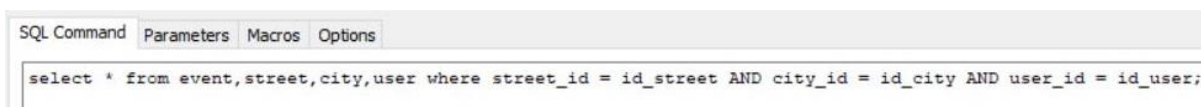
Zgoraj omenjeno SQL poizvedbo smo uporabili pri shranjevanju dogodkov v lokalno podatkovno bazo. Namen je, da so podatki, naloženi s strežnika, shranjeni v napravi ter

dostopni tudi brez internetne povezave. V metodi za shranjevanje dogodkov smo pred izvedbo FDQueryja k njemu dodali vse vrednosti parametrov (podatki o dogodku) ter šele nato izvršili to SQL poizvedbo (Slika 4.84).

```
void __fastcall TDatabaseForm::SaveEvent(CEvent *Event)
{
    CStreet Street = Event->GetStreet();
    CCity City = Street.GetCity();
    CUser User = Event->GetUser();
    InsertCityQuery->ParamByName("city")->AsString = City.GetCity();
    InsertCityQuery->ParamByName("postalCode")->AsString = City.GetPostalCode();
    InsertCityQuery->ParamByName("country")->AsString = City.GetCountry();
    InsertCityQuery->ExecSQL();
    int CityId = GetCity(City);
    InsertStreetQuery->ParamByName("street")->AsString = Street.GetStreet();
    InsertStreetQuery->ParamByName("streetNumber")->AsString = Street.GetStreetNumber();
    InsertStreetQuery->ParamByName("cityId")->AsString = CityId;
    InsertStreetQuery->ExecSQL();
    int UserId = GetUser(User);
    if (UserId == 0) {
        UserId = SaveUser(User);
    }
    int StreetId = GetStreet(Street.GetStreet(), Street.GetStreetNumber());
    InsertEventQuery->ParamByName("title")->AsString = Event->GetTitle();
    InsertEventQuery->ParamByName("description")->AsString = Event->GetDescription();
    InsertEventQuery->ParamByName("eventDate")->AsString = Event->GetEventDate();
    InsertEventQuery->ParamByName("eventTime")->AsString = Event->GetEventTime();
    InsertEventQuery->ParamByName("streetId")->AsInteger = StreetId;
    InsertEventQuery->ParamByName("userId")->AsInteger = UserId;
    InsertEventQuery->ExecSQL();
}
```

**Slika 4.84: Metoda za shranjevanje dogodkov v lokalno podatkovno bazo v RAD Studiu**

Za branje podatkov iz podatkovne baze je najprej prav tako potrebno pretvoriti SQL stavke v komponenti FDQuery (Slika 4.85).



```
SQL Command Parameters Macros Options
select * from event,street,city,user where street_id = id_street AND city_id = id_city AND user_id = id_user;
```

**Slika 4.85: SQL stavek za branje podatkov iz podatkovne baze v RAD Studiu**

Nato pridobljene podatke shranimo v DataSet, iz katerega lahko preoblikujemo podatke v željeno obliko. Ker je teh dogodkov lahko neomejeno število, smo uporabili DynamicArray, v katerega na posamezni indeks shranimo posamezni dogodek (Slika 4.86).

```

DynamicArray<CEvent> __fastcall TDatabaseForm::GetEvents()
{
    DynamicArray<CEvent> *Events = new DynamicArray<CEvent>();
    GetAllEventsQuery->Open();
    TDataSet *DataSet = GetAllEventsQuery->Adapter->DataSet;
    Events->set_length(DataSet->RecordCount);
    int i = 0;
    while (!DataSet->Eof){
        CCity *City = new CCity(DataSet->FieldValues["city"],DataSet->FieldValues["postal_code"],
            DataSet->FieldValues["country"]);
        CStreet *Street = new CStreet(DataSet->FieldValues["street"], DataSet->FieldValues["street_number"], *City);
        CUser *User = new CUser(DataSet->FieldValues["name"], DataSet->FieldValues["surname"],
            DataSet->FieldValues["email"], DataSet->FieldValues["password"]);
        CEvent *Event = new CEvent(DataSet->FieldValues["id_event"],DataSet->FieldValues["title"],
            DataSet->FieldValues["description"],DataSet->FieldValues["event_date"],DataSet->FieldValues["event_time"],
            *Street, *User);
        Events[0][i] = *Event;
        DataSet->Next();
        i++;
    }
    GetAllEventsQuery->Close();
    return *Events;
}

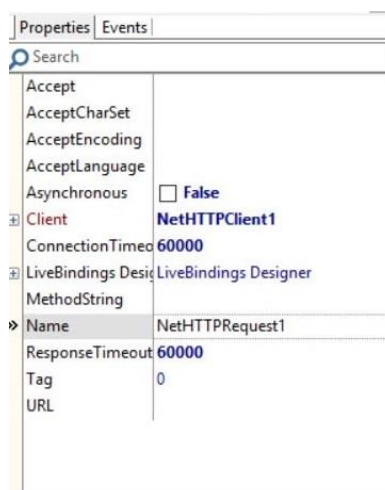
```

Slika 4.86: Metoda za branje podatkov iz lokalne podatkovne baze v RAD Studiu

Delo z SQLite v razvijalskem okolju RAD Studiu je enostavno in ni povzročalo resnih težav.

#### 4.4.5 Uporaba JSON-a in povezava s strežnikom

Za testiranje kakovosti implementacije za povezavo s strežnikom smo uporabili isti strežnik in iste podatke kot v Android Studiu in XCodu. Za začetek je bilo potrebno uporabiti komponento NetHTTPClient (odjemalec) in NetHTTPRequest (zahteva). V nastavitvi posamezne zahteve je bilo treba izbrati odjemalca, ki smo ga predhodno ustvarili (Slika 4.87).



Slika 4.87: Nastavitve zahteve v RAD Studiu

Za komunikacijo s strežnikom smo implementirali dve metodi:

- prva metoda pošlje zahtevo na strežnik (Slika 4.88);
- druga metoda pridobi odgovor s strežnika (prejme vse aktualne dogodke za uporabnika) (Slika 4.89).

```
void __fastcall TJsonForm::SaveEvents (String LastUpdatedDate)
{
    TMultiPartFormData *formData = new TMultiPartFormData(false);
    formData->AddField("LastUpdatedDate", LastUpdatedDate);
    GetEventsRequest->Post ("http://89.142.192.114/bunchup/GetEvents", formData);
}
```

**Slika 4.88: Metoda, ki pošlje zahtevo na strežnik v RAD Studiu**

```
void __fastcall TJsonForm::GetEvents(TObject * const Sender,
    IHTTPResponse * const AResponse)
{
    String result = AResponse->ContentAsString();
    DynamicArray<CEvent> *Events = new DynamicArray<CEvent>(CEvent::JsonToEventArray(result));
    TDatabaseForm *DatabaseForm = new TDatabaseForm(this);
    for (int i = 0; i < Events->get_length(); i++) {
        CEvent *Event = new CEvent(Events->operator[](i));
        DatabaseForm->SaveEvent(Event);
    }
}
```

**Slika 4.89: Metoda, ki prebere vse dogodke iz strežnika v RAD Studiu**

Strežnik nam vrne podatke v JSON obliki, zato je bilo potrebno le-te pretvoriti v objekte. V okolju RAD Studio je to povzročilo večje težave kot v Android Studiu in Xcodu, saj RAD Studio ne vsebuje knjižnice za pretvorbo in je bilo potrebno implementiranje.

V vsakem objektu smo ustvarili metodo, v kateri smo implementirali pretvorbo iz JSON oblike v objekt. Pogledali si bomo metodo, ki nam preslika JSON v polje dogodkov. Najprej je bilo potrebno implementirati metodo, ki nam izloči vsebino glede na podana znaka (JSONArray se začne z »[« in konča z »]«, JSONObject se začne z »{« in konča z »}«). Ker je znotraj posameznega JSONObject-a lahko le-teh več, smo morali znotraj metode biti previdni, da notranjih JSONObject-ov nismo upoštevali kot zaključek metode (Slika 4.90).



```

DynamicArray<String> Methods::Distinct(String string, char char1, char char2){
    DynamicArray<String> *Results = new DynamicArray<String>();
    String newString = "";
    bool start = false;
    int j=0;
    int sameChar = 0;
    for (int i = 0; i < string.Length(); i++) {
        if (string[i] == char2) {
            if (sameChar == 0) {
                Results->set_length(j + 1);
                start = false;
                Results[0][j] = newString;
                newString = "";
                j++;
            } else {
                sameChar--;
            }
        }
        if (start) {
            newString += string[i];
        }
        if (string[i] == char1) {
            if (start) {
                sameChar++;
            }
            start = true;
        }
    }
    return *Results;
}

```

Slika 4.90: Metoda, ki izloči vsebino znotraj podanih znakov

Znotraj JSONObject-a so imena parametrov in njegove vrednosti ločene z vejico, torej je bilo potrebno implementirati še metodo, ki nam razdeli niz na več delov glede na podani znak (na primer »Naslov«,«Dogodek1« razdeli na »Naslov« ter »Dogodek1« (Slika 4.91).

```

DynamicArray<String> Methods::Split(String string, char char1, char SpecialChar1, char SpecialChar2){
    DynamicArray<String> *Results = new DynamicArray<String>();
    String newString = "";
    int j = 0;
    int InsideSpecial = 0;
    for (int i = 0; i < string.Length(); i++) {
        if (string[i] == SpecialChar1){
            InsideSpecial++;
        }
        if (string[i] == SpecialChar2){
            InsideSpecial--;
        }
        if (string[i] == char1 && InsideSpecial == 0) {
            Results->set_length(j + 1);
            Results[0][j] = newString;
            newString = "";
            j++;
        } else if (string[i] != ''){
            newString += string[i];
        }
    }
    if (!newString.IsEmpty()){
        Results->set_length(j + 1);
        Results[0][j] = newString;
    }
    return *Results;
}

```

Slika 4.91: Metoda, ki razdeli niz na več delov glede na podani znak

Kot že omenjeno, je JSONObject sestavljen iz imena parametra in vrednosti. Ustvarili smo razred Parameter, ki vsebuje ti dve lastnosti (Slika 4.92).

```

CParameter::CParameter()
{
    this->name = "";
    this->value = "";
}
CParameter::CParameter(String name, String value)
{
    this->name = name;
    this->value = value;
}
String CParameter::GetName()
{
    return this->name;
}
String CParameter::GetValue()
{
    return this->value;
}

```

Slika 4.92: Razred Parameter

Na koncu je bilo potrebno še implementirati metodo, ki nam pretvori podani niz v razred Parameter, da lahko nato pridobimo vrednosti dogodkov (Slika 4.93).

```

CParameter Methods::SplitInParameter(String string, char charl){
    DynamicArray<String> *Results = new DynamicArray<String>();
    String name = "";
    String value = "";
    bool first = true;
    bool second = false;
    for (int i = 0; i < string.Length(); i++) {
        if (first && string[i] != charl) {
            name += string[i];
        } else if (!first && string[i] != charl) {
            value += string[i];
        } else if (!first && second) {
            value += string[i];
        }
        if (string[i] == charl) {
            second = true;
            first = false;
        }
    }
    CParameter *Parameter = new CParameter(name, value);
    return *Parameter;
}

```

Slika 4.93: Metoda, ki pretvori niz v razred Parameter

Po izdelavi vseh teh metod smo lahko le-te preprosto uporabili v vsakem objektu posebej ter pretvorili JSON v objekt. Torej smo za pretvorbo iz JSONArray-a v polje dogodkov implementirali metodo, kjer smo uporabili vse zgoraj našteje metode za razčlenitev niza JSON oblike v objekt Event (dogodek) (Slika 4.94).



```

DynamicArray<CEvent> CEvent::JsonToEventArray(String jsonString)
{
    DynamicArray<CEvent> *Events = new DynamicArray<CEvent>();
    DynamicArray<String> *Results = new DynamicArray<String>(Methods::Distinct(jsonString, '[', ']'));
    jsonString = Results->operator[](0);
    Results = new DynamicArray<String>(Methods::Distinct(jsonString, '{', '}'));
    int k = 0;
    for (int i = 0; i < Results->get_length(); i++) {
        jsonString = Results->operator[](i);
        DynamicArray<String> *Results1 =
            new DynamicArray<String>(Methods::Split(jsonString, ',', '{', '}'));
        String title = "";
        String description = "";
        String eventDate = "";
        String eventTime = "";
        CStreet *Street = new CStreet();
        CUser *User = new CUser();
        for (int j = 0; j < Results1->get_length(); j++) {
            CParameter *Parameter =
                new CParameter(Methods::SplitInParameter(Results1->operator[](j), ':'));
            if (Parameter->GetName() == "title") {
                title = Parameter->GetValue();
            } else if (Parameter->GetName() == "description") {
                description = Parameter->GetValue();
            } else if (Parameter->GetName() == "eventDate") {
                eventDate = Parameter->GetValue();
            } else if (Parameter->GetName() == "eventTime") {
                eventTime = Parameter->GetValue();
            } else if (Parameter->GetName() == "street") {
                Street = new CStreet(CStreet::JsonToStreet(Parameter->GetValue()));
            } else if (Parameter->GetName() == "user") {
                User = new CUser(CUser::JsonToUser(Parameter->GetValue()));
            }
        }
        CEvent *Event = new CEvent(title, description, eventDate, eventTime, *Street, *User);
        Events->set_length(k + 1);
        Events[0][k] = *Event;
        k++;
    }
    return *Events;
}

```

**Slika 4.94: Metoda za pretvorbo dogodka iz JSON oblike v objekt**

Kot lahko vidimo, je implementacija pretvorbe težavna (potrebno je veliko kode) ter nam je vzela veliko časa. Komunikacijo s strežnikom smo tako v Android Studio kot tudi v Xcodu lahko hitreje in preprostejše implementirali.

#### 4.5 Primerjava okolja RAD Studio z domorodnimi okolji

Razvojno okolje RAD Studio vsebuje veliko slabosti v primerjavi z domorodnimi okolji za Android (Android Studio) ter iOS (Xcode). Slabost RAD Studia se je pojavila že na samem začetku: ker nima dobro podprte Facebook povezave, smo morali implementirati ročno povezavo, kar nas je posledično privedlo do slabše uporabniške izkušnje ter smo porabili več časa za implementacijo.

Tudi implementacija prikaza zemljevida v Android Studiu in Xcodu je preprostejša ter nudi boljšo uporabniško izkušnjo. RAD Studio nam ni omogočal direktne spremembe barve označevalcev, temveč smo za doseg tega morali le-te vstaviti kot sliko, kar pripelje do slabše optimizacije aplikacije. Zahteva tudi več časa, saj smo le-te morali najti na spletu.

Zategadelj je tudi aplikacija manj stabilna, saj v primeru predolgega zagona lahko preneha delovati. Prav tako okolje RAD Studio ne podpira prikaza kroga na zemljevidu za trenutno lokacijo, kot to imata Android Studio ter Xcode, torej smo morali prikazati svoj označevalec za trenutno lokacijo, kar je pripeljalo do slabše uporabniške izkušnje in porabe več časa.

Za implementacijo GPS-a je potrebno nekoliko manj časa kot v Android Studiu ter Xcodu, vendar le-ta deluje počasneje. Včasih lahko traja tudi do 5 sekund, da se pridobi trenutna lokacija uporabnika, kar vpliva na slabšo uporabniško izkušnjo. Težava se je pojavila pri prikazu dialoga za dodajanje dogodkov. Ozadje dialogov je vplivalo na barvo gumbov, kar je onemogočilo črno barvo, medtem ko v Android Studiu in Xcodu to ni bil problem. Ta omejitev nam lahko povzroči resne težave, saj v določenih primerih ne moremo doseči zelenega videza uporabniškega vmesnika.

Implementacija shranjevanja v lokalno podatkovno bazo je bila enostavna in stabilna prav tako kot v Android Studiu in Xcodu. Pri delovanju nismo ugotovili težav. Edina, ki se je pojavila, je bila sprememba ozadja ListView, ker ta lastnost ni na vidnem oziroma pričakovanem mestu, vendar nam jo je dokaj hitro uspelo rešiti.

Komunikacijo s strežnikom sicer v osnovi naredimo hitreje kot v Android Studiu in Xcodu, a so nastale težave pri branju podatkov iz strežnika. RAD Studio ne vsebuje knjižnice za pretvorbo iz JSONArray-a in JSONObject-a v objekt, kar je zahtevalo našo implementacijo. Posledično nam je to vzelo veliko več časa, kot bi ga porabili v Android Studiu in Xcodu. Vendar ko nam je uspela implementacija, je ta povezava delovala hitro in brezhibno.

## 5. SKLEP

Z namenom ugotoviti, ali lahko z razvojnim okoljem RAD Studio izdelamo aplikacije enako ali celo kvalitetneje kot to v domorodnih okoljih (Android Studio in Xcode), smo izdelali identično aplikacijo v vseh treh orodjih. Ugotovili smo, da okolje RAD Studio predstavlja veliko ovir, oteži naše delo pri implementaciji ter vsebuje veliko napak, ki lahko posledično privedejo do slabše uporabniške izkušnje, manj stabilne aplikacije, slabše optimizirane ter porabe celo več časa, kot ga potrebujemo v domorodnih okoljih.

Okolje RAD Studio že samo po sebi ni tako stabilno, kot sta Android Studio in Xcode. Pogosto se nam je zgodilo, da se je okolje sesulo brez razloga ter ob ponovni vzpostavitvi orodja ni bilo vso delo shranjeno tako, kot bi moralo biti. Tudi sama hitrost delovanja je dosti slabša, saj je včasih potrebno čakati celo 10 sekund, preden se nam pojavi seznam možnih metod, ki jih lahko uporabimo na določeni spremenljivki, medtem ko Android Studio in Xcode delujeta brezhibno in se prikaže seznam metod takoj.

Zaradi vseh omenjenih slabosti predhodno podane trditve (v poglavju NAMEN) zavržemo, saj lahko ustvarimo hitreje veliko boljše aplikacije v domorodnih orodjih in nam okolje ne povzroča odvečnih težav.

## 6. VIRI

[1] AMD, What is Heterogeneous Computing? Dostopno na:

<http://developer.amd.com/resources/heterogeneous-computing/what-is-heterogeneous-computing> [01.07.2017].

[2] Grønli T, Hansen J, Ghinea G, Younas M, Mobile Application Platform Heterogeneity: Android vs Windows Phone vs IOS vs Firefox OS. Maj 2014. Dostopno na:

[https://www.researchgate.net/publication/269301609\\_Mobile\\_Application\\_Platform\\_Heterogeneity\\_Android\\_vs\\_Windows\\_Phone\\_vs\\_iOS\\_vs\\_Firefox\\_OS](https://www.researchgate.net/publication/269301609_Mobile_Application_Platform_Heterogeneity_Android_vs_Windows_Phone_vs_iOS_vs_Firefox_OS) [01.07.2017].

[3] Embacadero. Dostopno na: <https://www.embarcadero.com/products/rad-studio>

[01.07.2017].

[4] Techopedia, Heterogeneous System Architecture . Dostopno na:

<https://www.techopedia.com/definition/30886/heterogeneous-system-architecture-hsa>

[06.09.2017].

[5] Techopedia, Cross platform. Dostopno na:

<https://www.techopedia.com/definition/17056/cross-platform> [09.09.2017].

[6] Hongkiat, Top 10 Cross-Platform Mobile Development Tools. Dostopno na:

<http://www.hongkiat.com/blog/cross-mobile-platform-framework-wora> [09.09.2017].

[7] Reynolds, M. Xamarin Mobile Application Development for Android. Birmingham: Packt Publishing Ltd., 2014.

[8] Wargo, J. Apache Cordova 3 Programming. Boston: Addison.Wesley Professional, 2013.

[9] Anderson, J. Appcelator Titanium: Up and Running. Sebastopol: O'Reilly Media, Inc., 2013.



Univerza v Mariboru



Fakulteta za elektrotehniko,  
računalništvo in informatiko

### IZJAVA O AVTORSTVU IN ISTOVETNOSTI TISKANE IN ELEKTRONSKE OBLIKE ZAKLJUČNEGA DELA

Ime in priimek študent-a/-ke: SAMO KULEJ BRATEC  
Študijski program: INFORMATIKA IN TEHNOLOGIJE KOMUNICIRANJA  
Naslov zaključnega dela: RAZVOJ HETEROGENIH MOBILNIH  
APLIKACIJ V RAZVOJNEM OKOLJU RAD STUDIO 10

Mentor: doc. dr. DOMEN VERBER  
Somentor: \_\_\_\_\_

Podpisan-i/-a študent/-ka SAMO KULEJ BRATEC

- izjavljam, da je zaključno delo rezultat mojega samostojnega dela, ki sem ga izdelal/-a ob pomoči mentor-ja/-ice oz. somentor-ja/-ice;
- izjavljam, da sem pridobil/-a vsa potrebna soglasja za uporabo podatkov in avtorskih del v zaključnem delu in jih v zaključnem delu jasno in ustrezno označil/-a;
- na Univerzo v Mariboru neodplačno, neizključno, prostorsko in časovno neomejeno prenašam pravico shranitve avtorskega dela v elektronski obliki, pravico reproduciranja ter pravico ponuditi zaključno delo javnosti na svetovnem spletu preko DKUM; sem seznanjen/-a, da bodo dela deponirana/objavljena v DKUM dostopna široki javnosti pod pogoji licence Creative Commons BY-NC-ND, kar vključuje tudi avtomatizirano indeksiranje preko spleta in obdelavo besedil za potrebe tekstovnega in podatkovnega rudarjenja in ekstrakcije znanja iz vsebin; uporabnikom se dovoli reproduciranje brez predelave avtorskega dela, distribuiranje, dajanje v najem in priobčitev javnosti samega izvirnega avtorskega dela, in sicer pod pogojem, da navedejo avtorja in da ne gre za komercialno uporabo;
- dovoljujem objavo svojih osebnih podatkov, ki so navedeni v zaključnem delu in tej izjavi, skupaj z objavo zaključnega dela;
- izjavljam, da je tiskana oblika zaključnega dela istovetna elektronski obliki zaključnega dela, ki sem jo oddal/-a za objavo v DKUM.

Uveljavljam permisivnejšo obliko licence Creative Commons: \_\_\_\_\_ (navedite obliko)

#### Začasna nedostopnost:

Zaključno delo zaradi zagotavljanja konkurenčne prednosti, zaščite poslovnih skrivnosti, varnosti ljudi in narave, varstva industrijske lastnine ali tajnosti podatkov naročnika:

\_\_\_\_\_ (naziv in naslov naročnika/institucije) ne sme biti javno dostopno do \_\_\_\_\_ (datum odloga javne objave ne sme biti daljši kot 3 leta od zagovora dela). To se nanaša na tiskano in elektronsko obliko zaključnega dela.



Univerza v Mariboru

Fakulteta za elektrotehniko,  
računalništvo in informatiko

Koroška cesta 46  
2000 Maribor, Slovenija



### IZJAVA O OBJAVI OSEBNIH PODATKOV

Ime in priimek diplomant-a/ magistrant-/-ke: SAMO KULEJ BRATEC

ID številka: 1002166497

Študijski program: INFORMATIKA IN TEHNOLOGIJE KOMUNICIRANJA

Naslov zaključnega dela: RAZVOJ HETEROGENIH MOBILNIH APLIKACIJ  
V RAZVOJNEM OKOLJU RAD STUDIO 10

Mentor/-ica: doc. dr. DOMEN VERBER

Somentor/-ica: \_\_\_\_\_

Podpisan-i/-a izjavljam, da dovoljujem objavo osebnih podatkov, vezanih na zaključek študija (ime, priimek, leto zaključka študija, naslov zaključnega dela) na spletnih straneh Univerze v Mariboru in v publikacijah Univerze v Mariboru.

Datum in kraj:

19.9.2017, MARIBOR

Podpis diploman-ta/magistran-ta/-ke:

Samo Kulej





Univerza v Mariboru

Fakulteta za elektrotehniko,  
računalništvo in informatiko

Koroška cesta 46  
2000 Maribor, Slovenija



## IZJAVA O USTREZNOSTI ZAKLJUČNEGA DELA

Podpisani mentor/-ica : doc. dr. DOMEN VERBER  
(ime in priimek mentor-ja/-ice)

in somentor/-ica (eden ali več, če obstajajo): \_\_\_\_\_  
(ime in priimek somentor-ja/-ice)

Izjavlja-m/-va/-mo, da je študent/-ka

Ime in priimek: SAMO MULEJ BRATEC, ID številka: 1002166497,

vpisna številka: E1078132, na študijskem programu:

INFORMATIKA IN TEHNOLOGIJE KOMUNICIRANJA

izdelal/-a zaključno delo z naslovom:

RAZVOJ HETEROGENIH MOBILNIH APLIKACIJ V  
RAZVOJNEM OKOLJU RAD STUDIO 10

(naslov zaključnega dela v slovenskem jeziku)

v skladu z odobreno temo zaključnega dela, navodili o pripravi zaključnih del in mojimi (najinimi/našimi) navodili.

Preveril/-a/-i in pregledal/-a/-i sem/sva/smo poročilo o preverjanju podobnosti vsebin z drugimi deli (priloga) in potrujem/potrjujeva/potrjujemo, da je zaključno delo ustrezno.

Datum in kraj:

19. 9. 2017, Maribor

Datum in kraj:

Podpis mentor-ja/-ice:

Verber

Podpis somentor-ja/-ice (če obstaja):

Priloga:

- Poročilo o preverjanju podobnosti vsebin z drugimi deli.