

**ТЕХНОЛОГИЧНО УЧИЛИЩЕ “ЕЛЕКТРОННИ СИСТЕМИ”
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**

ДИПЛОМНА РАБОТА

Тема: Андроид приложение за срещи с приятели

Дипломант:

Симеон Георгиев

Научен ръководител:

Желязко Атанасов

СОФИЯ
2020



**ТЕХНОЛОГИЧНО УЧИЛИЩЕ “ЕЛЕКТРОННИ СИСТЕМИ”
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**

Дата на заданието: 15.11.2019 г.

Дата на предаване: 15.02.2020 г.

Утвърждавам:.....

/проф. д-р инж. Т. Василева/

ЗАДАНИЕ

за дипломна работа

на ученика Симеон Христов Георгиев 12 А клас

1.Тема: Мобилно приложение за срещи с приятели

2.Изисквания:

- 2.1 Регистриране с Facebook аутентикация
- 2.2 Търсене и добавяне на приятели от Facebook
- 2.3 Активен режим за откриване на събития
- 2.4 Карта, показваща приятелите около теб
- 2.5 “Хийтмап“ с най-голяма концентрация на хора
- 2.6 Създаване на планове/събития
- 2.7 Трупане на рейтинг в зависимост от посетени събития или среща с приятели

3.Съдържание: 3.1 Обзор

3.2 Същинска част

3.3 Приложение

Дипломант :.....

Ръководител:.....

/ Желязко Атанасов/

Директор:.....

/ доц. д-р инж. Ст. Стефанова /

Увод

Дори и в днешното забързано ежедневие имаме свободно време, което не можем да оползотворим правилно. Мисля, че много хора може да се съгласят, че са имали време да се “шляят” и им се е искало да го прекарат на по бира с приятел, а след седмица се е оказало, че техният познат е бил наблизо и е имал същата идея.

Настоящата дипломна работа има за цел да разработи приложение за срещи с приятели за операционната система Android. То ще предоставя многоброен набор от функционалности. Включвайки създаване на акаунт, намиране и добавяне на приятели, откриването им на картата около вас. Потребителите могат да създадат събитие и да получат QR код, който посетителите сканират така се отбелязват в събитието. За да могат потребителите да видят кои са най-популярните места за събиране, приложението ще разполага с Heatmap, който показва къде се срещат най-много хора.

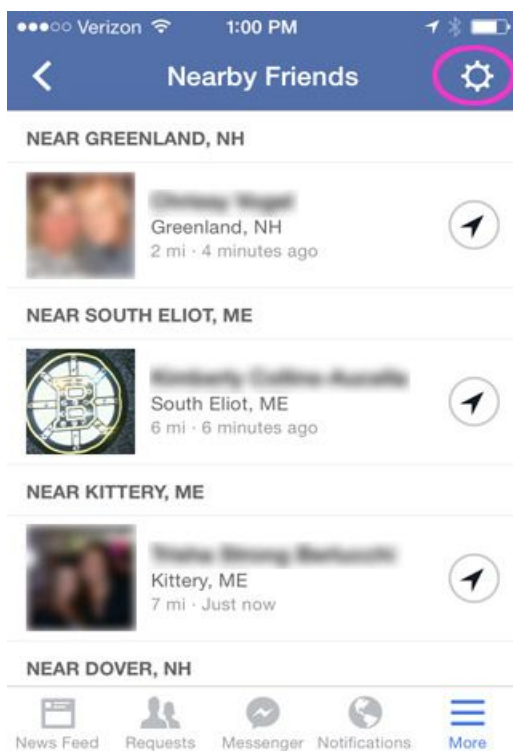
Първа Глава

Преглед на използвани методи и технологии. Проучване на подобни приложения

1.1 Съществуващи приложения

1.1.1 Facebook nearby friends

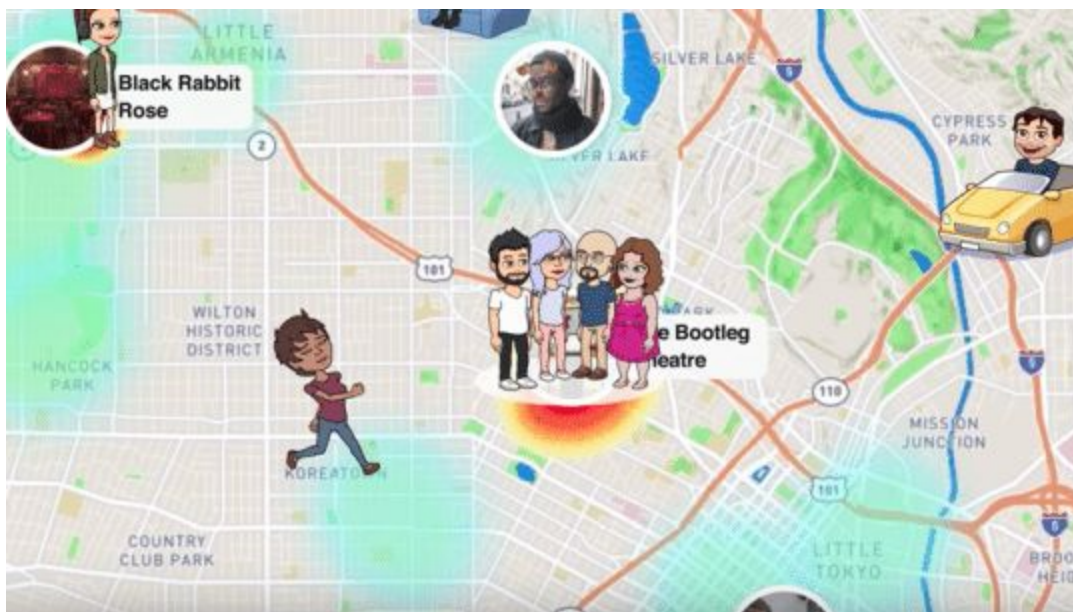
Facebook предоставя възможността да се види релативното местоположение на ваши приятели, които са включили локацията на устройствата си. Има browser версия, както и мобилна за Android и iOS. За съжаление, Facebook предпочитат да развиват другите функционалности по платформата си и не работят по Nearby Friends. За това и е ограничено само до гореспоменатата функционалност.



Фигура 1.1 Facebook Nearby Friends

1.1.2 Snapchat Map

Snapchat предоставя карта, на която могат да се гледат публичните истории на всички потребители. В мобилното приложение потребителите могат да видят и локацията на приятели. Освен локация, Snapchat следят и други дейности на потребителя, за да им предоставят интерактивни аватари. Например, ако се движим бързо, аватара ни се вози в кола, или ако сме включили слушалки в мобилното устройство, аватара ни слуша музика. Приложението има и хийтмап на публичните истории.

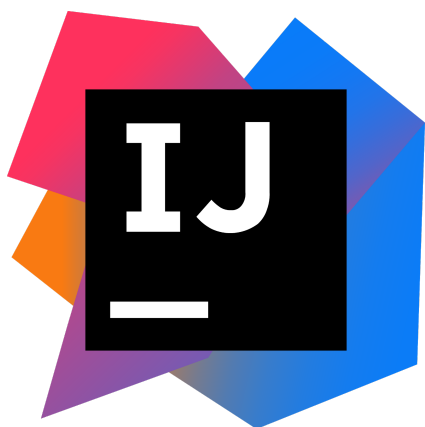


Фигура 1.2 SnapMap

1.2 Технологии, среди за развой и бази данни

1.2.1 IntelliJ IDEA

IntelliJ IDEA е продукт на JetBrains. Представява IDE (integrated development environment) или интегрирана среда за разработка. Към днешна дата е една от най-популярните интегрирани среди за разработка на Java. Разполага с редица функционалности за улеснение на работата на програмиста - мултиезикова асистенция, предложения за подобряване на кода, както и възможността потребителя да пише или използва вече написани плъгини. С помощта на Android плъгин средата става подходяща за разработка на Android приложения. Има комерсиална (платена) и безплатна версия.



Фигура 1.3 IntelliJ

1.2.2 Android Studio

Android Studio е изградено върху IntelliJ IDEA. То е официалното IDE за Android, проектирано специално за лесната разработка на Android приложения. Разполага с емулятор, с който програмиста може да тества приложението си на различни мобилни устройства, без да се налага да ги притежава физически. Android Studio има и визуален редактор за XML

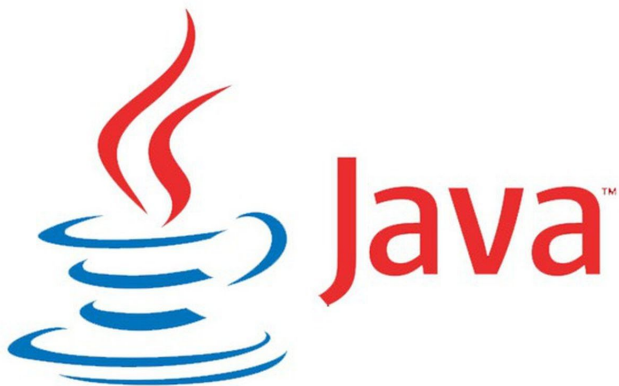
layouts, което позволява разработчика лесно да променя разположението и характеристиките на елементите на приложението.



Фигура 1.4 Android Studio

1.2.3 Java

Java е обектно-ориентиран език за програмиране. В Java почти всичко е обект – има данни и операции, които може да извършва над тях. Java предоставя възможността една програма да работи на разнообразни системи, тъй като кода се компилира до байт код (bytecode) и се изпълнява от виртуална машина (JVM).



Фигура 1.5 Java

1.2.4 Kotlin

Kotlin също е обектно-ориентиран език, който върви върху JVM. Създаден е с целта да бъде директно подобрение на Java - с повече

функционалност, null-safe оператори и възможността да се компилира до Javascript. Езикът е сравнително нов, но много Android разработчици в последните години мигрират към него.



Фигура 1.6 Kotlin

1.2.5 XML

XML е стандарт, дефиниращ правила за създаване на специализирани маркиращи езици, както и синтаксисът, на който тези езици трябва да се подчиняват. XML е метаязык - той няма семантика, не предава информация, а указва синтаксиса и структурата на един документ. В Android света, XML се използва за деклариране на потребителския интерфейс на приложението, чрез View класовете и техните подкласове - widgets и layouts (джунджурии и оформления)

1.2.6 Бази данни

Изборът на база данни е особено важен при разработката на мобилно приложение. Най-популярните опции са SQL (Structured Query Language -

релационни бази данни) и NoSQL (Not only Structured Query Language - нерелационни бази данни) базирани.

SQL базираните бази съхраняват данните като релации от записи, представени като таблици. Връзката между отделни таблици се осъществява чрез първични и вторични ключове. Извличането се извършва с заявки.

[insert плюсове и минуси на sql базите данни]. В Android света най-популярни релационни бази данни са: **PostgreSQL, MariaDB, SQLite и MySQL.**

NoSQL базираните бази съхраняват информацията под различни структури: ключ-стойност, колона, мултимодел, документ. По този начин се улесняват процесите по добавяне и извличане на информация и се увеличава производителността. Структурата с най-много имплементации са документните хранилища. Идеята зад тези имплементации е, че документите енкапсулират данните в някакви стандартни формати. Основните формати са: XML (Extensible Markup Language), YAML (YAML Ain't Markup Language) и JSON (JavaScript Object Notation), но се използват и бинарни формати като BSON (Binary JSON) и PDF (Portable Document Format). Най-често документите се групират и организират в колекции, тагове или в йерархия на директориите. Основни характеристики на документно-ориентираните бази данни е, че освен простото търсенето по ключ-документ или ключ-стойност, което може да се използва за извличане на документа, базите данни предоставят и потребителски интерфейс или език за заявки, които ще позволи документите да бъдат откривани въз основа на тяхното съдържание. Популярни имплементации от този тип са: **MongoDB, Firebase Realtime Database, eXist, BaseX и Oracle NoSQL Database**

1.2.7 Основни компоненти на Android приложението и Android SDK

Android SDK (Software development kit) компилира кода в APK (Android package) файл, чрез който мобилното устройство (с операционна система Android) лесно може да инсталира приложението. Всеки процес има собствена виртуална машина, по този начин приложенията са изолирани едно от друго и не използват ресурсите на другите.

Основните компоненти на Android приложението са:

Activity - представлява един екран с потребителския интерфейс.

Services - услугите, които се извършват на заден план, без потребителски интерфейс. Обикновено потребителя дори не знае, че услугата се изпълнява.

Broadcast receivers - позволява системата да изпраща съобщения към приложението, дори то да не се изпълнява. На този принцип работят известията (Notifications).

Content provider - менажира информация от файловата система, база данни от мрежата или от всяко място за съхранение, достъпно от приложението.

Activity, Service и Broadcast receiver се зареждат и активират с помощта на Intent. Intent е асинхронно съобщение, заявяващо на даден компонент да извърши някакво действие.

Android Manifest е XML файл, в който са декларирани:

- Всички компоненти на приложението. Те не могат да бъдат стартирани, ако не се намират в него.

- Всички разрешения, които приложението ще иска от системата, като достъп до камера, интернет, списък с контакти и други.
- Всички хардуерни свойства - bluetooth услуги, камера, multitouch и други.
- Минималното ниво на Android API - минималната версия на операционната система
- Всички библиотеки, различни от Android framework APIs, като например Google Maps библиотеката.

Втора Глава

Функционални изисквания към приложението. Проектиране на приложение за срещи с приятели

2.1 Функционални изисквания към проекта

- Регистриране с Facebook аутентикация
- Търсене и добавяне на приятели от Facebook
- Активен режим за откриване на събития
- Карта, на която можеш да виждаш приятелите около теб
- "Хийтмап" с най-голяма концентрация на хора
- Потребителите да могат да си добавят планове/събития
- Трупане на рейтинг в зависимост от посетени събития или срещи с приятели

2.2 Избор на програмен език и библиотеки

2.2.1 IDE - Android Studio

Android Studio е напълно безплатно и перфектно за целите на дипломната работа. Средата разполага с графичен редактор на .xml файлове, възможност да се създават директни инсталационни файлове за Android (.apk формат), вграден виртуален емулатор с многобройни дистрибуции и версии на Android

2.2.2 Език за програмиране - Java

При разработка на Android приложения има 2 основни езика - Java и Kotlin. Java е издържан език, за който има много библиотеки и документация, която би била полезна в процеса на разработка, за разлика от Kotlin, който е изключително мощен и разполага с редица предимства (като null-safe полета), но е сравнително нов и документацията е малко и трудно достъпна. Основна причина, да избира Java, е достъпността на източници, както и това, че се преподава в ТУЕС.

2.2.3 Firebase

Firebase е платформа на Google за разработка на мобилни и уеб приложения. Предоставя функционалности като аутентикация, аналитика, бази данни и crash reports. Firebase предлага безплатен начален план, който скалира заедно с потребителския трафик към приложението. Firebase Realtime Database е cloud (облачна) NoSQL базирана база данни. Информацията се съхранява в JSON формат и се синхронизира с всички свързани клиенти, това е постигнато като всички клиенти споделят една и съща инстанция на базата и автоматично получават промените. Това би било проблем, когато потребителя няма връзка с интернет, но Firebase Realtime Database SDK запазва локално данните на потребителя и когато отново има връзка с интернет базата се синхронизира - получава всички пропуснати промени и запазва локалните. Правилата за сигурност се задават в конзолата на Firebase и имат синтаксис, подобен на JavaScript. Всяка read/write заявка ще се изпълни само, ако отговаря на зададените правила.

2.2.4 Facebook SDK

Facebook SDK предоставя редица функционалности като аутентикация, както и възможността да бъде извлечена различна информация за потребителя - като име, email, приятели и тн. Също така, чрез него могат да се пишат постове на Facebook стената на потребителя. Аутентикацията може да бъде обвързана с Firebase. Firebase Authentication позволява използването на Facebook акаунти за по-лесна регистрация и ползване на приложението.

2.2.5 ZXing

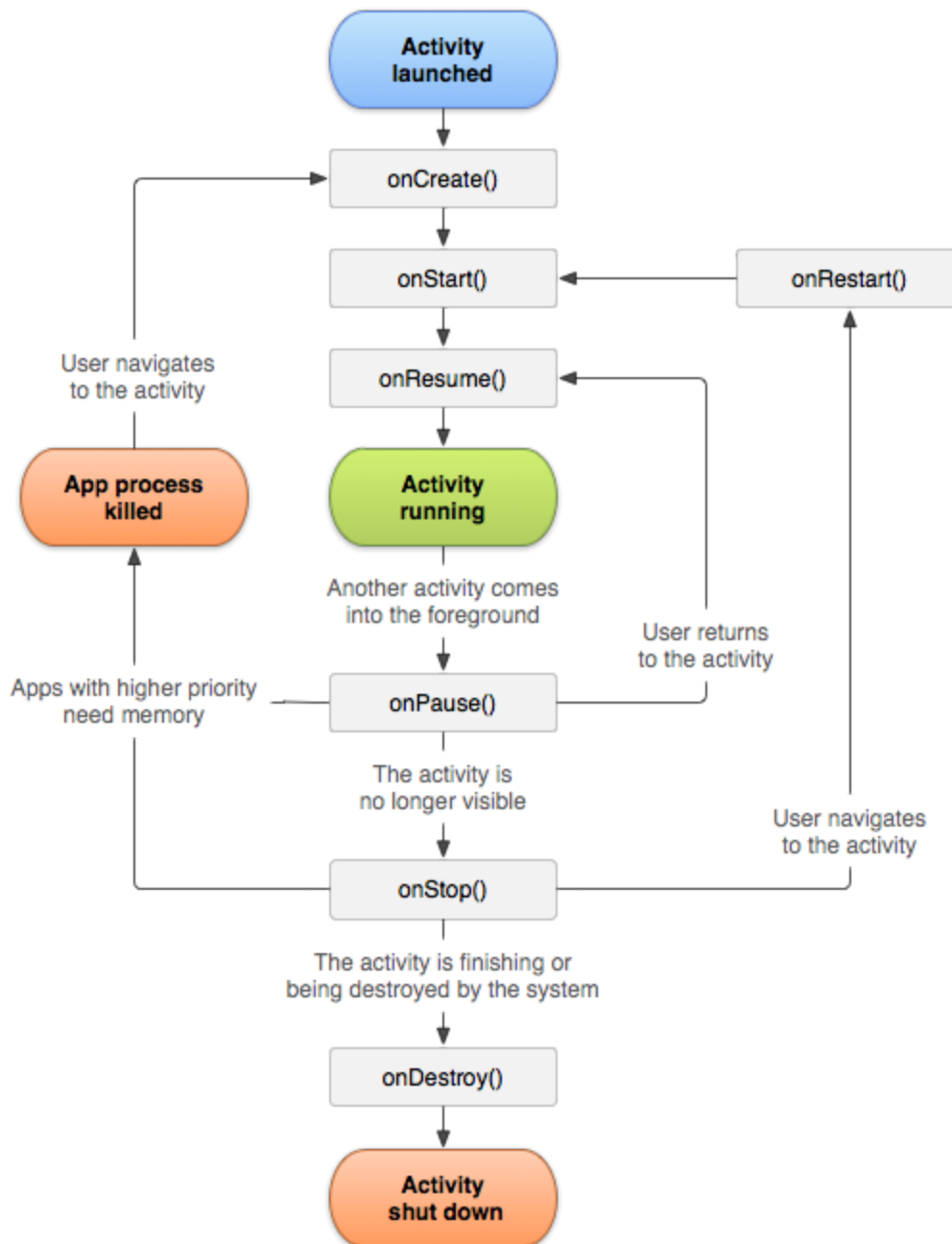
ZXing или Zebra Crossing е open-source библиотека за обработка на баркодове. Имплементирана е на Java, но има версии за C++, JavaScript, Python, PHP и C#. Поддържа 1D баркодове като UPC-A, EAN-13 и Codebar, както и 2D баркодове като QR code, Data Matrix и Aztec. Библиотеката се разработва и поддържа от Google. Те я използват като основа за баркод скенера на Android и е интегрирана в техни продукти, като например Google Book Search.

2.3 Проектиране

2.3.1 Структура на приложението

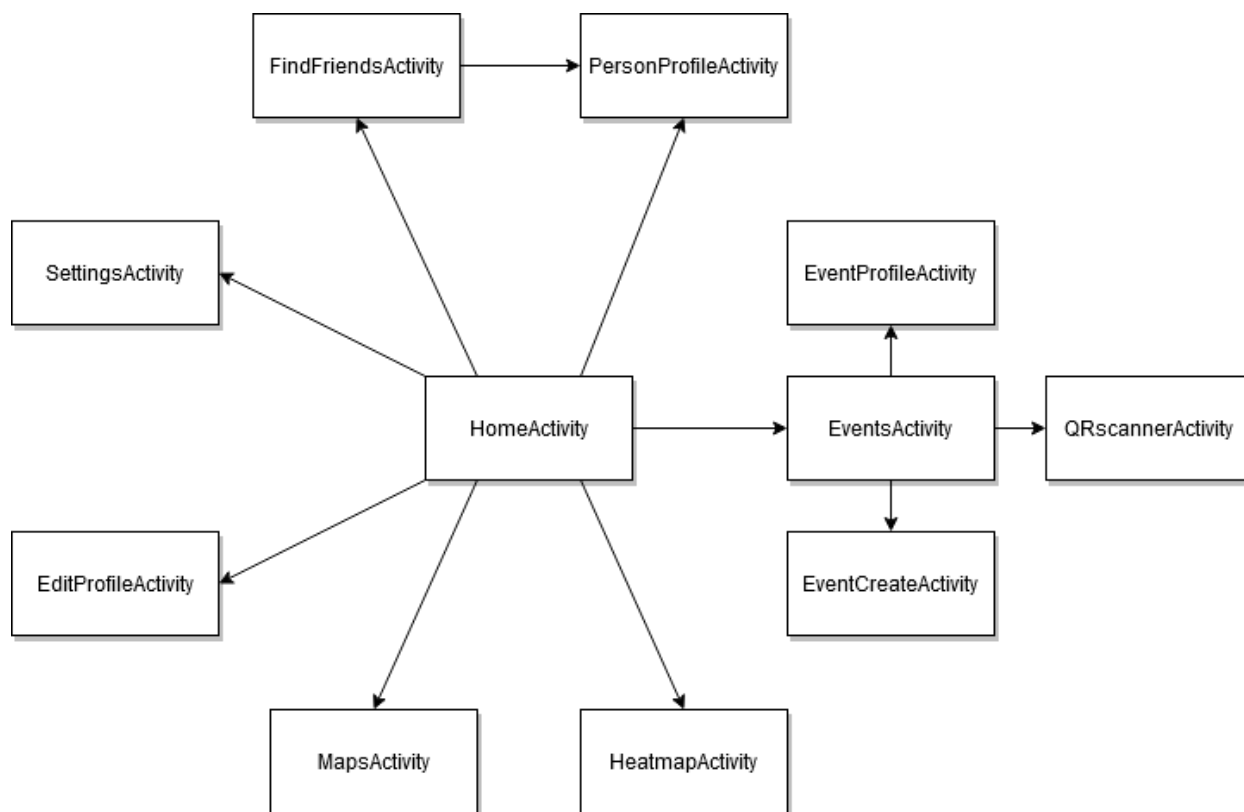
Основната структурна единица на Android приложението е Activity класът. Това е една страница на приложението (Главното меню, профила на приятел и т.н.) Activity-тата могат да се разглеждат като стек, когато ново Activity е стартирано, то се поставя най-отгоре на стека. Всяко

Activity има няколко стадия, на които са назначени различни методи, които могат да се пренапишат от програмиста.



Фигура 2.1 Жизненият цикъл на едно Activity

Приложението ще представлява няколко Activities с различна функционалност, представени на фигурата отдолу:



Фигура 2.2 Структура на приложението

2.3.2 Работа с Firebase Database

Firebase Database е NoSQL тип база данни намираща се в облак.

Информацията се пази като JSON и се синхронизира в реално време с всеки свързан клиент. Firebase работи и offline. Firebase Realtime Database SDK пази последното състояние на базата, което позволява потребителя да прави промени дори когато няма връзка с интернет, а когато клиента се свърже с интернет я синхронизира с облака.

2.3.2.1 Описание на колекцията Users

Структурата и примерни данни за Users може да се види на тази фигура:



Фигура 2.3 Колекцията Users

Всеки запис в Users е уникално id, генерирано от аутентикацията. В него има отделно записи за:

display_name - потребителско име на потребителя в низов тип

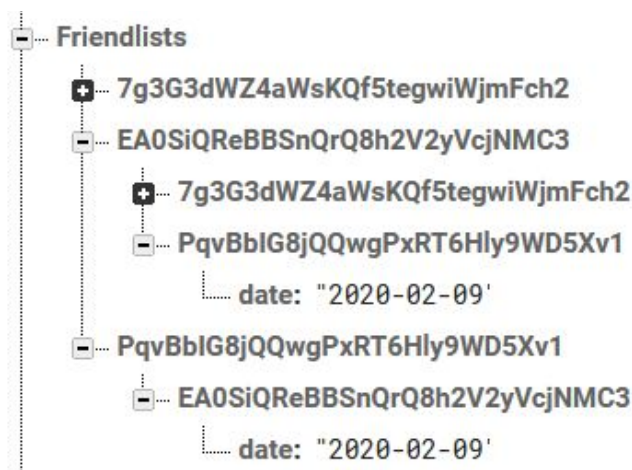
full_name - пълно име на потребителя в низов тип

gender - пол на потребителя в низов тип

Не е нужно да се пази електронната поща на потребителя в тази колекция, тъй като всички действия с нея се покриват от аутентикацията на Firebase и е много по-удобно и сигурно тя да бъде достъпвана от там.

2.3.2.2 Описание на колекцията Friendlists

Структурата и примерни данни за Friendlists може да се види на тази фигура:

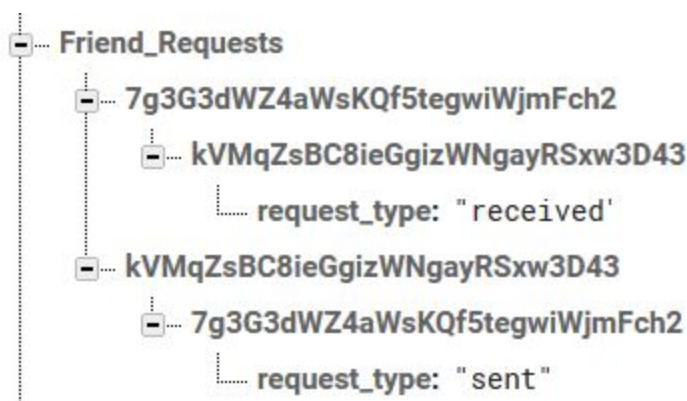


Фигура 2.4 Колекцията *Friendlists*

Всеки запис в тази колекция представлява листа от приятели на даден потребител, идентифициран по уникалното му id. Всеки запис в листа на даден потребител е id-то на друг потребител, показвайки по този начин, че те са приятели, а в него има поле date - датата, на която двамата са се сприятелили в уууу-ММ-dd date формат.

2.3.2.3 Описание на колекцията **Friend_Requests**

Структурата и примерни данни за Friend_Requests може да се види на тази фигура:

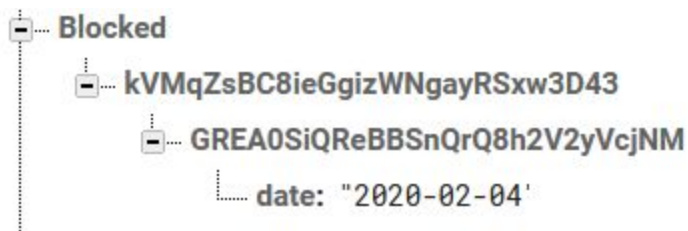


Фигура 2.5 Колекцията *Friend_Requests*

По подобен начин, както този в Friendlists записите в Friend_Requests са id на потребител съдържащо id на друг потребител, но в този случай второто id има поле request_type - типът на заявката в низов формат. Важно е да се отбележи, че при изпращането на покана за приятелство се правят два записа във Friend_Requests - една за потребителя, който е я пратил, и една за получателя

2.3.2.4 Описание на колекцията Blocked

Структурата и примерни данни за Blocked може да се види на тази фигура:

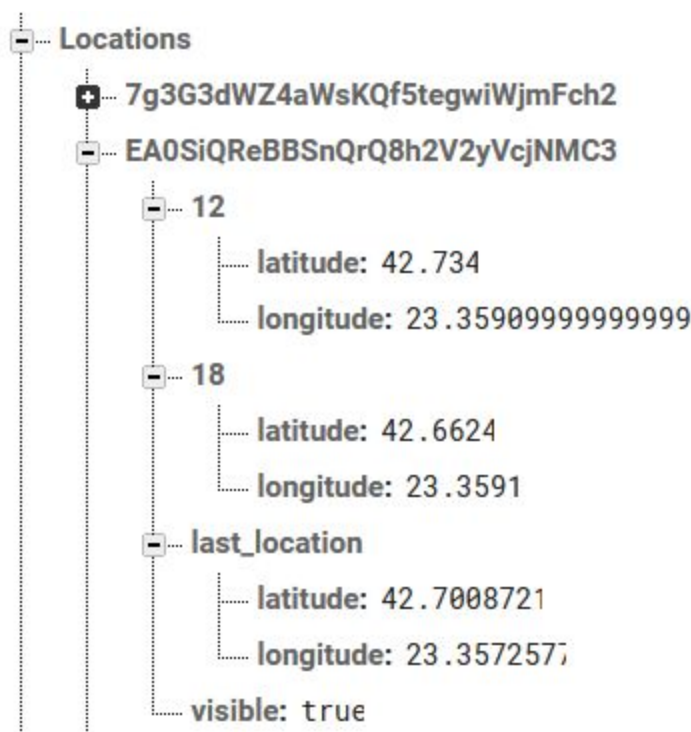


Фигура 2.6 Колекцията Blocked

Колекцията Blocked съхранява данни по напълно аналогичен на Friendlists начин. Единствената разлика между двете колекции е начина на ползването им в приложението.

2.3.2.5 Описание на колекцията Locations

Структурата и примерни данни за Locations може да се види на тази фигура:



Фигура 2.7 Колекцията *Locations*

Тази колекция показва локациите на потребителя. Всеки запис в нея представлява съответното id на потребителя с поле `visible`, чиято стойност е булева, определяща дали потребителя разрешава неговата локация да бъде показвана на картата, и три записа за локация:

`last_location` - представлява последните взети от потребителя координати, представени в `double` формат. Тази локация се използва за показване на потребителя на картата.

`12` - представлява последните взети от потребителя координати в диапазона между 12:00 и 13:00 часа, представени в `double` формат. Тези данни се използват за създаването на “Хийтмап” с популярните за посещение места.

18 - представлява последните взети от потребителя координати в диапазона между 18:00 и 19:00 часа, представени в double формат. Тези данни също се използват за “Хийтмап”.

Причината за избора точно на тези диапазони от време е, че статистически това са часовете, когато потребителите най-много ще използват приложението по предназначение (в обедна почивка или след работа) и по този начин данните ще са по-точни при голяма активност.

2.3.2.6 Описание на колекцията Radiuses

Структурата и примерни данни за Radiuses може да се види на тази фигура:



Фигура 2.8 Колекцията Radiuses

Записите в тази колекция отговарят на съответното id на потребителя и имат поле radius, което е избрания от потребителя радиус в метри, представен в long формат.

2.3.2.7 Описание на колекцията Events

Структурата и примерни данни за Events може да се види на тази фигура:



Фигура 2.9 Колекцията Events

Записите в колекцията Events отговарят на съответното id на събитието и имат полета:

date_start - начална дата в yyyy-MM-dd date формат.

desc - описание на събитието в низов формат.

ownerID - id на потребителя, създал събитието.

title - наименованието на събитието в низов формат.

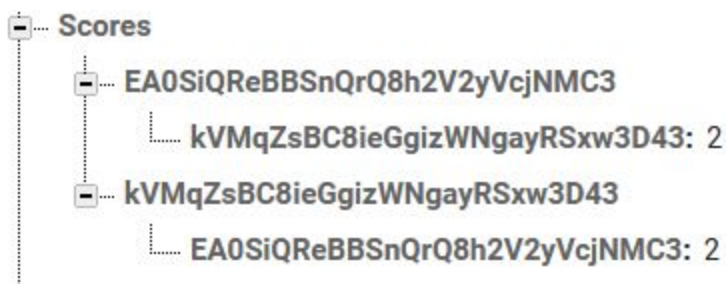
Освен тях, има и записи:

guests - колекция от потребители, всеки представен с съответното си id, присъстващи на събитието с поле date, представящо точната дата и време в което потребителят се е чекирал в събитието.

time_created и time_start, които представляват точната дата и време съответно на създаване на събитието и началото му, представени чрез всички полета на Date класа като ден от месеца, час, минута, изместване на часовата зона и тн. Това е направено така, с цел максимална точност при ползването на данните.

2.3.2.8 Описание на колекцията Scores

Структурата и примерни данни за Scores може да се види на тази фигура:



Фигура 2.10 Колекцията Scores

Колекцията Scores представя натрупания рейтинг между два потребителя.

Един запис отговарят на id на потребител и има поле с id на друг потребител и стойност техните точки. Отново има два записа за едно действие - по едно за всеки от двамата потребители.

Тази структура се възползва от удобствата на Firebase Realtime Database и осигурява бързина на достъп до нужната информация, като същевременно не усложнява процеса на работа с базата.

Трета Глава

Програмна реализация на приложение за срещи с приятели за Android

3.1 Аутентикация

Firebase Authentication предоставя разнообразни SDKs (Software development kits) и готови библиотеки за потребителски интерфейс, които да аутентекират потребителя дори през профилите им в популярни социални мрежи като Facebook, Twitter, дори Github.

Добавянето на Firebase Authentication става изключително лесно. В `app/build.gradle` се добавят зависимостта (dependency) за Firebase Authentication библиотеката - *“implementation 'com.google.firebase:firebase-auth:19.2.0’”* и тази за Facebook Login - *“implementation 'com.facebook.android:facebook-login:5.11.2’”*. След това от `developers.facebook.com` се взимат `id`-то и App secret хеша на приложението. В конзолата на Firebase се разрешава Facebook като метод за вписване и се попълват гореспоменатите `id` и `secret` в Auth секцията.

В кодовата част единствено остава да се инициализират `login` бутон и `LoginManager`.

```

41 // Initialize Facebook Login button
42 mCallbackManager = CallbackManager.Factory.create();
43 LoginButton loginButton = findViewById(R.id.buttonFacebookLogin);
44 loginButton.setPermissions("email", "public_profile");
45 loginButton.registerCallback(mCallbackManager, new FacebookCallback<LoginResult>() {
46     @Override
47     public void onSuccess(LoginResult loginResult) {
48         Log.d(TAG, "facebook:onSuccess:" + loginResult);
49         handleFacebookAccessToken(loginResult.getAccessToken());
50     }
51 }

```

Фигура 3.1 Login Button & CallbackManager

На бутона за вписване се задават разрешенията, които да бъдат ползвани когато потребителя се впише. След това се регистрира login callback към CallbackManager-а, това позволява, при успешно вписване във Facebook, да бъде извикан handleFacebookAccessToken(); метода, който да аутентикира потребителя и във Firebase.

3.2 Начална страница

Началната страница се намира в HomeActivity и представлява няколко бутона, които служат за навигиране из приложението :

- homeProfileBtn има закачен OnClickListener, чийто onClick метод е пренаписан да стартира и пренесе потребителя в PersonProfileActivity, като добавя като изпраща и допълнителна информация за id-то на потребителя.

- findFriendsBtn има закачен OnClickListener, чийто onClick метод е пренаписан да стартира и пренесе потребителя в FindFriendsActivity.

- editProfileBtn има закачен OnClickListener, чийто onClick метод е пренаписан да стартира и пренесе потребителя в EditProfileActivity.

- checkMapBtn има закачен OnClickListener, чийто onClick метод е пренаписан да стартира и пренесе потребителя в MapsActivity.

- eventsBtn има закачен OnClickListener, чийто onClick метод е пренаписан да стартира и пренесе потребителя в EventsActivity.
- settingsBtn има закачен OnClickListener, чийто onClick метод е пренаписан да стартира и пренесе потребителя в SettingsActivity.
- homePageLogoutBtn има закачен OnClickListener, чийто onClick метод е пренаписан да излезе от профила и да извика метода updateUI(). Той показва Toast съобщение, уведомяващо потребителя, че е излязъл от профила си и стартира и пренася потребителя в MainActivity - т.е. Login страницата

```
36      logoutBtn.setOnClickListener(new View.OnClickListener() {
37          @Override
38          public void onClick(View view) {
39              mAuth.signOut(); //this will logout from Firebase
40              LoginManager.getInstance().logout(); //this will logout from Facebook
41              updateUI();
42          }
43      });
44
45      openProfile.setOnClickListener((view) -> {
46          Intent profilePage = new Intent( packageContext: HomeActivity.this, UserProfileActivity.class);
47          startActivity(profilePage);
48      });
49
50      findFriends.setOnClickListener((view) -> {
51          Intent findFriendsPage = new Intent( packageContext: HomeActivity.this, FindFriendsActivity.class);
52          startActivity(findFriendsPage);
53      });
54
55      editProfile.setOnClickListener((view) -> {
56          Intent editProfile = new Intent( packageContext: HomeActivity.this, EditProfileActivity.class);
57          startActivity(editProfile);
58      });
59
60      checkMap.setOnClickListener((view) -> {
61          Intent maps = new Intent( packageContext: HomeActivity.this, MapsActivity.class);
62          startActivity(maps);
63      });
64
65      events.setOnClickListener((view) -> {
66          Intent eventsPage = new Intent( packageContext: HomeActivity.this, EventsActivity.class);
67          startActivity(eventsPage);
68      });
69
70      settings.setOnClickListener((view) -> {
71          Intent settingsPage = new Intent( packageContext: HomeActivity.this, SettingsActivity.class);
72          startActivity(settingsPage);
73      });
74  }
```

Фигура 3.2 Навигация

Методът onStart на MainActivity е пренаписан да следи дали потребителя има запис в колекцията Users, т.е. Дали е запазил своите потребителско име и пълно име, и ако не е, го пренася в EditProfileActivity.

```

106      @Override
107      public void onStart() {
108          super.onStart();
109          FirebaseAuth currentUser = mAuth.getCurrentUser();
110          if (currentUser == null) {
111              updateUI();
112          }
113          dbRef.addListenerForSingleValueEvent(new ValueEventListener() {
114              @Override
115              public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
116                  if (!dataSnapshot.child("Users").hasChild(mAuth.getCurrentUser().getUid())) {
117                      Intent setUser = new Intent( packageContext: HomeActivity.this, EditProfileActivity.class);
118                      startActivity(setUser);
119                  }
120              }
121              @Override
122              public void onCancelled(@NonNull DatabaseError databaseError) {
123              }
124          });
125      }
126  }
127
128  private void updateUI() {
129      Toast.makeText( context: HomeActivity.this, text: "You are logged out.", Toast.LENGTH_SHORT).show();
130
131      Intent HomePage = new Intent( packageContext: HomeActivity.this, MainActivity.class);
132      startActivity(HomePage);
133      finish();
134  }

```

Фигура 3.3 Методът updateUI()

3.3 Find Friends

Страницата за намиране на приятели представлява една търсачка с три елемента:

1. Поле за въвеждане на текст
2. Бутон за търсене
3. RecyclerView с резултатите от търсенето

Към бутона за търсене е закачен OnClickListener, чийто метод onClick() е пренаписан да взема написаното в полето за въвеждане и да извика метода SearchPeople()

```

private void searchPeople(String input) {
    Query searchPeopleQuery = allUsersRef.orderByChild("full_name").startAt(input).endAt(input + "\uf8ff");

    FirebaseRecyclerAdapter<User, FindFriendsViewHolder> firebaseRecyclerAdapter = new FirebaseRecyclerAdapter<>
    (
        User.class, R.layout.all_users_display_layout, FindFriendsViewHolder.class, searchPeopleQuery
    ) {
        @Override
        protected void populateViewHolder(FindFriendsViewHolder viewHolder, User user, final int position) {
            viewHolder.displayUser(user);

            viewHolder.mView.setOnClickListener((view) -> {
                String visit_user_id = getRef(position).getKey();

                Intent profileIntent = new Intent( packageContext, FindFriendsActivity.this, PersonProfileActivity.class);
                profileIntent.putExtra( name: "visit_user_id", visit_user_id);
                startActivity(profileIntent);
            });
        }
    };
    friendsList.setAdapter(firebaseRecyclerAdapter);
}

```

Фигура 3.4 Методът *searchPeople()*

Резултатите от търсенето са изложени в Recycler View. За целта трябва ViewHolder клас, направен специално за потребителите и техните полета. Той разширява RecyclerView. ViewHolder класа и има полета за пълно име, потребителско име и пол на потребителя.

```

85     public static class FindFriendsViewHolder extends RecyclerView.ViewHolder{
86         private View mView;
87         private TextView fullName;
88         private TextView displayName;
89         private TextView gender;
90
91         public FindFriendsViewHolder(@NonNull View itemView) {
92             super(itemView);
93             mView = itemView;
94             fullName = itemView.findViewById(R.id.all_users_fullname);
95             displayName = itemView.findViewById(R.id.all_users_display);
96             gender = itemView.findViewById(R.id.all_users_gender);
97         }
98
99
100     @
101     void displayUser(User user) {
102         fullName.setText(user.full_name);
103         displayName.setText(user.display_name);
104         gender.setText(user.gender);
105     }
106 }
107

```

Фигура 3.5 FindFriendsViewHolder

В метода SearchPeople() се създава Query което търси съвпадения между въведеното в полето за текст и пълните имена във Firebase базата данни. След което се създава нов FirebaseRecyclerAdapter, използвайки помощния клас User и ViewHolder. Методът populateViewHolder на адаптера е пренаписан да запълни полетата на ViewHolder и да закачи OnClickListener на всеки. Метода onClick на този Listener е пренаписан да стартира и пренесе потребителя в PersonProfileActivity, като също така изпраща и id на избрания потребителя, с цел да се зареди профила му.

```

9      public class User {
10         public String uid;
11         public String display_name;
12         public String full_name;
13         public String gender;
14         public Locale last_location;
15
16
17         @
18         public User() {
19             }
20
21         @
22         public User(String uid, String username, String fullName, String gender) {
23             this.uid = uid;
24             this.full_name = fullName;
25             this.display_name = username;
26             this.gender = gender;
27         }
28
29         @Exclude
30         public Map<String, Object> toMap() {
31             HashMap<String, Object> result = new HashMap<>();
32
33             result.put("display_name", display_name);
34             result.put("full_name", full_name);
35             result.put("gender", gender);
36
37             return result;
38         }
39     }

```

Фигура 3.6 Помощният клас User

3.4 Профили на потребители

3.4.1 Взимане на информация за потребителя

Към PersonProfileActivity пренасочват две страници :

FindFriendsActivity, през onClick() на ViewHolder класа за потребители и HomeActivity, през onClick() на бутона за профила на текущия потребител. И в двата случая се праща допълнителна информация между Activity-тата чрез метода Intent.putExtra("name", name). Тази информация се извлича след това с

метода `Intent.getExtras().get("name")` в `PersonProfileActivity`. В случая информацията, която се изпраща е `id`-то на избрания потребител.

Към референция към базата данни е закачен `ListenerForSingleValueEvent` за запис, с ключ полученото `id`, от колекцията потребители - `"Users"` и така се извличат пълното име, потребителското име и **пола** на потребителя.

Получената информация се задава в полетата за потребителско име, пълно име и **пол**. След това е извикан метода `maintainUI()`, който поддържа потребителския интерфейс.

3.4.2 MaintainUI();

За поддръжката на потребителския интерфейс се грижат метода `maintainUI` и `Enumerator-a FriendshipStatus`. `FriendshipStatus` има 4 възможни стойности - `NOT_FRIENDS`, `FRIENDS`, `REQUEST_SENT`, `REQUEST_RECEIVED`.

Методът `maintainUI()` закача `ListenerForSingleValueEvent` към референция за **"Friend_Requests"** колекцията на базата данни и проверява дали запис на текущия потребител има запис за потребителя, чийто профил бива разглеждан. Ако има, се запазва стойността на `"request_type"` запис в символен низ - `request_type`.

Ако `request_type` е `== "sent"`, `friendshipStatus` (инстанция на `FriendshipStatus enumerator-a`) става `"REQUEST_SENT"` и текста на бутона за изпращане на покана за приятелство се сменя на `"Cancel Friend Request"`

Ако `request_type` е `== "received"`, `friendshipStatus` става `"REQUEST_RECEIVED"` и текста на бутона за изпращане на покана за

приятелство се сменя на “Accept Friend Request”. Бутона за отказване на покана за приятелство става видим и активиран, а onClick(); метода на OnClickListener-а му извиква метода CancelFriendRequest();

В случая, че нито един от двамата потребители (текущия и този, чийто профил бива разглеждан) не е пратил покана за приятелство се закача ListenerForSingleValueEvent към референция за “**Friendlists**” колекцията на базата данни и се проверява дали записа за текущия потребител има запис за потребителя, чийто профил бива разглеждан. Ако има, friendshipStatus се променя на “FRIENDS” а текстът на бутона за изпращане на покана за приятелство се сменя на “Unfriend”

```
private void maintainUI() {
    friendRequestRef.child(senderUserID).addListenerForSingleValueEvent(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            if(dataSnapshot.hasChild(receiverUserID)){
                String request_type = dataSnapshot.child(receiverUserID).child("request_type").getValue().toString();

                if(request_type.equals("sent")){
                    friendshipStatus = FriendshipStatus.REQUEST_SENT;
                    sendBtn.setText("Cancel Friend Request");

                    declineBtn.setVisibility(View.INVISIBLE);
                    declineBtn.setEnabled(false);
                }
                else if(request_type.equals("received")){
                    friendshipStatus = FriendshipStatus.REQUEST_RECEIVED;
                    sendBtn.setText("Accept Friend Request");

                    declineBtn.setVisibility(View.VISIBLE);
                    declineBtn.setEnabled(true);

                    declineBtn.setOnClickListener((view) -> {
                        cancelFriendRequest();
                    });
                }
            }
            else {
                friendsRef.child(senderUserID).addListenerForSingleValueEvent(new ValueEventListener() {
                    @Override
                    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                        if(dataSnapshot.hasChild(receiverUserID)){
                            friendshipStatus = FriendshipStatus.FRIENDS;
                            sendBtn.setText("Unfriend");
                            declineBtn.setVisibility(View.INVISIBLE);
                            declineBtn.setEnabled(false);
                        }
                    }
                });
            }
        }
    });
}
```

Фигура 3.7 Методът *maintainUI()*

MaintainUI(); също оркестрира и блокирането на потребители като закача ListenerForSingleValueEvent към референция за “**Blocked**” колекцията на базата данни и проверява дали записа за текущия потребител има запис за потребителя, чийто профил бива разглеждан. Ако има, бутона за блокиране става деактивиран и невидим, а бутона за отблокиране - обратното. Ако няма, бутона за отблокиране става деактивиран и невидим, а бутона за блокиране - обратното.

```
21 blockedRef.child(senderUserID).addListenerForSingleValueEvent(new ValueEventListener() {
22     @Override
23     public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
24         if(dataSnapshot.hasChild(receiverUserID)){
25             blockBtn.setVisibility(View.INVISIBLE);
26             blockBtn.setEnabled(false);
27             unblockBtn.setVisibility(View.VISIBLE);
28             unblockBtn.setEnabled(true);
29         }
30         else if(!senderUserID.equals(receiverUserID)){
31             unblockBtn.setVisibility(View.INVISIBLE);
32             unblockBtn.setEnabled(false);
33             blockBtn.setVisibility(View.VISIBLE);
34             blockBtn.setEnabled(true);
35         }
36     }
}
```

Фигура 3.8 Методът *maintainUI()*

3.4.3 Покана за приятелство

За оркестриране на заявките за приятелство се използва инстанцията на Enumerator-a FriendshipStatus - friendshipStatus, според чиято стойност бутоните променят видимостта, текста или функционалността си.

Стойността ѝ се определя от метода MaintainUI();. По подразбиране е “NOT_FRIENDS”. Тогава бутона за отказване на покана за приятелство е невидим и деактивиран, а бутона за изпращане на покана има закачен OnClickListener, чийто onClick метод е пренаписан да извиква метода SendFriendRequest();

Когато стойността ѝ е “REQUEST_SENT” бутона за отказване на покана за приятелство е невидим и деактивиран, а бутона за изпращане на покана има закачен OnClickListener, чийто onClick метод е пренаписан да извиква метода CancelFriendRequest();

Когато стойността ѝ е “REQUEST_RECEIVED” бутона за отказване на покана за приятелство е видим и активиран, а бутона за изпращане на покана има закачен OnClickListener, чийто onClick метод е пренаписан да извиква метода AcceptFriendRequest();

Когато стойността ѝ е “FRIENDS” бутона за отказване на покана за приятелство е невидим и деактивиран, а бутона за изпращане на покана има закачен OnClickListener, чийто onClick метод е пренаписан да извиква метода Unfriend();

```

107     sendBtn.setOnClickListener(new View.OnClickListener() {
108         @Override
109         public void onClick(View view) {
110             sendBtn.setEnabled(false);
111             switch (friendshipStatus) {
112                 case NOT_FRIENDS:
113                     declineBtn.setVisibility(View.INVISIBLE);
114                     declineBtn.setEnabled(false);
115
116                     sendFriendRequest();
117                     break;
118                 case REQUEST_SENT:
119                     declineBtn.setVisibility(View.INVISIBLE);
120                     declineBtn.setEnabled(false);
121
122                     cancelFriendRequest();
123                     break;
124                 case REQUEST_RECEIVED:
125                     acceptFriendRequest();
126                     break;
127                 case FRIENDS:
128                     unfriend();
129                     break;
130             }
131         }
132     });
133     blockBtn.setOnClickListener((v) -> { block(); });

```

Фигура 3.9 Бутон за изпращане на покана

Бутонът за блокиране на потребител има закачен OnClickListener, чийто onClick метод е пренаписан да изпълнява метода Block();

Бутонът за отблокиране на потребител има закачен OnClickListener, чийто onClick метод е пренаписан да изпълнява метода Unblock();

Методът SendFriendRequest(); :

- задава “sent” за стойност на полето “request_type” на записа за текущия потребител на записа на разглеждания потребител на колекцията “Friend_Requests” на базата данни

- задава “recieved” за стойност на полето “request_type” на записа за разглеждания потребител на записа на текущия потребител на колекцията “Friend_Requests” на базата данни.

Използва се OnCompleteListener за да се подсили, че при невъзможност едното действие да се извърши и другото ще бъде прекратено. По този начин се избягва нежелано поведение.

A screenshot of a code editor showing the implementation of the `sendFriendRequest()` method. The code is in Java and uses the `friendRequestRef` object to manage database records. It first sets the `request_type` to "sent" and then, upon successful completion, sets it to "received". The method also updates the `friendshipStatus` to `REQUEST_SENT` and updates the UI by enabling the `sendBtn` and making the `declineBtn` invisible.

```
private void sendFriendRequest() {  
    friendRequestRef.child(senderUserID).child(receiverUserID)  
        .child("request_type").setValue("sent").addOnCompleteListener((task) -> {  
        if(task.isSuccessful()) {  
            friendRequestRef.child(receiverUserID).child(senderUserID)  
                .child("request_type").setValue("received").addOnCompleteListener((task)  
                if(task.isSuccessful()) {  
                    sendBtn.setEnabled(true); //true?  
                    friendshipStatus = FriendshipStatus.REQUEST_SENT;  
                    sendBtn.setText("Cancel Friend Request");  
  
                    declineBtn.setVisibility(View.INVISIBLE);  
                    declineBtn.setEnabled(false);  
                }  
            }  
        }  
    }  
}
```

Фигура 3.10 Методът `sendFriendRequest()`

Методът `CancelFriendRequest()` изтрива записа и на двамата потребители за другия от колекцията “Friend_Requests” на базата данни. Отново се използва `OnCompleteListener` за да се подсили, че при невъзможност едното действие да се извърши и другото ще бъде прекратено.

```

240 private void cancelFriendRequest() {
241     friendRequestRef.child(senderUserID).child(receiverUserID)
242         .removeValue()
243         .addOnCompleteListener((task) -> {
244             if(task.isSuccessful()){
245                 friendRequestRef.child(receiverUserID).child(senderUserID)
246                     .removeValue()
247                     .addOnCompleteListener((task) -> {
248                         if(task.isSuccessful()){
249                             sendBtn.setEnabled(true); //true?
250                             friendshipStatus = FriendshipStatus.NOT_FRIENDS;
251                             sendBtn.setText("Send Friend Request");
252
253                             declineBtn.setVisibility(View.INVISIBLE);
254                             declineBtn.setEnabled(false);
255                         }
256                     });
257             }
258         });
259     }
260 }
261
262
263
264
265
266

```

Фигура 3.11 Методът *cancelFriendRequest()*

Методът *AcceptFriendRequest()*; :

- Взима текущата дата и я запазва.
- Задава датата за стойност на полето “date” на записа за текущия потребител на записа на разглеждания потребител на колекцията “Friendlists” на базата данни.
- Задава датата за стойност на полето “date” на записа за разглеждания потребител на записа на текущия потребител на колекцията “Friendlists” на базата данни.
- Изтрива записа и на двамата потребители за другия от колекцията “Friend_Requests” на базата данни.

Отново се използва *OnCompleteListener* за да се подsigури, че при невъзможност предходното действие да се извърши и текущото няма да може.


```

31 private void acceptFriendRequest() {
32     Calendar date = Calendar.getInstance();
33     SimpleDateFormat currentDate = new SimpleDateFormat( pattern: "yyyy-MM-dd");
34     dateBefriended = currentDate.format(date.getTime());
35
36     friendsRef.child(senderUserID).child(receiverUserID).child("date").setValue(dateBefriended)
37     .addOnCompleteListener((task) -> {
38         if(task.isSuccessful()){
39             friendsRef.child(receiverUserID).child(senderUserID).child("date").setValue(dateBefriended)
40             .addOnCompleteListener((task) -> {
41                 if(task.isSuccessful()){
42                     friendRequestRef.child(senderUserID).child(receiverUserID)
43                     .removeValue()
44                     .addOnCompleteListener((task) -> {
45                         if(task.isSuccessful()){
46                             friendRequestRef.child(receiverUserID).child(senderUserID)
47                             .removeValue()
48                             .addOnCompleteListener((task) -> {
49                                 if(task.isSuccessful()){
50                                     sendBtn.setEnabled(true); //true?
51                                     friendshipStatus = FriendshipStatus.FRIENDS
52                                     sendBtn.setText("Unfriend");
53
54                                     declineBtn.setVisibility(View.INVISIBLE);
55                                     declineBtn.setEnabled(false);
56                                 }
57                             });
58                         }
59                     });
60                 }
61             });
62         }
63     });
64 }

```

Фигура 3.12 Методът *acceptFriendRequest()*

Методът *Unfriend()* :

- Изтрива записа и на двамата потребители за другия от колекцията “Friendlists” на базата данни.

Отново се използва *OnCompleteListener* за да се подсигури, че при невъзможност едното действие да се извърши и другото ще бъде прекратено.


```

private void unfriend() {
    friendsRef.child(senderUserID).child(receiverUserID)
        .removeValue()
        .addOnCompleteListener((task) -> {
            if(task.isSuccessful()){
                friendsRef.child(receiverUserID).child(senderUserID)
                    .removeValue()
                    .addOnCompleteListener((task) -> {
                        if(task.isSuccessful()){
                            sendBtn.setEnabled(true);
                            friendshipStatus = FriendshipStatus.NOT_FRIEND;
                            sendBtn.setText("Send Friend Request");

                            declineBtn.setVisibility(View.INVISIBLE);
                            declineBtn.setEnabled(false);
                        }
                    });
            }
        });
}
}

```

Фигура 3.13 Методът unfriend

Методът Block() запазва текущата дата в “yyyy-MM-dd” формат и я записва полето “date” на записа за разглеждания потребител на текущия потребител в колекцията “Blocked” на базата данни.

Методът Unblock() премахва записа, направен от Block();

```

149 private void block(){
150     Calendar date = Calendar.getInstance();
151     SimpleDateFormat currentDate = new SimpleDateFormat( pattern: "yyyy-MM-dd");
152     dateBlocked = currentDate.format(date.getTime());
153     blockedRef.child(senderUserID).child(receiverUserID).child("date").setValue(dateBlocked);
154     unblockBtn.setVisibility(View.VISIBLE);
155     unblockBtn.setEnabled(true);
156 }
157 private void unblock(){
158     blockedRef.child(senderUserID).child(receiverUserID).removeValue();
159     unblockBtn.setVisibility(View.INVISIBLE);
160     unblockBtn.setEnabled(false);
161 }

```

Фигура 3.14 Методите block() и unblock()

3.5 Карта

3.5.1 Google Maps SDK

Google Maps SDK предоставя лесна за интегриране и модулиране карта към Android приложението. Google Maps SDK се добавя към проекта като в `app/build.gradle` се добави зависимост за него - “implementation 'com.google.android.gms:play-services-maps:16.1.0'”

След това трябва да се постави генерирания от Google API ключ и разрешение за ползване на локацията в `AndroidManifest.xml` файла на проекта.

```
37 <meta-data
38     android:name="com.google.android.geo.API_KEY"
39     android:value="AIzaSyDwK4S8erxvKVm0M3WjiBE0yXXXXXXXXXXXX" />
40
```

Фигура 3.15 Ключът за Google Maps API

```
9 <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

Фигура 3.16 Разрешение за използване на локация

3.5.2 MapsActivity Setup

Класът `MapsActivity` разширява `FragmentActivity` и имплементира `OnMapReadyCallback`. В `onCreate` метода е инициализиран `SupportMapFragment`. Той поддържа основната функционалност на Google картата. Изгледът и картата се инициализират с `getMapAsync(OnMapReadyCallback)`.

```
68 SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
69     .findFragmentById(R.id.map);
70 mapFragment.getMapAsync(this);
```

Фигура 3.17 Създаване на картата

Не всички версии на Android поддържат картата и LocationManager. В случая е необходима минимална версия на Android API - 23, тоест, излязлата през 2015 - Android 6.0. За това над onMapReady методът е поставена анотацията - @RequiresApi(api = Build.VERSION_CODES.M)

LocationManager класът дава достъп до локацията на мобилното устройство и услугите свързани с нея. Инициализира се с
getSystemService(LOCATION_SERVICE);

Проверява се дали GPS доставчика е активиран. Извикан е метода на LocationManager - requestLocationUpdates(), който предоставя регистър на промените в гео-локацията на мобилното устройство. Като параметри са подадени доставчика, времето между всеки **update** и минималната дистанция, с която локацията трябва да се е променила, както и LocationListener.

```
194 } else if(locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER)){
195     locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, minTime: 5000, minDistance: 0, new LocationListener() {
```

Фигура 3.18 GPS доставчик

LocationListener има метод onLocationChanged, който се извиква при всеки **update**. Пренаписан е да записва географската ширина и дължина на устройството в базата данни, както и кога са взети.

```
121 final double latitude = location.getLatitude();
122 final double longitude = location.getLongitude();
123
124 currentLatLng = new LatLng(latitude, longitude);
125 Date date = Calendar.getInstance().getTime();
126 SimpleDateFormat sdf = new SimpleDateFormat( pattern: "yyyy-MM-dd HH:mm:ss");
127 String lastSeenOn = sdf.format(date);
128 locationsRef.child(currentUserID).child("last_seen_on").setValue(lastSeenOn);
129
130 locationsRef.child(currentUserID).child("last_location").child("latitude").setValue(latitude);
131 locationsRef.child(currentUserID).child("last_location").child("longitude").setValue(longitude);
```

Фигура 3.19 Запазване на локацията в базата

3.5.3 Представяне на информация върху картата

Потребителите са изобразени като маркери върху картата. За да се постави маркер на дадена локация се извика метода `map.addMarker()`, като аргумент е подадена инстанция на класа `MarkerOptions`, който има няколко опционални полета за видимост, икона, прозрачност, плоскост и др.

Радиусът, в който потребителят може да вижда приятелите си е представен от кръг с център локацията на потребителя и радиус, взет от базата данни, предварително зададен в `SettingsActivity`.

За изобразяване на **сегашният** потребител е използван син маркер.

```
414         currentLoc = mMap.addMarker(new MarkerOptions()
415             .position(new LatLng(latitude, longitude))
416             .icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_AZURE))
417         );
418
419         mMap.addCircle(new CircleOptions()
420             .center(currentLatLng)
421             .radius((float) rad)
422             .fillColor(Color.argb( alpha: 70,  red: 150,  green: 50,  blue: 50))
423         );
```

Фигура 3.20 Маркер и радиус около текущия потребител

За изобразяване на всички останали потребители е закачен `ValueEventListener` към колекцията за локация от базата данни. Той предоставя `dataSnapshot` - моменталното състояние на тази част от базата. Методът `dataSnapshot.getChildren()` връща списък със `snapshots`, като всеки `snapshot` представлява един запис от колекцията за локации, т.е. Това е списък от всички потребители и техните локации.

Всеки `snapshot` се разглежда във `for each` цикъл. Проверява се дали ключа - `id`-то на разглеждания не е съвпада с това на сегашният потребител. Методът `isInRadius()` проверява дали дистанцията между две подадени локации е по-малка или равна на радиуса, предварително зададен в `SettingsActivity`. Това става, като към референция към базата е закачен

ListenerForSingleValueEvent. Той предоставя dataSnapshot, от който е извлечен радиуса на текущия потребител от колекцията за радиуси - ”Radiuses”. Използван е ListenerForSingleValueEvent, а не ValueEventListener, защото справката е нужна само веднъж.

```
286 @ private boolean isInRadius(LatLng latLng, LatLng currentLatLng) {
287     float[] distance = new float[1];
288
289     Location.distanceBetween(latLng.latitude, latLng.longitude, currentLatLng.latitude, currentLatLng.longitude, distance);
290     dbRef.addListenerForSingleValueEvent(new ValueEventListener() {
291         @Override
292         public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
293             if(dataSnapshot.child("Radiuses").hasChild(currentUserID)) {
294                 rad = (long) dataSnapshot.child("Radiuses").child(currentUserID).child("radius").getValue();
295             }
296         }
297         @Override
298         public void onCancelled(@NonNull DatabaseError databaseError) {
299         }
300     });
301     return distance[0] <= rad;
302 }
303 }
```

Фигура 3.21 Проверка за радиуса

Ако потребител е избрал в SettingsActivity да не бъде откриваем, то той не трябва да се показва на картата и за това има поставена проверка.

Проверява се дали разглеждания е приятел на настоящия потребител. Това става, като се провери дали в dataSnapshot в колекцията за списъци с приятели - “Friendlists” в записа за id на текущия потребител има запис за id на разглеждания. По аналогичен начин се преглежда дали един от потребителите не е блокирал другия, но в колекцията за списъци с блокирани потребители - “Blocked”.

Ако разглеждания потребител отговаря на всички критерии, на картата се добавя маркер с неговата информация на неговата локация.

```
145 for(final DataSnapshot user : dataSnapshot.getChildren()){
146     if(user.hasChild( path: "last_location")) {
147         final double user_latitude = (double) user.child("last_location").child("latitude").getValue();
148         final double user_longitude = (double) user.child("last_location").child("longitude").getValue();
149         if (!currentUserID.equals(user.getKey()))
150             && isInRadius(new LatLng(user_latitude, user_longitude), currentLatLng)
151             && (user.hasChild( path: "visible") && (boolean)user.child("visible").getValue()) {
152             dbRef.addValueEventListener(new ValueEventListener() {
153                 @Override
154                 public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
155                     if(dataSnapshot.child("Friendlists").child(currentUserID).hasChild(user.getKey()))
156                         && !dataSnapshot.child("Blocked").child(currentUserID).hasChild(user.getKey())
157                     {
158                         mMap.addMarker(new MarkerOptions()
159                             .position(new LatLng(user_latitude, user_longitude))
160                             .title(dataSnapshot.child("Users").child(user.getKey()).child("full_name").getValue().toString()));
161                     }
162                 }
163             });
164         }
165     }
```

Фигура 3.22 Проверки за изобразяване на приятели

3.5.4 Heatmap

“Хийт Мап” е изнесен в отделно Activity - HeatmapActivity.

За изобразяване на Heatmap е необходим HeatmapTileProvider класа, който се намира в Google Maps Android API utility библиотеката.

Библиотеката се добавя към проекта като в app/build.gradle се добави зависимост към нея - *“implementation*

’com.google.maps.android:android-maps-utils:0.5”. Предварителната конфигурация е аналогична с тази на MapsActivity, с тази разлика, че тук е нужен само SupportMapFragment.

След като картата е заредена, от колекцията за локации “Locations” в базата данни се извличат всички координати от колекциите “12” и “18” и се записват в списъка в масиви data. След това е инициализиран HeatmapTileProvider с въпросната data и е зададен като TileOverlay на картата.


```

48      @Override
49      public void onMapReady(GoogleMap googleMap) {
50          mMap = googleMap;
51
52          data = new ArrayList<>();
53          dbRef.child("Locations").addValueEventListener(new ValueEventListener() {
54              @Override
55              public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
56                  for (DataSnapshot snapshot : dataSnapshot.getChildren()) {
57                      if (snapshot.hasChild( path: "12")) {
58                          double latitude = (double) snapshot.child("12").child("latitude").getValue();
59                          double longitude = (double) snapshot.child("12").child("longitude").getValue();
60                          data.add(new LatLng(latitude, longitude));
61                      }
62                      if (snapshot.hasChild( path: "18")) {
63                          double latitude = (double) snapshot.child("18").child("latitude").getValue();
64                          double longitude = (double) snapshot.child("18").child("longitude").getValue();
65                          data.add(new LatLng(latitude, longitude));
66                      }
67                  }
68                  HeatmapTileProvider mProvider = new HeatmapTileProvider.Builder()
69                      .data(data)
70                      .build();
71                  mMap.addTileOverlay(new TileOverlayOptions().tileProvider(mProvider));
72              }
73          });

```

Фигура 3.23 Събиране на локации и създаване на *HeatmapTileProvider*

3.6 Събития

3.6.1 Основно activity

Основното меню за събитията може да се раздели на 3 части:

1. Търсачка за събития
2. Бутон за създаване на събитие
3. Бутон за чекиране на събитие

3.X.1.1 Търсачка за събития

Търсачката има 3 елемента:

1. Поле за въвеждане на текст
2. Бутон за търсене
3. RecyclerView с резултатите от търсенето

Към бутона за търсене е закачен `OnClickListener`, чийто метод `onClick()` е пренаписан да взема написаното в полето за въвеждане и да извика метода `SearchEvents()`

```
49 searchEventsBtn.setOnClickListener(new View.OnClickListener() {
50     @Override
51     public void onClick(View view) {
52         String input = searchInputText.getText().toString();
53         SearchEvents(input);
54     }
55 });
```

Фигура 3.24 Търсене на събития

Резултатите от търсенето са изложени в `RecyclerView`. За целта е използван `ViewHolder` клас, направен специално за събитията и техните полета. Той **разширява** `RecyclerView`. `ViewHolder` класа и има полета за заглавие, описание и начална дата и час на събитието.

```
103 public static class FindEventsViewHolder extends RecyclerView.ViewHolder{
104     View mView;
105
106     public FindEventsViewHolder(@NonNull View itemView) {
107         super(itemView);
108         mView = itemView;
109     }
110
111     public void setTitle(String title){
112         TextView eTitle = (TextView) mView.findViewById(R.id.all_events_title);
113         eTitle.setText(title);
114     }
115     public void setDesc(String desc){
116         TextView eDesc = (TextView) mView.findViewById(R.id.all_events_desc);
117         eDesc.setText(desc);
118     }
119     public void setStartTime(String startTime){
120         TextView eStartTime = (TextView) mView.findViewById(R.id.all_events_startTime);
121         eStartTime.setText(startTime);
122     }
123 }
124 }
125
```

Фигура 3.25 `FindEventsViewHolder` класът

В метода SearchEvents() се създава Query което търси съвпадения между въведеното в полето за текст и заглавията във колекцията “Events” на базата данни. След което се създава нов FirebaseRecyclerAdapter, използвайки помощния клас Event и ViewHolder класа. Методът populateViewHolder на адаптера е пренаписан да запълни полетата на съответния ViewHolder и да закачи OnClickListener на всеки. Метода onClick на този Listener е пренаписан да стартира и пренесе потребителя в EventProfileActivity, като също така изпраща и id на избраното събитие, с цел да се зареди профила му.

```
9      public class Event {
10
11          public Date dateCreated;
12          public Date time_start;
13          public String title;
14          public String desc;
15          public String start_time_as_string;
16
17          @
18          public Event() {
19
20          }
21
22          @
23          public Event(Date dateCreated, Date dateStarted, String title, String description) {
24              this.dateCreated = dateCreated;
25              this.time_start = dateStarted;
26              this.title = title;
27              this.desc = description;
28          }
29
30
31          @Exclude
32          public Map<String, Object> toMap() {
33              HashMap<String, Object> result = new HashMap<>();
34
35              result.put("title", title);
36              result.put("desc", desc);
37              result.put("time_started", time_start);
38              result.put("start_time_as_string", time_start.toString());
39
40              return result;
41          }
42      }
43
```

Фигура 3.26 Помощният клас Event

```

75 private void SearchEvents(String input) {
76     Query searchEventsQuery = eventsRef.orderByChild("title").startAt(input).endAt(input + "\uf8ff");
77
78     FirebaseRecyclerAdapter<Event, EventsActivity.FindEventsViewHolder> firebaseRecyclerAdapter = new FirebaseRecyclerAdapter<
79     {
80         Event.class, R.layout.all_events_display_layout, EventsActivity.FindEventsViewHolder.class, searchEventsQuery
81     } {
82         @Override
83         protected void populateViewHolder(EventsActivity.FindEventsViewHolder viewHolder, Event event, final int position) {
84             viewHolder.setTitle(event.title);
85             viewHolder.setDesc(event.desc);
86             viewHolder.setStartTime(String.valueOf(event.time_start));
87
88             viewHolder.mView.setOnClickListener(new View.OnClickListener() {
89                 String visit_event_id = getRef(position).getKey();
90
91                 Intent eventIntent = new Intent( packageContext.EventsActivity.this, EventProfileActivity.class);
92                 eventIntent.putExtra( name:"visit_event_id", visit_event_id);
93                 startActivity(eventIntent);
94             });
95         }
96     };
97     Results.setAdapter(firebaseRecyclerAdapter);
98 }
99
100
101
102

```

Фигура 3.27 Методът searchEvents

3.6.2 Създаване на събитие

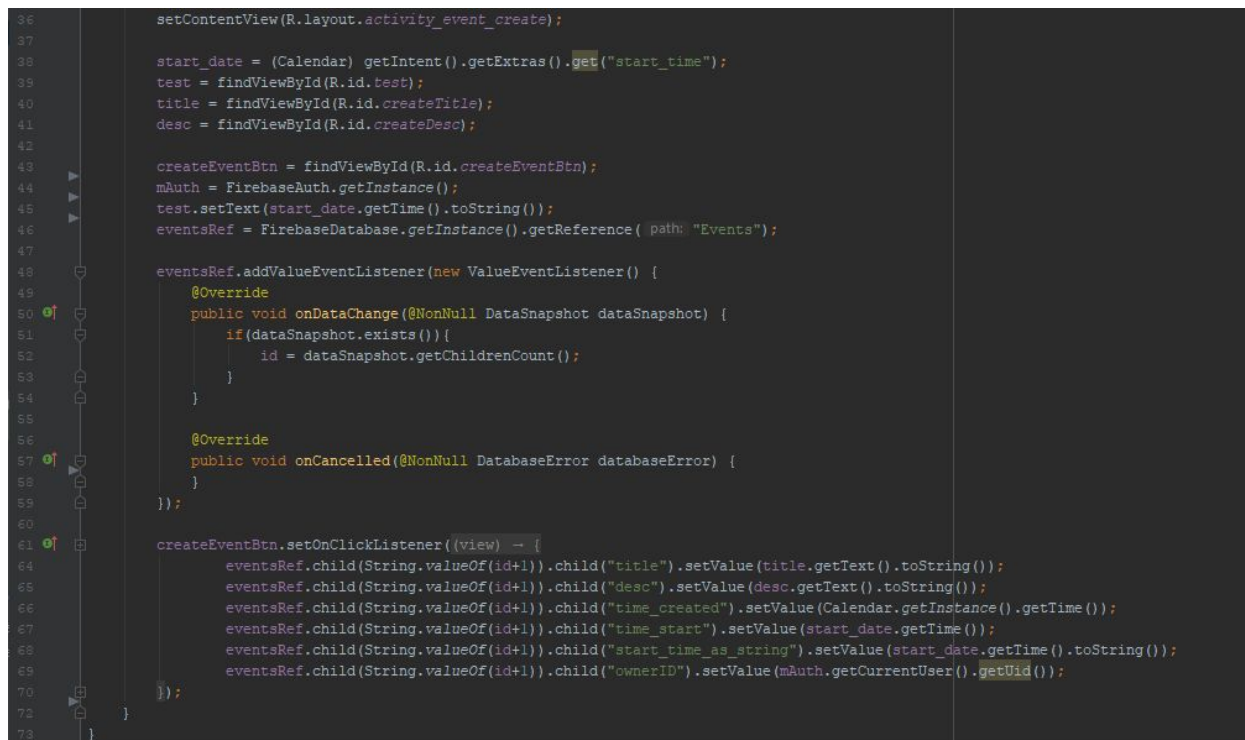
Към бутона за създаване на събитие е закачен OnClickListener, чийто метод OnClick() е пренаписан да зарежда нов Intent и да пренася към EventPickDateTimeActivity.

Там има DatePicker и TimePicker, чрез които потребителят избира началните дата и час на събитието, както и бутон, който прочита избраното от picker-ите и зарежда нов Intent и да пренася към EventCreateActivity.

В него има две полета за попълване на текст, един бутон за създаване и TextView. Полетата за попълване са съответно потребителя за въведе заглавие и описание на събитието. TextView-то представя избраните преди това дата и час в разбираем формат.

За да се различават събитията един от друг, те трябва да имат различно id. При създаване на събитие неговото id се създава като се гледа броя на събития в базата и се инкрементира с 1.

Към бутона за създаване на събитие е закачен `OnClickListener`, чийто метод `onClick()` е пренаписан да запази в базата данни съответните полета на събитието.



```
36 setContentView(R.layout.activity_event_create);
37
38 start_date = (Calendar) getIntent().getExtras().get("start_time");
39 test = findViewById(R.id.test);
40 title = findViewById(R.id.createTitle);
41 desc = findViewById(R.id.createDesc);
42
43 createEventBtn = findViewById(R.id.createEventBtn);
44 mAuth = FirebaseAuth.getInstance();
45 test.setText(start_date.getTime().toString());
46 eventsRef = FirebaseDatabase.getInstance().getReference("Events");
47
48 eventsRef.addValueEventListener(new ValueEventListener() {
49     @Override
50     public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
51         if(dataSnapshot.exists()){
52             id = dataSnapshot.getChildrenCount();
53         }
54     }
55
56     @Override
57     public void onCancelled(@NonNull DatabaseError databaseError) {
58     }
59 });
60
61 createEventBtn.setOnClickListener((view) -> {
62     eventsRef.child(String.valueOf(id+1)).child("title").setValue(title.getText().toString());
63     eventsRef.child(String.valueOf(id+1)).child("desc").setValue(desc.getText().toString());
64     eventsRef.child(String.valueOf(id+1)).child("time_created").setValue(Calendar.getInstance().getTime());
65     eventsRef.child(String.valueOf(id+1)).child("time_start").setValue(start_date.getTime());
66     eventsRef.child(String.valueOf(id+1)).child("start_time_as_string").setValue(start_date.getTime().toString());
67     eventsRef.child(String.valueOf(id+1)).child("ownerID").setValue(mAuth.getCurrentUser().getUid());
68 });
69
70
71
72
73 }
```

Фигура 3.28 Създаване и запазване на Event

3.6.3 Профил на събитието

След като потребителя е избрал събитие е пренесен към неговия профил. Там има три `TextViews`, които предоставят информацията извлечена от базата за съответното събитие. Освен това, има и `ImageView` с битова матрица за QR код, което е видимо само ако потребителя е създател на събитието.

```

91      eventsRef.child(eventID).addValueEventListener(new ValueEventListener() {
92          @Override
93          public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
94              String db_title = dataSnapshot.child("title").getValue().toString();
95              String db_desc = dataSnapshot.child("desc").getValue().toString();
96              String db_start_time = (dataSnapshot.child("start_time_as_string").getValue().toString());
97              if (mAuth.getCurrentUser().getUid().equals( dataSnapshot.child("ownerID").getValue().toString())){
98                  MultiFormatWriter multiFormatWriter = new MultiFormatWriter();
99
100                 Bitmap bitMatrix = null;
101                 try {
102                     bitMatrix = multiFormatWriter.encode(eventID, BarcodeFormat.QR_CODE, width: 500, height: 500);
103                 } catch (WriterException e) {
104                     e.printStackTrace();
105                 }
106                 BarcodeEncoder barcodeEncoder = new BarcodeEncoder();
107                 Bitmap bitmap = barcodeEncoder.createBitmap(bitMatrix);
108                 QRcode.setImageBitmap(bitmap);
109             }
110             title.setText( db_title);
111             desc.setText(db_desc);
112             time_start.setText(db_start_time);
113         }
114     }

```

Фигура 3.29 Вземане и използване на информацията за събитието

Има също и бутон за споделяне на събитието. Към него е закачен `OnClickListener`, чийто метод `OnClick()` е пренаписан да създава `ShareLinkContent` и да задава съответен текст, hashtag и линк към google maps с локация. С помощта на Facebook SDK този `ShareLinkContent` се споделя на стената на потребителя.

```

118      shareBtn.setOnClickListener((View) -> {
119          final String text = " is attending " + title.getText() + " with ";
120
121          dbRef.addListenerForSingleValueEvent(new ValueEventListener() {
122              @Override
123              public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
124                  String userName = dataSnapshot.child("Users").child(currentUserID).child("full_name").getValue().toString();
125                  String latitude = dataSnapshot.child("Locations").child(currentUserID).child("last_location").child("latitude").getValue().toString();
126                  String longitude = dataSnapshot.child("Locations").child(currentUserID).child("last_location").child("latitude").getValue().toString();
127
128                  ShareLinkContent linkContent = new ShareLinkContent.Builder()
129                      .setQuote(userName + text)
130                      .setContentUrl(Uri.parse("https://www.google.com/maps/search/google+maps/@"+ latitude + ", " + longitude + ",15.89z"))
131                      .setShareHashtag(new ShareHashtag.Builder()
132                          .setHashtag("#UsingPINS")
133                          .build())
134                      .build();
135                  if (ShareDialog.canShow(ShareLinkContent.class)){
136                      shareDialog.show(linkContent);
137                  }
138              }
139
140              @Override
141              public void onCancelled(@NonNull DatabaseError databaseError) {
142              }
143          });
144      }

```

Фигура 3.30 Споделяне във Facebook

3.6.4 Чекиране на събитие

Бутона за сканиране на събитие от главното меню за събития отвежда в QRscannerActivity, което имплементира ZXingScannerView.ResultHandler. При създаването на QRscannerActivity се иска позволение от потребителя да използва камерата на мобилното устройство.

```
53         if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
54             if (CheckPermission()) {
55                 Toast.makeText(context, QRscannerActivity.this, text: "Permission granted", Toast.LENGTH_LONG).show();
56             }
57             else {
58                 RequestPermission();
59             }
60         }
61     }
62
63     private void RequestPermission() {
64         ActivityCompat.requestPermissions(activity, this, new String[]{CAMERA}, REQUEST_CAMERA);
65     }
66
67     private boolean CheckPermission() {
68         return ContextCompat.checkSelfPermission(context, QRscannerActivity.this, CAMERA) == PackageManager.PERMISSION_GRANTED;
69     }
```

Фигура 3.31 Проверка на позволение за ползване на камерата

След като приложението има позволение да ползва камерата, ZXing се опитва да разпознае QR код през камерата. Потребителя трябва да сканира QR кода от телефона на създателя на събитието.

Методът `handleResult()` на `ZXingScannerView.ResultHandler` е пренаписан да проверява дали потребителя вече не е записан в базата като “присъстващ” на събитието, чието `id` е резултатът от сканирането на QR кода. След това създава диалог, който пита потребителя дали иска да се чекира като присъстващ на събитието и го пренася в профилната страница на събитието, а ако вече е “присъстващ” го информира за това. След като потребителя се чекира в колекцията за посетители “`guests`” на събитието се прави запис с `id`-то на потребителя и поле `date` - текущата дата в `yyyy-MM-dd`

формат. Освен това се извиква метода `increaseScore()` за всеки присъстващ, който увеличава рейтинга му със сегашния потребител.

```
219     private void increaseScore(final String user) {
220         dbRef.addListenerForSingleValueEvent(new ValueEventListener() {
221             @Override
222             public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
223                 long score;
224                 if(dataSnapshot.child("Scores").child(user).hasChild(currentUserId)) {
225                     score = (long) dataSnapshot.child("Scores").child(user).child(currentUserId).getValue();
226                     score += 1;
227                 } else {
228                     score = 1;
229                 }
230                 dbRef.child("Scores").child(user).child(currentUserId).setValue(score);
231                 dbRef.child("Scores").child(currentUserId).child(user).setValue(score);
232             }
233
234             @Override
235             public void onCancelled(@NonNull DatabaseError databaseError) {
236
237             }
238         });
239     }
240 }
```

Фигура 3.32 Методът `increaseScore()` за трупане на рейтинг


```

111 @ public void handleResult(Result rawResult) {
112     final String scanResult = rawResult.getText();
113
114     eventsRef.child(scanResult).addListenerForSingleValueEvent(new ValueEventListener() {
115         @Override
116         public void onDataChange(@NonNull final DataSnapshot dataSnapshot) {
117             if(!dataSnapshot.child("guests").hasChild(currentUserId)){
118                 AlertDialog.Builder builder = new AlertDialog.Builder( context: QRscannerActivity.this);
119                 builder.setTitle("scan result");
120                 builder.setPositiveButton( text: "OK", (dialogInterface, i) -> {
121                     Calendar date = Calendar.getInstance();
122                     SimpleDateFormat currentDate = new SimpleDateFormat( pattern: "yyyy-MM-dd");
123                     String date_attended = currentDate.format(date.getTime());
124                     final String ownerId = dataSnapshot.child("ownerID").getValue().toString();
125                     eventsRef.child(scanResult).child("guests").child(currentUserId).child("date")
126                         .setValue(date_attended);
127                     eventsRef.child(scanResult).child("guests").addListenerForSingleValueEvent(new ValueEventListener() {
128                         @Override
129                         public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
130                             for(DataSnapshot snapshot : dataSnapshot.getChildren()){
131                                 if(!currentUserId.equals(snapshot.getKey())) {
132                                     increaseScore(snapshot.getKey());
133                                 }
134                             }
135                             if(!currentUserId.equals(ownerID)) {
136                                 increaseScore(ownerID);
137                             }
138                         }
139                     }
140                     @Override
141                     public void onCancelled(@NonNull DatabaseError databaseError) {
142                     }
143                 });
144                 Intent eventPage = new Intent( packageContext: QRscannerActivity.this, EventProfileActivity.class);
145                 eventPage.putExtra( name: "visit_event_id", scanResult);
146                 startActivity(eventPage);
147             });
148             builder.setNegativeButton( text: "Cancel", (dialogInterface, i) -> {
149                 scannerView.resumeCameraPreview( resultHandler: QRscannerActivity.this);
150             });
151             builder.setMessage("Do you want to join " + dataSnapshot.child("title").getValue().toString() + "?");
152             AlertDialog alertDialog = builder.create();
153             alertDialog.show();
154         }
155     }
156 }
157 else {
158     AlertDialog.Builder builder = new AlertDialog.Builder( context: QRscannerActivity.this);
159     builder.setTitle("scan result");
160     builder.setPositiveButton( text: "OK", (dialogInterface, i) -> {
161         Intent eventPage = new Intent( packageContext: QRscannerActivity.this, EventProfileActivity.class);
162         eventPage.putExtra( name: "visit_event_id", scanResult);
163         startActivity(eventPage);
164     });
165     builder.setMessage("You have already joined " + dataSnapshot.child("title").getValue().toString());
166     AlertDialog alertDialog = builder.create();
167     alertDialog.show();
168 }
169 }
170
171 @Override
172 public void onCancelled(@NonNull DatabaseError databaseError) {
173
174 }
175
176 }
177
178 }
179
180 }

```

Фигура 3.33 Методът *handleResult()*

3.7 Настройки

Настойките на приложението се намират в *SettingsActivity*. В него има бутон за запазване на радиуса, *TextView*, което изписва радиуса, *SeekBar*, с който

потребителя избира радиуса в който иска да открива приятели на картата, както и превключвател (switch), с който потребителя избира дали иска да бъде показван на картата на приятелите си.

SeekBar представлява плъзгач, чиято стойност се увеличава от ляво надясно. В случая потребителя може да избира радиус от 0m до 25km.

Методът `onCheckedChanged`, на превключвателя, който се извиква при промяна в състоянието му, е пренаписан да задава за стойност сегашното си състояние в полето `visible` в записа за текущия потребител в колекцията “Locations” на базата данни.

На бутона за запазване на радиуса е закачен `onClick` listener, чийто `onClick` метод е пренаписан да запазва зададената стойност на seekbar-а в полето `radius` на записа за сегашния потребител в колекцията “Radiuses” на базата данни.

Със зареждането на `SettingsActivity` се извикват методите `maintainUI()` и `manipulateSeekBar()`. `MaintainUI()` се грижи да запази състоянието на плъзгача и превключвателя, като го задава според стойностите на съответните полета в базата данни. `ManipulateSeekBar()` променя стойността на текстовото поле за дължината на радиуса всеки път, когато той се промени.


```

68 private void maintainUI() {
69     dbref.addValueEventListener(new ValueEventListener() {
70         @Override
71         public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
72             if(dataSnapshot.child("Locations").child(currentUserID).hasChild( path: "visible")) {
73                 boolean visibility = (boolean) dataSnapshot.child("Locations").child(currentUserID).child("visible").getValue();
74                 if (visibility) {
75                     visibilitySwitch.setChecked(true);
76                     visibilityText.setText("Visible");
77                 } else {
78                     visibilityText.setText("Not Visible");
79                 }
80             }
81             if(dataSnapshot.child("Radiuses").hasChild(currentUserID)){
82                 long rad = (long) dataSnapshot.child("Radiuses").child(currentUserID).child("radius").getValue();
83                 seekBar.setProgress(((int) rad)/250);
84             }
85         }
86         @Override
87         public void onCancelled(@NonNull DatabaseError databaseError) {
88         }
89     });
90 }
91
92 public void manipulateSeekBar() {
93     seekBar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
94         int progress_value;
95         @Override
96         public void onProgressChanged(SeekBar seekBar, int i, boolean b) {
97             progress_value = i;
98
99             if(progress_value >= 16){
100                 radiusText.setText(progress_value/4 + "." + (progress_value%4)*250 + "km");
101             }else {
102                 radiusText.setText(progress_value*250 + "m");
103             }
104         }
105         @Override
106         public void onStartTrackingTouch(SeekBar seekBar) {
107         }
108         @Override
109         public void onStopTrackingTouch(SeekBar seekBar) {
110         }
111     });
112 }
113 }

```

Фигура 3.34 Методите maintainUI() & manipulateSeekBar()

Четвърта Глава

Изисквания. Ръководство на потребителя

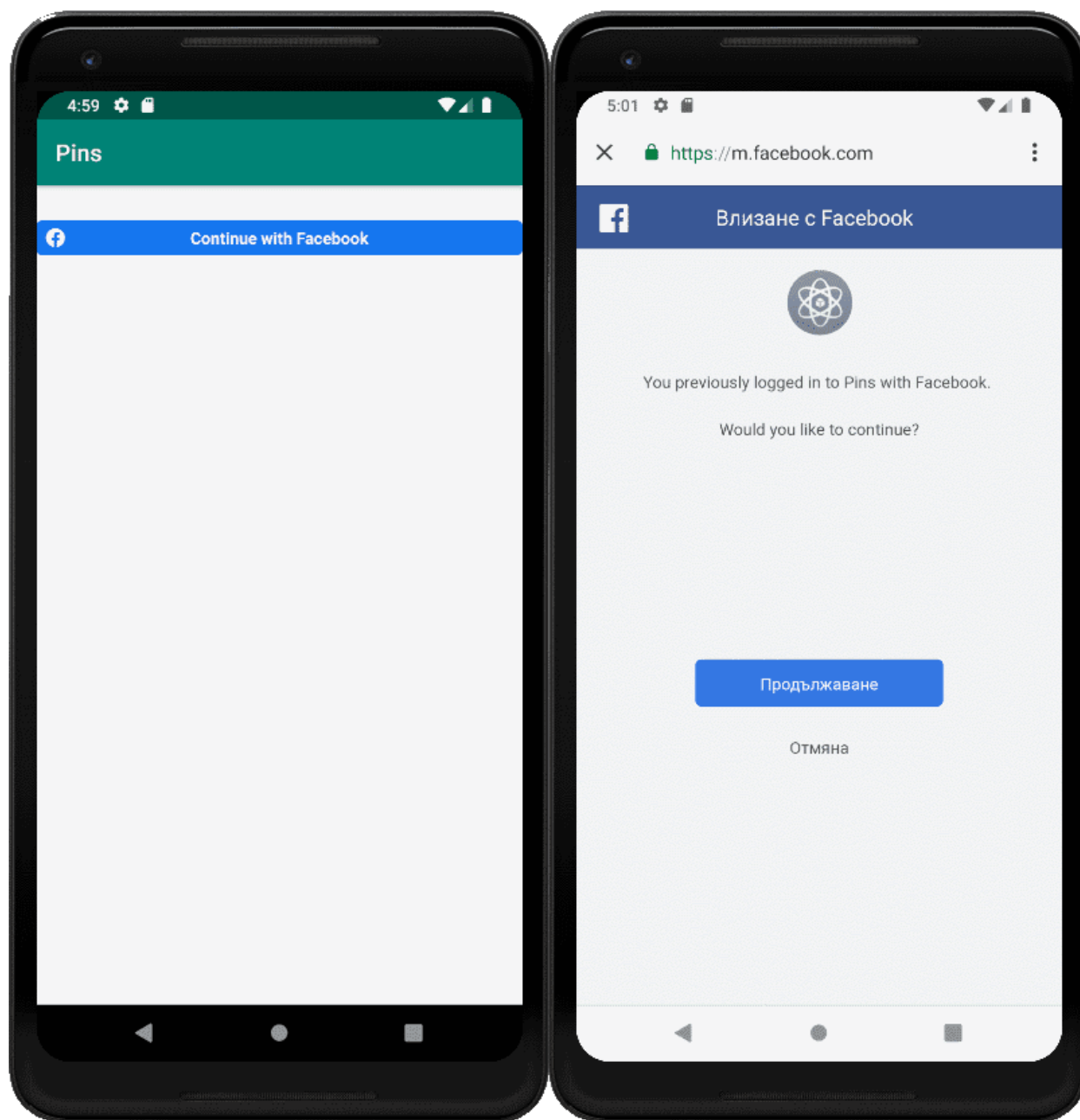
4.1 Изисквания

- Android устройството трябва да е с минимална версия 8.0 Lollipop (Api level 26).
- Потребителят трябва да разреши приложението да използва интернет, локация и камерата на устройството

4.2 Инструкции за ползване

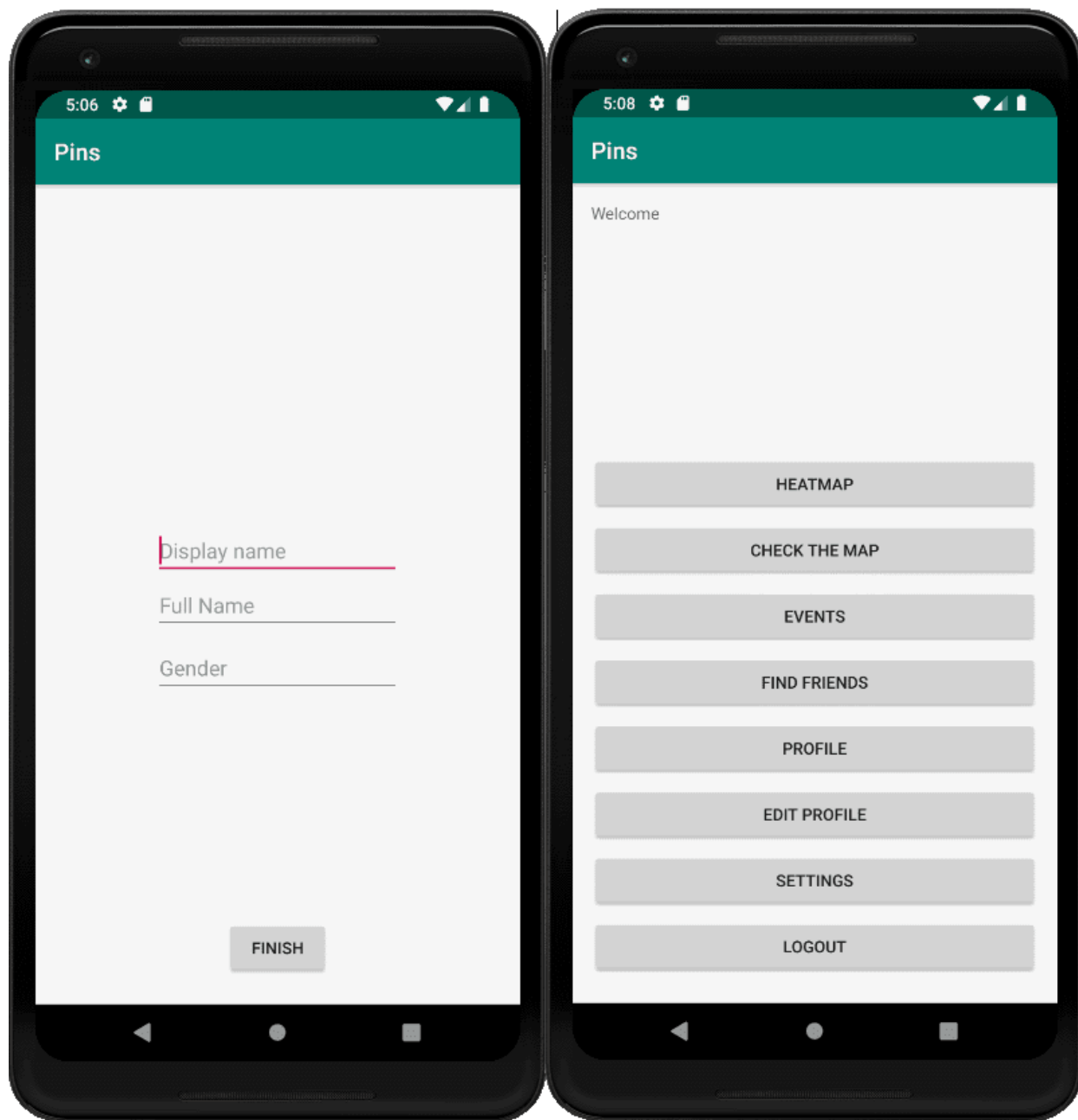
4.2.1 Регистрация

Регистрацията в приложението става посредством Facebook аутентикация. С отварянето на приложението потребителя ще види бутон “Continue with facebook”, който ще го отведе до мобилната версия на Facebook за да потвърди аутентикацията.



Фигура 4.1 Facebook login

След това новите потребители биват пренасочени към страница за редактиране на профила, където да въведат своите данни, а след това отиват на главната навигационна страница.



Фигура 4.2 Вписване на информация за потребителя & Началната страница

4.2.2 Навигация из приложението

От тук потребителя вече може да ползва приложението свободно.

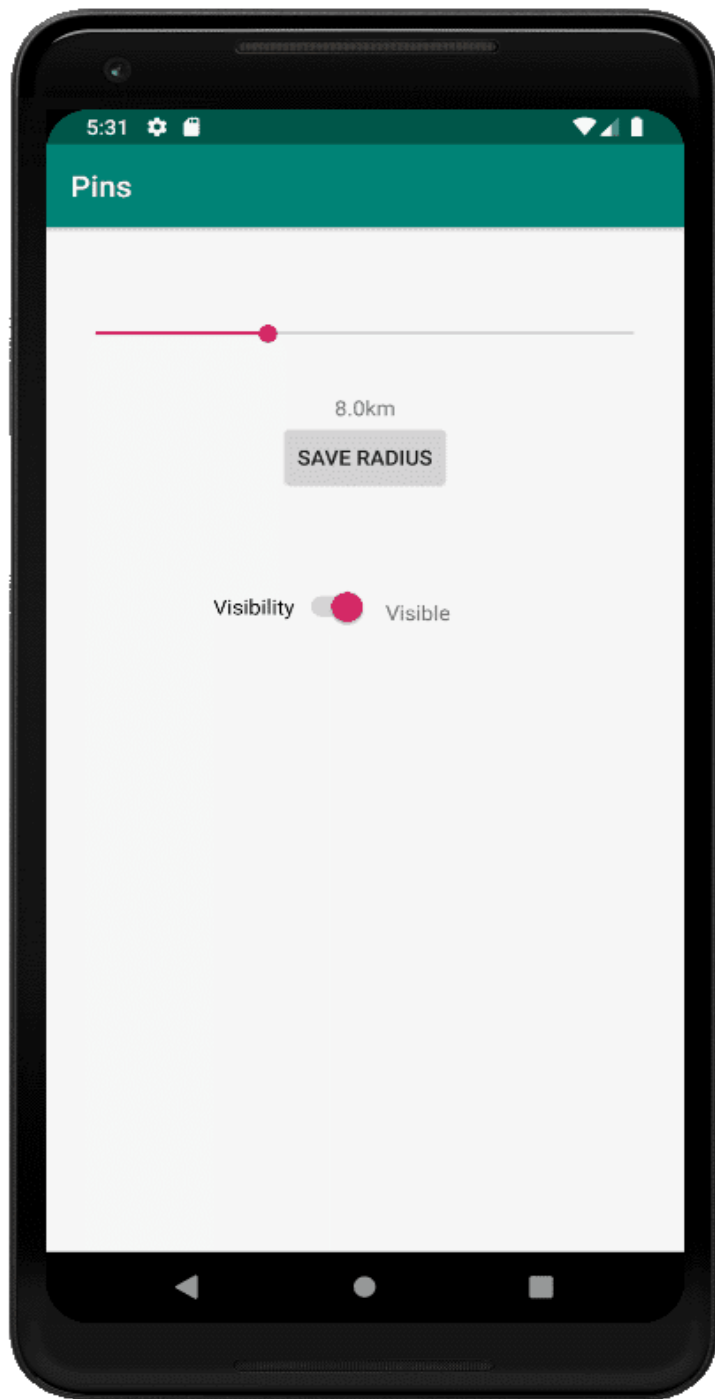
Бутонът за Heatmap го отвежда на страница с карта, на която са отбелязани най-популярните места за срещи с хора. Бутонът “Find friends”

води до страница, от която потребителя може да разглежда профили, да им праща покани за приятелство и блокира. От страницата на бутона “Settings” потребителя избира дали да бъде откриван на карта с приятели и задава радиуса, в който той може да открива хора. Бутонът “Profile” води до личния профил на потребителя, а “Edit Profile” до страницата за редакция на профила. До страницата за търсене на събития се води бутона “Events”. Бутонът “Check the map” отвежда потребителя на страница с карта, която показва всички негови приятели в зададен от него радиус.

Ако потребителя иска да се отпише от приложението, трябва да натисне бутона “Logout”.

4.2.3 Настройки

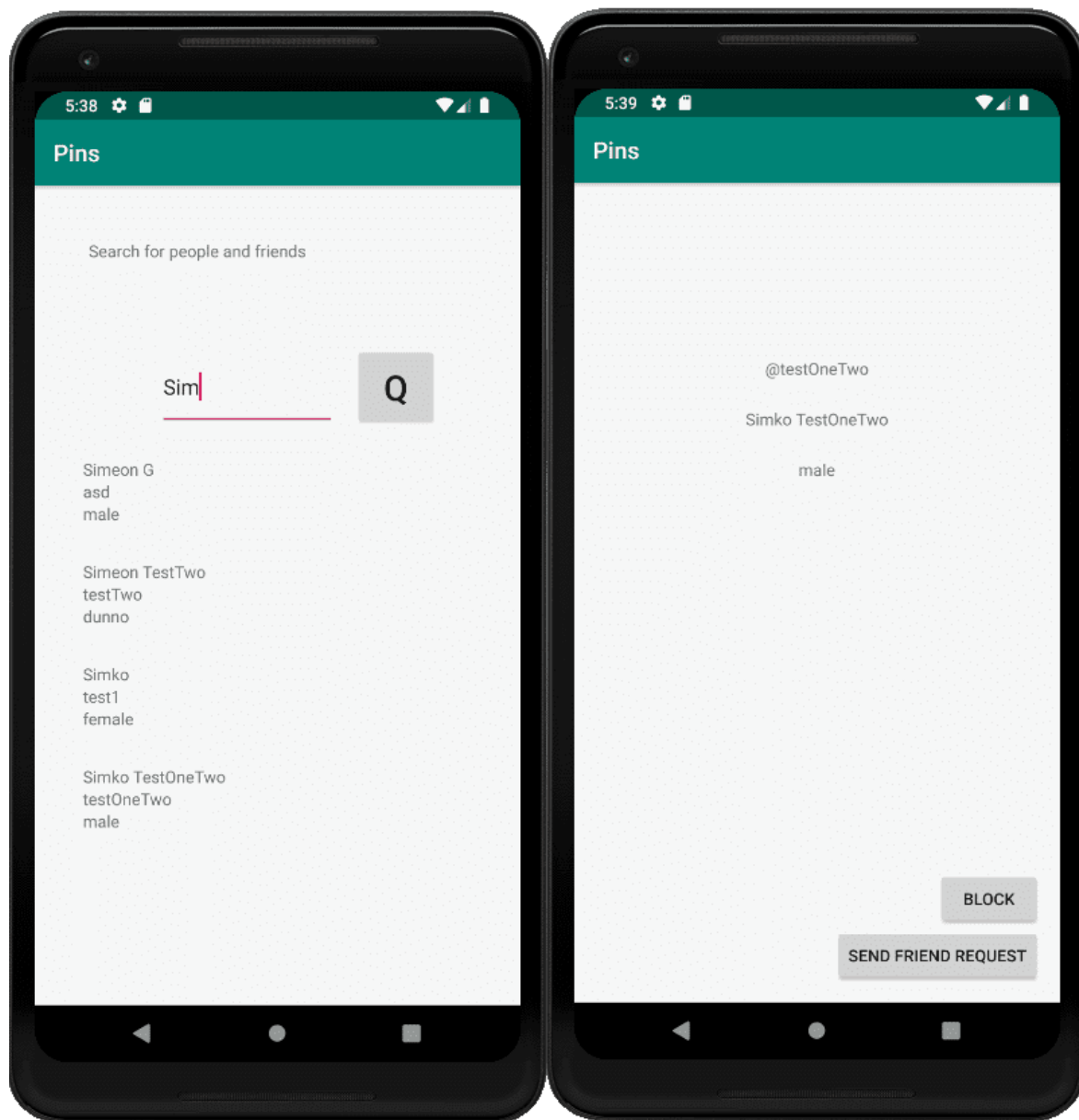
В страницата за настройки потребителя задава желания радиус чрез плъзгач и го запазва с бутона “Save Radius”. Освен това, потребителя избира дали да е видим за приятелите си с помощта на превключвател.



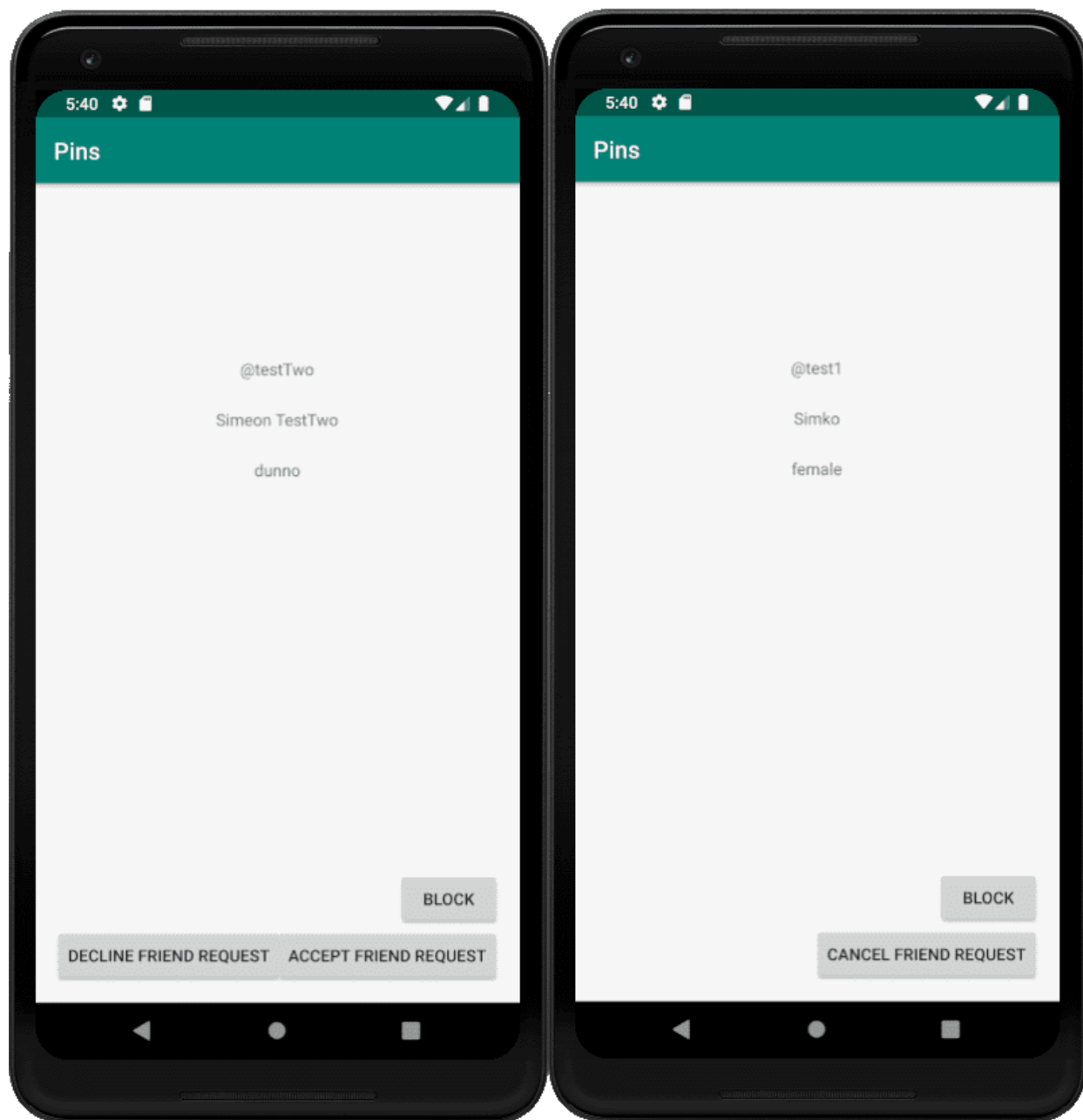
Фигура 4.3 Страницата за настройки

4.2.4 Намиране на приятели

В страницата за разглеждане на профили потребителите въвеждат името на човека, който търсят и получават списък от профили със сходни имена. След това с кликане върху потребител от списъка, потребителя бива отведен до профилната му страница, къде може да го блокира, да изпрати покана за приятелство, да приеме покана, ако е получил такава, да се откаже от приятелството и други.



Фигура 4.4 Търсачка за потребители & Профил на потребител

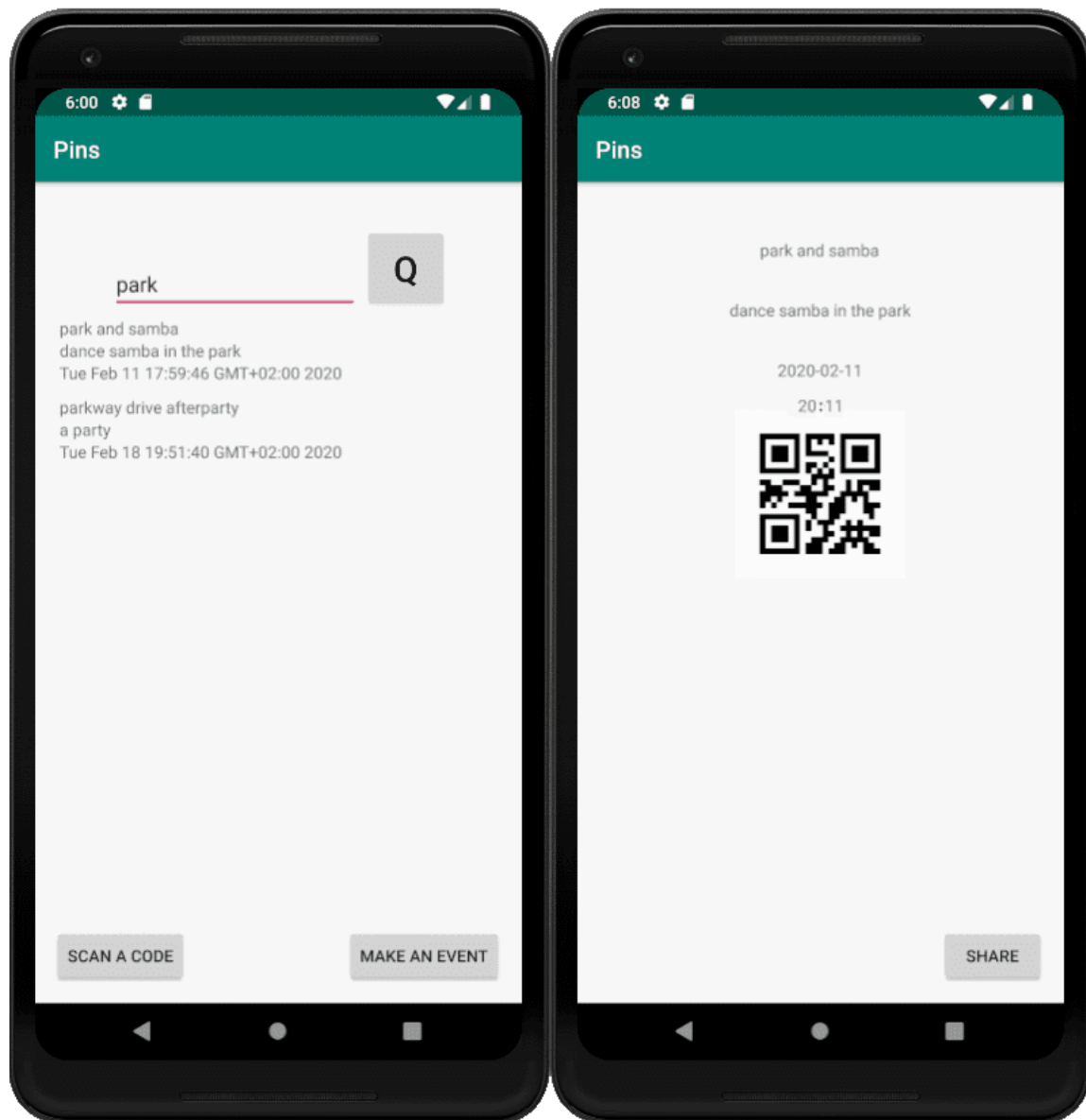


Фигура 4.5 Получена покана за приятелство & Изпратена покана

4.2.5 Събития

В страницата за събития потребителя може да търси събития по име, които са започнали до 2 дни преди търсенето, както и такива, които още не са започнали. Потребителя може да създава, както и да се чекира в събития.

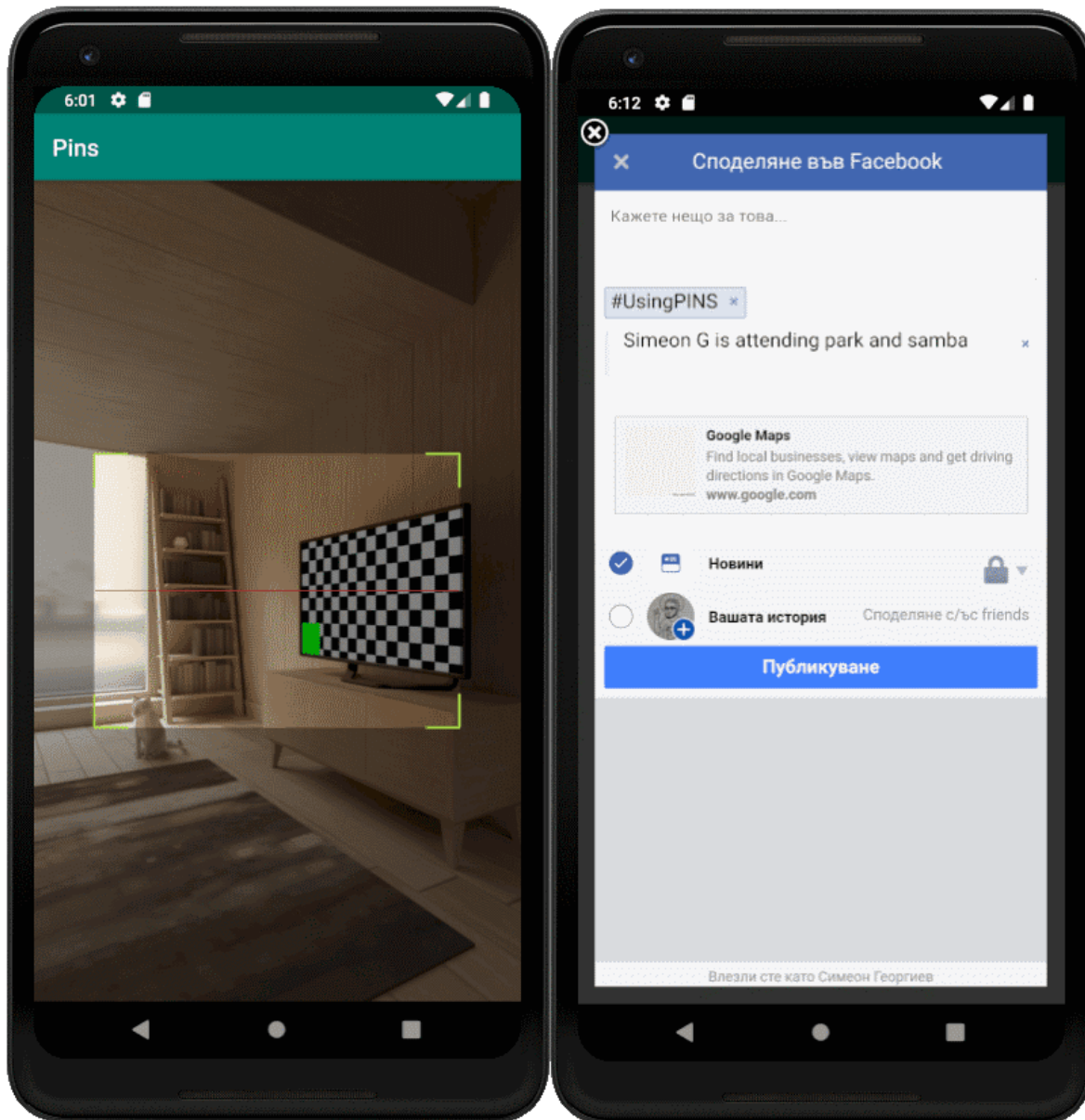
Бутона за създаване отвежда потребителя в страница, където той избира дата и часа на събитието, последвано то страница за име и описание. Ако потребителя е създател на дадено събитие, то когато отвори профила му, ще има QR код, който трябва да бъде сканиран от другите потребители, за да се чекират.



Фигура 4.6 Търсачка на събития & Профил на събитие

Чекирането става натискането на бутона “Scan a code”. Той включва камерата и потребителя трябва да намести QR кода от телефона на създателя на събитието в обозначената зона за да се запише като присъстващ.

След като вече присъства на събитието, потребителя може да натисне бутона “Share” и да сподели това на стената си във Facebook.

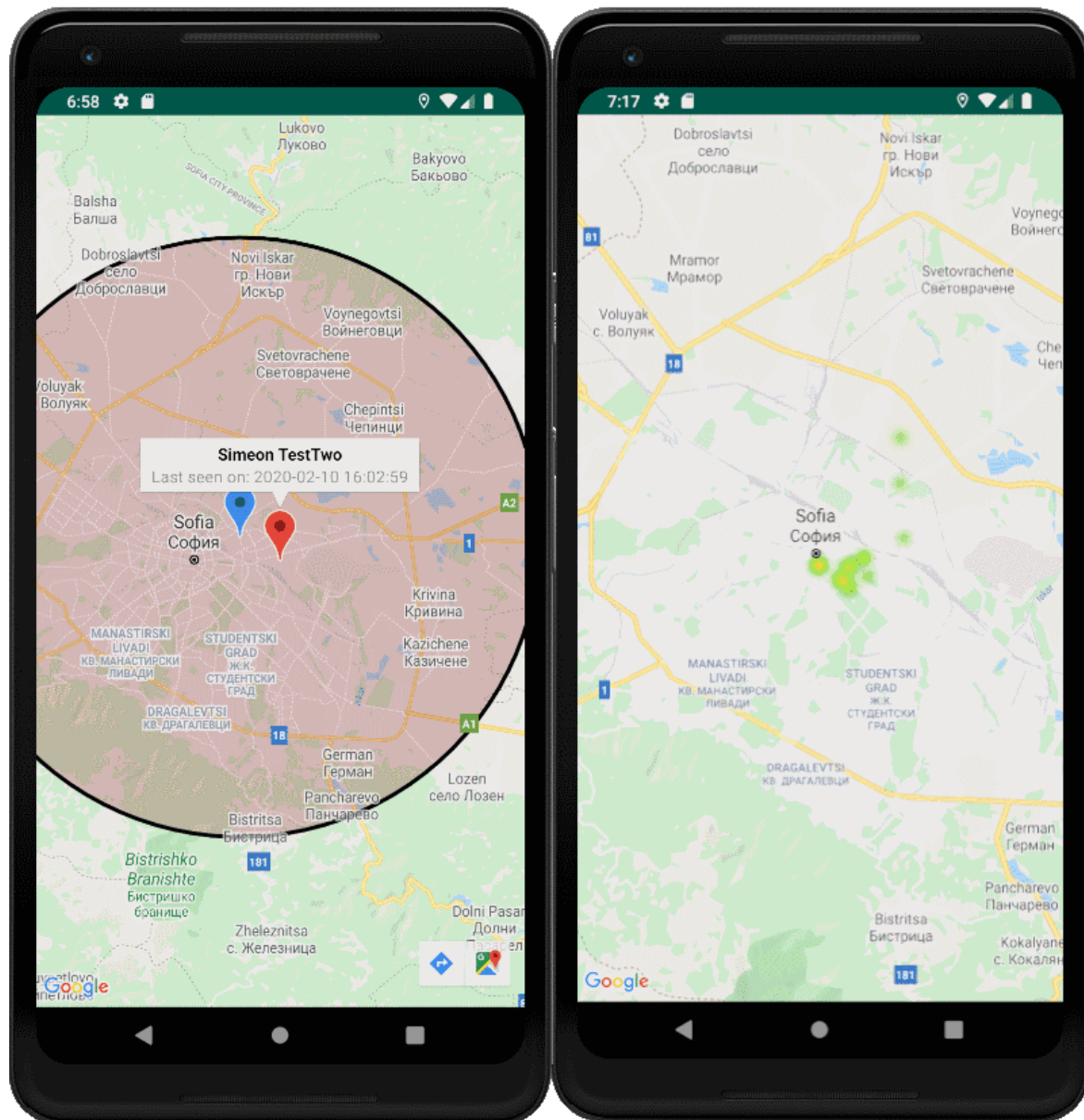


Фигура 4.7 Скенер на QR кодове & Споделяне във Facebook

4.2.6 Карти

4.2.6.1 Карта с приятели

На тази карта са показани локациите на всички откриваеми приятели, който са в избора от потребителя радиус. Изобразени са чрез червени маркери на картата, когато бъдат натиснати се показва повече информация за дадения потребител.



Фигура 4.8 Карта с приятели в близост & Heatmap

4.2.6.2 Heatmap

“Heatmap” предоставя информация за най-често посещаваните места, като по този начин потребителите могат да преценят къде е приятно място за среща с приятел.

Заклучение

Този проект е първия ми досег с Android света. Считам, че успях да се справя с поставеното задание и научих много за разработката на Android приложения. Изискванията в по-голямата си част са спазени. Приложението работи и може бъде използвано.

Бъдещото развитие на проекта се състои в добавянето на още функционалности, както и подобряване на вече имплементираните.

Исползвана литература

<https://developers.google.com/maps/documentation>

<https://developer.android.com/docs>

<https://firebase.google.com/docs/android/setup>

<https://developers.facebook.com/docs/android/getting-started/>

<https://github.com/zxing/zxing>

David Griffiths & Dawn Griffiths, Head First Android Development, 2015, pp.
2-162, 659-663

[https://stackoverflow.com/questions/30779596/best-to-use-android-studio-or-intelli
j-idea-wth-plugins/30784212#30784212](https://stackoverflow.com/questions/30779596/best-to-use-android-studio-or-intelli-j-idea-wth-plugins/30784212#30784212)

Съдържание

Увод	3
Първа Глава	4
1.1 Съществуващи приложения	4
1.1.1 Facebook nearby friends	4
1.1.2 Snapchat Map	5
1.2 Технологии, среди за развой и бази данни	6
1.2.1 IntelliJ IDEA	6
1.2.2 Android Studio	6
1.2.3 Java	7
1.2.4 Kotlin	7
1.2.5 XML	8
1.2.6 Бази данни	8
1.2.7 Основни компоненти на Android приложението и Android SDK	10
Втора Глава	12
2.1 Функционални изисквания към проекта	12
2.2 Избор на програмен език и библиотеки	12
2.2.1 IDE - Android Studio	12
2.2.2 Език за програмиране - Java	13
2.2.3 Firebase	13
2.2.4 Facebook SDK	14
2.2.5 ZXing	14
2.3 Проектиране	14
2.3.1 Структура на приложението	14
2.3.2 Работа с Firebase Database	16
2.3.2.1 Описание на колекцията Users	16
2.3.2.2 Описание на колекцията Friendlists	17
2.3.2.3 Описание на колекцията Friend_Requests	18
2.3.2.4 Описание на колекцията Blocked	19
2.3.2.5 Описание на колекцията Locations	19
2.3.2.6 Описание на колекцията Radiuses	21
2.3.2.7 Описание на колекцията Events	21
2.3.2.8 Описание на колекцията Scores	23
Трета Глава	25
3.1 Аутентикация	25

3.2 Начална страница	26
3.3 Find Friends	28
3.4 Профили на потребители	31
3.4.1 Вземане на информация за потребителя	31
3.4.2 MaintainUI();	32
3.4.3 Покана за приятелство	35
3.5 Карта	41
3.5.1 Google Maps SDK	41
3.5.2 MapsActivity Setup	41
3.5.3 Представяне на информация върху картата	43
3.5.4 Heatmap	45
3.6 Събития	46
3.6.1 Основно activity	46
3.6.2 Създаване на събитие	49
3.6.3 Профил на събитието	50
3.6.4 Чекиране на събитие	52
3.7 Настройки	54
Четвърта Глава	57
4.1 Изисквания	57
4.2 Инструкции за ползване	57
4.2.1 Регистрация	57
4.2.2 Навигация из приложението	59
4.2.3 Настройки	60
4.2.4 Намиране на приятели	62
4.2.5 Събития	64
4.2.6 Карты	67
4.2.6.1 Карта с приятели	67
4.2.6.2 Heatmap	68
Заклучение	69
Използвана литература	70
Съдържание	71