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. Утвърждавам:.....
Дата на предаване: 15.02.2020 г. /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 срещи 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 хората имат свободно време, което не могат да оползотворят правилно.

Обикновено срещите с приятели са от най-разведряващите събития, с които да се запълват празните дупки в графика. Разбира се, разходките в парка, ходенето на кино или заведение са все дейности, изискващи предварителна уговорка за място и час. Случайните срещи, колкото и приятни, са редки особено на подобни места забавление.

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 и така се отбелязват в него. 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.

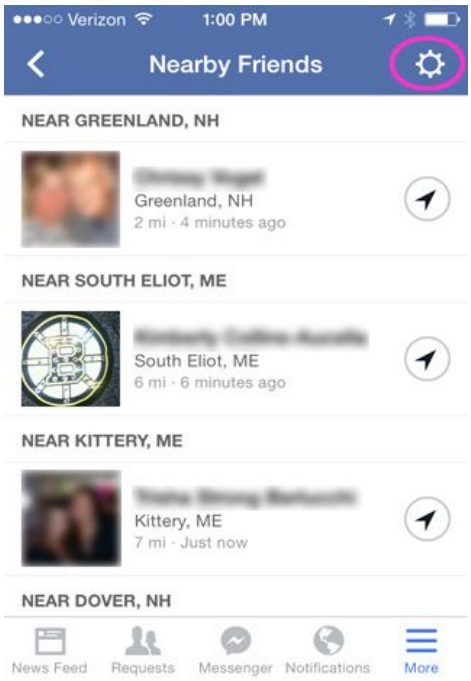
Първа Глава

Преглед на използвани методи и технологии. Проучване на подобни приложения

1.1 Съществуващи приложения

1.1.1 Facebook nearby friends

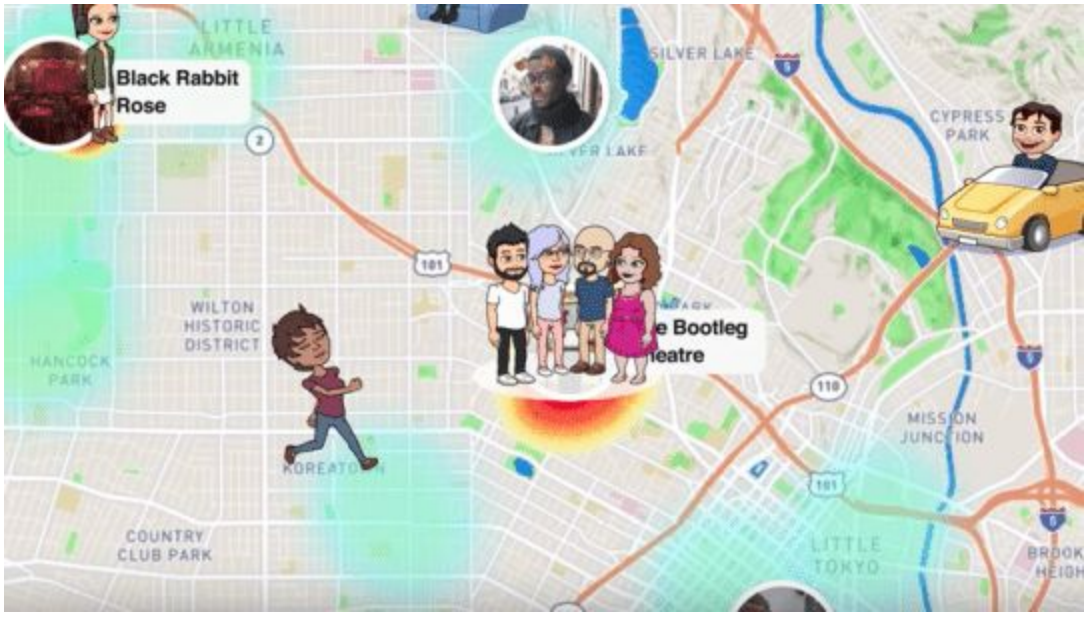
Facebook предоставя възможността да се види релативното местоположение на ваши приятели, които са включили локацията на устройствата си. Има browser версия, както и мобилна за Android и iOS. За съжаление, Facebook предпочитат да развиват другите функционалности по платформата си и не работят по Nearby Friends. За това и е ограничено само до гореспоменатата функционалност. Изглед от приложението представен на *Фигура 1.1.*



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 топлинна карта (Heatmap) на публичните истории. Примерен изглед на картата от Snapchat може да се открия на *фигура 1.2*.

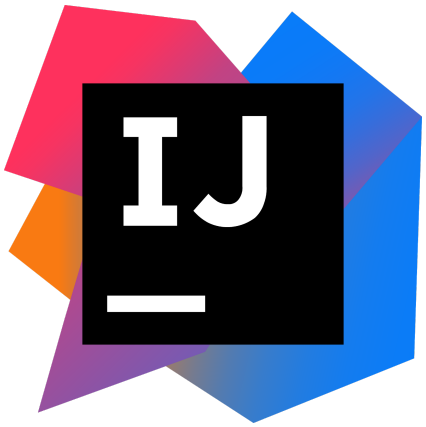


Фигура 1.2 SnapMap

1.2 Технологии, среди за развой и бази данни

1.2.1 IntelliJ IDEA

IntelliJ IDEA е продукт на JetBrains. Представлява 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. [7]



Фигура 1.3 Логото на IntelliJ

1.2.2 Android Studio

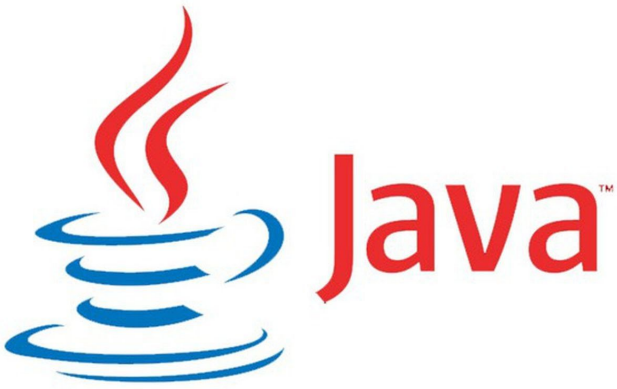
Android Studio е изградено върху Intellij IDEA. То е официалното IDE за Android, проектирано специално за лесната разработка на Android приложения. Разполага с емулатор, с който програмиста може да тества приложението си на различни мобилни устройства, без да се налага да ги притежава физически. Android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

потребителският интерфейс на приложението (дефиниран чрез XML layout файлове), което позволява на разработчика лесно да променя разположението и характеристиките на елементите на приложението. [2], [6], [7]

Фигура 1.4 Логото на 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 извършват 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 Логото на 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 Логото на 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 метаезик - 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), за дефиниране на размери, отстояния, цветове, на текстовете показани в потребителският интерфейс и много други.

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.

Предимство на тези бази данни е, че при добра структура, има малко излишна или повтаряща се информация. Освен това, в релационните бази данни предоставят множество оптимизации, които ги правят изключително бързи и надеждни.

Основните проблеми на SQL базираните бази са скалирането и мигрирането на данни. Ако програмистът често променя структурата на базата, релационните имплементации са непрактични.

В Android света най-популярна релационна база данни е **SQLite.**

NoSQL базираните бази съхраняват информацията под различни структурати: 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. Идея 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. Най-използваните NoSQL бази данни за Android са: **Firebase Realtime Database** и **Realm database**. [8]

1.2.7 Основни компоненти на Android приложението и Android SDK

Android SDK (Software development kit) компилира и пакетира 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 начи 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 na 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 компоненти, до които приложението ще поиска достъп от операционната система и потребителят. Такива са например : достъп до камера, интернет, локация, списък с контакти и други.
- Всички хардуерни свойства - bluetooth услуги, камера, multitouch и други.
- Минималното ниво на Android API - минималната версия на операционната система, за която приложението е предвидено да работи.
- Компоненти от някои библиотеки, различни от Android framework APIs, като например Google Maps библиотеката.

[2], [6]

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 е препоръчваната от Google среда за разработване на Android приложения. Това го прави очевиден избор за целите на дипломната работа. Средата разполага с графичен редактор за .xml файлове и потребителски интерфейс, възможност да се създават директни инсталационни файлове за Android (.apk формат), вградени виртуални емулатори с многобройни дистрибуции и версии на Android [7]

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 множество библиотеки и документация, която би била полезна в процеса на разработка. За разлика от Java, Kotlin, който е изключително мощен и разполага с редица предимства (като null-safe полета), е сравнително нов и документацията е малко и трудно достъпна. Основна причина, да избера Java, е достъпността на източници, както и това, че се преподава в ТУЕС.

2.2.3 Firebase

Firebase е платформа на Google за подпомагане на разработката на мобилни и уеб приложения. Предоставя функционалности като автентикация, аналитика, бази данни и crash reports. Всички услуги, предлагани от Firebase, скалират заедно с потребителския трафик към приложението, без да е необходима намеса от страна на програмиста. Едно от другите големи предимства на Firebase е наличието на безплатен начален план, който е достатъчен за старта на всяко едно приложение, докато събере голям брой потребители. 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 всяка промяна по данните в базата. 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

изпраща 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. [3]

2.2.4 Facebook SDK

Facebook SDK предоставя редица функционалности като автентикация, както и възможността да бъде извлечена различна информация за потребителя - като име, email, списък с приятели и тн. Също така, чрез него могат да се пишат постове на Facebook стената на потребителя. Автентикацията може да бъде обвързана с Firebase. Firebase Authentication позловляза използването на Facebook акаунти за по-лесни регистрация и ползване на приложението. [4]

2.2.5 ZXing

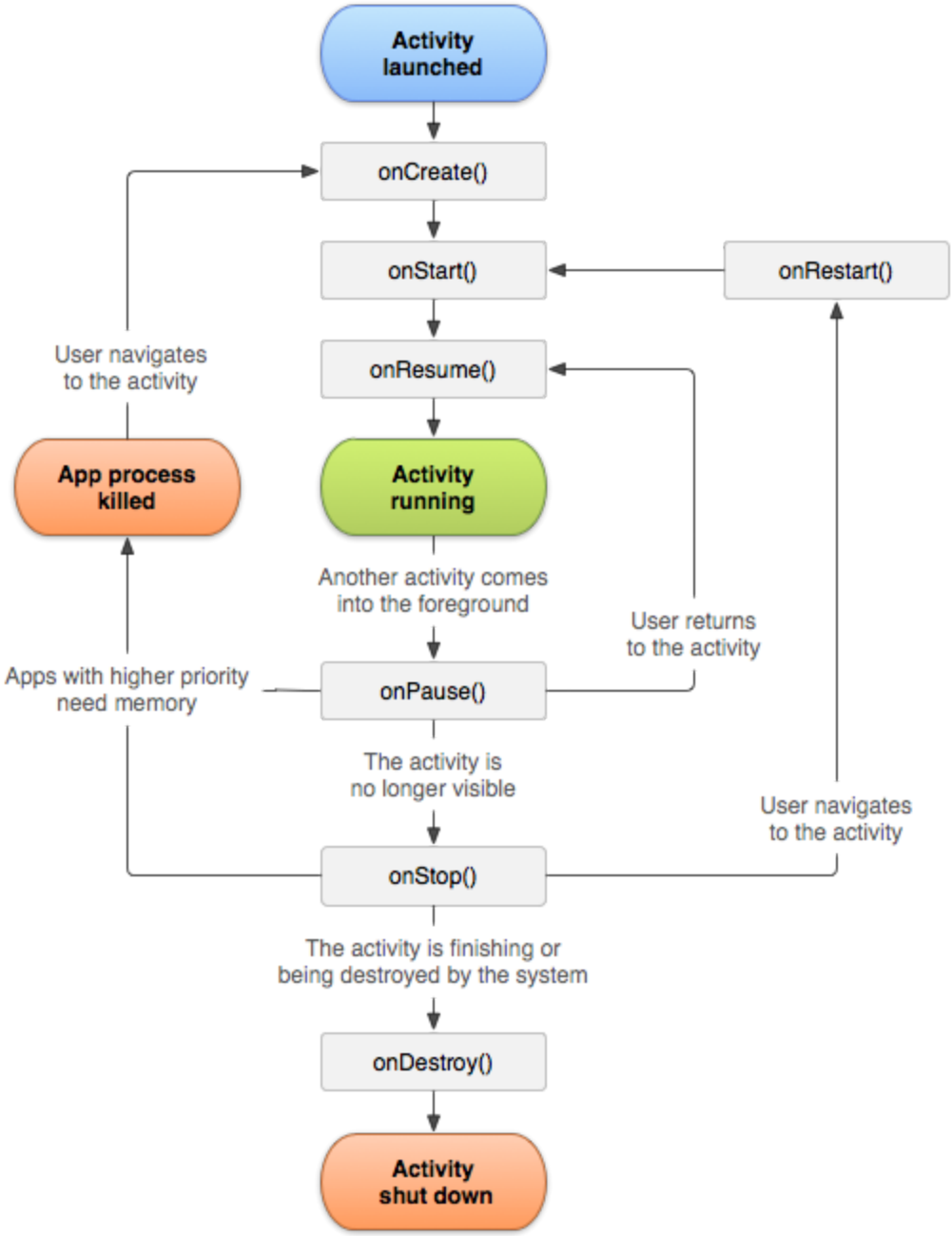
ZXing или Zebra Crossing е open-source библиотека за обработка на баркодове. Имплементирана е на Java, но има версии за C++, JavaScript, Python, PHP и C#. Поддържа 1D баркодове като UPC-A, EAN-13 и Codebar, както и 2D баркодове като QR code, Data Matrix и Aztec. Библиотеката се разработва и поддържа от Google. Те я използват като основа за баркод скенера на Android и е интегрирана в техни продукти, като например Google Book Search. [5]

2.3 Проектиране

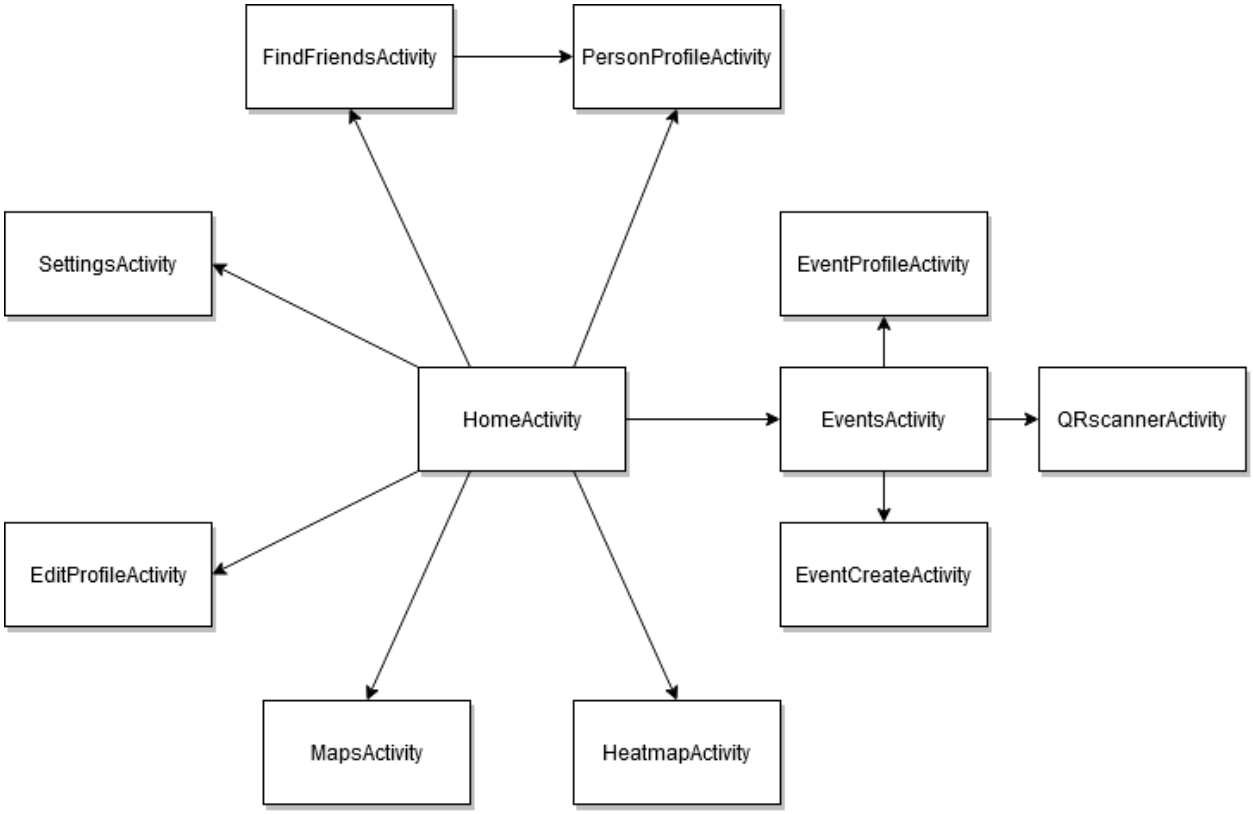
2.3.1 Структура на приложението

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, представени във *Фигура 2.1*. [2], [6]

Приложението ще представлява няколко Activities с различна функционалност, представени на *Фигура 2.2*.



Фигура 2.1 Жизненият цикъл на едно Activity



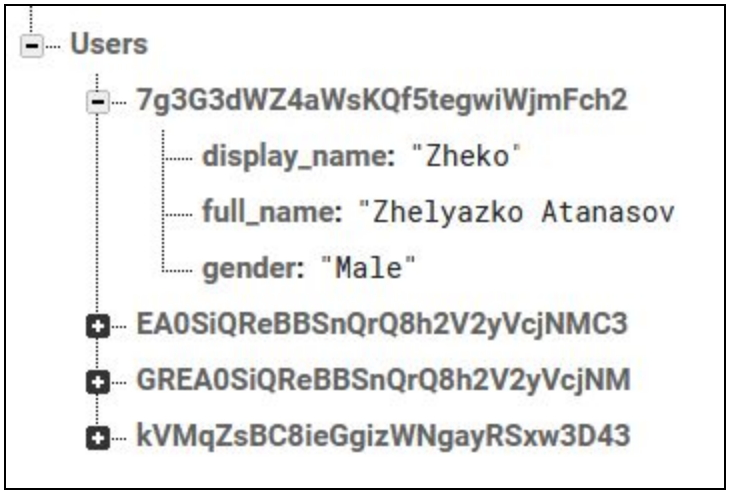
Фигура 2.2 Структура на приложението

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. [3]

2.3.2.1 Описание на колекцията Users

Структурата и примерни данни за Users може да се види на *Фигура 2.3*.



Фигура 2.3 Колекцията Users

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

display_name - потребителско име на потребителя в низов тип

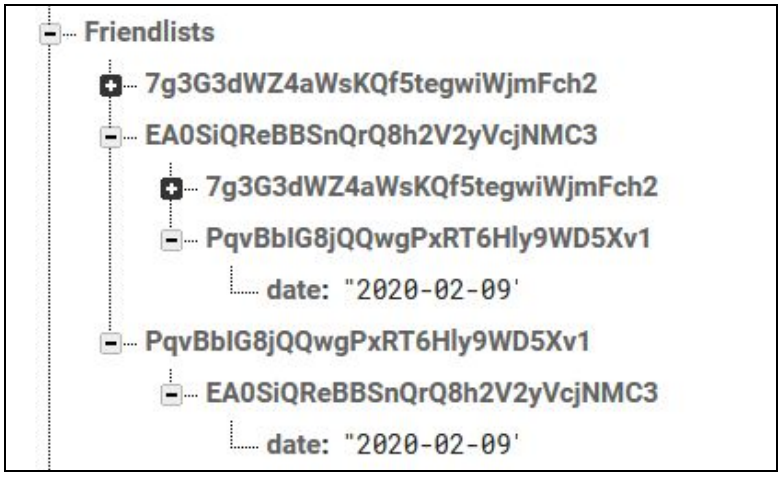
full_name - пълно име на потребителя в низов тип

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 автентикацията 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 *Фигура 2.4*.

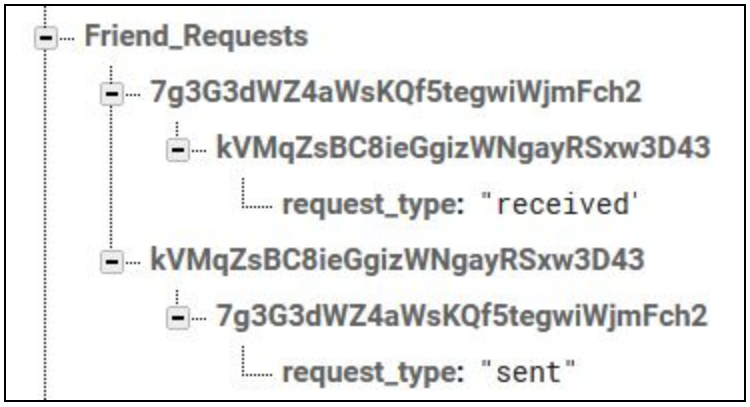


Фигура 2.4 Колекцията Friendlists

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 yyyy-MM-dd date формат.

2.3.2.3 Описание на колекцията Friend_Requests

Структурата и примерни данни за Friend_Requests може да се види на *Фигура 2.5*



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

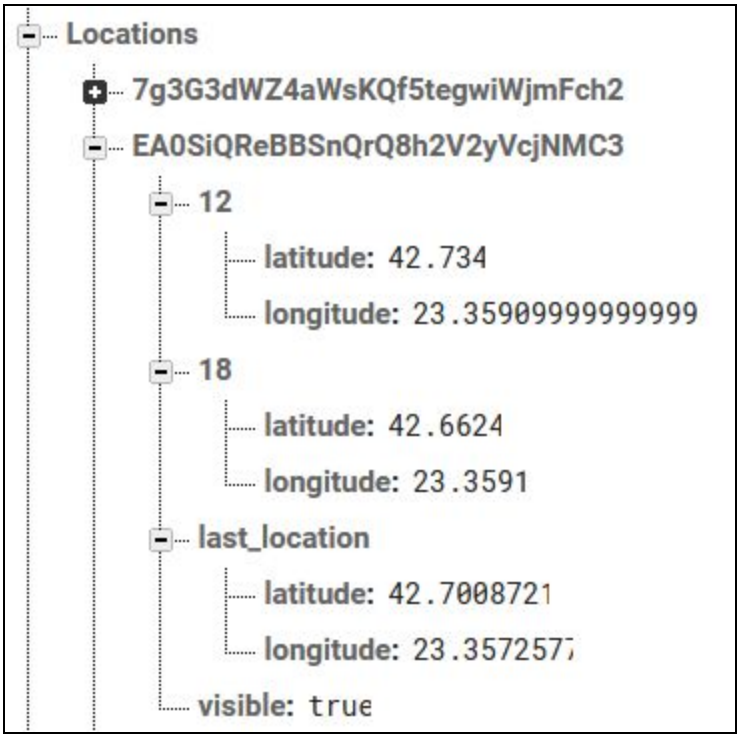
Структурата и примерни данни за Blocked може да се види на *Фигура 2.6*

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 *Фигура 2.7*.



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 *Фигура 2.8.*

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 *фигура 2.9*

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 *фигура 2.10*

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 Автентикация

Firebase Authentication предоставя разнообразни SDKs (Software development kits) и готови библиотеки за потребителски интерфейс, които да автентекират 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 трябва да се инициализират login бутон и CallbackManager по начина, показан във *фигура 3.1*.

```
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              }
```

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 автентикира 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 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 закъчен OnClickListener, чийто OnClick метод е пренаписан да стартира и пренесе потребителя в SettingsActivity.

- homePageLogoutBtn има закъчен OnClickListener, чийто OnClick метод е пренаписан да излезе от профила и да извика метода updateUI() (*фигура 3.3*). Той показва Toast съобщение, уведомяващо потребителя, че е излязъл от профила си и стартира и пренася потребителя в MainActivity - т.е. Login страницата. Организацията на бутоните е показана на *фигура 3.2*

```
45    logoutBtn.setOnClickListener((view) → {
48            mAuth.signOut(); //this will logout from Firebase
49            LoginManager.getInstance().logOut(); //this will logout from Facebook
50            updateUI();
51
52    });
54    openProfile.setOnClickListener((view) → {
57            Intent profilePage = new Intent( packageContext: HomeActivity.this, PersonProfileActivity.class);
58            profilePage.putExtra( name: "visit_user_id", mAuth.getCurrentUser().getUid());
59            startActivity(profilePage);
60    });
62    findFriends.setOnClickListener((view) → {
65            Intent findFriendsPage = new Intent( packageContext: HomeActivity.this, FindFriendsActivity.class);
66            startActivity(findFriendsPage);
67    });
69    editProfile.setOnClickListener((view) → {
72            Intent editProfile = new Intent( packageContext: HomeActivity.this, EditProfileActivity.class);
73            startActivity(editProfile);
74    });
76    checkMap.setOnClickListener((view) → {
79            Intent maps = new Intent( packageContext: HomeActivity.this, MapsActivity.class);
80            startActivity(maps);
81    });
83    events.setOnClickListener((view) → {
86            Intent eventsPage = new Intent( packageContext: HomeActivity.this, EventsActivity.class);
87            startActivity(eventsPage);
88    });
90    settings.setOnClickListener((view) → {
93            Intent settingsPage = new Intent( packageContext: HomeActivity.this, SettingsActivity.class);
94            startActivity(settingsPage);
95    });
97    heatmap.setOnClickListener((view) → {
100           Intent settingsPage = new Intent( packageContext: HomeActivity.this, HeatmapActivity.class);
101           startActivity(settingsPage);
102   });
```

Фигура 3.2 Навигация

Методът onStart() (*фигура 3.3)* 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         FirebaseUser currentUser = mAuth.getCurrentUser();
110         if (currentUser == null){
111             updateUI();
112         }
113         else {
114             dbRef.addListenerForSingleValueEvent(new ValueEventListener() {
115                 @Override
116                 public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
117                     if (!dataSnapshot.child("Users").hasChild(mAuth.getCurrentUser().getUid())) {
118                         Intent setUser = new Intent(HomeActivity.this, EditProfileActivity.class);
119                         startActivity(setUser);
120                     }
121                 }
122 
123                 @Override
124                 public void onCancelled(@NonNull DatabaseError databaseError) {
125 
126                 }
127             });
128         }
129     }
130 
131     private void updateUI(){
132         Toast.makeText(HomeActivity.this, "You are logged out.",  Toast.LENGTH_SHORT).show();
133 
134         Intent login = new Intent(HomeActivity.this, MainActivity.class);
135         startActivity(login);
136         finish();
```

Фигура 3.3 Методите onStart и 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. Recycler View 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(), представен на *фигура 3.4*

```
59      private void searchPeople(String input) {
60          Query searchPeopleQuery = allUsersRef.orderByChild("full_name").startAt(input).endAt(input + "\uf8ff");
61
62          FirebaseRecyclerAdapter<User, FindFriendsViewHolder> firebaseRecyclerAdapter = new FirebaseRecyclerAdapter<User, FindFriend
63                  (
64                      User.class, R.layout.all_users_display_layout, FindFriendsViewHolder.class, searchPeopleQuery
65                  ) {
66              @Override
67              protected void populateViewHolder(FindFriendsViewHolder viewHolder, User user, final int position) {
68                  viewHolder.displayUser(user);
69
70                  viewHolder.mView.setOnClickListener(new View.OnClickListener() {
71                      @Override
72                      public void onClick(View view) {
73                          String visit_user_id = getRef(position).getKey();
74
75                          Intent profileIntent = new Intent(FindFriendsActivity.this, PersonProfileActivity.class);
76                          profileIntent.putExtra("visit_user_id", visit_user_id);
77                          startActivity(profileIntent);
78                      }
79                  });
80              }
81          };
82          friendsList.setAdapter(firebaseRecyclerAdapter);
```

Фигура 3.4 Методът searchPeople()

Резултатите от търсенето са изложени в RecyclerView. За целта трябва ViewHolder клас, направен специално за потребителите и техните полета и показана на *фигура 3.5*. 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         void displayUser(User user) {
101             fullName.setText(user.full_name);
102             displayName.setText(user.display_name);
103             gender.setText(user.gender);
104         }
105     }
106 }
```

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 (*фигура 3.6*) и ViewHolder (*фигура 3.5*). 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 потребител, с цел да се зареди профила му.

```java
public class User {
    public String uid;
    public String display_name;
    public String full_name;
    public String gender;



    public User() {

    }

    public User(String uid, String username, String fullName, String gender) {
        this.uid = uid;
        this.full_name = fullName;
        this.display_name = username;
        this.gender = gender;

    }

    @Exclude
    public Map<String, Object> toMap() {
        HashMap<String, Object> result = new HashMap<>();

        result.put("display_name", display_name);
        result.put("full_name", full_name);
        result.put("gender", gender);

        return result;
    }
}
```

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 записа, 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(), показан на *фигури 3.7 и 3.8*, и един Enumerator - 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 записа на текущия потребител има запис за потребителя, чийто профил бива разглеждан. Ако има, се запазва стойността на “request_type” записа в символен низ - request_type.

Ако request_type е == “sent”, friendshipStatus (инстанция на FriendshipStatus enumerator-а) 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”.

```
267     private void maintainUI() {
268         friendRequestRef.child(senderUserID).addListenerForSingleValueEvent(new ValueEventListener() {
269             @Override
270             public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
271                 if(dataSnapshot.hasChild(receiverUserID)){
272                     String request_type = dataSnapshot.child(receiverUserID).child("request_type").getValue().toString();
273
274                     if(request_type.equals("sent")){
275                         friendshipStatus = FriendshipStatus.REQUEST_SENT;
276                         sendBtn.setText("Cancel Friend Request");
277
278                         declineBtn.setVisibility(View.INVISIBLE);
279                         declineBtn.setEnabled(false);
280                     }
281                     else if(request_type.equals("received")){
282                         friendshipStatus = FriendshipStatus.REQUEST_RECEIVED;
283                         sendBtn.setText("Accept Friend Request");
284
285                         declineBtn.setVisibility(View.VISIBLE);
286                         declineBtn.setEnabled(true);
287
288                         declineBtn.setOnClickListener((view) → {
291                                 cancelFriendRequest();
292                         });
294                     }
295                 }
296                 else {
297                     friendsRef.child(senderUserID).addListenerForSingleValueEvent(new ValueEventListener() {
298                         @Override
299                         public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
300                             if(dataSnapshot.hasChild(receiverUserID)){
301                                 friendshipStatus = FriendshipStatus.FRIENDS;
302                                 sendBtn.setText("Unfriend");
303                                 declineBtn.setVisibility(View.INVISIBLE);
304                                 declineBtn.setEnabled(false);
305                             }
306                         }
```

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 деактвиран и невидим, а бутона за отблокиране - обратното. Ако няма, бутона за отблокиране става деакиивиран и невидим, а бутона за блокиране - обратното.

```
321        blockedRef.child(senderUserID).addListenerForSingleValueEvent(new ValueEventListener() {
322            @Override
323            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
324                if(dataSnapshot.hasChild(receiverUserID)){
325                    blockBtn.setVisibility(View.INVISIBLE);
326                    blockBtn.setEnabled(false);
327                    unblockBtn.setVisibility(View.VISIBLE);
328                    unblockBtn.setEnabled(true);
329                }
330                else if(!senderUserID.equals(receiverUserID)){
331                    unblockBtn.setVisibility(View.INVISIBLE);
332                    unblockBtn.setEnabled(false);
333                    blockBtn.setVisibility(View.VISIBLE);
334                    blockBtn.setEnabled(true);
335                }
336            }
```

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. Това е показано на *фигура 3.9*

Стойността й 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 й 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 й 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 й 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();

Фигура 3.9 Функцията на sendBtn, спрямо friendshipStatus

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() (*фигура 3.10*) :

- 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 подсигури, че при невъзможност едното действие да се извърши и другото ще бъде прекратено. По този начин се избягва нежелано поведение.

Фигура 3.10 Методът sendFriendRequest()

Методът cancelFriendRequest() (*фигура 3.11*) изтрива записа и на двамата потребители за другия от колекцията “Friend_Requests” на базата данни. Отново се използва OnCompleteListener за да се подсигури, 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) → {
246                 if(task.isSuccessful()){
247                     friendRequestRef.child(receiverUserID).child(senderUserID)
248                             .removeValue()
249                             .addOnCompleteListener((task) → {
252                             if(task.isSuccessful()){
253                                 sendBtn.setEnabled(true); //true?
254                                 friendshipStatus = FriendshipStatus.NOT_FRIENDS;
255                                 sendBtn.setText("Send Friend Request");
256
257                                 declineBtn.setVisibility(View.INVISIBLE);
258                                 declineBtn.setEnabled(false);
259                             }
260                     });
262                 }
263         });
265     }
```

Фигура 3.11 Методът cancelFriendRequest()

Методът acceptFriendRequest() (*фигура 3.12*) :

- 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 подсигури, че при невъзможност предходното действие да се извърши и текущото няма да може.

```
191    private void acceptFriendRequest() {
192        Calendar date = Calendar.getInstance();
193        SimpleDateFormat currentDate = new SimpleDateFormat( pattern: "yyyy-MM-dd");
194        dateBefriended = currentDate.format(date.getTime());
195
196        friendsRef.child(senderUserID).child(receiverUserID).child("date").setValue(dateBefriended)
197                .addOnCompleteListener(new OnCompleteListener<Void>() {
198                    @Override
199                    public void onComplete(@NonNull Task<Void> task) {
200                        if(task.isSuccessful()){
201                            friendsRef.child(receiverUserID).child(senderUserID).child("date").setValue(dateBefriended)
202                                    .addOnCompleteListener((task) → {
205
206                                        if(task.isSuccessful()){
207                                            friendRequestRef.child(senderUserID).child(receiverUserID)
208                                                    .removeValue()
209                                                    .addOnCompleteListener((task) → {
212                                                        if(task.isSuccessful()){
213                                                            friendRequestRef.child(receiverUserID).child(senderUserID)
214                                                                    .removeValue()
215                                                                    .addOnCompleteListener((task) → {
218                                                                        if(task.isSuccessful()){
219                                                                            sendBtn.setEnabled(true);
220                                                                            friendshipStatus = FriendshipStatus.FRIENDS;
221                                                                            sendBtn.setText("Unfriend");
222
223                                                                            declineBtn.setVisibility(View.INVISIBLE);
224                                                                            declineBtn.setEnabled(false);
225                                                                        }
```

Фигура 3.12 Методът acceptFriendRequest()

Методът Unfriend() (*фигура 3.13*) :

- 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.

```
164    private void unfriend() {
165        friendsRef.child(senderUserID).child(receiverUserID)
166                .removeValue()
167                .addOnCompleteListener((task) → {
170                        if(task.isSuccessful()){
171                            friendsRef.child(receiverUserID).child(senderUserID)
172                                    .removeValue()
173                                    .addOnCompleteListener((task) → {
176                                            if(task.isSuccessful()){
177                                                sendBtn.setEnabled(true);
178                                                friendshipStatus = FriendshipStatus.NOT_FRIENDS;
179                                                sendBtn.setText("Send Friend Request");
180
181                                                declineBtn.setVisibility(View.INVISIBLE);
182                                                declineBtn.setEnabled(false);
183                                            }
184                                    });
186                        }
187                });
```

Фигура 3.13 Методът unfriend()

Методът block() (*фигура 3.14*) 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() (*фигура 3.14*) 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 SKD 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,както е показано на *фигурите 3.15 и 3.16*. [1]

```
3       <meta-data
4           android:name="com.google.android.geo.API_KEY"
5           android:value="AIzaSyDwK4S8erxvKVm0M3Wji[redacted]" />
```

Фигура 3.15 Ключът за Google Maps API

```
10      <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), представено в *фигура 3.17*.

```
71      SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
72              .findFragmentById(R.id.map);
73      mapFragment.getMapAsync( onMapReadyCallback: this);
74
```

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 актуализациите (updates) и

минималната дистанция, с която локацията трябва да се е променила, както и LocationListener (*фигура 3.18*).

```
else if(locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER)){
  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, както е показано във *фигура 3.19*.

```
33          final double latitude = location.getLatitude();
34          final double longitude = location.getLongitude();
35
36          currentLatLng = new LatLng(latitude, longitude);
37          Date date = Calendar.getInstance().getTime();
38          SimpleDateFormat sdf = new SimpleDateFormat( pattern: "yyyy-MM-dd HH:mm:ss");
39          String lastSeenOn = sdf.format(date);
40          locationsRef.child(currentUserID).child("last_seen_on").setValue(lastSeenOn);
41
42          locationsRef.child(currentUserID).child("last_location").child("latitude").setValue(latitude);
43          locationsRef.child(currentUserID).child("last_location").child("longitude").setValue(longitude)
```

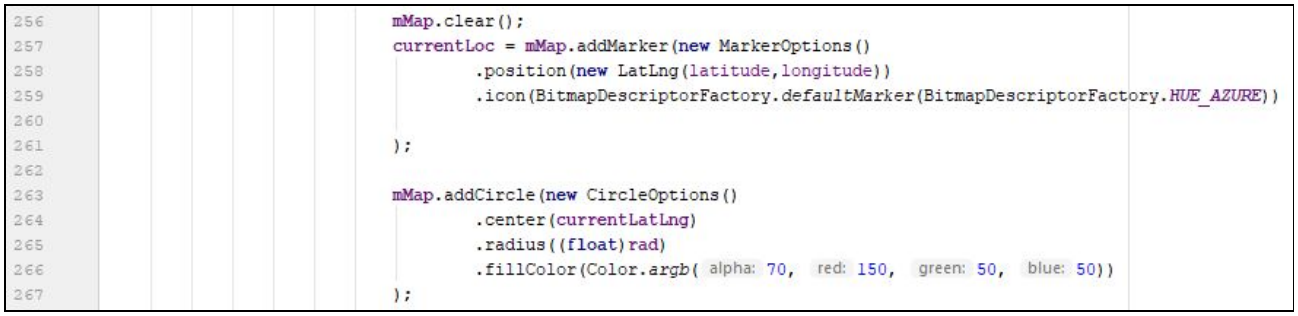
Фигура 3.19 Запазване на локацията в базата

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. Това е представено във *фигура 3.20.*



```
256                 mMap.clear();
257                 currentLoc = mMap.addMarker(new MarkerOptions()
258                         .position(new LatLng(latitude,longitude))
259                         .icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_AZURE))
260
261                 );
262
263                 mMap.addCircle(new CircleOptions()
264                         .center(currentLatLng)
265                         .radius((float)rad)
266                         .fillColor(Color.argb( alpha: 70,  red: 150,  green: 50,  blue: 50))
267                 );
```

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 на разглеждания не съвпада с това на сегашният потребител. Методът isInRadius() (*фигура 3.21*) 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.

```
347 @   private boolean isInRadius(LatLng latLng, LatLng currentLatLng) {
348         float[] distance = new float[1];
349
350         Location.distanceBetween(latLng.latitude, latLng.longitude, currentLatLng.latitude, currentLatLng.longitude, distance);
351         dbRef.addListenerForSingleValueEvent(new ValueEventListener() {
352             @Override
353             public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
354                 if(dataSnapshot.child("Radiuses").hasChild(currentUserID)) {
355                     rad = (long) dataSnapshot.child("Radiuses").child(currentUserID).child("radius").getValue();
356                 }
357             }
358             @Override
359             public void onCancelled(@NonNull DatabaseError databaseError) {
360             }
361         });
362         return distance[0] <= rad;
363
364     }
```

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 (*фигура 3.22*).

```
for(final DataSnapshot user : dataSnapshot.getChildren()){
    if(user.hasChild( path: "last_location")) {
        final double user_latitude = (double) user.child("last_location").child("latitude").getValue();
        final double user_longitude = (double) user.child("last_location").child("longitude").getValue();
        if (!currentUserID.equals(user.getKey())
                && isInRadius(new LatLng(user_latitude, user_longitude), currentLatLng)
                && (user.hasChild( path: "visible") && (boolean)user.child("visible").getValue())) {
            dbRef.addListenerForSingleValueEvent(new ValueEventListener() {
                @Override
                public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                    if(dataSnapshot.child("Friendlists").child(currentUserID).hasChild(user.getKey())
                            && !dataSnapshot.child("Blocked").child(currentUserID).hasChild(user.getKey())
                    )
                    {
                        if(dataSnapshot.child("Locations").child(user.getKey()).hasChild( path: "last_seen_on")){
                            mMap.addMarker(new MarkerOptions()
                                    .position(new LatLng(user_latitude, user_longitude))
                                        .title(dataSnapshot.child("Users").child(user.getKey()).child("full_name").getValue().toString())
                                        .snippet("Last seen on: " + dataSnapshot.child("Locations").child(user.getKey()).child("last_seen_on").getValue()

                            );
                        } else {
                            mMap.addMarker(new MarkerOptions()
                                    .position(new LatLng(user_latitude, user_longitude))
                                    .title(dataSnapshot.child("Users").child(user.getKey()).child("full_name").getValue().toString())
                            );
```

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. Това е показано на *фигура 3.23* [9]

```
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));
```

Фигура 3.23 Извличане на локации от базата данни 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

Търсачката има 3 елемента:

1. Поле за въвеждане на текст
2. Бутон за търсене

3. Recycler View 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() (*фигура 3.24*)

```
52          searchEvemtsBtn.setOnClickListener(new View.OnClickListener() {
53              @Override
54              public void onClick(View view) {
55                  String input = searchInputText.getText().toString();
56                  SearchEvents(input);
57              }
58          });
```

Фигура 3.24 Търсене на събития

Резултатите от търсенето са изложени в Recycler View. За целта е използван ViewHolder клас, направен специално за събитията и техните полета - FindEventsViewHolder(*фигура 2.25*). Той разширява RecyclerView.ViewHolder класа и има полета за заглавие, описание и начална дата и час на събитието.

```
128     public static class FindEventsViewHolder extends RecyclerView.ViewHolder{
129         View mView;
130
131         public FindEventsViewHolder(@NonNull View itemView) {
132             super(itemView);
133             mView = itemView;
134         }
135
136         public void setTitle(String title){
137             TextView eTitle = (TextView) mView.findViewById(R.id.all_events_title);
138             eTitle.setText(title);
139         }
140         public void setDesc(String desc){
141             TextView eDesc = (TextView) mView.findViewById(R.id.all_events_desc);
142             eDesc.setText(desc);
143         }
144         public void setStartTime(String startTime){
145             TextView eStartTime = (TextView) mView.findViewById(R.id.all_events_startTime);
146             eStartTime.setText(startTime);
147         }
148
```

Фигура 3.25 FindEventsViewHolder класът

В метода searchEvents() (*фигура 3.27*) 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 (*фигура 3.26*) и ViewHolder (*фигура 3.25*) 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.

```
10  public class Event {
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      public String date_start;
17
18      public Event() {
19      }
20
21      public Event(Date dateCreated, Date dateStarted, String title, String description) {
22          this.dateCreated = dateCreated;
23          this.time_start = dateStarted;
24          this.title = title;
25          this.desc = description;
26      }
27
28      @Exclude
29      public Map<String, Object> toMap() {
30          HashMap<String, Object> result = new HashMap<>();
31          result.put("title", title);
32          result.put("desc", desc);
33          result.put("time_started", time_start);
34          SimpleDateFormat sdf = new SimpleDateFormat( pattern: "yyyy-MM-dd");
35          result.put("date_start", sdf.format(time_start.getTime()));
36          return result;
37      }
38  }
```

Фигура 3.26 Помощният клас Event

```
private void SearchEvents(String input) {
    Query searchEventsQuery = (eventsRef.orderByChild("title").startAt(input).endAt(input + "\uf8ff"));

    FirebaseRecyclerAdapter<Event, EventsActivity.FindEventsViewHolder> firebaseRecyclerAdapter = new FirebaseRecyclerAdapter<~>
            (
                    Event.class, R.layout.all_events_display_layout, EventsActivity.FindEventsViewHolder.class, searchEventsQuery
            ) {
        @Override
        protected void populateViewHolder(EventsActivity.FindEventsViewHolder viewHolder, Event event, final int position) {
            SimpleDateFormat sdf = new SimpleDateFormat( pattern: "yyyy-MM-dd HH:mm");
            Date start = Calendar.getInstance().getTime();
            try {
                start = sdf.parse(event.date_start);
            } catch (ParseException e) {
                e.printStackTrace();
            } catch (NullPointerException e){
                e.printStackTrace();
            }
            Toast.makeText( context: EventsActivity.this, event.date_start, Toast.LENGTH_LONG).show();
            if(start.after(new Date( Calendar.getInstance().getTime().getTime()- (2 * 24 * 60 * 60 * 1000)))) {
                viewHolder.setTitle(event.title);
                viewHolder.setDesc(event.desc);
                viewHolder.setStartTime(event.date_start);
                viewHolder.mView.setOnClickListener((view) → {
                        String visit_event_id = getRef(position).getKey();
                        Intent eventIntent = new Intent( packageContext: EventsActivity.this, EventProfileActivity.class);
                        eventIntent.putExtra( name: "visit_event_id", visit_event_id);
                        startActivity(eventIntent);
                });
            }
            else {
```

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 ибраното 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.

```
40          start_date = (Calendar) getIntent().getExtras().get("start_time");
41          test = findViewById(R.id.test);
42          title = findViewById(R.id.createTitle);
43          desc = findViewById(R.id.createDesc);
44
45          createEventBtn = findViewById(R.id.createEventBtn);
46          mAuth = FirebaseAuth.getInstance();
47          test.setText(start_date.getTime().toString());
48          eventsRef = FirebaseDatabase.getInstance().getReference( path: "Events");
49
50          eventsRef.addValueEventListener(new ValueEventListener() {
51              @Override
52              public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
53                  if(dataSnapshot.exists()){
54                      id = dataSnapshot.getChildrenCount();
55                  }
56              }
57
58              @Override
59              public void onCancelled(@NonNull DatabaseError databaseError) {
60              }
61          });
62
63          createEventBtn.setOnClickListener(new View.OnClickListener() {
64              @Override
65              public void onClick(View view) {
66                  eventsRef.child(String.valueOf(id+1)).child("title").setValue(title.getText().toString());
67                  eventsRef.child(String.valueOf(id+1)).child("desc").setValue(desc.getText().toString());
68                  eventsRef.child(String.valueOf(id+1)).child("time_created").setValue(Calendar.getInstance().getTime());
69                  SimpleDateFormat sdf = new SimpleDateFormat( pattern: "yyyy-MM-dd HH:mm");
70                  String formattedStartTime = sdf.format(start_date.getTime());
71                  eventsRef.child(String.valueOf(id+1)).child("date_start").setValue(formattedStartTime);
72                  eventsRef.child(String.valueOf(id+1)).child("ownerID").setValue(mAuth.getCurrentUser().getUid());
73                  Intent eventsPage = new Intent( packageContext: EventCreateActivity.this, EventsActivity.class);
74                  startActivity(eventsPage);
75              }
76          });
```

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 (*фигура 3.29)*. Освен това, има и ImageView с битова матрица за QR код, което е видимо само ако потребителя е създател на събитието.

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

Фигура 3.29 Взимане 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, както е показано на *фигура 3.30*. С помощта на Facebook SDK този ShareLinkContent се споделя на стената на потребителя.

```
shareBtn.setOnClickListener((view) → {
        final String text = " is attending " + title.getText() + " with ";

        dbRef.addListenerForSingleValueEvent(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                String userName = dataSnapshot.child("Users").child(currentUserID).child("full_name").getValue().toString();
                String latitude = dataSnapshot.child("Locations").child(currentUserID).child("last_location").child("latitude").getValue().toStri
                String longitude = dataSnapshot.child("Locations").child(currentUserID).child("last_location").child("latitude").getValue().toStr

                ShareLinkContent linkContent = new ShareLinkContent.Builder()
                        .setQuote(userName + text)
                        .setContentUrl(Uri.parse("https://www.google.com/maps/search/google+maps/@"+ latitude + "," + longitude +",15.89z"))
                        .setShareHashtag(new ShareHashtag.Builder()
                            .setHashtag("#UsingPINS")
                            .build())
                        .build();
                if(ShareDialog.canShow(ShareLinkContent.class)){
                    shareDialog.show(linkContent);
                }
            }

            @Override
            public void onCancelled(@NonNull DatabaseError databaseError) {

            }
        });
```

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. Това е представено във *фигура 3.31*

```
    if(Build.VERSION.SDK_INT >= Build.VERSION_CODES.M){
        if(CheckPermission()){
            Toast.makeText( context: QRscannerActivity.this,  text: "Permission granted", Toast.LENGTH_LONG).show();
        }
        else {
            RequestPermission();
        }
    }
}

private void RequestPermission() {
    ActivityCompat.requestPermissions( activity: this, new String[]{CAMERA}, REQUEST_CAMERA);
}

private boolean CheckPermission() {
    return (ContextCompat.checkSelfPermission( context: QRscannerActivity.this, CAMERA) == PackageManager.PERMISSION_GRANTED);
}
```

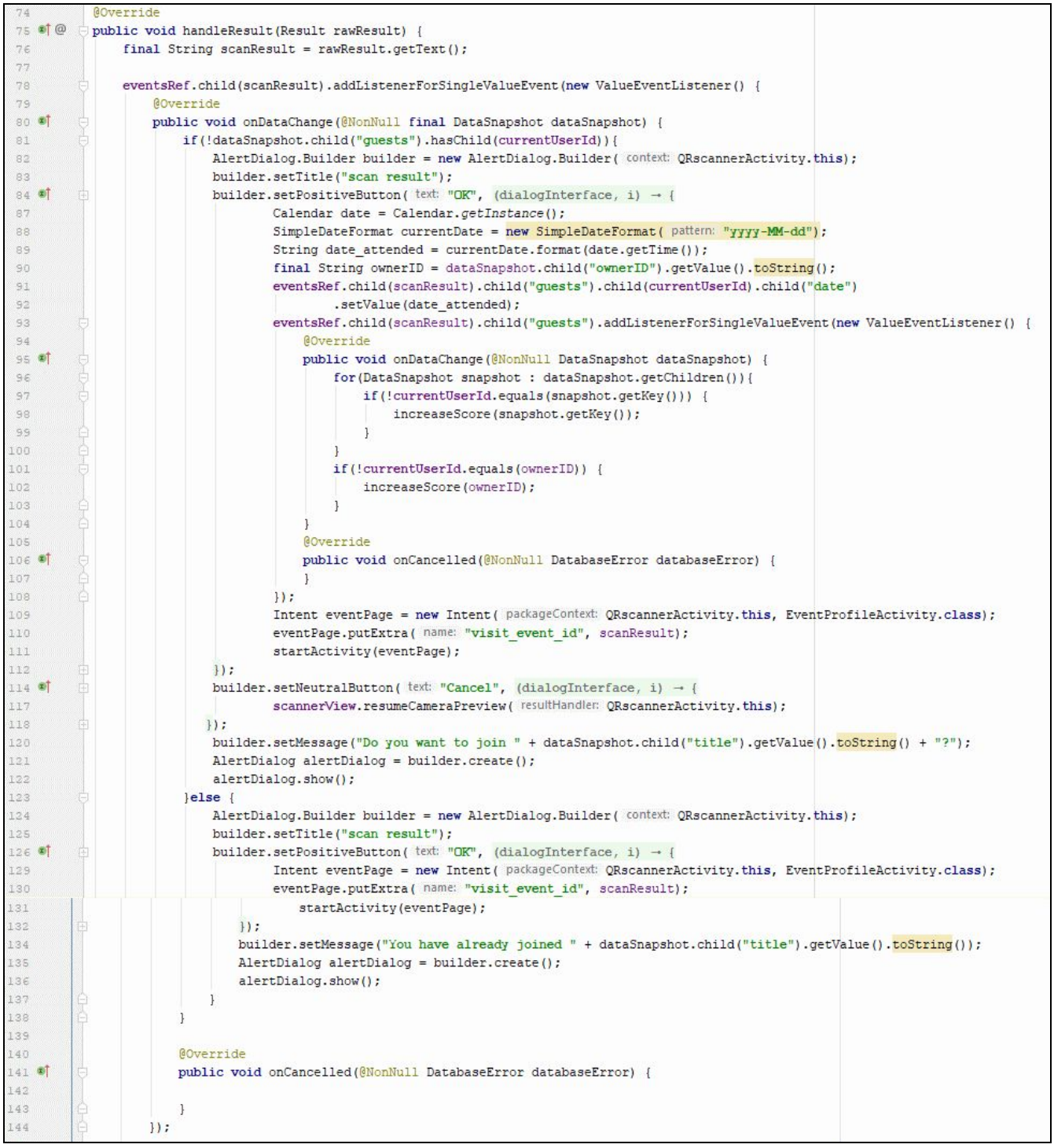
Фигура 3.31 Проверка на позволение за ползване на камерата

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() (*фиг. 3.33*) 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 уууу-MM-dd формат. Освен това се извиква метода increaseScore() (*фигура 3.32*) за всеки присъстващ, който увеличава рейтина му със сегашния потребител.

```
149     private void increaseScore(final String user) {
150         dbRef.addListenerForSingleValueEvent(new ValueEventListener() {
151             @Override
152             public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
153                 long score;
154                 if(dataSnapshot.child("Scores").child(user).hasChild(currentUserId)) {
155                     score = (long) dataSnapshot.child("Scores").child(user).child(currentUserId).getValue();
156                     score += 1;
157                 } else {
158                     score = 1;
159                 }
160                 dbRef.child("Scores").child(user).child(currentUserId).setValue(score);
161                 dbRef.child("Scores").child(currentUserId).child(user).setValue(score);
162             }
163
164             @Override
165             public void onCancelled(@NonNull DatabaseError databaseError) {
166
167             }
168         });
```

Фигура 3.32 Методът increaseScore() за трупане на рейтинг



```
@Override
public void handleResult(Result rawResult) {
    final String scanResult = rawResult.getText();

    eventsRef.child(scanResult).addListenerForSingleValueEvent(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull final DataSnapshot dataSnapshot) {
            if(!dataSnapshot.child("guests").hasChild(currentUserId)){
                AlertDialog.Builder builder = new AlertDialog.Builder( context: QRscannerActivity.this);
                builder.setTitle("scan result");
                builder.setPositiveButton( text: "OK", (dialogInterface, i) -> {
                        Calendar date = Calendar.getInstance();
                        SimpleDateFormat currentDate = new SimpleDateFormat( pattern: "yyyy-MM-dd");
                        String date_attended = currentDate.format(date.getTime());
                        final String ownerID = dataSnapshot.child("ownerID").getValue().toString();
                        eventsRef.child(scanResult).child("guests").child(currentUserId).child("date")
                                .setValue(date_attended);
                        eventsRef.child(scanResult).child("guests").addListenerForSingleValueEvent(new ValueEventListener() {
                            @Override
                            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                                for(DataSnapshot snapshot : dataSnapshot.getChildren()){
                                    if(!currentUserId.equals(snapshot.getKey())) {
                                        increaseScore(snapshot.getKey());
                                    }
                                }
                                if(!currentUserId.equals(ownerID)) {
                                    increaseScore(ownerID);
                                }
                            }
                            @Override
                            public void onCancelled(@NonNull DatabaseError databaseError) {
                            }
                        });
                        Intent eventPage = new Intent( packageContext: QRscannerActivity.this, EventProfileActivity.class);
                        eventPage.putExtra( name: "visit_event_id", scanResult);
                        startActivity(eventPage);
                });
                builder.setNeutralButton( text: "Cancel", (dialogInterface, i) -> {
                        scannerView.resumeCameraPreview( resultHandler: QRscannerActivity.this);
                    });
                builder.setMessage("Do you want to join " + dataSnapshot.child("title").getValue().toString() + "?");
                AlertDialog alertDialog = builder.create();
                alertDialog.show();
            }else {
                AlertDialog.Builder builder = new AlertDialog.Builder( context: QRscannerActivity.this);
                builder.setTitle("scan result");
                builder.setPositiveButton( text: "OK", (dialogInterface, i) -> {
                        Intent eventPage = new Intent( packageContext: QRscannerActivity.this, EventProfileActivity.class);
                        eventPage.putExtra( name: "visit_event_id", scanResult);
                            startActivity(eventPage);
                    });
                    builder.setMessage("You have already joined " + dataSnapshot.child("title").getValue().toString());
                    AlertDialog alertDialog = builder.create();
                    alertDialog.show();
                }
            }

            @Override
            public void onCancelled(@NonNull DatabaseError databaseError) {

            }
        });
```

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 onClickListener, 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(), представени във *фигура 3.34*. 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)
74                              .child("visible").getValue();
75                      if (visibility) {
76                          visibilitySwitch.setChecked(true);
77                          visibilityText.setText("Visible");
78                      } else {
79                          visibilityText.setText("Not Visible");
80                      }
81                  }
82                  if(dataSnapshot.child("Radiuses").hasChild(currentUserID)){
83                      long rad = (long)dataSnapshot.child("Radiuses").child(currentUserID)
84                              .child("radius").getValue();
85                      seekBar.setProgress(((int) rad)/250);
86                  }
87              }
88              @Override
89              public void onCancelled(@NonNull DatabaseError databaseError) {
90              }
91          });
92      }
93
94      public void manipulateSeekbar(){
95          seekBar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
96              int progress_value;
97              @Override
98              public void onProgressChanged(SeekBar seekBar, int i, boolean b) {
99                  progress_value = i;
100
101                 if(progress_value >= 16){
102                     radiusText.setText(progress_value/4 + "." +(progress_value%4)*250 + "km");
103                 }else {
104                     radiusText.setText(progress_value*250 + "m");
105                 }
106             }
```

Фигура 3.34 Методите maintainUI() & manituplateSeekbar()

Четвърта Глава

Изисквания. Ръководство на потребителя

4.1 Изисквания

- Android устройството трябва да е с минимална версия 8.0 Lolipop (Api level 26).
- Потребителят трябва да разреши приложението да използва интернет, локация и камерата на устройството.
- Потребителят трябва да разреши на устройството да инсталира приложения от неизвестни източници, за което трябва да навигира през ***Настройки*** > ***Сигурност*** (на по-стари устройства ***Настройки*** > ***Приложения***).

4.2 Инструкции за ползване

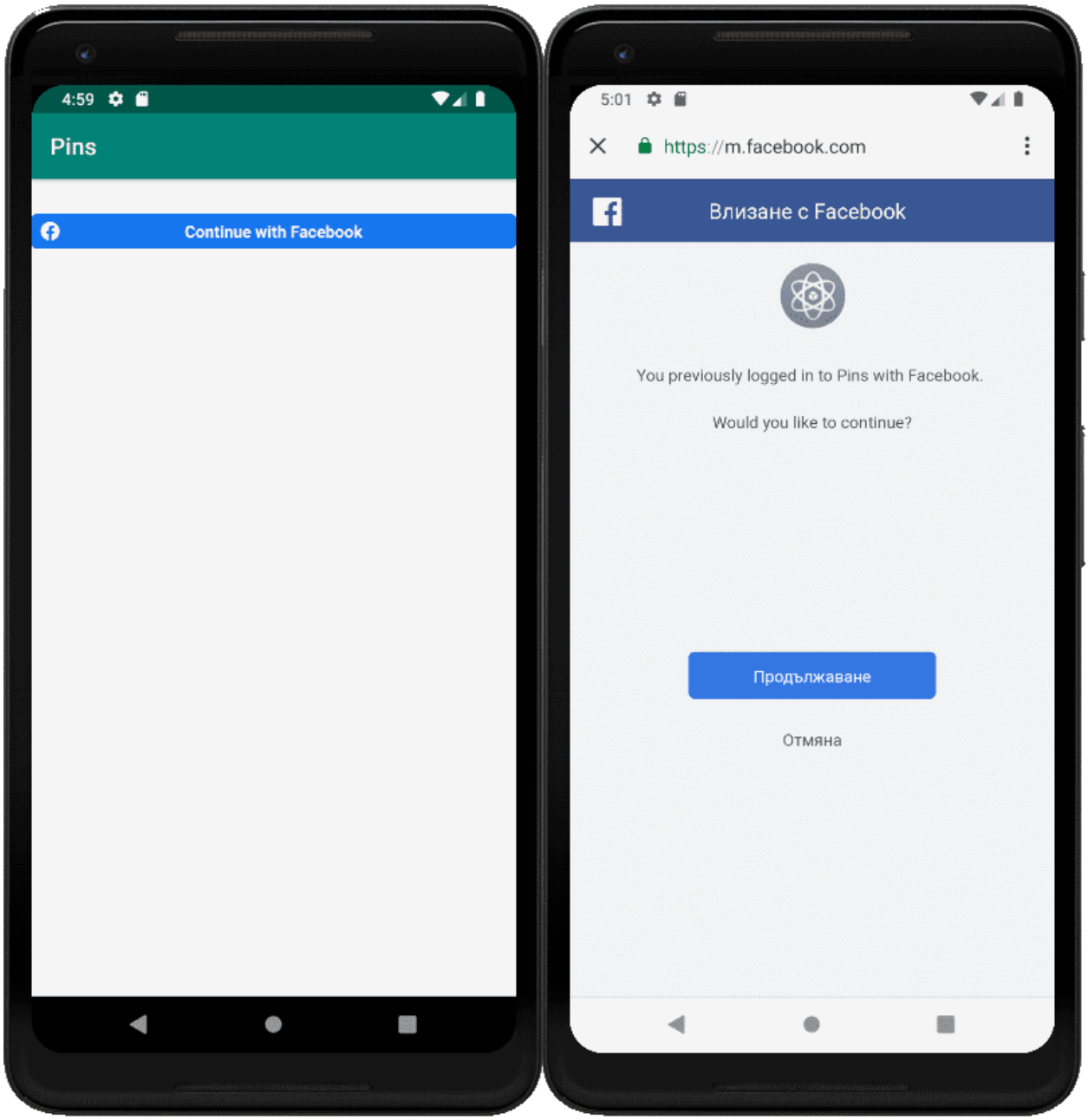
4.2.1 Инсталация

За инсталация на приложението, потребителите трябва да изтеглят на мобилното устройство APK (Android Package) файла **app-release.apk**, намиращ се в app/release папката на проекта, или в приложения диск. Потребителите могат да открият проекта и **app-release.apk** файлът също в github хранилището - https://github.com/SimeonHG/Pins. Ако потребителят вече е разрешил инсталацията от неизвестни източници, той трябва само да отвори **app-release.apk**.

4.2.2 Регистрация

Регистрацията в приложението става посредством Facebook автентикация. С отварянето на приложението потребителя ще види бутон

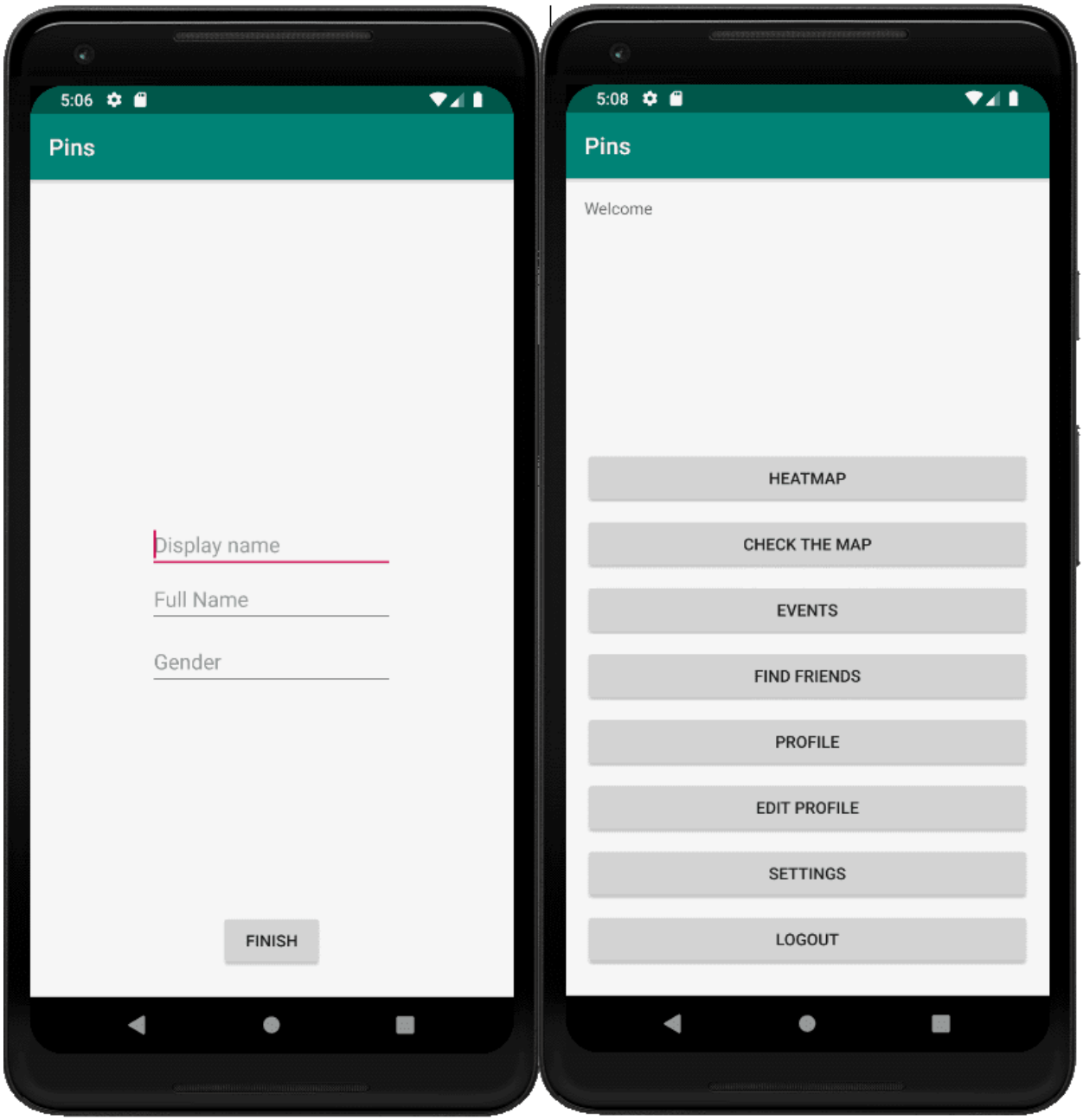
“Continue with facebook”, който ще го отведе до мобилната версия на Facebook за да потвърди автентикацията, както е показано на *фигура 4.1*.



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. Двете страници са представени на *фигура 4.2*.



Фигура 4.2 Вписване на информация за потребителя & Началната страница

4.2.3 Навигация из приложението

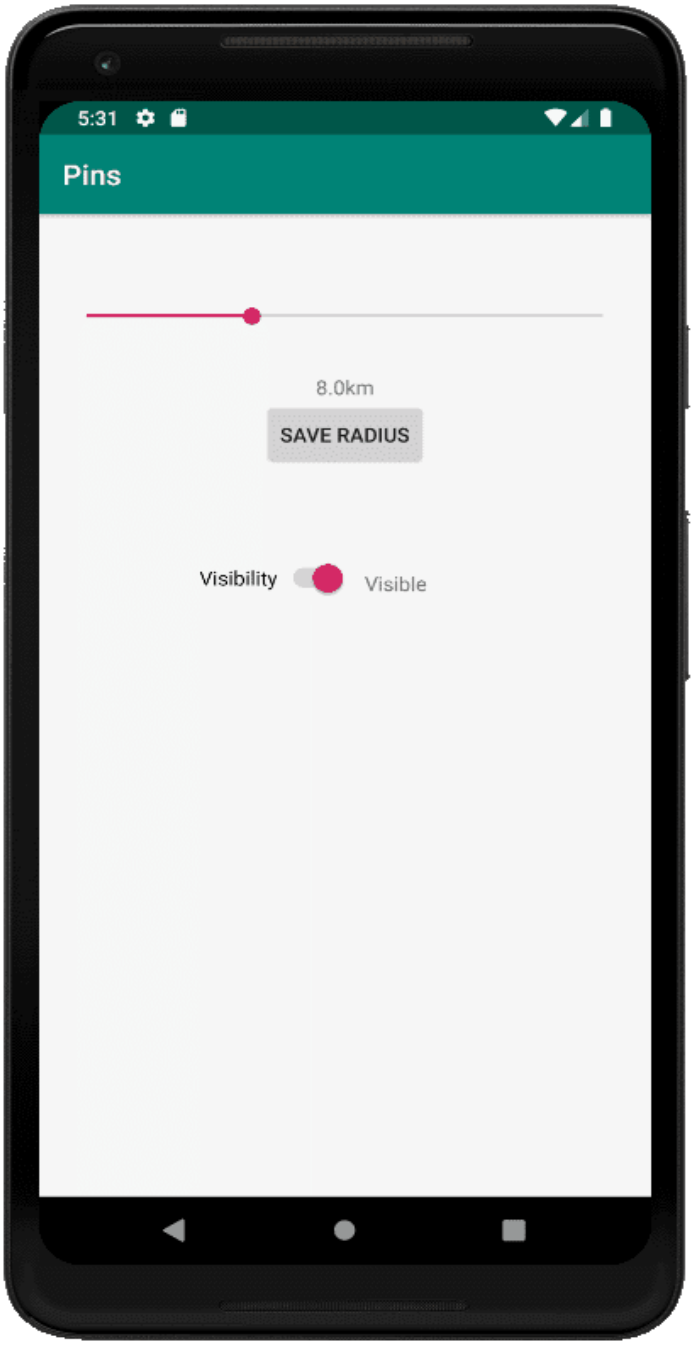
От тук потребителя вече може да ползва приложението свободно.

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”.

4.2.4 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. Страницата е показана на *фигура 4.3*

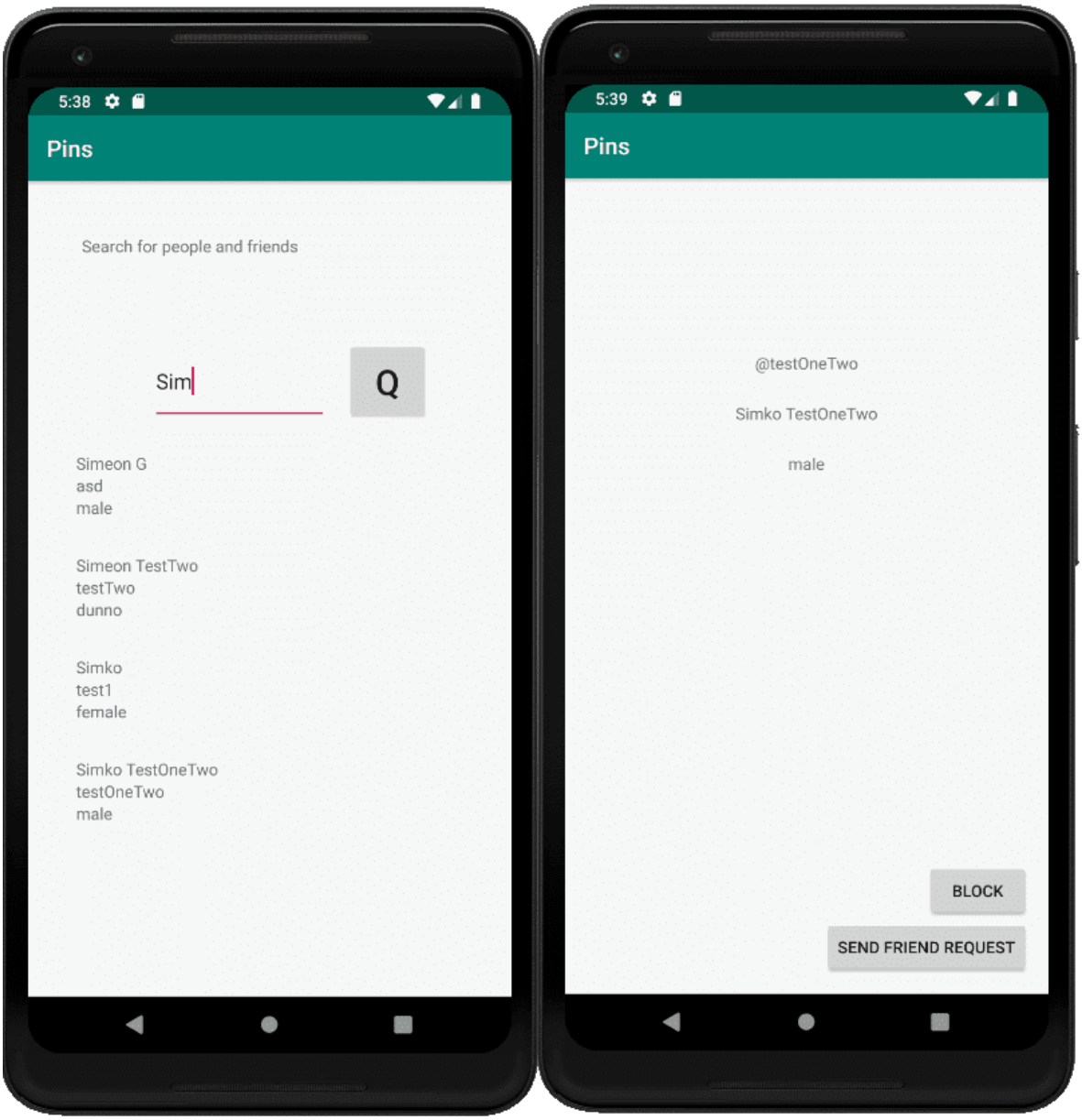


Фигура 4.3 Страницата за настройки

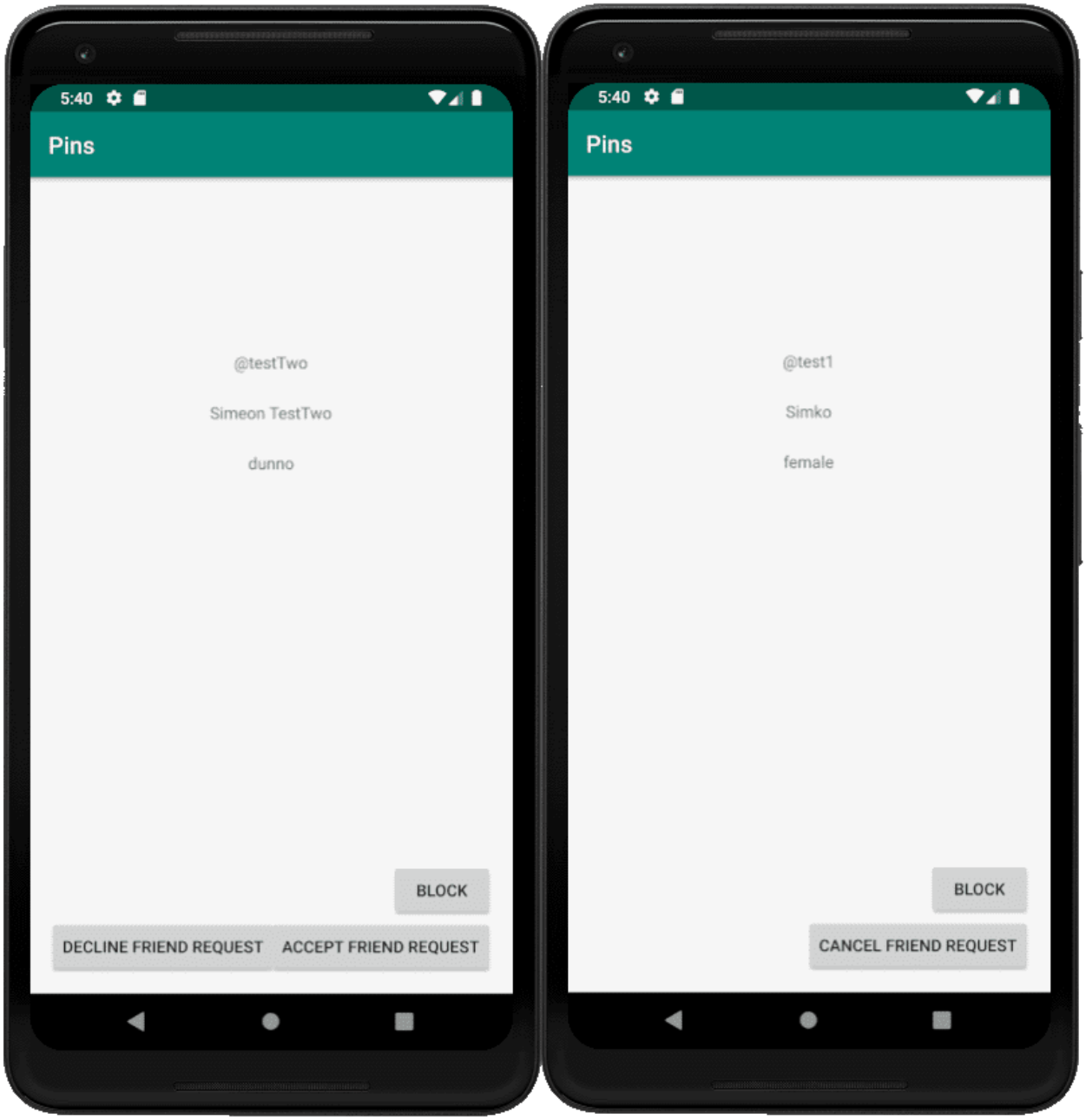
4.2.5 Намиране на приятели

В страницата за разглеждане на профили (*фигура 4.4*) 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 (*фигури 4.4 и 4.5*), където 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

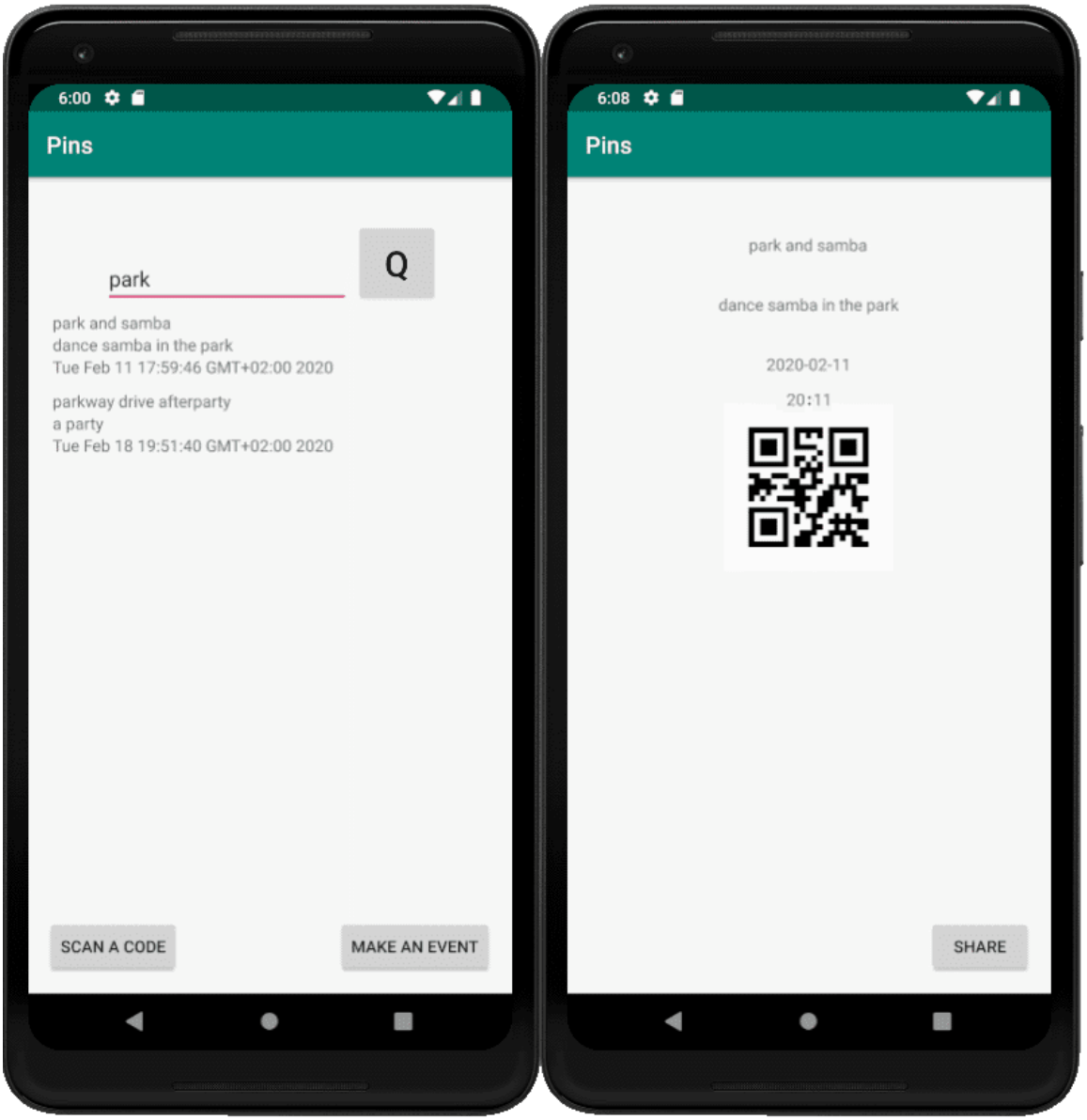


Фигура 4.5 Получена покана за приятелство & Изпратена покана

4.2.6 Събития

В страницата за събития (*фигура 4.6*) 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

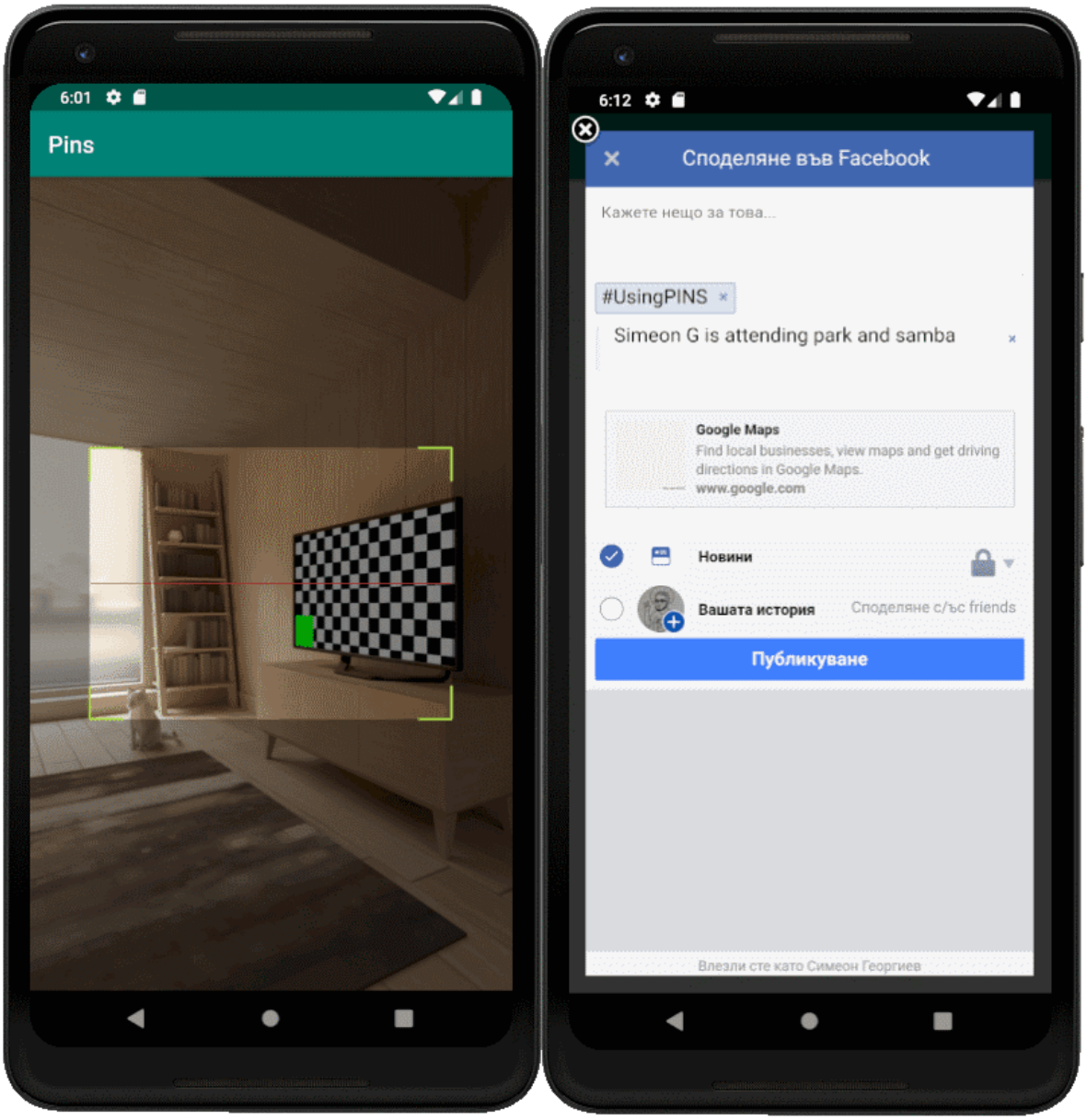
чекира в събития. Бутона за създаване отвежда потребителя в страница, където той избира дата и часа на събитието, последвано то страница за име и описание. Ако потребителя е създател на дадено събитие, то когато отвори профила му (*фигура 4.6*), ще има QR код, който трябва да бъде сканиран от другите потребители, за да се чекират.



Фигура 4.6 Търсачка на събития & Профил на събитие

Чекирането става чрез 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. Това е показано на *фигура 4.7*

След като вече присъства на събитието, потребителя може да натисне бутона “Share” и да сподели това на стената си във Facebook (*фигура 4.7*).

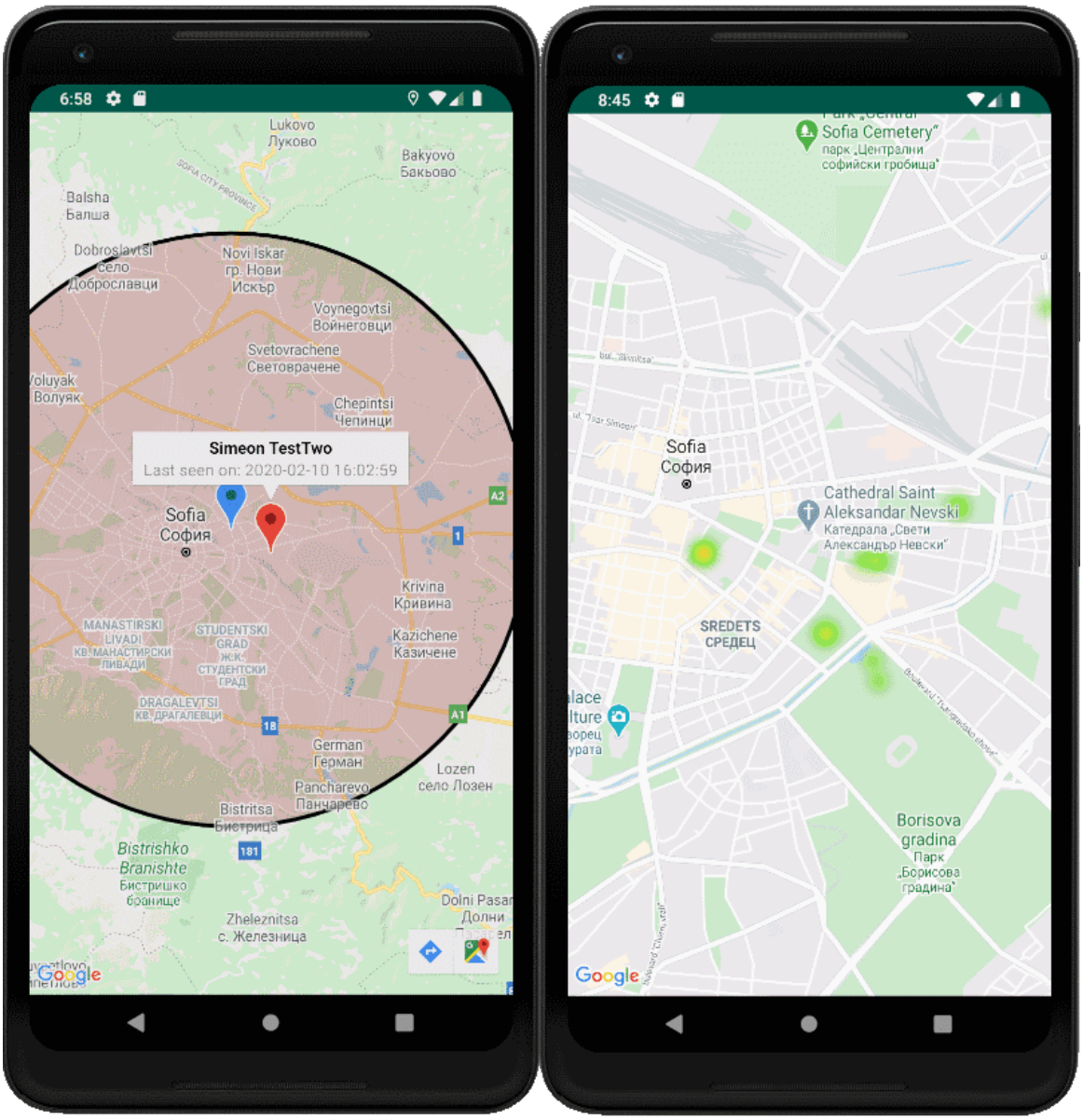


Фигура 4.7 Скенер на QR кодове & Споделяне във Facebook

4.2.7 Карти

4.2.7.1 Карта с приятели

На тази карта (*фигура 4.8 вляво*) са показани локациите на всички откриваеми приятели, които са в избрания 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

4.2.7.2 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. Тази карта е представена на *фигура 4.8 вдясно*

Заключение

Функционалните изисквания на дипломната работа в по-голямата си част са спазени. Въпреки това приложението е все още в разработка, но може да бъде използвано и в сегашния му вариант.

Бъдещото развитие на проекта се състои в добавянето на още функционалности, като например:

- Покани за събития
- Интеграция на Google Places или Facebook Places, като по този начин потребителите ще могат да добавят определени заведения, паркове или градинки за събитията.
- Известия, когато приятел е наблизо
- Профилни снимки и връзки към социални мрежи
- Запазване на любими локации

Планирани са също подобрения на вече имплементираните функционалности, добавяне на привлекателен за потребителите дизайн и други.

Използвана литература

1. https://developers.google.com/maps/documentation

2. https://developer.android.com/docs

3. https://firebase.google.com/docs/android/setup

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

5. https://github.com/zxing/zxing

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

7.https://stackoverflow.com/questions/30779596/best-to-use-android-studio-or-intellij-idea-wth-plugins/30784212#30784212

8. https://capgemini.github.io/design/sql-vs-nosql/

9. https://developers.google.com/maps/documentation/android-sdk/utility/heatmap

Съдържание