POLITECNICO
MILANO 1863

Neighborhood Security

Software Design Document

Simone Ripamonti, Luca Stornaiuolo

July 11, 2017

Neighborhood Security Design Document
Version v1.0, June 2017

Contents

1 Introduction

1.1 About the Design Document

In this section we want to introduce briefly our application.

The application is developed for the course of "Design and Implementation of Mobile Applications" at Politecnico di Milano. The goal of the course is to efficiently design and implement a mobile application on a platform of our choice. This documents illustrates the decisions we made in order to accomplish this goal.

This Software Design Document is a document that provides documentation that will be used as a overall guidance to the architecture of the software project. In this document we provide a documentation of the software design of the project, including use case models, class and sequence diagrams.

The purpose of this document is to provide a full description of the design of *Neighborhood Security*, a native Android application, providing insights into the structure and design of each component.

1.2 Platform

The choice of the platform was left to us. We decided to develop for Android.

Main reasons:

- Android native applications are written in Java, which is a language that is familiar to us
- We both use Android devices everyday, so we are familiar with the environment and we have the possibility to test the application on our physical devices

1.3 Choice of the application

We had full choice on the purpose of the application.

The choice fall on an application that could support neighborhood watch associations, that are group of civilians devoted to crime and vandalism prevention within a neighborhood. The application gives the possibility to alert other user about criminal events that have just happened.

The particularity that characterizes this application is the possibility to receive notification about events in the user's favorite locations in real time and to brows a map of all the signaled events.

The idea for the application came from personal experience. In our towns, groups of people are currently reporting criminal events using WhatsApp group chat, but this is not user friendly:

- You need to know and contact one of the administrators of the group
- You need to share your personal phone number and information
- You cannot have an ensemble view of all the reported events

1.4 Risk analysis

During the problem analysis, we identified some risks that can compromise the correct development of the project.

We had to be very careful during the requirements collection phase, because the requirements must be clear so to avoid delays due to misunderstandings. Formal modeling of the application allows the reader not to be confused by it and to learn requirements in a clear way.

Another possible delay was learning Android concepts or techniques in advanced stages of the project, leading to possible code rewriting and inevitable loss of time.

1.5 Time constraints

Our time constraints were not as strict as they would have been in a project developed for real stakeholders. We had no precise and punctuated deadlines, but to deliver the project among the different call dates for the course.

We begin to develop our application at the beginning of the second semester. Developing, testing and creating documentation took less than 6 months. Even the server side part was built and maintained during the same time span. We started development at February 2017 and complete it in July 2017.

The team is composed by two people with both solid Java knowledge but little experience in Android, thus we had to dedicate some time to learn the environment.

1.6 Stakeholders

The main stakeholders of our project is the professor and the other students that might be attending the presentation. The audience wants to have a clear idea about the project idea and realization. Professor's main goal instead is to check if the concepts taught during the course are clear to us and if we succeeded in implementing them inside our project.

Even though we had not currently planned to release our application in the Google Play Store, we designed it keeping simple and intuitive, suitable for every kind of users.

We used English as main language for our project, due to its diffusion in the world, and we currently provide an Italian translation. If we ever decide to officially release *Neighborhood Security*, we will introduce other major languages to make it more usable. This can be easily done using localized string resource feature of Android.

2 General overview

2.1 Idea

Neighborhood Security is a native Android application that provides users the possibility to report criminal events and receive notification about favorite areas.

The application does not require users to be authenticated in order to browse the reported events, indeed everyone can easily browse the reported events over a map and understand the safety of a certain location by simply checking how many reports have been done and of which kind.

Users can register an account, using a combination of email and password or using their Facebook or Google account. Registered users have access to a new set of functionalities, such as subscribing to notification about a certain location, reporting events and voting already submitted events. Once subscribed to a location, users can now receive real time notifications as soon as another user report a criminal event.

The main components of the project are:

- A server side part, composed by a relational database that contains events and subscriptions and the application logic that is in charge of notifying interested users about events;
- A client side part, the Android application, which tasks are to query the relational database, show the gathered informations and provide users a point of interaction with the service.

2.2 Core features

Here we list the core functionalities that can be found in our application. We decided to divide them considering the fragment in which is organized, allowing the reader to easily understand them and where they could be found in the application.

- Home
  - First point of interaction with the user. A simple showcase presents to the user the main features of the application, such as seeing reported events and subscribe to notifications.
  - A lateral menu is provided to allow the user to access other features, such as the creation of an account that gives access to the interesting features of the application.
  - A set of icons provides the user with insights about his usage of the application.
- Authentication
  - Allows the user to authenticate using three different ways: a combination of email and password, a Google or Facebook account.
  - If they choose email and password authentication, users also have the capability to reset the chosen password in case they have forgotten it.

  - Choosing the other methods, users are required to allow the application to access their personal information such as profile data and email address.

- Event map
  - Provides users the possibility to see reported events placed on a map. Different kind of events are represented by means of different marker icons.
  - Last week tweets with the hashtag #NeighborhoodSecurity are displayed in the map.
  - If the user gave permission to access location informations, the map is centered in the current position.
  - By clicking on an event marker, user can see more detailed information about the reported event.
  - By clicking on a tweet marker, the content of the Tweet is displayed.
  - In order to decrease the number of markers placed in the map, events are clustered according to their position. By clicking on a cluster, users can see the list of events that are contained in that cluster.
  - A search bar is provided in order to easily find and move to a particular location, with an auto-complete feature provided by Google.

- Event list
  - Shows a list of events that can be refreshed if the list is related to a specific location, subscription or submitting user.
  - Users can easily understand the kind, by looking at characteristic icon, the location and date of the event.
  - By clicking on the event, a new view containing its details is shown.
  - If the user is the creator of a particular event, he can delete it by simply long clicking on the event.

- Event creation
  - Only authenticated users have access to this feature.
  - Events are categorized according to the kind of event: carjacking, burglary, robbery, theft, shady people and scammers.
  - Users must insert a brief description of the event.
  - Position of the event can be chosen by inserting the place name (using Google Places auto-complete service) or by inserting pure coordinates, that can be obtained by using the location service if authorized.

- Event detail
  - Shows all the informations available about the selected event. Reporter name is never shown to other users by design.
  - A small map allows users to understand more precisely where the event happened.
  - Authenticated users can vote the event. The more an event has been voted, the more we consider reliable the reported event.

- Subscription list
  - Shows a list of subscriptions, if any, of the currently logged in user. The user can enable or disable notifications about a particular subscription and delete them if he is no longer interested in.
  - By clicking on a subscription, a new view containing the list of events that match the subscription is shown
- Subscription creation
  - Only authenticated users have access to this feature
  - Subscription are characterized by two components: a position, that can be a place or GPS coordinates, and a radius, up to 2000 metres.

2.3 General qualities

Our application has several characteristics of accessibility and usability. The user interface is easy to use and intuitive, so that the user quickly learns what to do. The design is enthralling and captivating to guarantee the best experience possible.

- Usability: the main actor of the system is the end user. For this reason we decided to make the user interface as easy as possible, but still keeping all the functionalities needed to provide the best user experience.
- Nice User Interface: we tried to make our application as nice as possible, following Google's Material Design guidelines to have a clean and simple design.
- No Account Required: we tried to provide the greatest number of features without needing users to register for an account.
- Offline Mode: even when the user is not connected to a network, he can continue to use the application thanks to our caching system.

2.4 Functional requirements

In this section we present the requirements necessary to the correct behavior of the system:

General requirements:

- The application has to be comprehensible by as many people as possible, so we decided to use English language and provide an Italian translation.
- The application has to start with a splash activity, meanwhile the initial settings are done.
- The application has to provide a home activity that gives the user access to all the application functionalities.
- The application needs to allow user to create an account but still partially work if the user doesn't want to.
- The application has to provide users the possibility to report new events and create subscriptions.
- The application has to provide a way to browse events and to manage active subscriptions.

Home requirements

- On first run, Home activity should introduce the application features to the user.
- Home activity should provide immediate access to Subscription List functionality and Event Map functionality.
- Home activity should have a left drawer to give user access to Authentication and additional features, such as Event and Subscription creation and user's Event list which are available only to authenticated users. The drawer should also display the current logged in user, if any.

Event map requirements

- Event map activity should display events in the map in a clear and distinguishable way, clustering events by positions and using different icons according to the event type.
- Event map activity should display the most recent Tweets containing the hashtag #NeighborhoodSecurity, placing them on the map according to their position.
- Event map activity should provide a search bar in which it is possible to search for a location and then update map position to the request location.
- Event map activity should provide the possibility to create an event or subscription at a particular map position.

Event create requirements

- Event create activity should be accessible only to registered users.
- Event create activity should display an event type list where to select: carjacking, burglary, robbery, theft, shady people and scammers.
- Event create activity should allow users to enter a brief description of the event.
- Event create activity should allow user to select a location based on its address or GPS coordinate.
- Event create activity should inform the user about the success or failure of the request.

Event list requirements

- Event list activity should display all the events matching one criteria (id of the creating user, area of the event, subscription id) or provided in a list.
- Event list activity should provide a way to refresh the list of events if they do not belong to a provided list.
- Event list activity should provide a button to move to Event create activity.
- Event list activity should provide a search bar to select a new location to search for new events to be displayed.
- Event list activity should allow event's creator to delete the events by using a contextual menu.
- Event list activity should sort events by creation date.

Event detail requirements

- Event detail activity should display all the informations belonging to a single event: event type, description, location, creation date, number of votes.
- Event detail activity should provide a map to easily understand where the event took place.
- Event detail activity should provide a button to vote the currently displayed event, and a way to un-vote it.

Subscription create requirements

- Subscription create activity should allow the choice of the location based on address or GPS coordinates.
- Subscription create activity should allow the choice of the radius of interest up to 2000m.
- Subscription create activity should inform the user about the success or failure of the request.

Subscription list requirements

- Subscription list activity should be accessible only by registered users.
- Subscription list activity should only display the subscriptions of the currently logged in user.
- Subscription list activity should provide a way to refresh the list of subscription.
- Subscription list activity should provide a button to move to Subscription create activity.
- Subscription list activity should provide a way to enable or disable notification about a given subscription.

Authentication requirements

- Authentication activity should be accessible only to users not currently logged in.
- Authentication activity should allow users to register or login using email or to use their Facebook and Google accounts.
- Authentication activity should delegate registration/access using Facebook or Google to the respective libraries.
- Authentication activity should delegate registration/access using email to Email Authentication activity.
- Authentication activity should return the user back to Home activity if the authentication is successful.

Email Authentication requirements

- Email Authentication activity should allow user to register, login or reset the password.
- Email Authentication should display fields for username, password and email, which might be hidden according to the operation the user is going to perform.

2.5 Non-functional requirements

These are the non-functional requirements that are necessary to guarantee a functional application.

- **Portability**: in order to be used by the largest number of user possible, the application should also be extended to iOS platform and desktop users. This are modifications that will be further introduced by us in the future.
- **Stability**: the system should always be available to be used at any time it is needed. System failures and server side crashes must be avoided in order to guarantee this requirement.
- **Availability**: the services must always be up and running even during a failure period. An administrator must restore the service as soon as possible. A backup facility must be present in order to do so.
- **Reliability**: the data must be reliable, meaning that they are trustworthy and not corrupted by any other. So the remote database must be protected and secure for this purpose.
- **Efficiency**: the application needs to use as less resource as possible. Algorithms and data structures have been developed to optimize the consumption of resources in the best way possible.
- **Extensibility**: the application was programmed with the idea that further extensions could be added in a simple way without deeply modifying the core of the application.
- **Maintainability**: code is easily readable and commented, in this way it is maintainable by future programmers.

3 Data design

3.1 Internal software data structure

The structure of our application is divided in two main parts: client-side and server-side.

All the event, subscription and user data reside on a external database. These data are cached on the local database to allow the application to work also if Internet access is currently unavailable. Neighborhood Security is based on data displaying and data creation, so the data is isolated and accessed only with a Model-View-Controller system. The user by navigating in the different sections of the application will trigger data request from the database. Locally available data is shown immediately, meanwhile new fresh data is downloaded in an asynchronous way and the views are update as soon as possible. The communication between the server and the database is handled by a Retrofit client that communicates to our Rest web service and performs CRUD operations. The web service is hosted on Heroku[1] and has been developed using Java 8 and Jersey[2]. The permanent database storage is a MySQL database, called JawsDB[3], also hosted by Heroku.

On the client side there is a mirrored data structure of the data base, that is represented by model classes for Event, Subscription and User that will be filled after a specific request to the web server and stored in these objects. Locally stored data is timestamped, in this way we can remove old data in an automatized way. All the data obtained as the result of a request is stored in the local SQLite database. We also use shared preferences system of Android to locally store other information such as the list of already voted events and the list of active subscription notifications. In this way, Internet connection is not needed to show already cached events and subscriptions

3.2 Database design & implementation

This kind of design has the aim to provide an abstract description of the data used by the application, independently from the database model. In our case only two simple classes represent the entire data structure needed on the client side and are Event and Subscription. On the server side, we need also to know information about registered users, in order to handle create or delete operations and to allow sending notifications.

Below we put the attribute of each table object, their meaning, and their eventual relationship with other elements:

- **Event**: This class represents the event object
  1. **ID**: primary key, is a positive integer
  2. **Date**: is the timestamp of creation
  3. **Event Type**: is a value from an enumeration that categorizes the events: car-jacking, burglary, robbery, theft, shady people and scammers
  4. **Description**: is a string describing the event

[1] Heroku, https://www.heroku.com/
[2] Jersey, https://jersey.github.io/
[3] JawsDB, https://www.jawsdb.com/

5. **Country**: is a string representing the country where the event took place
6. **City**: is a string representing the city where the event took place
7. **Street**: is a string representing the street where the event took place
8. **Latitude**: is a double representing the latitude where the event took place
9. **Longitude**: is a double representing the latitude where the event took place
10. **Votes**: is an integer representing the number of votes received by the event
11. **Submitter ID**: is a string representing the creator user, foreign key to User ID

- **Subscription**:
  1. **ID**: primary key, is a positive integer
  2. **User ID**: is a string representing the creator user, foreign key to User ID
  3. **Minimum Latitude**: is a double representing the bottom bound of the area of interest
  4. **Maximum Latitude**: is a double representing the top bound of the area of interest
  5. **Minimum Longitude**: is a double representing the left bound of the area of interest
  6. **Maximum Longitude**: is a double representing the right bound of the area of interest
  7. **Radius**: is an integer approximating the radius of the area
  8. **Country**: is a string representing the country in the center of the area
  9. **City**: is a string representing the city in the center of the area
  10. **Street**: is a string representing the street in the center of the area
- **User**:
  1. **ID**: primary key, is a string provided by Firebase and guaranteed to be unique across all the users registered to the application
  2. **Name**: is a string representing the name of the user
  3. **Email**: is a string representing the email of the user
  4. **FCM**: is a string representing a Firebase Cloud Messaging token of the user's last used device
  5. **Superuser**: is a boolean that identifies if the user is a superuser or not, superusers bypass ownership check when deleting events or subscriptions

At the end the choice falls on a SQL database for these reasons:

- All the operations we are interested in are easily performed by writing simple SQL statements
- Both of us are self-confident in using SQL databases

Our remote database is simply organized in five tables:

1. Event table: contains all the events
2. Votes table: contains all the votes registered as a pair event id and user id

3. Subscription table: contains all the subscriptions
4. User table: contains all the users
5. Authorization table: contains all particular authorizations as a pair user id and user level, where 1 is super user

Our local cache database is simpler since all the data is contained in two tables:

1. Event table: contains all the cached events, also with votes and timestamps
2. Subscription table: contains all the cached subscriptions, also with timestamps

4 Architectures and component level design

4.1 System architecture

Neighborhood Security is divided in two main components: one is the Android client-side application and the other one is a server-side Java application. Now we will see them in detail.

4.1.1 Client-side application

This part of the application is divided in two different parts: the functional components represented by Java code and the graphical ones written in XML. The **Java code** provides all the functionality and methods to handle the graphics and user's interactions with the application. Some of the tasks it completes are for example: retrieve and store data, visualize data, display notifications, trigger events when the user interacts with the UI, etc.

The graphical component, all the **XML layout files**, are simply the interface that is visualized to the users. It is composed by various elements such as the navigation drawer, button, text boxes, images and other views of any kind, that are used to display all the informations and allow the user to interact easily with the application.

4.1.2 Server-side application

The server component of *Neighborhood Security* is represented by a Java EE (Enterprise Edition) application, that provides a REST web service built using Jersey. The application is hosted by Heroku, which also hosts our SQL database based on JawsDB. The application interacts with the database using a JDBC (Java DataBase Connectivity) driver. The server application has two main objectives: a) expose the data on the database to the clients; b) send notifications to interested users, using Firebase Cloud Messaging. Because of this, the communication can be considered bidirectional.

The architectural style is the usual of the enterprise application, based on different layers, distributed in different physical devices.

4.1.3 Client-server interaction

Now we will focus on the details regarding the internal structure of the mobile application, in order to better understand the interaction between the client device and the server.

The paradigm we chose is the Model-View-Controller pattern for the entire system, because is the most suited for Android development and works well with the functionalities that we implemented in our application. In the Android client, the interfaces displayed to the user are the views. The controller logic is the part of application that allows interaction with the data, that is both stored locally and remotely, which represents our model. The server-side application has an important part of controller logic, in particular regarding

the storage of new data and the delivery of notifications, but also the entire model of our data.

Below we can see a visualization of the paradigm:

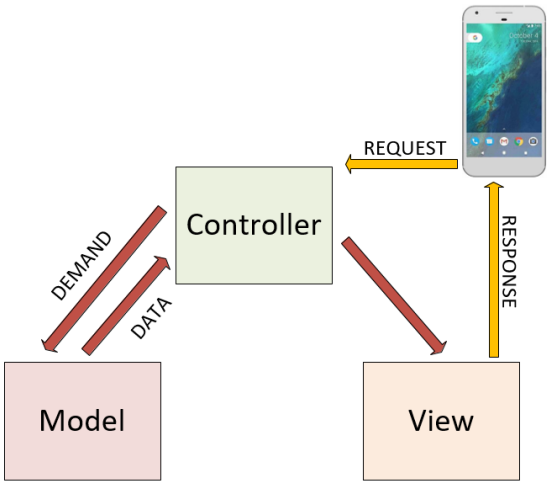

Figure 4.1: The MVC paradigm

As we said before the controller is split between client and server. For example the logic to display events is only present on the client side part as well as other classes that have the task to retrieve the data requested by the user. On the other hand, in the server part we provide the entire logic that handles the sending of notifications to the interested users based on their subscriptions and the created events

After these considerations, we can assume that our client is a not a fat nor a thin one, because both the server-side and the client-side of the application have a considerable amount of logic inside.

Finally, as said before, the model is initially present in the server-side part, because the database stores the entire data about events, subscriptions and users; however it is partially replicated in the client-side part, since we need to cache data in order to allow the application to work even if no Internet connection is available.

4.2 Architectural design

In this section we will focus on briefly describe how the two part, the server-side and the client-side, are composed.

The server side is composed by two main layers:

- **Data Layer**: contains the DBMS module and allow the data to be stored and be persistent
- **Business Layer**: encapsulates the business logic and manages the communication with the stable memory of the database, retrieving the correct informations when needed

On the client side part, we have three divisions:

- **Data Layer**: includes those classes that are used to support the caching of the data on the local storage, it interacts with the business layer

- **Business Layer**: contains different kind of classes, from the ones needed to query local database and the remote web services, to the ones that manage the application logic and support the view
- **View Layer**: provides interaction with the user. Since our is a mobile application, this layer is represented as the touch screen of the device used to visualize the application. This layer communicates with the logic level underneath that are used to communicate between the user interface and the application logic.

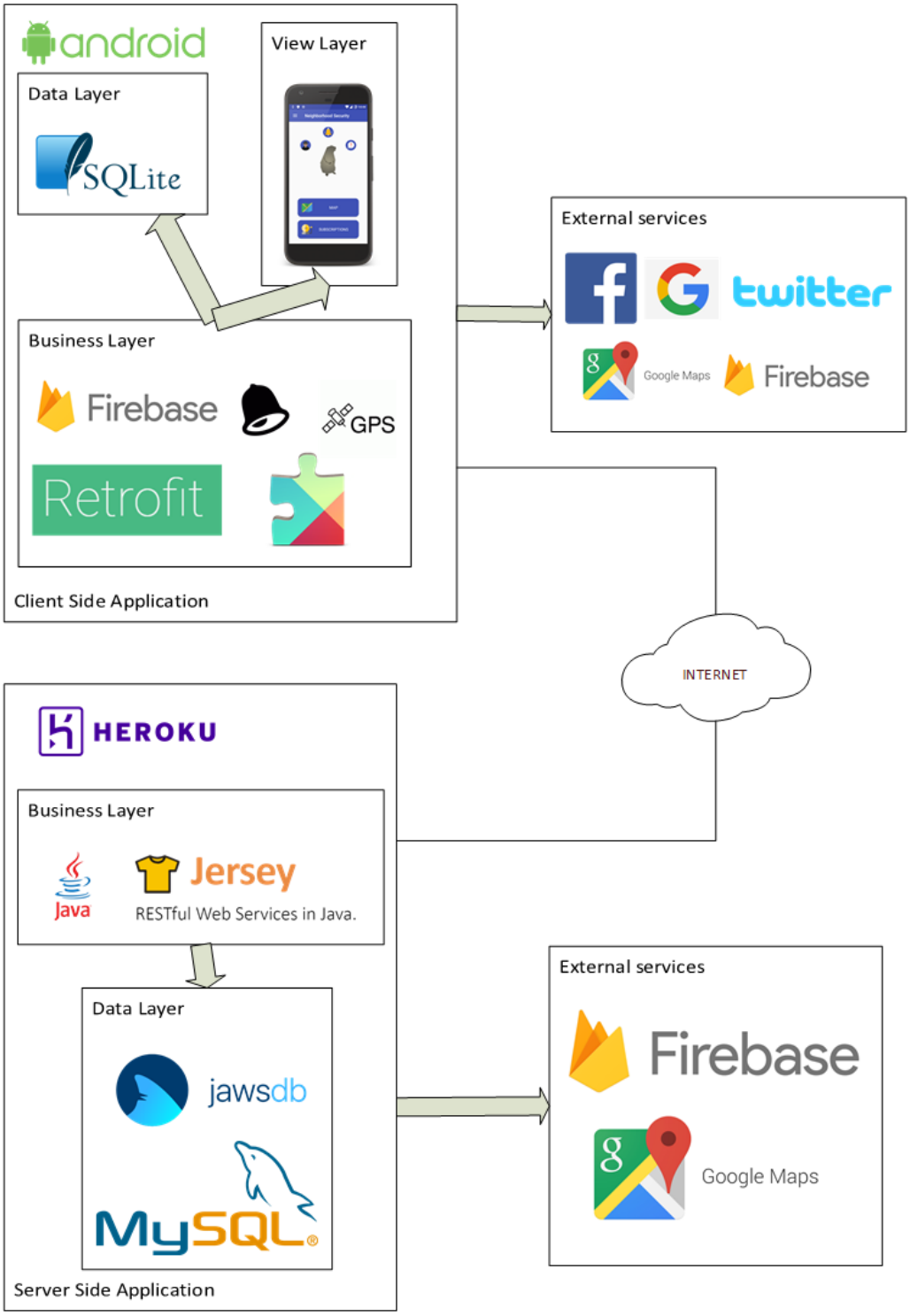

Figure 4.2: Application architecture

4.3 Java package organization

4.3.1 Application organization

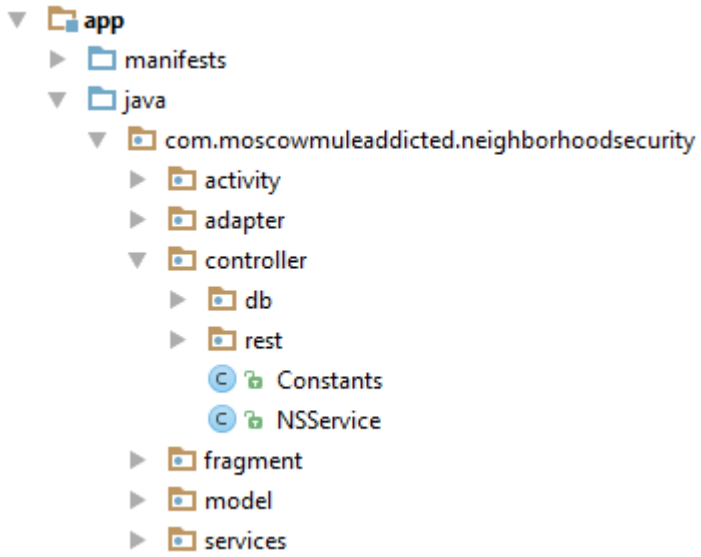

Figure 4.3: Application packages

In this picture it is possible to see the overall organization of the application packages. The division was chosen in order to separate the application components. Now we will explain what each package contains.

4.3.2 Model

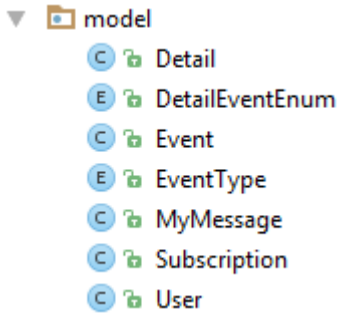

Figure 4.4: The *model* package

In this package we grouped the classes that represent the information that can be obtained from the web service and stored in the local database.

4.3.3 Activity

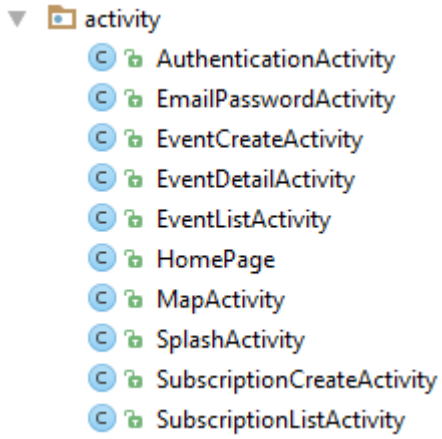

Figure 4.5: The *activity* package

This package contains all the Android *Activity* that have been implemented in the application, in most of the cases our activities are in charge of displaying the corresponding fragment and provide additional interactions to the user, such as floating buttons or action bars.

4.3.4 Fragment

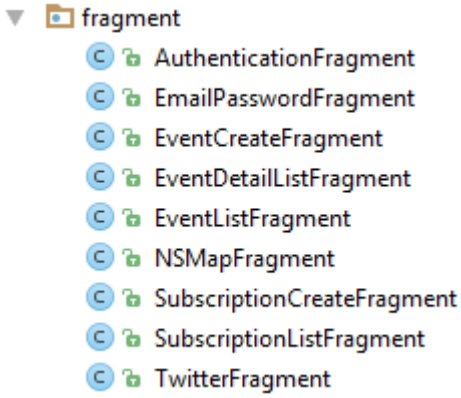

Figure 4.6: The *fragment* package

This package contains all the Android *Fragment* that have been implemented in the application. We decided to use fragments in order to be ready to reuse them when we will develop ad hoc interfaces for tablets or other devices.

4.3.5 Adapter

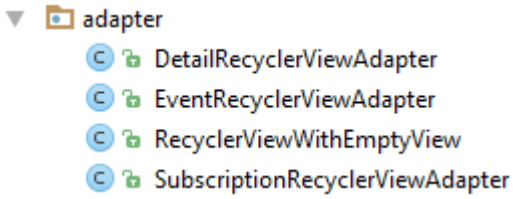

Figure 4.7: The *adapter* package

This package contains the required implementations of the *Recycler View Adapter* that is needed to display custom objects in a Recycler View. We decided to extend the default behavior of Android's Recycler View by using *RecyclerViewWithEmptyView* in order to show a default view when there are no items (events or subscriptions) in the adapter.

4.3.6 Controller

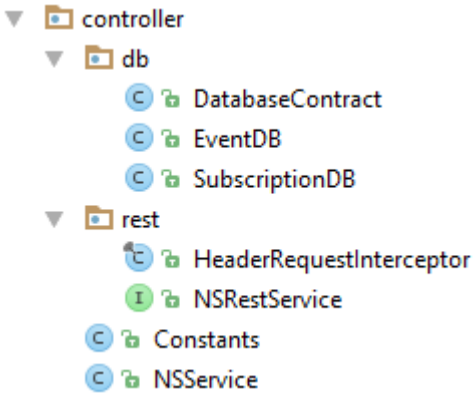

Figure 4.8: The *controller* package

This package contains all the classes that are needed to interact with the local storage, in the *db* sub-package, and with the REST web service, in the *rest* sub-package. Fragments and activities should not interact directly with those packages, but only with *NSService* that contains a set of auxiliary methods to perform specific operations that concern data or authentication. *Constants* contains a set of useful predefined values that are used across the application.

4.3.7 Services

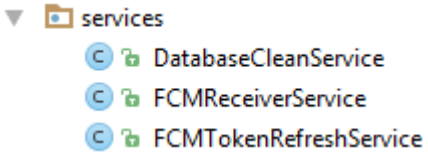

Figure 4.9: The *services* package

This package contains all the useful Android *Service* that will run in the background even when the application is not open. In order for our application to work, we needed:

- *DatabaseCleanService* to perform database clean
- *FCMReceiverService* to display notifications
- *FCMTokenRefreshService* to update the device token in the web service

4.4 Security

Very little security countermeasure must be taken in consideration in our application.

The unique way in which the DB could be compromised could be only a direct attack to our provider Heroku, but we assume that strict protection policies are taken against these type of attacks.

User authentication is handled by Google's Firebase platform, which we assume to be trustworthy and employing policies to protect their user data.

Moreover, our application does not contain sensible user data, so even if a potential attacker succeed in breaking the system, the damage will be minimum. SQL injections, that could be possible by exploiting query fields, are prevented by using prepared statements.

5 User interfaces

In this section we will provide a certain number of screenshots of the application. We focused our attention in designing the application for mobile phones, even if some portions of code are already prepared to be displayed in larger screens. Layouts, dimensions and colors were chosen following the Android Material Design guidelines.[1]

5.1 Splash screen

The splash screen welcomes the user when the application is starting, meanwhile it checks if a valid version of Google Play Services is installed on the device. If it is necessary, user is prompted to install a newer version of Google Play Services.

Figure 5.1: The splash screen

[1] Material Design Guidelines, https://material.io/guidelines/

5.2 Home

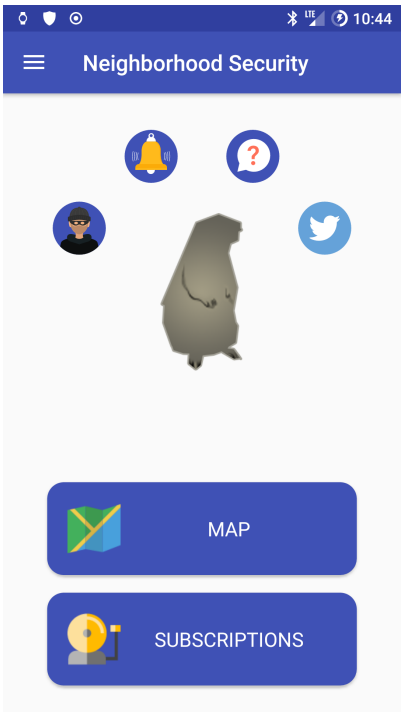

Figure 5.2: The home screen

This screen is the first the user sees as soon as the splash activity is ended. If this is the first time the user launches the application, he will see a brief showcase that introduces the main functionalities of the application. It is mainly composed by two buttons: map and subscription list. These are the two main features of the application, so they are easily accessible. The icons in the top of the screen display auxiliary informations about the applications: statistics about events and subscriptions, general and Twitter help.

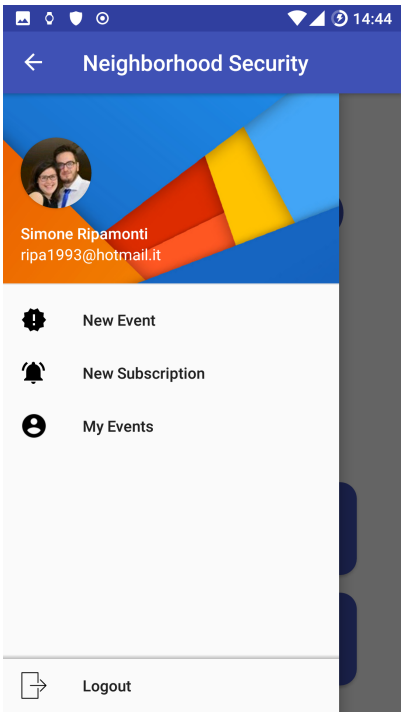

Figure 5.3: The opened drawer

Through a swipe from left to right, or by pressing the hamburger button, the user can access a lateral menu. In the menu the user gets access to other functions that are available in the applications, such as authentication and others that concern event and subscription creation or listing. This interface is really effective, since it keeps all the functionalities at a tap of distance, so the users will not lose time in searching how to reach functionalities.

5.3 Authentication

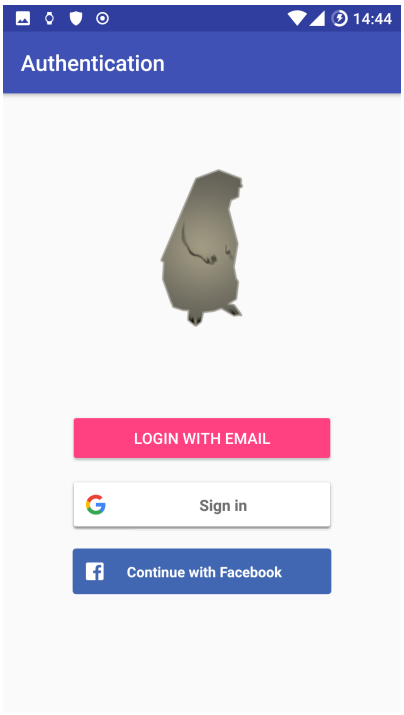

Figure 5.4: The authentication screen

The interface is simple and it just shows the three authentication methods, that are Email, Google and Facebook. By clicking on Google or Facebook buttons, the user will be prompted to choose which of his accounts he would like to use inside the application. This is the fastest way to create an account in *Neighborhood Security*, since no other fields are required to be filled.

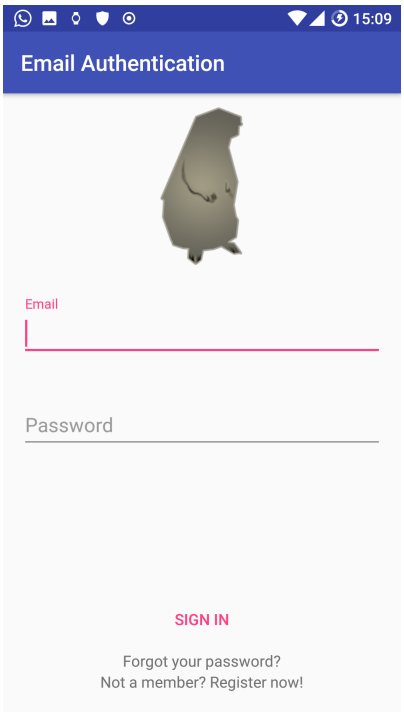

Figure 5.5: The email authentication screen

By clicking on "Login with email", a new page will open that and will let the user perform operations like registering a new account, signing in and resetting the password.

5.4 Map

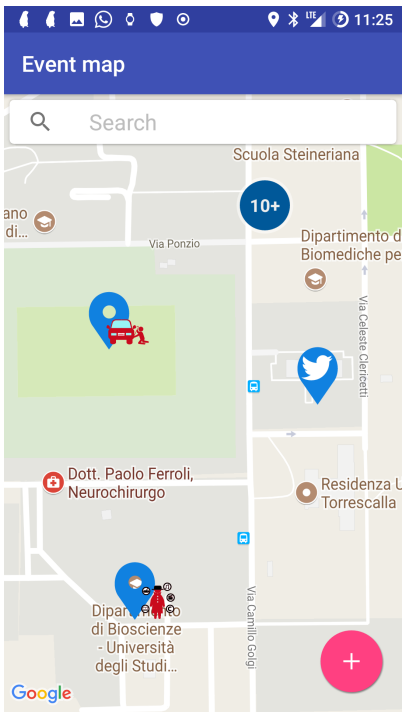

Figure 5.6: The event map screen

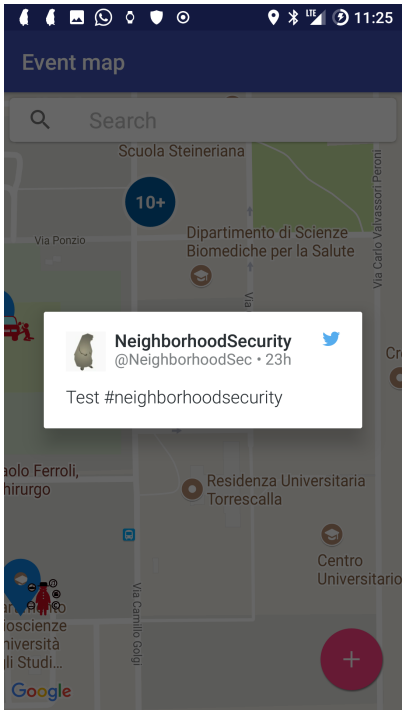

Figure 5.7: The twitter dialog fragment

The user can move and zoom in or out with the gesture he is used to (scroll and pinch-in / pinch-out). Users can also search for a location by using the search form provided by Google Place Autocomplete, that also suggests places meanwhile the user is writing. By long clicking on a point of the map, a dialog is displayed showing additional functions to the user, allowing him to create an event or subscription at that specified point or to obtain th list of events that happened near there. Also, a floating action button is displayed, it is a simple shortcut to the event create page. There are three class of markers that are displayed on the map: event, Twitter and cluster marker. An event marker is characterized by an icon that suggests the kind of event that took place in the given position. By clicking on the event, the user will move to the event's detail page. The cluster marker instead, is characterized by the number of event that have been reported in that location. By clicking on the cluster, the user will see the list of events that happened in that location.

The Twitter marker is characterized by the social network logo. When the user clicks on a Twitter marker, a *DialogFragment* is shown in order to display the Tweet content. In order to appear on the map, Tweets should use the hashtag #**NeighborhoodSecurity** and provide fine or coarse location. Fine location is pure GPS coordinates, instead coarse location is based on points of interest, such as cities or neighborhoods. By clicking on the Tweet, Twitter application, if available, or website will open in order to allow further interaction with the user.

5.5 Subscription list

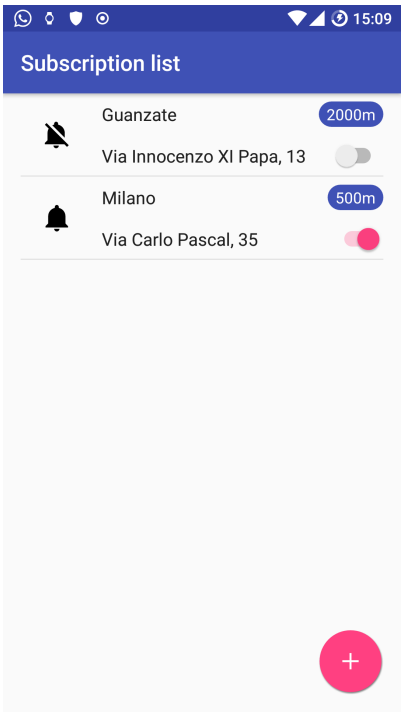

Figure 5.8: The subscription list screen

This page displays the list of active subscriptions, if any, otherwise a courtesy image is displayed, inviting the user to create new subscriptions. Each subscription displayed is characterized by it's notification status, which can be selected using the toggle on the right, and is easily understandable by the icon on the left. The main informations about the subscriptions are the location and its radius, both of them are easily readable on the screen. By clicking on a subscription, the user obtains the list of events that match that subscription. By long clicking on it, instead, the user will be asked if he wants to remove the subscription. The user can force a refresh of his subscriptions by simply doing a swipe top to bottom.

5.6 Subscription creation

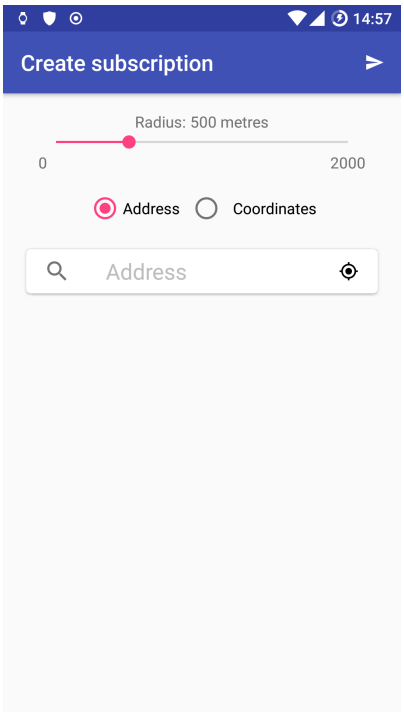

Figure 5.9: The subscription creation screen

Its simple allows the user to create a subscription in few taps. A *SeekBar* is provided in order to decide the radius of the subscription, from a minimum of 0 meters to a maximum of 2000. The user can decide the main location of the subscription in two ways, by searching for its name, for example by inserting the city and street name, or by using absolute coordinates. A button is provided to easily get the user current position, once he gave the application the permission to use location sensors. Creation is actually performed when the user clicks on the button in the top action bar.

5.7 Event list

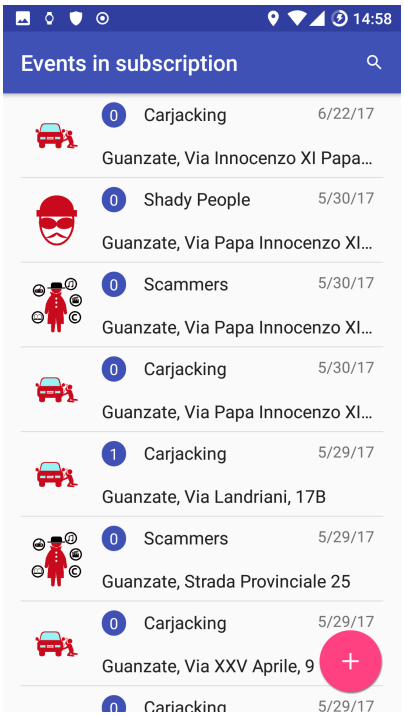

Figure 5.10: The event list screen

This page shows the list of events that can match different criteria (submitted by a user, are matching a given subscription or belong to a particular area) or are provided in a list. Each displayed event is characterized by the kind of event, both using an icon and by text, the date it took place, the location and the number of votes it has received. All these information are easily understandable at a first glance. By clicking on a event, the user will see the full details of the event. By long clicking on a event instead, the user is asked if he wants to delete the even, this is only available if the event has been created by the current logged in user. The user can force a refresh of the events by simply performing a swipe from top to bottom.

5.8 Event creation

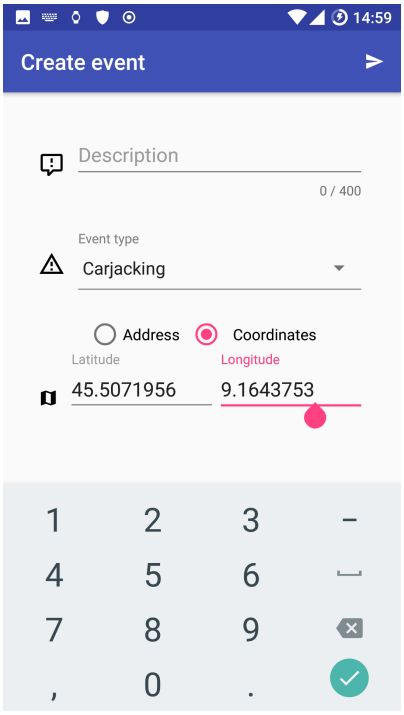

Figure 5.11: The event creation screen

This interface is very similar to the subscription create one. Both share the fact that the location can be select both by address and by coordinates. The main difference is the presence of a text field, used to add a brief description of the event, and a spinner, that allows the user to select the kind of event. Creation is actually performed when the user clicks the menu button in the action bar.

5.9 Event detail

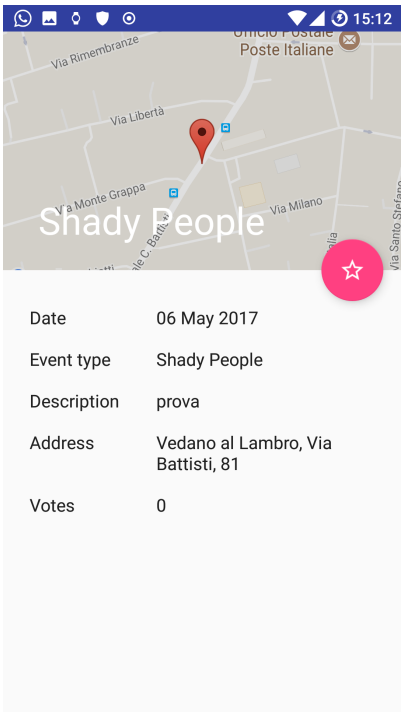

Figure 5.12: The event detail screen

In this page the user obtains the full details of an event, that are displayed on a list. These details include date, type of event, description, location and number of events. The position of the event is also displayed in the map in the top portion of the screen, this is useful to easily localize the location where it took place. The button with a star is the way the user can vote an event. By clicking on it, the vote will be added and a snack bar is displayed to undo the operation, in case the user misclicks. The star will become full once the event has been voted.

5.10 Notifications

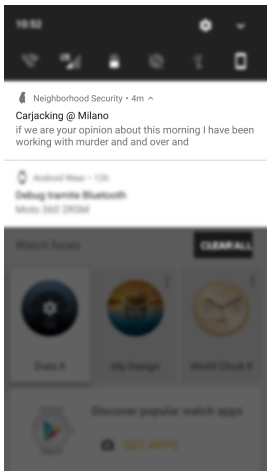

Figure 5.13: Notification on Android phone

Figure 5.14: Notification text on Android Wear

Figure 5.15: Notification map on Android Wear

Notifications are used to show the user informations about the events that he is interested in, since they belong to one of his active subscription. By clicking on phone notification, the application will open and display the event details. On the wearable device instead, the notification is composed by two pages: on the first one, the user can see the location and description of the event, on the second one, a map showing the location of the event.

6 External services and Libraries

We have used several external services for our application, some of them are necessary for the proper behavior of the system, such as Retrofit to communicate with the server-side part of the application, and other only to enrich the overall experience of the user. They are completely transparent to the user and fully integrated within the application.

The main advantage in using these kind of services is first of all the ease to not to have to implement specific portion of code in order to achieve the same result that these external services already provide.

Now we present the main external services that we have used and integrated in our application.

6.1 Retrofit[1]

Is the fundamental service that allows our application to communicate with the REST interface of our server application, and thus with the remote database. The service provides an easy way to translate a simple Java method call into a request to the web service, handling all the marshalling/unmarshalling and serialization job. The services uses annotated Java classes to transform JSON objects into Java objects.

6.2 Google Firebase[2]

6.2.1 Authentication

This service is of foundamental importance since it handles all the authentication procedures for the users. In the client-side part of the application, it is used to allow user to register or login using their preferred method among Google, Facebook or classic email and password. In the server-side of the application, it is used to verify that the user token received in the header of a request are of a valid user. It has been easily integrated in both part of the application.

6.2.2 Cloud Messaging

This service handles the sending of messages from the server to the client. These messages are triggered by the creation or deletion of events on the server-side. The messages are of two different types: event creation and event deletion, that are sent to users that have an active subscription that matches the location of the events. In this way we can optimize the bandwidth consumption of our client-side application, indeed it is not required that it periodically checks if new events are available on the DB but it is informed by the server itself, result in a huge saving of bandwidth but also of workload on the server, since we reduce the number of requests. Although an official library exists for Android, the same is not true for Java, so we needed to use a third-party open source library to allow communication between our Java EE application and FCM Api.

[1] Retrofit, http://square.github.io/retrofit/
[2] Google Firebase, https://firebase.google.com/

6.2.3 Job Dispatcher

This secondary service is used in order to keep data stored in the local database consistent with the data available remotely, in particular we are interested in removing old events/subscriptions from the caching database. A simple way is to run a scheduled job every midnight in order to remove events and subscriptions that have not been refreshed since 7 days- In this way we prevent the user from seeing events available locally that have been deleted remotely

6.3 Google Play Services[3]

6.3.1 Authentication

The service is used to obtain the Google user profile and email, that is then used by Firebase Authentication to handle new user creation or authentication.

6.3.2 Location Places

The service is used to allow user to search for locations in an easy way, since they can exploit an autocomplete feature for easier and faster location search.

6.3.3 Maps

It is a really important service, since in this way we can present the user the various event placed on a interactive map. Being able to see events placed in the map it's a really important feature of the application, since it is a quick way for the users to understand if a particular area is having particular intense criminal activity or if other users are active in the zone. A map is also displayed in the view that shows the detail of an event and also in the extended notification for Android Wear devices.

6.3.4 Fused Location Provider

This secondary service is used to obtain the device's last known position, it is useful to provide a fast way to let the user localize himself in the map, but also when creating an event or subscription at the user's position. This service requires the user to allow the application to have access to coarse or fine position. The application can continue to work even thought the user doesn't give these permissions.

6.4 Facebook[4]

The service is used to obtain the Facebook user profile and email, that is then used by Firebase Authentication to handle new user creation or authentication.

6.5 Twitter[5]

6.5.1 Core

This is the core of the Twitter service, it provides a way to interact with the REST Api of Twitter and perform searches across tweets. We used this service in order to be able

[3]Google Play Services, https://developers.google.com/android/
[4]Facebook, https://developers.facebook.com/
[5]Twitter, https://dev.twitter.com/

to display Tweets on our Event Map. The library easily allowed us to search for Tweets matching our hashtag #NeighborhoodSecurity and near a particular location.

6.5.2 TweetUi

We used this library to easily display Tweets in a fancy way, indeed it fully matches Twitter style guidelines when displaying the given Tweet.

6.6 Other minor libraries

Other libraries and services of secondary importance have been used in the application, mostly to make the development easy and quick, without the need to reimplement some basic functions.

- Glide[6]: an image loading and caching library, used to download the user profile picture and the static Google Map that is displayed on wearable devices during the notification of an event.
- Xdroid Enum Format[7]: a library that allows to easily localize enumerations by means of annotations and XML files
- UsefulViews[8]: a collection of views that follow current Google design guidelines for material design
- Floating Action Button[9]: a reimplementation of Google's floating action button that is enhanced with easier and richer customizations
- Material Drawer[10]: it's a flexible, easy to use and all in one drawer implementation that provides the easiest possible implementation of a navigation drawer
- Arc Layout[11]: a simple and customizable arc layout, we used id to display icons in the home page in a fancy way, without the need to use absolute positions
- Fancy ShowCase View[12]: a library that easily allowed us to build a welcome and introduction screen for the new users

[6] Glide, https://github.com/bumptech/glide
[7] Xdroid, https://github.com/shamanland/xdroid
[8] UsefulViews,https://github.com/FarbodSalamat-Zadeh/UsefulViews
[9] Floating Action Button,https://github.com/Scalified/fab
[10] Material Drawer, https://github.com/mikepenz/MaterialDrawer
[11] Arc Layout, https://github.com/ogaclejapan/ArcLayout
[12] Fancy ShowCase View, https://github.com/faruktoptas/FancyShowCaseView

7 UML diagrams

In this section we present some useful diagrams that helps the reader to better understand the interaction between the user and the application. Here we present some of them.

7.1 Use Case diagrams

These diagrams show the flow of operation triggered by a specific actor when it tries to perform some task. The actor can be the user (a human actor) or a system.

The use cases we propose show the main operations that are possible to perform in the system.

7.1.1 Unlogged user interaction

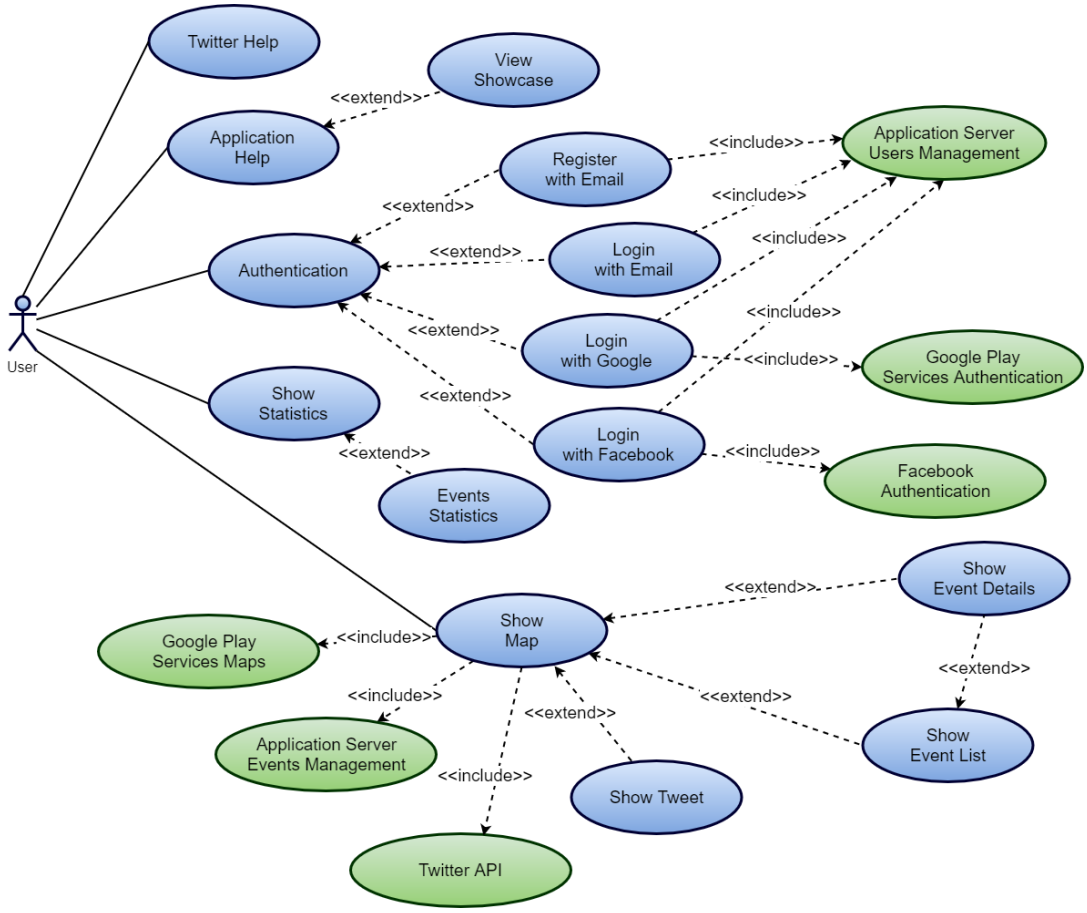

Figure 7.1: Use case: Unlogged user

7.1.2 Logged user interaction

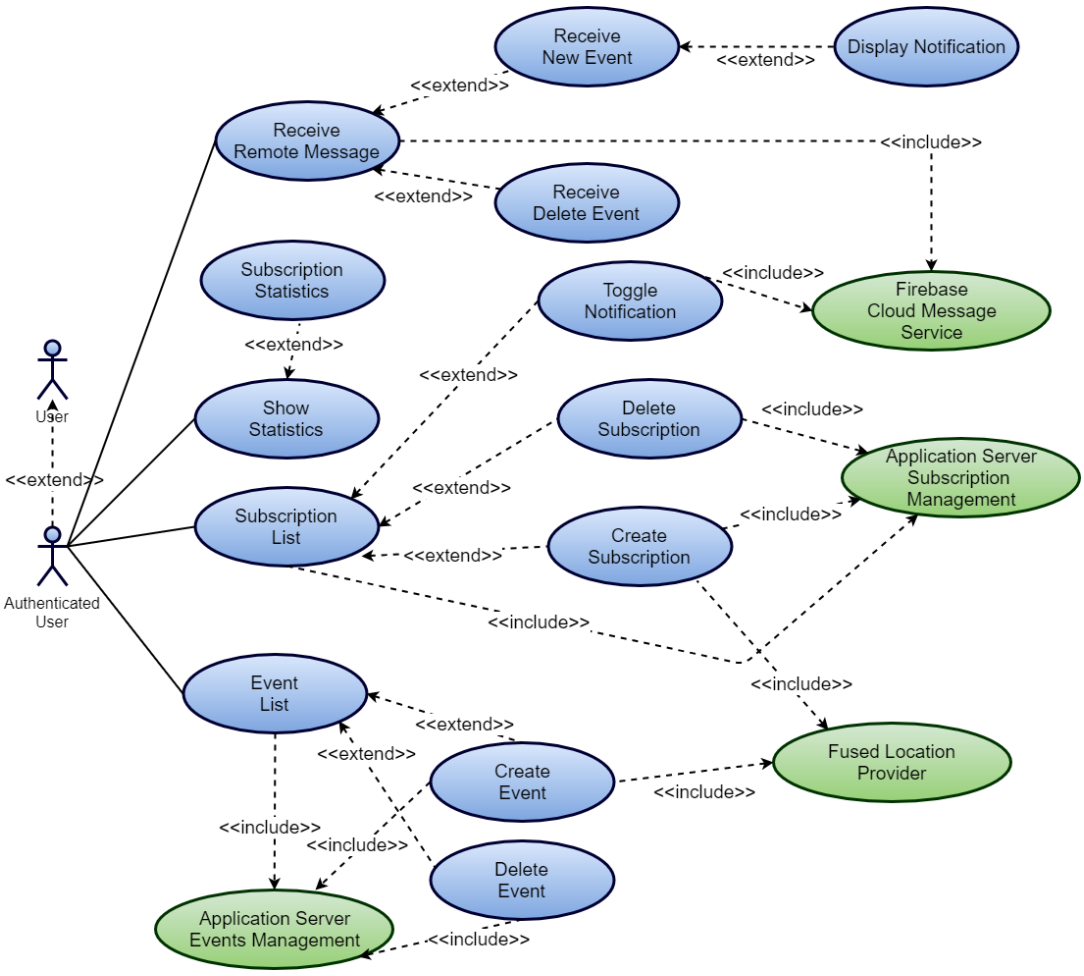

Figure 7.2: Use case: Logged user

7.2 Class diagrams

We decided to include a cumulative class diagram for our application. The structure is simple: almost all activity classes interact with their respective fragments, which may interact with the controller in order to show information to the user. The controller can interact with both the rest service and the local database.

Figure 7.3: Class diagram

7.3 Sequence diagrams

In this section we provide few sequence diagrams concerning the flow of the logical operations performed by our application once a certain operation is executed. These diagrams will show the interaction between methods and activities, rather than user and application.

We decided to include only two examples of Sequence diagrams, in order to make the document not too complex and less readable.

The two operation chosen are:

- Displaying events given an area. This diagram shows how an activity can obtain a list of events, for example with the objective of displying them on a map or on a list. The activity asks NSService to retrieve the requested data, this class acts as a controller and interacts with both the Rest service and the local storage.

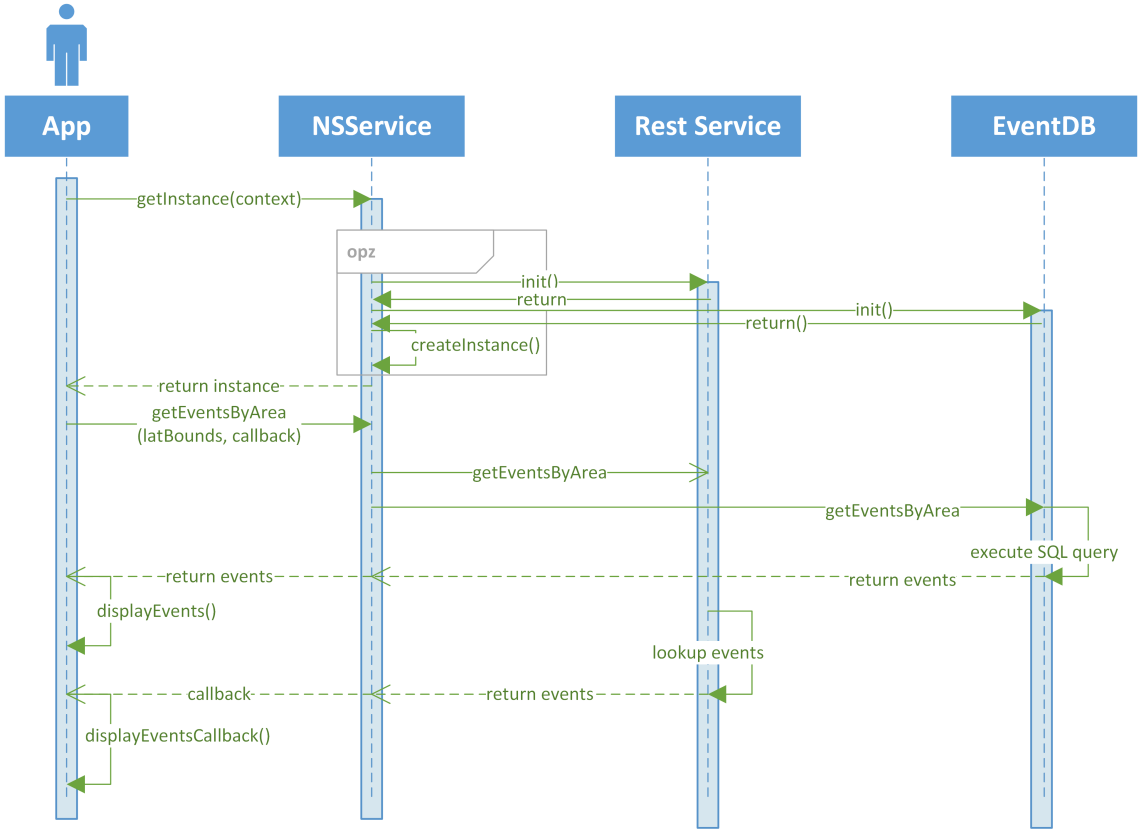

Figure 7.4: Displaying events by area sequence diagram

- Reaching Event Detail activity. This diagram shows all the possible way to reach the Event Detail activity throughout all the possible user interactions, notice that some of the ways require particular conditions, such as the user to be logged in or to have received a notification

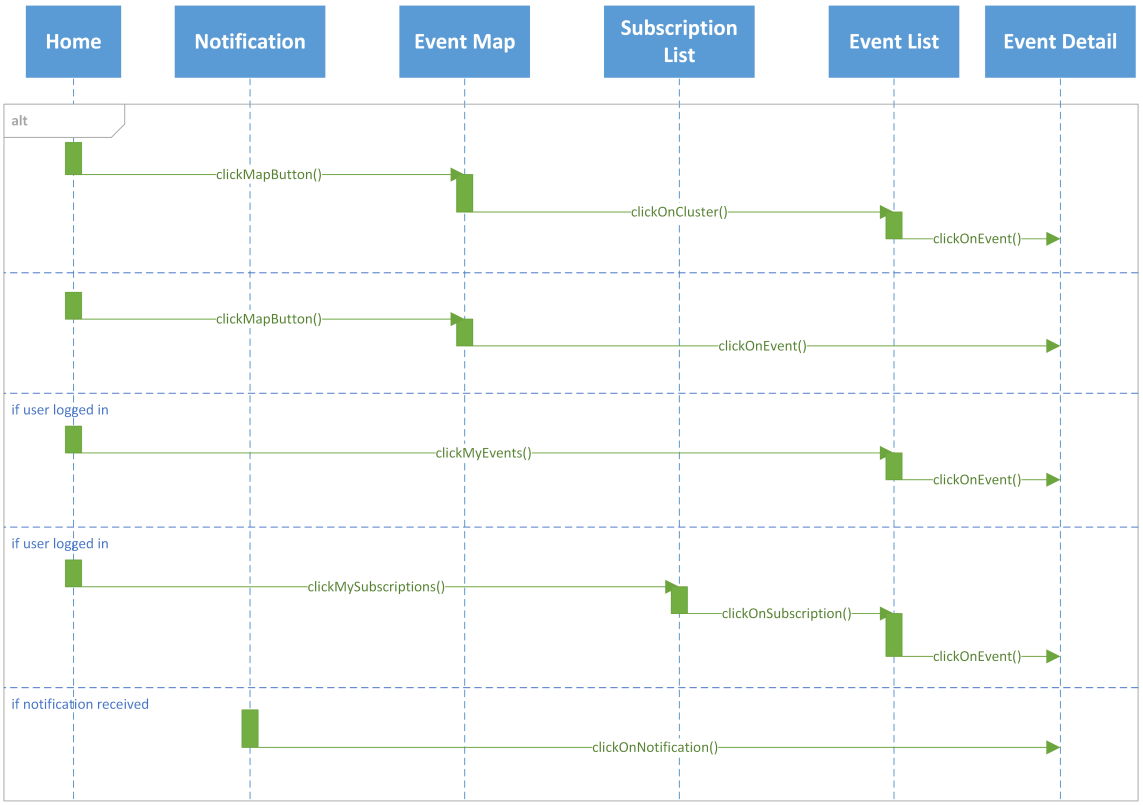

Figure 7.5: Reaching Event Detail activity sequence diagram

8 Test cases

This section describes the results of the main tests done on Neighborhood Security application.

Test Case	**Display event list**
Goal	Display event list
Input	Select a cluster marker in "Event map" activity or select "What's here" in "Event map" activity or select "My Events" in "Home" activity drawer or select a subscription in "Subscription list" activity
Expected outcome	The corresponding list of event is displayed
Actual outcome	CORRECT: after loading events from the local storage and downloading new ones from the remote service, events are displayed in a list meanwhile the new ones are stored in local storage

Test Case	**Display subscription list**
Goal	Display subscription list
Input	Select "Subscription" button in "Home" activity
Expected outcome	The list of subscription of the active user is displayed
Actual outcome	CORRECT: if the user is authneticated, after loading subscription from the local storage and downloading new ones from the remote service, subscriptions are displayed in a list meanwhile the new ones stored in local storage

Test Case	**Display event map**
Goal	Display event map
Input	Select "Map" button in "Home" activity
Expected outcome	The event map should load and display events near the user position
Actual outcome	CORRECT: if the application was granted permission to access current location, the map will load centered in the user position meanwhile events are loaded from disk, refreshed from the remote service and stored on the device.

Test Case	**Create subscription (1)**
Goal	Create subscription using Address
Input	After selecting a radius in the range 0-2000 and a location obtained using Google Place Autcomplete, press the "Submit" button in the Action Bar
Expected outcome	The subscription is created
Actual outcome	CORRECT: the parameters are valid, so if network is available and a user logged in, the subscription is created and becomes active

Test Case	**Create subscription (2)**
Goal	Create subscription using Coordinates
Input	After selecting a radius in the range 0-2000 and a pair of legal coordinates, press the "Submit" button in the Action Bar
Expected outcome	The subscription is created
Actual outcome	CORRECT: the parameters are valid, so if network is available and a user logged in, the subscription is created and becomes active

Test Case	**Create event (1)**
Goal	Create event using Address
Input	After inserting a description, an event type and a location obtained using Google Place Autcomplete, press the "Submit" button in the Action Bar
Expected outcome	The event is created
Actual outcome	CORRECT: the parameters are valid, so if network is available and a user logged in, the event is created and interested users are notified

Test Case	**Create event (2)**
Goal	Create event using Coordinates
Input	After inserting a description, an event type and a pair of legal coordinates, press the "Submit" button in the Action Bar
Expected outcome	The event is created
Actual outcome	CORRECT: the parameters are valid, so if network is available and a user logged in, the event is created and interested users are notified

Test Case	**Disable notification for a subscription**
Goal	Disable notification for a subscription
Input	Toggle the switch for the desired subscription in the user's Subscription List
Expected outcome	The switch is set on "off" and the bell icon changes
Actual outcome	CORRECT: if the subscription was previously enabled, as soon as the user moves the switch, the notification is disabled. If an event is received for that subscription, it will not be notified.

Test Case	**Enable notification for a subscription**
Goal	Enable notification for a subscription
Input	Toggle the switch for the desired subscription in the user's Subscription List
Expected outcome	The switch is set on "on" and the bell icon changes
Actual outcome	CORRECT: if the subscription was previously disabled, as soon as the user moves the switch, the notification is enabled. If an event is received for that subscription, it will be notified.

Test Case	**Delete event**
Goal	Delete event
Input	Long click on a current user's event on Event List
Expected outcome	The event is deleted and disappears from the list
Actual outcome	CORRECT: if the user is the creator of the selected event, after long clicking on the event, a popup menu will appear in order to allow the user to delete the event. Network connection is required in order to perform the operation.

Test Case	**Delete subscription**
Goal	Delete subscription
Input	Long click on a current user's subscription on Subscription List
Expected outcome	The subscription is deleted and disappears from the list
Actual outcome	CORRECT: after long clicking on the subscription, a popup menu will appear in order to allow the user to delete the subscription. Network connection is required in order to perform the operation.

Test Case	**Vote event**
Goal	Vote the event
Input	Click the star button on Event Detail screen
Expected outcome	The current user vote is added and he is notified of the operation
Actual outcome	CORRECT: if the button is not disabled, which means that the user has already voted the event, as soon as he clicks, the vote is added to the event. A snackbar is shown in order to undo the operation and the button icon changes.

Test Case	**Undo Vote event**
Goal	Undo Vote event
Input	After having voted an event, click the Undo button on the bottom bar
Expected outcome	The current user vote is removed and he is notified of the operation
Actual outcome	CORRECT: as soon as the user clicks the button, the event vote is removed and the snackbar disappears. After that the user can redo the vote.

Test Case	**Login using Facebook**
Goal	Login using Facebook
Input	Click on Facebook button in Authentication Activity
Expected outcome	The user is logged in
Actual outcome	CORRECT: after clicking on the button, the user is asked to access using his Facebook account and to grant the application permission to read profile information. If the user accepts, he is successfully authenticated in Neighborhood Security. If this is the first user access, then his informations are stored on the remote database. A network connection is required to perform the operation.

Test Case	**Login using Google**
Goal	Login using Google
Input	Click on Google button in Authentication Activity
Expected outcome	The user is logged in
Actual outcome	CORRECT: after clicking on the button, the user is asked to access using his Google account and to grant the application permission to read profile information. If the user accepts, he is successfully authenticated in Neighborhood Security. If this is the first user access, then his informations are stored on the remote database. A network connection is required to perform the operation.

Test Case	**Login using Email and Password**
Goal	Login using Email and Password
Input	Click on "Sign in" in Email Authentication activity
Expected outcome	The user is logged in
Actual outcome	CORRECT: if the user inserts a valid email and password pair, after clicking on the "Sign in" button he is authenticated. A network connection is required to perform the operation.

Test Case	**Register using Email and Password**
Goal	Register using Email and Password
Input	Click on "Register now" and then "Sign up" in Email Authentication activity
Expected outcome	The user is registered
Actual outcome	CORRECT: if the user inserts valid email, password and username, after clicking on the "Sign up" button a new account is registered. A network connection is required to perform the operation.

Test Case	**Send Reset Password email**
Goal	Send Reset Password email
Input	Click on "Forgot password" and then "Reset password" in Email Authentication activity
Expected outcome	The user receives an email with the instructions to perform password reset
Actual outcome	CORRECT: if the user inserts his email, after clicking "Reset password" button, an email will be sent to his address. A network connection is required to perform the operation.

Test Case	**Show event markers when moving map (1)**
Goal	Show event markers when moving map
Input	Move the map in Event Map activity using gestures
Expected outcome	Events matching the displayed portion of map are shown
Actual outcome	CORRECT: as soon as the user moves the map using his fingers, events available locally are shown meanwhile a search is performed on the web service, in order to search for new events to display.

Test Case	**Show event markers when moving map (2)**
Goal	Show event markers when moving map
Input	Move the map in Event Map activity using Address search
Expected outcome	Events matching the displayed portion of map are shown
Actual outcome	CORRECT: as soon as the user selects a place using Place Autocomplete search, the map is moved to center the desired location, events available locally are shown meanwhile a search is performed on the web service, in order to search for new events to display.

Test Case	**Show Tweet markers when moving map (1)**
Goal	Show tweet markers when moving map
Input	Move the map in Event Map activity using gestures
Expected outcome	Tweets matching the displayed portion of map are shown
Actual outcome	CORRECT: as soon as the user moves the map using his fingers, a search for Tweets matching the displayed location and the hashtag #NeighborhoodSecurity is performed, the resulting Tweets are displayed.

Test Case	**Show Tweet markers when moving map (2)**
Goal	Show tweet markers when moving map
Input	Move the map in Event Map activity using Address search
Expected outcome	Events matching the displayed portion of map are shown
Actual outcome	CORRECT: as soon as the user selects a place using Place Autocomplete search, the map is moved to center the desired location, a search for Tweets matching the displayed location and the hashtag #NeighborhoodSecurity is performed, the resulting Tweets are displayed.

Test Case	**Show Tweet content**
Goal	Show Tweet content
Input	Click on Tweet marker on Event Map activity
Expected outcome	The Tweet content is displayed on a dialog
Actual outcome	CORRECT: When the user clicks on the Tweet marker, a DialogFragment is shown in order to display the content of the selected Tweet. The user can further click on the dialog, in order to open the Tweet in the browser or Twitter application.

9 Cost estimation

Here we include a cost estimation for our project using the intermediate CoCoMo (Constructive Cost Model) model. This analysis has the goal to determine the number of month needed for the develop of a project of this type.

The first step of the CoCoMo approach is to choose which type of software is being produced. This will classify the model parameters *"a"* and *"b"*.

CoCoMo applies to three classes of projects:

- **ORGANIC** projects have "small" teams with "good" experience working with "less than rigid" requirements
- **SEMI-DETACHED** projects have "medium" teams with mixed experience working with a mix of rigid and less than rigid requirements
- **EMBEDDED** projects are developed within a set of "tight" constraints. It is also combination of organic and semi-detached projects

The basic equation for our model is:

$$EFFORT = EAF * a * (KLOC)^b$$

where

- *EFFORT* is in terms of man-months
- *EAF* is the effort adjustment factor
- *KLOC* is the estimated number of thousand lines of code for the project

Intermediate Cocomo	**a**	**b**
Organic	3.2	1.05
Semi-detached	3.0	1.12
Embedded	2.8	1.20

We decided to consider our project an ORGANIC one. Our team is a small one, just two people, and although this is our first time developing an Android application, we have a solid experience with Java. For this reason our coefficients are $a = 3.2$ and $b = 1.05$.

Now in order to estimate *EAF* we will use the standard provided tables, giving it a reasonable value. The value is calculated considering a set of four cost drivers: product attributes, hardware attributes, personnel attributes and project attributes.

Attributes	**Rating**	**Value**
Size of application database	Low	0.94
Complexity of the product	Low	0.85
Volatility of the virtual machine environment	Low	0.87
Applications experience	Nominal	1.00
Software engineer capability	Nominal	1.00
Programming language experience	High	0.95
Application of software engineering methods	High	0.91
Use of software tools	High	0.91
All others	Nominal	1.00

For our project we can calculate the number of person/month required:

$$EFFORT = (0.94 * 0.85 * 0.87 * 1.00 * 1.00 * 0.95 * 0.91 * 0.91 * 1.00) * 3.2 * (6)^{1.05} \approx 11.5$$

The number obtained is the number of people necessary to make the project in a month. In our case, we need 11.5 person month estimated. A manager will then schedule the number of people and time required accordingly.

However, we are a team of two people developing the application, so we can do the inverse procedure and determine th estimated time span required for our team.

We consider this equation of people needed based on duration and effort:

$$PEOPLE = EFFORT/DURATION$$

So we can calculate the inverse and obtain:

$$DURATION = EFFORT/PEOPLE = \frac{11.5\ person \cdot month}{2\ person} = 5.75\ month$$

The time of development is estimated in more or less 6 months of development.