FriendsTrackerWorldwide (FTW) – Android Application Smartphones

Project Documentation
in
Reutlingen University
Alteburgstraße 150
72762 Reutlingen

developed by
Philipp Bahnmüller
Felix Rosa
Kevin Schrötter

Advisor
Natividad Martinez

Inhalt

Abstract

This paper gives an overview of the FriendsTrackerWorldwide Android App and its functionality. It will illustrate the purpose of the app and explain the user interface, user input handling, the feedback system and activities as well as basic functions of FriendsTrackerWorldwide (FTW).

1 Planning

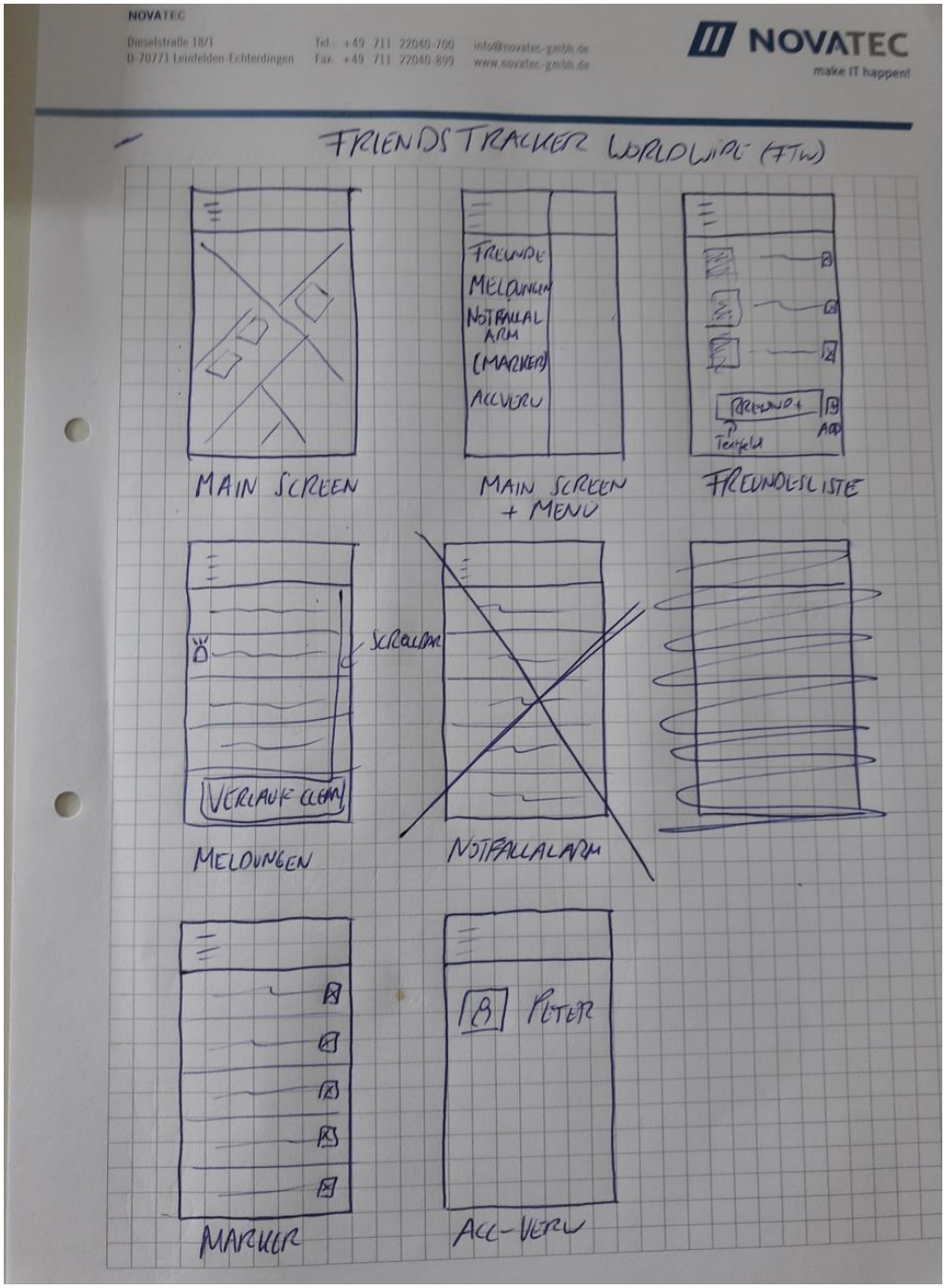
This part is supposed to mirror the thought processes of the developers.

1.1 First Thoughts

Those were the initial thoughts that came to our minds when thinking about a possible app that would suit the title “FriendsTrackerWorldwide”.

1. We want to make an application that draws positions of a user, his friends and different kinds of markers on a map.
2. The map could either be from Google Maps, Bing Maps or OpenStreetMap.
3. The App should save a user with a username and his GPS data in an external database.
4. Smartphones send their GPS data, which is used for drawing a BoundingBox of a map and display positions
5. Grouping functionality of friends/foes
6. Add and delete of custom markers

1.2 Scetch of possible Activities



Main Screen displaying a map

Main Screen with menu for selecting operations

Friendlist Activity for displaying friends as well as managing friends

Notifications Activity to get information about markers and distances

Marker Activity to list all markers

Account Settings to change user settings

1.3 Choosing a Maps API

We got three different maps APIS to choose from.

1.3.1 Google Maps API

Google API is free until it gets up to 2500 requests a day. This could have been an option, but the risk of probably getting attacked to push the requests above 2500 requests per day is still there. So we decided to not take Google Maps API.

1.3.2 Bing Maps API

Not free of charge.

1.3.3 OpenStreetMaps API

Completely free to use, this was the main reason why we decided to take OpenStreetMap.

1.4 Choosing a database

The possibilities were between an SQLite database within the application or an external database. Since we want to communicate through the internet, we decided to take an external database.

Since we all got used to working with a flexible document based database called MongoDB, we quickly decided to use it as fundamental for the Project.

1.5 Geo-coordinates

This part handles different geo coordinate related issues.

1.5.1 Notation

For the notation of longitude and latitude pairs there are three ways:

- **decimal degree**
  This type of notation is mostly used for coordinates on maps like for example Google Maps, Bing or OpenStreetMap.
  The advantage of this notation is, that calculations within the coordinates are easier and more efficient than the other notations.
  The precision of this notation is depends on the amount of decimal places. The more you have, the more precise you are!
  For instance: 2 decimal places get a precision which can vary to up to 1 km. 4 decimal places can be precise to up to a 10m radius and 6 decimal places get a 1m radius.

- **degree, decimal minutes**
  This notation is also called the nautic notation. It is mostly used for nautic, seafaring and aviation, because those areas need quite precise coordinates.

- **degree, minutes, seconds**
  This is also called the historic notation, sexagesimal. It has the most precise location determination, but is harder to calculate.

1.5.2 Conversion

All three notation types can be converted to one another.

As an example I will use the following coordinates as reference, written in all three notation styles:

1. Decimal degrees: 48.5115599°N 9.0783088°E
2. degree, decimal minutes: N48° 30.6936 E9° 4.69853
3. degrees, minutes, seconds: N48° 30‘ 41.616“ E9° 4‘ 41.912“

Keep in mind, that the calculation is only done for the latitude in the example. For the complete coordinates, you have to calculate the same process for the longitude as well!

- **Calculation decimal degree to degree, minutes, seconds**
  48.5115599°N – decimal degree notation
  DEGREE is the value from the left side of the notation → 48

  (48.5115599 – DEGREE) * 60
  = (48.5115599 – 48) * 60
  = 0,5115599 * 60
  = 30. 693594 → 30 minutes

  (30.693594 – MINUTES) * 60
  = (30.693594 – 30) * 60
  = 0.693594 * 60 = 41.61564 → 41.616 seconds

  In total the solution is: N48° 30‘ 41,616“ as latitude. North is used for positive numbers, South for negative ones.
  For the latitude positive numbers would be East, negative numbers West

- **Calculation decimal degree to degree, decimal minutes**
  48.5115599°N – decimal degree notation
  DEGREE is the value from the left side of the notation → 48

  (48,5115599 – DEGREE) * 60
  = (48,5115599 – 48)*60
  = 0,5115599 * 60
  = 30,3693594 → 30,6936 decimal minutes

  In total the solution is: N48° 30.6936

- **Calculation degree, minutes, seconds to decimal degree**
  N48° 30‘ 41.616“ - degree, minutes, seconds notation
  DEGREE is the degree value → 48
  MINUTES is the minutes value → 30
  SECONDS is the seconds value → 41.616

  (((SECONDS / 60) + MINUTES) / 60) + DEGREE
  = (((41.616 / 60) + 30) / 60) + 48
  =(0.6936 + 30) / 60 + 48
  = 30.6936 / 60 * 48
  = 48.5115599°N

  In total the solution is: 48.5115599°N

- **Calculation degree, minutes, seconds to degree, decimal seconds**
  N48° 30‘ 41.616“ - degree, minutes, seconds notation
  DEGREE is the degree value → 48
  MINUTES is the minutes value → 30
  SECONDS is the seconds value → 41.616

  For this conversion, you keep the degree (48) as it is and only calculate a value for the decimal minutes:

  (SECONDS / 60) + MINUTES
  = (41.616 / 60) + 30
  = 0.6936 + 30
  = 30.6936

  In total the solution is: N48° 30.6936

- **Calculation degree, decimal minutes to decimal degree**
  N48° 30.6936 - degree, decimal minutes notation
  DEGREE is the degree value → 48
  MINUTES is the minutes value → 30.6936

  MINUTES / 60 + DEGREE
  = 30.6936 / 60 + 48
  = 48.51156°N

  In total the solution is: 48.51156°N

- **Calculation degree, decimal minutes to degree, minutes, seconds**
  N48° 30.6936 - degree, decimal minutes notation

DEGREE is the degree value → 48
MINUTES is the minutes value → 30.6936

For this conversion, you keep the degree 48. The value for the minutes is the value on the left side from the comma of the decimal minutes (30). The calculation of the seconds is the following:

MINUTES – MINUTES LEFT SIDE OF COMMA * 60
= 30.6936 – 30 * 60
= 41.616 seconds

In total the solution is: N48° 30‘ 41.616‘‘

1.5.3 Distance-Calculation

When it comes to the calculation of distances between two coordinates, you can only use the decimal degree notation.

As an example, I will use the coordinates of Frankfurt and Berlin and calculate the distance in km between the two locations.

Google Maps delivers coordinates in decimal degree notation.
To get them, you can simply open Google Maps, search for some location, right click it and then click on “what is here?”. Then you can see the coordinates on the bottom of the website in a small box.

Bing Maps delivers coordinates in decimal degree notation as well.
To get them, you open bing Maps, type in any city and just right click in somewhere in the map. A small menu will pop up that shows you the coordinates.

In Open Street Maps there is not an easy way to display the coordinates. But still, they are also saves in decimal degree notation and accessible through an API from OpenStreetMap

Latitude Frankfurt (lat1): 50.11222°
Longitude Frankfurt (lon1): 08.68194°

Latitude Berlin (lat2): 52.52222°
Longitude Berlin (lon2): 13.29750°

The calculation itself uses the so called arcus-cosinus, which is the inverse of the cosinus.

However, there is an important issue to consider when calculating geo-location-distances: Whether to use the RAD system or the normal angle system.

If the system supports angles, you can use this schema:

calculation using angle

e = ARCCOS [SIN(lat1) * SIN(lat2) + COS(lat1) * COS(lat2) *COS(lon2-lon1)]

e = ARCCOS[SIN(50,11222) * SIN(52.52222) + COS(50.11222) * COS(52.52222) * COS(13.2970-8.68194)]

e= ARCCOS [0.60892 + 0.38893]

e= 3.75781 / 180 * PI

e = **0.06559**

At the end, multiply the result e = **0.06559** by the radius of the equator r = **6378.137 km**
Distance = e * r
Distance = **0.06559** * **6378.137 km**
Distance = **418.34 km**

calculation using RAD

When using RAD, you need to convert the normal coordinates to decimal numbers first.

Latitude Frankfurt (lat1): 50.11222° / 180 * PI = 0.87462
Longitude Frankfurt (lon1): 08.68194° / 180 * PI = 0.15153

Latitude Berlin (lat2): 52.52222° / 180 * PI = 0.91669
Longitude Berlin (lon2): 13.29750° / 180 * PI = 0.23209

e = ARCCOS [SIN(lat1) * SIN(lat2) + COS(lat1) * COS(lat2) * COS(lon2-lon1)]

e = ARCCOS [SIN(0.87462) * SIN(0.91669) + COS(0.87462) * COS(0.91669) * COS(0.23209-0.15153)]

e = ARCCOS [0.60892 + 0.38893]

e = **0.06559**

At the end, multiply the result e = **0.06559** by the radius of the equator r = **6378.137 km**
Distance = e * r
Distance = **0.06559** * **6378.137 km**
Distance = **418.34 km**

1.6 Requirements for the Application

- Display own position on the map
- Display own markers on the map
- Friend management with a friendlist
- Display positions of friends
- Display markers of friends
- Calculate distance between own position and everything else
- Send a notification when someone gets into a defined radius
- Send emergency messages to all friends
- Delete notifications from friends
- Save location information in a central database
- Save user in a central database
- Save marker in a central database
- Add additional information to markers like a description
- When markers get new information, the old variables should be overwritten
- Friendrequests with confirm/deny schema
- Identification and login to app via MAC-Adress of the smartphone
- App shall be reachable from everywhere
- Completely FREE OF CHARGE application

1.7 Task prioritization

1. Create app Layout (Frontend)
2. Implement personal GPS functionality including database functionality
3. Implement friend list with interaction and management
4. Display markers
5. Interact with markers
6. Show notifications
7. Change account settings

2 Development

For the development we devided specific tasks for each member of the project group:

2.1 Task Distribution

Philipp Bahmüller:

1. Evaluating and choosing Maps API
2. Prioritize tasks
3. Implement geo calculation functionality
4. Implement markers and marker interaction
5. Create Presentation

Felix Rosa:

1. Scetches and rough design
2. Evaluate and choose database
3. Specify layout
4. Specify requirements
5. Implement basics of classes Person, Marker and GPS-Position
6. Create frontend of MainActivity and Menu
7. Create Poster

Kevin Schrötter:

1. Research calculation of geo locations and distances between them
2. Manage git repository
3. Manage database connection and implement FriendsTrackerWorldwide API
4. Implement friendlist and friendlist management
5. Write Documentation

2.2 Actual development process

Tasks are shown in chronological order

1. Create project and implement OpenStreetMap View
2. Get mobile device geo data
3. Draw own position in map
4. Update own position within a defined time interval
5. Implement API calls using a selfwritten ApiCaller Java Class
6. Update own position locally
7. Get friend positions using the API call
8. Handle friend positions and draw the friends positions on the map
9. Create a new activity for login and registration

10. Handle login and registration requests using API calls
11. Create new activity for displaying friends in a list
12. Write a new Java Class for a custom list adapter for adding friends and a delete button in the friendlist
13. Manage friendlist update when deleting a friend
14. Add a new activity for adding friends
15. Handle communication between MainActivity, FriendActivity and AddFriendActivity so that the AddFriendActivity can send a result back to the MainActivity for getting some kind of reaction
16. Implement a method for sending notifications whenever a marker is in a given radius of the users position
17. Create a new activity for displaying the “in range” markers in a list
18. Implement another custom ArrrayListAdapter for displaying the markers together with a clickable button to center the map on that chosen marker
19. Handling onClick-Events from within an Adapter to have an effect in the MainActivity
20. Implement displaying own markers and friend markers that have been created in the database before
21. Implement a method for adding new markers on the map by long tapping
22. Implement a method for removing markers by long tabbing
23. Changing the method of displaying markers in a radius to displaying all markers, but with an additional distance information
24. Sort markers by distance
25. Implementing different algorithms for sorting
26. Adding a new activity for showing a legend
27. Complete bugfixing and test app on multiple smartphones

2.3 Problems and Solutions

There were a few harder to solve problems while developing FriendsTrackerWorldwide:

2.3.1 Getting geo location information of the mobile device

There was a problem according to getting geolocations of a smartphone. There is no functionality in Android Studio to get the real current location, only a way to get the last known location and update location in a time interval.

We solved this problem by showing a loading screen at the initialization process of the app until it got a location update from the device.

2.3.2 Requesting permissions for accessing geo data and internet

A problem occurred when we tried to run the app on different Build versions of android. To access geo location information, an application using build version 22 or below would not want to ask a specific permission. However, when we were trying the same code on our own mobiles using Android Build Versions above 23 we could not get the app running because it would throw an error because of not granting permissions. There is a way to check which version is used, but it still did not work, so we decided to set Build Version 23 or higher (starting from Android 6.0, Marshmallow) as a requirement for the app to avoid the problem. With that solution we could normally ask for the permissions when installing the app and it runs as intended.

2.3.3 Sending variables from Activity A to C, passing through B

When we got to the point where we wanted to create a clickable button when displaying all markers within range, we got the problem of not being able to send the needed information from the MainActivity over the NotifyActivity up to the custom Adapter of the Notifications for handling a onClickEvent when clicked in the ListView. To solve this problem, we could not only try to send information via an intent, but we had to create an interface within the custom Adapter, so that the NotificationsActivity would call a function regarding a string of the custom Adapter whenever something is clicked and then sending the information back to the MainActivity. The interface written for that problem is the IPassString interface in the code.

2.3.4 Adding a marker with long tap

There was no way to add a marker by tapping using the simple openstreetmap marker overlay. To solve this problem, we had to add an additional overlay to the mapView underneath the original one. This then enabled the possibility to interact with the map itself, and therefore we could overwrite the onLongTap method of the new overlay to get latitude and longitude of the tapped area.

2.3.5 Removing a marker with long tap

Although the additional map overlay does provide getting geo locations out of a map, it still does not function as a long tab on a marker itself. A workaround for this issue was to add a dragListener to each marker, that should be able to be long tapped. This is not a real long tap, but the method “onDragStart” of a dragListener acts like it would be a long click, since a marker itself does not have a long Tap method.

2.3.6 Removing a marker using id

Because friends markers to not come with their object id used in MongoDB and markers to not have an additional attribute for an id, we had to come up with an idea of how we can delete our own markers. The solution was to not create a marker object, but to create a new class SpecialMarker that extends the classic Marker class. With that new class we could simply add a new attribute for saving the id delivered from getMyMarkers API call.

2.3.7 Sorting listViews using Double.compareTo

Somehow comparing doubles with each other worked well when running the app on the local emulated devices. However, when trying out the application on our own smartphones, the app would suddenly crash without any information on why it crashed. We tried out different kinds of mechanisms for sorting an ArrayList by double values, even using a quicksort. It always worked well on the emulated device, but still crashed on real smartphones. We could not figure out what the problem was but still wanted to implement the feature of sorting the markers by distance. The solution was rather embarrassing, because it took a very long time to realize a tiny difference between the displayed doubles in the emulator and the real smartphone: the emulator used a dot as delimiter, the mobile device used a comma as delimiter. So apparently the emulator does not make a difference between dot or comma, but the mobile device does. A simple string replace, replacing the comma with a dot in the code fixed the problem completely.

2.3.8 Overall Android Studio Problem anecdote

From time to time Android Studio refuses to build new debug_apk files. This slows down the development a lot. A simple solution would be to just delete the existing file to enable Android Studio to build a new one or to run the project clean from within the Studio. Unfortunately when this error occurs, android Studio somehow puts an administrator restriction on the file so that it can't be deleted, even when logged in as administrator in windows.

To solve this problem it is necessary to completely close Android Studio, the Emulators and every folder that belongs to the project. After that, a hard delete using shift + delete most of the times deletes the debug_apk file (or better the whole build folder). But sometimes even this workaround does not work and a computer restart is needed before deleting the files.

We figured out that the problem occurs when manually building a debug_apk file and not closing the folder afterwards. So keep an eye on what is opened and what not.

3 FriendsTackerWorldwide API

The FriendsTrackerWorldwide API is an API which provides the communication between the FriendsTrackerWorldwide Mobile APP (or any other kind of frontend) and an external database from MongoDB. The API delivers the main functions for adding registration, login, creating users and marker objects as well as calculating distances between a specific users location and all other GPS related objects that are connected to that user. The results typically are in a JSON format sent to the frontend and can then be used. In the FriendsTrackerWorldwide Project overall, the FriendsTrackerWorldwide Mobile App uses the JSON data retrieved from the FriendsTrackerWorldwide API to draw a map using elements and style from OpenStreetMap (using osmdroid together with osmbonuspack) and for managing a friendlist within the application.

3.1 Accessing the API

The API is accessible through https://friendstrackerworldwide-api.mybluemix.net/api

You can download the FTW API Server source code and run it on your own server.

Make sure to have node.js installed on the computer and run the “npm install” command from the shell before starting the node server using “npm start”.

3.2 Downloading the API

To download the API and run it on your own server, simply visit https://github.com/KevinSchroetter/FriendsTrackerWorldwide_API and download the repository.

3.3 MongoDB Database

The API uses a specific mongoDB database that. However, you can simply add another mongoDB database by creating a new external database on https://mlab.com/ and changing the db.connect(url,function(){}) in the server.js file of the ftwapi folder so that it matches the mongoDB link of the new database. MLab will then automatically create all required Collections.

3.4 FTW API Collections

The API uses two collections. Both are used for drawing objects in an OpenStreetMap:

- User
    - Used for saving user information, user GPS-locations and friend list-management
- Marker
    - Used for saving user-generated GPS-Locations with a description and a creator name

3.4.1 Collection Models

In the past, node.js could use mongoose for establishing a connection to the mongoDB database which also added the possibility to generate a mongoose.Schema for collection. By now mongoose is deprecated and it is better to use MongoClient instead. However, mongo client can no longer define Schemas as mongoose used to do. The following display of the collections uses old mongoose to see how the structure should be to work properly in the API:

3.4.1.1 user

```
{
   "_id": {
      "$oid": AUTOGENERATED,
   },
   "username": {Type: String},
   "password": {Type: String (hashed Value!) },
   "geoLocation": {
      "latitude": {Type: Number},
      "longitude": {Type: Number}
   },
   "friends": {Type: String[ ] },
   “openRequests”: {Type: String[]},
   “sentRequestts”: {Type: String[]},
   "description": {Type: String}"
}
```

3.4.1.2 marker

```
{
   "_id": {
      "$oid": AUTOGENERATED
   },
   "owner": {Type: String},
   "geoLocation": {
      "latitude": {Type: Number},
      "longitude": {Type: Number}
   },
   "description": {Type: String}
}
```

3.5 required Node Modules

Here you can see which node_modules are required for correct functionality of the API as well as a short description of what they are used for.

3.5.1 body-parser

- Version 1.17.2
- Used for working with parameters

3.5.2 express

- Version 4.15.3
- Used mainly for the API itself

3.5.3 bcrypt (not used in final V. due to bluemix problems)

- Version 1.0.2
- Used for hashing and saving hashed passwords in Database

3.5.4 Geopoint (not used in final V. due to calculating on APP instead)

- Version 1.0.1
- Used for calculating distances between different GPS-locations

3.5.5 Mongodb

- Version 2.2.28
- Used for connecting to mongodb database

3.6 FTW API Routes

The API uses several routes for access. A server ip address and a port are required for correct functionality. In the following routes, the server address and port are replaced by “…”.

3.6.1 …/api/user/test

Methode	Parameters	Type	Return Value
GET	-	-	„message“ String

3.6.2 …/api/user/getFriends/:username

Methode	Parameters	Type	Return Value
GET	username	String	Nameless Array containing friend names OR „err“ String

3.6.3 …/api/user/getFriendsLocation/:username

Methode	Parameters	Type	Return Value
GET	username	String	„friendsLocation“ Array OR “err” String

3.6.4 …/api/user/getMyLocation/:username

Methode	Parameters	Type	Return Value
GET	username	String	„myLocation“ Array OR “err” String

3.6.5 …/api/user/getRequests/:username

Methode	Parameters	Type	Return Value
GET	username	String	„sentRequests“ Array AND “openRequests” Array OR “err” String

3.6.6 …/api/user/loginUser

Methode	Parameters	Type	Return Value
POST	username password	String String	„success“ String OR “err” String

3.6.7 …/api/user/addUser

Methode	Parameters	Type	Return Value
POST	username password latitude longitude	String String Double Double	„message“ String OR “err” String

3.6.8 …/api/user/addFriend

Methode	Parameters	Type	Return Value
PUT	username friend	String String	„message“ String OR “err” String

3.6.9 …/api/user/updateUser

Methode	Parameters	Type	Return Value
PUT	username latitude longitude	String Double Double	„message“ String OR “err” String

3.6.10 …/api/user/confirmFriendRequest

Methode	Parameters	Type	Return Value
PUT	username friend	String String	„message“ String OR “err” String

3.6.11 …/api/user/deleteFriend

Methode	Parameters	Type	Return Value
DELETE	username friend	String String	„message“ String OR “err” String

3.6.12 …/api/user/denyFriendRequest

Methode	Parameters	Type	Return Value
DELETE	username friend	String String	„message“ String OR “err” String

3.6.13 …/api/marker/test

Methode	Parameters	Type	Return Value
GET	-	-	„message“ String

3.6.14 …/api/marker/getMyMarker/:username

Methode	Parameters	Type	Return Value
GET	username	String	„myMarker“ Array OR “err” String

3.6.15 …/api/marker/getFriendsMarker/:username

Methode	Parameters	Type	Return Value
GET	username	String	„friendsMarker“ Array OR “err” String

3.6.16 …/api/marker/addMarker

Methode	Parameters	Type	Return Value
POST	owner latitude longitude description	String Double Double String	„message“ String OR “err” String

3.6.17 …/api/marker/deleteMarker

Methode	Parameters	Type	Return Value
DELETE	id	String	„message” String

4 FriendsTrackerWorldwide Application

FTW is designed to use a smartphones GPS-location to sent it to a MongoDB database and draw the users‘ position in a MapView from osmdroid, an OpenStreetMap tool for android. The user can then add and remove his own Markers on the map provided by a description that he can write himself to describe the marker.

A friend management system using friendrequests that need to be confirmed by the counterpart is included, which supports adding other users of the app to the account to see the friends positions together with their markers, if they decided to add some to their account.

Furthermore a user is able to display a list of every marker on the map, except his own position. The items of the list are sorted by their distance to the users’ own position. When clicking on an item of the list, the map will center the selected marker so that it is possible to directly see where its location is.

This enables to see all markers and their distance to users own position, so that it is possible to track those locations.

4.1 Technical Requirements

There are a few things to consider when running the app.

4.1.1 Device

FTW is designed for smartphones that have an internet connection and GPS enabled

4.1.2 Operating System and Version

FTW requires an Android operating system.

The Minimum Build Version is:

- Android Build Version 23
- Android 6.0 (Marshmallow)

4.1.3 Permissions

To run the app properly, you need to allow internet access and GPS-location when installing the application. In detail:

- NETWORK_STATE
- WIFI_STATE
- COARSE_LOCATION
- INTERNET
- WRITE_EXTERNAL_STORAGE

4.2 Activities

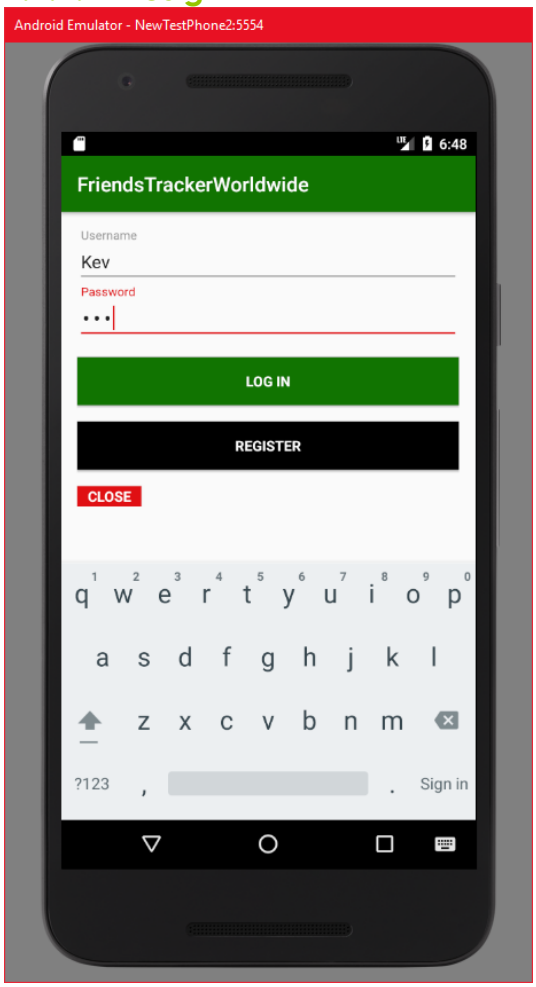
FriendsTrackerWorldwide uses 7 activites for its computing.

- LoginActivity
- MainActivity
- FriendActivity
- AddFriendActivity
- NotificationActivity
- RequestActivity
- LegendActivity

4.2.1 LoginActivity

This activity is used for user registration and user login.

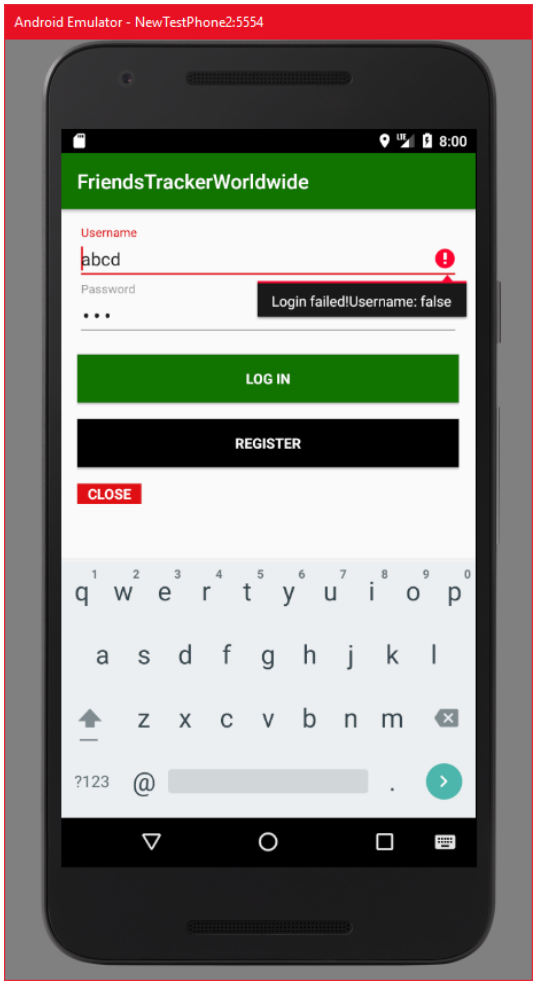
4.2.1.1 Design



4.2.1.2 Interface interaction

4.2.1.2.1 Username field and password field

These fields will show error messages, when the entered input does not match the restrictions or the registration/login attempts fail:



4.2.1.2.2 "LOG IN" button

By clicking on the login button, the app will start the login process calling ServerAdress + api/user/loginUser.

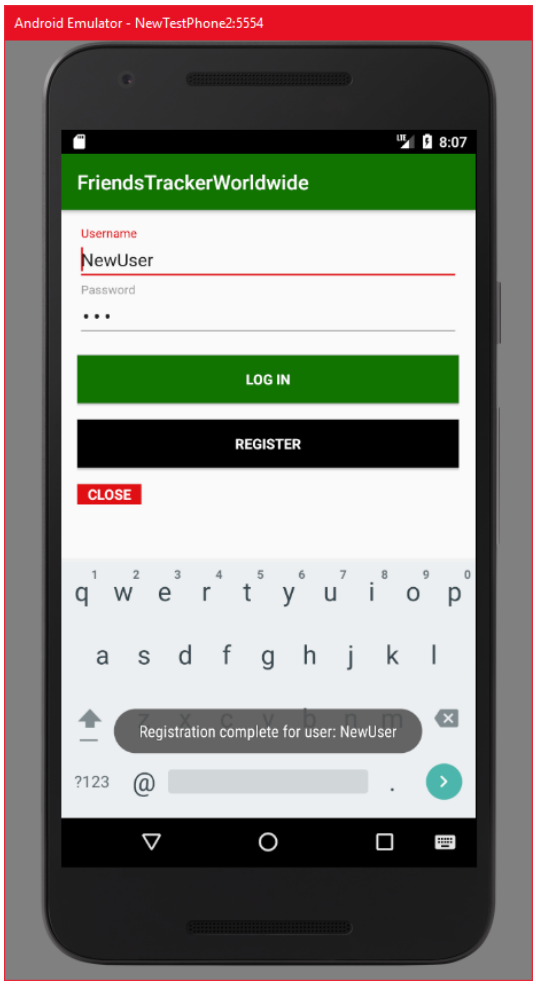
If the attempt is successful, the MainActivity is called and the initialization process starts.

If the login attempt fails, an error message as shown in 3.1.2.1 will be displayed.

4.2.1.2.3 "REGISTER" button

Clicking on register will attempt a registration in the database for the user calling ServerAdress + api/user/addUser.

If the attempt is successful, the user gets a Toast displaying that the registration was successful and the user can proceed to log in:



A failed attempt displays an error message as shown in 3.1.2.1.

4.2.1.2.4 "CLOSE" button

The "CLOSE" button will close the application.

4.2.1.3 Activity Calls

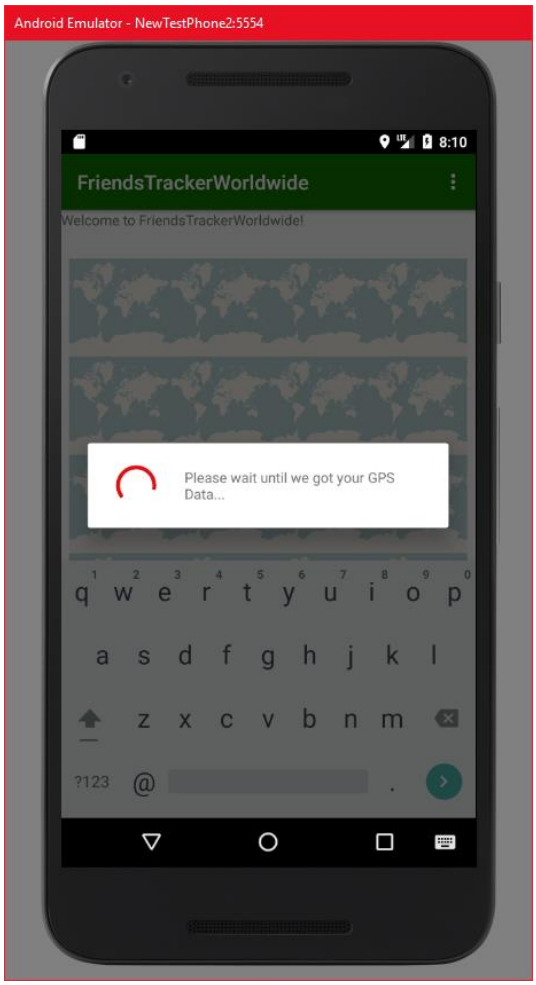
- MainActivity

4.2.2 MainActivity

MainActivity is the core of the app. It handles interaction with the map and invications of the other activities.

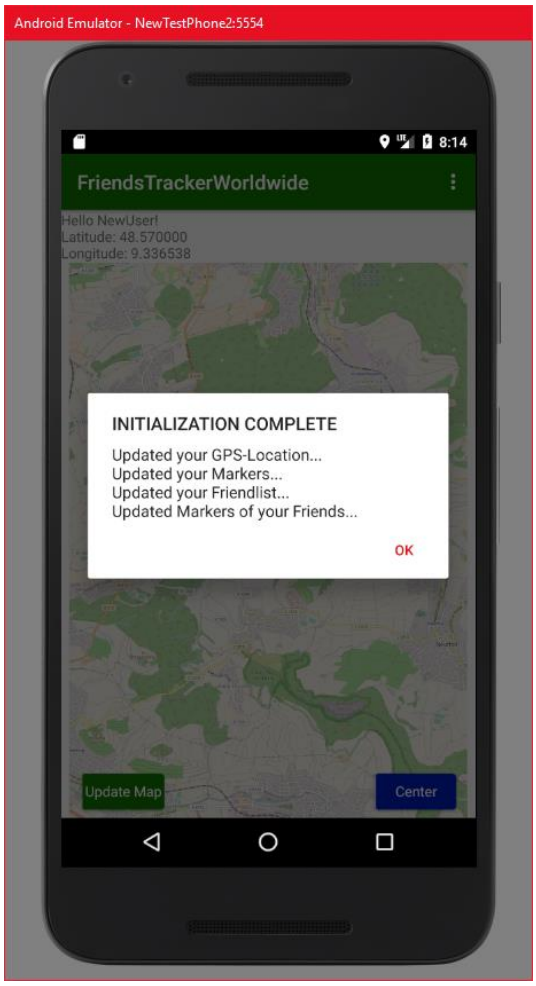
4.2.2.1 Initialization

The activity will start by waiting for the device to send GPS-data:



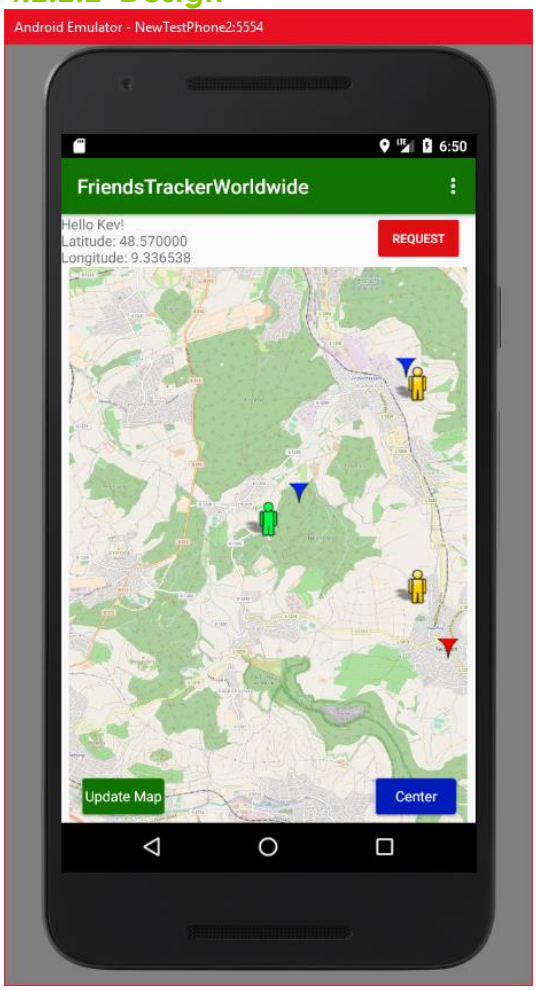
As soon as the app got the GPS-location of the smartphone, it will start to fulfill all required tasks such as saving location information in the database, receiving friends, friend markers, and the users marks from the database. Also it checks the amount of friends and open requests to either show or hide the “Requests” button.

On a successful initialization, a dialog will be displayed showing that the initialization is complete:



After clicking “OK” the app will then load the initial design of the MainActivity.

4.2.2.2 Design



4.2.2.3 Interface interaction

This activity offers the following user interactions:

4.2.2.3.1 Location Textfield

This field shows the geo coordinats of the users current location.

4.2.2.3.2 "Map Interaction"

The user can interact with the map in different ways:

4.2.2.3.2.1 "Swiping on the map"

By swiping on the screen, the user can move the map. Also this will display the ZoomIn and ZoomOut buttons.

4.2.2.3.2.2 "Single Tapping Map"

Single tapping a point in the map will display the ZoomIn and ZoomOut buttons.

4.2.2.3.2.3 "ZoomIn" button

The ZoomIn button will zoom in the map.

4.2.2.3.2.4 "ZoomOut" button

The ZoomOut button will zoom out of the map.

4.2.2.3.2.5 "Double Tapping Map"

Double tapping the map will zoom in.

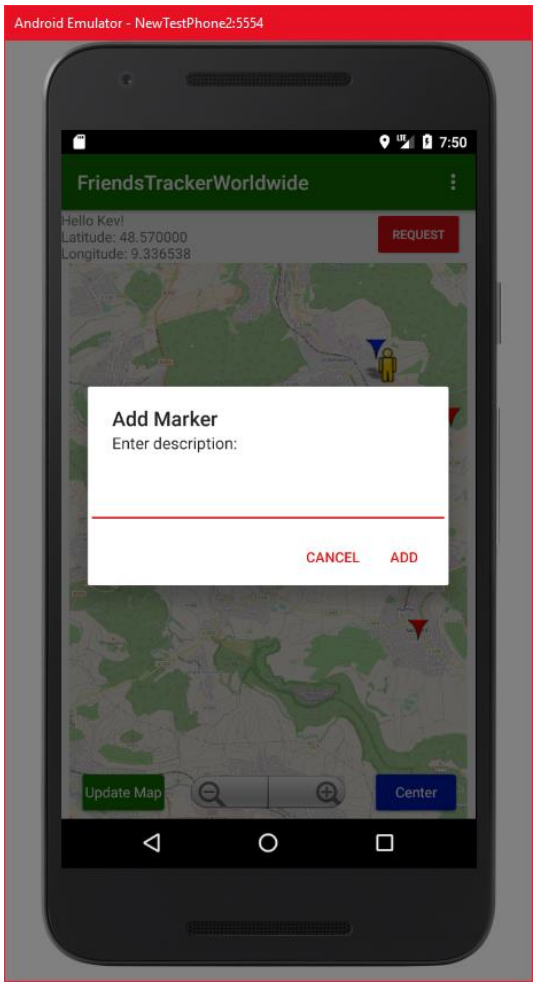
4.2.2.3.2.6 "Swiping with two fingers"

This will zoom in and out the map.

4.2.2.3.2.7 "Long Tapping Map"

This will create a personal marker.

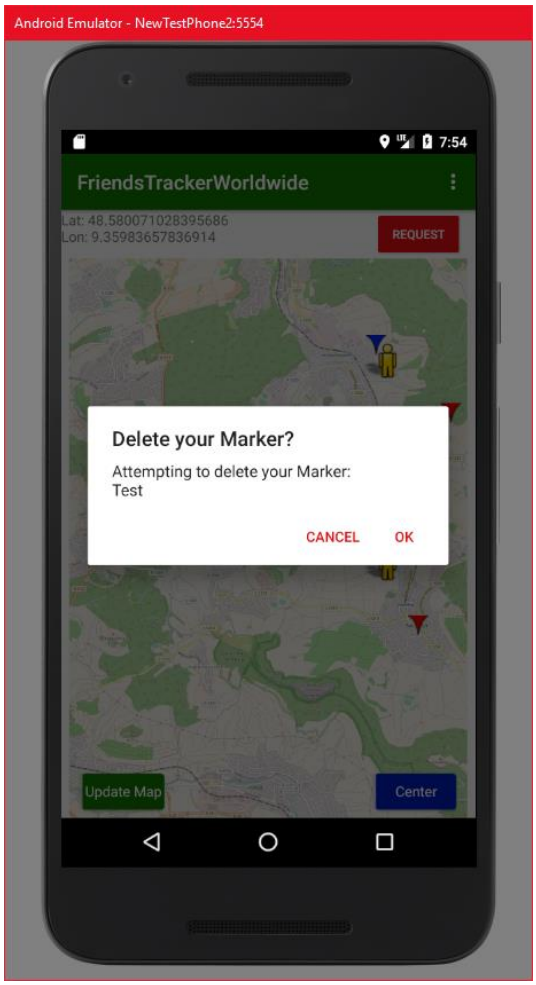
A Dialog field will pop up to enter a marker description:



Clicking on "CANCEL" will return to the map, clicking on "ADD" will then create a new personal marker on the map and send its data to the database using ServerAdress + api/marker/addMarker and update the own marker information in the database and locally.

4.2.2.3.2.8 "Long Tapping personal Marker"

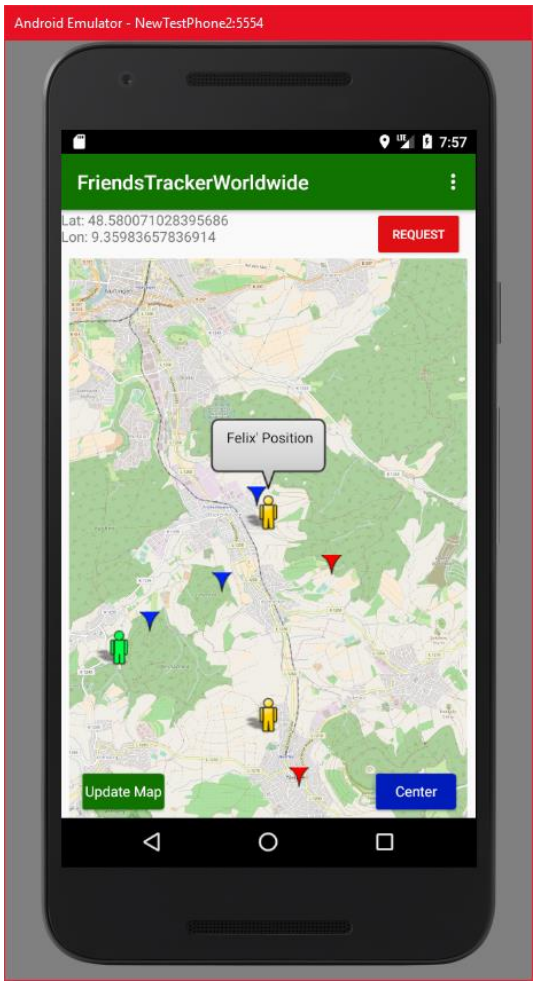
This will call a dialog field asking if one really wants to delete the marker:



Clicking on “CANCEL” will return to the normal map, clicking “OK” will delete the marker from the map and as well delete it from the database using ServerAdress + api/marker/deleteMarker.

4.2.2.3.2.9 “Single Tapping Markers or People”

By single tapping either the users own position, the users markers, friend position or friends markers the marker will display a bubble containing the markers description while centering the map at the tapped element:



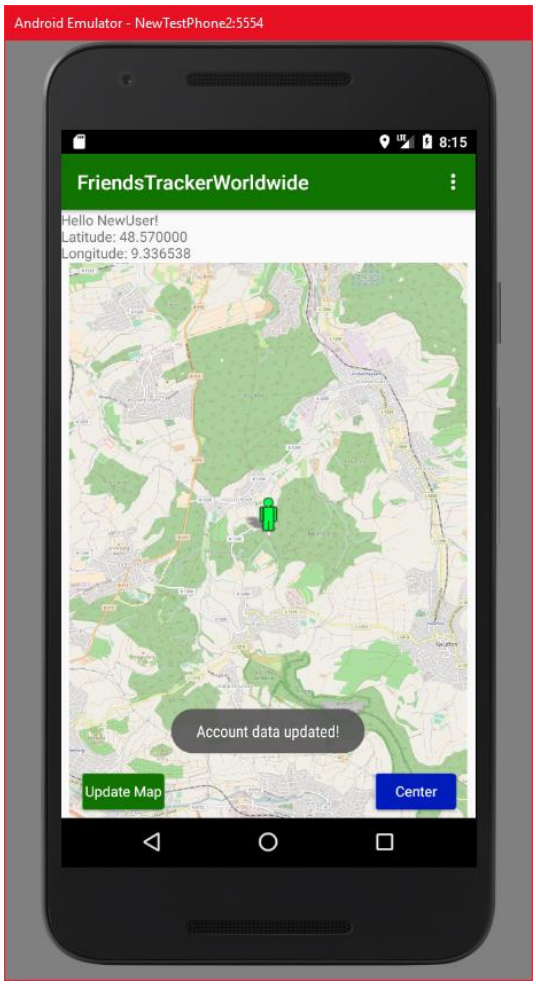
Another click on the bubble will make it disappear again.

4.2.2.3.3 "Update Map" Button

The "Update Map" button will update all information in the database including:

- Send user location calling ServerAdress + api/user/updateUser
- Update Friends and Friend Position calling ServerAdress + api/user/getFriendsLocation/:username
- Update Friends Marker calling ServerAdress + api/marker/getFriendsMarker/:username
- Update own markers calling ServerAdress + api/marker/getMyMarker/:username

When the update was successful, the app will show a Toast showing that the data has been updated:



If it fails, another Toast will show that something went wrong, displayed at the same position as the successful Toast.

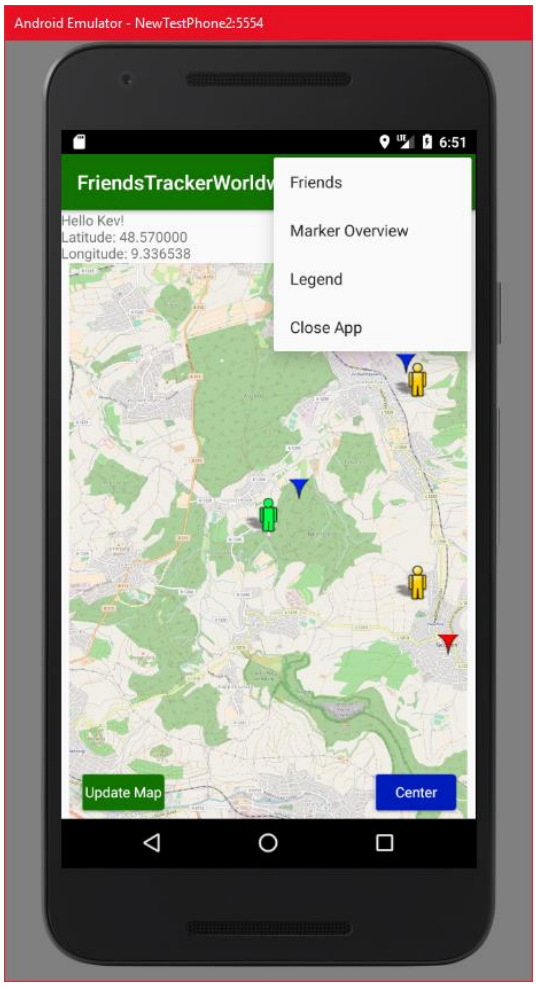
4.2.2.3.4 "Center" Button

By clicking the "Center" Button, the map will focus on the users position.

4.2.2.3.5 "Requests" Button

This button opens the RequestActivity. The Button only appears, when the user got requests. If not, the button will simply be hidden.

4.2.2.3.6 Menu Bar



The MenuBar on the top right corner is used to open the Friends (FriendsActivity), Marker Overview (NotificationActivity), Legend (LegendActivity) and closes the app to return to the login screen.

4.2.2.3.6.1 "Friends" Button

This button will open the FriendActivity.

4.2.2.3.6.2 "Marker Overview" Button

By clicking on the "Marker Overview" button, the app will calculate the distances from users location to users markers, friends positions and friends markers and then call the NotificationsActivity.

For updating information, the following FriendsTrackerWorldwide API methods are called:

- ServerAdress + api/users/getFriendsLocation/:username
- ServerAdress + api/marker/getMyMarker/:username
- ServerAdress + api/marker/getFriendsMarker

4.2.2.3.6.3 "Legend" Button

By clicking the "Legend" Button, the app will open the LegendActivity.

4.2.2.3.6.4 "Close App" Button

This will close the MainActivity and return to the LoginActivity.

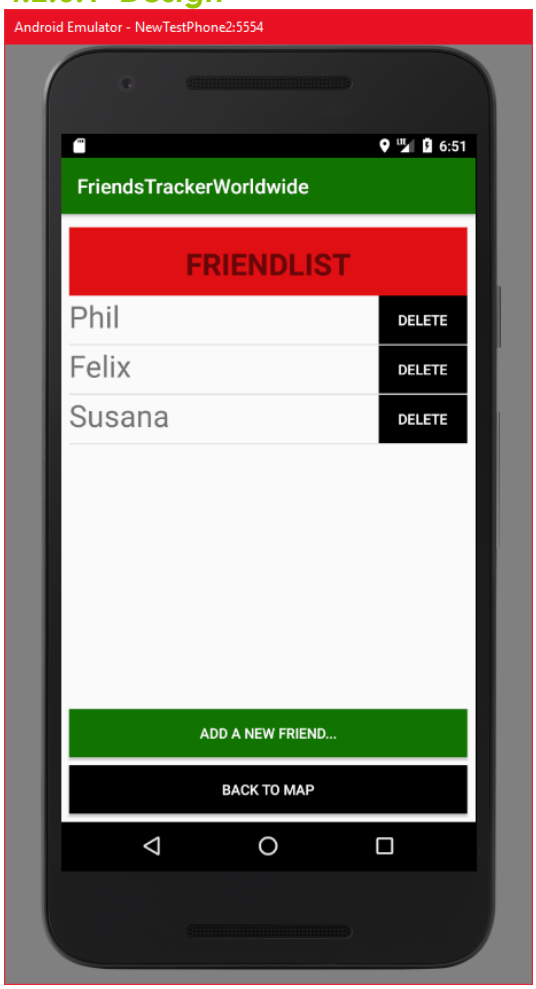
4.2.2.4 Activity Calls

- FriendActivity
- NotificationActivity
- RequestActivity
- LegendActivity
- LoginActivity

4.2.3 FriendActivity

The FriendActivity manages displaying the friendlist, sending friend requests to other users and deleting friends from the friendlist.

4.2.3.1 Design



4.2.3.2 Interface Interaction

4.2.3.2.1 "ADD A NEW FRIEND..." Button

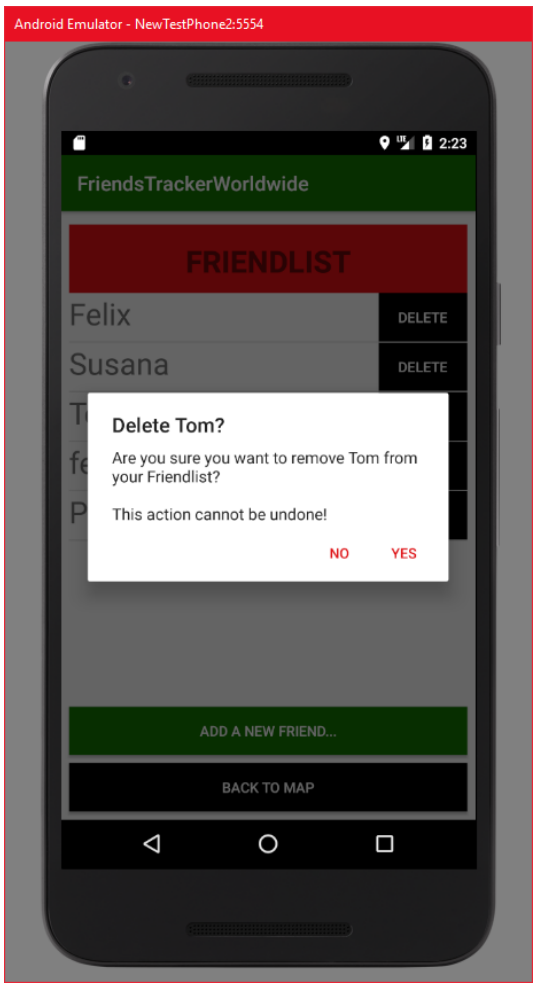
A click on this button starts the AddFriendActivity.

4.2.3.2.2 "BACK TO MAP" Button

This button closes the current activity and goes back to the main activity. If the friend data has been changed, the MainActivity will then perform an update on the friend and friend marker information.

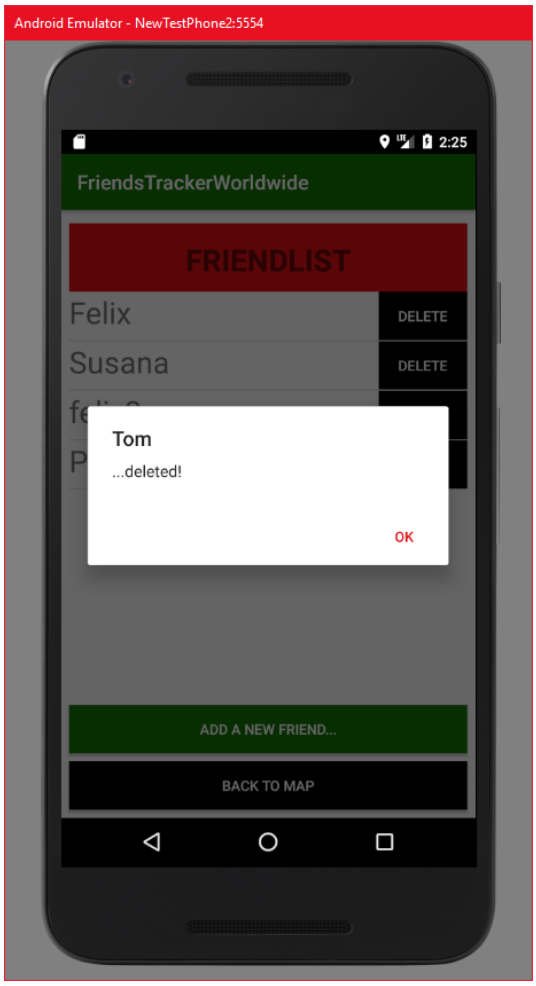
4.2.3.2.3 "DELETE" Button

The "DELETE" button will pop up an alert dialog:



"CANCEL" will return to the acitivty.

"OK" will delete a friend from the friend list by using the api call ServerAdress + api/user/deleteFriend and creating a confirmation dialog:



4.2.3.3 Activity Calls

- AddFriendActivity
- MainActivity

4.2.4 AddFriendActivity

This activity communicates with the database in the background to send friend requests and then updates the friendlist in FriendActivity and the requestlist in RequestActivity.

4.2.4.1 Design

4.2.4.2 Interace Interaction

4.2.4.2.1 "enter friendname…" Field

Enables entering a name.

4.2.4.2.2 "Back" Button

This will close the current view and go back to the FriendActivity.

4.2.4.2.3 "ADD" Button

Clicking on "ADD" will use the entered name and attempt to send a friend request to this user by using api call ServerAdress + api/user/addFriend.

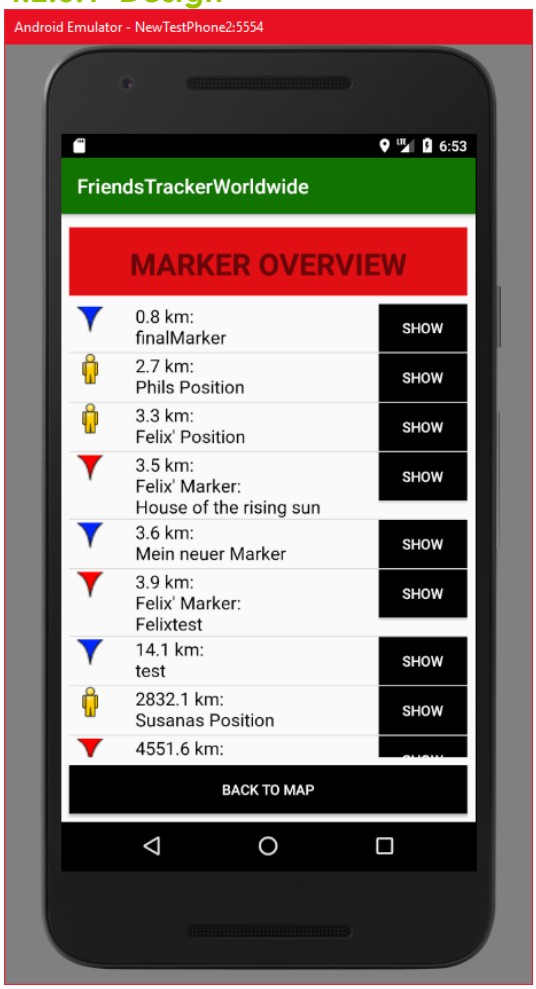
4.2.4.3 Activity Calls

- FriendActivity

4.2.5 NotificationActivity

NotificationActivity enables displaying all markers in a listView sorted by distance using a custom ArrayListAdapter.

4.2.5.1 Design



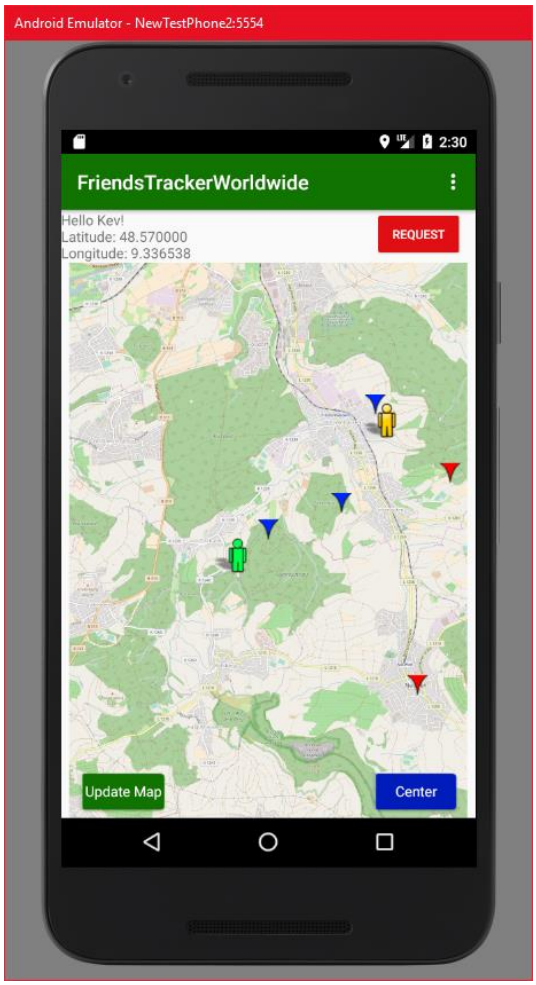
4.2.5.2 Interace Interaction

4.2.5.2.1 "BACK TO MAP" Button

Closes the current activity and goes back to the MainActivity.

4.2.5.2.2 "SHOW" Button

A click on that button will take the related marker element, close this activity and center the map of the MainActivity to the chosen marker:



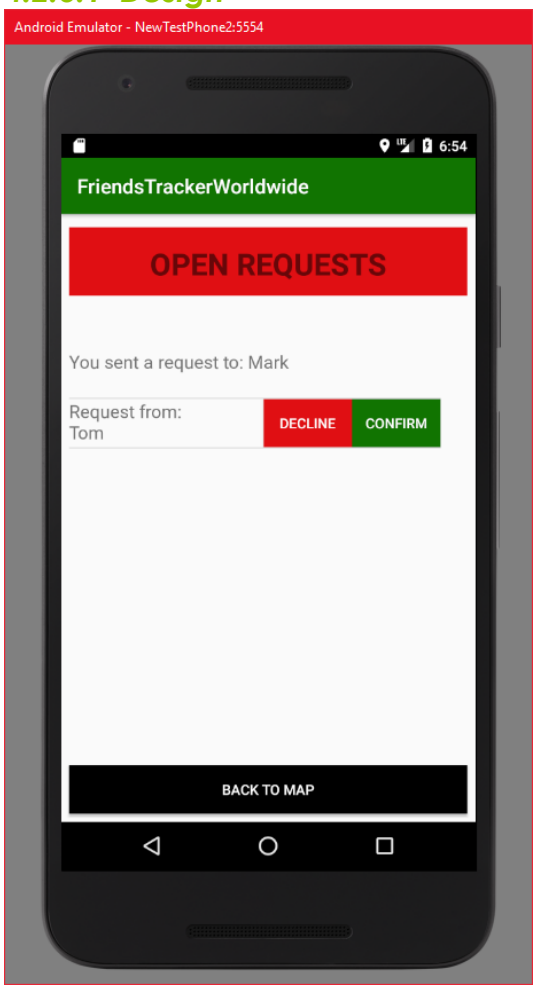
4.2.5.3 Activity Calls

- MainActivity

4.2.6 RequestActivity

This activity displays all open friend requests, which are either received or sent by the user, in a list using a custom ArrayListAdapter.

4.2.6.1 Design



4.2.6.2 Interface Interaction

4.2.6.2.1 "BACK TO MAP" Button

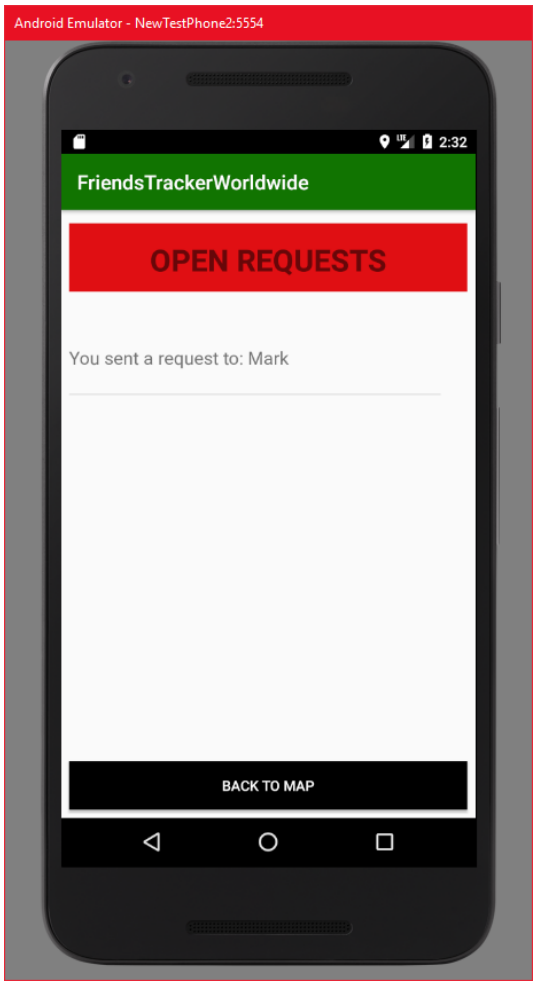
Closes the activity and goes back to MainActivity.

4.2.6.2.2 "DECLINE" Button

This will remove the request from the request list and remove the friendrequest in the database using ServerAdress + api/user/denyFriendRequest.

4.2.6.2.3 "CONFIRM" Button

This will remove the request from the requestlist and add a friend to the friendlist. Also it will update the request Arrays and friendlist array in the database calling ServerAdress + api/user/confirmFriendRequest:



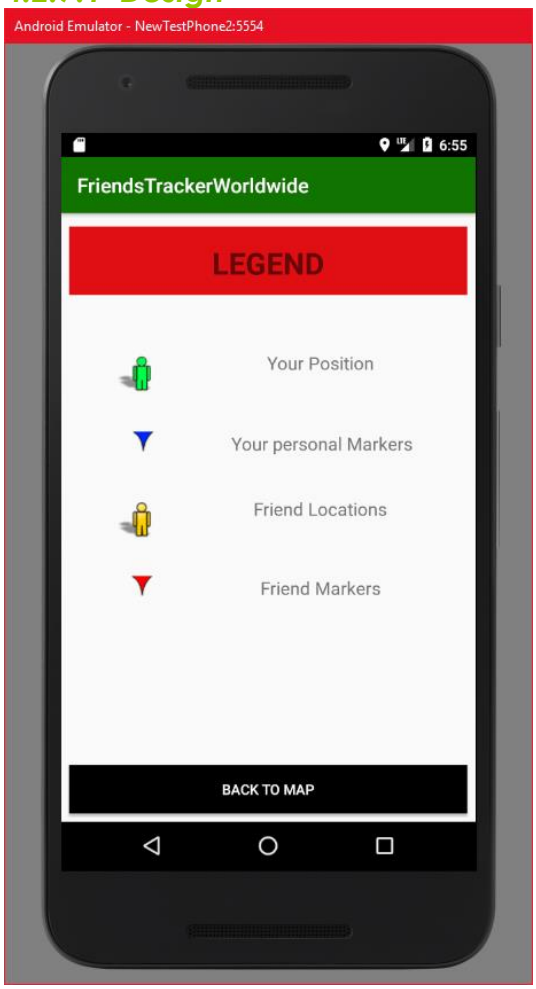
4.2.6.3 Activity Calls

- MainActivity

4.2.7 LegendActivity

This activity simply displays a legend of the symbols used on the mapView.

4.2.7.1 Design



4.2.7.2 Interface Interaction

4.2.7.2.1 "BACK TO MAP" Button

This will close the current activity and go back to the MainActivity.

4.2.7.3 Activity Calls

- MainActivity

4.3 Update Intervall

On the initial load when calling the MainActivity after login, the app will update the users location together with friends, requests, and friend markers by communicating with the database.

4.3.1 Users location

After the initial load the users location will locally update every 30 seconds by retrieving GPS data from the device, but this will not update the location in the database! The users location will only be sent to the database when manually pressing the “Update Map” button in the main activity.

4.3.2 Friends and friend markers

Friends and the friendlist as well as the friends markers will be updated locally and in the database everytime the user changes friend information. The changes will be applied when:

- removing a friend
- sending a friend request
- confirming a friend request
- denying a friend request
- pressing “Update Map” button

4.3.3 Personal marker

The personal markers will be updated locally and in the database everytime the user changes his own marker information. The changes will be applied when:

- adding a marker
- removing a marker
- pressing “Update Map” button

4.3.4 Marker Overview

The marker overview will always be updated when:

- user locally changes position
- friends get changed
- friend markers get changed
- personal markers get changed

4.4 Additional Java Classes

For developing the Application we needed to create some additional classes in the project.

4.4.1 ApiCaller

This class has the methods executePut, executePost, executeGet and executeDelete. It is used quite often in the application for invoking any database request.

4.4.2 MarkerCompareHelper

This class was used for one way of sorting markers in the NotificationActivity at the beginning for using a custom comparator. However, it is not used in the moment.

4.4.3 MyArrayAdapter

This class inherits from the ArrayAdapter. It is used for displaying friends in the FriendActivity listView using a specified friendlist_entry layout for displaying a textbox together with a delete button.

4.4.4 NotificationAdapter

This class inherits from the ArrayAdapter as well, but it is used in the NotificationActivity for displaying all markers and friend positions with a little icon, a textbox and a button for centering the map on the chosen list item.

4.4.5 MyArrayAdapterRequest

This class again inherits from ArrayAdapter, but this time it is used and designed for displaying the sent friend requests and the open friend requests in the RequestActivity. Sent requests will only be displayed as a textbox, open requests will be displayed with a confirm and decline button for handling the requests.

4.4.6 Quicksort

This was used for using a quicksort algorithm on double values given by an ArrayList containing MarkerCompareHelper items. This is not used in the app anymore.

4.4.7 SpecialMarker

This is a class that inherits from Marker. It is used for adding an id attribute to personal markers to enable deleting them using a long tap motion.