Compulsory Project 2

Android Project

Hasan Hasanli

Maksud Ganapijev

Nilam Koirala

Table of Contents

Introduction (Maksud Ganapijev) 3
Related Work 4
- OpenSignal.com 4
- Network Cell Info 5

High-level statement 6
- Goal of the project 6

Time planning 7
- Project plan 7
- Team roles 8
- Project versioning platform 9
  - GitHub 9
  - Ticketing system 10
- Subtask division 11

Technical documentation 12
- Android application 12
  - Architecture 12
  - Code libraries in use, functions implemented 14
  - Layout and UI of an application 18
- Application Front-End Implementation (Nilam Koirala) 19
  - Drawing overlay 19
  - Location 20
  - Drawing Polyline 21
  - Coordinates Approximation 22
- Server application (Hasan Hasanli) 24
  - Architecture 24
  - Fetch, retrieve, convert data from query (coordinates, signal strength) 27
  - File structure 28
  - Updating data 29

Lessons Learned (Maksud Ganapijev) 30
- What went wrong 30
- What went right 31

Demo description 32

Introduction (Maksud Ganapijev)

Today staying connected to the Internet is crucial. Network operators are making sure that every piece of land is covered by signal waves, generated by one of their cell towers. However signal strength varies drastically. Thus an attempt to create a tool, which would help to see in advance signal coverage and its strength all around the earth, was made.

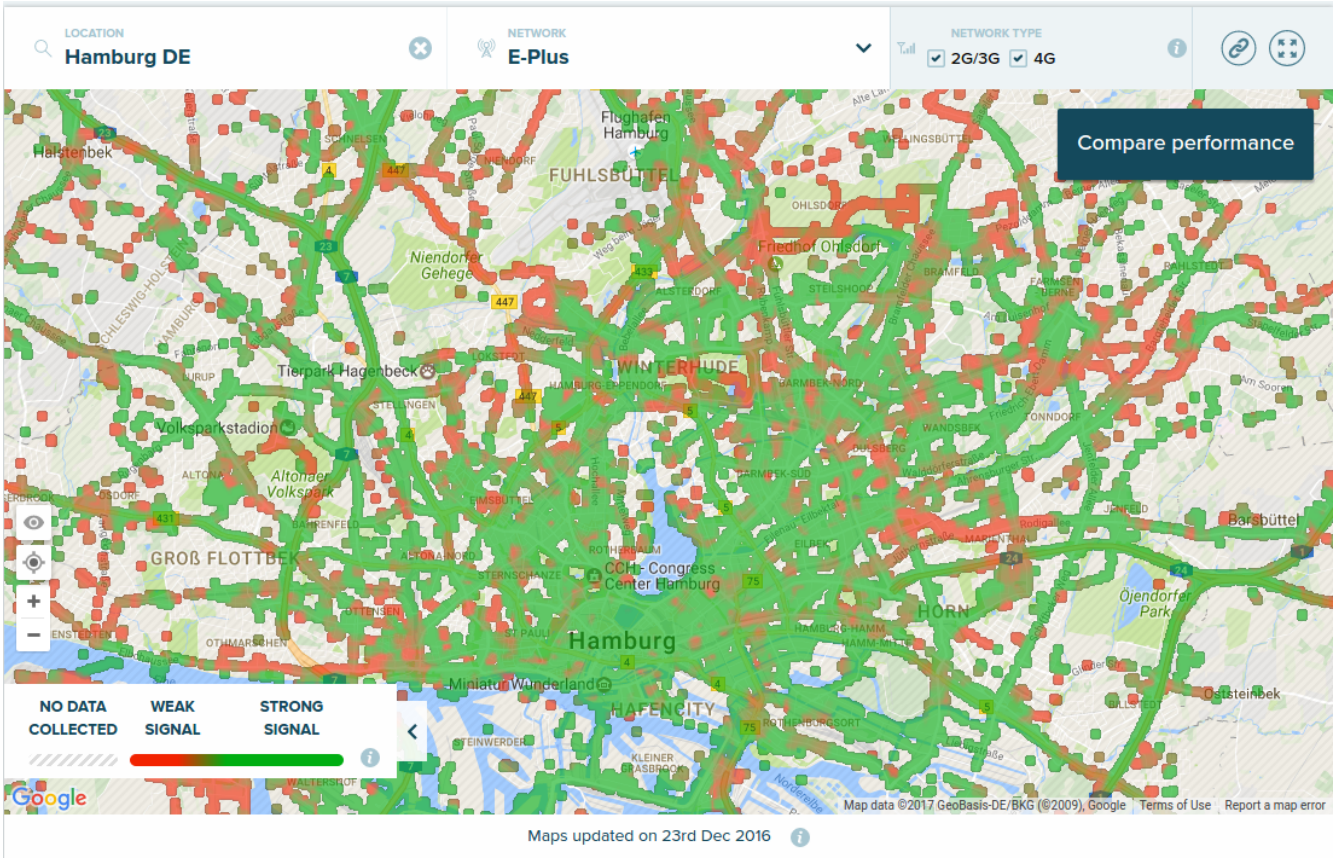
The following report contains information on a development of this tool, to be exact a mobile application for an Android platform.

Related Work

Research for analogous systems had been accomplished before starting, in order to understand an end product better. Additionally to previous group year results, a best example of such a tool was found.

OpenSignal.com

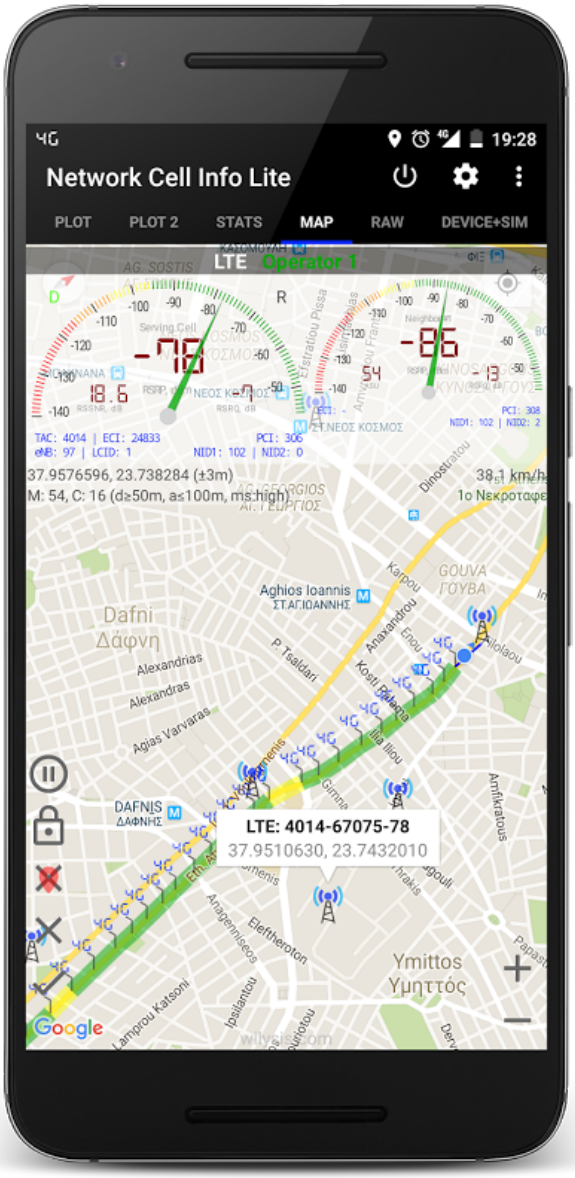
System is based on a GoogleMaps API and contains a large, well structured database. It provides a selection between operators and 3G/4G networks. However it was noticed, only two colors are used in a heat map, from which is hard to understand an actual signal reception strength.



Pic. 1: OpenSignal.com web application

Network Cell Info

Tool provides same basic functionality, smaller database, however useful features like cell towers locations and connectivity speed tests are provided.



Pic. 2: Network Cell Info Android application

High-level statement

Goal of the project

A goal of the project is to create an application for an Android platform and a back-end system, located on a server, which will process collected data and generate an informative heat map from it. An application must have the following basic functionality: a map of surrounding area, a pointer to a location of the user, a menu which will allow a user to enable or disable a heat map with signal reception strength information, an automated system which will collect information on signal reception strength and upload it to a server for further processing.

Time planning

Project plan

The following time table was developed at the beginning of the project.

Team left spare 3 weeks for implementation of additional features, or bug fixing.

Owner	Task	Week 39	Week 40	Week 41	Week 42	Week 43	Week 44	Week 45	Week 46	Week 47	Week 48	Week 49	Week 50	Week 51
Hasan	Basic research													
Maksud														
Nilam														
Maksud	Project flowchart													
Hasan	Server backend													
Hasan	Location coordinates storage data structure (server-side)													
Maksud	GET method communication interfaces													
Hasan	Image mapping (server-side)													
Maksud	GANTT chart													
Maksud	TelephonyManager and signal stregth													
Maksud	Operator fetching													
Nilam	Map overlay													
Maksud	Code repository setup													
Hasan														
Maksud	Differentiation between network types (3G/4G)													
Hasan	Overlay placement logic													
Maksud	Fetching and saving content from server													
Hasan	Detect coordinates with best signal around													
Maksud	Receive best signal coordinates from server													
Nilam	Latitude and longitude fetch from gps													

Table 1: GANTT chart

Team roles

There were 4 major parts in project, with team consisting out 3 members:

- Hasan Hasanli – server-side architect, server-side software developer
- Maksud Ganapijev – android application architect, android application software developer
- Nilam Koirala – android application software developer

Primary project plan was drawn, so team members would have understanding of general project

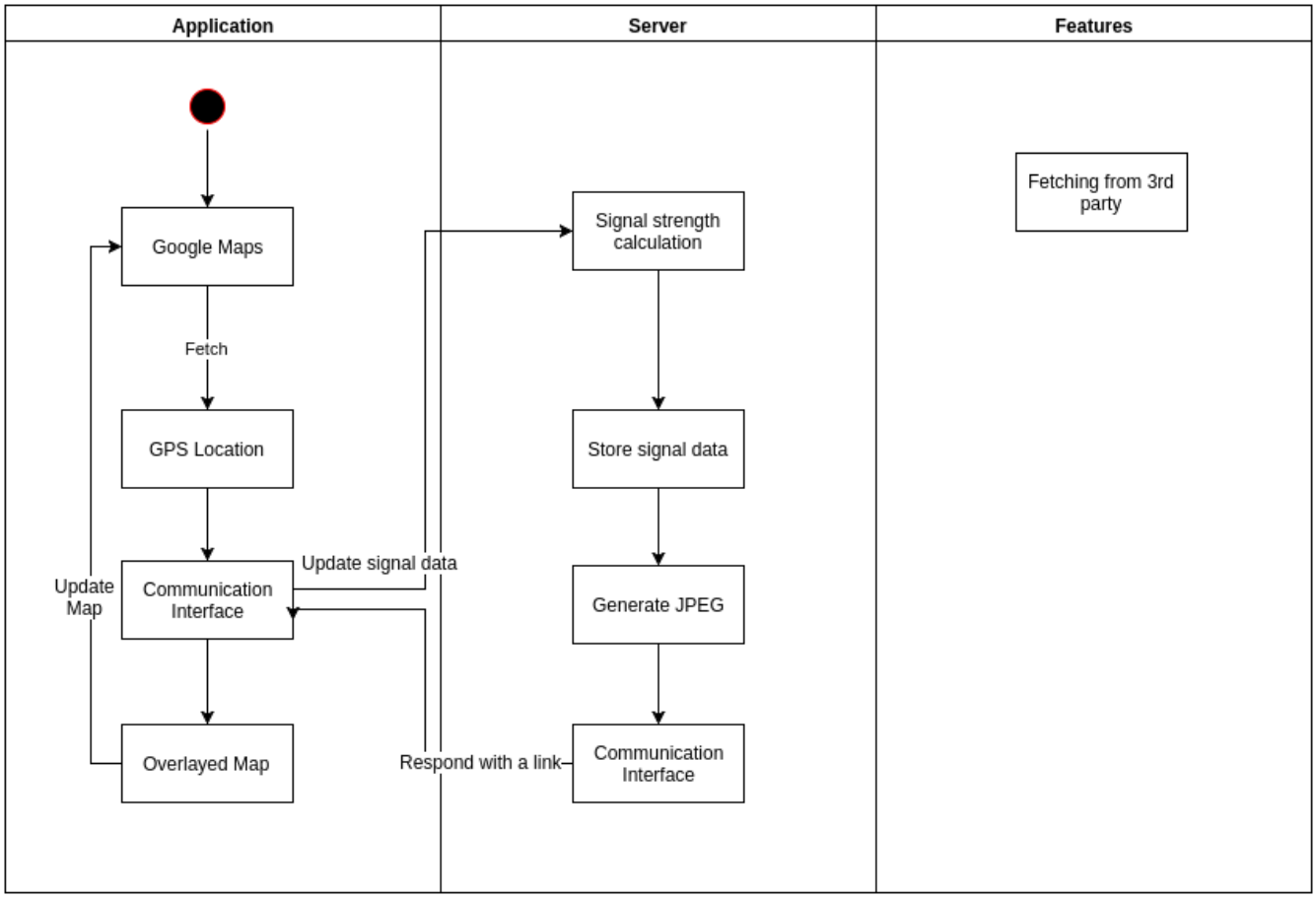


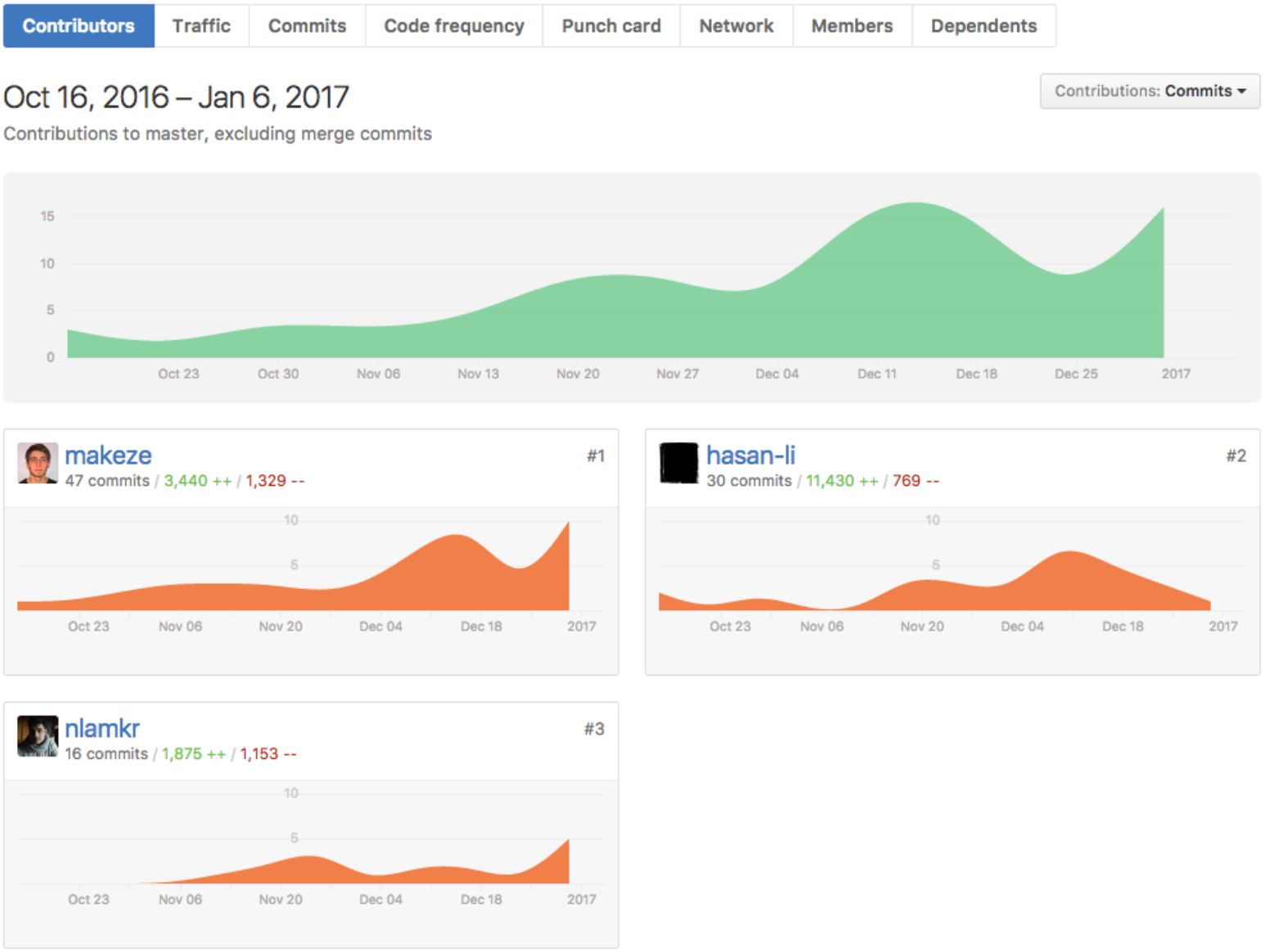
Diagram 1: Primary project flowchart

From diagram can be seen, Google Maps API was chosen as a main development tool. There were two largest APIs suitable for this project: Open Street Maps and Google Maps. Since Google Maps API provided better documentation and larger involved larger user community, it was decided to stick to Google Maps API.

Project versioning platform

GitHub

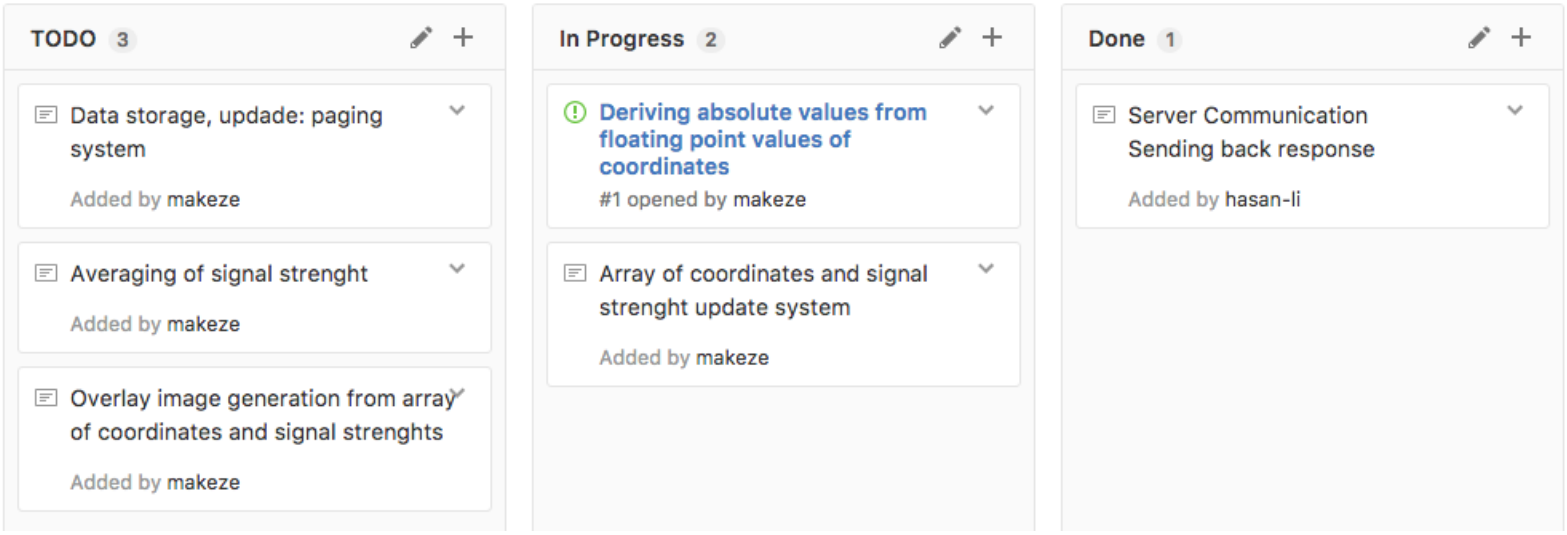
GitHub VCS was selected as a code database. A system played critical part in organization and update conflict solving.



Pic. 3: GitHub project contributors page

Ticketing system

Task sharing was organized in a very liberal way. Every team member could have added a ticket with problem/feature implementation to “TODO” section, as well as work on any problem/feature that was open by moving it to “In Progress” section, finally get credit for accomplishing it by moving to “Done” section.



Pic. 4: GitHub project organization

Subtask division

The following team members took care of corresponding tasks

Task	Team member	Status
Android applicatoin architecture	Maksud Ganapijev	Done
Server architecture	Hasan Hasanli	Done
Main application layout	Maksud Ganapijev	Done
Location fetching	Nilam Koirala/Maksud Ganapijev	Done
Signal strength fetching	Maksud Ganapijev	Done
CGI programm	Hasan Hasanli	Done
Map overlay	Nilam Koirala	?
Server update service	Maksud Ganapijev	Done
Heat map request service	Maksud Ganapijev	Done
Information storage on server side	Hasan Hasanli	Done
HTML page parsing for image url	Maksud Ganapijev	Done
Image download and storage	Maksud Ganapijev	Done
Permission class	Maksud Ganapijev	Done
Heat map generation	Hasan Hasanli	?
Point with best location detection	Hasan Hasanli	Done
Network operator detection	Maksud Ganapijev	Done
Difference between 3G/4G	Maksud Ganapijev	Done
Data averaging	Hasan Hasanli/Maksud Ganapijev	Done
Fetched information storage in a temporary data structure, in case of application crash	Maksud Ganapijev	?

Table 2: Tasks completed by members

Technical documentation

Android application

Architecture

Several implementations of an Android application architecture were available. It was decided to use a single running Activity, with several constantly running services (LocationCoordinates, SignalStrength, UploaderService, DownloaderService)

An event for the data collection is realized in an UploaderClass, it is triggered every N seconds (N is configurable). As diagram shows UploaderClass is called by mainMapActivity, which later fetches latitude, longitude from LocationCoordinatesService, and signal reception strength from SignalStrengthService. All data is shaped into a URL with additional parameters*, then forwarded to a server for further processing.

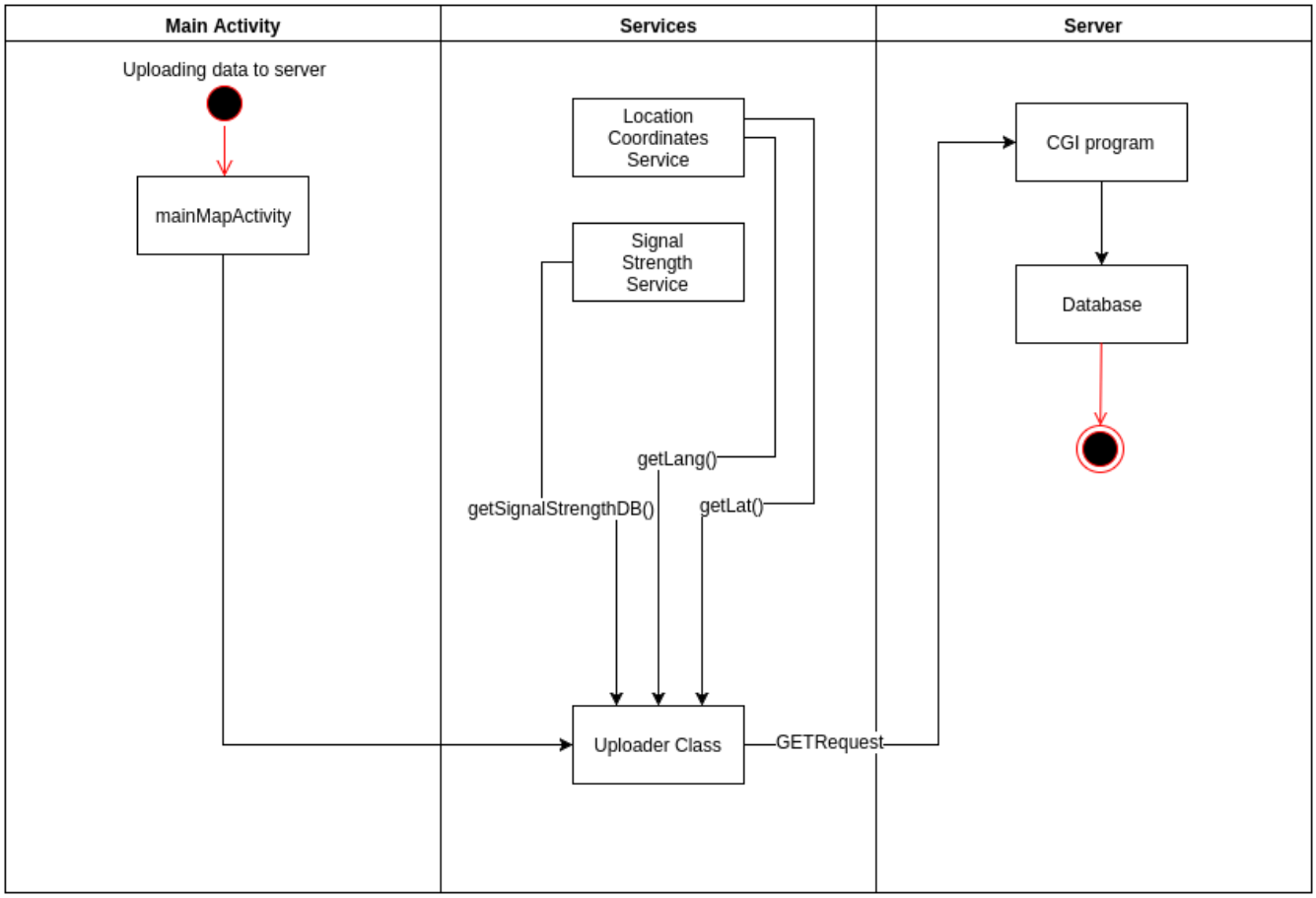


Diagram 2: Uploading data to server

*Example of a URL: http://r1482a-02.etech.haw-hamburg.de/~w16cpteam1/cgi-bin/index?x=53.589358&y=10.018434&y=-139

A heat map request from a server is a more complex task, since requires asynchronous tasks to be executed sequentially.

MainMapActivity triggers Uploader class, which fetches coordinates from LocationCoordinatesService, then send URL request with additional parameters to a server. Server responds with an HTML page containing a link to a heat map image location on a server and coordinates of a best signal strength around. HTML contents are then passed to a LinkDownloaderClass, it parses the link out of it and passes it to ImageDownloaderClass, which downloads a bitstream and saves as a bmp file to a device storage.

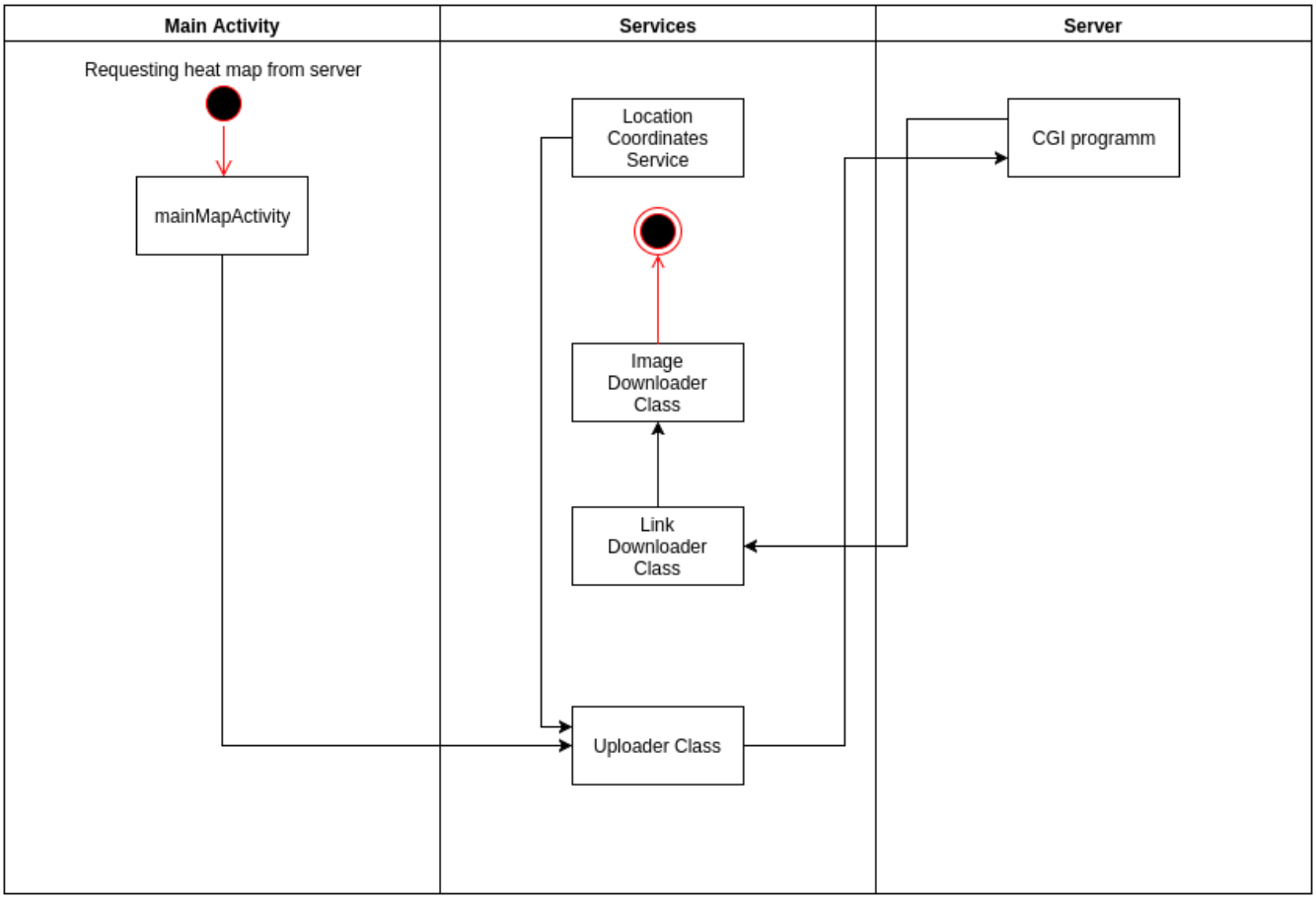


Diagram 3: Fetching a heat map from server

Code libraries in use, functions implemented

Java.net.HTTPUrlConnection

A proprietary* Java library provides instrumentary to accomplish network operations.

With use of this library the following features were implemented:

- Data upload to a server with use of GET requests. Signal strength and location coordinates are put into URL as an extra parameters, then directed to server for post processing.
- Data streams download from server with use of GET requests. Location coordinates are put into URL as an extra parameters, then directed to server, server responses with a data stream, which is stored with use of Java utilities.

Code below demonstrates how one can easily reach remote server with a GET request.

```
protected Integer doInBackground(Void... urls) {
        try {
            URL url = new URL(urls[0]);
            Log.i("chat", baseUrl+params);
            connection = (HttpURLConnection) url.openConnection();
            connection.setConnectTimeout(10000);
            connection.setRequestMethod("GET");
            connection.setRequestProperty("User-Agent", "Mozilla/5.0");
            connection.connect();
            res = connection.getResponseCode();
            Log.i("UpdaterLog", "+ server response (200 - everything ok): "
                    + res.toString());
        } catch (Exception e) {
            Log.i("UpdaterLog", "+ connection error: " + e.getMessage());
        } finally {
            connection.disconnect();
        }
        return res;
}
```

If a web server answers with a bit stream, one can buffer it with basic Java utilities, like BufferedReader and save to a device memory.

```
InputStream is = new BufferedInputStream(connection.getInputStream());
Bitmap bitmap = BitmapFactory.decodeStream(is);
saveBmp(bitmap);
```

*A proprietary Java library – it is important to note, that Apache library HttpConnect, used by Android before is now deprecated and no more used by Android. New name of a library HttpUrlConnection.

Android.telephony.TelephonyManager

A library provides all sort of information on mobile device network.

With use of this library the following features were implemented:

- SignalStrength service getSignalStrengthDBm(), service function returns signal strength in DBm in form of integer number.
- SignalStrength service getAllInfo(), service function returns a list with rest information about mobile network, such as operator name, country, phone number and so on. Function is implemented however not used.

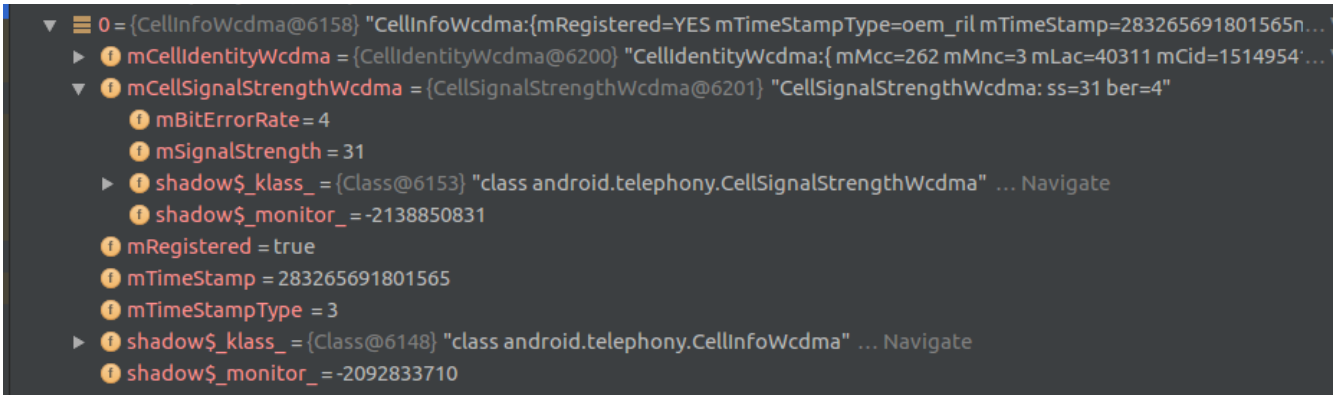
An information of greatest interest for the project was obtained the following way.

```
public void onCreate(){
    mPhoneStateListener = new MyPhoneStateListener();
    mTelephonyManager = (TelephonyManager)
getSystemService(Context.TELEPHONY_SERVICE);
    mTelephonyManager.listen(mPhoneStateListener,
PhoneStateListener.LISTEN_SIGNAL_STRENGTHS);
}
class MyPhoneStateListener extends PhoneStateListener {
    @Override
    public void onSignalStrengthsChanged(SignalStrength signalStrength) {
        super.onSignalStrengthsChanged(signalStrength);
        mSignalStrength = signalStrength.getCdmaDbm();
          allCellInfo = mTelephonyManager.getAllCellInfo();

    }
}
```

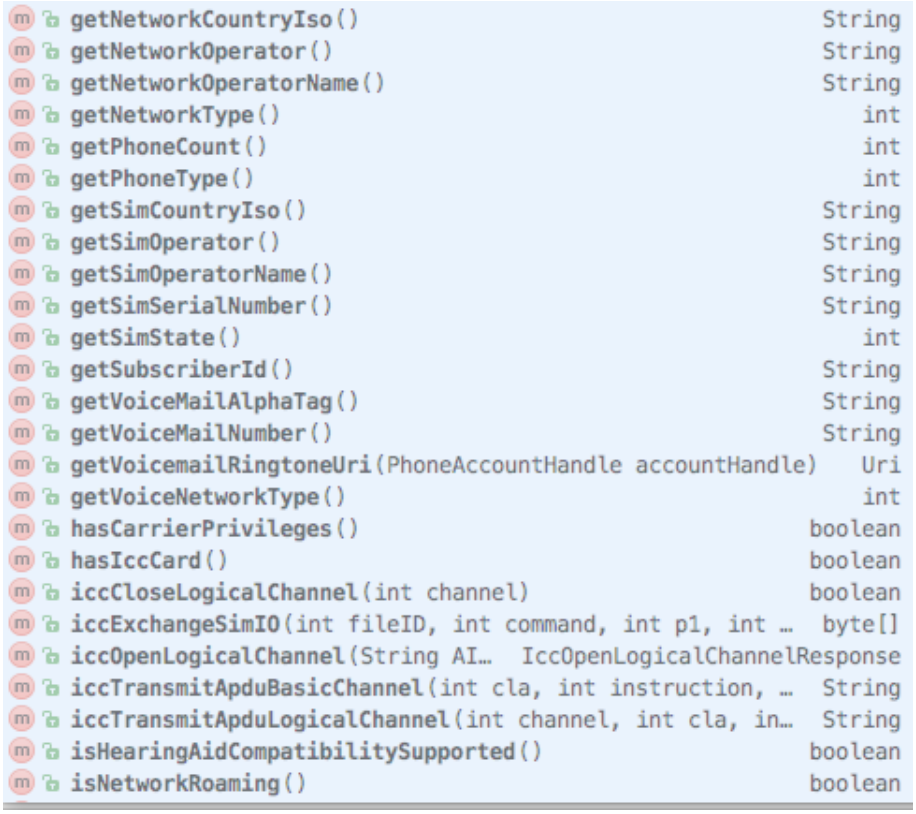
It was planned to differentiate between 3G and 4G networks, even greater achievement would be to have a separate data structures for different network operators, however some limitations didn't allow a team to implement this feature on server side, so services developed left unused.

Below, a chunk of information returned by `TelephonyManager.getAllCellInfo();` is shown. On line 4 and 5 useful information is provided, in form of BitError and SignalStrength.



Pic. 5: a screenshot from debug section

A large set of tools is provided by Android.TelephonyManager library, single get function returns required information.



Pic. 6: Information that is possible to gather from TelephonyManager

Android.location.LocationManager

A library that provides simplifies use of location services. Even though GPS is most common location system, a library allows location fetching from all possible location providers with a single function.

```
provider = locManager.getBestProvider(new Criteria(), false);
```

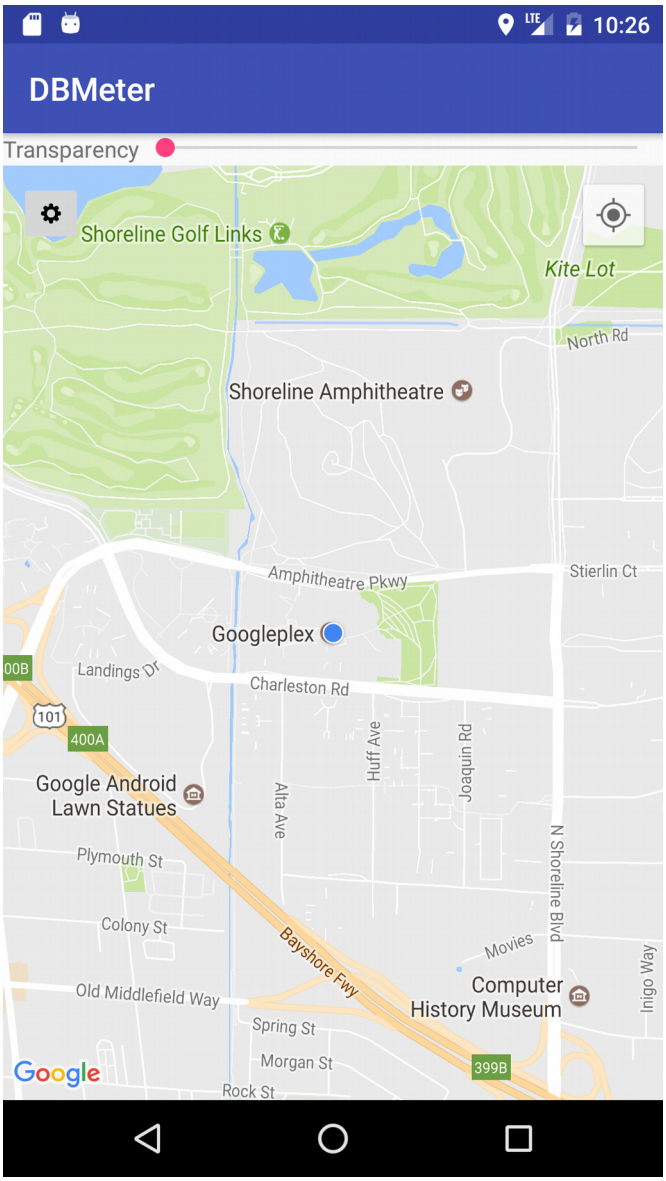
A function LocationManager.getBestProvider() returns first available location service, even if devices possesses more than one location provider (I.e GPS or Glonas).

With use of this library the following features were implemented:

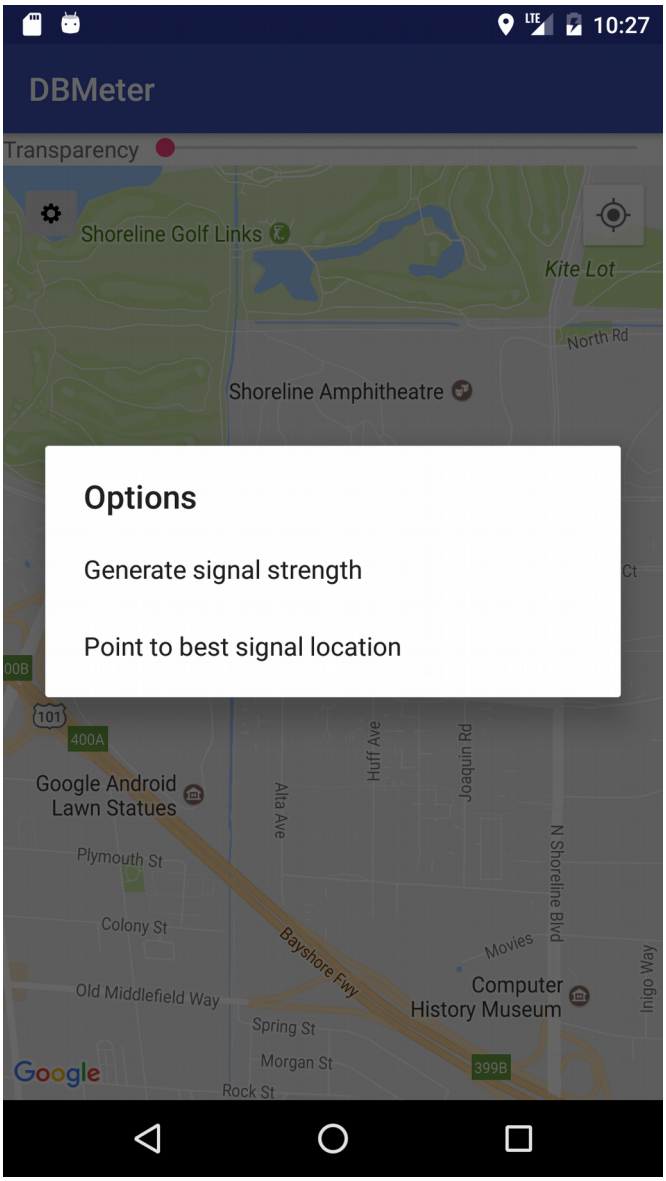
- getLangitude(), getLongitude(), function names are self-explanatory.

Layout and UI of an application

Very simplistic design has been chosen, with basic controls.



Pic. 7: MainActivity view



Pic. 8: Functions available

A menu works the following way. Button with “Gear” icon, on a long press creates a floating menu, with list of two function calls.

- Generate signal strength heat-map, draws a signal strength heat-map on top op original map. If map is not present, heat-map is requested from server.
- Point to best signal location, marks best signal location on a map and an arrow to it.
- Transparency scroll bar, changes transparency of a heat-map.

Application Front-End Implementation (Nilam Koirala)

Android application called “DBMeter” has been developed as a compulsory project. The main project tasks in the front-end part was to display the signal strength as an overlay on the top of the map, find the nearest strong signal point and show the route on the map. The following has been implemented in this project.

1. Drawing overlay on the map to show the signal strength
2. Location
3. Drawing polyline instead of route to show the strong signal point.
4. Coordinates approximation for the overlay

Drawing overlay

Overlay is successfully implemented in this project. ‘GroundOverlay’ class gives the possibility to create overlay on the map. A sample bitmap image has been used to test the functionality of the implementation. The application loads the image from the directory created by the app and it is displayed on the application screen based on the predefined bounds. More about bounds will be explained in the coordinates approximation section.

```
bound = new LatLngBounds(SW, NE);
image = BitmapDescriptorFactory.fromPath(dir + "/" + imageToOverlay);
mGroundOverlay = mMap.addGroundOverlay(new GroundOverlayOptions()
        .image(image)
        .positionFromBounds(bound)
        .transparency(0.1f));
```

Pic. 9: code for drawing overlay

SW stands for South West (bottom right corner of the image) and NE stands for the North East (top left corner of the image) edges of the image. Bounds has been used instead of the position (X-axis & Y-axis length) because bound has an advantage of placing the image exactly on the top of map.

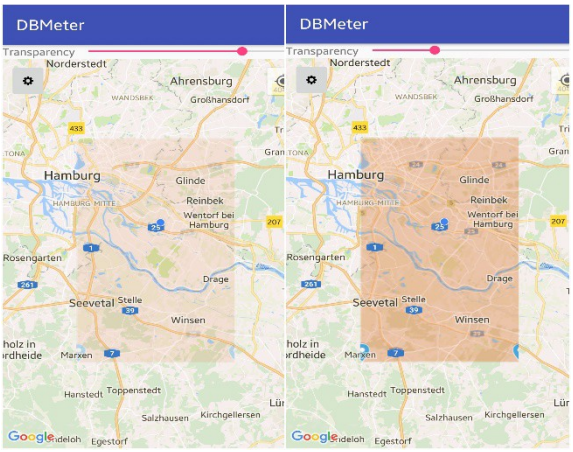
A custom transparency bar has been implemented on the app so that user can set the visibility of the overlay as desired. ‘Seekbar’ is used to set the transparency of the image overlay. The transparency range lies between the float values of 0 to 1 for the overlay and lies between 0 to 100 for the Seekbar.

A method called onProgressChanged() has been used to track the progress of the Seekbar. This method takes the integer value progress as a parameter which will be divided by the maximum transparency of Seekbar (100) to obtain the actual transparency range of the overlay (0.0 – 1.0).

For example:

```
mGroundOverlay.setTransparency((float) progress / (float) TRANSPARENCY_MAX);
```

When overlay exist, then we set the transparency as above example inside the method onProgressChanged().



Pic. 10: Overlay with transparency

Location

For application that uses the maps, location is the most essential part. A different class for the location has been created named LocationCoordinates. This class implements the LocationManager which provides access to the location provider of the system. Likewise, LocationListener has been implemented, that is responsible for the continuous update of the location.

Permission has been granted for the location services on the application Manifest. Location class returns the location values by using the method onLocationChanged(). Location class returns the double value of latitude and longitude from the methods implemented on the class.

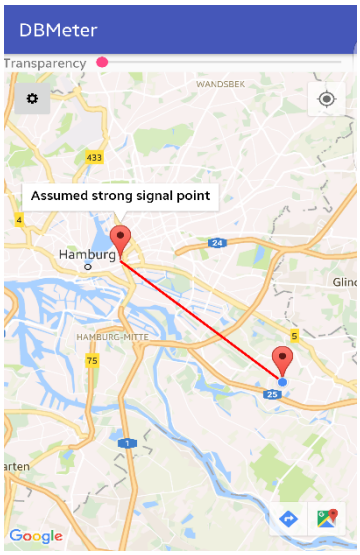
Drawing Polyline

One of the main task for this project was to implement the route. User should be able to see a path for the nearest strong signal point. Due to the time constraint, this has not been implemented properly. Instead of the route, a polyline has been drawn on the map.

```
mMap.addPolyline(new PolylineOptions()
.add(new LatLng(latitude, longitude), new LatLng(53.555037, 10.022218))
.width(10)
.color(Color.RED));
```

Pic. 11: Polyline implementation

The implementation on Pic. 7, draws the polyline from the current location point to the strong signal point. Location with LatLng (53.555037, 10.022218) is assumed to be the strong signal point. This is the point where IBM building lies in Berliner Tor. A red line of width 10 has been drawn for the test.



Pic 12: Polyline implementation

Coordinates Approximation

Since displaying heat map of signal strength as an overlay is the main task of the front-end implementation, coordinates approximation is one of the most important task to track the overlay. The basic idea behind the coordinates approximation is that the heat map of signal strength is based on location. Whenever the location changes, application should display the overlay for that particular location area. This simply means, displaying a signal map of Berlin for Hamburg makes no sense at all.

At first we decided to save the image on the application directory as the coordinates name. So whenever the location changes, app searches for an image in the directory and displays it if found or else download if not found. This was a bad idea because in every meter there are thousands of different location values. Saving millions of images on the application directory is not a good idea for users and developers as well.

So unlike gird algorithm, we decided to address this issue in a different way. We have taken the integer part of the location and divided the decimal part into the range of 0.3.

So unlike gird algorithm, it has been decided to implement it as in server. The integer part of the location has been separated and divided the decimal part into the range of 0.3.

For example:

If our location is (53.555037, 10.022218),

The new location bound now is (53.3, 10.0) and (53.6, 10.3), where (53.6, 10.3) is the top left corner of the overlay and (53.3, 10.0) is the bottom right corner of the overlay.

Refer to server side documentation for more details.

The bitmap image to be downloaded will be named after the bound. For an instance, the image name on the above location will be **53.3_10.0_53.6_10.3.bmp**. This image will be saved in the application directory and will be loaded if we are inside the particular location boundary.

We have used regex to split the image name, so that the approximated bound can be compared with the coordinates separated from the image name. If they are same, image for overlay has been found in the directory.

```
I/System.out: DEBUG imageName to overlay is : 53.3_10.0_53.6_10.3.bmp
I/System.out: DEBUG: Hamburg Overlay found for: [53.3, 10.0, 53.6, 10.3]
I/System.out: DEBUG: Image name should be: 53.3_10.0_53.6_10.3
```

Pic. 13: android log showing overlay image information

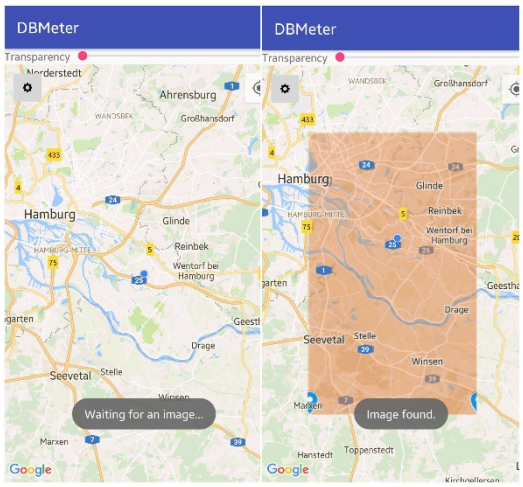
In Pic. 9, we have an image named 53.3_10.0_53.6_10.3.bmp in the directory. The coordinates are compared and expected image name without image extension is displayed (in figure 4 third line). This expected image name is generated from the approximation of location values. If we use this algorithm for the approximation, we will have two images for Hamburg.

```
//coordinates are changed to image name. if the image name is different, go to this loop.
if (!oldImageToOverlay.equals(imageToOverlay)) {
    if (imageFoundInDirectory) {

        if (mGroundOverlay != null) {
            mGroundOverlay.remove();
        }
        //codes for drawing overlay here
        oldImageToOverlay = imageToOverlay;
    }
    //if it is not is directory, download the image
    else {
        //waiting for the image download
        Toast.makeText(getApplicationContext(), "Waiting for an image...", Toast.LENGTH_LONG).show();
    }
```

Pic. 14: code for checking overlay

As shown in Pic. 10, location coordinates trigger the overlay on the map. If we change the location, coordinates will be analyzed in a certain interval of time. If the changed location expects a different bound, the image name will be changed as expected by new bound. This new image should be used for the overlay on that location. If we have a new image on the directory, it will be displayed otherwise image is downloaded from the server.



Pic. 15: drawing overlay algorithm

In Pic. 11, If the image is not available on directory it waits until the image is downloaded. If the image is available, it displays it as an overlay. Handler has been implemented that checks for an image continuously on a certain time interval in the background.

Server application (Hasan Hasanli)

Architecture

In this project the tasks of the server were:

- Define file structure for saving data
- Fetch, retrieve, convert data from query (coordinates, signal strength)
- File structure
- Updating data

Define file structure for saving data

For saving signal strength for each coordinate for whole Earth we are using .dat file. But we didn't store all signal strength values in one file because:

- for simplicity
- searching data in one huge file would take too much time and too much RAM
- easier for testing

Instead of saving data of whole Earth in one file we divided map to areas with ≈ 30 km x 30 km. After conversation from km to degrees:

30 km ≈ 0.3° longitude

30 km ≈ 0.3° latitude

For example:

for longitude: 32km = (53.571948, 9.735556; 53.558490, 10.221014)

for latitude: 30km = (53.681894, 10.000601; 53.415086, 9.980688)

We say each square is approximately 30 km x 30 km because we are operating not with distance, but with coordinates. And depending on position on the surface of Earth it can vary: near the equator 0.3 degrees corresponds to 33 km, and at poles it will be only 0.8 km.

So, each file contains value for 30 km2 in other words, it contains values for each 0.3 degree square. And each file has corresponding name in following format:

x.x_y.y.dat

For example:

53.6_10.3.dat

or for negative values:

53.6_n10.3.dat

where “n” stands for “negative”.

As we discussed we save data for each 0.3 degree square. It means the range between each decimal part of coordinates can be divided into 3 parts:

(a) 0 .. 0.3

(b) 0.300001 .. 0.6

(c) 0.600001 .. 0.999999

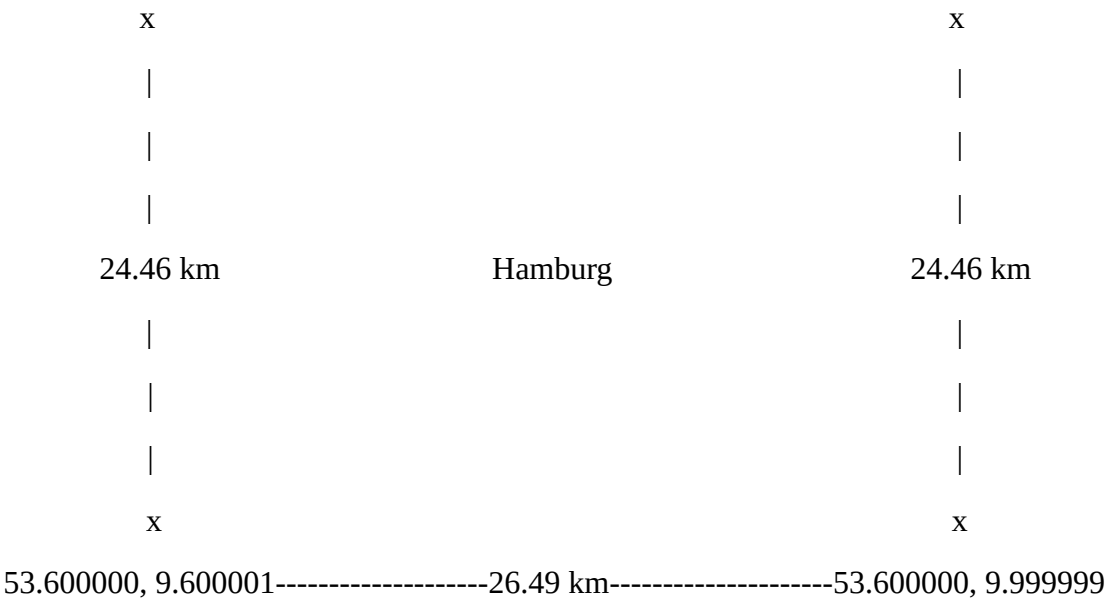
We use this logic in file names. If mantissa part of coordinate is in range (a) - after decimal part of coordinate we write 0.3. If in range (b) - after decimal part of coordinate we write 0.6. If in range (c) - 0.9.

For example, we received object’s coordinates: x = 53.571948, y = 9.735556. As you can see mantissa of x is in range (b) and mantissa of y in range (c).

It means signal strength will be saved in file named:

53.6_9.9.dat

This file name makes it easier to determine area and return layer. For example, area for Hamburg



Fetch, retrieve, convert data from query (coordinates, signal strength)

Server gets values of coordinates and signal strength from query in url.

For this purpose we use CGI. CGI is simply an interface between HTML forms and server-side scripts.

Example of query:

http://r1482a-02.etech.haw-hamburg.de/~w16cpteam1/cgi-bin/index?x=32.256488&y=-145.121515&s=100

Where:

- x=32.256488– longitude coordinate of object
- y=-145.1215151 – latitude coordinate of object
- s =100– signal strength

We retrieve x and y, convert from string to double and then separate decimal and mantissa parts to determine in which file data should be saved. Based on example above we will get:

- x_decimal = 32
- x_mantissa = 0.256488
- y_decimal = -145
- y_mantissa = 0.121515

File structure

As we discussed files will contain values divided by this range.

(a) 0 .. 0.3

(b) 0.300001 .. 0.6

(c) 0.600001 .. 0.999999

We store values for every 0.000150 degree and it means each range contains 0.3 / 0.000150 = 2000 values.

Therefore each file contains two dimensional integer array with 2000*2000 values (2000 for each longitude and 2000 for latitude). To determine position of value inside of array we use function locateStrength:

```
int * locateStrength(double x, double y){
    double step = 0.000150;
    static int coord_step_val[2];

    if ((x > 0.3) && (x < 0.6)){
        x = x - 0.3;
    } else if ((x > 0.6) && (x < 0.999999)){
        x = x - 0.6;
    }


    if ((y > 0.3) && (y < 0.6)){
        y = y - 0.3;
    } else if ((y > 0.6) && (y < 0.999999)){
        y = y - 0.6;
    }

    coord_step_val[0] =(int)floor(x/step);
    coord_step_val[1] =(int)floor(y/step);

    return coord_step_val;
}
```

Function as an arguments gets mantissa of x and y, determines to which range they belong and divides mantissa part by step value (0.000150). Then casts it to integer. Final value is the position of signal strength in array.

We use this function also for updating value in an array.

Updating data

For updating value of signal strength in array we are using averaging formula:

new_value = 0.2 * old_value + 0.8 * new value.

Lessons Learned (Maksud Ganapijev)

As in the end every project, after totaling all the pulses up, team members make conclusions of what went well, wrong and what could have been done better.

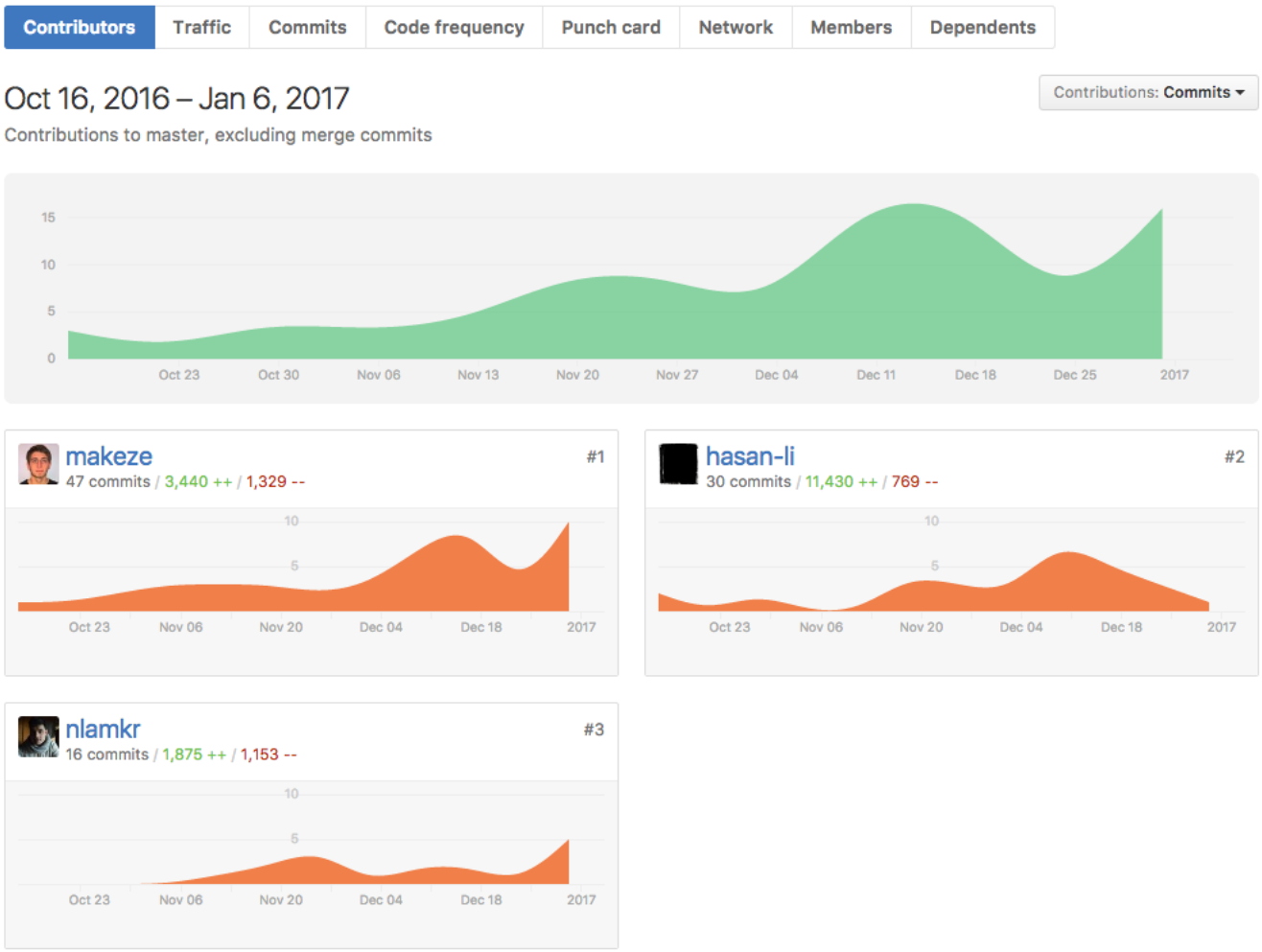
A list of problems that a team faced is introduced below.

What went wrong

Motivation. Every team member showed up at least one week unprepared, or with problems from the last week. Not being able to solve problems and not wishing for assistance leads to the next common problem.

Communication. Even though it is 2017, team member still manage to get lost for several days with no response on progress.

Time-management. Relaxed attitude at the beginning of the project, stressful tensed work, when getting closer to deadline. GitHub platform very well illustrates this problem.



Pic. 16: Amount of contributions, against time

Disinformation. Claiming feature to be working without testing it. Even worse when feature is implemented in own manner, without first discussing it with other team members.

Inexperience in technology. Every team member was at point, when a problem at start looked easy, however with progress an understanding would come, that a way chosen will not work in the end, which forced one to start over, waste time. To some extent it is understandable, since many courses for professional preparation are usually at least 1 year long.

Overloading. Task complexity miscalculation lead to some of team members being overwhelmed with tasks, as a consequence broken deadlines.

What went right

Code base organization. There is no a single complain on code base organization. Git repository was setup in one week and for all the time of project, the system was saving large amounts of time on code merging, upload, recovery and transportation.

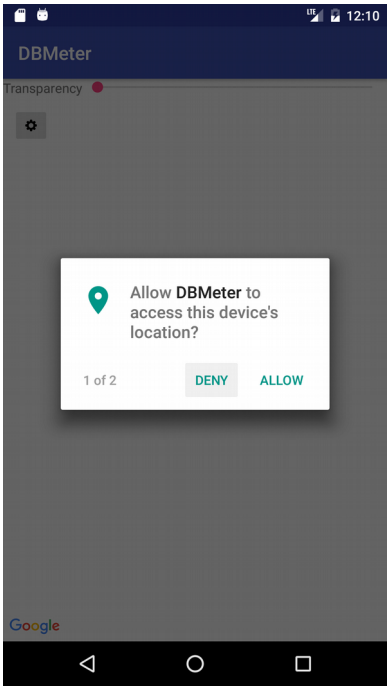
Knowledge gained. Since the team consisted only out of 3 members, every team member had a large set of tasks to work on, which forced one to gather information on subject.

Responsibility. Again a small group played in favor to each team member. Every task assigned was important and not completing it would affect results of all the team.

Feeling of work. One gets in working rhythm after working on such a projects. For some members for sure, it have woken up an appetite for creating more.

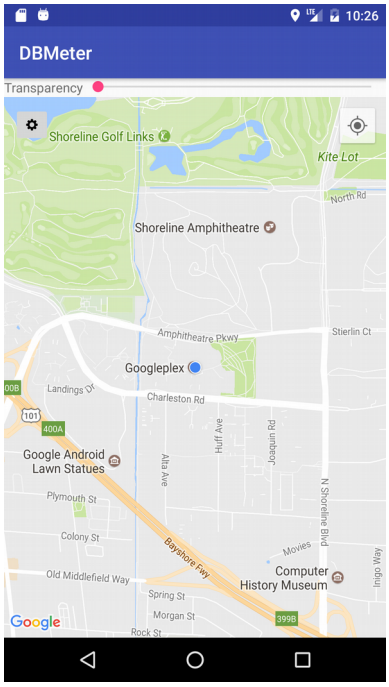
Demo description

After installing the app. An application will greet user with a Permission Requests dialogue, since some sensitive information is being collected.



Pic. 17: On first start

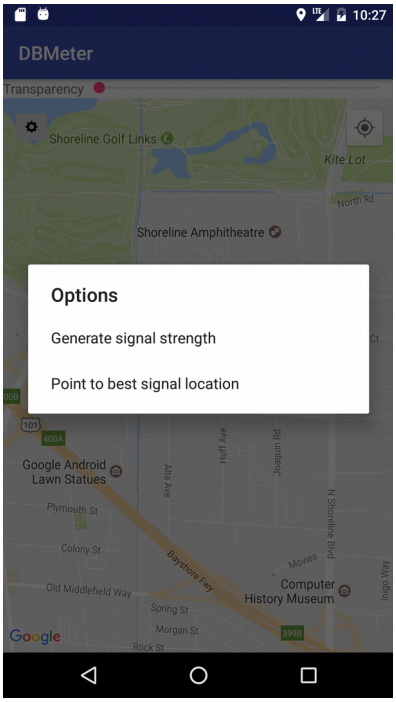
A main map will appear with two buttons, and “Transparency” scroller.



Pic. 18: mainActivityView

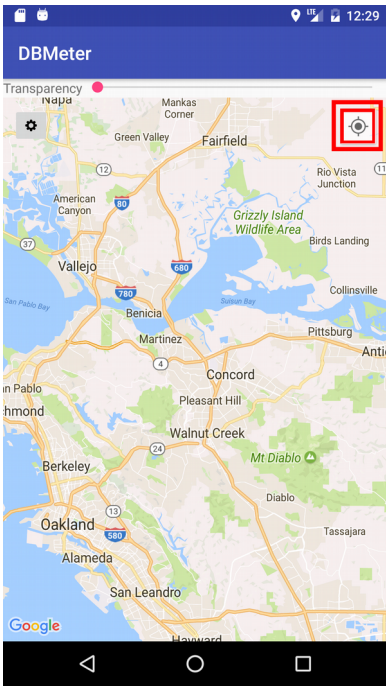
Button with “Gear” icon, on a long press creates a floating menu, with list of two function calls.

- Generate signal strength heat-map, draws a signal strength heat-map on top op original map. If map is not present, heat-map is requested from server.
- Point to best signal location, marks best signal location on a map and an arrow to it.
- Transparency scroll bar, changes transparency of a heat-map.



Pic. 19: mainActivityView

Red marked button on the top right will show current location and focus map view to it



Pic. 20: mainActivityView