MICS 2017

Luxembourg City Bus and Bike

A project for Principles of Software Development

BELIAKOV Alexander, KHRAMSTOVA Ekaterina
1-27-2017

About

"Luxembourg City Bus and Bike" is Android application is intended to look for closest bus stops and to show the schedule of buses. The application must have the graphical interface with the interactive map. The application must can keep user's queries.

Functional requirements

The application has to give the next user possibilities:

- show the current user on the map;
- show all bus stops on the map:
- the map should be interactive (user can to walk by map, change zoom etc.);
- the search of the closest bus stop to the user location;
- the search of the all bus stops for selected distance (XX meters);
- show information about future buses on the selected bus stop (bus number, time of arrival, direction);
- keep user queries;
- show all veloh stations on the map;
- find the route to a chosen point.

Technical requirements

The application must be designed for Android OS, the minimal supporting version is 4.0 (IceCreamSandwich).

The data about buses can be get from Mobiliteit API.

The database is default Android SQLite Database.

Interface requirements

There are the next requirements for the interface:

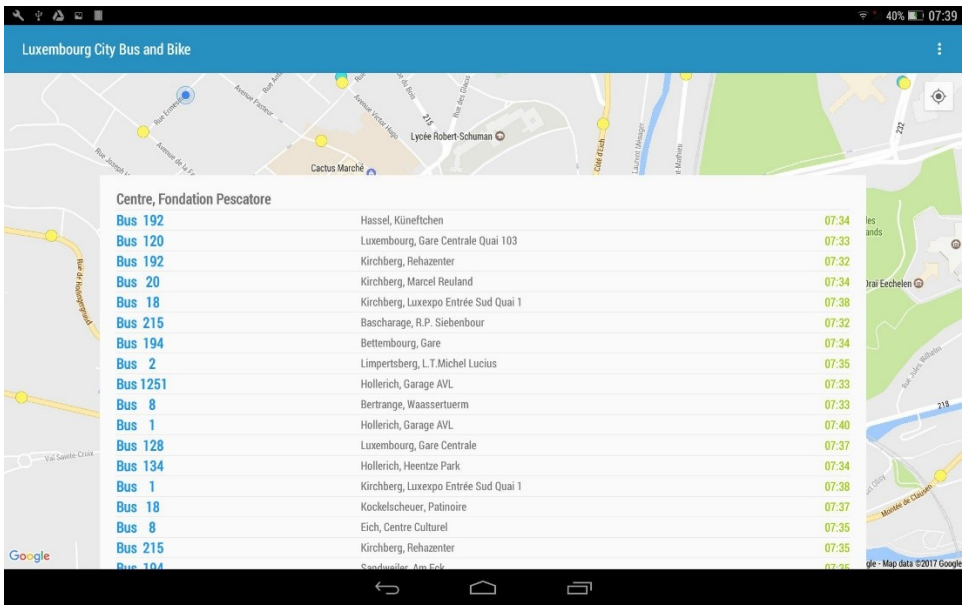
- the application must support different types of display resolution;
- the application must have a usability interface;
- the graphical interface should be beautiful;
- all buttons and elements of the interface must be interactive and understandable;
- all graphic elements should be doubled by text information;
- the graphical interface should be created in traditional Luxembourgish colors (blue or red).

Secure requirements

There are not special secure requirements.

Presentation of App

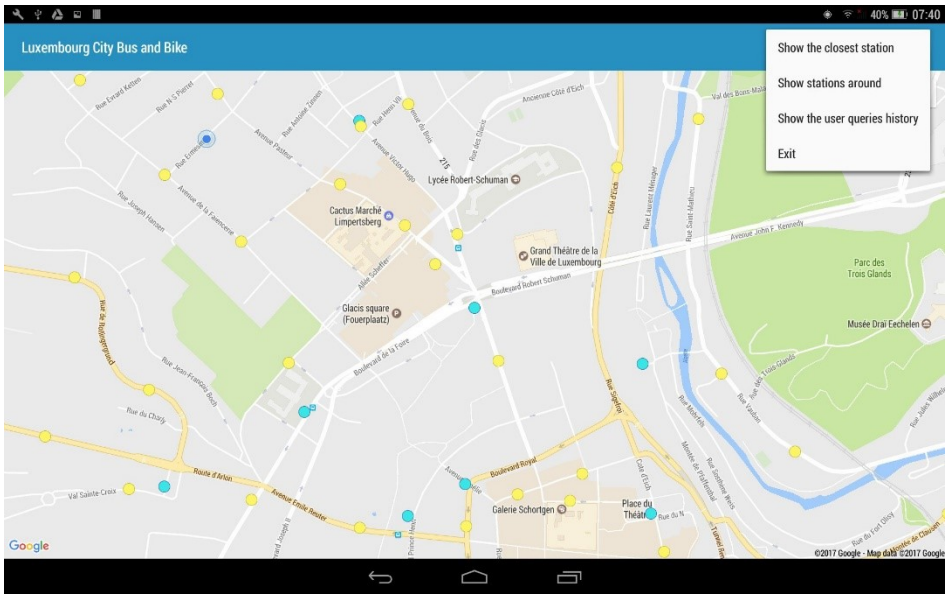
After launching the application, the map appears on the screen. The user's location is indicated by a dark blue point, bus stops are represented as yellow points, bike – stations as blue points. By pressing on one of the stops, we can see the information about the arriving buses, including their number, destination and the time of departure from this stop:



The same with vello stations – after pressing the point a popup window appears with the name of the station and the information about available bikes.

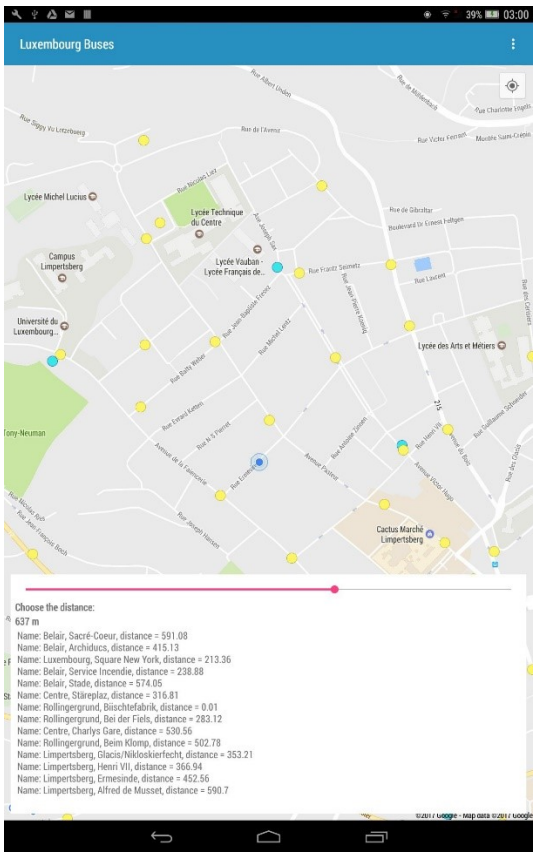
Zooming in and zooming out, as well as shifting of the map is available using fingers moves. If we zoom out too much, the stations will disappear in order to save the clarity of the map (otherwise, we don't see anything but rounds).

On the top right corner of the map there is a symbol of menu. It consists of 4 items:

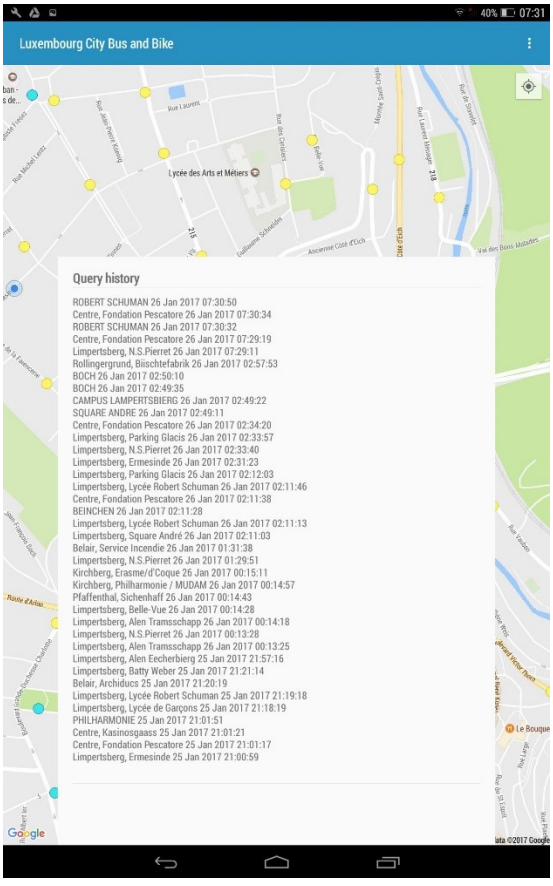


By pressing on the first item, the information about the closest station and the buses, which will soon depart from this station, will appear.

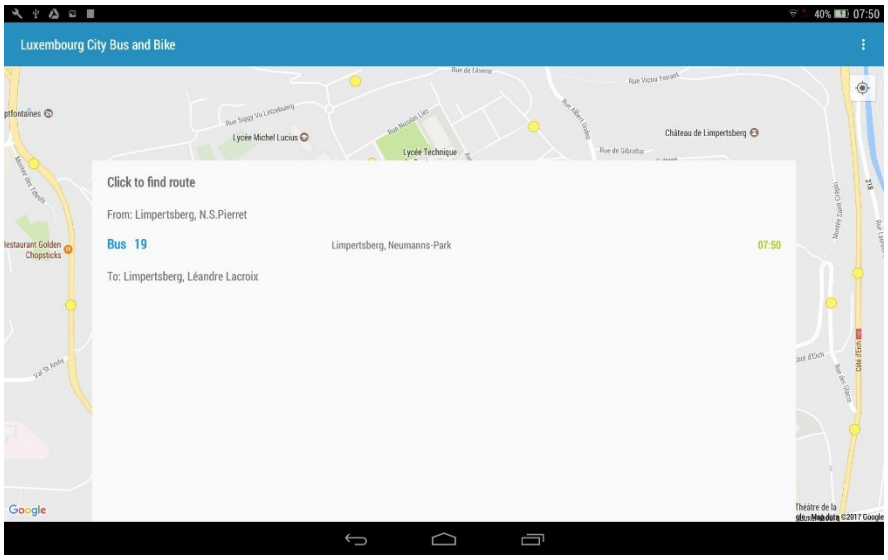
“Show stations around” is a dialog window to show stations around a user by the chosen radius, distance. The user could see all information about stations around as names and the distances between user’s location and this stations.



User queries is the list of bus stations that a user has looked before.



Finally, by a long push on some point in the map, the route building activity will start. It calculate, what bus is preferable to take, from which station start and where to get off, and the nearest time of this bus's departure:



Architecture

In our project we tried to separate and structure our code as optimal as possible, logically dividing the code in different classes.

To facilitate the representation of information and to show the knowledge of Object Oriented Programming, there were implemented two classes: Station and Bus.

Station.java

```
public class Station{
    String name;
    double longitude;
    double latitude;
    double distance;
    Marker marker;
…
}
```

This class collects information about stations: name, map coordinates, distance and special field Marker.

Bus.java

```
public class Bus {
    String name;
    Date datetime;
    Date rtDatetime;
    String direction;
    String station;
    String user_station;
…
}
```

Class, represented above, collects information about buses: name; date and time of departure; name of station, from where the departure is. Parameters station and user station are to compile the route : stop for the user, where he have to take the bus and the station to get off the bus.

Manifest

The manifest file provides essential information about application to the Android system, which the system must have before it can run any of the application's code. This file has high importance, so let's analyze it precisely.

Our manifest file contain three permissions:

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

First two are to access the user's current position, the last – to connect to the Internet.

Next we have the declaration of the application. When the application process is started, class AppController is instantiated before any of the application's components. In particular, this class is added to implement the Volley algorithm.

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="Luxembourg Buses"
    android:supportsRtl="true"
    android:theme="@style/AppTheme"
    android:name="com.example.bsanc.luxembourgishbuses.app.AppController">
```

Additional data is added by meta-data tag, it serve to store the APY-key to connect with google maps.

```
<meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value="@string/google_maps_key" />
```

Finally, we have the declaration of all our activities :

```
<activity
    android:name=".MapsActivity"
    android:configChanges="orientation|screenSize">
    android:label="@string/title_activity_maps">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

<activity android:name=".BusStopPop"
    android:theme="@style/AppTheme.CustomTheme"
    ></activity>

<activity android:name=".RouteFinding"
    android:theme="@style/AppTheme.CustomTheme"
    ></activity>

<activity android:name=".QueryHistory"
    android:theme="@style/AppTheme.CustomTheme"
    ></activity>
```

Activity

There are 4 activities in the project:

- Bus Stop Pop
- Query History
- Route Finding
- Maps Activity

Maps Activity

```
public class MapsActivity extends AppCompatActivity implements OnMapReadyCallback,
ConnectionCallbacks, OnConnectionFailedListener, OnMarkerClickListener,
GoogleMap.OnMapClickListener, GoogleMap.OnMapLongClickListener,
SeekBar.OnSeekBarChangeListener {
private GoogleMap mMap;
private ArrayList<Station> myStations = new ArrayList<Station>();
private ArrayList<Marker> myMarkers = new ArrayList<Marker>();
private ArrayList<Marker> myVeloh = new ArrayList<Marker>();
private static final int MY_PERMISSION_ACCESS_COARSE_LOCATION = 11;
GoogleApiClient mGoogleApiClient;
protected Location mLastLocation;
protected double mLatitudeText;
protected double mLongitudeText;
//Initialize to a non-valid zoom value
private float previousZoomLevel = -1.0f;
private boolean isZooming = false;
public Marker currentMarker;
public int currentMarkerType;
private static final int ID_BUS_STATIONS = 0;
private static final int ID_VELOH_STATIONS = 1;
private static final String TAG = "Volley";
private TextView mTextValue;

protected void onCreate(Bundle savedInstanceState) {}
private void seekBarRun() {}
public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {}
public void onStartTrackingTouch(SeekBar seekBar) {}
```

```
public void onStopTrackingTouch(SeekBar seekBar) {}
public boolean onCreateOptionsMenu(Menu menu) {}
public boolean onOptionsItemSelected(MenuItem item) {}
private void showStationsAround() {}
public void onConnected(Bundle connectionHint) {}
public OnCameraChangeListener getCameraChangeListener() {}
public void getClosestStation() {}
private void drawBusStations() {}
private void addBusStopMarker(Station station, int id) {}
private BitmapDescriptor getBitmapDescriptor(int id) {}
private ArrayList<Station> getBusStationsList(String filename) {}
public boolean onMarkerClick(final Marker marker) {}
public void writeToFile(Marker marker) {}
public void openStation(Marker marker) {}
public void onActivityResult(int requestCode, int resultCode, Intent data) {}
public void onConnectionSuspended(int cause) {}
public void onConnectionFailed(ConnectionResult result) {}
protected void onStart() {}
protected void onStop() {}
public void onMapReady(GoogleMap googleMap) {}
public void onMapClick(LatLng latLng) {}
public void onMapLongClick(LatLng point) {}
private void bestRouteFinding(LatLng point) {}
public void getCurrentLocation(GoogleMap googleMap) {}
public String loadJSONFromAsset(String file_name) {}
public void Volley_json(double dist){}

}
```

“Maps Activity” is the main activity of the project. The activity shows the map interface and enables a user to work with other components of the application.
The main view of the activity is the Google map interface with an Action bar.
Action bar represents a top line, with a name of the application and menu button.
Menu button allows a user to get quick access to different parts of the application:

- show the closest station;
- show stations around;
- show the history of user requests;
- exit (finish of the application).

The activity receives a few events:

- getCameraChangeListener;
- onMarkerClick;
- onMapClick;
- onMapLongClick.

The events allow to work not only with the map directly but and with objects on it.
The activity contains the next objects:

- bus stations;
- bike stations (veloh stations).

All objects of the map are clickable and call new activities.
A component of the application “show stations around” works directly in the Maps Activity. When a user wants to look all stations near him, he can just call the module from the main menu.
This action opens a layout with the module data in the form of a pop-up window. Then, the user can change a position of seek bar cursor and gets different information according to requests.

Bus Stop Pop

```
public class BusStopPop extends Activity {

    private static int station_type;
    protected void onCreate(Bundle savedInstanceState) {}
    private void getBusStationsList(double latitude, double longitude) {}
    protected void onStop() {}
```

```
}
```

The activity shows information about selected marker (station):

- name of the station;
- future buses with information about (number of a line, direction, time of arrival) if there are.

To receive the information about buses, the activity calls WebRequest class in asynchronous mode. When methods of the class are executed, BusStopPop activity displays received information.

RouteFinding

```
public class RouteFinding extends Activity {
    protected void onCreate(Bundle savedInstanceState) {}
    private void getAsyncRequest(double point_latitude, double point_longitude, double
user_latitude,  double user_longitude){}
}
```

The activity shows the shortest way to reach a destination point from current user position. The RouteFinding activity can be called from the main menu of the application.
The activity display the next information:

- a station, from which a user should take a bus;
- information about the bus (number of like, direction, time of arrival);
- the final station of the destination point.

The activity receives the information from the AsyncRouteFindingRequest class by using asynchronous mode of message sending.
When the class executes, the activity displays supporting information about the data loading.

QueryHistory

```
public class QueryHistory extends Activity {
    protected void onCreate(Bundle savedInstanceState) {}
    private String getQueriesList(){}
}
```

The class to get list of a history of user queries from Internal storage. The class reads information from a file with query history and displays it in the activity.

Request algorithms

All the information about busses and vello stations is dynamically obtain from the Luxembourg data platform https://data.public.lu/en/. This is an API which allows to know the list and location of public transport stops, the timetable of the busses and a lot of additional information, given by a certified public service. The result of the request is represented by JSON response, which hereafter have to be parsed.

When looking to parse JSON for use with Android, many different ways of doing so can be found.

For the first time to put all the stations into the map it is more rational to do the request just once, because the position of stops is statistic. As long as the JSON request is too volume, separate files, called “stations” and “veloh.json”, were created. After parsing this files, bus and vello icons are putted into the map; they are not redrawn after launching another activities.

For the rest the dynamic requests were used. However, in our project different methods and tools for their realization are different.

The first way is represented in files Web Request and AsyncRouteFinding. For connection to the Net java.net.HttpURLConnection and java.net.URL libraries are included. Received data is temporary stored in BufferedWriter.

Another way is shown in Volley_json function Maps.activity.java class. Volley is a great choice as it runs network operations asynchronously by default, so we don't have to worry about using asynctask anymore. In addition, Volley can handle request queuing and cancellation simply and effectively, effective manage request cache and memory.

Both solutions are asynchronous (first case - extends AsyncTask, second case – by default), so even if the computation takes some time, it doesn't affect much the efficiency of other activities.

Testing

Unit testing

For testing there was used JUnit Test Framework. JUnit is a Regression Testing Framework used by developers to implement unit testing in Java, and accelerate programming speed and increase the quality of code. JUnit Framework can be easily integrated with an application code.

In the result was written 11 tests for different methods of application classes.

The example of a written test:

```
public class Tests {
   String json =
"{"Departure":[{"JourneyDetailRef":{"ref":"1|141|6|82|27012017"},"Product":{"
name":"Bus   13","num":"2208","line":"13","catOut":"Bus
","catIn":"AVS","catCode":"5","catOutS":"AVS","catOutL":"Bus
","operatorCode":"AVL","operator":"Ville de Luxembourg - Service des
Transports en Commun","admin":"AVL---"},"name":"Bus
13","type":"S","stop":"Belair, Sacré-Coeur","stopid":"A=1@O=Belair, Sacré-
Coeur@X=6113204@Y=49610280@U=82@L=200403005@","stopExtId":"200403005","progno
sisType":"PROGNOSED","time":"08:38:00","date":"2017-01-
27","rtTime":"08:43:00","rtDate":"2017-01-
27","direction":"Luxembourg/Strassen, Centre
Hospitalier","trainNumber":"2208","trainCategory":"AVS"}]}";
   MessageUtil messageUtil = new MessageUtil(message);
   @Test
   public void testParseJSON() {
      System.out.println("Inside testPrintMessage()");
      assertEquals(json, messageUtil.ParseJSON());
   }
}
```

In the result was written 11 tests for different methods of application classes.

Test runner is used for executing the test cases. Here is an example of test launching.

```
public class TestRunner {
   public static void main(String[] args) {
      Result result = JUnitCore.runClasses(TestJunit.class);
      for (Failure failure : result.getFailures()) {
         System.out.println(failure.toString());
      }
      System.out.println(result.wasSuccessful());
   }
}
```

In the result of program working, all tests ware executed, overall time of test executing is 15.45 seconds.

Alpha testing

For the alpha testing there was held a user testing by a group of target users.

Users got the application for trying to use it in real cases. In the result of alpha-testing was detected:

- scroll problem (lack of usability for using);
- omission of notification for data loading;
- crash of the application when received data are empty.

The problems have been eliminated, and speed of application working was increased.

Future work.

As in every application, some improvements could be suggested.

The main improvement would be to give the opportunity to redirect to selected station from auxiliary activities. For example, in the popup menu “show stations around” it is useful not only to show the list of closest stations, but the possibility to chose one of them and to see the timetable of nearest buses from this station. In more general way, in our application the communication between the auxiliary activities is not totally developed.

We would like also to improve the style of main menu, the popup menus etc.

Finally, some computations in some devices take too much time, the optimization will not be superfluous.