

ANDROID PRACTICE

Mobile Device Applications

Course 19-20

- 4 sessions

DESCRIPTION

In this practice, you will develop an application for Android devices. Specifically, the application will be about Valenbisi, the bike rental system of the City of Valencia. The app will allow users to check the status of the different bike stations and to create incident reports. We will use the services of the City Council of Valencia to retrieve data of Valenbisi bike stations. This query will initially be carried out using a JSON file that can be requested from the web service of Valencia's open data website. However, in the first part, we will work with internal files in order to simplify: the JSON file with the complete list of bike stations will be downloaded and copied to the project. On the list of bike stations, we will show the information about each of them. Next, we will create a local database that will allow the creation of new incident reports associated with each of the bike station. Finally, we will perform the query about the Valenbisi bike stations through the web service, in order to display real-time data to our users. This practice will be developed using Android Studio. The next link shows a simplified version of the application once finished (The look can be different): <http://ir.uv.es/mX01NCc>

GOALS

The goals of the practice are the following:

- Work with JSON format files.
- Work with Activities and Intents.
- Work with ListView and Adapter.
- Develop layouts to display the different elements of the user interface.

- Create a local SQLite database.
- Manage databases, including select, insert, update and delete operations.
- Work with HTTP services by using JSON.
- Work with data from sensors.

ANDROID PROJECT CREATION

Once opened Android Studio, we will create a project called **Valenbisi** with the domain **<studentname1>.<studentname2>.uv.es**. We will select the option application for Phone or Tablet compatible with the **API 23**. Finally, we will add an empty activity called **"ListaParadas"** (BikeStationsList)

ORGANIZATION OF THE PRACTICE

In order to develop the application, the practice is organized in four sessions according to the four sections of this guide. In each session, there is a mandatory development and an optional development. In order to opt to the maximum mark, all the optional part must be developed.

SESSION 1: LIST OF VALENBISI BIKE STATIONS

In this first session we will focus on reading the JSON file that will have the data for the *ListView* control, where the bike stations will be displayed. The file can be downloaded at the following link:

➤ <http://mapas.valencia.es/lanzadera/opendata/Valenbisi/JSON>.

This file should be saved in the application internal storage; this implies that we must save it in the appropriate resource folder of the Android project. Figure 1 shows an example of the list of Valenbisi bike stations as a result of this session:



Valenbisi		
268	0	1330.5001m
Plaza Luis Cano, 5		
269	0	1425.9086m
Campamento, 81		
267	0	2574.508m
Beniferri - Vicent Tomás Martí		
239	0	2702.6716m
Florista - T4 (Palau de Congressos)		
240	0	2751.7207m
Camp del Turia - Corts Valencianes		
241	0	2896.5447m
La Vall d'Albaida - Corts Valencianes		
266	0	2909.0542m
Canal de Navarrés - Maestro Rodrigo		
270	0	3019.8542m
Ninot - Regino Mas		
228	0	3030.1072m
Doctor Nicasio Benlloch - L'Horta Sud		
238	0	3057.1106m
San Jose Artesano - Francisco Morote Greus		
227	0	3077.629m

Figure 1 List of Valenbisi bike stations

First, we will create a *ListView* into the *Activity*, which will be fed with a class that extends from *BaseAdapter*. The class structure is as follows:

```
public class AdapterParadas extends BaseAdapter {
    private ArrayList<Parada> paradas;
    Context context;

    static class ViewHolder {
        TextView number;
        TextView address;
        TextView partes;
    }

    public AdapterParadas(Context c)
    {
        context=c;
        Init();
    }

    public void Init() {
        // We read the JSON file and fill the "paradas" (bike stations) ArrayList
    }

    @Override
    public int getCount() {
        return paradas.size();
    }

    @Override
    public Object getItem(int position) {
        return paradas.get(position);
    }

    @Override
    public long getItemId(int position) {
        return paradas.get(position).number;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {

        // We use the ViewHolder pattern to store the views of every list element
        //to display them faster when the list is moved
        View v = convertView ;
        ViewHolder holder = null;
        if ( v == null ) {
            // If is null, we create it from "layout"
            LayoutInflater li =
                (LayoutInflater)context.getSystemService(Service.LAYOUT_INFLATER_SERVICE);
            v = li.inflate (R.layout.listparadaview , null ) ;
            holder = new ViewHolder ( ) ;
            holder.number = ( TextView ) v.findViewById (R.id.paradaviewnumber ) ;
            holder.address = ( TextView ) v.findViewById(R.id.paradaviewaddress ) ;
            holder.partes = ( TextView ) v.findViewById(R.id.paradaviewpartes ) ;
            v.setTag ( holder ) ;
        } else {
            // If is not null, we re-use it to update it.
            holder = ( ViewHolder ) v.getTag ( ) ;
        }

        //Fill in the "holder" with the bike station information which is in the
        // position "position" of the ArrayList
    }
}
```

Once the *Adapter* is created, an analysis of the file structure must be carried out for parsing it later. It is recommended to use the *org.json.JSONArray* and *org.json.JSONObject* classes for parsing. Initially we will read the file and pass it to a String.

```
public void Init() {

    paradas=new ArrayList<Parada>();

    InputStream is = context.getResources().openRawResource(R.raw.paradasvalenbici);
    Writer writer = new StringWriter();
    char[] buffer = new char[1024];
    try {
        Reader reader = new BufferedReader(new InputStreamReader(is, "UTF-8"));
        int n;
        while ((n = reader.read(buffer)) != -1) {
            writer.write(buffer, 0, n);
        }
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        try {
            is.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    //The String writer.toString() must be parsed in the bike stations ArrayList by
    using JSONArray and JSONObject

}
```

As noted in the code, it is necessary to create a layout for each of the items in the list that will display bike station number and name information. There is also a third field that shows the number of free bikes on the bike stations. This number or, alternatively, the background, should appear in green color if the number is greater than 10 bikes, in orange if the number is between 5 and 10 bikes and in red color if the number is less than 5 bikes.

OPTIONAL DEVELOPMENT:

As an optional development of this session, we propose to order the list of bike stations by proximity to a fixed location. For simplicity in this session, we will consider a fixed location represented by the latitude and longitude coordinates. The value of the coordinates can be established in a variable. For example, the ETSE location can be established as follows:

```
Location ETSEPoint=new Location("locationETSE");
ETSEPoint.setLatitude(39.512634);
ETSEPoint.setLongitude(-0.424035);
```

The JSON provided by the city of Valencia returns the location in UTM coordinates (c1 and c2), while the geolocation service of Google Maps needs latitude-longitude. The calculation to move from one system to another is quite complex, so is better to use an external library that you can download from:

➤ <http://www.jstott.me.uk/jcoord/jcoord-1.0.jar>

The code to change from one system to another:

```
import uk.me.jstott.jcoord.*;

UTMRef utm = new UTMRef(coordinate1, coordinate2, 'N', 30);

Location ParadaPoint = new Location("locationA");
ParadaPoint.setLatitude(utm.toLatLng().getLat());
ParadaPoint.setLongitude(utm.toLatLng().getLng());
```

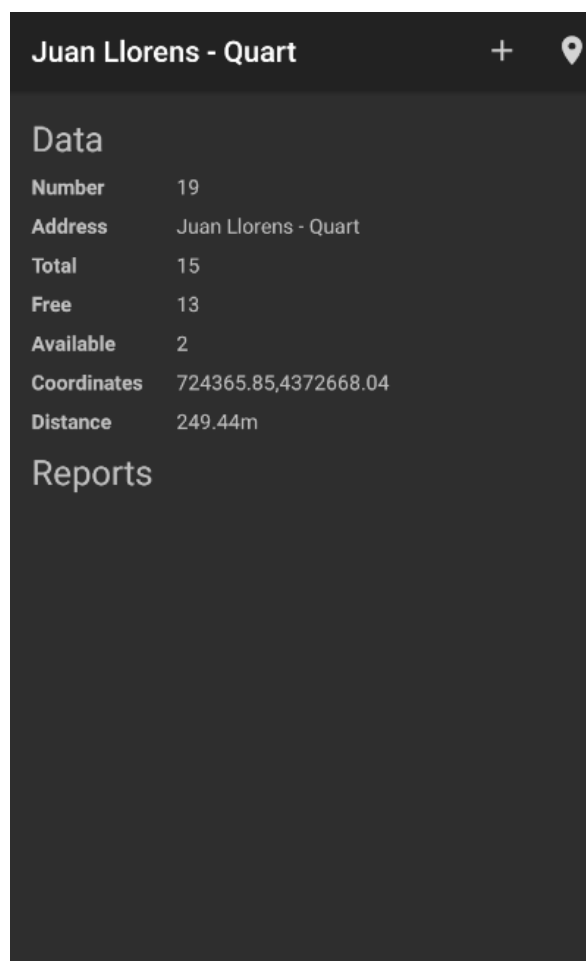
In order to calculate the distance between a bike station and the fixed location the following code can be used:

```
double distance=ParadaPoint.distanceTo(ETSEPoint);
```

SESSION 2: DETAIL OF THE BIKE STATION

In the second session, we will create a new empty activity that will show the information for each of the bike stations. This activity will be called by clicking each of the items in the List of Bike Stations activity list. This activity must be passed as an extra in the Intent. The activity title must be changed to the name of the consulted bike station. The data table should display the following information within a *GridLayout*:

- Bike station number
- Address
- Total bikes
- Available bikes
- Free
- Coordinates



Juan Llorens - Quart	
Data	
Number	19
Address	Juan Llorens - Quart
Total	15
Free	13
Available	2
Coordinates	724365.85,4372668.04
Distance	249.44m
Reports	

Figure 2. Detail of the bike station

At the bottom we will add a menu in the action bar with two items. The first item will allow to add an incident report, by starting a new activity. This will be developed in the next session. The second item must create a Google Maps activity to display the bike station position on the application (see an excerpt of the code below).

```
Uri gmmIntentUri;
gmmIntentUri = Uri.parse("geo:0,0?q="+altitud+", "+longitud+"("+nombrepareda+")");
Intent mapIntent = new Intent(Intent.ACTION_VIEW, gmmIntentUri);
mapIntent.setPackage("com.google.android.apps.maps");
//If is installed in the application, the Google Maps activity is launched
if (mapIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(mapIntent);
}
```

In order to create the menu, you must define a menu and all its items in an XML menu resource. Here's an example menu named game_menu.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/new_game"
        android:icon="@drawable/ic_new_game"
        android:title="@string/new_game"
        android:showAsAction="always"/>
    <item android:id="@+id/help"
        android:icon="@drawable/ic_help"
        android:title="@string/help" />
</menu>
```

Then, you must override onCreateOptionsMenu() method, in order to inflate your menu resource XML file. Here's an example:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.game_menu, menu);
    return true;
}
```

Finally, in order to handle click events in the menu (when the user selects an item from the options menu), the system calls your activity's onOptionsItemSelected() method. This method passes the MenuItem selected. You can identify the item by calling getItemId(), which returns the unique ID for the menu item. Here's an example:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
        case R.id.new_game:
            // Do something when the user clicks on the new game
            return true;
    }
}
```



```
        case R.id.help:
            // Do something when the user clicks on the help item
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

You can also check the Android documentation for further information:

➤ <https://developer.android.com/guide/topics/ui/menus#options-menu>

OPTIONAL DEVELOPMENT:

As an optional development of this session, we propose to show the distance of the bike station to our location (a fixed location established in the first session).

SESSION 3: INCIDENT REPORTS

In the third session, we will implement a local report management system to the application that will be stored in an **SQLite database**. A report must have the following information:

- Name
- Description
- Bike station
- Status:
 - Open
 - Processing
 - Closed
- Type:
 - Mechanical
 - Electric
 - Painting
 - Masonry

A class that extends from *SQLiteOpenHelper* must be implemented. This class will implement the following functions to work with the database:

- *InsertReport*
- *UpdateReport*
- *DeleteReport*
- *FindReportByBikeStation*
- *FindReportByID*

The functions that return reports must return a *CURSOR* with the data of the executed *SELECT*. In order to simplify the database design, the state and type fields could be defined like a string of chars. In addition, it is recommended the creation of an ID column like primary key in the table, in order to index and to find a report every time is needed.

In the Bike Station Detail layout must be inserted a *ListView* that displays the reports associated with the current bike station. The *ListView* must be fed with a class that

extends from *CursorAdapter*. The following code shows an example about how must be initialized both the *Cursor* and the *Adapter*:

```
PartesDbHelper db = new PartesDbHelper(getApplicationContext());
Cursor partesByParada = db.getPartesByParada(p.number);
partesAdapter = new PartesCursorAdapter(getApplicationContext(), partesByParada);
ListView lv=(ListView) findViewById(R.id.idlistapartesdetalle);
lv.setAdapter(partesAdapter);
```

We will create a *Listener* to open a new activity by clicking on each report of the *ListView*. This new activity will display detailed information about the report. This activity will be useful both display an existent report details and for create a new one.

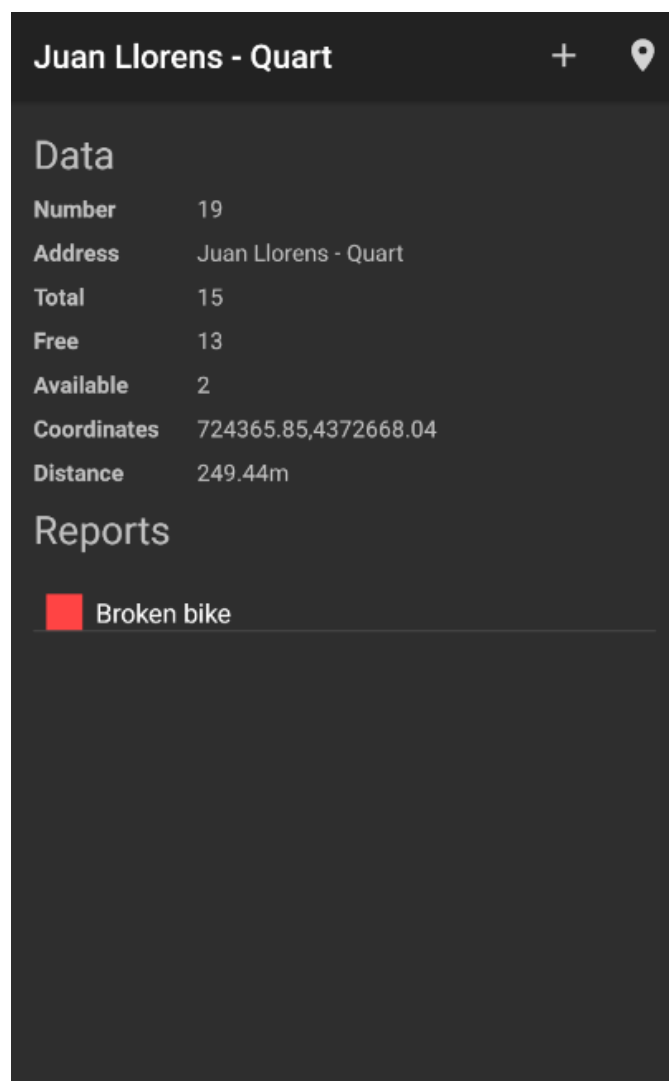


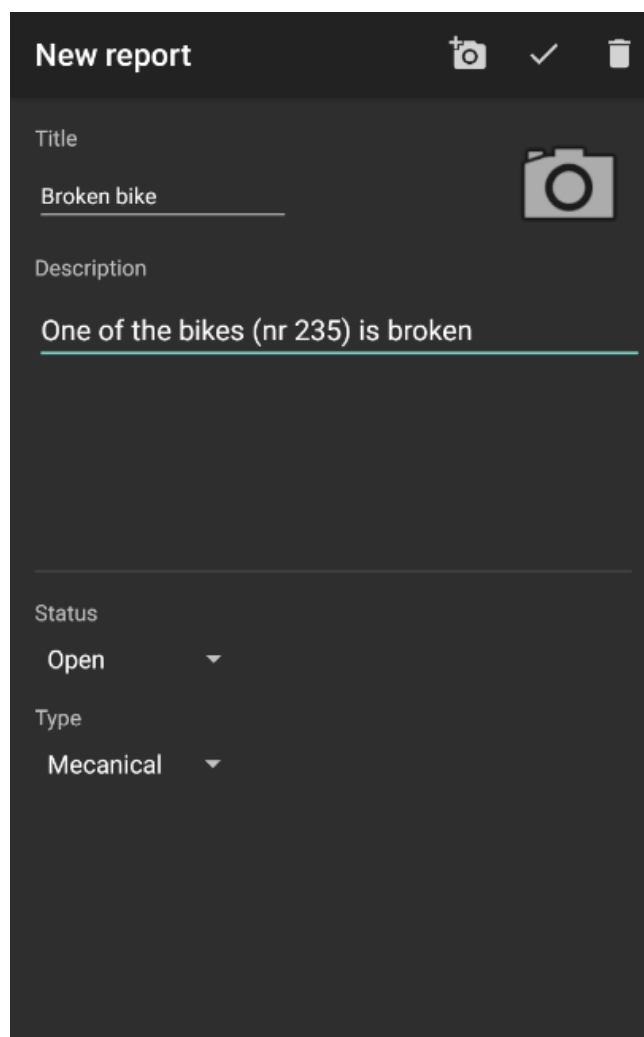
Figure 3. Detail of the bike station with reports

The way to know if we are creating a new report or we are updating it will be checking the extra that receives the activity. If it receives an integer with the report ID

will be an update. Otherwise, it will be a new report creation and it will receive an extra with the bike station, which the report will be associated.

The elements of the Detail of the Report layout are the following:

- Report subject
- Description
- A spinner with the report status
- A spinner with the report type
- A menu containing:
 - An item to insert/update the report
 - An item to delete the report



The screenshot shows a mobile application interface for creating a new report. The title bar at the top is dark grey with the text 'New report' in white. To the right of the title are three icons: a camera with a plus sign, a checkmark, and a trash can. Below the title bar, there are four main sections. The first section is labeled 'Title' and contains a text input field with the value 'Broken bike' and a camera icon to its right. The second section is labeled 'Description' and contains a text input field with the value 'One of the bikes (nr 235) is broken'. The third section is labeled 'Status' and contains a dropdown menu with the value 'Open'. The fourth section is labeled 'Type' and contains a dropdown menu with the value 'Mecanical'.

Figure 4. Detail of the report

OPTIONAL DEVELOPMENT:

As an optional development of this session, we propose to take a photo of the incident and attach it to the incident report. Android offers the possibility to retrieve a thumbnail or full-size photo. In this case, we will just keep the thumbnail and store it in our local database.

In order to do that we shall:

- (1) Add a new button view (to launch the camera app) and an image view (to display the thumbnail).
- (2) Add a new BLOB column in our database, to store the data from thumbnail.

Here's an example of code that launches an intent to capture a photo:

```
static final int REQUEST_IMAGE_CAPTURE = 1;

Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
if (takePictureIntent.resolveActivity(getPackageManager()) != null) {
    startActivityForResult(takePictureIntent, REQUEST_IMAGE_CAPTURE);
}
```

After taking the picture, the thumbnail is delivered to our application in the return Intent to *onActivityResult()*, under the key "data" of the extras. Here's an example of code:

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == REQUEST_IMAGE_CAPTURE && resultCode == RESULT_OK) {
        Bundle extras = data.getExtras();
        Bitmap imageBitmap = (Bitmap) extras.get("data");
        imageView.setImageBitmap(imageBitmap);
    }
}
```

In order to store the Bitmap provided by the camera app into the database you should convert it to a byte array, and the other way round when reading it from the database you should convert it to a Bitmap.

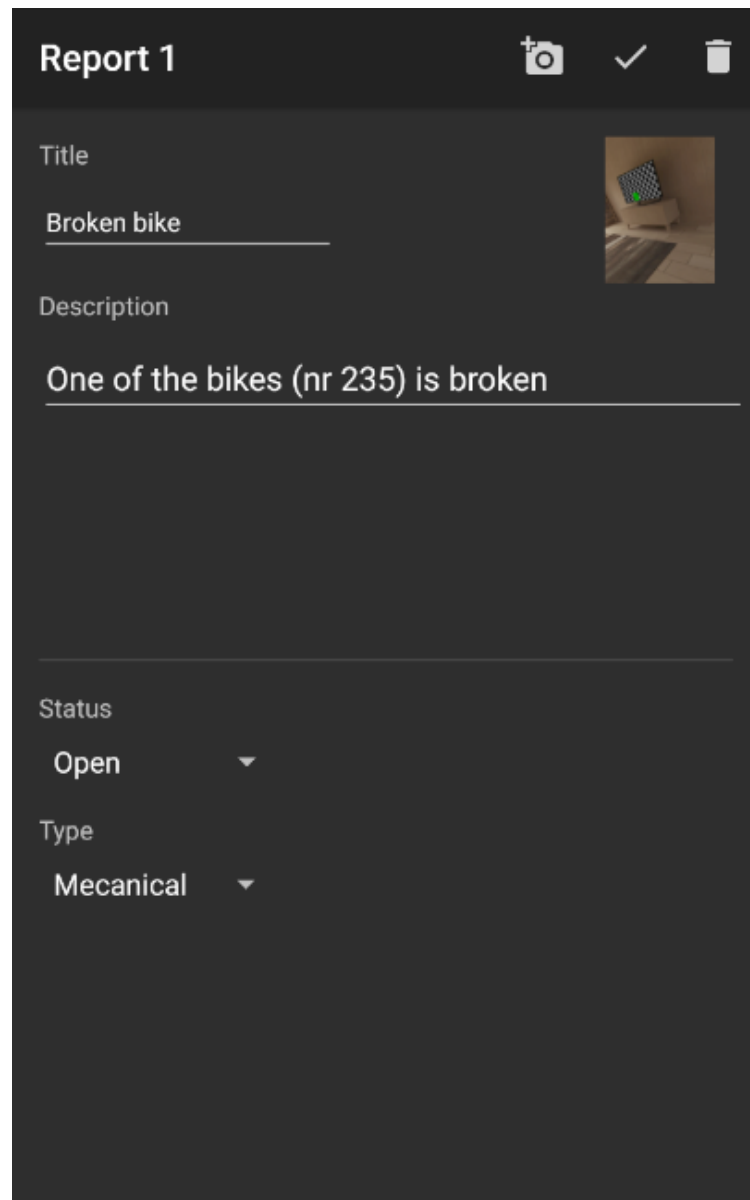
Here're some utility functions to do that:

```
// convert from bitmap to byte array
public static byte[] getBytes(Bitmap bitmap) {
    ByteArrayOutputStream stream = new ByteArrayOutputStream();
    bitmap.compress(Bitmap.CompressFormat.PNG, 0, stream);
    return stream.toByteArray();
}
```

```
// convert from byte array to bitmap
public static Bitmap getImage(byte[] image) {
    return BitmapFactory.decodeByteArray(image, 0, image.length);
}
```

You can also check the Android documentation for further information:

➤ <https://developer.android.com/training/camera/photobasics>



Report 1

Title

Broken bike

Description

One of the bikes (nr 235) is broken

Status

Open

Type

Mecanical

Figure 5. Detail of the report with picture

SESSION 4: HTTP SERVICE

In this session, the query for obtaining the Valenbisi bike stations list will be performed directly from the open data web service of the city council of Valencia. For working with *HTTP* we will use the API that Java offers for managing the http protocol.

To ask for the message will be used the *HTTP GET* method. The *GET* request will be sent to the URL: <http://mapas.valencia.es/lanzadera/opendata/Valenbisi/JSON>

In addition, the header field *Accept: application/json* will be established to indicate that the format of the answer must be JSON. Next, it is exposed an example of the use of GET request:

```
String url = "http://mapas.valencia.es/lanzadera/opendata/Valenbisi/JSON";
Writer writer = new StringWriter();
char[] buffer = new char[1024];
try {
    URL obj = new URL(url);
    HttpURLConnection con = (HttpURLConnection) obj.openConnection();
    con.setRequestMethod("GET");

    //add request header
    con.setRequestProperty("user-Agent", "Mozilla/5.0 (Windows NT 10.0; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.87 Safari/537.36");
    con.setRequestProperty("accept", "application/json;");
    con.setRequestProperty("accept-language", "es");
    con.connect();

    int responseCode = con.getResponseCode();
    if (responseCode != HttpURLConnection.HTTP_OK) {
        throw new IOException("HTTP error code: " + responseCode);
    }
    BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream(),
"UTF-8"));
    int n;
    while ((n = in.read(buffer)) != -1) {
        writer.write(buffer, 0, n);
    }
    in.close();
} catch (UnsupportedEncodingException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
```

It is important to note that any network connection must be performed outside the UI thread and it needs permission:

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

The connection must be performed in an internal class that extends from Activity following the pattern for carrying out a petition in a separated thread with *AsyncTask*.

```
class HTTPConnector extends AsyncTask<String, Void, ArrayList> {  
    @Override  
    protected ArrayList doInBackground(String... params) {  
        ArrayList paradas=new ArrayList<Parada>();  
  
        //Perform the request and get the answer  
  
        return paradas;  
    }  
  
    @Override  
    protected void onPostExecute(ArrayList paradas) {  
        // Create the ListView  
    }  
}
```

OPTIONAL DEVELOPMENT:

As an optional development of this session, we propose to add, in the Main Activity, the following functionalities:

- Use the GPS sensor in order to obtain the current location of the user instead of use a fixed location (as used for session 1 and 2).
- Implement a “Refresh” button, preferably in the ActionBar of the MainActivity) in order to fetch the latest data from Valenbisi bike stations and update the location of the user from the GPS sensor.

Valenbisi			
104	0	57.05m	
Albalat dels Tarongers - Paseo Facultades			3 
102	0	244.25m	
Ramón Llull - Serpis			17 
95	0	246.06m	
Naranjos (Magisterio)			0 
105	0	264.60m	
Aularios Universidad de Valencia			0 
261	0	278.03m	
Plaza Xuquer - Vinalopó			12 
103	0	416.32m	
Rubén Darío - Plaza Fray Luis Colomer			8 
114	0	438.48m	
UPV Informàtica			0 
94	0	445.99m	
Blasco Ibañez - Clariano			20 
112	0	462.10m	
Manuel Broseta i Pont - Naranjos			0 
96	0	488.86m	

Figure 5. List of bike station with refresh button

DELIVERABLES OF EACH SESSION

Students must deliver the zipped project folder to the “Aula Virtual”. Before compressing it, they should run the *gradlew* clean command to reduce the size of the deliverable. A delivery must be carried out after each session, no later than the day before the next session begins. Deliveries will be incremental (each delivery will contain the previous implementations). Thus, until a final delivery with the completed project, which in the same way will be delivered the previous day to the next session. The deliverables are the following:

- Deliverable 1: JSON file processed and Valenbisi bike stations list.
- Deliverable 2. Detail of the bike stations.
- Deliverable 3: Local report management system by using SQLite database.
- Deliverable 4: HTTP request to get the JSON file from the service of the City Council of Valencia.

DISTRIBUTION OF MARKS

The marks that the students can obtain at the end of the project are distributed in the following manner:

- Mandatory development: 7 points
- Optional development session 1 and 2: 1 point
- Optional development session 3: 1 point
- Optional development session 4: 1 point