ANDROID PRACTICE

Mobile Device Applications

Course 20-21 - 4 sessions

DESCRIPTION

In this practice, you will develop an application for Android devices. Specifically, the application will be about the incidence of COVID-19 in the Comunitat Valenciana. The app will allow users to check the status of the COVID-19 of the different municipalities. We will use the services of Generalitat Valenciana to retrieve data of COVID-19. This query will initially be carried out using a JSON file that can be requested from the data API of Generalitat Valenciana open data website[1]. However, in the first part, we will work with internal files in order to simplify: the JSON file with the incidence of COVID-19 per municipalities will be downloaded and copied to the project. On the list of municipalities of the Comunitat Valenciana, we will show the information about each of them. Next, we will create a local database that will allow the creation of new COVID-19 data associated with each of the municipalities. Finally, we will perform the query about the COVID-19 data through the web service, in order to display real-time data to our users. This practice will be developed using Android Studio.

GOALS

The goals of the practice are the following:

- Work with JSON format files.
- Work with Activities and Intents.
- Work with RecyclerViews and Adapters.

[1] https://dadesobertes.gva.es/es/dataset/covid-19-casos-confirmats-pcr-casos-pcr-en-els-ultims-14-dies-i-persones-mortes-per-municipi-2021

- 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 notifications.
- 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 **Covid19CV** with the domain **<studentname>.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 "**MunicipalitiesActivity**"

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 functionality development and an optional development. In order to opt to the maximum mark, all the optional part must be developed.

SESSION 1: LIST OF COVID-19 DATA PER MUNICIPALITIES

In this first session we will focus on reading the JSON file that will have the data and display the data of COVID-19 per municipalities in an Activity.

https://dadesobertes.gva.es/es/dataset/covid-19-casos-confirmats-pcr-casos-pcr-en-els-ultims-14-dies-i-persones-mortes-per-municipi-2021

The file can be downloaded at the following link:

https://dadesobertes.gva.es/es/api/3/action/datastore_search?resource_id=f6cb1e39-2839-4d38-8fd7-74430775a97e&limit=1000

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. Analyse the structure of the JSON file to understand which data is contained and how it is structured.

To show the data in the Activity we can use either a *ListView* (legacy) or a *RecyclerView* (recommended).

In this example, we will create a *RecyclerView* into the *Activity* (in the layout XML file), which will be fed with a class that extends from *RecyclerView.Adapter*. An example of the class structure is as follows:

```
public class AdapterMunicipios extends
RecyclerView.Adapter<AdapterMunicipios.ViewHolder> {

    private ArrayList<Municipio> municipios;
    Context context;

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

    public void Init() {
    // We read the JSON file and fill the "municipios" ArrayList

    }

    @Override
    public int getItemCount() {
```

```
        return municipios.size();
    }


/**
 * Provide a reference to the type of views that you are using
 * (custom ViewHolder).
 */
public static class ViewHolder extends RecyclerView.ViewHolder {
    private final TextView textView;

    public ViewHolder(View view) {
        super(view);
        textView = (TextView) view.findViewById(R.id.textViewNumber);
    }

    public TextView getTextView() {
        return textView;
    }
}

    // Create new views (invoked by the layout manager)
    @Override
    public ViewHolder onCreateViewHolder(ViewGroup viewGroup, int viewType) {
        // Create a new view, which defines the UI of the list item
        View view = LayoutInflater.from(viewGroup.getContext())
                .inflate(R.layout.listparadaview, viewGroup, false);

        return new ViewHolder(view);
    }

    public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
         // Get element from your dataset at this position and replace the
        // contents of the view with that element
        holder.getTextView().setText(String.valueOf(paradas.get(position).number));
    }
}
```

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() {

    municipios=new ArrayList<Municipio>();

    InputStream is = context.getResources().openRawResource(R.raw.municipiosCV);
    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 municipalities 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 the specific information of each municipality. It is not necessary to show all the information, only the most important that you consider.

In the activity class, the *RecyclerView* can be instantiated as follows:

```
        //Set up the RecyclerView
        RecyclerView recyclerView = findViewById(R.id.recyclerview);
        recyclerView.setLayoutManager(new LinearLayoutManager(this));
        MyRecyclerViewAdapter adapter = new MyRecyclerViewAdapter(this);
        recyclerView.setAdapter(adapter);
```

OPTIONAL DEVELOPMENT:

- Order the municipalities by the cumulative incidence (by default they are ordered alphabetically).
- Apply different colours of the rows if the cumulative incidence is low, medium, or high, as you consider.
- Add some buttons to switch between different ordering options (ascending/descending, other parameters)

REFERENCES:

- https://developer.android.com/reference/androidx/recyclerview/widget/RecyclerView.html

- https://developer.android.com/reference/androidx/recyclerview/widget/RecyclerView.Adapter
- Cómo crear listas dinámicas con RecyclerView: https://developer.android.com/guide/topics/ui/layout/recyclerview?hl=es-419
- Android: Listas dinámicas usando RecyclerView: https://programacionymas.com/blog/listas-dinamicas-android-usando-recycler-view-card-view

---

SESSION 2: DETAIL OF EACH MUNICIPALITY

In the second session, we will implement some additional functionalities to our application:

- Full information of any municipality in another activity
- Menu for accessing main options on each activity
- Search bar (optional)

Full information of any municipality

First of all, we will implement a listener so when the user clicks on any municipality in the list, a new activity will display all the information of the selected municipality. There are several approaches to implement this. One option is to create an OnClickListener in the adapter. The steps are the following:

1. Implement View.OnClickListener() in the activity containing the recycler view:

```
private View.OnClickListener onItemClickListener = new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        // This viewHolder will have all required values.
        RecyclerView.ViewHolder viewHolder = (RecyclerView.ViewHolder) view.getTag();
        // Implement the listener!

    }
};
```

2. Add the following method to the implementation of your ViewHolder in the adapter class and set the onClick listener in the onCreateViewHolder method of the ViewHolder:

```
public void setOnItemClickListener(View.OnClickListener itemClickListener) {
    mOnItemClickListener = itemClickListener;
}

// Put this line in the code of the ViewHolder constructor
view.setTag(this);

// Put this line in the code of the onCreateViewHolder method
view.setOnClickListener(mOnItemClickListener);
```

3. Set the listener to your adapter in the Activity that use the RecyclerView when created the adapter:

```
adapterMunicipios.setOnItemClickListener(onItemClickListener);
```

Consequently, the new activity will be called by clicking each of the items in the list of municipalities activity list. The municipality can be passed as an extra in the Intent.

Another way to implement a listener is by creating an interface in the adapter and implementing this interface in the recycler view activity. The steps are the following:

1. Define an interface in the Adapter class and a method to indicate the listener:

```
public interface ItemClickListener {
    void onRVItemClick(View view, int position);
}
// El Activity que incluya el Recycler View que utilice este adapter llamará a
este método para indicar que es el listener.
void setClickListener(ItemClickListener itemClickListener) {
    this.mClickListener = itemClickListener;
}
```

2. The implementation of our ViewHolder in the Adapter class shoul implement View.OnClickListener and override the method onClick(). Moreover, it has to set the listener on the constructor of the ViewHolder:

```
// Put this line in the code of the ViewHolder constructor
view.setOnClickListener((View.OnClickListener) this);

}

@Override
public void onClick(View view) {
    if (mClickListener != null)
        mClickListener.onRVItemClick(view, getAdapterPosition());
}
```

3. Implement the interface of the listener in the Activity that use the RecyclerView with the functionality to do when the user click on each item and set the listener to our adapter:

```
// Put this line when create the adapter

adapter.setClickListener(MainActivity.this);


@Override
public void onRVItemClick(View view, int position) {
    Toast.makeText(this, "Has pulsado en " +
adapter.getItemAtPosition(position).getName() + " que es el item número " +
position, Toast.LENGTH_SHORT).show(); //If you want to use the method
getItemAtPosition you should implement it in the adapter.

    // Implement the listener! E.g., start a new activity to show details of
the municipality. The municipality can be passed as an extra in the intent
}
```

An example to implement the listener in this way can be seen in: https://umhandroid.momrach.es/basicrecyclerview/

The activity title must be changed to the name of the municipality.

Create an attractive user interface (for example, using graphical elements) to display all the information.

Menu for accessing main options on each activity

Now that we have two activities, we will implement menus in each of these activities for accessing the main options.

In the first activity, the menu will offer the following options:

- Open in the device's browser the Generalitat Valenciana website on COVID-19 (with an implicit intent)
- Sorting options (if implemented in the previous session).

In the second activity the menu will offer the following option:

- Open in the device's maps application the location of the municipality (with an implicit intent[2]).

In order to create the menu, you must define a menu and all its items in an XML menu resource. The XML should be defined in the *res/menu* folder. 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);
    }
}
```

[2] Note the string containing the name of the municipality must be encoded, so "Albalat dels Tarongers" would be "Albalat%20dels%20Tarongers" or "Albalat+dels+Tarongers". More information on: https://developer.android.com/guide/components/intents-common#Maps

You can also check the Android documentation for further information:

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

OPTIONAL DEVELOPMENT:

Since the number of municipalities is quite long, we will implement a search option in our application, so the user can type the first letters and only matching municipalities are displayed in the list.

To this effect:

1. We will implement the filterable interface[3] in our adapter, overriding the getFilter() method. This method contains the filter condition used to search through a list (in our case, the municipalities name):

```
class AdapterMunicipios extends RecyclerView.Adapter<A AdapterMunicipios.ViewHolder>
implements Filterable {
```

Here you have an example of the getFilter method for filtering a contact list. Note that we will use two ArrayLists, one for the full list and other for the filtered list:

```
@Override
public Filter getFilter() {
    return new Filter() {
        @Override
        protected FilterResults performFiltering(CharSequence charSequence) {
            String charString = charSequence.toString();
            if (charString.isEmpty()) {
                contacts_filtered = contacts;
            } else {
                ArrayList<Contact> filteredList = new ArrayList<>();
                for (Contact row : contacts) {
                    if (row.name.toLowerCase().contains(charString.toLowerCase())) {;
                        filteredList.add(row);
                    }
                }

                contacts_filtered = filteredList;
            }
```

[3] Filterable: https://developer.android.com/reference/android/widget/Filterable

```
            FilterResults filterResults = new FilterResults();
            filterResults.values = contacts_filtered;
            return filterResults;
        }

        @Override
        protected void publishResults(CharSequence charSequence, FilterResults
filterResults) {
            contacts_filtered = (ArrayList<Contact>) filterResults.values;
            notifyDataSetChanged();
        }
    };
}
```

Note that now in the adapter you have to use the filtered arraylist of data, e.g., *municiciosFiltered* instead of *municipios*.

2. We will add a search widget in the menu of our main activity (the one displaying the list of municipalities) and make it always visible. For example:

```
<item
    android:id="@+id/app_bar_search"
    android:icon="@android:drawable/ic_menu_search"
    android:title="Search"
    app:actionViewClass="android.widget.SearchView"
    app:showAsAction="always" />
```

3. Consequently, onCreateOptionsMenu() will inflate the menu and display the SearchView. Then, we will need to implement the listener searchView.setOnQueryTextListener() which will listen to character changes while the user types. The search query will be passed to adapter class using the method we just implemented: getFilter().filter(query).

   Here you have an example:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.appbar, menu);
    // Associate searchable configuration with the SearchView
    SearchManager searchManager = (SearchManager)
getSystemService(Context.SEARCH_SERVICE);
    searchView = (SearchView) menu.findItem(R.id.app_bar_search)
            .getActionView();
    searchView.setSearchableInfo(searchManager
            .getSearchableInfo(getComponentName()));
```

```
    searchView.setMaxWidth(Integer.MAX_VALUE);

    // listening to search query text change
    searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {
        @Override
        public boolean onQueryTextSubmit(String query) {
            // filter recycler view when query submitted
            myAdapter.getFilter().filter(query);
            return false;
        }

        @Override
        public boolean onQueryTextChange(String query) {
            // filter recycler view when text is changed
            myAdapter.getFilter().filter(query);
            return false;
        }
    });
    return true;
}
```

Finally, the RecyclerView will be updated with filtered data.

SESSION 3: SYMPTOMS 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 positive COVID-19 case report must contain the following information:

- Diagnostic code
- Symptom start date
- Symptoms (multiple selection):
    - Fever or chills
    - Cough
    - Shortness of breath or difficulty breathing
    - Fatigue
    - Muscle or body aches
    - Headache
    - New loss of taste or smell
    - Sore throat
    - Congestion or runny nose
    - Nausea or vomiting
    - Diarrhea
- Are you a close contact?:
    - Yes
    - No
- Municipality

We will access this report management system from a floating button that will be available in both activities (list of municipalities and detail of municipality). When the report management is called from the detail activity the municipality will be filled automatically.

- https://developer.android.com/guide/topics/ui/floating-action-button

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

- *InsertReport*
- *UpdateReport*

- *DeleteReport*
- *FindReportsByMunicipality*

The functions that return reports must return a *CURSOR* with the data of the executed *SELECT*. In order to simplify the database design, the symptoms could be defined as booleans. 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 order to implement the database and the related classes you can check these links:

https://developer.android.com/training/data-storage/sqlite

https://www.develou.com/android-sqlite-bases-de-datos/

In the Municipality Detail layout must be inserted a *RecyclerView* or *ListView* that displays the reports associated with the current municipality. The view must be fed with a class that extends from *CursorAdapter*.

The following code shows an example of how to initialize both the *Cursor* and the *Adapter,* in a *ListView*:

```
ReportDbHelper db = new ReportDbHelper(getApplicationContext());
Cursor reportsByMunicipality = db.findReportsByMunicipality(p.name);
reportsAdapter = new ReportsCursorAdapter(getApplicationContext(),
reportsByMunicipality, 0);
ListView lv=(ListView) findViewById(R.id.idlist_reports);
lv.setAdapter(reportsAdapter);
```

We will create a *Listener* to open a new activity by clicking on each report. 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.

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 municipality, which the report will be associated.

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

- Report diagnostic code
- Symptom start date
- A checkbox with the symptoms

- A spinner with the question about the close contact.
- A menu containing:
    - An item to insert/update the report
    - An item to delete the report

OPTIONAL DEVELOPMENT:

As an optional development of this session, we propose to use the GPS sensor to obtain the current location of the user and use it to get the name of the municipality.

Examples to obtain the current location can be found in the following links:

https://www.androfast.com/2015/07/como-obtener-la-ubicacion-del-gps.html

https://academiaandroid.com/geolocalizacion-obtencion-coordenadas-desde-app-android/

Once you have the location of your device, you can use Geocoder to retrieve the name of the city from a given Location and fill in the municipality of the report.

```
Geocoder geocoder = new Geocoder(this, Locale.getDefault());
try {
    List<Address> addresses = geocoder.getFromLocation(myLocation.getLatitude(),
(Double) myLocation.getLongitude(), 1);
} catch (IOException e) {
    e.printStackTrace();
}

String city = addresses.get(0).getLocality();
```

Examples to use Geocoder can be found here: https://www.codota.com/code/java/classes/android.location.Geocoder

Then, we will store this municipality in the report, so the user does not need to write the municipality manually (even if this can be changed manually).

SESSION 4: HTTP SERVICE

In this session, the query for obtaining the municipalities list will be performed directly from the open data web service of the Generalitat Valenciana. 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 same URL:

https://dadesobertes.gva.es/es/api/3/action/datastore_search?resource_id=65d57138-14aa-467b-849d-2b604d398fd9&limit=1000

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 = "
https://dadesobertes.gva.es/es/api/3/action/datastore_search?resource_id=382b283c-03fa-433e-9967-9e064e84f936&limit=1000";
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 as an internal class of a class that extends from Activity, e.g., the Main 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 municipios=new ArrayList<Parada>();

        //Perform the request and get the answer

        return municipios;
}

    @Override
    protected void onPostExecute(ArrayList municipios) {
       // Create the RecyclerView
         }
}
```

The JSON file returned from the HTTP request is a bit different from the downloaded by the download button, so you will have to modify the parsing of the JSON file if you was using the JSON of the download button. In this version, the information of each municipality is stored in a json object, not in a json array and the "records" array belongs to the "result" object.

If the HTTP connection fails, make sure you have added the permissions on the Manifest, and uninstall the application from Android Emulator and then run again the application from Android Studio.

OPTIONAL DEVELOPMENT:

As an optional development of this session, we propose to add the following functionality (choose just one!):

- Add a "refresh" option to retrieve the latest dataset from Generalitat Valenciana. The problem here is that the resource_id for any new dataset changes, so first you will need to obtain the id of the latest dataset. You can get all the information on the package

from this JSON file, including the id of the latest dataset (in the JSON, check for results → resources and you will get a list of all the resources): https://dadesobertes.gva.es/api/3/action/package_show?id=38e6d3ac-fd77-413e-be72-aed7fa6f13c2

- Alternatively, implement that the app checks automatically whether there is a more recent dataset when it starts.
- Implement a warning notification when the device location corresponds to a municipality where the cumulative incidence is greater than 100.

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 municipalities list.
- Deliverable 2. Search municipalities and display municipalities detail.
- Deliverable 3: Symptoms report management system by using SQLite database.
- Deliverable 4: HTTP request to get the JSON file from Generalitat Valenciana open data service.

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