Structure of Computer Systems

2021-2022

Project

Creating an application for phones that use the Android operating system

-

Laboratory guide and 3 Android applications

Chereji Iulia-Adela

Group 30434-1

CONTENTS

1.INTRODUCTION

1.1 Android Operating System

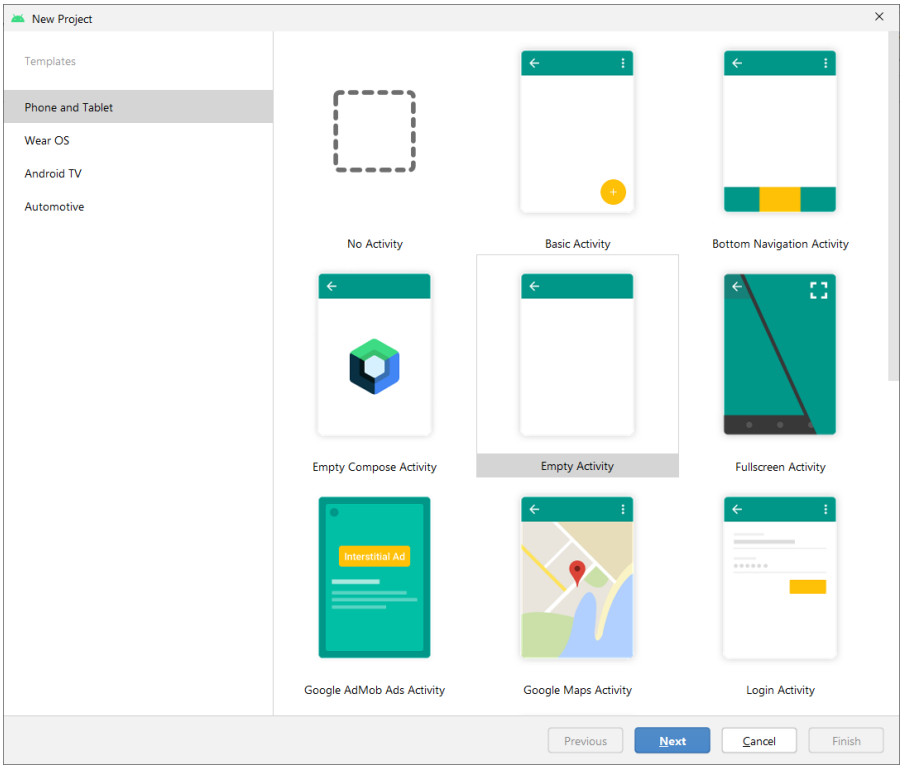
Android is a mobile operating system designed for touchscreen mobile devices such as smartphones and tablets. Mobile applications can be Web (sites that look good on phones and can be accessed via a browser without having to install or update anything), Native (written specifically for Android or IOS) and Hybrid (that send and receive data to a remote server but also have an installed app but they are not written specifically for one operating system so they can't use all the device's functionalities).
Java is the main language used for the user interface of the Android OS Native applications (the other used language is Kotlin). Applications use touch inputs from the user and also internal sensors to do actions (e.g.: rotating an image when the phone is oriented from portrait to landscape). Applications can use Wi-Fi, Bluetooth, GPS, camera and other phone functionalities. They can also access the file manager, create and delete files, save data using databases and access other applications. The users can download apps from the Google Play Store but they must match the version of the Android OS. The 2020's latest update is Android 11.

1.2 Android Studio

Android Studio is the official integrated development environment (IDE) for Google's Android operating system. It is built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. It created a binding between the functionality and the visibility of one 'page': source files (.java) can access any specific element added into the layout file (.xml).
It is available for download on Windows, macOS and Linux at https://developer.android.com/studio?gclid=CjwKCAiAv_KMBhAzEiwAs-rX1Es2pt0sE23Lkr7NKHGBfnWkwRCA4hTjTdNctaPUvbF5rM3Apo7LVhoCTvgQAvD_BwE&gclsrc=aw.ds

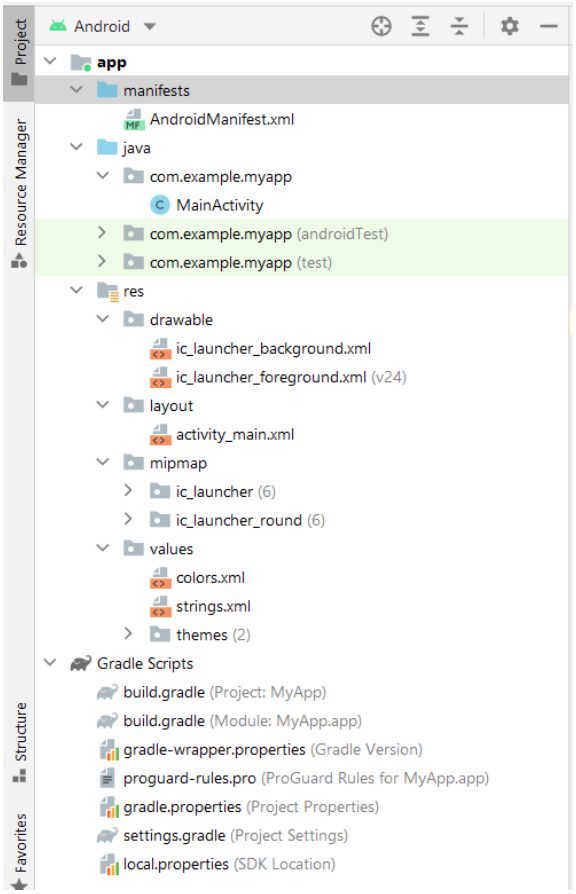
2.WRITING THE FIRST APPLICATION

1. Open Android Studio, click New Project, select Phone and Tablet and choose a type of activity. We will choose Empty Activity. The IDE also provides a lot of other templates like Login Activity, Google Maps Activity, Settings Activity, Scrolling Activity, etc.



2. Click Next, give the app a name, and choose Java as the language then click Finish.

Any Android application will contain an Android Manifest folder where the AndroidManifest.xml file will be found. This file is a special one because it contains different details about the application, such as the name of the application, the icon and dependencies (several physical aspects which need to be allowed in the hardware in order to be used in the software). Another key part of the Android application is the Gradle Scripts folder. This one controls the different software dependencies of the application. The res folder contains the layouts, the drawable elements (pictures, icons, etc.), values as stored strings, colors declared, styles, etc. The java folder contains the source files which will define the functionality, using the resources.

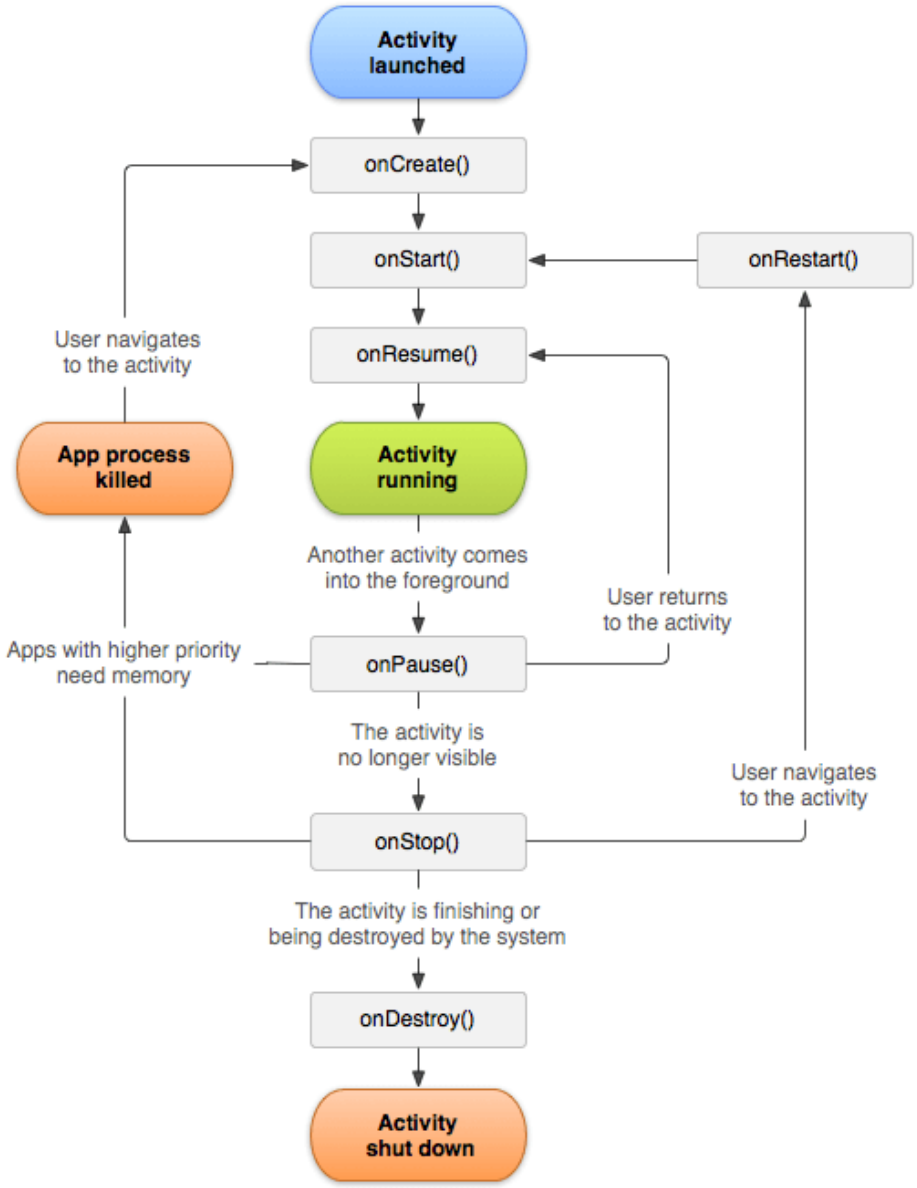


The first source file that is created automatically is MainActivity.java. The only thing that the class does is to override the onCreate method which is called when the app starts.

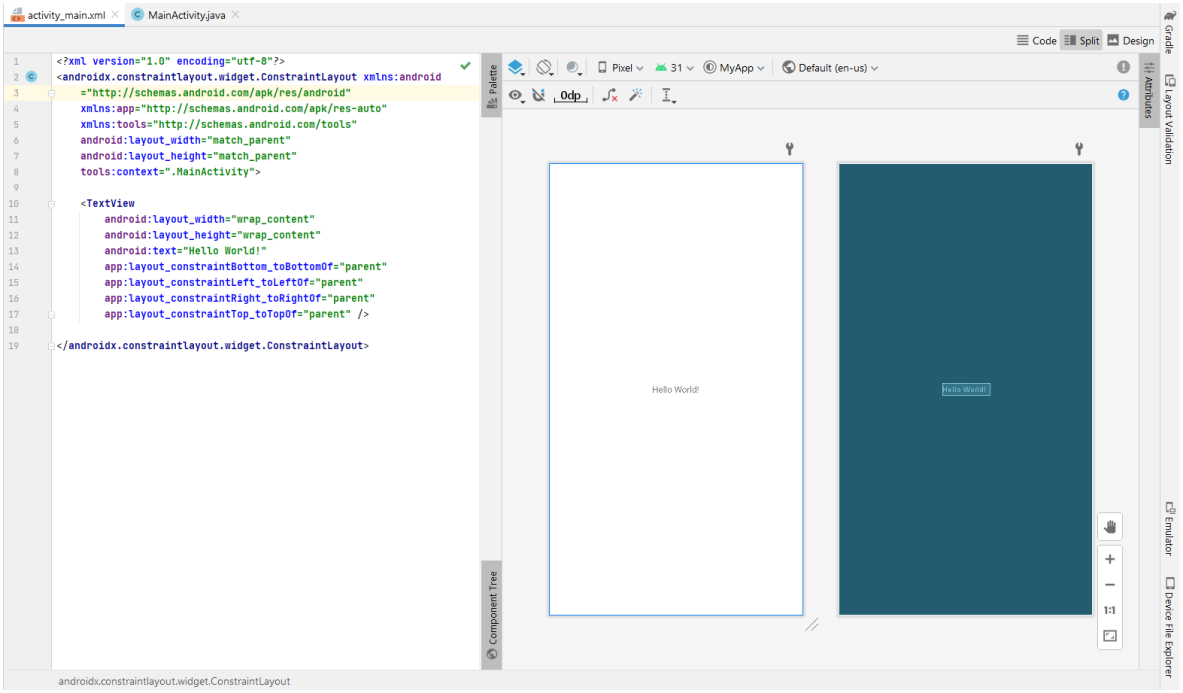
activity_main.xml | MainActivity.java

```
1  package com.example.myapp;
2
3  import androidx.appcompat.app.AppCompatActivity;
4
5  import android.os.Bundle;
6
7  public class MainActivity extends AppCompatActivity {
8
9      @Override
10     protected void onCreate(Bundle savedInstanceState) {
11         super.onCreate(savedInstanceState);
12         setContentView(R.layout.activity_main);
13     }
14 }
```

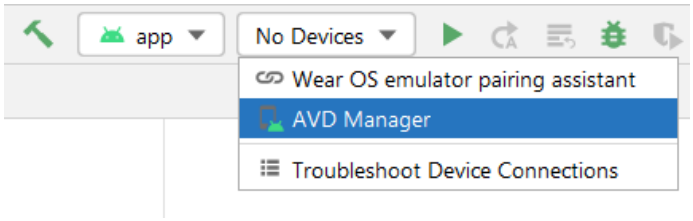
The lifecycle of an activity. By default the MainActivity is the first one to be launched when the app is started but that can be modified. Activities can start other activities.

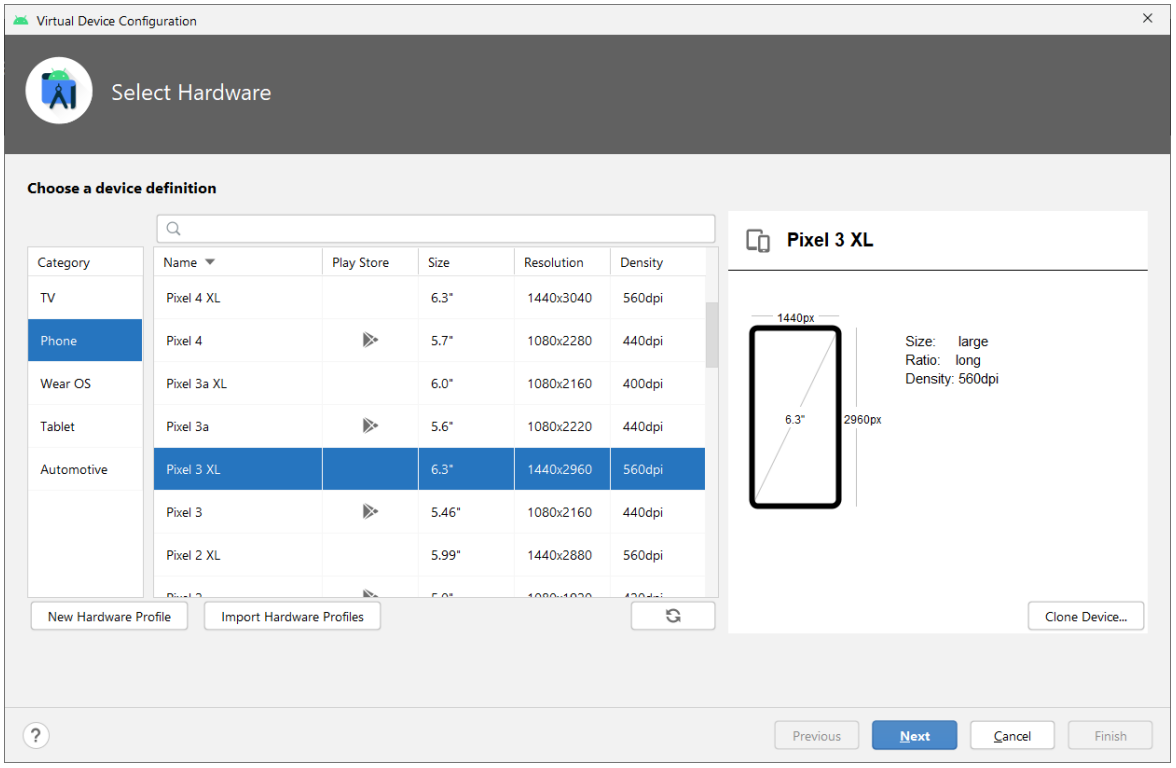


The activity_main.xml is the layout of the activity. It has a code and you can see how it looks. You can add things to the layout (e.g. buttons, fields, checkboxes) by drag and drop to the Design or by writing code in the code section.

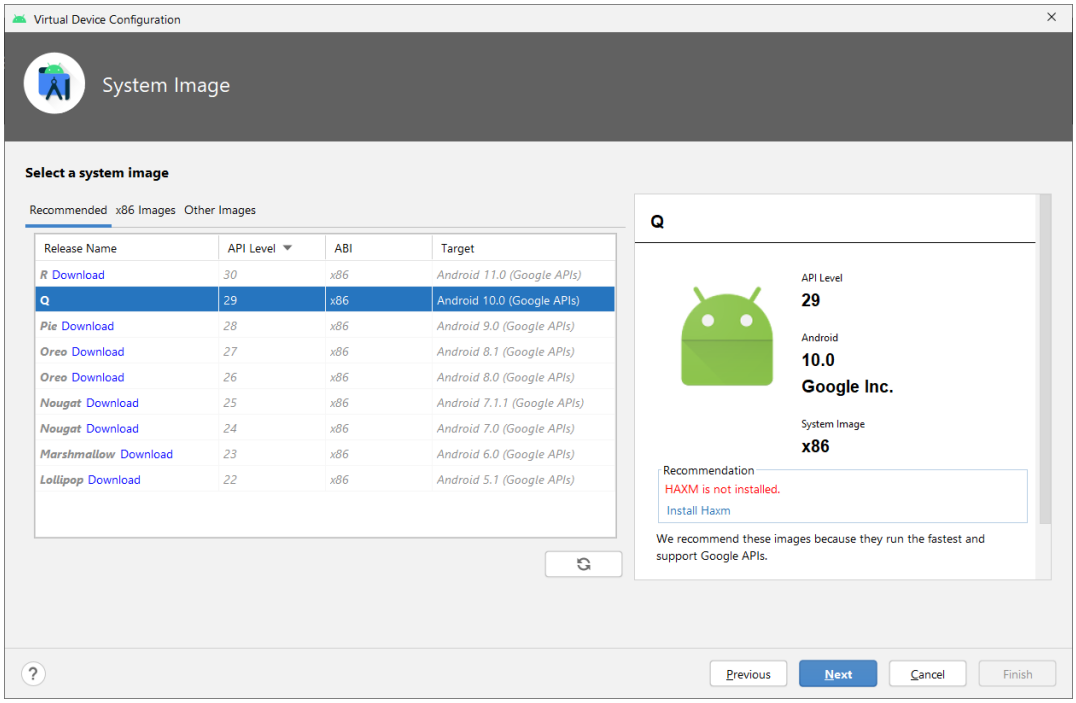


3. To test your application you must create a virtual device (emulator). Click on AVD Manager, Create Virtual Device, then select a resolution, then click next.





Select a version of Android and make the necessary downloads, click next, click finish. Now you should be able to run the app on a virtual device.



4. To test your app on your Android physical device connect it to the PC via an USB cable and go to settings on your phone to allow debugging.
5. We can add items and their attributes to activity_main.xml, for example:

```
<TextView
    android:id="@+id/textHelloWorld"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="43dp"
    android:layout_marginTop="131dp"
    android:layout_marginEnd="43dp"
    android:layout_marginBottom="170dp"
    android:fontFamily="serif"
    android:text="Hello World!"
    android:textColor="@color/black"
    android:textSize="60sp"
    app:layout_constraintBottom_toTopOf="@+id/spinnerColor"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

And we can use them in MainActivity.java like this:

```
TextView textViewHello = findViewById(R.id.textHelloWorld);
textViewHello.setTextColor(Color.RED);
```

We can also add a spinner in activity_main.xml and assign an OnItemSelectedListener to it in MainActivity.java to handle user input.

6. In the build.gradle (:app) file we can add more dependencies

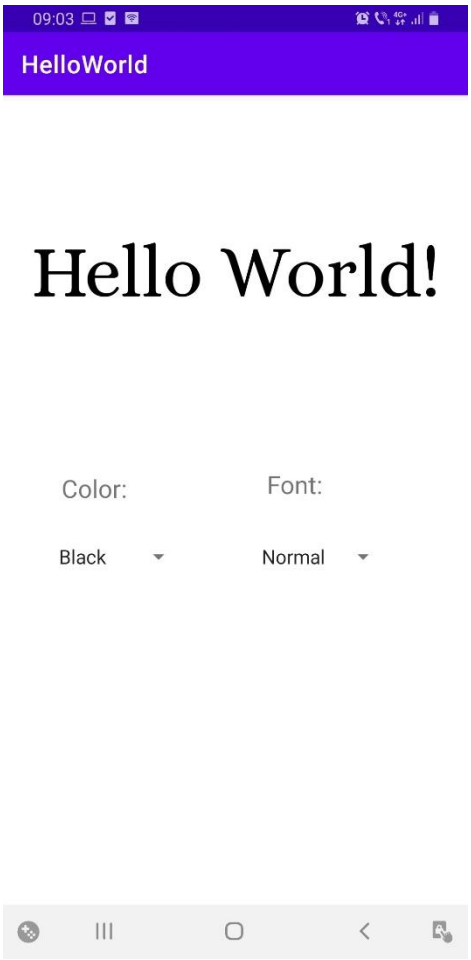
```
dependencies {

    implementation 'androidx.appcompat:appcompat:1.3.1'
    implementation 'com.google.android.material:material:1.4.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.1'
    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
}
```

3. IMPLEMENTATION

To illustrate best the power of the Android software mechanism I designed 3 Android Applications:

1. A HelloWorld application that displays the text and lets the user choose from a set of colors and fonts for it.
2. A Paint application in which you can choose the pen width and color, draw images, erase and then save the drawings as pictures into the phone's memory.
3. A Task Management application which lets you add tasks, check them, update them and delete them.

3.1 HelloWorld Application



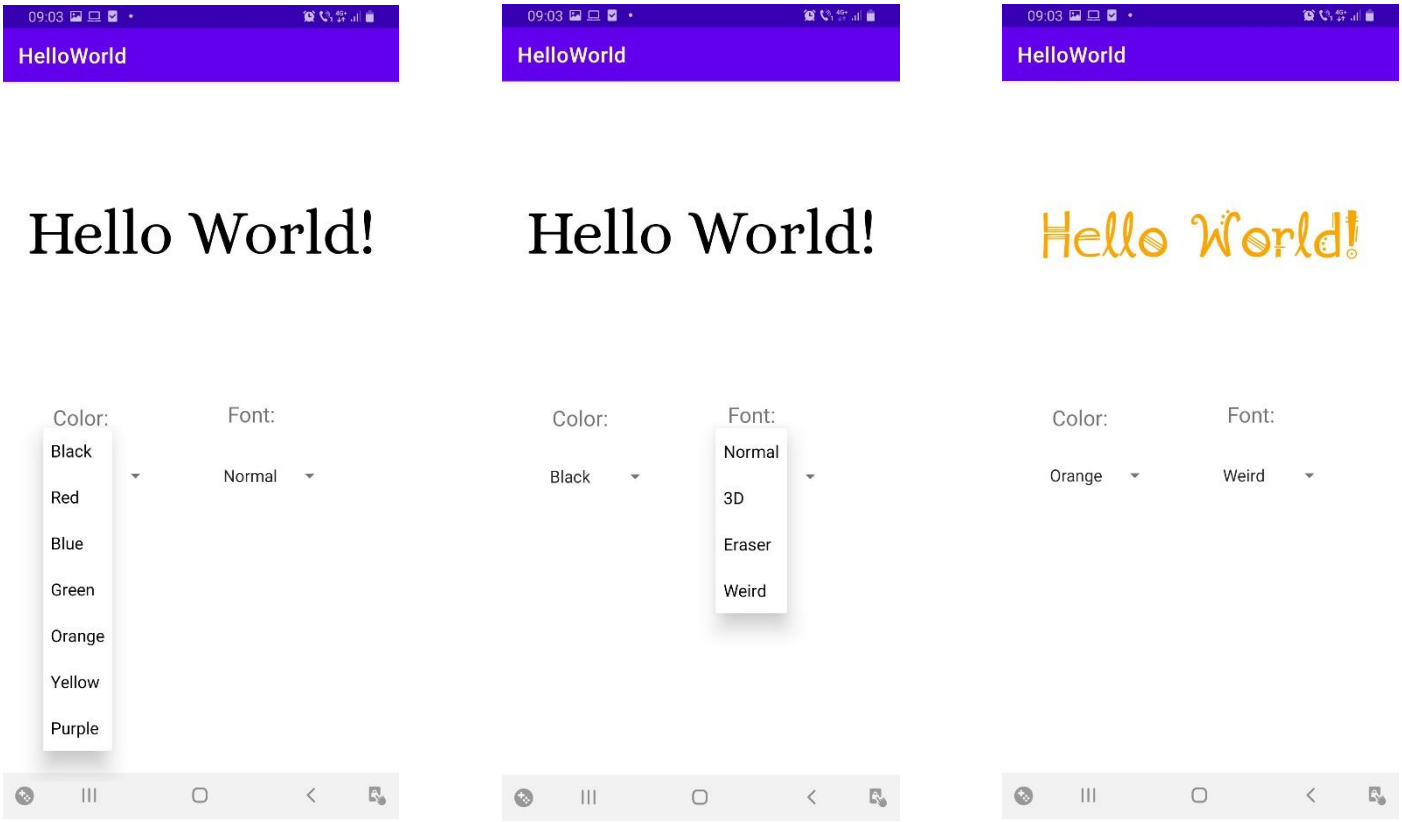
The application displays the text "Hello World!" with different colors and fonts that the user can choose.

In activity_main.xml I used a TextView for the text and two Spinners for the color and font drop down lists.

```
<TextView
    android:id="@+id/textHelloWorld"
    …
/>
<Spinner
    android:id="@+id/spinnerColor"
    …
/>
<Spinner
    android:id="@+id/spinnerFont"
    …
/>
```

In the MainActivity class I created attributes for these objects and I used the method findViewById to take them from the layout in the onCreate method.

```
private Spinner colorSpinner;
private Spinner fontSpinner;
private TextView textViewHello;

colorSpinner=findViewById(R.id.spinnerColor);
textViewHello=findViewById(R.id.textHelloWorld);
fontSpinner=findViewById(R.id.spinnerFont);
```



I created two lists of Strings, colors and fonts and used them to create two ArrayAdapters of Strings, then I assigned them to the spinners.

```
ArrayAdapter<String> colorAdapter = new ArrayAdapter<>(this,
android.R.layout.simple_spinner_item, colors);
colorAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown
_item);
colorSpinner.setAdapter(colorAdapter);

ArrayAdapter<String> fontAdapter = new ArrayAdapter<>(this,
android.R.layout.simple_spinner_item,fonts);
fontAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_
item);
fontSpinner.setAdapter(fontAdapter);
```

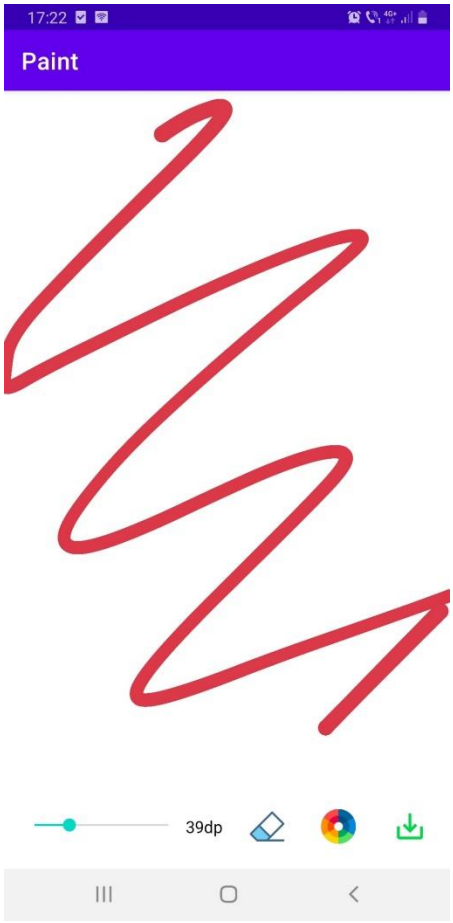
For both spinners I added an OnItemSelectedListener to handle user input.

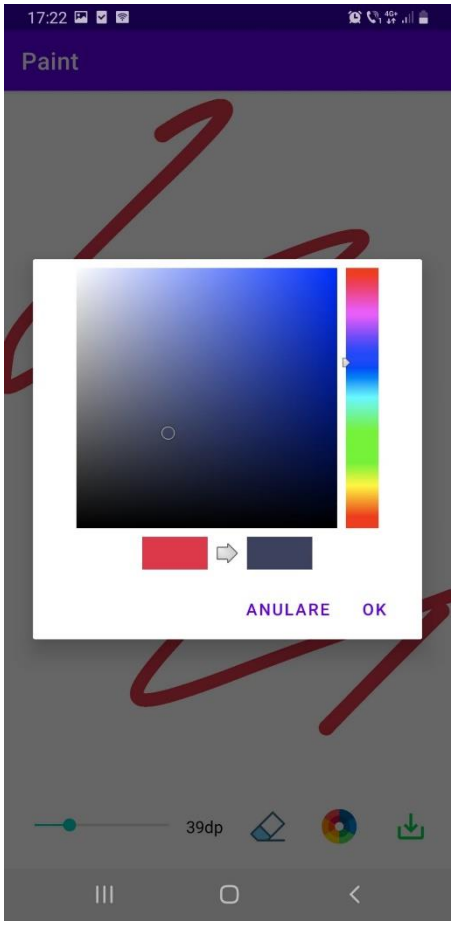
```
colorSpinner.setOnItemSelectedListener(new
AdapterView.OnItemSelectedListener() {
    @Override
    public void onItemSelected(AdapterView<?> parent, View view, int
position, long id) {
        String choice = parent.getItemAtPosition(position).toString();
        switch (choice){
            case "Red": textViewHello.setTextColor(Color.RED);
            break;
            …
            case "Orange": textViewHello.setTextColor(Color.rgb(255,165,0));
                break;
        }
    }


});
```

```
fontSpinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener()
{
    @Override
    public void onItemSelected(AdapterView<?> parent, View view, int position,
long id) {
        String choice = parent.getItemAtPosition(position).toString();
        Typeface typeface;
        switch (choice){
            case "Normal":
                typeface = Typeface.createFromAsset(getAssets(),"normal.ttf");
                textViewHello.setTypeface(typeface);
                break;
            case "3D":
                typeface = Typeface.createFromAsset(getAssets(),"3d.ttf");
                textViewHello.setTypeface(typeface);
                break;
            case "Eraser":
                typeface = Typeface.createFromAsset(getAssets(),"eraser.ttf");
                textViewHello.setTypeface(typeface);
                break;
            case "Weird":
                typeface = Typeface.createFromAsset(getAssets(),"weird.ttf");
                textViewHello.setTypeface(typeface);
                break;
        }
    }
});
```

For the fonts I added the files .ttf in the assets folder (I downloaded them from the internet)

3.2 Paint Application





The application allows users to draw on a canvas and then save their paintings on their device as a .png file which will appear in the gallery in a folder called My Paintings. The name of the file will be the date and time of the save in the format: yyyymmdd_hhmmss.png. The application asks for permission to access phone storage when launched the first time. In the AndroidManifest.xml file I wrote the required permissions:

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

In the MainActivity class in the onCreate method I call the askPermission method:

```
private void askPermission() {
    Dexter.withContext(this)
            .withPermissions(Manifest.permission.READ_EXTERNAL_STORAGE,
Manifest.permission.WRITE_EXTERNAL_STORAGE)
            .withListener(new MultiplePermissionsListener() {
                @Override
                public void onPermissionsChecked(MultiplePermissionsReport
multiplePermissionsReport) {
                }

                @Override
                public void
onPermissionRationaleShouldBeShown(List<PermissionRequest> list, PermissionToken
permissionToken) {
                    permissionToken.continuePermissionRequest();
                }
            }).check();
}
```

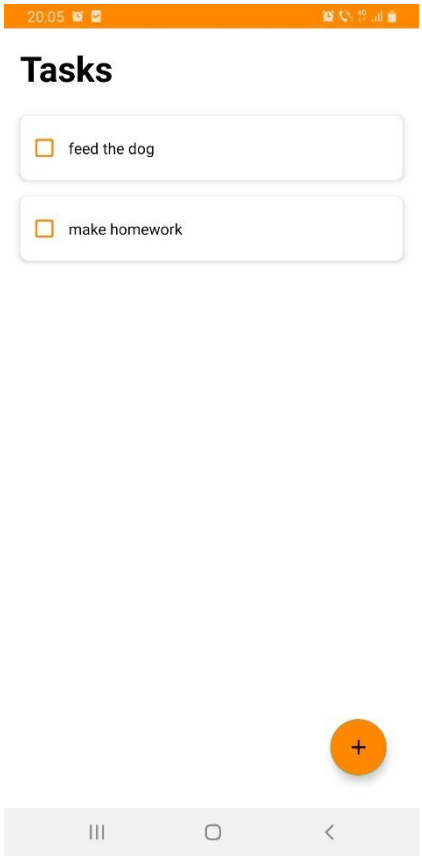
The application lets users pick the thickness of the pen and the color, and also to erase the whole drawing. I downloaded the icons for color, save and eraser and put them in the res.drawable folder.

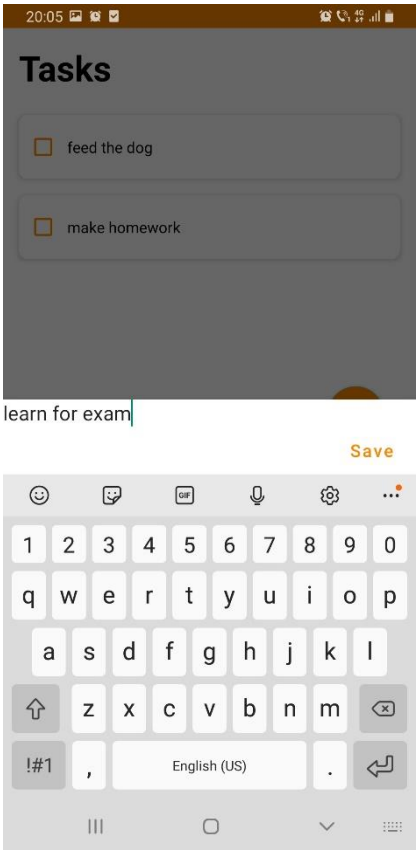
For this application I added some dependencies from the internet in the build.gradle(:app):

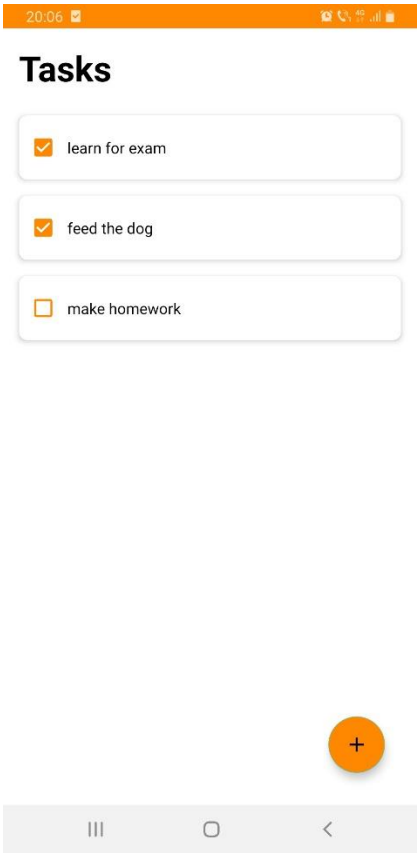
```
implementation 'com.karumi:dexter:6.2.3'
implementation 'com.github.yukuku:ambilwarna:2.0.1'
implementation 'com.kyanogen.signatureview:signature-view:1.2'
```

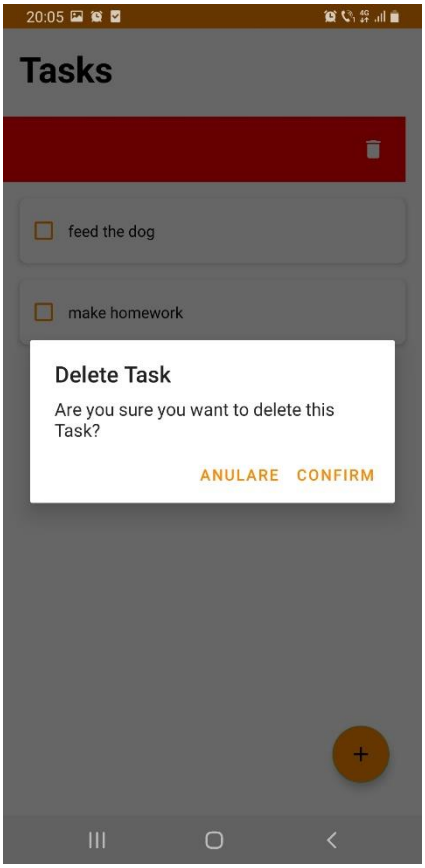
For choosing the pen size I used a SeekBar and an OnSeekBarChangeListener, and for the eraser, color and download icons I used ImageButtons and OnClickListeners. For the canvas area I used a SignatureView. In

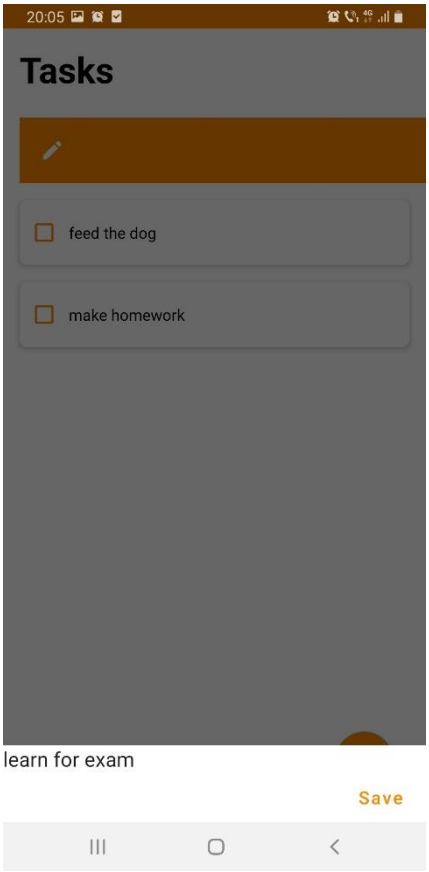
3.3 Task Management Application











This application stores your tasks (or reminders). It lets you view your tasks, add new task, check or uncheck a task, delete and update a task.

Adding a new task is done by touching the "+" button; then a window for text appears where you can type in text, then click on the "save" button.

To check and uncheck a task just click on it's checkbox.

To delete a task, drag it to the left (swipe). A pop up window willappear asking you if you want to delete it, then click "CONFIRM".

To update a tack swipe to the other direction and when the window appears just type in the modified text then click save.

When the app is started, first it shows an intro image. For this I created the class SplashActivity.java and a .xml file (activity_splash.xml), and I set it to be the activity that is first launched, in the AndroidManifest.xml: `android:name=".SplashActivity"`

```
public class SplashActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_splash);
        getSupportActionBar().hide();
        final Intent i =  new Intent(SplashActivity.this, MainActivity.class);
        new Handler().postDelayed(new Runnable() {
            @Override
            public void run() {
                startActivity(i);
                finish();
            }
        },1000);
    }
}
```

After this activity ends (after 1 second), I set the MainActivity to be the next one in AndroidManifest.xml: `android:name=".MainActivity">`

The DatabaseHandler class extends SQLiteOpenHelper and provides the methods that are needed in order to work with the database:

```
public void onCreate(SQLiteDatabase db)
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
public void openDatabase()
public void insertTask(ToDoModel task)
public List<ToDoModel> getAllTasks()
public void updateStatus(int id, int status)
public void updateTask(int id, String task)
public void deleteTask(int id)
```

The ToDoModel class represents the model for a task and has the fields:

```
private int id, status;
private String task;
```

and getters and setters.

The ViewHolder class holds a view.

```
public static class ViewHolder extends RecyclerView.ViewHolder{
    CheckBox task;
    ViewHolder(View view){
        super(view);
        task=view.findViewById(R.id.todoCheckBox);
    }
}
```

The ToDoAdapter class holds the list of all the tasks and has references to the database and the MainActivity.

The onBindViewHolder method calls the updateStatus of the database for the task that has been checked or unchecked:

```
public void onBindViewHolder(ViewHolder holder, int position){
    db.openDatabase();
    ToDoModel item = todoList.get(position);
    holder.task.setText(item.getTask());
    holder.task.setChecked(toBoolean(item.getStatus()));
    holder.task.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
        @Override
        public void onCheckedChanged(CompoundButton buttonView, boolean
isChecked) {
            if(isChecked){
                db.updateStatus(item.getId(),1);
            }
            else db.updateStatus(item.getId(),0);
        }
    });
}
```

It does the same for deleting or editing a task:

```
public void deleteItem(int position){
    ToDoModel item = todoList.get(position);
    db.deleteTask(item.getId());
    todoList.remove(position);
    notifyItemRemoved(position);
}

public void editItem(int position){
    ToDoModel item = todoList.get(position);
    Bundle bundle = new Bundle();
    bundle.putInt("id",item.getId());
    bundle.putString("task", item.getTask());
    AddNewTask fragment = new AddNewTask();
    fragment.setArguments(bundle);
    fragment.show(activity.getSupportFragmentManager(), AddNewTask.TAG);
}
```

The AddNewTask class extends BottomSheetDialogFragment and it is responsible for adding or updating a task. It gets the text from the dialog, checks if it is a new task or a task is updated and calls the database to do the necessary action.

The RecyclerItemTouchHelper class is responsible for the effect of swiping. It handles the color and the icon displayed, it shows the message for confirmation in case of delete and it opens the dialog in case of edit, and then calls the ToDoAdapter to make the database updates.

The MainActivity is the activity that starts after the SplashActivity. The class has the necessary fields:

```
private RecyclerView tasksRecyclerView;
private ToDoAdapter tasksAdapter;
private List<ToDoModel> taskList;
private DatabaseHandler db;
```

4.BIBLIOGRAPHY

https://developer.android.com/studio/run?gclid=CjwKCAiAv_KMBhAzEiwAs-rX1DpsVnfSx2L2xixX9Fr92D7eLa1RlpZAW7rRf9mcX8DZBWlv_o795hoC8dcQAvD_BwE&gclsrc=aw.ds

https://developer.android.com/training/basics/firstapp/creating-project

https://developer.android.com/training/basics/firstapp/running-app

https://developer.android.com/training/basics/firstapp/building-ui

https://developer.android.com/training/basics/firstapp/starting-activity

https://drive.google.com/drive/folders/1qDdZiwpUNvGA8BTi7eURenuNMq9MxefW

https://github.com/zahid-ali-shah/SignatureView