



Module CP50071E

Mobile Application Development

Assignment

Submitted to: Dr. Peter Rosner

Dated: 20/01/2019

Consideration: I developed this project on the University's high-performance PC that was able to handle the emulator. From that PC, I obtained screenshots of myApp and used them for User Guide. I further edited the project on my personal laptop whose 4GB RAM cannot handle the Emulator at all. I am expecting that the uploaded ZipFile would work fine but if it doesn't, I have provided screenshots of a working application. You should also be able to match work through the "Design" in the XML files. Thanks.

User Guide:

This User Guide explains the use of “myApp”, a mobile application that enables you to use the following features.

- Open the app (ideally from your mobile phone, if pre-installed OR by importing “myApp” file into Android Studio, thereby, running it in the Emulator).
- Once “myApp” is successfully run (either on mobile phone or Emulator), you should see a Dialer which allows you to insert your desired number and call it.
- All dialled numbers must start with 0, as per the application specification.
- The dialer screen will not allow you to call numbers longer than 11 digits as per the specification.
- Once you have dialled a number and placed a call, the application should show you a mock call in progress.
- You should be able to end that call and place another call.
- You should be able to manage a list of contacts that you can dial, like a normal phone function used on your mobile phone.
- The Contact interface should further offer you the following features:
 - Create a New contact
 - Edit an existing contact
 - Delete a new contact
 - View list of contacts
 - Make a phone call to any one contact at a time.

Let us explore the various features of “myApp” with screenshots that were taken successfully by running it in the Emulator (Android Studio).

(Please see next page for further illustration)



Figure 1: Home Screen

After running “myApp”, you should land on this Home Screen. Here, you will see the app name (myApp) and two more icons at the bottom of the screen (Ignoring the standard static features of the phone, e.g. time, signal strength, battery life etc.).

The icon on the left, displayed as a person with a +sign, is for creating a new contact. This also implies that you have yet to create a contact.

The icon on the right, displayed as a small keyboard is the dialer pad. Clicking this would take you to a new screen to start making a call.

On the Dialer screen, you will see a touch KeyPad to start dialing a number, evidently starting with 0 (which is highlighted in black and white, with other digits greyed-out). You will also see a Person icon on the top right hand of the mobile screen, which will navigate you to your contacts list, should you wish to call one of your contacts. However, we have just started using this app and as such, we have not added any contacts. This will be discussed later.



Figure 2: Dialer Screen

Once you have pressed 0, all other numbers also appear in black and white, meaning you can start typing the other digits of your desired number.

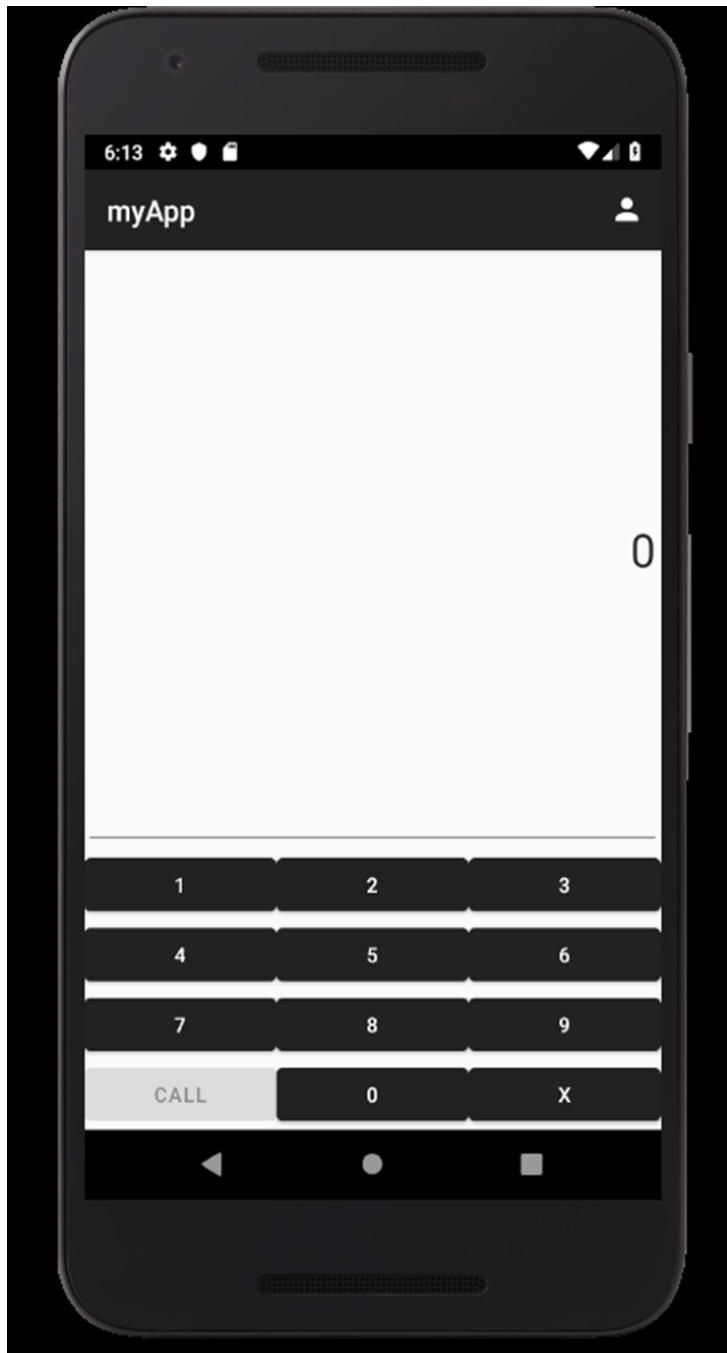


Figure 3: After Pressing 0

The X sign enables to delete any erroneous digits during this process. Once you have entered 11 digits of your number, the CALL button at the bottom left also becomes black and white, meaning you can use it.

Press CALL button (at the bottom left of your screen) and see the magic of this app!



Figure 4: After entering 11 digits

You will see a new screen showing a call being made to the number you entered (**calling....**). Please note that this is a mock call and as such, you are not calling anybody on their phone.

You will see a black and white icon, similar to a Phone Handle. Pressing this icon will end the call.

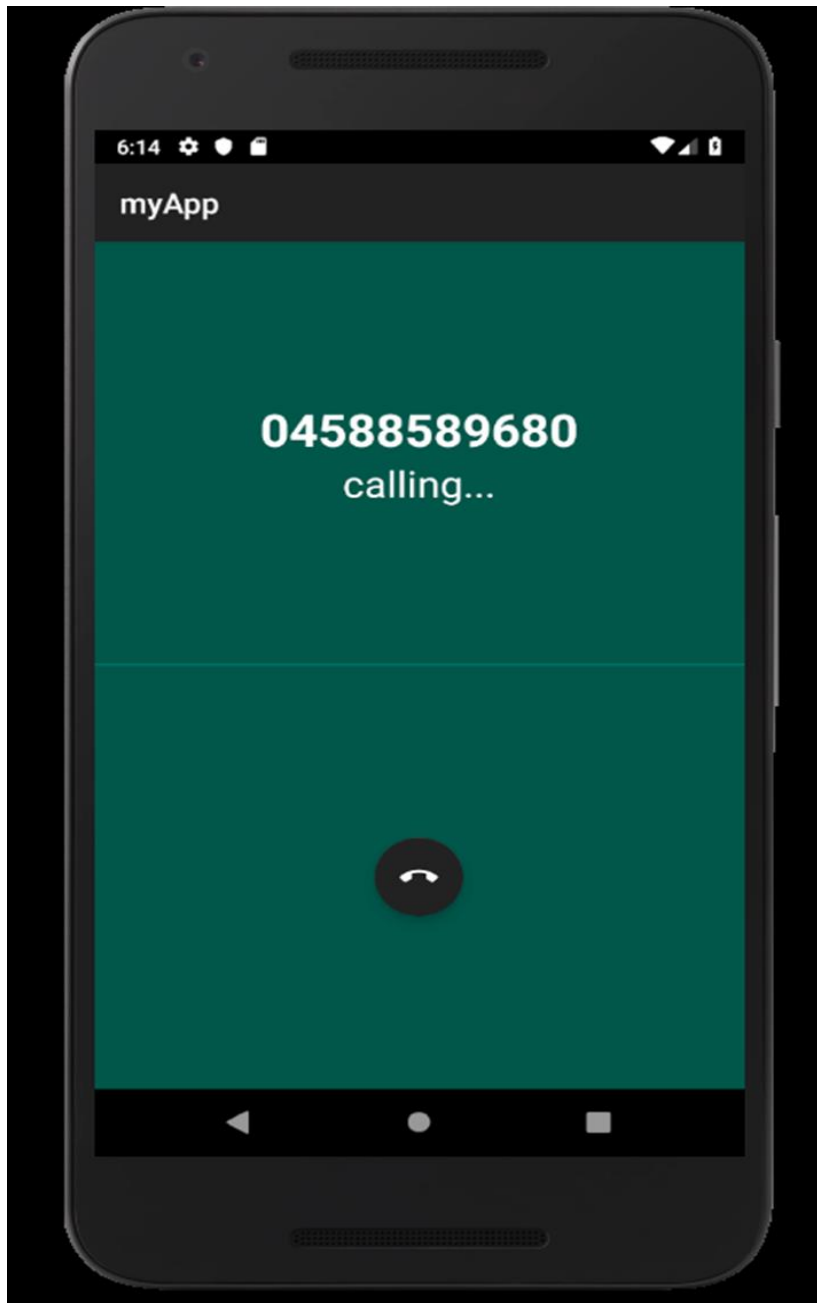


Figure 5: Calling the Number

Now return to either the application's Home Screen (Figure 1) and click the icon showing a Person with a + sign. Clicking this will take you to a new screen for adding a new Contact.

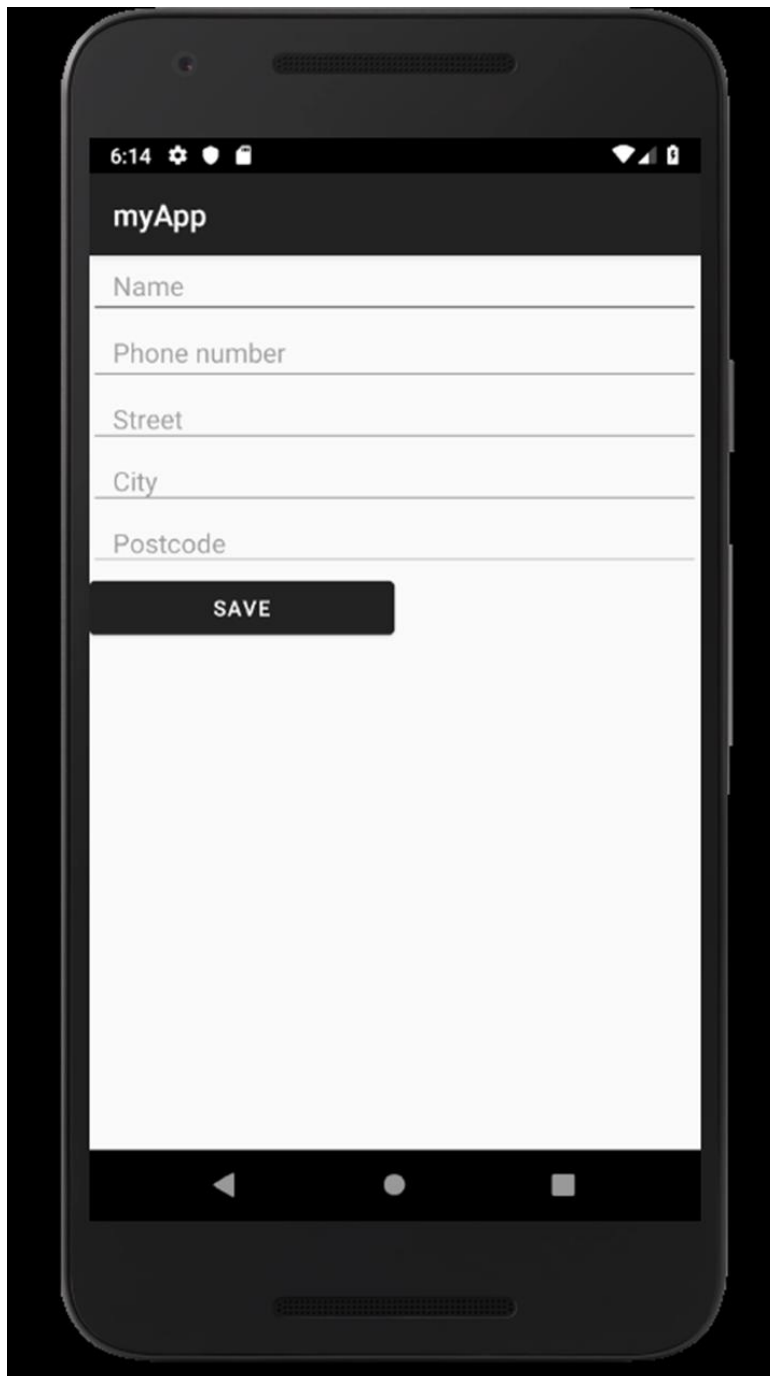
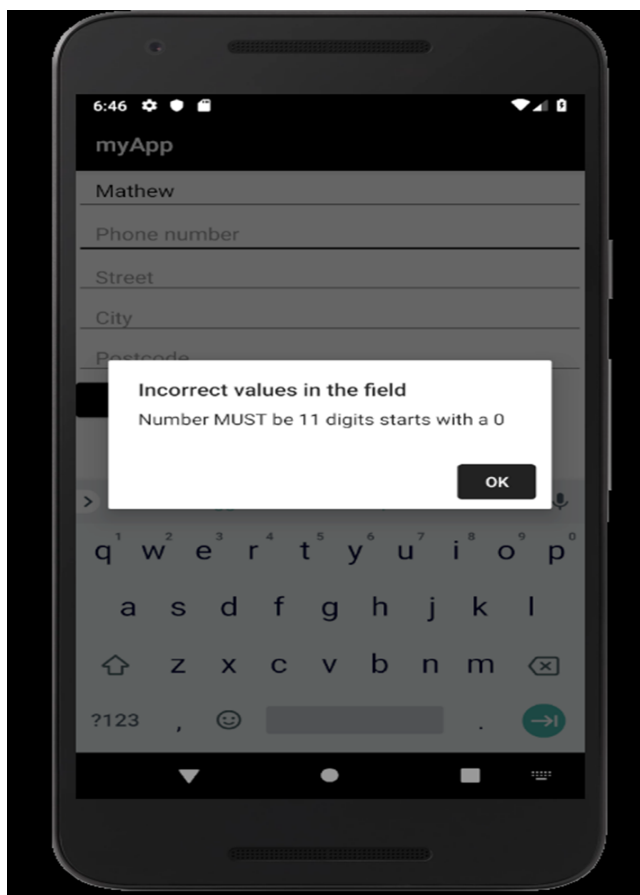
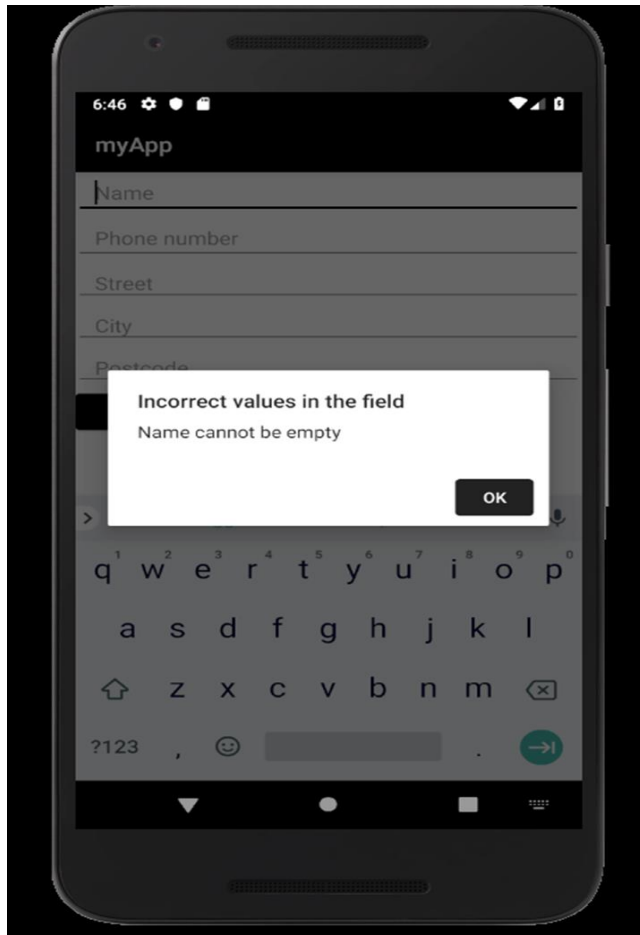


Figure 6: Add a Contact

The moment you place your finger cursor on any of the above fields, a fully functional keyboard will appear allowing to you enter the required details.

You will be expected to enter Contact Name with spaces, phone number comprised of 11 digits, Street, City and a valid Post Code. You cannot leave any fields empty. (See next page)



Once you have completed the details, press “SAVE” button, the purpose of which is quite self-explanatory.

Congratulations! You have successfully added your first contact.

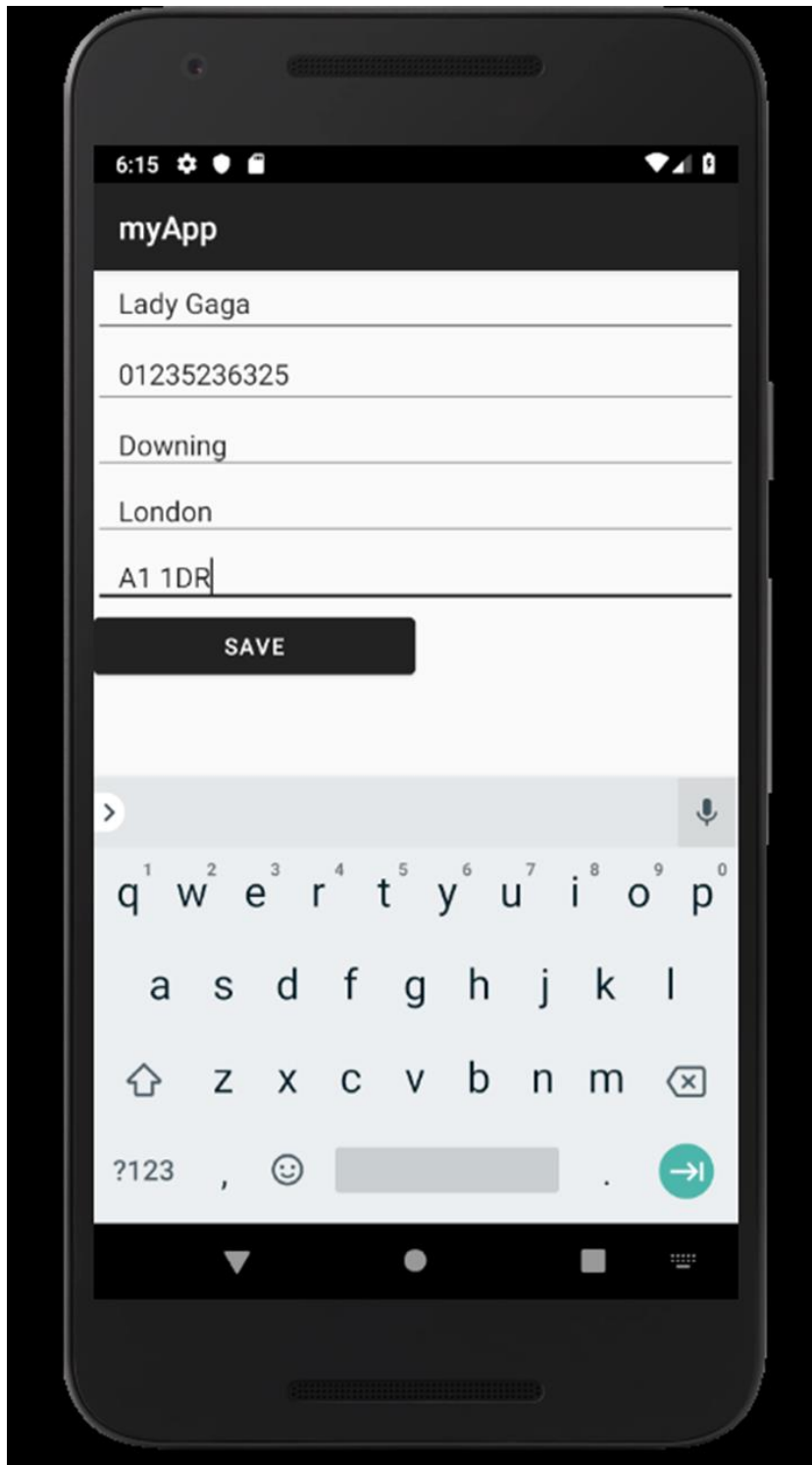


Figure 7: Adding Contact Details

You can change Lady Gaga's contact details by clicking the pencil icon, which essentially lets you edit a given contact and save details. Not only that, you can also "DELETE" it.

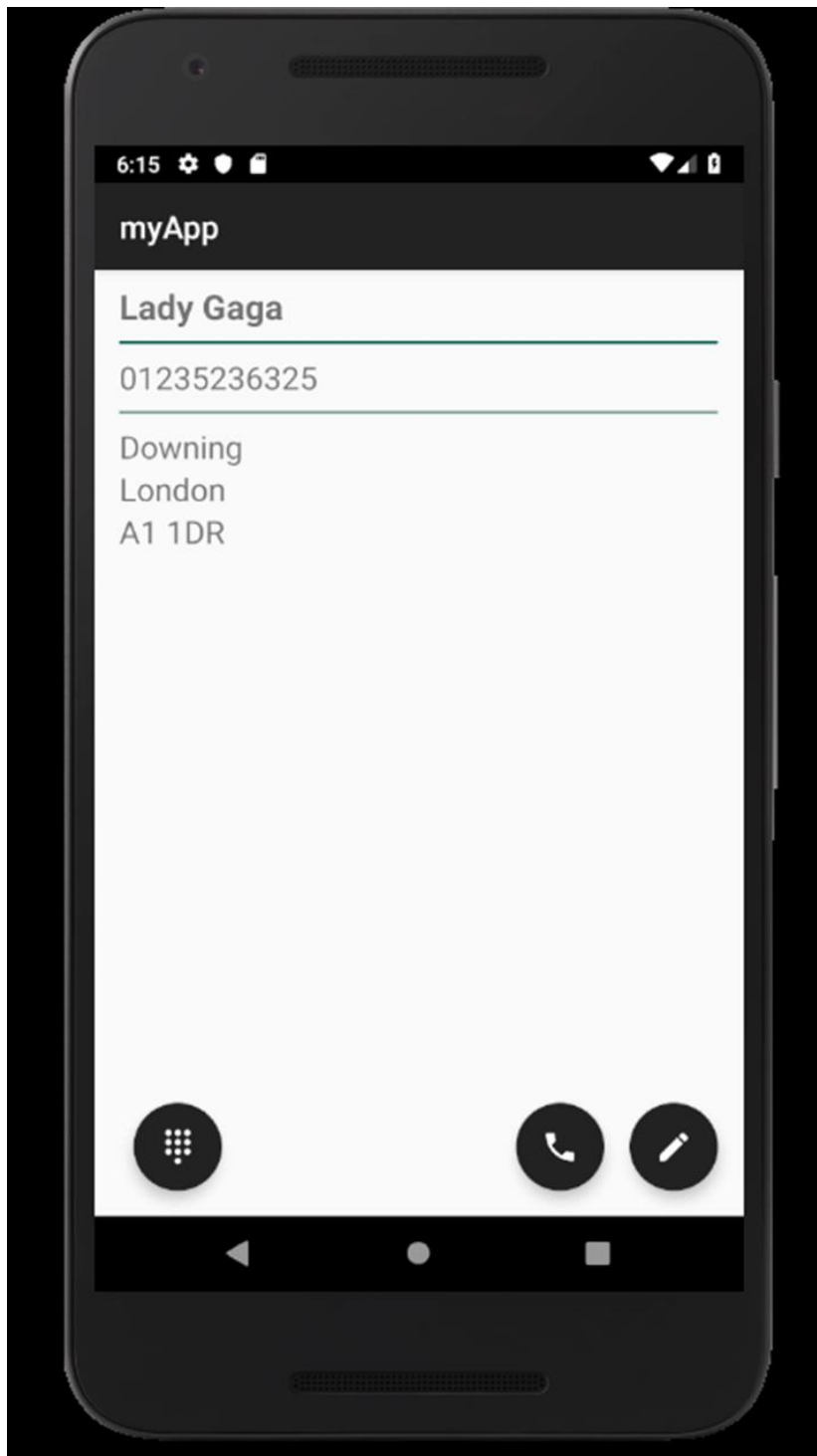


Figure 8: Contact Created and is Editable

Be tempted to add more contacts, edit and/or delete them as demonstrated on the next page.

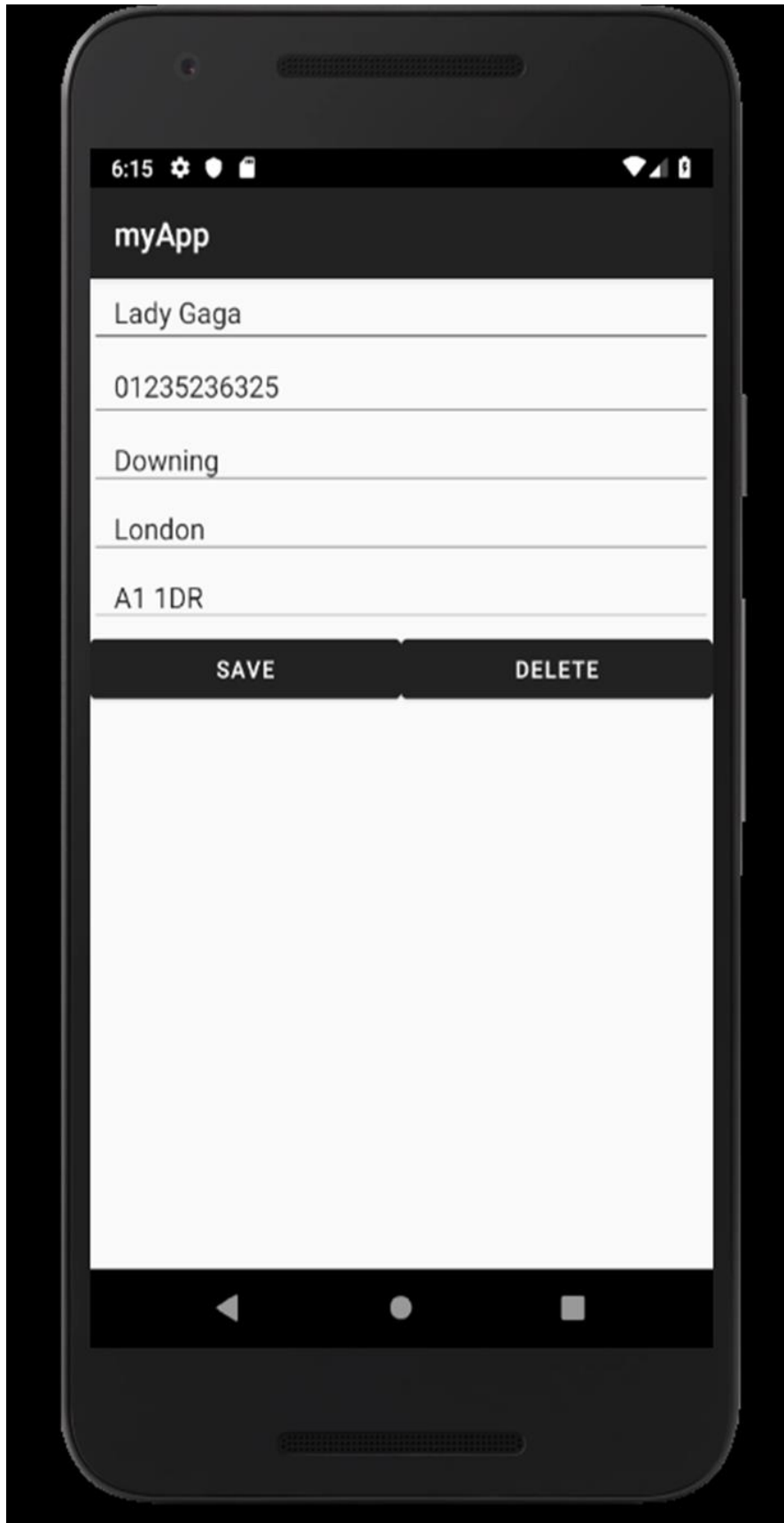


Figure 9: Edit, Save, Delete a Contact

When you go to your Home screen, you will a list of Contacts you have already added, while deleted Contacts will no longer appear (Lady Gaga in this case).

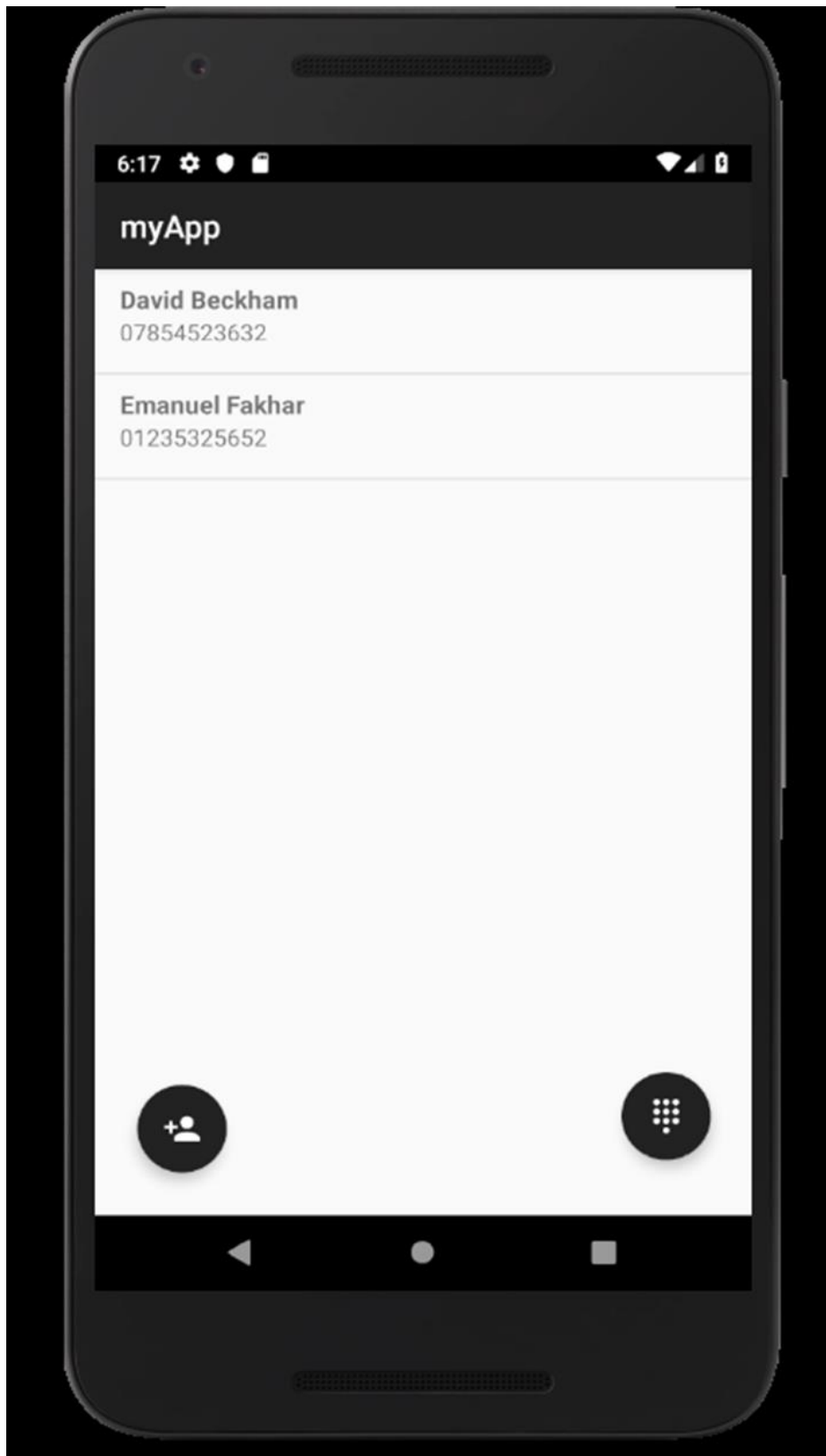


Figure 10: Lady Gaga deleted. New Contacts Added.

Design and Implementation:

The most significant part of learning about Mobile Application Development was the first two weeks, where we studied how Android Studio works when developing any app. The underlying principle is that for each activity, we have Java class or classes that underpin the logic of the application. The second equally important part is the visual enablement of that logic using XML. Connecting one activity to another (essentially, change of scene or behaviour on / to the app or app's content) requires the user of Intents.

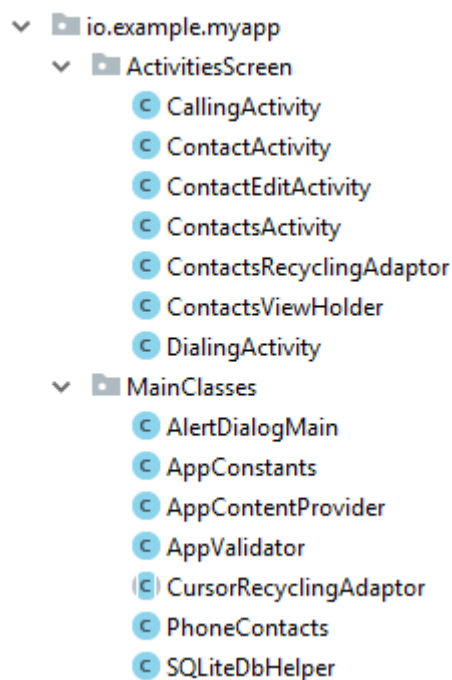
By definition, an Intent is a simple message object that is used to communicate between android components such as activities, content providers, broadcast receivers and services. Intents are also used to transfer data between activities. Be it implicit or explicit, four key functions that an intent performs (using a range of pre-defined methods) are:

- For Launching an Activity
- To start a New Service
- For Broadcasting Messages
- To Display a list of contacts in ListView

We also learnt that Android Studio is a highly sophisticated Development environment that auto-generates complementary files and manages automatic download and configuration of dependencies or other libraries using powerful tools such as Gradle.

For my project, I also acquired Persistence services from SQLite database to set and enable the use of values for Contact List.

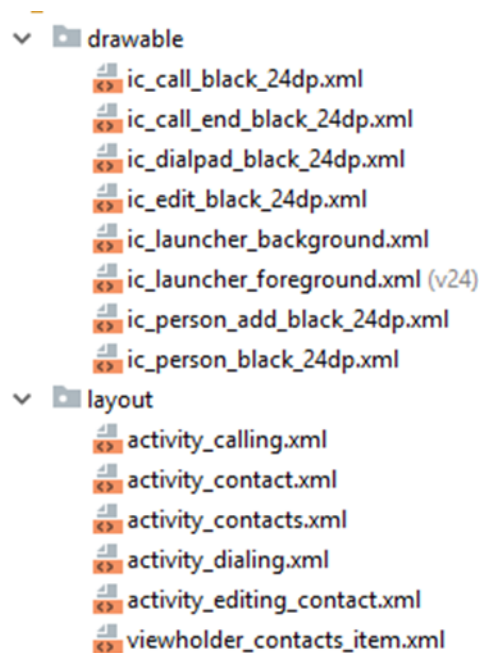
My project's main classes have been divided into two categories:



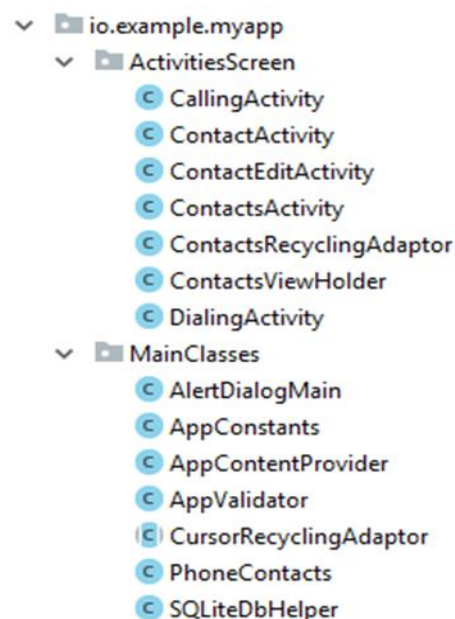
The “ActivitiesScreen” Folder pertains to java classes that handle user activities when running an app on the device.

The “MainClasses” Folder is divided across Java classes that deal with Validations, Print Dialogues and managing Persistence services using SQLite.

The corresponding Layout Folder Structure is as under (comprised of XML files). As aforementioned, the most significant relationship for any app to work is between the logic (Activity Classes with Java code) and the Presentation (XML files).



Touching upon each of the Java Classes that form the business logic of myApp:



Main Classes:

The AlertDialogMain class was developed using the in-built AlertDialog class to provide key validation and print dialogues.

This AlertDialogMain was further made specific by developing the AppValidator classes where I have used Java Regex. Regex provides support for the use of regular expressions when validating data input activities.

If you look in the AppValidator class, I have used Regex to prompt user when they are entering an invalid postcode, or a number less than 11 digits or if required fields (Name, Address lines) were left empty.

```
String regex = "[A-Z]{1,2}[0-9R][0-9A-Z]?[0-9][ABD-HJLNP-UW-Z]{2}";
Pattern pattern = Pattern.compile(regex);
Matcher matcher = pattern.matcher(text);

if (!matcher.matches()) {
    return Pair.create( a: false, b: "Invalid Post Code");
} else {
    return Pair.create( a: true, b: "OK");
}
}

private static Pair<Boolean, String> validatePhoneNumber(String text) {
    String regex = "[0]\\d{10}";
    Pattern pattern = Pattern.compile(regex);
    Matcher matcher = pattern.matcher(text);

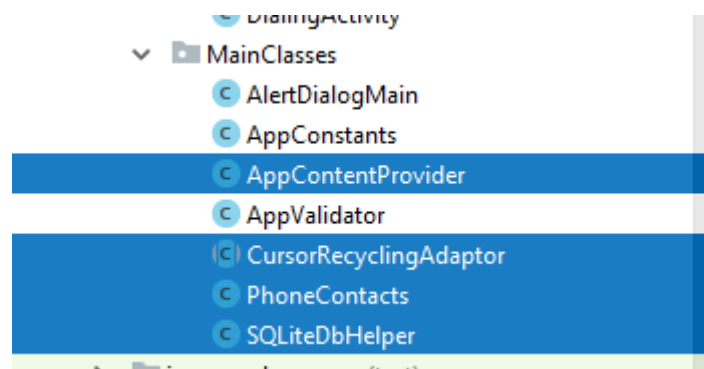
    if (!matcher.matches()) {
        return Pair.create( a: false, b: "Number MUST be 11 digits starts with a 0");
    } else {
        return Pair.create( a: true, b: "OK");
    }
}

private static Pair<Boolean, String> validateName(String text) {
    if (TextUtils.isEmpty(text)) {
        return Pair.create( a: false, b: "Name cannot be empty");
    } else {
        return Pair.create( a: true, b: "OK");
    }
}
```

The AppConstants class was a helpful addition. Typically used to simplify application functionality, the use of Constant values helps achieve better readability compared to just hard-coding some value into our code. In Android we have three choices to define constants:

- final static variable (used for this app)
- Enum
- @IntDef or @StringDef

The SQLite database related coding is available in the remaining classes



The CursorRecyclingAdaptor class is quite self-explanatory. It was developed to act as an adaptor between the SQLite database and the List of Contacts that a user views on the device screen. This class makes use of both the ViewHolder and Adapter (sub-classes of the RecyclerView class).

Adapters provide a binding from an app-specific data set to views that are displayed within a RecyclerView

```

A
A
A/
public abstract class CursorRecyclingAdaptor<VH extends RecyclerView.ViewHolder> extends RecyclerView.Adapter<VH> {

    private final DataSetObserver dataSetObservers;
    protected Cursor cursor;
    protected boolean dataValid;
    private int rowIdColumn;

    public CursorRecyclingAdaptor(Cursor cursor) {
        this.cursor = cursor;
        dataValid = cursor != null;
        rowIdColumn = dataValid ? this.cursor.getColumnIndex( columnName: "_id" ) : -1;
        dataSetObservers = new NotifyingDataSetObserver();
        if (this.cursor != null) {
            this.cursor.registerDataSetObserver(dataSetObservers);
        }
    }

    public Cursor getCursor() { return cursor; }

    @Override
    public int getItemCount() {
        if (dataValid && cursor != null) {
            return cursor.getCount();
        }
    }
}


```

Working with SQLite required the use of specific libraries in Android Studio, which is one of the advantageous discussed earlier in this report.

There are three steps spanning across database implementation:

- Creating a Database
- Creating a table
- Creating Queries.

Using SQLiteOpenHelper within Android Studio




```
public class SQLiteDatabaseHelper extends SQLiteOpenHelper {

    public SQLiteDatabaseHelper(Context context) { super(context,

    @Override
    public void onCreate(SQLiteDatabase db) { db.execSQL(Ph

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion
```

Creating a Database



```
public class PhoneContacts {

    public static final String T_NAME = "Contacts";
    public static final String T_ID = "_id";
    public static final String E_NAME = "name";
    public static final String E_PHONE_NUMBER = "phone_number";
    public static final String E_STREET = "street";
    public static final String E_CITY = "city";
    public static final String E_POSTCODE = "postcode";

    public static final String DROP_TABLE = "DROP TABLE IF EXISTS " + T_NAME;

    public static final String CREATE_TABLE =
        "CREATE TABLE " + T_NAME +
        " (" +
        T_ID + " INTEGER PRIMARY KEY AUTOINCREMENT" +
        ", " +
        E_NAME + " TEXT" +
        ", " +
        E_PHONE_NUMBER + " TEXT" +
        ", " +
        E_STREET + " TEXT" +
        ", " +
        E_CITY + " TEXT" +
        ", " +
        E_POSTCODE + " TEXT" +
        ");";
```

Use of (getReadableDatabase) and (getWritableDatabase) within Android for Query Management.

```
public String getType(@NonNull Uri uri) {
    Log.d(TAG, msg: "getType() called with: uri = [" + uri + "]");

    return BuildConfig.APPLICATION_ID + ".item";
}

@Override
public Cursor query(@NonNull Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder) {
    Log.d(TAG, msg: "query() called with: uri = [" + uri + "], projection = [" + projection + "], selection = [" + selection + "], selectionArgs = [" + selectionArgs + "], sortOrder = [" + sortOrder + "]");

    SQLiteDatabase db = sqliteOpenHelper.getReadableDatabase();
    SQLiteQueryBuilder sqb = new SQLiteQueryBuilder();

    switch (uriMatcher.match(uri)) {
        case CONTACTS:
            sqb.setTables(PhoneContacts.T_NAME);
            break;
        default:
            throw new IllegalArgumentException("uri \"" + uri + "\" not handled!");
    }

    Cursor query = sqb.query(db, projection, selection, selectionArgs, null, null, null, sortOrder);

    if (query != null) {
        query.setNotificationUri(getContext().getContentResolver(), uri);
    }

    return query;
}
```

ActivitiesScreen Classes:

- ▼ ActivitiesScreen
 - CallingActivity
 - ContactActivity
 - ContactEditActivity
 - ContactsActivity
 - ContactsRecyclingAdaptor
 - ContactsViewHolder
 - DialingActivity

These classes were developed to managed activities carried out by the user when interacting with myApp. Th name of the classes is self-explanatory in understanding the relevant activities and have been coded in the myApp project folder zipped and uploaded.



References:

AcadGild. (2019). Android Intent | Introduction to Intent in Android With Examples. [online] Available at: <https://acadgild.com/blog/intent-in-android-introduction> [Accessed 20 Jan. 2019].