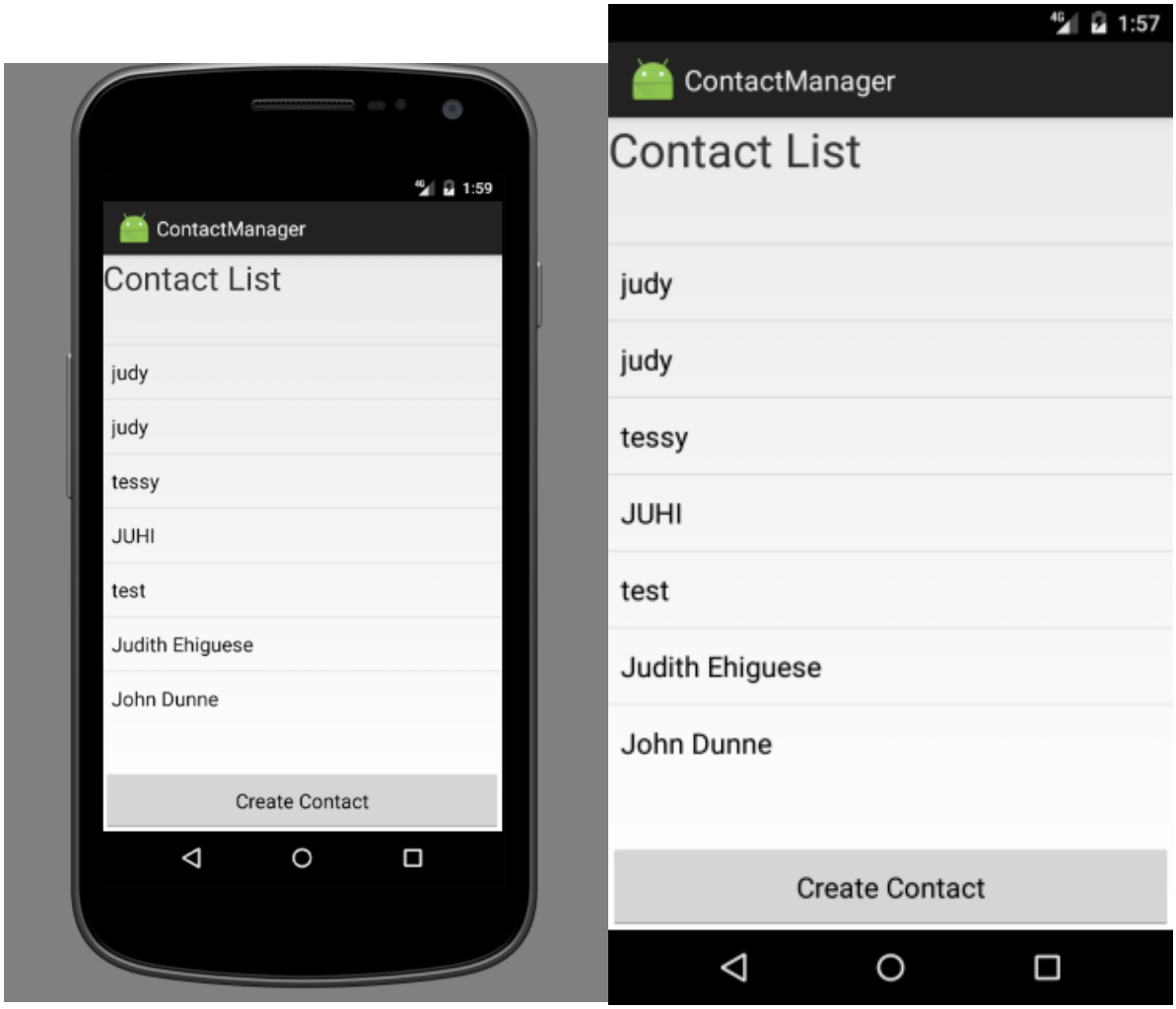
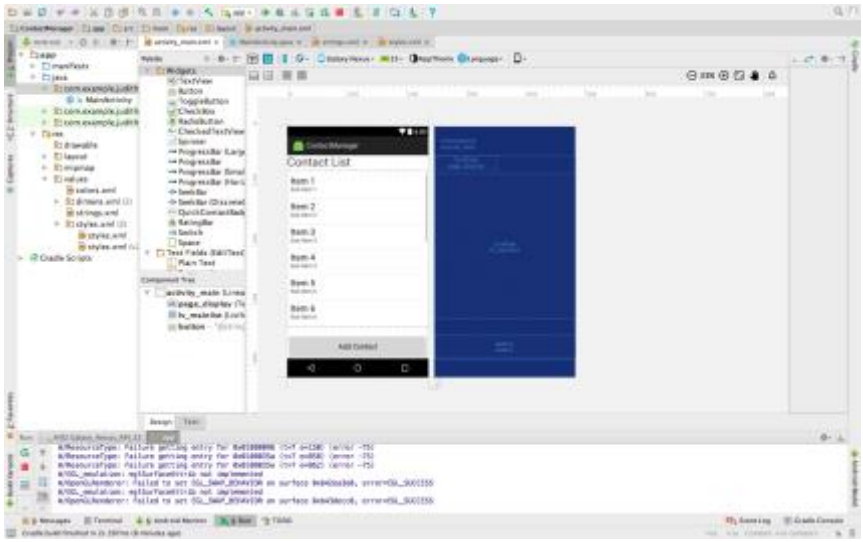
Contact App Manager – Judith Ehiguese





I decided to create a simple UI that would allow the user to navigate around the application smoothly. A contact manager enables the user to view and create contacts. I will describe the steps I used below.

Android Database

Android uses the SQLite Databases for saving its files. SQLite is an open source embeddable database engine written in C. It's self-contained with no external dependencies and is suitable for mobile devices and applications alike. Google exposes the SQLite functionality by using the android.database.sqlite package. A recommended way of using it is to use Cursor to point to return the findings of the queries and using Content Values to save data values in the database.

Contacts Application Objective

We wish to demonstrate using the database functionality in the android stack by making a contact application. In real life, such an application might not be used for saving contacts in the phone itself, it can be used to replace the default contacts application if done properly using the Content Provider class. However our objective here is to use the application to interface with the SQLite database in the phone and save contact information.

Let's Begin!

First, we start out with a new Eclipse project which can be created from File>New>Project and select Android Project from the dialog. These following attributes can be set in the dialog

-Project Name: Contact Manager
-Package Name: com.example.judithehiguese.contactmanager
-Application Name: Contact Manager

Creating the resource file

We would like to add a few strings to the application, so we are not hard coding strings and values into the application.
Open: Res/values/strings.xml

```
<resources>
   <string name="app_name">ContactManager</string>

   <string name="page_display">Contact List</string>
   <string name="create_contact">Create Contact</string>
   <string name="add_contact">Add Contact</string>
   <string name="email">Email</string>
   <string name="home_phone">Home Phone</string>
   <string name="mobile_phone">Mobile Phone</string>
   <string name="name">Name</string>
   <string name="name2">Name:</string>
   <string name="mobile_phone2">Mobile Phone:</string>
   <string name="home_phone2">Home Phone:</string>
   <string name="email2">Email:</string>
</resources>
```

Making the interfaces

Applications in android are designed using the xml files at res/layout.
I have the following res files for each class:

-activity_final.xml
-activity_main.xml
-activity_second.xml

We can choose to create these xml files without creating a class. Here is the activity_main.xml format which allows us to edit the display of text on screen. There are two different layouts, the linear and relative. I decided to use the linear layout because of the list view widget.

Linear Layout : A layout that organizes its children into a single horizontal or vertical row. This xml file is special for list activity as it assigns a list view if the content is available, else it prints an unavailable entry.

Res/layout/activity_main.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.example.judithehiguese.contactmanager.MainActivity"
    android:weightSum="1">

    <TextView
        android:id="@+id/page_display"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/page_display"
        android:textSize="30sp" />

    <ListView
        android:layout_width="wrap_content"
        android:layout_height="373dp"
        android:id="@+id/lv_mainlist"
        android:layout_marginTop="10dp"
        android:layout_marginBottom="40dp"
        />

    <Button
        android:text="@string/create_contact"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/button"
        android:onClick="secondActivityBtn"
        />

</LinearLayout>
```

Relative Layout: Enables you to specify the location of child objects relative to each other or to the parent. The xml file here uses the Relative Layout for laying out the buttons and EditText views for editing the content.

Res/layout/activity_final.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_final"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="com.example.judithehiguese.contactmanager.FinalActivity">

    <TextView
        android:text="@string/name2"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:id="@+id/viewName"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:textSize="24sp" />

    <TextView
        android:text="@string/home_phone2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="18dp"
        android:id="@+id/viewHome"
        android:textSize="24sp"
        android:layout_below="@+id/viewMobile"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />

    <TextView
        android:text="@string/mobile_phone2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:id="@+id/viewMobile"
        android:textSize="24sp"
        android:layout_below="@+id/viewName"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />

    <TextView
        android:text="@string/email2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="23dp"
        android:id="@+id/viewEmail"
        android:textSize="24sp"
        android:layout_below="@+id/viewHome"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />


</RelativeLayout>
```

Making DatabaseHelper

A DatabaseHelper class is created to keep the SQL codes away from the application and avoiding hardcoding everything in the application.
The following fields in our table is needed, the strings will help us access the values later from our application.

Src/main/java/com/example/judithehiguese/contactmanager/DatabaseHelper

/*The table and database name*/

```
//define database name and table name
public static final String DATABASE_NAME = "contact.db";
public static final String TABLE_NAME = "contact_table";
```

/*strings to help us access the values later from our application*/

```
public static final String C_1 = "ID";
public static final String C_2 = "NAME";
public static final String C_3 = "MOBILE";
public static final String C_4 = "HOME";
public static final String C_5 = "EMAIL";
```

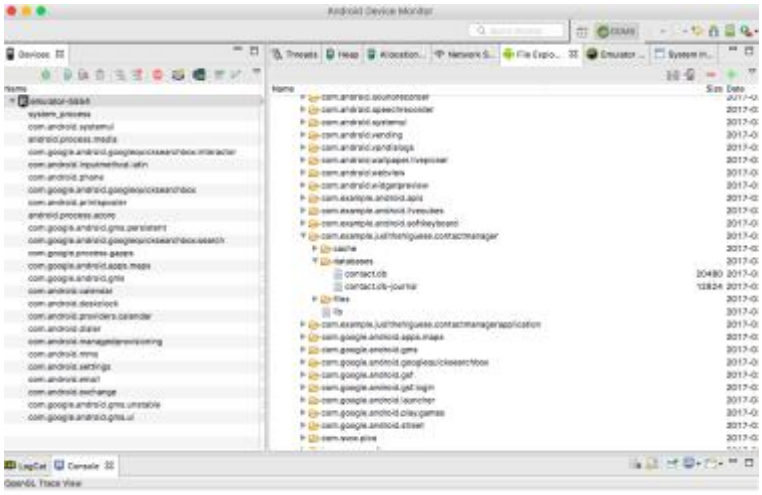
/*SQL code for creating the table*/

```
//creates table when this method is called
@Override
public void onCreate(SQLiteDatabase db) {
   db.execSQL("CREATE TABLE " + TABLE_NAME + " ( " + C_1 + "INTEGER PRIMARY KEY
AUTOINCREMENT," + C_2 + " VARCHAR, " + C_3 + " VARCHAR, " + C_4 + " VARCHAR, " + C_5 + "
VARCHAR);");
}
```

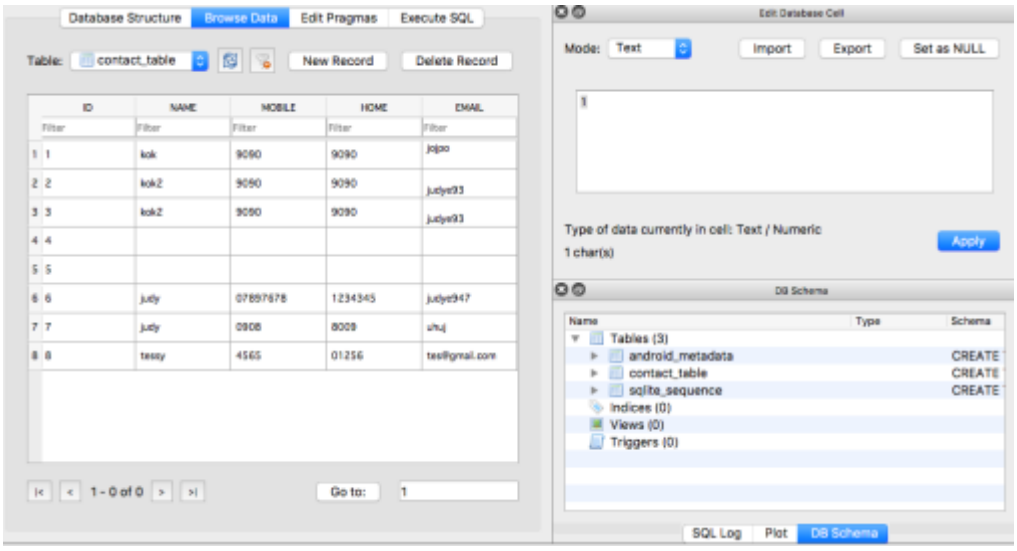
/*Creating and deleting rows*/

```
//method to insert data
public boolean insertData(String name, String mobile, String home, String email){
   SQLiteDatabase db = this.getWritableDatabase(); //create database and table for checking
   ContentValues contentValues = new ContentValues();
   contentValues.put(C_2, name);
   contentValues.put(C_3, mobile);
   contentValues.put(C_4, home);
   contentValues.put(C_5, email);
   long result = db.insert(TABLE_NAME, null, contentValues); //if data is not inserted it will return minus 1
   if(result == -1)
      return false;
   else
      return true;

}
```

Android device monitor enabled me to view my database. Can be found in: Tools/Android/Android Device Monitor



I used DB Browser for SQLite to view what was in my database during the testing stage. Can be found here: http://sqlitebrowser.org



Listing Contacts

We need to list the contacts from the database in the MainActivity.

Src/main/java/com/example/judithehiguese/contactmanager/MainActivity

```
DatabaseHelper myDB;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
```

```
final ListView listView = (ListView) findViewById(R.id.lv_mainlist);
myDB = new DatabaseHelper(this);
```

The above code gets the list view layout from activity_main.xml. We want the application to load the data from the database when it loads.

```
//populate an ArrayList<String> from the database and then view it
ArrayList<String> theList = new ArrayList<>();
Cursor data = myDB.getAllData();
if(data.getCount() == 0){
   Toast.makeText(this, "There are no contents in this list!",Toast.LENGTH_LONG).show();
}else{
   while(data.moveToNext()){
      theList.add(data.getString(1));
      ListAdapter listAdapter = new ArrayAdapter<>(this,android.R.layout.simple_list_item_1,theList);
      listView.setAdapter(listAdapter);
      registerForContextMenu(listView);
   }

}


listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
   @Override
   public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
      Intent intent = new Intent(MainActivity.this, FinalActivity.class);
      intent.putExtra("cName", listView.getItemAtPosition(position).toString());
      intent.putExtra("cMobile", listView.getItemAtPosition(position).toString());
      intent.putExtra("cHome", listView.getItemAtPosition(position).toString());
      intent.putExtra("cEmail", listView.getItemAtPosition(position).toString());
      startActivity(intent);

   }
});
```

The code above gets an adapter based on the main activity layout.

Getting Contacts

We want to fetch rows to display.

DatabaseHelper.java

```
public Cursor getAllData(){
   SQLiteDatabase db = this.getWritableDatabase(); //create database and table for checking
   Cursor res = db.rawQuery("SELECT * FROM " + TABLE_NAME, null);
   return res;
}
```

Creating Contact/ Clicks detection

We want to open the contact page for the person to view and edit the details every time the user clicks a list item. What we need to do is, we need to get all the details of the database's current cursor position and send it to the contact editing interface. We use intents to do so. When we spawn an activity, we pass an intent that we want to run the activity with certain characteristics.
Here the Bundles come into picture because now we can pass data in the form of key value pairs in the bundle that is passed to an activity.

MainActivity.java

```
listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        Intent intent = new Intent(MainActivity.this, FinalActivity.class);
        intent.putExtra("cName", listView.getItemAtPosition(position).toString());
        intent.putExtra("cMobile", listView.getItemAtPosition(position).toString());
        intent.putExtra("cHome", listView.getItemAtPosition(position).toString());
        intent.putExtra("cEmail", listView.getItemAtPosition(position).toString());
        startActivity(intent);

    }
});


public void secondActivityBtn(View view){

    Intent intent = new Intent(MainActivity.this, SecondActivity.class);
    startActivity(intent);
}
```

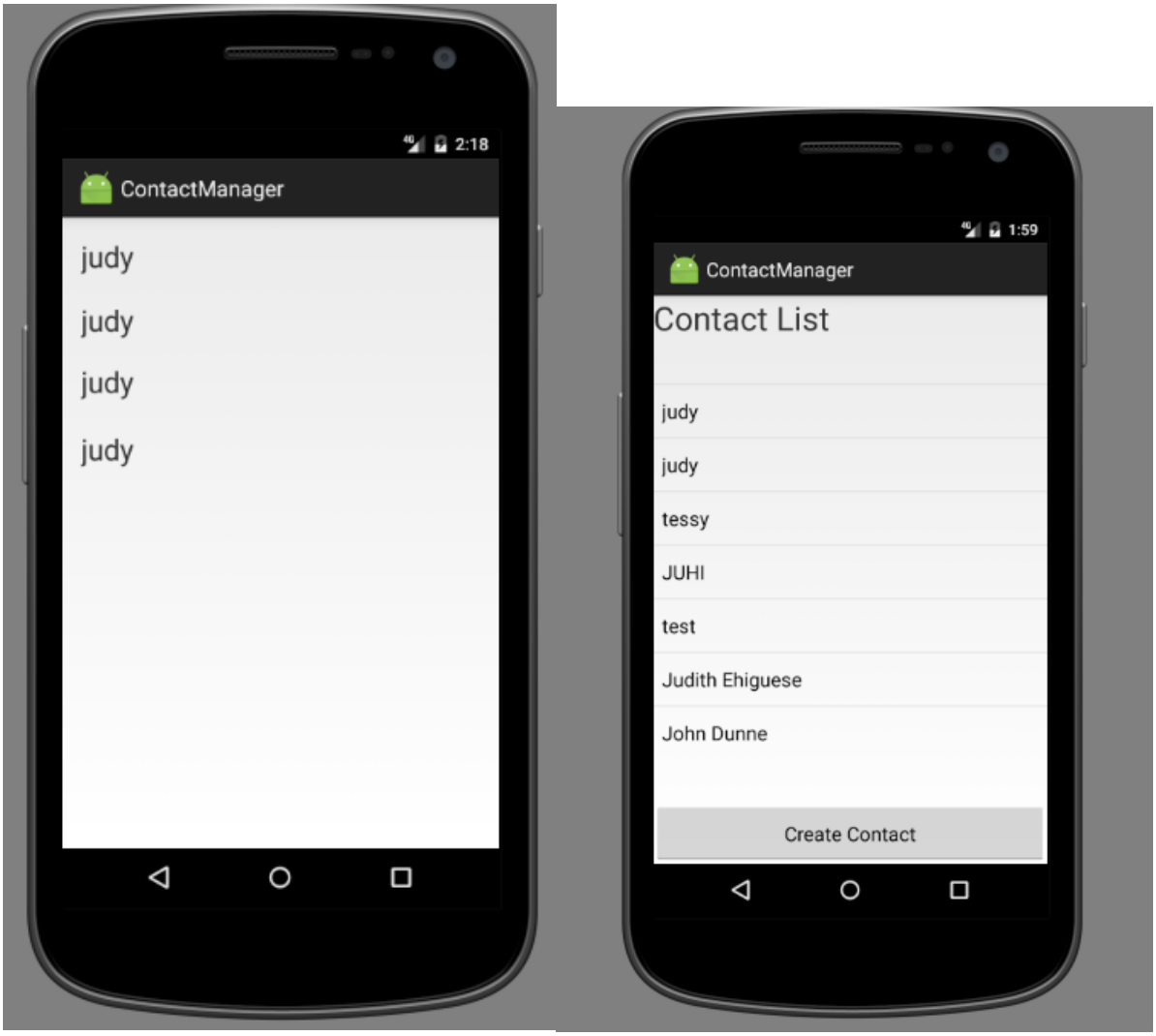
This links us to the second activity where the user can create a new contact

Main Activity :

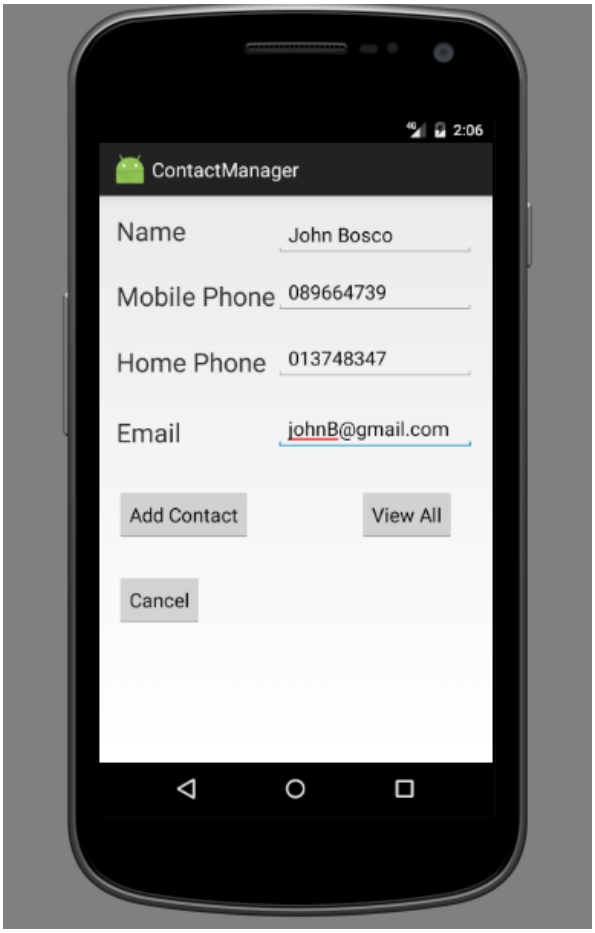
The first activity which contains the main Contact List, is shown in a List View.
The list view allows the user to scroll in search of who they're looking for.
The contact button "Create Contact" button is easy to see and allows the user to navigate to the next activity which will view the contact details.

When a user clicks on a contact name, they can view the contact details in the second activity
When user clicks on the create contact button, it redirects them to the Second Activity:

```
public void secondActivityBtn(View view){

    Intent intent = new Intent(MainActivity.this, SecondActivity.class);
    startActivity(intent);
}
```



Second Activity



This is a relative layout of the page in which the user can create a contact.
Add Contact will send user data to the database.
View All will redirect user back to contact list after a contact has been created.

Cancel will bring them back to the contact list without having to enter data, in case they change their mind.

SecondActivity.java

/*Button to move to main activity once data is added*/

```
public void viewAllBtn(View view){

    Intent intent = new Intent(SecondActivity.this, MainActivity.class);
    startActivity(intent);
}
```

/*Button to move back to main activity if user doesn't want to do anything */

```
public void cancelBtn(View view){

    Intent intent = new Intent(SecondActivity.this, MainActivity.class);
    startActivity(intent);
}
```

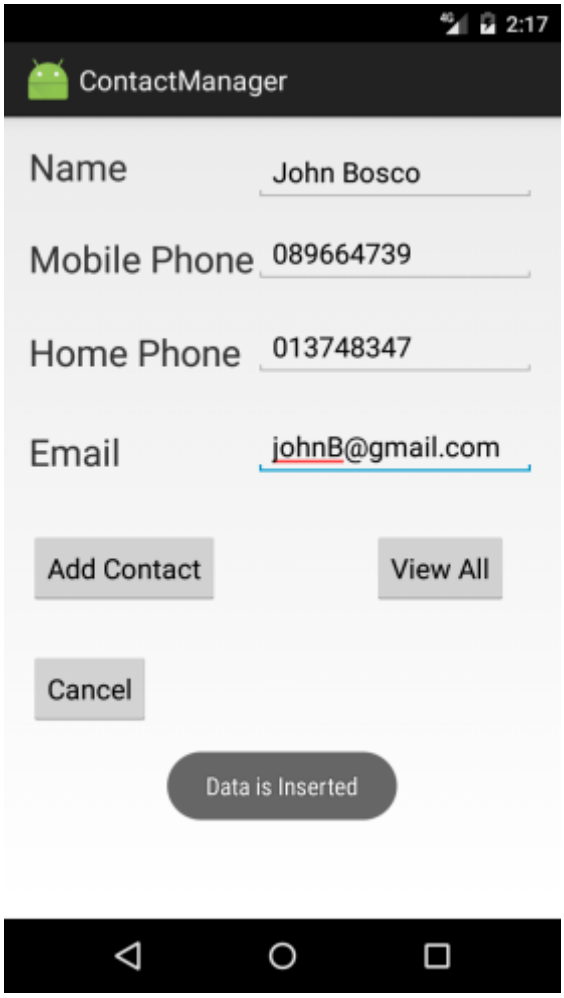
/*function to add data once add data button has been clicked*/

```
btnAddData = (Button)findViewById(R.id.button_add);
AddData();


public void AddData(){
    btnAddData.setOnClickListener(
            new View.OnClickListener(){
```

User will be notified on whether data has been inserted or not into the database. The toast item pops up to display result on screen.

SecondActivity.java

```
if(isInserted=true)
        Toast.makeText(SecondActivity.this, "Data is Inserted", Toast.LENGTH_LONG).show();
    else
        Toast.makeText(SecondActivity.this, "Data is not Inserted", Toast.LENGTH_LONG).show();

}
```



End of Document