Lambton College

MAD-3125

#### Android Development Fundamentals

#### Assignment 2 – Build a Contact App

Requirements:

1. Your contacts list application should allow to:

    1.1 - Search for an existing contact and retrieve the full contact's information

     1.2 - Add new contact

     1.3 - Edit or Delete and existing contact

2. Create a database called "MyContactsList" using SQLite, contains the following 3 tables:

    2.1 - "Contact" table stores the active contact list records.

    2.2 - "DeletedContacts" table that stores any contact record before it is deleted from the "Contact" table

    2.3 - "Log" table where your application records all the transactions against the "Contact" table such as the add new contact, edit a contact record and delete a contact. Each log record must be associated with the date and timestamp when the transaction is performed

3. The record structure of the Contact table is as follow:

    3.1 - **ID**:  INTEGER AUTOINCREMENT (Primary Key)

    3.2 - **NAME**: TEXT (50)

    3.3 - **PHONE**: TEXT (10)

    3.4 - **EMAIL**: TEXT (50)

4. Notes:

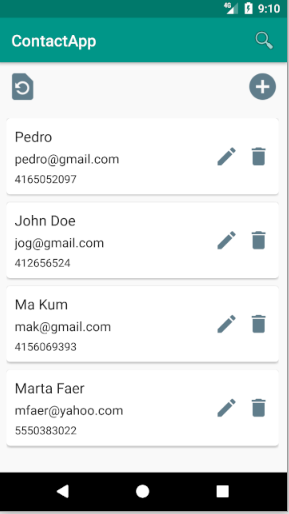
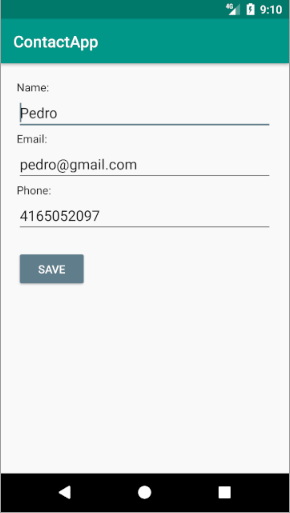
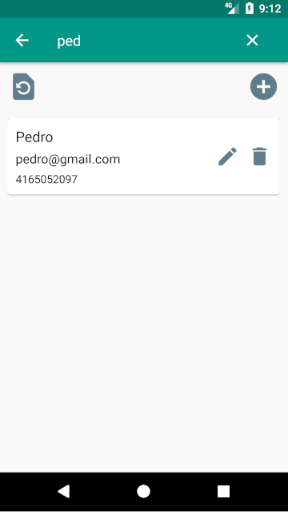
    4.1 - The ID column is an auto incremented field that does not show in any UI window

    4.2 - For simplicity, you can assume that the combination of the 2 columns: Name and Phone would uniquely identify each contact record

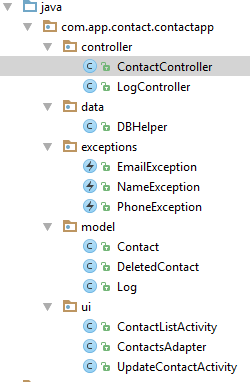
    4.3 - MVC design pattern must be followed in your contacts list application as explained and demonstrated in today's class, otherwise will lose 50% of the assignment's mark

    4.4 - Comments must be added very thoroughly and clearly, otherwise will lose 25% of the assignment's mark

App Screenshots

MVC Packages Structure



Contact List Activity Code – Launcher Activity

**import** android.app.SearchManager;  
**import** android.content.Context;  
**import** android.content.DialogInterface;  
**import** android.content.Intent;  
**import** android.os.Bundle;  
**import** android.support.v7.app.AlertDialog;  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.support.v7.widget.LinearLayoutManager;  
**import** android.support.v7.widget.RecyclerView;  
**import** android.support.v7.widget.SearchView;  
**import** android.view.Menu;  
**import** android.view.MenuInflater;  
**import** android.view.View;  
  
**import** com.app.contact.contactapp.R;  
**import** com.app.contact.contactapp.controller.ContactController;  
**import** com.app.contact.contactapp.model.Contact;  
  
**import** java.util.List;  
  
*// Launcher activity, which is the first activity to be opened. Extends AppCompatActivity which is the Activity of the android support library.  
// ContactListActivity implements AdapterListener so it knows when the user clicked to edit or remove a contact, information that comes from the adapter.  
// Implements and handle view click listeners. e.g. button add contact and button restore page.***public class** ContactListActivity **extends** AppCompatActivity **implements** ContactsAdapter.AdapterListener, View.OnClickListener {  
  
 *//Recyclerview to show all the contacts stored in the database as a list in the screen.* **private** RecyclerView **listContacts**;  
 **private** View **btnAddContact**, **btnRestorePage**;  
 *//Adapter to show handle and show all the contacts as views on the screen.* **private** ContactsAdapter **contactsAdapter**;  
 *//Request code passed to the UpdateContactActivity and used to validate the return.* **public static final int *REQUEST\_CODE*** = 1;  
 *// Constant used as a key to the Intent.putExtra, giving the contact id in case of editing.* **public static final** String ***EXTRA\_CONTACT\_ID*** = **"contact\_id"**;  
  
 *//Oncreate is the first method of the activity life cycle, here we bind the xml view with this activity and bring the views in to be used later.  
 // Also instantiate the adapter giving the own activity as the listener for the buttons.  
 // Request the contact list from the database and fill it in the recycler view.* @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_contact\_list***);  
  
 *// Bringing the views from the xml to java code. findViewById(id).* **this**.**listContacts** = (RecyclerView) findViewById(R.id.***listContacts***);  
 **this**.**btnAddContact** = findViewById(R.id.***btnAddContact***);  
 **this**.**btnRestorePage** = findViewById(R.id.***btnRestorePage***);  
 *// Setting the click listener passing the activity as the handler, since this activity implements OnClickListener this is possible.* **this**.**btnAddContact**.setOnClickListener(**this**);  
 **this**.**btnRestorePage**.setOnClickListener(**this**);  
  
 *// LayoutManager required by the recyclerview to inform which kind of layout will be this recyclerview. e.g. Linear, Grid.* LinearLayoutManager mLayoutManager = **new** LinearLayoutManager(**this**);  
 **this**.**listContacts**.setLayoutManager(mLayoutManager);  
  
  
 *//Instantiate the adapter passing the activity as the listener for the buttons.* **this**.**contactsAdapter** = **new** ContactsAdapter(**this**);  
 **this**.**listContacts**.setAdapter(**contactsAdapter**);  
  
 *// Load contacts from the database and fill the list.* loadContactsIntoList(**null**);  
 }  
  
  
 *// Method to handle the edit button click sent from the adapter. When edit is clicked start activity responsible to edit the contact.* @Override  
 **public void** editContact(Contact contact) {  
 startUpdateActivity(contact.getId());  
 }  
  
 *// Method to handle the delete button click sent from the adapter. When delete is clicked, shows a confirmation popup, if true then  
 // delete from the database, reload the list view and close the dialog. If false just close the dialog.* @Override  
 **public void** deleteContact(**final** Contact contact) {  
  
 *//AlertDialog is a default dialog from Android, setting a message and giving implementation to the buttons click listeners.* AlertDialog.Builder builder = **new** AlertDialog.Builder(**this**);  
 builder.setMessage(**"Delete contact "** + contact.getName() + **"?"**)  
 .setPositiveButton(**"Delete"**, **new** DialogInterface.OnClickListener() {  
 **public void** onClick(DialogInterface dialog, **int** id) {  
 ContactController.*getInstance*(ContactListActivity.**this**).delete(contact);  
 loadContactsIntoList(**null**);  
 dialog.cancel();  
 }  
 })  
 .setNegativeButton(**"Cancel"**, **new** DialogInterface.OnClickListener() {  
 **public void** onClick(DialogInterface dialog, **int** id) {  
 dialog.cancel();  
 }  
 });  
 *//Showing the dialog on the screen.* builder.create().show();  
 }  
  
 *// Click listeners to the add contact and restore page buttons.* @Override  
 **public void** onClick(View view) {  
 *//When add contact is clicked, start the activity which will handle the add contact UpdateContactActivity.  
 //If restore page is clicked then reload the list from the database to the recycler view, restoring the initial state.* **if** (view.getId() == R.id.***btnAddContact***)  
 startUpdateActivity(0);  
 **else if** (view.getId() == R.id.***btnRestorePage***) {  
 loadContactsIntoList(**null**);  
 }  
 }  
  
 *//Start the activity which will handle add or edit contact giving the contact id in case of editing.  
 //Start the activity waiting for the result, using the REQUEST\_CODE for this.* **private void** startUpdateActivity(**int** contactId) {  
 Intent intent = **new** Intent(**this**, UpdateContactActivity.**class**);  
 **if** (contactId > 0)  
 intent.putExtra(***EXTRA\_CONTACT\_ID***, contactId);  
  
 startActivityForResult(intent, ***REQUEST\_CODE***);  
 }  
  
 *//Handle the result from the UpdateContactActivity, if the result is OK then reload the contact list into the list view,  
 //because either a new contact was added or a existing contact was updated.* @Override  
 **protected void** onActivityResult(**int** requestCode, **int** resultCode, Intent data) {  
 **super**.onActivityResult(requestCode, resultCode, data);  
  
 **if** (requestCode == ***REQUEST\_CODE*** && resultCode == ***RESULT\_OK***) {  
 loadContactsIntoList(**null**);  
 }  
 }  
  
 *//Fetch the contacts from the database. If text is false then load all the existing contacts.  
 //If text is set then give the text to the controller to be used in the query to find the contact.  
 //After fetch the list of contacts load into the adapter and notify it that the data was updated.* **private void** loadContactsIntoList(String text) {  
 List<Contact> contactList;  
 **if** (text != **null**)  
 contactList = ContactController.*getInstance*(**this**).findByText(text);  
 **else** contactList = ContactController.*getInstance*(**this**).findAll();  
  
 **contactsAdapter**.setContactList(contactList);  
 **contactsAdapter**.notifyDataSetChanged();  
 }  
  
 *//Receive the intent sent by the Search Bar when user searched for a text. Android sent this intent,  
 //as the activity is "singleTop", just one instance of it will exists then a new intent will be sent through onNewIntent  
 //with the query as a parameter inside the intent.* @Override  
 **protected void** onNewIntent(Intent intent) {  
 *//set intent simply tells the activity that this is the current intent received, then the getExtra will work.* setIntent(intent);  
 handleIntent(intent);  
 }  
  
 *//Handle the received intent from the Search Bar. Extract the query from the intent Extra paramenter with getStringExtra,  
 //SearchManager.QUERY is the default key sent by the Android. With the result load the contacts list based on this query  
 //typed by the user.* **private void** handleIntent(Intent intent) {  
 **if** (Intent.***ACTION\_SEARCH***.equals(intent.getAction())) {  
 String query = intent.getStringExtra(SearchManager.***QUERY***);  
 loadContactsIntoList(query);  
 }  
 }  
  
 *//Instantiate the SearchView to the top menu on the screen. Default Android implementation.  
 //Find the search in the options\_menu xml menu file and connects to the activity.  
 //This same implementation can be find in the documentation.* @Override  
 **public boolean** onCreateOptionsMenu(Menu menu) {  
 MenuInflater inflater = getMenuInflater();  
 inflater.inflate(R.menu.***options\_menu***, menu);  
  
 *// Associate searchable configuration with the SearchView* SearchManager searchManager =  
 (SearchManager) getSystemService(Context.***SEARCH\_SERVICE***);  
 SearchView searchView =  
 (SearchView) menu.findItem(R.id.***search***).getActionView();  
 searchView.setSearchableInfo(  
 searchManager.getSearchableInfo(getComponentName()));  
  
  
 **return true**;  
 }  
}

Contact List Activity XML Code

*<?***xml version="1.0" encoding="utf-8"***?>*<**RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:background="@color/background"**>  
  
 <**ImageView  
 android:id="@+id/btnRestorePage"  
 android:layout\_width="40dp"  
 android:layout\_height="40dp"  
 android:layout\_alignParentStart="true"  
 android:layout\_marginLeft="10dp"  
 android:layout\_marginTop="10dp"  
 android:src="@drawable/ic\_restore"  
 android:tint="@color/colorAccent"** />  
  
 <**ImageView  
 android:id="@+id/btnAddContact"  
 android:layout\_width="40dp"  
 android:layout\_height="40dp"  
 android:layout\_alignParentEnd="true"  
 android:layout\_marginRight="10dp"  
 android:layout\_marginTop="10dp"  
 android:src="@drawable/ic\_add"  
 android:tint="@color/colorAccent"** />  
  
  
 <**android.support.v7.widget.RecyclerView  
 android:id="@+id/listContacts"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:layout\_below="@+id/btnAddContact"  
 android:layout\_marginTop="20dp"  
 android:paddingLeft="10dp"  
 android:paddingRight="10dp"  
 android:scrollbars="vertical"** />  
  
</**RelativeLayout**>

Contact Adapter Code

**import** android.support.v7.widget.RecyclerView;  
**import** android.view.LayoutInflater;  
**import** android.view.View;  
**import** android.view.ViewGroup;  
**import** android.widget.TextView;  
  
**import** com.app.contact.contactapp.R;  
**import** com.app.contact.contactapp.model.Contact;  
  
**import** java.util.List;  
  
*/\*\*  
 \* Created by macstudent on 2017-06-30.  
 \*/  
  
//ContactsAdapter is the Adapter pattern to handle the main contacts list. It is a recyclerView adapter.  
//AdapterListener is a interface to be the bridge between the edit and remove click and the handler which is the activity in this case.***public class** ContactsAdapter **extends** RecyclerView.Adapter<ContactsAdapter.ViewHolder> {  
  
 List<Contact> **contactList**;  
 **private** AdapterListener **listener**;  
  
 *//Receive the AdapterListener as a parameter in the constructor and sets to a object variable. The activity is the listener here.* **public** ContactsAdapter(AdapterListener listener) {  
 **this**.**listener** = listener;  
 }  
  
 **public void** setContactList(List<Contact> contactList) {  
 **this**.**contactList** = contactList;  
 }  
  
 *//AdapterListener is a interface which two methods to be listen by the activity. editContact and deleteContact.* **public interface** AdapterListener {  
 **void** editContact(Contact contact);  
  
 **void** deleteContact(Contact contact);  
 }  
  
 *//Adapter method responsible to inflate the layout for each item of the list, list\_contacts\_item in this case and  
 //Create the ViewHolder giving this layout. ViewHolder is a pattern used for performance since the same view  
 //can be used again when user scroll the list, so with ViewHolder the same instance can be reused.* @Override  
 **public** ContactsAdapter.ViewHolder onCreateViewHolder(ViewGroup parent, **int** viewType) {  
  
 View inflate = LayoutInflater.*from*(parent.getContext())  
 .inflate(R.layout.***list\_contacts\_item***, parent, **false**);  
  
 ViewHolder vh = **new** ViewHolder(inflate);  
 **return** vh;  
 }  
  
 *//Adapter method used to fill the layout of each item with the data object values.  
 //In this case it is filling the list\_contacts\_item view holders with the contactList information.  
 //And setClickListeners and send it to the AdapterListener received in the constructor.* @Override  
 **public void** onBindViewHolder(ContactsAdapter.ViewHolder holder, **int** position) {  
  
 **final** Contact contact = **contactList**.get(position);  
 holder.**txtName**.setText(contact.getName());  
 holder.**txtEmail**.setText(contact.getEmail());  
 holder.**txtPhone**.setText(contact.getPhone());  
  
 holder.**btnEditContact**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 **listener**.editContact(contact);  
 }  
 });  
  
 holder.**btnDeleteContact**.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View view) {  
 **listener**.deleteContact(contact);  
 }  
 });  
  
 }  
  
 *//Adapter method to know the size of the list of models. contactList in this case.* @Override  
 **public int** getItemCount() {  
 **return contactList**.size();  
 }  
  
 *//View holder implementation to hold the views created in order to reuse it. This is an Android Pattern.  
 //Receive the view inflated and bind the views to variables to be used in the adapter.* **public class** ViewHolder **extends** RecyclerView.ViewHolder {  
 **public** TextView **txtName**;  
 **public** TextView **txtEmail**;  
 **public** TextView **txtPhone**;  
 **public** View **btnEditContact**;  
 **public** View **btnDeleteContact**;  
  
 **public** ViewHolder(View itemView) {  
 **super**(itemView);  
  
 **txtName** = itemView.findViewById(R.id.***txtName***);  
 **txtEmail** = itemView.findViewById(R.id.***txtEmail***);  
 **txtPhone** = itemView.findViewById(R.id.***txtPhone***);  
  
 **btnEditContact** = itemView.findViewById(R.id.***btnEditContact***);  
 **btnDeleteContact** = itemView.findViewById(R.id.***btnDeleteContact***);  
 }  
 }  
}

Update Contact Activity Code

**import** android.os.Bundle;  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.view.View;  
**import** android.widget.Button;  
**import** android.widget.EditText;  
**import** android.widget.Toast;  
  
**import** com.app.contact.contactapp.R;  
**import** com.app.contact.contactapp.controller.ContactController;  
**import** com.app.contact.contactapp.model.Contact;  
  
**import static** com.app.contact.contactapp.ui.ContactListActivity.***EXTRA\_CONTACT\_ID***;  
  
*/\*\*  
 \* Created by Pedro on 2017-06-29.  
 \*/  
//Activity responsible to handle the Add and Edit of Contacts objects in the database.  
// This activity can received a contact\_id meaning that it will be an edit action or not meaning that will be an add action.***public class** UpdateContactActivity **extends** AppCompatActivity **implements** View.OnClickListener {  
  
 **private** Contact **contact**;  
 **private** EditText **edtName**, **edtEmail**, **edtPhone**;  
 **private** Button **btnSave**;  
 **private boolean isUpdating**;  
  
 *//Bind the xml view to the activity through setContentView and bind the xml views to java Objects through the findViewById.* @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_update\_contact***);  
  
 **this**.**edtName** = (EditText) findViewById(R.id.***edtName***);  
 **this**.**edtEmail** = (EditText) findViewById(R.id.***edtEmail***);  
 **this**.**edtPhone** = (EditText) findViewById(R.id.***edtPhone***);  
 **this**.**btnSave** = (Button) findViewById(R.id.***btnSave***);  
  
 **this**.**btnSave**.setOnClickListener(**this**);  
  
 *//Verify if has the intent and then fill the views with the received contact.* **if** (getIntent().hasExtra(***EXTRA\_CONTACT\_ID***)) {  
 fillScreen();  
 }  
 }  
  
 *//Handle the save button click. Get the textx from the EditTexts Views and save to local variables.  
 //Set the values typed by the user to the contact and call the insert or updated to the database.  
 //If no error then set the result as ok and finish the activity, returning the message to the activity caller  
 //as result ok then the previous activity can update the list.  
 //If a error happened then show a Toast with the message inside the Exception create by the Controller.* @Override  
 **public void** onClick(View view) {  
 String name = **this**.**edtName**.getText().toString();  
 String email = **this**.**edtEmail**.getText().toString();  
 String phone = **this**.**edtPhone**.getText().toString();  
  
 **if** (!**isUpdating**) {  
 **this**.**contact** = **new** Contact();  
 }  
  
 **this**.**contact**.setName(name);  
 **this**.**contact**.setEmail(email);  
 **this**.**contact**.setPhone(phone);  
  
 **try** {  
 **if** (!**isUpdating**) {  
 ContactController.*getInstance*(**this**).insert(**contact**);  
 } **else** {  
 ContactController.*getInstance*(**this**).update(**contact**);  
 }  
 setResult(***RESULT\_OK***);  
 finish();  
 } **catch** (Exception exception) {  
 showToast(exception.getMessage());  
 }  
 }  
  
 *//show Toast with the message received as parameter. LENGTH\_SHORT to be a Toast which will disappear fast.* **private void** showToast(String message) {  
 Toast.*makeText*(**this**, message, Toast.***LENGTH\_SHORT***).show();  
 }  
  
 *//If is a edit then fetch the contact with the contact\_id received from the previous activity and  
 //fill the EditTexts with the contact information from the database.* **private void** fillScreen() {  
 **int** contactId = getIntent().getIntExtra(***EXTRA\_CONTACT\_ID***, 0);  
 **this**.**contact** = ContactController.*getInstance*(**this**).find(contactId);  
  
 **this**.**edtName**.setText(**contact**.getName());  
 **this**.**edtEmail**.setText(**contact**.getEmail());  
 **this**.**edtPhone**.setText(**contact**.getPhone());  
 }  
}

Update Contact Activity XML Code

*<?***xml version="1.0" encoding="utf-8"***?>*<**LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:background="@color/background"  
 android:orientation="vertical"  
 android:padding="20dp"**>  
  
 <**TextView  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:text="Name:"  
 android:textColor="@color/textColor"** />  
  
 <**EditText  
 android:id="@+id/edtName"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:inputType="textPersonName"** />  
  
 <**TextView  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:text="Email:"  
 android:textColor="@color/textColor"** />  
  
 <**EditText  
 android:id="@+id/edtEmail"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:inputType="textEmailAddress"** />  
  
 <**TextView  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:text="Phone:"  
 android:textColor="@color/textColor"** />  
  
 <**EditText  
 android:id="@+id/edtPhone"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:inputType="phone"** />  
  
 <**Button  
 android:id="@+id/btnSave"  
 style="@style/Widget.AppCompat.Button.Colored"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginTop="20dp"  
 android:text="Save"** />  
  
</**LinearLayout**>

Log Model Code

**import** android.provider.BaseColumns;  
  
**import static** com.app.contact.contactapp.model.Log.LogEntry.***TABLE\_NAME***;  
  
*/\*\*  
 \* Created by Pedro on 2017-06-29.  
 \*/  
//Log POJO class to represent the model.  
//Log class will store all the operation under Contact table.  
//It has the contact id as foreign key in the database and it has status of the operation (Inserted, Deleted...)  
//and date when the operation happened.***public class** Log {  
  
 **private int id**;  
 **private int contactId**;  
 **private** String **status**;  
 **private** String **date**;  
  
 **public int** getId() {  
 **return id**;  
 }  
  
 **public void** setId(**int** id) {  
 **this**.**id** = id;  
 }  
  
 **public int** getContactId() {  
 **return contactId**;  
 }  
  
 **public void** setContactId(**int** contactId) {  
 **this**.**contactId** = contactId;  
 }  
  
 **public** String getStatus() {  
 **return status**;  
 }  
  
 **public void** setStatus(String status) {  
 **this**.**status** = status;  
 }  
  
 **public** String getDate() {  
 **return date**;  
 }  
  
 **public void** setDate(String date) {  
 **this**.**date** = date;  
 }  
  
 *//Base column names and table name to be used by the database and controllers.* **public static class** LogEntry **implements** BaseColumns {  
 **public static final** String ***TABLE\_NAME*** = **"log"**;  
 **public static final** String ***COLUMN\_CONTACT\_ID*** = **"contact\_id"**;  
 **public static final** String ***COLUMN\_STATUS*** = **"status"**;  
 **public static final** String ***COLUMN\_DATE*** = **"date"**;  
 }  
  
 *//Create table query giving the contact id as the foreign key referencing the contact table.  
 //All the column names come from the LogEntry created above.* **public static final** String ***SQL\_CREATE\_TABLE*** =  
 **"CREATE TABLE "** + ***TABLE\_NAME*** + **" ("** +  
 LogEntry.***\_ID*** + **" INTEGER PRIMARY KEY AUTOINCREMENT,"** +  
 LogEntry.***COLUMN\_CONTACT\_ID*** + **" INTEGER,"** +  
 LogEntry.***COLUMN\_STATUS*** + **" TEXT,"** +  
 LogEntry.***COLUMN\_DATE*** + **" TEXT,"** +  
 **" FOREIGN KEY("** + LogEntry.***COLUMN\_CONTACT\_ID*** + **") REFERENCES "** + Contact.ContactEntry.***TABLE\_NAME*** + **"("** + Contact.ContactEntry.***\_ID*** + **"))"**;  
}

Deleted Contact Model Code

**import** android.provider.BaseColumns;  
  
**import static** com.app.contact.contactapp.model.DeletedContact.DeletedContactEntry.***TABLE\_NAME***;  
  
*/\*\*  
 \* Created by Pedro on 2017-06-30.  
 \*/  
  
//DeletedContact POJO class to represent the model. This class will store the deleted contacts.  
//It has a id to be used by the database and the contact information.***public class** DeletedContact {  
  
 **private int id**;  
 **private** Contact **contact**;  
  
  
 *//As this class uses contact it just have the table name as different.  
 //All the other column names or validation comes from the contact model.* **public static class** DeletedContactEntry **implements** BaseColumns {  
 **public static final** String ***TABLE\_NAME*** = **"deleted\_contact"**;  
 }  
  
 *//SQL scrypt to create the table using the column names from the contact model.* **public static final** String ***SQL\_CREATE\_TABLE*** =  
 **"CREATE TABLE "** + ***TABLE\_NAME*** + **" ("** +  
 DeletedContactEntry.***\_ID*** + **" INTEGER PRIMARY KEY AUTOINCREMENT,"** +  
 Contact.ContactEntry.***COLUMN\_NAME*** + **" TEXT,"** +  
 Contact.ContactEntry.***COLUMN\_PHONE*** + **" TEXT,"** +  
 Contact.ContactEntry.***COLUMN\_EMAIL*** + **" TEXT)"**;  
}

Contact Model Code

**import** android.provider.BaseColumns;  
  
**import static** com.app.contact.contactapp.model.Contact.ContactEntry.***TABLE\_NAME***;  
  
*/\*\*  
 \* Created by Pedro on 2017-06-29.  
 \*/  
  
//Contact POJO class to represent the model. It has 4 attributes as the requirements asked for.  
// All the attributes have getters and setters which are used in the App logic.***public class** Contact {  
  
 **private int id**;  
 **private** String **name**;  
 **private** String **phone**;  
 **private** String **email**;  
  
 **public int** getId() {  
 **return id**;  
 }  
  
 **public void** setId(**int** id) {  
 **this**.**id** = id;  
 }  
  
 **public** String getName() {  
 **return name**;  
 }  
  
 **public void** setName(String name) {  
 **this**.**name** = name;  
 }  
  
 **public** String getPhone() {  
 **return phone**;  
 }  
  
 **public void** setPhone(String phone) {  
 **this**.**phone** = phone;  
 }  
  
 **public** String getEmail() {  
 **return email**;  
 }  
  
 **public void** setEmail(String email) {  
 **this**.**email** = email;  
 }  
  
 *//Create a static class to represents the database values, table name, column names and validations of size.* **public static class** ContactEntry **implements** BaseColumns {  
 **public static final** String ***TABLE\_NAME*** = **"contact"**;  
 **public static final** String ***COLUMN\_NAME*** = **"name"**;  
 **public static final** String ***COLUMN\_PHONE*** = **"phone"**;  
 **public static final** String ***COLUMN\_EMAIL*** = **"email"**;  
  
 **public static final int *MAX\_NAME\_LENGTH*** = 50;  
 **public static final int *MAX\_EMAIL\_LENGTH*** = 50;  
 **public static final int *MAX\_PHONE\_LENGTH*** = 10;  
 }  
  
 *//This class also has the create table query to be run by the database.  
 //Id in this case is the primary key and auto increment so the database will set this value when inserted.* **public static final** String ***SQL\_CREATE\_TABLE*** =  
 **"CREATE TABLE "** + ***TABLE\_NAME*** + **" ("** +  
 ContactEntry.***\_ID*** + **" INTEGER PRIMARY KEY AUTOINCREMENT,"** +  
 ContactEntry.***COLUMN\_NAME*** + **" TEXT,"** +  
 ContactEntry.***COLUMN\_PHONE*** + **" TEXT,"** +  
 ContactEntry.***COLUMN\_EMAIL*** + **" TEXT)"**;  
  
}

Exceptions Code

*//Exception used by the controller to inform the activity when the Contact Phone is not valid.***public class** PhoneException **extends** Exception {  
 **public** PhoneException(String message) {  
 **super**(message);  
 }  
}

*//Exception used by the controller to inform the activity when the Contact Name is not valid.***public class** NameException **extends** Exception {  
 **public** NameException(String message) {  
 **super**(message);  
 }  
}

*//Exception used by the controller to inform the activity when the Contact Email is not valid.***public class** EmailException **extends** Exception {  
 **public** EmailException(String message) {  
 **super**(message);  
 }  
}

DB Helper Code

**import** android.content.Context;  
**import** android.database.sqlite.SQLiteDatabase;  
**import** android.database.sqlite.SQLiteOpenHelper;  
  
**import** com.app.contact.contactapp.model.Contact;  
**import** com.app.contact.contactapp.model.DeletedContact;  
**import** com.app.contact.contactapp.model.Log;  
  
  
*/\*\*  
 \* Created by Pedro on 2017-06-29.  
 \*/  
//DBHelper is the class the give access to the Database, it extends from SQLiteOpenHelper, which is an  
//Android class that gives access to the database.***public class** DBHelper **extends** SQLiteOpenHelper {  
  
 *//Constant that holds the database version.* **public static final int *DATABASE\_VERSION*** = 1;  
 *//Constant that holds the database name.* **public static final** String ***DATABASE\_NAME*** = **"MyContactsList.db"**;  
  
 *//Constructor receives a Context object and gives to the super class as required.* **public** DBHelper(Context context) {  
 **super**(context, ***DATABASE\_NAME***, **null**, ***DATABASE\_VERSION***);  
 }  
  
 *//Execute the models create table queries. This method is called by the Android system on the first interaction with the database.* @Override  
 **public void** onCreate(SQLiteDatabase db) {  
 db.execSQL(Contact.***SQL\_CREATE\_TABLE***);  
 db.execSQL(Log.***SQL\_CREATE\_TABLE***);  
 db.execSQL(DeletedContact.***SQL\_CREATE\_TABLE***);  
 }  
  
 *//Method called by the Android when a upgraded in the Database version occurs.  
 //If so, drop all tables and recreate. It is not being used in this App, since we do not upgrade the DATABASE\_VERSION.* @Override  
 **public void** onUpgrade(SQLiteDatabase db, **int** oldVersion, **int** newVersion) {  
 db.execSQL(**"DROP TABLE IF EXISTS "** + Contact.ContactEntry.***TABLE\_NAME***);  
 db.execSQL(**"DROP TABLE IF EXISTS "** + Log.LogEntry.***TABLE\_NAME***);  
 db.execSQL(**"DROP TABLE IF EXISTS "** + DeletedContact.DeletedContactEntry.***TABLE\_NAME***);  
 onCreate(db);  
 }  
}

Log Controller Code

**import** android.content.ContentValues;  
**import** android.content.Context;  
**import** android.database.sqlite.SQLiteDatabase;  
  
**import** com.app.contact.contactapp.data.DBHelper;  
**import** com.app.contact.contactapp.model.Log;  
  
**import** java.text.SimpleDateFormat;  
**import** java.util.Date;  
**import** java.util.Locale;  
  
*/\*\*  
 \* Created by Pedro on 2017-06-29.  
 \*/  
  
//Log Controller is the Controller in the MVC pattern. It is responsible to all the Log operations that involves database and business logic.***public class** LogController {  
 *//LogStatus enum is created to maintain the data integrity when the status is sent to the database as Text.* **public enum** LogStatus {  
 ***INSERTED***, ***UPDATED***, ***DELETED***;  
 }  
  
 **private static** LogController *instance* = **null**;  
 **private** DBHelper **dbHelper**;  
  
 *//LogController is a singleton since just one instance is needed.* **public static** LogController getInstance(Context context) {  
 **if** (*instance* == **null**) {  
 *instance* = **new** LogController(context);  
 }  
 **return** *instance*;  
 }  
  
 *//Constructor receives the context and instantiate the DBHelper object to communicate with the database.* **public** LogController(Context context) {  
 **this**.**dbHelper** = **new** DBHelper(context);  
 }  
  
 *//Creates the Log object using the parameters and insert into the database using the DBHelper object.* **public long** insert(**int** contactId, LogStatus logStatus) {  
 Log log = **new** Log();  
 log.setContactId(contactId);  
 log.setStatus(logStatus.toString());  
 log.setDate(getDateTime());  
  
 SQLiteDatabase db = **this**.**dbHelper**.getWritableDatabase();  
 ContentValues contentValues = **new** ContentValues();  
 contentValues.put(Log.LogEntry.***COLUMN\_CONTACT\_ID***, log.getContactId());  
 contentValues.put(Log.LogEntry.***COLUMN\_STATUS***, log.getStatus());  
 contentValues.put(Log.LogEntry.***COLUMN\_DATE***, log.getDate());  
 **return** db.insert(Log.LogEntry.***TABLE\_NAME***, **null**, contentValues);  
 }  
  
 *//Get the current date and time, format it to be inserted as the Log date into the database.* **private** String getDateTime() {  
 SimpleDateFormat dateFormat = **new** SimpleDateFormat(  
 **"yyyy-MM-dd HH:mm:ss"**, Locale.*getDefault*());  
 Date date = **new** Date();  
 **return** dateFormat.format(date);  
 }  
}

Contact Controller Code

**import** android.content.ContentValues;  
**import** android.content.Context;  
**import** android.database.Cursor;  
**import** android.database.sqlite.SQLiteDatabase;  
  
**import** com.app.contact.contactapp.data.DBHelper;  
**import** com.app.contact.contactapp.exceptions.EmailException;  
**import** com.app.contact.contactapp.exceptions.NameException;  
**import** com.app.contact.contactapp.exceptions.PhoneException;  
**import** com.app.contact.contactapp.model.Contact;  
**import** com.app.contact.contactapp.model.DeletedContact;  
  
**import** java.util.ArrayList;  
**import** java.util.List;  
  
*/\*\*  
 \* Created by Pedro on 2017-06-29.  
 \*/  
//Contact Controller is the Controller in the MVC pattern. It is responsible to all the Contact operations that involves database and business logic.***public class** ContactController {  
  
 **private static** ContactController *instance* = **null**;  
 **private** DBHelper **dbHelper**;  
 **private** Context **context**;  
  
 *//Implements a singleton since just one instance is needed of this class.* **public static** ContactController getInstance(Context context) {  
 **if** (*instance* == **null**) {  
 *instance* = **new** ContactController(context);  
 }  
 **return** *instance*;  
 }  
  
 *//Constructor received the context and created the DBHelper object.* **public** ContactController(Context context) {  
 **this**.**dbHelper** = **new** DBHelper(context);  
 **this**.**context** = context;  
 }  
  
 *//Insert the contact object into the database.  
 //Validate the contact and throws exceptions regarding the current error.  
 //After Contact inserted successfully then insert the Log showing that it was Inserted. Calling the LogController for it.* **public long** insert(Contact contact) **throws** EmailException, PhoneException, NameException {  
 validate(contact);  
 SQLiteDatabase db = **this**.**dbHelper**.getWritableDatabase();  
 ContentValues values = **new** ContentValues();  
 values.put(Contact.ContactEntry.***COLUMN\_NAME***, contact.getName());  
 values.put(Contact.ContactEntry.***COLUMN\_PHONE***, contact.getPhone());  
 values.put(Contact.ContactEntry.***COLUMN\_EMAIL***, contact.getEmail());  
  
 **long** rowId = db.insert(Contact.ContactEntry.***TABLE\_NAME***, **null**, values);  
  
 **if** (rowId >= 0) {  
 LogController.*getInstance*(**this**.**context**).insert((**int**) rowId, LogController.LogStatus.***INSERTED***);  
 }  
 **return** rowId;  
 }  
  
 *//Update the contact object into the database.  
 //Validate the contact and throws exceptions regarding the current error.  
 //After Contact updated successfully then insert the Log showing that it was Updated. Calling the LogController for it.* **public long** update(Contact contact) **throws** EmailException, PhoneException, NameException {  
 validate(contact);  
 SQLiteDatabase db = **this**.**dbHelper**.getWritableDatabase();  
 ContentValues values = **new** ContentValues();  
 values.put(Contact.ContactEntry.***COLUMN\_NAME***, contact.getName());  
 values.put(Contact.ContactEntry.***COLUMN\_PHONE***, contact.getPhone());  
 values.put(Contact.ContactEntry.***COLUMN\_EMAIL***, contact.getEmail());  
  
 **int** rowId = db.update(Contact.ContactEntry.***TABLE\_NAME***, values, Contact.ContactEntry.***\_ID*** + **" = ? "**, **new** String[]{Integer.*toString*(contact.getId())});  
  
 **if** (rowId >= 0) {  
 LogController.*getInstance*(**this**.**context**).insert(rowId, LogController.LogStatus.***UPDATED***);  
 }  
  
 **return** rowId;  
 }  
  
 *//Fetch all the Contacts from the database and insert them into a ArrayList and return it.* **public** List<Contact> findAll() {  
 List<Contact> contactList = **new** ArrayList<>();  
 SQLiteDatabase db = **this**.**dbHelper**.getReadableDatabase();  
  
 Cursor cursor = db.rawQuery(**"SELECT \* FROM "** + Contact.ContactEntry.***TABLE\_NAME***, **null**);  
 cursor.moveToFirst();  
  
 **while** (!cursor.isAfterLast()) {  
 contactList.add(makeContact(cursor));  
 cursor.moveToNext();  
 }  
  
 cursor.close();  
 **return** contactList;  
 }  
  
 *//Find the contact which has the giving id in the database, if found then return it, if not return null.* **public** Contact find(**int** id) {  
 Contact contact = **null**;  
 SQLiteDatabase db = **this**.**dbHelper**.getReadableDatabase();  
 Cursor cursor = db.rawQuery(**"SELECT \* FROM "** + Contact.ContactEntry.***TABLE\_NAME*** + **" WHERE "** + Contact.ContactEntry.***\_ID*** + **" = "** + id, **null**);  
 cursor.moveToFirst();  
  
 **while** (!cursor.isAfterLast()) {  
 contact = makeContact(cursor);  
 cursor.moveToNext();  
 }  
  
 cursor.close();  
 **return** contact;  
 }  
  
 *//Find the contacts that have the text parameter in either name, phone or email fields.  
 //Using SQL Like operator to match the text with the contacts fields.  
 //Return the list of contacts.* **public** List<Contact> findByText(String text) {  
 List<Contact> contactList = **new** ArrayList<>();  
 SQLiteDatabase db = **this**.**dbHelper**.getReadableDatabase();  
  
 Cursor cursor = db.rawQuery(**"SELECT \* FROM "** + Contact.ContactEntry.***TABLE\_NAME*** + **" WHERE NAME LIKE '%"** + text + **"%' OR PHONE LIKE '%"** + text + **"%' OR EMAIL LIKE '%"** + text + **"%'"**, **null**);  
 cursor.moveToFirst();  
  
 **while** (!cursor.isAfterLast()) {  
 contactList.add(makeContact(cursor));  
 cursor.moveToNext();  
 }  
  
 cursor.close();  
 **return** contactList;  
 }  
  
 *//Create a contact object based on the cursor information from the database.  
 //Uses the ContactEntry class to get the column names.  
 //Return the new contact object.* **private** Contact makeContact(Cursor cursor) {  
 Contact contact = **new** Contact();  
 contact.setId(cursor.getInt(cursor.getColumnIndex(Contact.ContactEntry.***\_ID***)));  
 contact.setName(cursor.getString(cursor.getColumnIndex(Contact.ContactEntry.***COLUMN\_NAME***)));  
 contact.setPhone(cursor.getString(cursor.getColumnIndex(Contact.ContactEntry.***COLUMN\_PHONE***)));  
 contact.setEmail(cursor.getString(cursor.getColumnIndex(Contact.ContactEntry.***COLUMN\_EMAIL***)));  
 **return** contact;  
 }  
  
 *//Delete the contact from the database that match with the contact id given.  
 //If delete successfully then insert this contact in the DeletedContact Table and in the Log Table.  
 //return the number of rows deleted returned by the DBHelper delete().* **public long** delete(Contact contact) {  
 SQLiteDatabase db = **this**.**dbHelper**.getWritableDatabase();  
  
 **int** wasDeleted = db.delete(Contact.ContactEntry.***TABLE\_NAME***, Contact.ContactEntry.***\_ID*** + **" = ? "**, **new** String[]{Integer.*toString*(contact.getId())});  
  
 **if** (wasDeleted >= 0) {  
 insertDeletedContact(contact);  
 LogController.*getInstance*(**this**.**context**).insert(contact.getId(), LogController.LogStatus.***DELETED***);  
 }  
  
 **return** wasDeleted;  
 }  
  
 *//Insert the contact into the DeletedContact table after the contact is deleted in the Contact Table.  
 //Keeping track of the deleted contacts.  
 //Return > 0 if it was inserted.* **private long** insertDeletedContact(Contact contact) {  
 SQLiteDatabase db = **this**.**dbHelper**.getWritableDatabase();  
  
 ContentValues values = **new** ContentValues();  
 values.put(Contact.ContactEntry.***COLUMN\_NAME***, contact.getName());  
 values.put(Contact.ContactEntry.***COLUMN\_PHONE***, contact.getPhone());  
 values.put(Contact.ContactEntry.***COLUMN\_EMAIL***, contact.getEmail());  
  
 **return** db.insert(DeletedContact.DeletedContactEntry.***TABLE\_NAME***, **null**, values);  
 }  
  
 *//Validate all the contact fields and throw exception if an error was caught.  
 //For each error the exception message changes letting the interface (Activity) knows what to show to the user.  
 //The last check is if the contact already exists in the database.* **private void** validate(Contact contact) **throws** NameException, EmailException, PhoneException {  
 **if** (contact.getName().length() > Contact.ContactEntry.***MAX\_NAME\_LENGTH***) {  
 **throw new** NameException(**"Name max lenght reached"**);  
 } **else if** (contact.getName().isEmpty()) {  
 **throw new** NameException(**"Name Empty"**);  
 }  
  
 **if** (contact.getEmail().length() > Contact.ContactEntry.***MAX\_EMAIL\_LENGTH***) {  
 **throw new** EmailException(**"Email max lenght reached"**);  
 } **else if** (contact.getEmail().isEmpty()) {  
 **throw new** EmailException(**"Email Empty"**);  
 }  
  
 **if** (contact.getPhone().length() > Contact.ContactEntry.***MAX\_PHONE\_LENGTH***) {  
 **throw new** PhoneException(**"Phone max lenght reached"**);  
 } **else if** (contact.getPhone().isEmpty()) {  
 **throw new** PhoneException(**"Phone Empty"**);  
 }  
  
 **if** (alreadyExists(contact))  
 **throw new** RuntimeException(**"Contact already exsits."**);  
  
 }  
  
 *//Check whether the contact was already added to the database or not.  
 //If the name and phone is the same of the contact received as parameter then it already exists.  
 //To check if the Android Cursor object has the method getCount() showing how many objects were returned from the query.  
 //If get count > 0 then the database found and object with the same name and phone.* **private boolean** alreadyExists(Contact contact) {  
  
 SQLiteDatabase db = **this**.**dbHelper**.getReadableDatabase();  
  
 Cursor cursor = db.rawQuery(**"SELECT \* FROM "** + Contact.ContactEntry.***TABLE\_NAME*** + **" WHERE NAME = '"** + contact.getName() + **"' AND PHONE = '"** + contact.getPhone() + **"'"**, **null**);  
  
 **if** (cursor.getCount() > 0) {  
 **return true**;  
 }  
  
 **return false**;  
 }  
}

Manifest Code

Manifest declared both activity and gives ContactListActivity the Search intent filter needed because the SearchView Widget. Widget also requires the meta-data. Sets ContactListActivity as singleTop allowing just one instance of that activity at time.

*<?***xml version="1.0" encoding="utf-8"***?>*<**manifest xmlns:android="http://schemas.android.com/apk/res/android"  
 package="com.app.contact.contactapp"**>  
  
 <**application  
 android:allowBackup="true"  
 android:icon="@mipmap/ic\_launcher"  
 android:label="@string/app\_name"  
 android:roundIcon="@mipmap/ic\_launcher\_round"  
 android:supportsRtl="true"  
 android:theme="@style/AppTheme"**>  
 <**activity  
 android:name=".ui.ContactListActivity"  
 android:launchMode="singleTop"**>  
 <**intent-filter**>  
 <**action android:name="android.intent.action.MAIN"** />  
 <**action android:name="android.intent.action.SEARCH"** />  
  
 <**category android:name="android.intent.category.LAUNCHER"** />  
 </**intent-filter**>  
 <**intent-filter**>  
 <**action android:name="android.intent.action.SEARCH"** />  
 </**intent-filter**>  
  
 <**meta-data  
 android:name="android.app.searchable"  
 android:resource="@xml/searchable"** />  
  
 </**activity**>  
 <**activity android:name=".ui.UpdateContactActivity"** />  
 </**application**>  
  
</**manifest**>