

# COMP 1786

## Logbook Upload Template

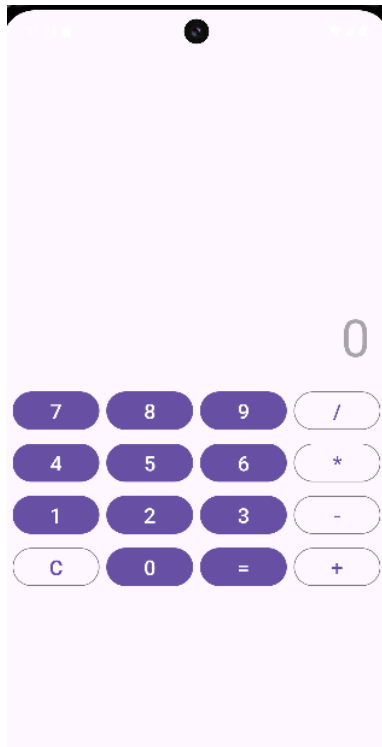
### 1. Basic Information

1.1 Student name	Vu Thanh Nguyen
1.2 Who did you work with? Note that for logbook exercises you are allowed to work with one other person as long as you give their name and login id and both contribute to the work.	<b>Name:</b>  <b>Login id:</b>
1.3 Which Exercise is this? Tick as appropriate.	<ul style="list-style-type: none"> <li>Exercise 1 <input checked="" type="checkbox"/></li> <li>Exercise 2 <input checked="" type="checkbox"/></li> <li>Exercise 3 <input checked="" type="checkbox"/></li> </ul>
1.4 How well did you complete the exercise? Tick as appropriate.	<ul style="list-style-type: none"> <li>I tried but couldn't complete it <input type="checkbox"/></li> <li>I did it but I feel I should have done better <input type="checkbox"/></li> <li>I did everything that was asked <input checked="" type="checkbox"/></li> <li>I did more than was asked for <input type="checkbox"/></li> </ul>
1.5 Briefly explain your answer to question 1.4.  Without any explanation/justification, your scores will be deducted.	

## 2. Exercise answer

### 2.1 Screen shots demonstrating what you achieved.

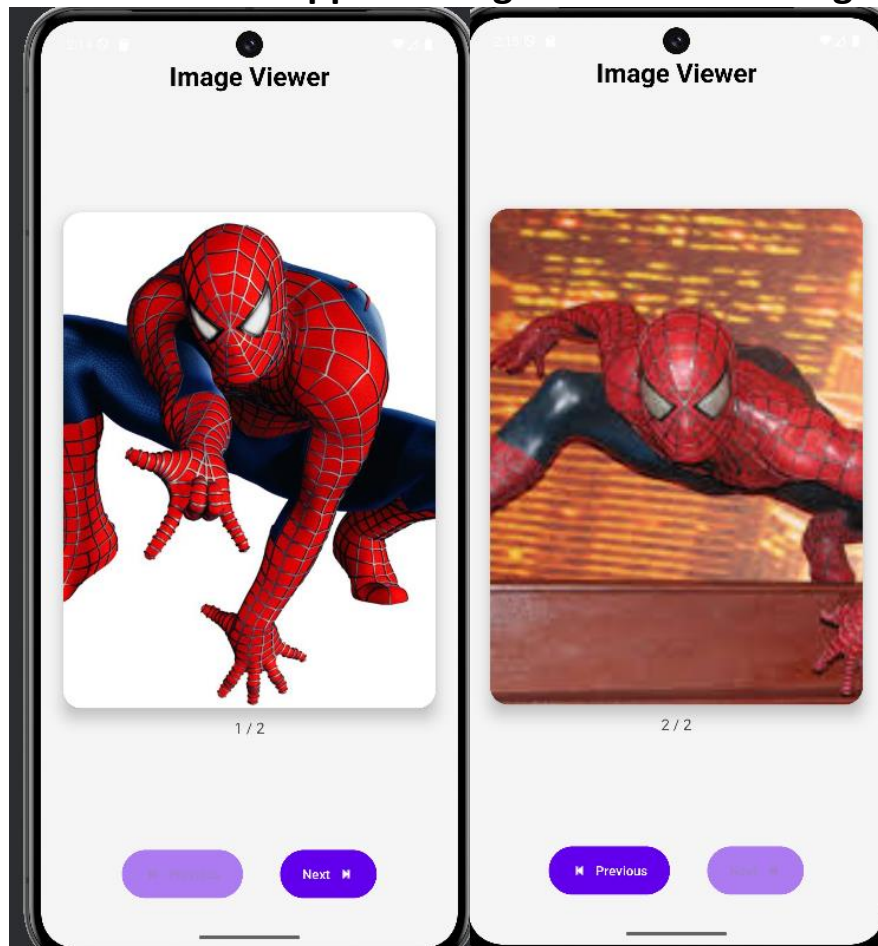
#### Exercise 1: Develop a simple calculator application



**Figure 1: Screen for exercise 1**

- Number Buttons (0-9): They are all assigned the same numberClickListener. When one is pressed, it gets the button's text (the digit) and appends it to the display TextView.
- Operator Buttons (+, -, \*, /): They are all assigned the same operatorClickListener. When one is pressed, it stores the first operand and the selected operator, then clears the display so you can enter the second operand.
- Equals Button (=): This has a separate click listener. It gets the second operand from the display, performs the calculation based on the stored operator, and displays the final result.
- Clear Button (C): This has its own click listener that clears all data—both operands, the operator, and the text on the display—to start a new calculation.

## Exercise 2: Create an Android App allowing users to view images

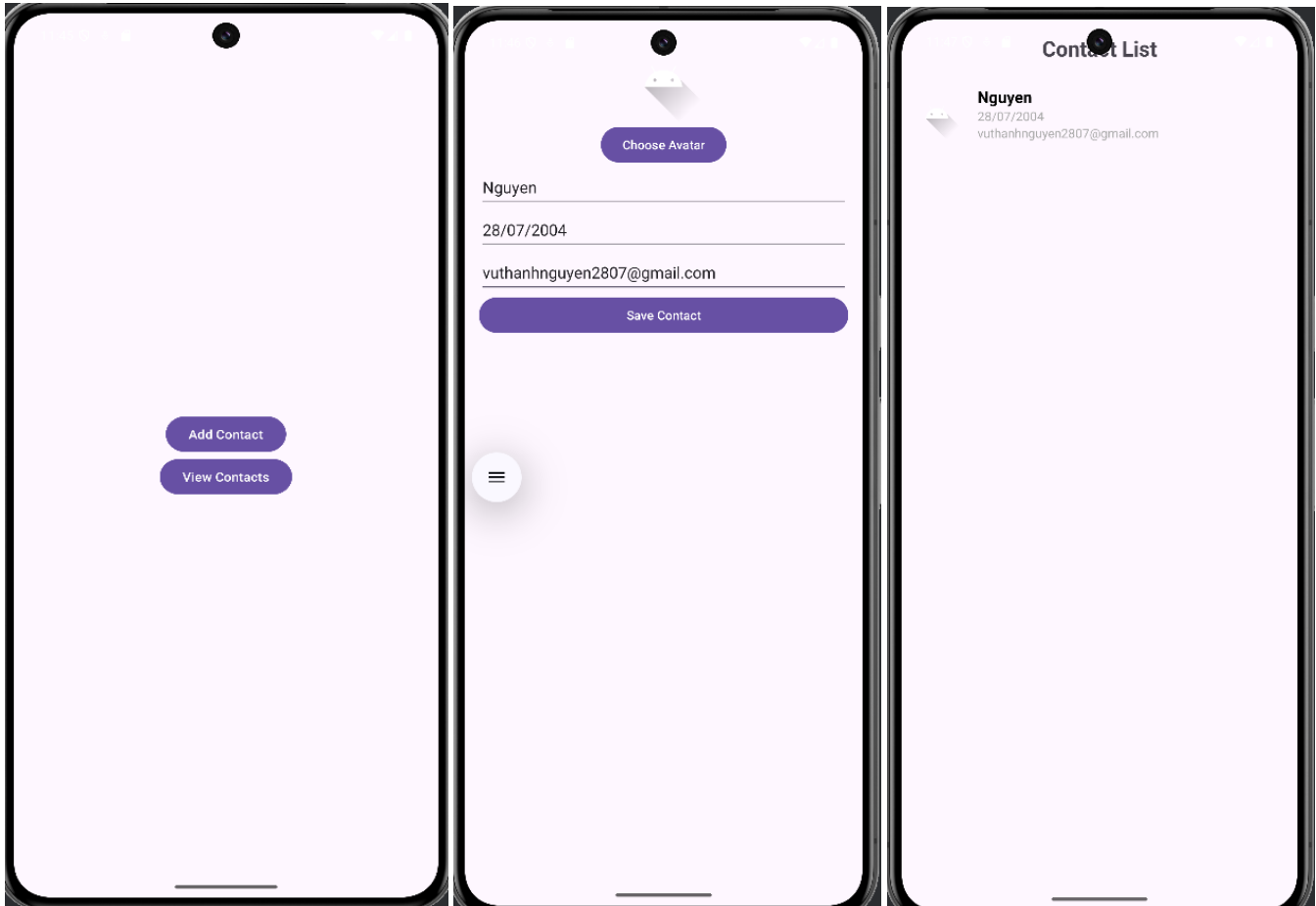


**Figure 2: 2 screen for exercise 2**

This application is a Simple Image Viewer (Image Slider). Its primary function is to allow users to navigate through a pre-defined collection of images (an Album) stored within the app.

- Image Display: Shows images one by one using an ImageView.
- Navigation: Users can move forward and backward through the list using "Next" and "Prev" buttons.
- Page Counter: Displays the current position (e.g., "1 / 2") so the user knows where they are in the list.
- Smart UX (User Experience): Buttons are automatically disabled and dimmed (faded out) when the user reaches the beginning or the end of the list to prevent invalid clicks.

## Exercise 3: Use Android Persistence to store data



**Figure 3: 3 screen for exercise 3 (Menu screen, Add contact screen, List contact screen)**

1. Menu Screen (Main Screen) This is the first screen users see when they open the application. It features a minimalist design and acts as a starting point, allowing users to quickly choose one of the two core functions:

- Add a new contact: Navigates the user to the information entry page.
- View contacts: Opens the list of all saved contacts.

2. Add Contact Page This is where a user creates a new contact entry. The page provides a clear form to input the necessary information for a contact, including:

- Name
- Date of Birth
- Email

Additionally, users can select an avatar to help visually identify the contact more easily. After filling out the information, the user simply presses the "Save" button to finalize and add the contact to the directory.

3. Contact List Page This screen intuitively displays the user's entire contact directory. Each contact is presented in a scrollable list, showing key summary information such as:

- Avatar
- Contact Name
- Email

This design makes it easy for users to efficiently browse, find, and manage their contacts.

## 2.2 Code that you wrote

### Exercise 1: Develop a simple calculator application

#### 1. MainActivity.java

•**Purpose:** This acts as the "brain" of the application. It listens for user interactions (like tapping a button) and performs the actual calculations.

•**In this app:** It detects when a button is pressed, stores the numbers, performs the math (+, -, \*, /), and updates the XML display with the result.

```
package com.example.computer;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

    private TextView resultTextView;
    private String operand1 = "";
    private String operand2 = "";
    private String operator = "";

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

        resultTextView = findViewById(R.id.resultTextView);

        View.OnClickListener numberClickListener = v -> {
            Button button = (Button) v;
            resultTextView.append(button.getText().toString());
        };

        findViewById(R.id.button0).setOnClickListener(numberClickListener);
        findViewById(R.id.button1).setOnClickListener(numberClickListener);
        findViewById(R.id.button2).setOnClickListener(numberClickListener);
        findViewById(R.id.button3).setOnClickListener(numberClickListener);
        findViewById(R.id.button4).setOnClickListener(numberClickListener);
        findViewById(R.id.button5).setOnClickListener(numberClickListener);
        findViewById(R.id.button6).setOnClickListener(numberClickListener);
        findViewById(R.id.button7).setOnClickListener(numberClickListener);
        findViewById(R.id.button8).setOnClickListener(numberClickListener);
        findViewById(R.id.button9).setOnClickListener(numberClickListener);

        View.OnClickListener operatorClickListener = v -> {
            Button button = (Button) v;
            if (!resultTextView.getText().toString().isEmpty()) {
                operand1 = resultTextView.getText().toString();
                operator = button.getText().toString();
                resultTextView.setText("");
            }
        };

        findViewById(R.id.buttonAdd).setOnClickListener(operatorClickListener);
        findViewById(R.id.buttonSubtract).setOnClickListener(operatorClickListener);
```

```

        findViewById(R.id.buttonMultiply).setOnClickListener(operatorClickListener);
        findViewById(R.id.buttonDivide).setOnClickListener(operatorClickListener);

        findViewById(R.id.buttonEquals).setOnClickListener(v -> {
            if (!operand1.isEmpty() && !operator.isEmpty() &&
!resultTextView.getText().toString().isEmpty()) {
                operand2 = resultTextView.getText().toString();
                double result = 0;
                try {
                    double num1 = Double.parseDouble(operand1);
                    double num2 = Double.parseDouble(operand2);

                    switch (operator) {
                        case "+":
                            result = num1 + num2;
                            break;
                        case "-":
                            result = num1 - num2;
                            break;
                        case "*":
                            result = num1 * num2;
                            break;
                        case "/":
                            if (num2 != 0) {
                                result = num1 / num2;
                            } else {
                                resultTextView.setText("Error");
                                return;
                            }
                            break;
                    }
                    resultTextView.setText(String.valueOf(result));
                } catch (NumberFormatException e) {
                    resultTextView.setText("Error");
                }
                operand1 = "";
                operand2 = "";
                operator = "";
            }
        });

        findViewById(R.id.buttonClear).setOnClickListener(v -> {
            resultTextView.setText("");
            operand1 = "";
            operand2 = "";
            operator = "";
        });
    }
}

```

On the Java Logic side, the functionality relies on three synchronized stages. First is the Input Handling stage, where the application must distinguish between simple button presses and multi-digit entry. When a user taps a number, the code uses string concatenation (e.g., *currentString* + "5") rather than addition, allowing users to form numbers like "15" or "100". The second stage is State Management during operator selection. When a user taps + or -, the app performs a "context switch": it parses the current string into a numeric variable (saving it as *firstOperand*), stores the selected operator symbol, and clears the display buffer to await the second number. The final stage is the Execution Logic triggered by the = button. This uses a *switch-case* structure to efficiently route the logic based on the stored operator. Crucially, robust implementations here often include a safety check for division by zero (e.g., *if (second != 0)*) to prevent the application from crashing before converting the calculated result back into a String to update the `resultTextView`.

## 2. activity\_main.xml

•**Purpose:** This acts as the "blueprint" for your screen. It defines the layout, colors, text sizes, and positions of every element (Buttons, TextViews).

•**In this app:** It arranges a display screen at the top and a grid of buttons (numbers and operators) at the bottom.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center"
    android:padding="8dp"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/resultTextView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="end|bottom"
        android:padding="16dp"
        android:text=""
        android:textSize="60sp"
        android:hint="0"/>

    <GridLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:columnCount="4">

        <com.google.android.material.button.MaterialButton
            android:id="@+id/button7"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_columnWeight="1"
            android:layout_margin="4dp"
            android:text="7"
            android:textSize="24sp" />

        <com.google.android.material.button.MaterialButton
            android:id="@+id/button8"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_columnWeight="1"
            android:layout_margin="4dp"
            android:text="8"
            android:textSize="24sp" />

        <com.google.android.material.button.MaterialButton
            android:id="@+id/button9"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_columnWeight="1"
            android:layout_margin="4dp"
            android:text="9"
            android:textSize="24sp" />

        <com.google.android.material.button.MaterialButton
            android:id="@+id/buttonDivide"
            style="?attr/materialButtonOutlinedStyle"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_columnWeight="1"
```

```
        android:layout_margin="4dp"
        android:text="/"
        android:textSize="24sp" />

<com.google.android.material.button.MaterialButton
    android:id="@+id/button4"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_columnWeight="1"
    android:layout_margin="4dp"
    android:text="4"
    android:textSize="24sp" />

<com.google.android.material.button.MaterialButton
    android:id="@+id/button5"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_columnWeight="1"
    android:layout_margin="4dp"
    android:text="5"
    android:textSize="24sp" />

<com.google.android.material.button.MaterialButton
    android:id="@+id/button6"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_columnWeight="1"
    android:layout_margin="4dp"
    android:text="6"
    android:textSize="24sp" />

<com.google.android.material.button.MaterialButton
    android:id="@+id/buttonMultiply"
    style="?attr/materialButtonOutlinedStyle"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_columnWeight="1"
    android:layout_margin="4dp"
    android:text="*"
    android:textSize="24sp" />

<com.google.android.material.button.MaterialButton
    android:id="@+id/button1"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_columnWeight="1"
    android:layout_margin="4dp"
    android:text="1"
    android:textSize="24sp" />

<com.google.android.material.button.MaterialButton
    android:id="@+id/button2"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_columnWeight="1"
    android:layout_margin="4dp"
    android:text="2"
    android:textSize="24sp" />

<com.google.android.material.button.MaterialButton
    android:id="@+id/button3"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_columnWeight="1"
    android:layout_margin="4dp"
    android:text="3"
    android:textSize="24sp" />
```

```

        <com.google.android.material.button.MaterialButton
            android:id="@+id/buttonSubtract"
            style="?attr/materialButtonOutlinedStyle"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_columnWeight="1"
            android:layout_margin="4dp"
            android:text="-"
            android:textSize="24sp" />

        <com.google.android.material.button.MaterialButton
            android:id="@+id/buttonClear"
            style="?attr/materialButtonOutlinedStyle"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_columnWeight="1"
            android:layout_margin="4dp"
            android:text="C"
            android:textSize="24sp" />

        <com.google.android.material.button.MaterialButton
            android:id="@+id/button0"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_columnWeight="1"
            android:layout_margin="4dp"
            android:text="0"
            android:textSize="24sp" />

        <com.google.android.material.button.MaterialButton
            android:id="@+id/buttonEquals"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_columnWeight="1"
            android:layout_margin="4dp"
            android:text="="
            android:textSize="24sp" />

        <com.google.android.material.button.MaterialButton
            android:id="@+id/buttonAdd"
            style="?attr/materialButtonOutlinedStyle"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_columnWeight="1"
            android:layout_margin="4dp"
            android:text="+"
            android:textSize="24sp" />

    </GridLayout>
</LinearLayout>

```

Regarding the XML Interface, the most critical component is the *GridLayout* container. By setting *android:columnCount="4"*, you create a rigid matrix that forces buttons to wrap automatically, eliminating the need for complex row-nesting. Inside this grid, the use of *android:layout\_columnWeight="1"* on the *MaterialButton* elements is a sophisticated layout technique; it instructs the Android rendering engine to ignore fixed widths and instead distribute the available horizontal space equally among buttons in the same row (25% each). This ensures the keypad scales perfectly from small phones to large tablets. Visually, the distinction between number keys and action keys is handled by applying *style="?attr/materialButtonOutlinedStyle"* to operators, giving them a unique look that improves user navigation.

## Exercise 2: Create an Android App allowing users to view images

## 1.MainActivity.java

This is the logic file (the brain) for the main screen. It handles user events (like button clicks), manages data (the list of images), and updates the user interface (displaying the image and the counter).

```
package com.example.ex2;

public class MainActivity extends AppCompatActivity {

    private ImageView imageView;
    private MaterialButton btnPrev, btnNext;
    private TextView tvCounter;
    private List<Integer> imageList;

    private int currentIndex = 0;

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

        initData();

        imageView = findViewById(R.id.imageViewDisplay);
        btnPrev = findViewById(R.id.btnPrev);
        btnNext = findViewById(R.id.btnNext);
        tvCounter = findViewById(R.id.tvCounter);

        updateImage();

        btnNext.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (currentIndex < imageList.size() - 1) {
                    currentIndex++;
                    updateImage();
                } else {
                    Toast.makeText(MainActivity.this, "Last image",
Toast.LENGTH_SHORT).show();
                }
            }
        });

        btnPrev.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (currentIndex > 0) {
                    currentIndex--;
                    updateImage();
                } else {
                    Toast.makeText(MainActivity.this, "First image",
Toast.LENGTH_SHORT).show();
                }
            }
        });
    }

    private void initData() {
        imageList = new ArrayList<>();
        imageList.add(R.drawable.aaa);
        imageList.add(R.drawable.images);
    }

    private void updateImage() {
        imageView.setImageResource(imageList.get(currentIndex));
    }
}
```

```

        String counterText = (currentIndex + 1) + " / " + imageUrl.size();
        tvCounter.setText(counterText);

        updateButtonState();
    }

    private void updateButtonState() {
        if (currentIndex > 0) {
            btnPrev.setEnabled(true);
            btnPrev.setAlpha(1.0f);
        } else {
            btnPrev.setEnabled(false);
            btnPrev.setAlpha(0.5f);
        }

        if (currentIndex < imageUrl.size() - 1) {
            btnNext.setEnabled(true);
            btnNext.setAlpha(1.0f);
        } else {
            btnNext.setEnabled(false);
            btnNext.setAlpha(0.5f);
        }
    }
}

```

The *MainActivity.java* class orchestrates the logic for the image viewer. In the *onCreate* method, it first links the Java code to the XML layout file using *setContentView(R.layout.activity\_main)*. It then initializes the UI components by finding each view by its ID (using *findViewById*) and storing references to them in private variables like *imageView*, *btnPrev*, etc. A list of image resources is populated in the *initData* method, which simply adds drawable resource IDs to an *ArrayList*. The core functionality resides in the *OnClickListeners* set for the "Next" and "Previous" buttons. When the "Next" button is clicked, the code checks if it's not the last image, increments a *currentIndex* counter, and calls *updateImage()* to refresh the screen. The "Previous" button works similarly but decrements the index. The *updateImage()* helper method is central to the logic; it sets the new image resource on the *ImageView*, updates the counter *TextView*, and calls *updateButtonState()*. The *updateButtonState()* method enhances the user experience by enabling or disabling the navigation buttons based on the current position. For instance, the "Previous" button is disabled when the first image is shown, and the "Next" button is disabled at the last image, preventing the user from navigating out of bounds.

## 2.activity\_main.xml

This is the layout file (the appearance) for the main screen. It defines the structure and look of the components on the screen, such as the image view, text, and buttons.

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical"
    android:padding="24dp">

    <TextView
        android:id="@+id/tvTitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="16dp"
        android:text="Simple Image Viewer"
        android:textSize="24sp"
        android:textStyle="bold" />

```

```

<androidx.cardview.widget.CardView
    android:layout_width="match_parent"
    android:layout_height="300dp"
    app:cardCornerRadius="16dp"
    app:cardElevation="8dp">

    <ImageView
        android:id="@+id/imageViewDisplay"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scaleType="centerCrop"
        android:src="@mipmap/ic_launcher" />

</androidx.cardview.widget.CardView>

<TextView
    android:id="@+id/tvCounter"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="16dp"
    android:text="1 / 5"
    android:textSize="18sp" />

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="24dp"
    android:gravity="center"
    android:orientation="horizontal">

    <com.google.android.material.button.MaterialButton
        android:id="@+id/btnPrev"
        android:layout_width="120dp"
        android:layout_height="60dp"
        android:layout_marginEnd="16dp"
        android:text="Previous" />

    <com.google.android.material.button.MaterialButton
        android:id="@+id/btnNext"
        android:layout_width="120dp"
        android:layout_height="60dp"
        android:layout_marginStart="16dp"
        android:text="Next" />

</LinearLayout>
</LinearLayout>

```

The *activity\_main.xml* file defines the user interface for the main screen using a vertical *LinearLayout* as the root element. This layout arranges all its children vertically and centers them on the screen. At the top, a *TextView* displays the title "Simple Image Viewer." Below the title, a *CardView* creates a container with rounded corners and a shadow effect, giving the image inside a modern, elevated look. This *CardView* contains an *ImageView*, which is the component responsible for actually displaying the pictures. Below the image, another *TextView* acts as a counter, showing the current image's position in the list (e.g., "1 / 5"). Finally, a horizontal *LinearLayout* at the bottom holds two *MaterialButton* widgets side-by-side: a "Previous" button and a "Next" button. These buttons allow the user to navigate through the image gallery. Each important view has a unique id (e.g., *@+id/imageViewDisplay*, *@+id/btnNext*) so they can be referenced and controlled from the *MainActivity.java* code.

## Exercise 3: Use Android Persistence to store data

### 1. MainActivity.java

```
public class MainActivity extends AppCompatActivity {

    Button btnAdd, btnView;

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

        btnAdd = findViewById(R.id.btnAdd);
        btnView = findViewById(R.id.btnView);

        btnAdd.setOnClickListener(v ->
            startActivity(new Intent(MainActivity.this,
AddContactActivity.class))
        );

        btnView.setOnClickListener(v ->
            startActivity(new Intent(MainActivity.this,
ContactListActivity.class))
        );
    }
}
```

This *MainActivity.java* file serves as the startup screen and main menu for the application. When the activity is created, the *onCreate* method is called to set up the user interface from *the activity\_main.xml* layout file. Within this method, it initializes two buttons: *btnAdd* and *btnView*. Event listeners are then set for each button. When a user clicks *btnAdd*, the application navigates to the *AddContactActivity* screen to add a new contact. Similarly, clicking *btnView* opens the *ContactListActivity* screen to display the list of saved contacts. Essentially, *MainActivity* acts as a simple dispatcher, directing users to the main features of the app.

### 2. DBConnect.java

```
public class DBConnect extends SQLiteOpenHelper {

    public static final String DB_NAME = "contacts.db";
    public static final int DB_VERSION = 1;

    public static final String TABLE = "contacts";
    public static final String COL_ID = "id";
    public static final String COL_NAME = "name";
    public static final String COL_DOB = "dob";
    public static final String COL_EMAIL = "email";
    public static final String COL_AVATAR = "avatar";

    public DBConnect(Context context) {
        super(context, DB_NAME, null, DB_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        String sql = "CREATE TABLE " + TABLE + " ("
            + COL_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, "
            + COL_NAME + " TEXT, "
            + COL_DOB + " TEXT, "
            + COL_EMAIL + " TEXT, "
            + COL_AVATAR + " INTEGER)";
    }
}
```

```

        db.execSQL(sql);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldV, int newV) {
        db.execSQL("DROP TABLE IF EXISTS " + TABLE);
        onCreate(db);
    }

    public void insertContact(Contact c) {
        SQLiteDatabase db = this.getWritableDatabase();
        ContentValues cv = new ContentValues();

        cv.put(COL_NAME, c.getName());
        cv.put(COL_DOB, c.getDob());
        cv.put(COL_EMAIL, c.getEmail());
        cv.put(COL_AVATAR, c.getAvatar());

        db.insert(TABLE, null, cv);
    }

    public ArrayList<Contact> getAllContacts() {
        ArrayList<Contact> list = new ArrayList<>();
        SQLiteDatabase db = this.getReadableDatabase();

        Cursor cursor = db.rawQuery("SELECT * FROM " + TABLE, null);

        if (cursor != null && cursor.moveToFirst()) {
            do {
                list.add(new Contact(
                    cursor.getInt(0),
                    cursor.getString(1),
                    cursor.getString(2),
                    cursor.getString(3),
                    cursor.getInt(4)
                ));
            } while (cursor.moveToNext());
        }
        return list;
    }
}

```

This *DBConnect.java* file is a crucial helper class that manages all database operations for the application using Android's built-in SQLite functionality. It defines the database schema, including the database name, table name, and all column names as constants for consistency. When the application runs for the first time, the *onCreate* method is executed to create the *contacts* table with columns for an ID, name, date of birth, email, and an avatar resource ID. The class provides two main public methods for data manipulation: *insertContact*, which takes a *Contact* object and safely adds its data as a new row in the database, and *getAllContacts*, which queries the database for all records, constructs a *Contact* object for each row, and returns them in an *ArrayList*. In essence, this class encapsulates the entire database logic, offering a clean and simple API for other parts of the app to create and retrieve contact data without directly handling SQL queries or database connections.

### 3. ContactListActivity.java

```

import java.util.ArrayList;

public class ContactListActivity extends AppCompatActivity {

    RecyclerView recyclerView;
    DBConnect db;

    @Override

```

```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_contact_list);

    db = new DBConnect(this);
    recyclerView = findViewById(R.id.recyclerContacts);

    ArrayList<Contact> list = db.getAllContacts();
    ContactAdapter adapter = new ContactAdapter(list, this);

    recyclerView.setLayoutManager(new LinearLayoutManager(this));
    recyclerView.setAdapter(adapter);
}
}

```

This *ContactListActivity.java* file is responsible for displaying the contact list to the user. The entire process takes place within the *onCreate* method, beginning by setting the user interface from the *activity\_contact\_list.xml* file and locating the *RecyclerView* component within it. Immediately after, it initializes a database connection via the *DBConnect* class and uses it to call the *getAllContacts()* method, which queries for and retrieves an *ArrayList* containing all saved contact objects. This data list is then provided to a *ContactAdapter*, a custom class tasked with converting each contact object into a displayable item. Finally, the *RecyclerView* is configured to arrange items vertically and is set to use the created adapter, resulting in the full list of contacts being rendered on the screen.

#### 4. ContactAdapter

```

public class ContactAdapter extends RecyclerView.Adapter<ContactAdapter.Holder> {

    ArrayList<Contact> list;
    Context context;

    public ContactAdapter(ArrayList<Contact> list, Context context) {
        this.list = list;
        this.context = context;
    }

    @Override
    public Holder onCreateViewHolder(ViewGroup parent, int viewType) {
        View v = LayoutInflater.from(context).inflate(R.layout.item_contact, parent,
false);
        return new Holder(v);
    }

    @Override
    public void onBindViewHolder(Holder h, int pos) {
        Contact c = list.get(pos);
        h.txtName.setText(c.getName());
        h.txtDob.setText(c.getDob());
        h.txtEmail.setText(c.getEmail());
        h.img.setImageResource(c.getAvatar());
    }

    @Override
    public int getItemCount() {
        return list.size();
    }

    class Holder extends RecyclerView.ViewHolder {
        TextView txtName, txtDob, txtEmail;
        ImageView img;

        public Holder(View v) {
            super(v);
            txtName = v.findViewById(R.id.txtName);

```

```

        txtDob = v.findViewById(R.id.txtDob);
        txtEmail = v.findViewById(R.id.txtEmail);
        img = v.findViewById(R.id.imgAvatar);
    }
}
}

```

This *ContactAdapter.java* file acts as the essential bridge between the contact data list (*ArrayList<Contact>*) and the *RecyclerView* that displays it. Once attached to the *RecyclerView*, the adapter first informs it of the total number of items to display via the *getItemCount()* method. The *RecyclerView* then calls the *onCreateViewHolder* method to create a few initial view "frames" by inflating the *item\_contact.xml* layout. These frames are managed by an inner Holder class, which finds and caches references to the child views like *TextView* and *ImageView* just one time for performance optimization. Finally, when an item needs to be shown on screen, the *RecyclerView* calls *onBindViewHolder*, which retrieves the appropriate *Contact* object from the list and uses the cached references in the Holder to quickly populate the view frame with its data (name, email, and avatar). This process, especially the recycling of Holders, allows the *RecyclerView* to display long lists smoothly and efficiently.

## 5. Contact.java

```

package com.example.ex3;

public class Contact {
    private int id;
    private String name;
    private String dob;
    private String email;
    private int avatar;

    public Contact(String name, String dob, String email, int avatar) {
        this.name = name;
        this.dob = dob;
        this.email = email;
        this.avatar = avatar;
    }

    public Contact(int id, String name, String dob, String email, int avatar) {
        this.id = id;
        this.name = name;
        this.dob = dob;
        this.email = email;
        this.avatar = avatar;
    }

    public int getId() { return id; }
    public String getName() { return name; }
    public String getDob() { return dob; }
    public String getEmail() { return email; }
    public int getAvatar() { return avatar; }
}

```

This *Contact.java* file is a simple data model class that acts as a "blueprint" or a "template" for a *Contact* object within the application. It encapsulates all the necessary information for a single contact—including its *id*, *name*, *dob*, *email*, and *avatar*—into a single, organized object. The class provides two constructors: one for creating a brand new contact that does not yet have a database ID, and another for reconstructing an existing contact object from data that has been read from the database (which includes the ID). By providing public "getter" methods, it allows other components of the application, like the *ContactAdapter* and *DBConnect*, to safely and consistently access these data properties, ensuring that contact data is handled in a standardized structure throughout the entire app.

## 6. AvatarPickerActivity.java

```
public class AvatarPickerActivity extends AppCompatActivity {

    ImageView a1, a2, a3;

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

        a1 = findViewById(R.id.avatar1);
        a2 = findViewById(R.id.avatar2);
        a3 = findViewById(R.id.avatar3);

        a1.setOnClickListener(v -> choose(R.drawable.ic_avatar1));
        a2.setOnClickListener(v -> choose(R.drawable.ic_avatar2));
        a3.setOnClickListener(v -> choose(R.drawable.ic_avatar3));
    }

    private void choose(int res) {
        Intent result = new Intent();
        result.putExtra("avatar", res);
        setResult(RESULT_OK, result);
        finish();
    }
}
```

This *AvatarPickerActivity.java* file defines a simple sub-activity whose sole function is to allow a user to select an avatar. When the activity is created, it displays an interface containing three clickable images. Each image is assigned an event listener, and when a user clicks on any image, it immediately calls the *choose* method. This method creates a new *Intent*—not for navigation, but to act as an "envelope" for data—and then puts the resource ID of the selected image into it. Finally, it sets the activity's result to success (*RESULT\_OK*) along with the data-carrying *Intent*, and calls *finish()* to immediately close the avatar picker screen, sending the chosen result back to the activity that called it.

## 7. AddContactActivity.java

```
public class AddContactActivity extends AppCompatActivity {

    EditText edtName, edtDob, edtEmail;
    ImageView imgAvatar;
    Button btnChoose, btnSave;
    int selectedAvatar = R.drawable.ic_launcher_foreground;

    DBConnect db;

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

        db = new DBConnect(this);

        edtName = findViewById(R.id.edtName);
        edtDob = findViewById(R.id.edtDob);
        edtEmail = findViewById(R.id.edtEmail);
        imgAvatar = findViewById(R.id.imgAvatar);
        btnChoose = findViewById(R.id.btnChooseAvatar);
        btnSave = findViewById(R.id.btnSave);

        btnChoose.setOnClickListener(v -> {
            Intent i = new Intent(AddContactActivity.this,
AvatarPickerActivity.class);
```

```

        startActivityForResult(i, 111);
    });

    btnSave.setOnClickListener(v -> {
        db.insertContact(new Contact(
            edtName.getText().toString(),
            edtDob.getText().toString(),
            edtEmail.getText().toString(),
            selectedAvatar
        ));
        finish();
    });
}

@Override
protected void onActivityResult(int req, int res, Intent data) {
    super.onActivityResult(req, res, data);

    if (req == 111 && res == RESULT_OK) {
        selectedAvatar = data.getIntExtra("avatar",
R.drawable.ic_launcher_foreground);
        imgAvatar.setImageResource(selectedAvatar);
    }
}
}

```

This *AddContactActivity.java* file defines the screen where users can add a new contact. In the *onCreate* method, it sets up the user interface from its corresponding XML layout, initializes a connection to the database via *DBConnect*, and links all the UI elements like *EditTexts*, *ImageView*, and *Buttons*. An event listener on the "Choose" button starts the *AvatarPickerActivity* to allow the user to select an avatar, expecting a result back. The "Save" button's listener gathers all the input text and the selected avatar's ID, creates a new *Contact* object with this data, and inserts it into the database before closing the screen. The *onActivityResult* method handles the result from the avatar picker; when the user successfully selects an avatar, this method receives the choice, updates the *ImageView* on the screen to display the new avatar, and stores its resource ID in the *selectedAvatar* variable to be saved with the contact.

## 8. item\_contact.xml

```

<?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="wrap_content"
    android:orientation="horizontal"
    android:padding="10dp"
    android:gravity="center_vertical"
    android:background="?android:attr/selectableItemBackground">

    <!-- Hình ảnh Avatar -->
    <ImageView
        android:id="@+id/imgAvatar"
        android:layout_width="60dp"
        android:layout_height="60dp"
        android:src="@drawable/ic_launcher_foreground"
        android:contentDescription="Avatar"
        android:scaleType="centerCrop" />

    <!-- Thông tin chi tiết -->
    <LinearLayout
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:orientation="vertical"
        android:paddingStart="16dp">

```

```

<TextView
    android:id="@+id/txtName"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="User Name"
    android:textSize="18sp"
    android:textStyle="bold"
    android:textColor="@android:color/black" />

<TextView
    android:id="@+id/txtDob"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="01/01/2000"
    android:textSize="14sp"
    android:textColor="@android:color/darker_gray" />

<TextView
    android:id="@+id/txtEmail"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="email@example.com"
    android:textSize="14sp"
    android:textColor="@android:color/darker_gray" />
</LinearLayout>
</LinearLayout>

```



**User Name**

01/01/2000

email@example.com



TextView  
TextView  
TextView

This XML code defines the layout for a single row within a RecyclerView, serving as a template for each contact item. The root element is a horizontal LinearLayout which arranges an ImageView on the left and a nested, vertical LinearLayout on the right. The ImageView is a fixed-size container for the contact's avatar, while the nested LinearLayout is configured with a layout\_weight of 1, allowing it to expand and fill all remaining horizontal space. Inside this expanding section, three TextViews are stacked vertically to display the contact's name, date of birth, and email, with styling applied to create a clear visual hierarchy. The entire row is given padding and a selectable item background to provide visual feedback upon user interaction, resulting in a clean and standard list item design.

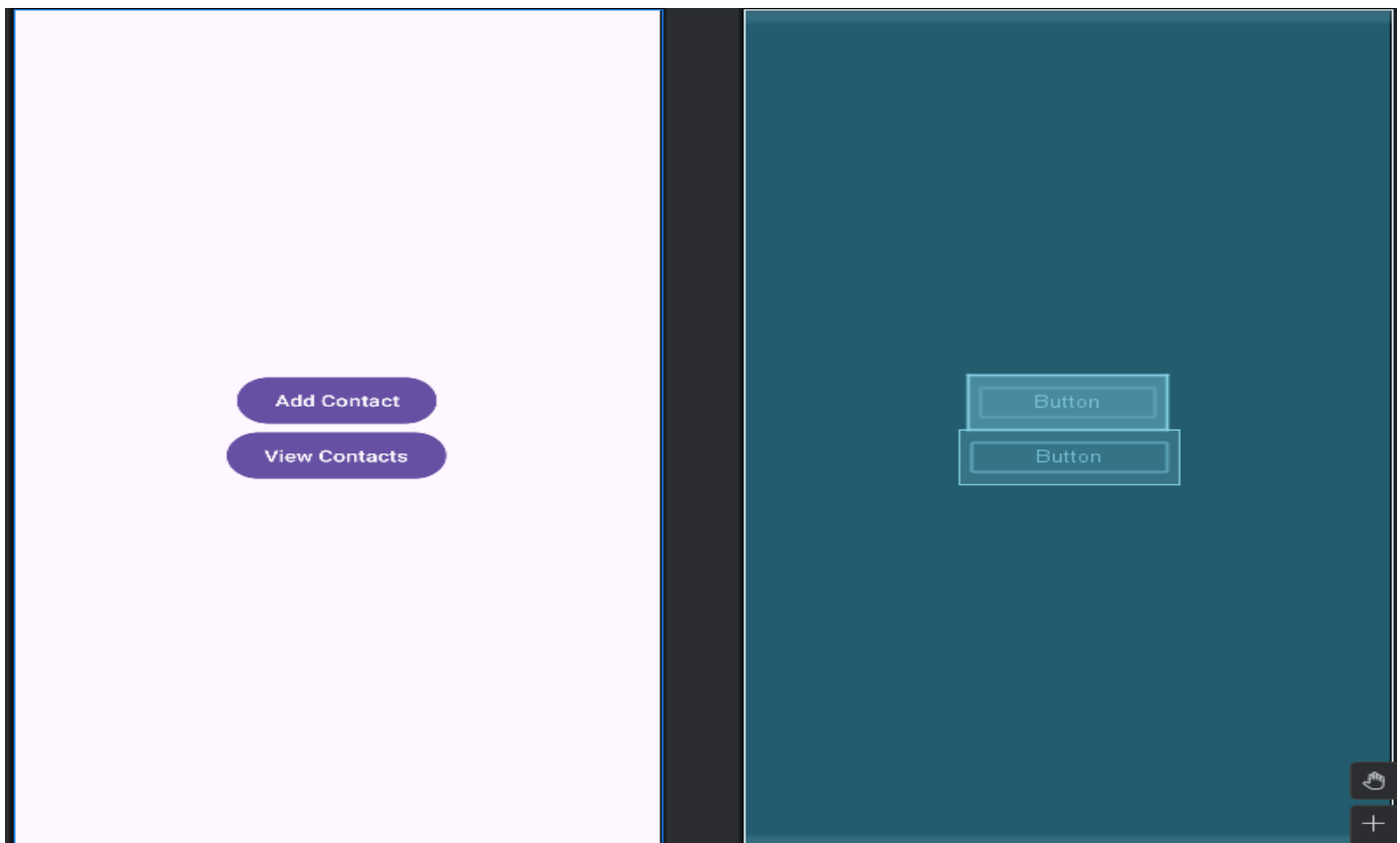
## 9. activity\_main.xml

```
<?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:orientation="vertical"
    android:gravity="center">

    <Button
        android:id="@+id/btnAdd"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Add Contact" />

    <Button
        android:id="@+id/btnView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="View Contacts" />

</LinearLayout>
```



This XML layout file defines the user interface for the application's main screen. It uses a `LinearLayout` as its root container, which is configured to fill the entire screen and to arrange its child elements in a vertical stack. By setting the `gravity` attribute to "center", it ensures that all the content within the layout is positioned in the middle of the screen, both horizontally and vertically. Inside this container, there are two `Button` widgets: the first one displays the text "Add Contact" and is identified by `btnAdd`, while the second one, identified by `btnView`, displays "View Contacts", creating a simple and centered navigation menu for the user.

## 10. activity\_contact\_list.xml

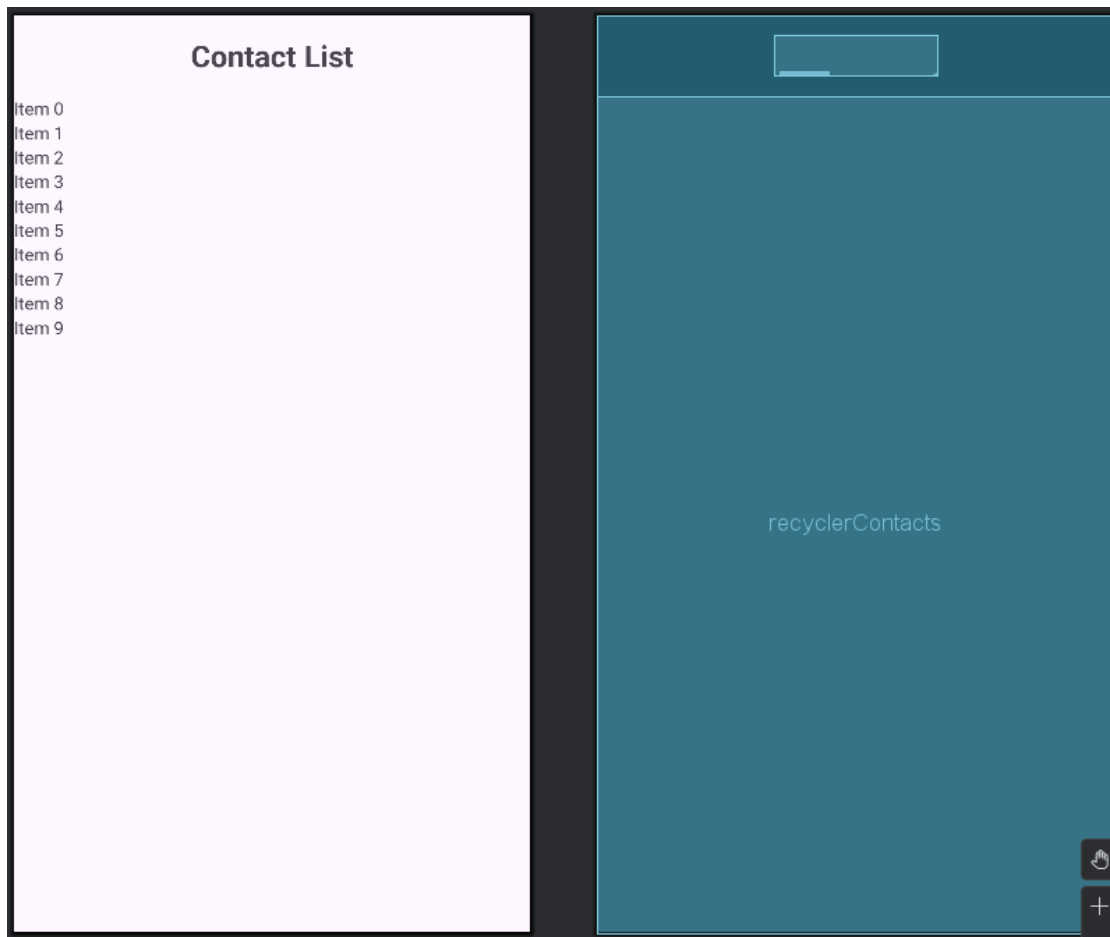
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".ContactListActivity">

    <TextView
        android:id="@+id/tvTitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Contact List"
        android:textSize="24sp"
        android:textStyle="bold"
        android:layout_marginTop="16dp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"/>

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/recyclerContacts"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:layout_marginTop="16dp"
        app:layout_constraintTop_toBottomOf="@id/tvTitle"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

This XML layout file defines the user interface for the contact list screen using a `ConstraintLayout`, which allows for flexible positioning of elements. It features a `TextView` at the top, identified as `tvTitle`, which is centered horizontally and displays the title "Contact List" in a bold, 24sp font. Below this title, a `RecyclerView` with the ID `recyclerContacts` is configured to fill all the remaining screen space. This is achieved by setting its width and height to `0dp` (which means "match constraints") and constraining its top edge to the bottom of the title, while its bottom, start, and end edges are constrained to the parent layout, effectively creating a scrollable list area that dynamically adapts to the screen size.



## 11. activity\_add\_contact.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:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp"
    tools:context=".AddContactActivity">

    <ImageView
        android:id="@+id/imgAvatar"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_gravity="center"
        android:contentDescription="User Avatar"
        android:src="@drawable/ic_launcher_foreground" />

    <Button
        android:id="@+id/btnChooseAvatar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:text="Choose Avatar" />

    <EditText
        android:id="@+id/edtName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:minHeight="48dp"
        android:inputType="textPersonName"
        android:hint="Name" />

```

```

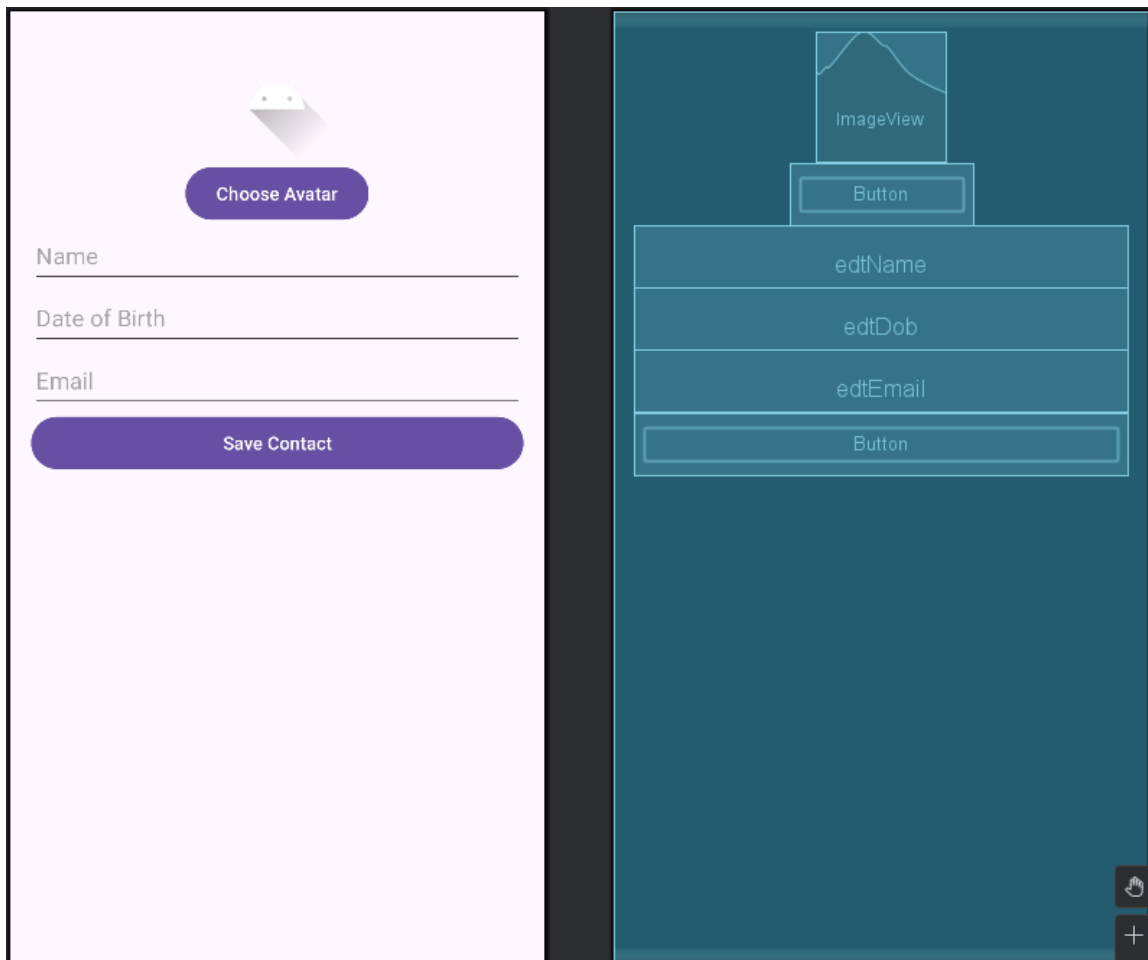
<EditText
    android:id="@+id/edtDob"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:minHeight="48dp"
    android:inputType="date"
    android:hint="Date of Birth" />

<EditText
    android:id="@+id/edtEmail"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:minHeight="48dp"
    android:inputType="textEmailAddress"
    android:hint="Email" />

<Button
    android:id="@+id/btnSave"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Save Contact" />
</LinearLayout>

```

This XML layout file defines the user interface for the contact list screen using a `ConstraintLayout`, which allows for flexible positioning of elements. It features a `TextView` at the top, identified as `tvTitle`, which is centered horizontally and displays the title "Contact List" in a bold, 24sp font. Below this title, a `RecyclerView` with the ID `recyclerContacts` is configured to fill all the remaining screen space. This is achieved by setting its width and height to 0dp (which means "match constraints") and constraining its top edge to the bottom of the title, while its bottom, start, and end edges are constrained to the parent layout, effectively creating a scrollable list area that dynamically adapts to the screen size.



I have used AI while undertaking my assignment in the following ways:

To develop research questions on the topic – YES

To create an outline of the topic – NO

To explain concepts – YES

To support my use of language – NO