



**UNIVERSITY OF  
GREENWICH**

Alliance with **FPT** Education

**UNIVERSITY OF  
GREENWICH**

**COMP1786 – Mobile  
Application Design and  
Development**

Student name	Chi-Trien Nguyen
ID number (00xxxxxxx)	001353546
Lecturer/Tutor name	Thai-Huy Le
Student submission date	November 16, 2023

# Table of Contents

1. Introduction .....	6
2. Exercise Answers .....	7
2.1. Exercise 1: Develop a simple calculator application .....	7
2.1.1. Screenshots demonstrating what is achieved .....	7
2.1.2. Code snippets .....	10
2.2. Exercise 2: Create an Android App allowing users to view images .....	15
2.2.1. Screenshots demonstrating what is achieved .....	15
2.2.2. Code snippets .....	18
2.3. Exercise 3: Use Android Persistence to store data .....	24
2.3.1. Screenshots demonstrating what is achieved .....	24
2.3.2. Code snippets .....	28
References .....	41

# Table of Figures

Figure 1: The main interface of the Calculator application .....	7
Figure 2: Interface of addition calculation and results of the Calculator application .....	8
Figure 3: Interface of multiplication calculation and results of the Calculator application .....	8
Figure 4: Interface of subtraction calculation and results of the Calculator application .....	9
Figure 5: Interface of division calculation and results of the Calculator application .....	9
Figure 6: Interface of the complex calculations and results of the Calculator application .....	10
Figure 7: The code snippet of MainActivity.java (code line 19-46) in Exercise 1 .....	10
Figure 8: The code snippet of MainActivity.java (code line 48-51) in Exercise 1 .....	11
Figure 9: The code snippet of MainActivity.java (code line 53-80) in Exercise 1 .....	12
Figure 10: The code snippet of MainActivity.java (code line 82-end) in Exercise 1 .....	13
Figure 11: The code snippet of activity_main.xml (code line 1-33) in Exercise 1 .....	14
Figure 12: The code snippet of activity_main.xml (code line 34-63) in Exercise 1 .....	14
Figure 13: The main interface of the Image Review application .....	15
Figure 14: The interface when the users click on “Import images from Gallery” .....	16
Figure 15: The interface of choosing images .....	16
Figure 16: The interface when importing photos successfully .....	17
Figure 17: The interfaces when users use the “Backward” and “Forward” buttons to browse the application .....	17
Figure 18: The code snippet of RecyclerViewAdapter.java (code line 20-46) in Exercise 2 .....	18
Figure 19: The code snippet of MainActivity.java (code line 32-49) in Exercise 2 .....	19
Figure 20: The code snippet of MainActivity.java (code line 50-64) in Exercise 2 .....	20
Figure 21: The code snippet of MainActivity.java (code line 89-106) in Exercise 2 .....	20
Figure 22: The code snippet of MainActivity.java (code line 122-144) in Exercise 2 .....	21
Figure 23: The code snippet of custom_single_img.xml in Exercise 2 .....	22
Figure 24: The code snippet of activity_main.xml (code line 1-22) in Exercise 2 .....	23
Figure 25: The code snippet of activity_main.xml (code line 23-51) in Exercise 2 .....	23
Figure 26: Main Interface without contact of Contact Database application Contact Database .....	24
Figure 27: Main Interface with contacts of Contact Database application Contact Database .....	25
Figure 28: Add Contact Interface (Blank) .....	25
Figure 29: Add Contact Interface (Full fill available fields) .....	26
Figure 30: Update Contact interface when we access from an available contact (example: Harry contact) .....	26
Figure 31: Main Interface (After updating the name of contact Harry to HarryBodison) .....	27
Figure 32: Details Interface (example: HarryBodison contact) .....	27
Figure 33: The code snippet of MainActivity.java (code line 38-68) in Exercise 3 .....	28
Figure 34: The code snippet of MainActivity.java (code line 69-89) in Exercise 3 .....	28
Figure 35: The code snippet of MainActivity.java (code line 90-105) in Exercise 3 .....	29
Figure 36: The code snippet of MainActivity.java (code line 106-131) in Exercise 3 .....	29
Figure 37: The code snippet of MainActivity.java (code line 132-154) in Exercise 3 .....	30
Figure 38: The code snippet of MainActivity.java (code line 155-end) in Exercise 3 .....	30
Figure 39: The code snippet of AddContactActivity.java (code line 29-42) in Exercise 3 .....	31
Figure 40: The code snippet of AddContactActivity.java (code line 43-61) in Exercise 3 .....	31
Figure 41: The code snippet of AddContactActivity.java (code line 62-89) in Exercise 3 .....	32
Figure 42: The code snippet of AddContactActivity.java (code line 90-109) in Exercise 3 .....	32
Figure 43: The code snippet of AddContactActivity.java (code line 110-136) in Exercise 3 .....	33

Figure 44: The code snippet of AddContactActivity.java (code line 137-end) in Exercise 3 .....	33
Figure 44: The code snippet of UpdateContactActivity.java (code line 73-77) in Exercise 3 .....	34
Figure 44: The code snippet of UpdateContactActivity.java (code line 82-101) in Exercise 3 .....	34
Figure 44: The code snippet of UpdateContactActivity.java (code line 138-165) in Exercise 3 .....	34
Figure 45: The code snippet of DetailstActivity.java (code line 23-58) in Exercise 3 .....	35
Figure 46: The code snippet of DetailstActivity.java (code line 59-76) in Exercise 3 .....	36
Figure 47: The code snippet of DatabaseHelper.java (code line 21-40) in Exercise 3 .....	36
Figure 48: The code snippet of DatabaseHelper.java (code line 41-76) in Exercise 3 .....	37
Figure 49: The code snippet of DatabaseHelper.java (code line 77-end) in Exercise 3 .....	37
Figure 50: The code snippet of activity_main.xml (Code line 1-35) in Exercise 3 .....	38
Figure 52: The code snippet of item.xml in Exercise 3 .....	38
Figure 54: The code snippet of activity_add_contact.xml (code line 1-28) in Exercise 3 .....	39
Figure 55: The code snippet of activity_update_contact.xml (code line 1-28) in Exercise 3 .....	39
Figure 56: The code snippet of activity_details.xml (code line 1-29) in Exercise 3 .....	40

## Table of Tables

Table 1: The self-assessment of the “Develop a simple calculator application” exercise .....	7
Table 2: The self-assessment of the “Create an Android App allowing users to view images” exercise .....	15
Table 3: The self-assessment of the “Use Android Persistence to store data” exercise .....	24

# 1. Introduction

We employ Android Studio and the Java programming language for app development. To facilitate the creation of innovative and reliable Android applications, Google introduced Android Jetpack in 2018. Android Jetpack, comprised of tools, frameworks, and architectural concepts, streamlines the development process. The latest release of Android Studio Development Essentials encompasses critical components of Android Jetpack, such as the Integrated Development Environment (IDE), Android Architecture Components, and Modern App Architecture Guidelines, providing developers with a comprehensive toolkit (Smyth, 2020). The "Develop a Simple Calculator Application" project is a great place to start for new Android app developers. The goal is to develop a simple calculator application that can carry out addition, subtraction, multiplication, and division and other fundamental arithmetic operations. For this task, computing methods must be developed with a user-friendly interface with buttons for each action. In this job, going above and beyond the minimal criteria reveals a deep understanding of Android programming and produces a more complete and user-friendly calculator application.

The advanced task titled "Develop an Android App Enabling Image Viewing" aims to deliver an application tailored for developers with a passion for pictures. The software's user interface is designed to present diverse photos, allowing users to load and navigate through them effortlessly. By incorporating advanced features into the image viewer program, we aim to elevate user experience, showcase our technological expertise, and transform it into a versatile and feature-rich tool for seamless picture exploration. The successful execution of this project reflects our commitment to delivering a sophisticated and user-friendly image-viewing solution.

The "Use Android Persistence to Store Data" exercise centres around leveraging Android apps for data management and storage. To save and retrieve data, developers are tasked with employing Android's inherent persistence mechanisms, such as SQLite databases or SharedPreferences. The primary objective is to create a user-friendly note-taking application that enables users to store and access their notes. This project emphasises efficiently addressing data storage requirements, establishing a robust foundation for data durability within the software, and adhering to specified guidelines to ensure consistent prioritisation of reliable data management.

The potent synergy between Java and Android Studio empowers developers to craft intricate, feature-laden Android applications. Our commitment to excellence is underscored by completing three pivotal Android app development tasks. Furthermore, our ingenuity and imaginative prowess shine through as we surpass the stipulated calculator and picture viewer program requirements. Addressing the requisites of the data persistence exercise, we establish a solid foundation for effective data management. These abilities are indispensable assets for any developer aiming to thrive in the dynamic landscape of Android app development.

## 2. Exercise Answers

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

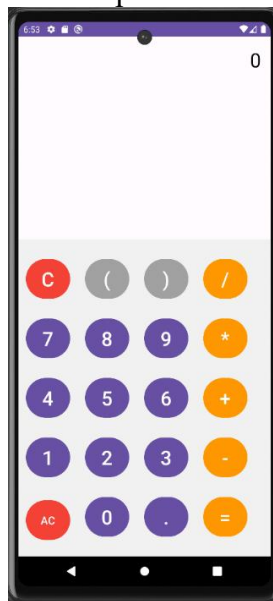
The table below shows the self-assessment of the “Develop a simple calculator application” exercise:

*Table 1: The self-assessment of the “Develop a simple calculator application” exercise*

1.1. Student name.	Phat Dat Truong – 001353261
1.2. Who did you work with?	Minh Nghia Nguyen – 001353656 Chi Trien Nguyen – 001353546
1.3. Which Exercise is this?	<input checked="" type="checkbox"/> Exercise 1 <input type="checkbox"/> Exercise 2 <input type="checkbox"/> Exercise 3
1.4. How well did you complete the exercise?	<input type="checkbox"/> I tried but couldn't complete it. <input type="checkbox"/> I did it, but I feel I should have done better. <input type="checkbox"/> I did everything that was asked. <input checked="" type="checkbox"/> I did more than was asked for.
1.5. Briefly explain your answer to question 1.4.	The assignment initially called for four operators-addition, subtraction, multiplication, and division-paired with two operands each. However, my system surpasses this requirement by accommodating more than two operands, with the number of operations per calculation now limited to the device's cache. Furthermore, I have enhanced the user interface to maximize user-friendliness, elevating the overall experience of using my calculator.

#### 2.1.1. Screenshots demonstrating what is achieved

I revamped the calculator's UI, emphasising the four basic operators (addition, subtraction, multiplication, and division) and introducing brackets for more complex calculations.



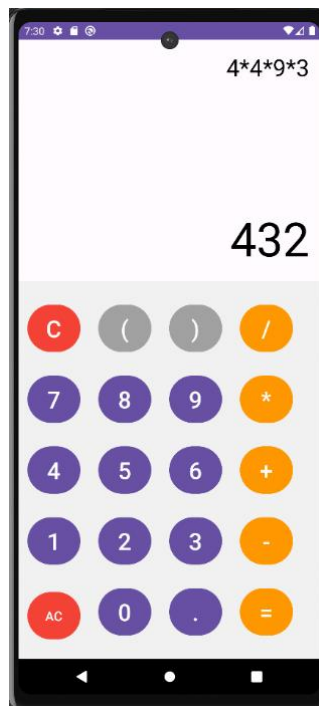
*Figure 1: The main interface of the Calculator application*

I updated my calculator's primary user interface to include the four basic operators (addition, subtraction, multiplication, and division) and introduced a parenthesis feature to accommodate more complex computations.



*Figure 2: Interface of addition calculation and results of the Calculator application*

Through this interface, the addition operation was executed accurately and successfully. Despite the initial requirement for the simultaneous use of only two integers, the current outcome allows for the computation of up to six numbers. This enhancement aims to facilitate more seamless calculations, especially for more significant numbers.



*Figure 3: Interface of multiplication calculation and results of the Calculator application*

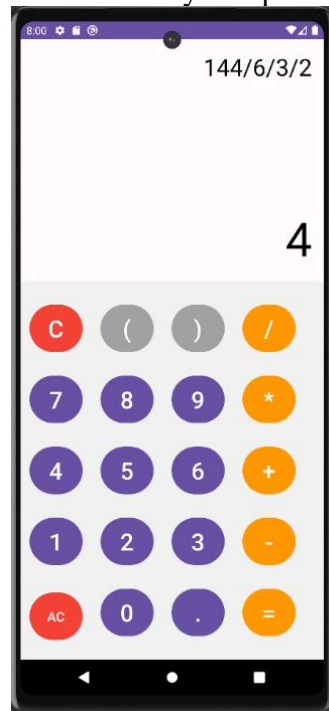
The user interface illustrates the successful completion of the multiplication computation.





*Figure 4: Interface of subtraction calculation and results of the Calculator application*

The user interface indicates that the algorithm accurately completed the subtraction calculation.



*Figure 5: Interface of division calculation and results of the Calculator application*

The system interface indicates the successful completion of the division computation.



*Figure 6: Interface of the complex calculations and results of the Calculator application*

The interface housing my system's calculations highlights its capability to handle highly complex computations.

### 2.1.2. Code snippets

For this exercise, the following are code source excerpts and explanations:

- File **MainActivity.java**: The code begins with several import lines, incorporating essential Android libraries and a third-party JavaScript library for evaluating mathematical equations.

```

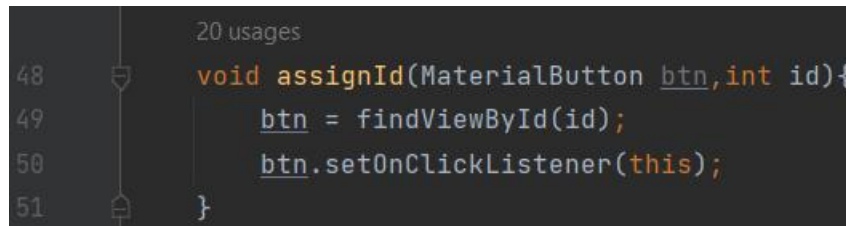
19      @Override
20      protected void onCreate(Bundle savedInstanceState) {
21          super.onCreate(savedInstanceState);
22          setContentView(R.layout.activity_main);
23          resultTv = findViewById(R.id.result_tv);
24          solutionTv = findViewById(R.id.solution_tv);
25
26          assignId(buttonC, R.id.button_c);
27          assignId(buttonBrackOpen, R.id.button_open_bracket);
28          assignId(buttonBrackClose, R.id.button_close_bracket);
29          assignId(buttonDivine, R.id.button_divine);
30          assignId(buttonMultiply, R.id.button_multiply);
31          assignId(buttonPlus, R.id.button_plus);
32          assignId(buttonMinus, R.id.button_minus);
33          assignId(buttonEquals, R.id.button_equals);
34          assignId(button0, R.id.button_0);
35          assignId(button1, R.id.button_1);
36          assignId(button2, R.id.button_2);
37          assignId(button3, R.id.button_3);
38          assignId(button4, R.id.button_4);
39          assignId(button5, R.id.button_5);
40          assignId(button6, R.id.button_6);
41          assignId(button7, R.id.button_7);
42          assignId(button8, R.id.button_8);
43          assignId(button9, R.id.button_9);
44          assignId(buttonAC, R.id.button_ac);
45          assignId(buttonDot, R.id.button_dot);
46      }

```

*Figure 7: The code snippet of MainActivity.java (code line 19-46) in Exercise 1*

The following sentence is the description of this section:

- + **super.onCreate(savedInstanceState);**: This line invokes the onCreate function of the parent class, a crucial step for the correct initialization of the activity. It often contains the necessary setup code for the Android framework to function correctly.
- + **setContentView(R.layout.activity\_main);**: This line directs the activity's content view to utilize the layout defined in the activity\_main.xml file. This XML layout outlines the arrangement of UI elements, such as buttons and text views, within the user interface.
- + **resultTv = findViewById(R.id.result\_tv);** and **solutionTv = findViewById(R.id.solution\_tv);**: The result\_tv and solution\_tv TextView widgets from the activity\_main.xml layout are identified and referenced in the subsequent lines using the findViewById method. TextView widgets are utilized to display text or results on the screen.
- + With examples like **assignId(buttonC, R.id.button\_c)** and **assignId(buttonBrackOpen, R.id.button\_open\_bracket)**, in the subsequent lines, a method called assignId is employed for various UI components, including buttons. This method establishes a connection between UI components and the IDs defined in the activity\_main.xml layout file, even though its precise functionality might only partially be evident in the provided code snippet. This connection facilitates programmatic communication between the code and specific UI components.



```
20 usages
48 void assignId(MaterialButton btn,int id){
49     btn = findViewById(id);
50     btn.setOnClickListener(this);
51 }
```

Figure 8: The code snippet of MainActivity.java (code line 48-51) in Exercise 1

The following sentence is the description of this section:

- + **assignId void(MaterialButton btn, int id)**: This section outlines the assignId method, which takes two parameters: btn, a MaterialButton type representing a button widget for Material Design, and id, an integer representing the resource ID of the UI element to be linked to the button.
- + **btn = findViewById(id)**: This line utilizes the provided UI element's ID to search the current layout, locating and associating it with the btn variable through the findViewById function. This effectively establishes a connection between the UI element identified by id and the MaterialButton represented by btn.
- + For the **MaterialButton** with **btn.setOnClickListener(this)**: This code snippet suggests that each time the button is pressed, the **onClick** function will be invoked. Moreover, it indicates that the current class, where this code resides, implements the OnClickListener interface. Consequently, the class must provide the code for handling the button's click event.

```

53      @Override
54      public void onClick(View view) {
55          MaterialButton button =(MaterialButton) view;
56          String buttonText = button.getText().toString();
57          String dataToCalculate = solutionTv.getText().toString();
58
59          if(buttonText.equals("AC")){
60              solutionTv.setText("");
61              resultTv.setText("0");
62              return;
63          }
64          if(buttonText.equals("=")){
65              solutionTv.setText(resultTv.getText());
66              return;
67          }
68          if(buttonText.equals("C")){
69              dataToCalculate = dataToCalculate.substring(0,dataToCalculate.length()-1);
70          }else{
71              dataToCalculate = dataToCalculate+buttonText;
72          }
73          solutionTv.setText(dataToCalculate);
74
75          String finalResult = getResult(dataToCalculate);
76
77          if(!finalResult.equals("Err")){
78              resultTv.setText(finalResult);
79          }
80      }

```

Figure 9: The code snippet of MainActivity.java (code line 53-80) in Exercise 1

The following sentence is the description of this section:

- + **onClick(View view):** Every time a UI element connected to this click listener is clicked, this function is called. It is given an argument as a citation to the related opinion.
- + **MaterialButton button = (MaterialButton) view;** In this case, the view argument is used to build a **MaterialButton** object. To access the clicked view's attributes and features as if it were a **MaterialButton**, this step is required.
- + **String buttonText = button.getText().toString();** This line captures the text content of the clicked button and converts it into a string. Depending on which button was pressed, the message here will determine the subsequent course of action.
- + **String dataToCalculate = solutionTv.getText().toString();** The **solutionTv** **TextView**'s most recent text content is retrieved with this command. This **TextView** presumably shows the output of a calculation or an input string.
- + Verify if the clicked button's text has the letters "AC" (which presumably means "All Clear"). If so, the text in **solutionTv** is cleared, and the text in **resultTv** is set to "0," assuming it reflects the outcome.
- + Check if the text of the clicked button contains the "=" character; if it does, modify the text in **solutionTv** to mirror the information in **resultTv**. This operation may indicate the completion of a computation.
- + In the case of "C," the last character from the **dataToCalculate** string is removed for buttons lacking the labels "AC" or "=", seemingly simulating a backspace function. The content of the clicked button is appended to the **dataToCalculate** string, and if not, the text in **solutionTv** is updated.
- + The **getResult(dataToCalculate)** method returns the result, which is then saved in the **finalResult** variable. The text in **resultTv** is configured to display this result if **finalResult** is not "Err" (meaning a valid result has been calculated).

```

82 String getResult(String data){
83     try{
84         Context context = Context.enter();
85         context.setOptimizationLevel(-1);
86         Scriptable scriptable = context.initStandardObjects();
87         String finalResult = context.evaluateString(scriptable,data, "Javascript", 1, securityDomain: null).toString();
88         if(finalResult.endsWith(".0")){
89             finalResult = finalResult.replace( target: ".0", replacement: "");
90         }
91         return finalResult;
92     }catch (Exception e){
93         return "Err";
94     }
95 }
96 }

```

Figure 10: The code snippet of MainActivity.java (code line 82-end) in Exercise 1

The **getStringResult** method in this code converts input text to JavaScript code using the Mozilla Rhino JavaScript engine. Here is a description of how this code operates:

- + **getStringResult(String data)**: This method yields a new string as output after taking another string as input.
- + **try... catch (Exception e) ...**: A try-catch block surrounding the code signifies its readiness to manage any exceptions that might occur during execution. In the event of an exception, the function returns "Err" to indicate an error.
- + **Context context = Context.enter();**: Before executing JavaScript code, this line establishes a JavaScript environment using the Rhino JavaScript engine.
- + **context.setOptimizationLevel(-1);**: The optimization level of the JavaScript context is adjusted to -1, disabling optimization. This is done to eliminate any potential for unexpected behaviour when evaluating code containing JavaScript expressions.
- + **Scriptable scriptable = context.initStandardObjects();**: In this instance, the execution of JavaScript code is facilitated by establishing a shared JavaScript scope (Scriptable) within the context.
- + **String finalResult = context.executeString(scriptable, data, "Javascript", 1, null).toString();**: This segment of the code is its core. Enclosed within the specified context and scope, the data string is interpreted as JavaScript code, and the resulting evaluation is stored in the **finalResult** string. Essentially, this line executes the JavaScript script present in the data string. JavaScript code can be run within a scriptable scope. The source identification (for error reporting) is "Javascript". The security parameter is configured to 1, and although not utilized in this case, the unused object null can be employed in JavaScript code to define the "this" value.
- + If the process proceeds as intended, the function returns the **finalResult** as a string. This occurs when the JavaScript code within the data string is accurately evaluated.
- + In case of any exceptions during the execution of JavaScript code or issues with the evaluation process, the catch block is triggered, and the function returns "Err" to signify an error.

- File **activity\_main.xml**: The code above incorporates the calculator's main interface, crafted in XML file format. Comprising a **TextView** for displaying results, a **LinearLayout** for system buttons, and distinct function buttons, each assigned a unique ID for computational purposes when accessed from the main screen; the interface components are organized within a **ConstraintLayout**.

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      tools:context=".MainActivity">
8
9      <TextView
10         android:id="@+id/solution_tv"
11         android:layout_width="match_parent"
12         android:layout_height="match_parent"
13         android:layout_margin="16dp"
14         android:text="0"
15         android:textAlignment="textEnd"
16         android:textColor="@color/black"
17         android:textSize="32dp"
18         app:layout_constraintBottom_toTopOf="@+id/result_tv"
19         tools:layout_editor_absoluteX="16dp">
20
21     </TextView>
22
23     <TextView
24         android:id="@+id/result_tv"
25         android:layout_width="match_parent"
26         android:layout_height="wrap_content"
27         android:layout_margin="16dp"
28         android:textAlignment="textEnd"
29         android:textColor="@color/black"
30         android:textSize="64dp"
31         app:layout_constraintBottom_toTopOf="@+id/button_layout"
32         tools:layout_editor_absoluteX="16dp">
33

```

Figure 11: The code snippet of activity\_main.xml (code line 1-33) in Exercise 1

```

34     </TextView>
35     <LinearLayout
36         android:id="@+id/button_layout"
37         android:layout_width="match_parent"
38         android:layout_height="wrap_content"
39         android:layout_alignParentBottom="true"
40         android:background="#F1F1F1"
41         android:orientation="vertical"
42         android:paddingVertical="16dp"
43         app:layout_constraintBottom_toBottomOf="parent"
44         tools:layout_editor_absoluteX="16dp">
45
46         <LinearLayout
47             android:layout_width="match_parent"
48             android:layout_height="wrap_content"
49             android:layout_gravity="center"
50             android:orientation="horizontal">
51
52             <com.google.android.material.button.MaterialButton
53                 android:id="@+id/button_c"
54                 android:layout_width="72dp"
55                 android:layout_height="72dp"
56                 android:layout_margin="12dp"
57                 android:backgroundTint="#F44336"
58                 android:text="C"
59                 android:textColor="@color/white"
60                 android:textSize="32dp"
61                 app:cornerRadius="36dp">
62             </com.google.android.material.button.MaterialButton>
63

```

Figure 12: The code snippet of activity\_main.xml (code line 34-63) in Exercise 1



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

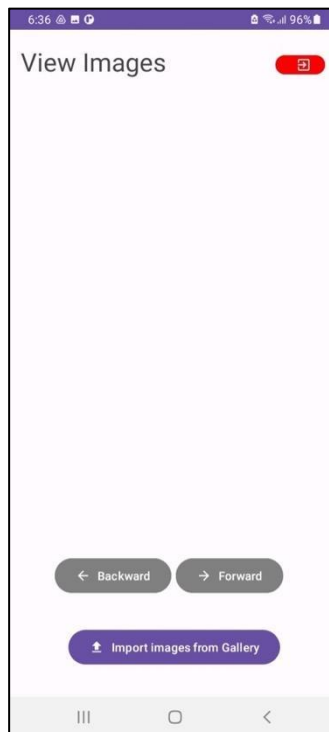
The table below shows the self-assessment of the “Create an Android App allowing users to view images” exercise:

*Table 2: The self-assessment of the “Create an Android App allowing users to view images” exercise*

1.1. Student name.	Minh Nghia Nguyen – 001353656
1.2. Who did you work with?	Phat Dat Truong – 001353261 Chi Trien Nguyen – 001353546
1.3. Which Exercise is this?	<input type="checkbox"/> Exercise 1 <input checked="" type="checkbox"/> Exercise 2 <input type="checkbox"/> Exercise 3
1.4. How well did you complete the exercise?	<input type="checkbox"/> I tried but couldn't complete it <input type="checkbox"/> I did it, but I feel I should have done better <input type="checkbox"/> I did everything that was asked <input checked="" type="checkbox"/> I did more than was asked for
1.5. Briefly explain your answer to question 1.4.	I checked off every need for this workout in the logbook. I also included a function that lets users contribute pictures from their device's photo library to the app.

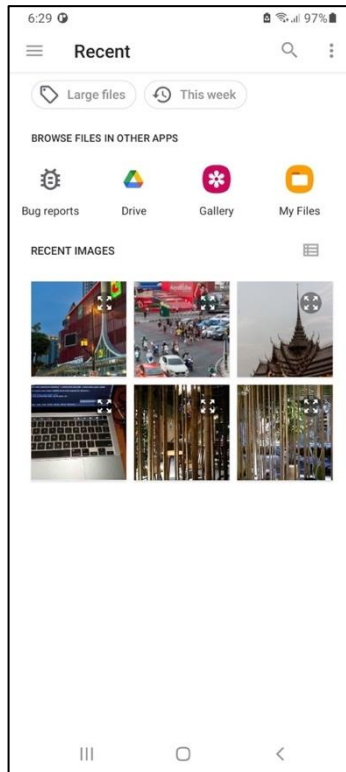
### 2.2.1. Screenshots demonstrating what is achieved

Users may import photographs from their Gallery into the app using the "Forward" and "Backward" buttons under the Image Review function.



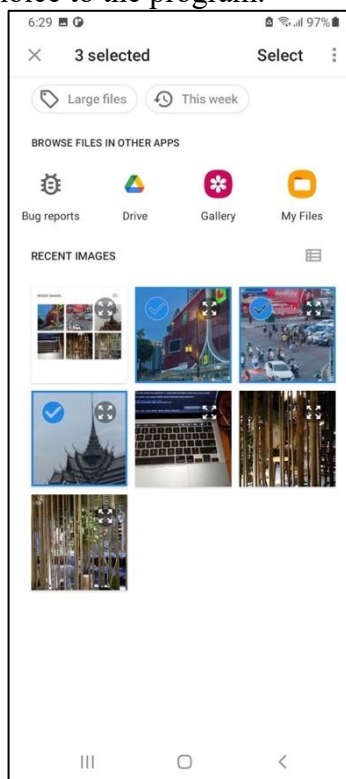
*Figure 13: The main interface of the Image Review application*

When users launch the program, they see this interface.



*Figure 14: The interface when the users click on "Import images from Gallery"*

Upon clicking the "Import images from Gallery" button, the system will open the Gallery, enabling users to select and upload photographs of their choice to the program.



*Figure 15: The interface of choosing images*

The "Select" button is situated in the top-right corner of the device, and users can select photographs by marking them and confirming their choices.



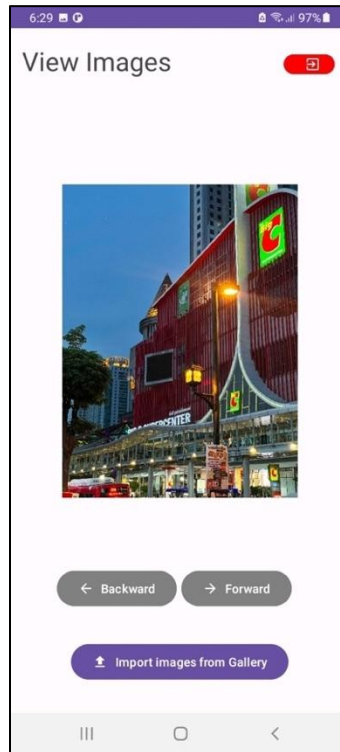
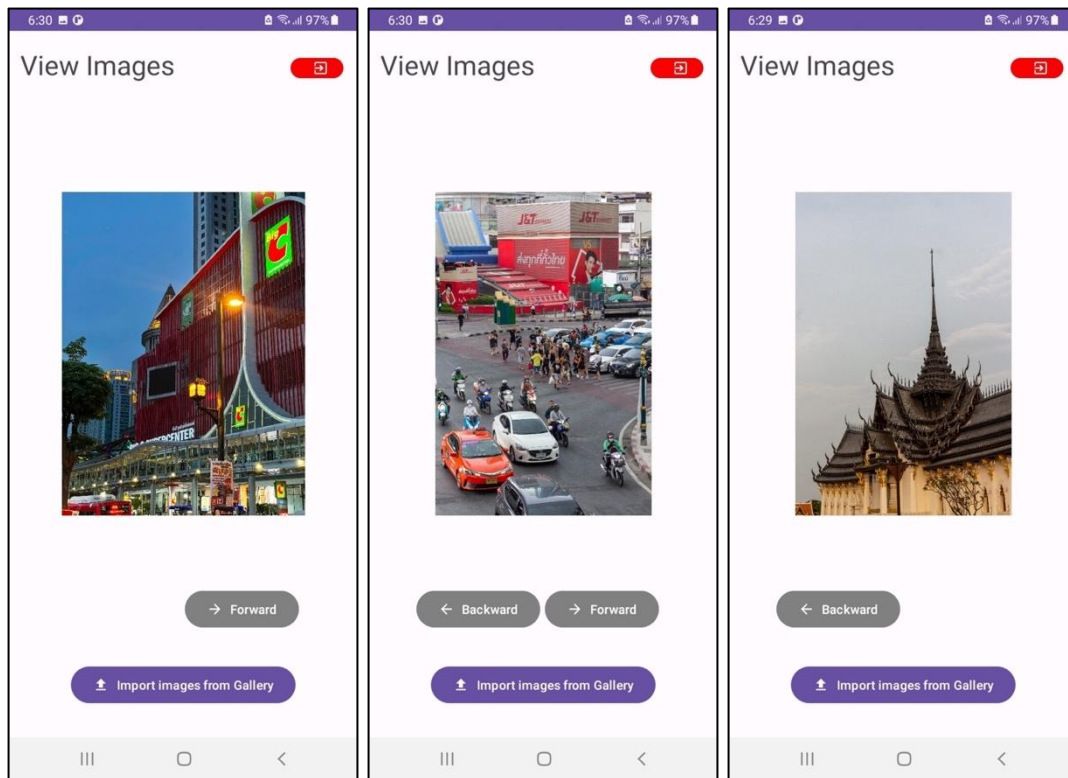


Figure 16: The interface when importing photos successfully

Choose the images that will be displayed in the application's centre.



a) Image at the top of the list b) Image at the middle of the list c) Image at the end of the list

Figure 17: The interfaces when users use the "Backward" and "Forward" buttons to browse the application

From the first to the final image, users may use the "Backward" and "Forward" buttons to navigate. The "Exit" button, the last one, ends the program.

## 2.2.2 Code snippets

Each of the following code lines includes a description of the exercise:

- File **RecyclerViewAdapter.java**: The **RecyclerView** adapter is essential for building a dynamic list of photos in an Android app. This is accomplished by creating a specific layout for each item, attaching data to an **ImageView**, controlling how items are shown, and ensuring **Uri** objects are used effectively.

```
20      @NonNull
21      @Override
22      public RecyclerView.ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
23          LayoutInflater inflater = LayoutInflater.from(parent.getContext());
24          View view = inflater.inflate(R.layout.custom_single_img, parent, attachToRoot: false);
25
26          return new ViewHolder(view);
27      }
28
29      @Override
30      public void onBindViewHolder(@NonNull RecyclerView.ViewHolder holder, int position) {
31          holder.imageView.setImageURI(uriArrayList.get(position));
32      }
33
34      @Override
35      public int getItemCount() { return uriArrayList.size(); }
36
37      4 usages
38
39      public class ViewHolder extends RecyclerView.ViewHolder {
40          2 usages
41          ImageView imageView;
42          1 usage
43          public ViewHolder(@NonNull View itemView) {
44              super(itemView);
45              imageView = itemView.findViewById(R.id.imageView);
46          }
47      }
```

Figure 18: The code snippet of *RecyclerViewAdapter.java* (code line 20-46) in Exercise 2

A description of the inner components is provided below:

- + **onCreateViewHolder()**: The **onCreateViewHolder** function is called whenever the **RecyclerView** needs to generate a new **ViewHolder** for an item. The **R.layout.custom\_single\_img** layout XML file is used to inflate a view. A single item in the **RecyclerView** is represented by this inflated view. After that, a **ViewHolder** object is created to contain it and return it.
- + **onBindViewHolder() Method**: When a **RecyclerView** item has to be modified or connected to data, this function is invoked. It sets up the **ImageView** within the **ViewHolder** in this case to show the picture associated with the **Uri** at the current location in the **uriArrayList**.
- + **getItemCount() Method**: The number of elements in the **uriArrayList** is returned by this method. The **RecyclerView** receives information from this value regarding how many things should be shown.
- + **ViewHolder Inner Class**: a **RecyclerViewAdapter** inner class that manages holding views. In the **RecyclerView**, it symbolizes the idea of a single item. It keeps track of a widget called an **ImageView** that will show the picture. Within the constructor of the **ViewHolder**, the **ImageView** is found and initialized from the inflated view.
- File **MainActivity.java**: This code exemplifies the fundamental functionality of an Android application, allowing users to view and navigate through a collection of photos. Users can add photos to the program, switch between them, and delete them. The **RecyclerViewAdapter** class governs the logic

for displaying images, although another section of your code delineates the specific implementation of this logic. A RecyclerView showcases the photographs, creating a streamlined and efficient UI.

```
32  @Override
33  protected void onCreate(Bundle savedInstanceState) {
34      super.onCreate(savedInstanceState);
35      setContentView(R.layout.activity_main);
36
37      recyclerView = findViewById(R.id.recyclerView);
38      btnAdd = findViewById(R.id.btnAdd);
39      btnPrevious = findViewById(R.id.btnPrevious);
40      btnNext = findViewById(R.id.btnNext);
41      btnExit = findViewById(R.id.btnExit);
42
43      adapter = new RecyclerViewAdapter(uri);
44      recyclerView.setLayoutManager(new GridLayoutManager(context, MainActivity.this, spanCount: 1));
45      recyclerView.setAdapter(adapter);
46
47      if (ContextCompat.checkSelfPermission(context, Manifest.permission.READ_EXTERNAL_STORAGE) != PackageManager.PERMISSION_GRANTED) {
48          ActivityCompat.requestPermissions(activity, MainActivity.this, new String[]{Manifest.permission.READ_EXTERNAL_STORAGE}, Read_Permission);
49      }
}
```

Figure 19: The code snippet of MainActivity.java (code line 32-49) in Exercise 2

A description of the inner components is provided below:

- + **onCreate(Bundle savedInstanceState):** Upon the creation of the activity, this method, an Android lifecycle method, is called. It is the initial point for configuring the user interface and initializing components.
- + **setContentView(R.layout.activity\_main);** This line modifies the layout specified in the **activity\_main.xml** file, setting it as the content view for the activity. It defines the organizational structure of the activity.
- + **adapter = new RecyclerViewAdapter(uri);** In this section, an instance of the RecyclerViewAdapter class is instantiated and provided with the "uri" list as input. This adapter is responsible for supplying information to the RecyclerView.
- + **setLayoutManager(new GridLayoutManager(MainActivity.this, 1));** The layout manager for the **RecyclerView** is set in this line. It describes how to create a grid with one item per row by using a **GridLayoutManager** with a single column.
- + **recyclerView.setAdapter(adapter);** This line designates the adapter, which governs how data is shown within the **RecyclerView**, to the "adapter" instance.
- + Permission check:
  - **ContextCompat.checkSelfPermission(...):** This code segment verifies whether the app possesses the **READ\_EXTERNAL\_STORAGE** permission. ContextCompat is employed to ensure compatibility across various Android versions.
  - **ActivityCompat.requestPermissions(...):** This code initiates a request to the user for the **READ\_EXTERNAL\_STORAGE** permission if it has yet to be granted (**PERMISSION\_GRANTED**). The Read\_Permission constant is utilized as the permission request code in a permission dialogue, facilitating this request.

```
50
51  btnAdd.setOnClickListener(new View.OnClickListener() {
52      @Override
53      public void onClick(View v) {
54          Intent intent = new Intent();
55          intent.setType("image/*");
56
57          if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.JELLY_BEAN_MR2) {
58              intent.putExtra(Intent.EXTRA_ALLOW_MULTIPLE, value: true);
59          }
60          intent.setAction(Intent.ACTION_GET_CONTENT);
61          startActivityForResult(Intent.createChooser(intent, title: "Select Image: "), requestCode: 1);
62      }
63  });
64  });
65  });
66  });
67  });
68  });
69  });
70  });
71  });
72  });
73  });
74  });
75  });
76  });
77  });
78  });
79  });
80  });
81  });
82  });
83  });
84  });
85  });
86  });
87  });
88  });
89  });
90  });
91  });
92  });
93  });
94  });
95  });
96  });
97  });
98  });
99  });
100  });
101  });
102  });
103  });
104  });
105  });
106  });
107  });
108  });
109  });
110  });
111  });
112  });
113  });
114  });
115  });
116  });
117  });
118  });
119  });
120  });
121  });
122  });
123  });
124  });
125  });
126  });
127  });
128  });
129  });
130  });
131  });
132  });
133  });
134  });
135  });
136  });
137  });
138  });
139  });
140  });
141  });
142  });
143  });
144  });
145  });
146  });
147  });
148  });
149  });
150  });
151  });
152  });
153  });
154  });
155  });
156  });
157  });
158  });
159  });
160  });
161  });
162  });
163  });
164  });
165  });
166  });
167  });
168  });
169  });
170  });
171  });
172  });
173  });
174  });
175  });
176  });
177  });
178  });
179  });
180  });
181  });
182  });
183  });
184  });
185  });
186  });
187  });
188  });
189  });
190  });
191  });
192  });
193  });
194  });
195  });
196  });
197  });
198  });
199  });
200  });
201  });
202  });
203  });
204  });
205  });
206  });
207  });
208  });
209  });
210  });
211  });
212  });
213  });
214  });
215  });
216  });
217  });
218  });
219  });
220  });
221  });
222  });
223  });
224  });
225  });
226  });
227  });
228  });
229  });
230  });
231  });
232  });
233  });
234  });
235  });
236  });
237  });
238  });
239  });
240  });
241  });
242  });
243  });
244  });
245  });
246  });
247  });
248  });
249  });
250  });
251  });
252  });
253  });
254  });
255  });
256  });
257  });
258  });
259  });
260  });
261  });
262  });
263  });
264  });
265  });
266  });
267  });
268  });
269  });
270  });
271  });
272  });
273  });
274  });
275  });
276  });
277  });
278  });
279  });
280  });
281  });
282  });
283  });
284  });
285  });
286  });
287  });
288  });
289  });
290  });
291  });
292  });
293  });
294  });
295  });
296  });
297  });
298  });
299  });
300  });
301  });
302  });
303  });
304  });
305  });
306  });
307  });
308  });
309  });
310  });
311  });
312  });
313  });
314  });
315  });
316  });
317  });
318  });
319  });
320  });
321  });
322  });
323  });
324  });
325  });
326  });
327  });
328  });
329  });
330  });
331  });
332  });
333  });
334  });
335  });
336  });
337  });
338  });
339  });
340  });
341  });
342  });
343  });
344  });
345  });
346  });
347  });
348  });
349  });
350  });
351  });
352  });
353  });
354  });
355  });
356  });
357  });
358  });
359  });
360  });
361  });
362  });
363  });
364  });
365  });
366  });
367  });
368  });
369  });
370  });
371  });
372  });
373  });
374  });
375  });
376  });
377  });
378  });
379  });
380  });
381  });
382  });
383  });
384  });
385  });
386  });
387  });
388  });
389  });
390  });
391  });
392  });
393  });
394  });
395  });
396  });
397  });
398  });
399  });
400  });
401  });
402  });
403  });
404  });
405  });
406  });
407  });
408  });
409  });
410  });
411  });
412  });
413  });
414  });
415  });
416  });
417  });
418  });
419  });
420  });
421  });
422  });
423  });
424  });
425  });
426  });
427  });
428  });
429  });
430  });
431  });
432  });
433  });
434  });
435  });
436  });
437  });
438  });
439  });
440  });
441  });
442  });
443  });
444  });
445  });
446  });
447  });
448  });
449  });
450  });
451  });
452  });
453  });
454  });
455  });
456  });
457  });
458  });
459  });
460  });
461  });
462  });
463  });
464  });
465  });
466  });
467  });
468  });
469  });
470  });
471  });
472  });
473  });
474  });
475  });
476  });
477  });
478  });
479  });
480  });
481  });
482  });
483  });
484  });
485  });
486  });
487  });
488  });
489  });
490  });
491  });
492  });
493  });
494  });
495  });
496  });
497  });
498  });
499  });
500  });
501  });
502  });
503  });
504  });
505  });
506  });
507  });
508  });
509  });
510  });
511  });
512  });
513  });
514  });
515  });
516  });
517  });
518  });
519  });
520  });
521  });
522  });
523  });
524  });
525  });
526  });
527  });
528  });
529  });
530  });
531  });
532  });
533  });
534  });
535  });
536  });
537  });
538  });
539  });
540  });
541  });
542  });
543  });
544  });
545  });
546  });
547  });
548  });
549  });
550  });
551  });
552  });
553  });
554  });
555  });
556  });
557  });
558  });
559  });
560  });
561  });
562  });
563  });
564  });
565  });
566  });
567  });
568  });
569  });
570  });
571  });
572  });
573  });
574  });
575  });
576  });
577  });
578  });
579  });
580  });
581  });
582  });
583  });
584  });
585  });
586  });
587  });
588  });
589  });
590  });
591  });
592  });
593  });
594  });
595  });
596  });
597  });
598  });
599  });
600  });
601  });
602  });
603  });
604  });
605  });
606  });
607  });
608  });
609  });
610  });
611  });
612  });
613  });
614  });
615  });
616  });
617  });
618  });
619  });
620  });
621  });
622  });
623  });
624  });
625  });
626  });
627  });
628  });
629  });
630  });
631  });
632  });
633  });
634  });
635  });
636  });
637  });
638  });
639  });
640  });
641  });
642  });
643  });
644  });
645  });
646  });
647  });
648  });
649  });
650  });
651  });
652  });
653  });
654  });
655  });
656  });
657  });
658  });
659  });
660  });
661  });
662  });
663  });
664  });
665  });
666  });
667  });
668  });
669  });
670  });
671  });
672  });
673  });
674  });
675  });
676  });
677  });
678  });
679  });
680  });
681  });
682  });
683  });
684  });
685  });
686  });
687  });
688  });
689  });
690  });
691  });
692  });
693  });
694  });
695  });
696  });
697  });
698  });
699  });
700  });
701  });
702  });
703  });
704  });
705  });
706  });
707  });
708  });
709  });
710  });
711  });
712  });
713  });
714  });
715  });
716  });
717  });
718  });
719  });
720  });
721  });
722  });
723  });
724  });
725  });
726  });
727  });
728  });
729  });
730  });
731  });
732  });
733  });
734  });
735  });
736  });
737  });
738  });
739  });
740  });
741  });
742  });
743  });
744  });
745  });
746  });
747  });
748  });
749  });
750  });
751  });
752  });
753  });
754  });
755  });
756  });
757  });
758  });
759  });
760  });
761  });
762  });
763  });
764  });
765  });
766  });
767  });
768  });
769  });
770  });
771  });
772  });
773  });
774  });
775  });
776  });
777  });
778  });
779  });
780  });
781  });
782  });
783  });
784  });
785  });
786  });
787  });
788  });
789  });
790  });
791  });
792  });
793  });
794  });
795  });
796  });
797  });
798  });
799  });
800  });
801  });
802  });
803  });
804  });
805  });
806  });
807  });
808  });
809  });
810  });
811  });
812  });
813  });
814  });
815  });
816  });
817  });
818  });
819  });
820  });
821  });
822  });
823  });
824  });
825  });
826  });
827  });
828  });
829  });
830  });
831  });
832  });
833  });
834  });
835  });
836  });
837  });
838  });
839  });
840  });
841  });
842  });
843  });
844  });
845  });
846  });
847  });
848  });
849  });
850  });
851  });
852  });
853  });
854  });
855  });
856  });
857  });
858  });
859  });
860  });
861  });
862  });
863  });
864  });
865  });
866  });
867  });
868  });
869  });
870  });
871  });
872  });
873  });
874  });
875  });
876  });
877  });
878  });
879  });
880  });
881  });
882  });
883  });
884  });
885  });
886  });
887  });
888  });
889  });
890  });
891  });
892  });
893  });
894  });
895  });
896  });
897  });
898  });
899  });
900  });
901  });
902  });
903  });
904  });
905  });
906  });
907  });
908  });
909  });
910  });
911  });
912  });
913  });
914  });
915  });
916  });
917  });
918  });
919  });
920  });
921  });
922  });
923  });
924  });
925  });
926  });
927  });
928  });
929  });
930  });
931  });
932  });
933  });
934  });
935  });
936  });
937  });
938  });
939  });
940  });
941  });
942  });
943  });
944  });
945  });
946  });
947  });
948  });
949  });
950  });
951  });
952  });
953  });
954  });
955  });
956  });
957  });
958  });
959  });
960  });
961  });
962  });
963  });
964  });
965  });
966  });
967  });
968  });
969  });
970  });
971  });
972  });
973  });
974  });
975  });
976  });
977  });
978  });
979  });
980  });
981  });
982  });
983  });
984  });
985  });
986  });
987  });
988  });
989  });
990  });
991  });
992  });
993  });
994  });
995  });
996  });
997  });
998  });
999  });
1000  });
1001  });
1002  });
1003  });
1004  });
1005  });
1006  });
1007  });
1008  });
1009  });
1010  });
1011  });
1012  });
1013  });
1014  });
1015  });
1016  });
1017  });
1018  });
1019  });
1020  });
1021  });
1022  });
1023  });
1024  });
1025  });
1026  });
1027  });
1028  });
1029  });
1030  });
1031  });
1032  });
1033  });
1034  });
1035  });
1036  });
1037  });
1038  });
1039  });
1040  });
1041  });
1042  });
1043  });
1044  });
1045  });
1046  });
1047  });
1048  });
1049  });
1050  });
1051  });
1052  });
1053  });
1054  });
1055  });
1056  });
1057  });
1058  });
1059  });
1060  });
1061  });
1062  });
1063  });
1064  });
1065  });
1066  });
1067  });
1068  });
1069  });
1070  });
1071  });
1072  });
1073  });
1074  });
1075  });
1076  });
1077  });
1078  });
1079  });
1080  });
1081  });
1082  });
1083  });
1084  });
1085  });
1086  });
1087  });
1088  });
1089  });
1090  });
1091  });
1092  });
1093  });
1094  });
1095  });
1096  });
1097  });
1098  });
1099  });
1100  });
1101  });
1102  });
1103  });
1104  });
1105  });
1106  });
1107  });
1108  });
1109  });
1110  });
1111  });
1112  });
1113  });
1114  });
1115  });
1116  });
1117  });
1118  });
1119  });
1120  });
1121  });
1122  });
1123  });
1124  });
1125  });
1126  });
1127  });
1128  });
1129  });
1130  });
1131  });
1132  });
1133  });
1134  });
1135  });
1136  });
1137  });
1138  });
1139  });
1140  });
1141  });
1142  });
1143  });
1144  });
1145  });
1146  });
1147  });
1148  });
1149  });
1150  });
1151  });
1152  });
1153  });
1154  });
1155  });
1156  });
1157  });
1158  });
1159  });
1160  });
1161  });
1162  });
1163  });
1164  });
1165  });
1166  });
1167  });
1168  });
1169  });
1170  });
1171  });
1172  });
1173  });
1174  });
1175  });
1176  });
1177  });
1178  });
1179  });
1180  });
1181  });
1182  });
1183  });
1184  });
1185  });
1186  });
1187  });
1188  });
1189  });
1190  });
1191  });
1192  });
1193  });
1194  });
1195  });
1196  });
1197  });
1198  });
1199  });
1200  });
1201  });
1202  });
1203  });
1204  });
1205  });
1206  });
1207  });
1208  });
1209  });
1210  });
1211  });
1212  });
1213  });
1214  });
1215  });
1216  });
1217  });
1218  });
1219  });
1220  });
1221  });
1222  });
1223  });
1224  });
1225  });
1226  });
1227  });
1228  });
1229  });
1230  });
1231  });
1232  });
1233  });
1234  });
1235  });
1236  });
1237  });
1238  });
1239  });
1240  });
1241  });
1242  });
1243  });
1244  });
1245  });
1246  });
1247  });
1248  });
1249  });
1250  });
1251  });
1252  });
1253  });
1254  });
1255  });
1256  });
1257  });
1258  });
1259  });
1260  });
1261  });
1262  });
1263  });
1264  });
1265  });
1266  });
1267  });
1268  });
1269  });
1270  });
1271  });
1272  });
1273  });
1274  });
1275  });
1276  });
1277  });
1278  });
1279  });
1280  });
1281  });
1282  });
1283  });
1284  });
1285  });
1286  });
1287  });
1288  });
1289  });
1290  });
1291  });
1292  });
1293  });
1294  });
1295  });
1296  });
1297  });
1298  });
1299  });
1300  });
1301  });
1302  });
1303  });
1304  });
1305  });
1306  });
1307  });
1308  });
1309  });
1310  });
1311  });
1312  });
1313  });
1314  });
1315  });
1316  });
1317  });
1318  });
1319  });
1320  });
1321  });
1322  });
1323  });
1324  });
1325  });
1326  });
1327  });
1328  });
1329  });
1330  });
1331  });
1332  });
1333  });
1334  });
1335  });
1336  });
1337  });
1338  });
1339  });
1340  });
1341  });
1342  });
1343  });
1344  });
1345  });
1346  });
1347  });
1348  });
1349  });
1350  });
1351  });
1352  });
1353  });
1354  });
1355  });
1356  });
1357  });
1358  });
1359  });
1360  });
1361  });
1362  });
1363  });
1364  });
1365  });
1366  });
1367  });
1368  });
1369  });
1370  });
1371  });
1372  });
1373  });
1374  });
1375  });
1376  });
1377  });
1378  });
1379  });
1380  });
1381  });
1382  });
1383  });
1384  });
1385  });
1386  });
1387  });
1388  });
1389  });
1390  });
1391  });
1392  });
1393  });
1394  });
1395  });
1396  });
1397  });
1398  });
1399  });
1400  });
1401  });
1402  });
1403  });
1404  });
1405  });
1406  });
1407  });
1408  });
1409  });
1410  });
1411  });
1412  });
1413  });
1414  });
1415  });
1416  });
1417  });
1418  });
1419  });
1420  });
1421  });
1422  });
1423  });
1424  });
1425  });
1426  });
1427  });
1428  });
1429  });
1430  });
1431  });
1432  });
1433  });
1434  });
1435  });
1436  });
1437  });
1438  });
1439  });
1440  });
1441  });
1442  });
1443  });
1444  });
1445  });
1446  });
1447  });
1448  });
1449  });
1450  });
1451  });
1452  });
1453  });
1454  });
1455  });
1456  });
1457  });
1458  });
1459  });
1460  });
1461  });
1462  });
1463  });
1464  });
1465  });
1466  });
1467  });
1468  });
1469  });
1470  });
1471  });
1472  });
1473  });
1474  });
1475  });
1476  });
1477  });
1478  });
1479  });
1480  });
1481  });
1482  });
1483  });
1484  });
1485  });
1486  });
1487  });
1488  });
1489  });
1490  });
1491  });
1492  });
1493  });
1494  });
1495  });
1496  });
1497  });
1498  });
1499  });
1500  });
1501  });
1502  });
1503  });
1504  });
1505  });
1506  });
1507  });
1508  });
1509  });
1510  });
1511  });
1512  });
1513  });
1514  });
1515  });
1516  });
1517  });
1518  });
1519  });
1520  });
1521  });
1522  });
1523  });
1524  });
1525  });
1526  });
1527  });
1528  });
1529  });
1530  });
1531  });
1532  });
1533  });
1534  });
1535  });
1536  });
1537  });
1538  });
1539  });
1540  });
1541  });
1542  });
1543  });
1544  });
1545  });
1546  });
1547  });
1548  });
1549  });
1550  });
1551  });
1552  });
1553  });
1554  });
1555  });
1556  });
1557  });
1558  });
1559  });
1560  });
1561  });
1562  });
1563  });
1564  });
1565  });
1566  });
1567  });
1568  });
1569  });
1570  });
1571  });
1572  });
1573  });
1574  });
1575  });
1576  });
1577  });
1578  });
1579  });
1580  });
1581  });
1582  });
1583  });
1584  });
1585  });
1586  });
1587  });
1588  });
1589  });
1590  });
1591  });
1592  });
1593  });
1594  });
1595  });
1596  });
1597  });
1598  });
1599  });
1600  });
1601  });
1602  });
1603  });
1604  });
1605  });
1606  });
1607  });
1608  });
1609  });
1610  });
1611  });
1612  });
1613  });
1614  });
1615  });
1616  });
1617  });
1618  });
1619  });
1620  });
1621  });
1622  });
1623  });
1624  });
1625  });
1626  });
1627  });
1628  });
1629  });
1630  });
1631  });
1632  });
1633  });
1634  });
1635  });
1636  });
1637  });
1638  });
1639  });
1640  });
1641  });
1642  });
1643  });
1644  });
1645  });
1646  });
1647  });
1648  });
1649  });
1650  });
1651  });
1652  });
1653  });
1654  });
1655  });
1656  });
1657  });
1658  });
1659  });
1660  });
1661  });
1662  });
1663  });
1664  });
1665  });
1666  });
1667  });
1668  });
1669  });
1670  });
1671  });
1672  });
1673  });
1674  });
1675  });
1676  });
1677  });
1678  });
1679  });
1680  });
1681  });
1682  });
1683  });
1684  });
1685  });
1686  });
1687  });
1688  });
1689  });
1690  });
1691  });
1692  });
1693  });
1694  });
1695  });
1696  });
1697  });
1698  });
1699  });
1700  });
1701  });
1702  });
1703  });
1704  });
1705  });
1706  });
1707  });
1708  });
1709  });
1710  });
1711  });
1712  });
1713  });
1714  });
1715  });
1716  });
1717  });
1718  });
1719  });
1720  });
1721  });
1722  });
1723  });
1724  });
1725  });
1726  });
1727  });
1728  });
1729  });
1730  });
1731  });
1732  });
1733  });
1734  });
1735  });
1736  });
1737  });
1738  });
1739  });
1740  });
1741  });
1742  });
1743  });
1744  });
1745  });
1746  });
1747  });
1748  });
1749  });
1750  });
1751  });
1752  });
1753  });
1754  });
1755  });
1756  });
1757  });
1758  });
1759  });
1760  });
1761  });
1762  });
1763  });
1764  });
1765  });
1766  });
1767  });
1768  });
1769  });
1770  });
1771  });
1772  });
1773  });
1774  });
1775  });
1776  });
1777  });
1778  });
1779  });
1780  });
1781  });
1782  });
1783  });
1784  });
1785  });
1786  });
1787  });
1788  });
1789  });
1790  });
1791  });
1792  });
1793  });
1794  });
1795  });
1796  });
1797  });
1798  });
1799  });
1800  });
1801  });
1802  });
1803  });
1804  });
1805  });
1806  });
1807  });
1808  });
1809  });
1810  });
1811  });
1812  });
1813  });
1814  });
1815  });
1816  });
1817  });
1818  });
1819  });
1820  });
1821  });
1822  });
1823  });
1824  });
1825  });
1826  });
18
```

Figure 20: The code snippet of MainActivity.java (code line 50-64) in Exercise 2

The click listeners dictate the actions triggered when the "Add" button is pressed. The internal mechanisms of the onClick method encompass:

- + **if (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.JELLY\_BEAN\_MR2) { ... }:** This condition verifies whether the device is running Jelly Bean MR2 (Android 4.3) or a higher Android version. If this condition is true, the code executes **intent.putExtra(Intent.EXTRA\_ALLOW\_MULTIPLE, true);**, enabling the selection of multiple images by adding an extra parameter to the intent. This functionality lets users choose multiple photos simultaneously if their device supports it.
- + The intent action is set to "get content," a common configuration used for selecting files or content from the device, accomplished by using **intent.setAction(Intent.ACTION\_GET\_CONTENT)**.
- + **Intent.createChooser(intent, "Select Image: ");** launches an activity to select and choose photos. The intention is to enhance user-friendliness by enveloping the process in a chooser dialogue using the createChooser function. The request code 1, provided as the second input, serves to identify the outcome when the picture selection activity is completed. The subsequent onActivityResult section will manage the associated tasks.



```
89  @Override
90  protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
91      super.onActivityResult(requestCode, resultCode, data);
92
93      if (requestCode == 1 && resultCode == Activity.RESULT_OK) {
94          if (data.getClipData() != null) {
95              int x = data.getClipData().getItemCount();
96
97              for (int i = 0; i < x; i++) {
98                  uri.add(data.getClipData().getItemAt(i).getUri());
99              }
100             adapter.notifyDataSetChanged();
101         } else if (data.getData() != null) {
102             String imageURL = data.getData().getPath();
103         }
104     }
105 }
106 }
```

Figure 21: The code snippet of MainActivity.java (code line 89-106) in Exercise 2

With the **onActivityResult()**, here is a breakdown of each section of the code:

- + Functionality: **if (requestCode == 1 and resultCode = Activity.RESULT\_OK) {...}**: This condition evaluates whether the result originates from the activity initiated with request code 1 (associated with selecting photos) and determines whether the outcome is successful (**Activity.RESULT\_OK**).
- + Inside this condition:
  - **if (data.getClipData() != null) { ... }**: This section examines whether the selected data contains multiple items (multiple images). If multiple items are selected, it counts them using **data.getClipData().getItemCount()**. It then iterates through each chosen item, retrieves the **Uri**, and adds it to the uri list. After adding the images to the uri list, it calls **adapter.notifyDataSetChanged()** to inform the RecyclerView's adapter that the data has changed, refreshing the view.
  - **else if (data.getData() != null) { ... }**: This code section uses the **image.getData().getPath()** function to try to get the path of the single item (image) that has been selected. It's crucial to remember that this specific line doesn't have any significant functionality or purpose within the context of the code that has been presented.

```

122     private void displayImage(int index) {
123         if (index >= 0 && index < uri.size()) {
124             currentIndex = index;
125             if (adapter != null) {
126                 recyclerView.scrollToPosition(index);
127             }
128             updateButtonVisibility();
129         }
130     }
131
132     1 usage
133     private void updateButtonVisibility() {
134         if (currentIndex == 0) {
135             btnPrevious.setVisibility(View.INVISIBLE);
136         } else {
137             btnPrevious.setVisibility(View.VISIBLE);
138         }
139
140         if (currentIndex == uri.size() - 1) {
141             btnNext.setVisibility(View.INVISIBLE);
142         } else {
143             btnNext.setVisibility(View.VISIBLE);
144         }
145     }

```

Figure 22: The code snippet of MainActivity.java (code line 122-144) in Exercise 2

These techniques control how photos are shown in the app's user interface and how the "Forward" and "Backward" navigation buttons operate. Here is the description of each technique:

+ **displayImage(int index):** Functionality:

- **if (index >= 0 && index < uri.size()) { ... }:** This condition verifies that the supplied index is safe, valid, and does not exceed the size of the uri list by determining if it is within the permissible range of indices for the uri list.
- If the index is valid:
  - **currentIndex = index;** The supplied index is updated in the currentIndex variable, which keeps track of the index of the image that is now being shown.
  - **if (adapter != null) { recyclerView.scrollToPosition(index); }:** To make sure that the chosen picture is visible within the RecyclerView's viewport, the RecyclerView is scrolled to the location indicated by the index.
  - **updateButtonVisibility():** To change the visibility of the "Forward" and "Backward" buttons based on the current picture index, it calls the updateButtonVisibility function.

+ **updateButtonVisibility():** Functionality:

- **if (currentIndex == 0) { ... }:** This condition is true if the currentIndex is at index 0, which is the first image in the list. In such case
- If the index is valid: **btnPrevious.setVisibility(View.INVISIBLE);** The "Backward" button can be hidden by setting its visibility to View.INVISIBLE. If there are no prior photographs, the user cannot access one.
- If the currentIndex (index higher than 0) is not at the first image: **btnPrevious.setVisibility(View.VISIBLE);** It sets the "Backward" button's visibility to View, making it visible. This enables the user to go back to an earlier picture.



- **if (currentImageIndex == uri.size() - 1) { ... }:** This condition checks if the currentImageIndex is at the last image in the list (**index equal to uri.size() - 1**). If it is: **btnNext.setVisibility(View.INVISIBLE);** By changing the "Forward" button's visibility to View, it is hidden.INVISIBLE. When there are no more photos to display, this blocks the user from moving on to the subsequent image.
  - If the last image is not where the **currentImageIndex** is: **btnNext.setVisibility(View.VISIBLE);** It sets the "Forward" button's visibility to View, making it visible.VISIBLE. The user may now go to the next image, thanks to this.
- File **custom\_single\_img.xml**: One ImageView widget is defined in the XML layout code and takes up the full ConstraintLayout to which it belongs. This ImageView has a particular size and uses constraints to keep it in the layout's centre.

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent">
7
8      <ImageView
9          android:id="@+id/imageView"
10         android:layout_width="398dp"
11         android:layout_height="358dp"
12         app:layout_constraintBottom_toBottomOf="parent"
13         app:layout_constraintEnd_toEndOf="parent"
14         app:layout_constraintStart_toStartOf="parent"
15         app:layout_constraintTop_toTopOf="parent"
16         tools:srcCompat="@tools:sample/avatars" />
17
18 </androidx.constraintlayout.widget.ConstraintLayout>

```

Figure 23: The code snippet of custom\_single\_img.xml in Exercise 2

- File **activity\_main.xml**: The layout is made for an app that lets users import photographs from their gallery, browse through images using the "Forward" and "Backward" buttons, and quit the program using the "Exit" button. photographs are shown in a RecyclerView in this style. The activity's name or header is represented by the "View Images" TextView. The actual implementation of button functionality and RecyclerView behaviour would occur in the relevant activity, such as "MainActivity."

```

1      <?xml version="1.0" encoding="utf-8"?>
2
3      <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
4          xmlns:app="http://schemas.android.com/apk/res-auto"
5          xmlns:tools="http://schemas.android.com/tools"
6          android:layout_width="match_parent"
7          android:layout_height="match_parent"
8          tools:context=".MainActivity">
9
10         <androidx.recyclerview.widget.RecyclerView
11             android:id="@+id/recyclerView"
12             android:layout_width="368dp"
13             android:layout_height="366dp"
14             android:layout_marginStart="5dp"
15             android:layout_marginEnd="5dp"
16             app:layout_constraintBottom_toBottomOf="parent"
17             app:layout_constraintEnd_toEndOf="parent"
18             app:layout_constraintHorizontal_bias="0.484"
19             app:layout_constraintStart_toStartOf="parent"
20             app:layout_constraintTop_toTopOf="parent"
21             app:layout_constraintVertical_bias="0.419" />
22

```

Figure 24: The code snippet of activity\_main.xml (code line 1-22) in Exercise 2

```

23         <Button
24             android:id="@+id/btnAdd"
25             android:layout_width="wrap_content"
26             android:layout_height="wrap_content"
27             android:layout_marginTop="20dp"
28             android:layout_marginBottom="20dp"
29             android:text="Import images from Gallery"
30             app:icon="@drawable/baseline_upload_24"
31             app:layout_constraintBottom_toBottomOf="parent"
32             app:layout_constraintEnd_toEndOf="parent"
33             app:layout_constraintHorizontal_bias="0.502"
34             app:layout_constraintStart_toStartOf="parent"
35             app:layout_constraintTop_toBottomOf="@+id/recyclerView"
36             app:layout_constraintVertical_bias="0.911" />
37
38         <Button
39             android:id="@+id/btnNext"
40             android:layout_width="wrap_content"
41             android:layout_height="wrap_content"
42             android:layout_marginEnd="64dp"
43             android:backgroundTint="#808080"
44             android:text="Forward"
45             app:icon="@drawable/baseline_arrow_forward_24"
46             app:layout_constraintBottom_toBottomOf="parent"
47             app:layout_constraintEnd_toEndOf="parent"
48             app:layout_constraintTop_toBottomOf="@+id/recyclerView"
49             app:layout_constraintVertical_bias="0.39"
50             app:strokeColor="#E91E63" />
51

```

Figure 25: The code snippet of activity\_main.xml (code line 23-51) in Exercise 2

### 2.3. Exercise 3: Use Android Persistence to store data

The table below shows the self-assessment of the “Use Android Persistence to store data” exercise:

*Table 3: The self-assessment of the “Use Android Persistence to store data” exercise*

1.1. Student name.	Chi Trien Nguyen - 001353546
1.2. Who did you work with?	Phat Dat Truong - 001353261 Minh Nghia Nguyen - 001353656
1.3. Which Exercise is this?	<input type="checkbox"/> Exercise 1 <input type="checkbox"/> Exercise 2 <input checked="" type="checkbox"/> Exercise 3
1.4. How well did you complete the exercise	<input type="checkbox"/> I tried but couldn't complete it <input type="checkbox"/> I did it, but I feel I should have done better <input checked="" type="checkbox"/> I did everything that was asked <input type="checkbox"/> I did more than was asked for
1.5. Briefly explain your answer to question 1.4.	I have met all the requirements outlined in the logbook for this task.

#### 2.3.1. Screenshots demonstrating what is achieved

This is a Contact Database application; the user can store their contact information, such as name, date of birth, email, and picture.

- **Main Interface** (The main interface displays a list of contacts if the database has them, otherwise, it shows "no contacts available." A button at the bottom suitable leads to the Add Contact Interface, and clicking on an available contact leads to the Update Contact Interface for updating new information).



*Figure 26: Main Interface without contact of Contact Database application Contact Database*



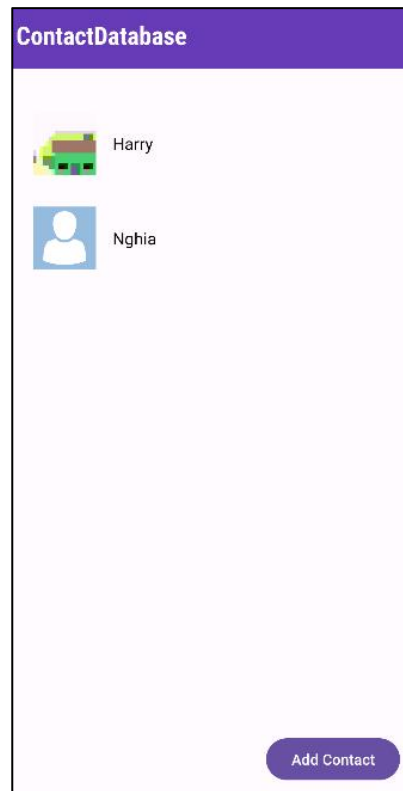



Figure 27: Main Interface with contacts of Contact Database application

- **Add Contact Interface:** The contact interface allows users to add new contacts to a database, including name, date of birth, email, and avatar. If an avatar is not selected, the default avatar is used. The user can save their details by clicking the "Save Details" button or by clicking "Back" to return to the main interface.

Figure 28: Add Contact Interface (Blank)

**ContactDatabase**



Name:

Date of Birth:

Email:

*Figure 29: Add Contact Interface (Full fill available fields)*

- **Update Contact Interface:** The update contact interface is a tool that updates available contacts in a database. It can only be accessed from the Main Interface. When a contact clicks, it passes the information to the interface, allowing users to update the new information. The interface automatically fills all the information from the Main Interface, and users can replace existing fields with desired ones. Other buttons include "View Details" and "Back".

**ContactDatabase**

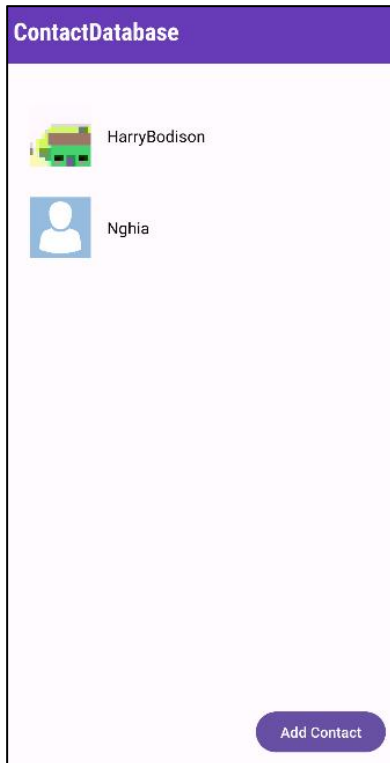


Name:

Date of Birth:

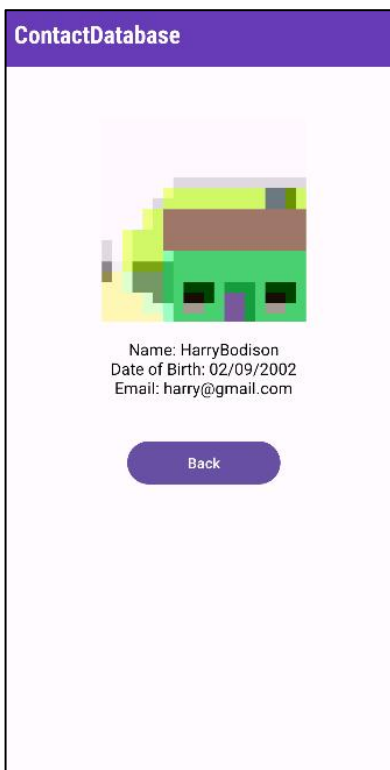
Email:

*Figure 30: Update Contact interface when we access from an available contact (example: Harry contact)*



*Figure 31: Main Interface (After updating the name of contact Harry to HarryBodison)*

- **Details Interface:** The Details Interface displays all contact information, accessible only from the Update Contact Interface. This interface expands the Update Contact Interface, which displays all contact details. The "Back" button returns to the Update Contact Interface, as it is an extension of the Update Contact Interface.



*Figure 32: Details Interface (example: HarryBodison contact)*

### 2.3.2. Code snippets

These are all code source snippets with an explanation of this exercise:

- File **MainActivity.java**: The main activity class for an Android app that handles a contact database is the code that has been provided. In this activity, which displays a list of contacts, users may explore contact information, add new contacts, and edit current connections.

At the beginning of the code is the libraries and variables required for importation and declaration.

```
38
39
40 @Override
41 protected void onCreate(Bundle savedInstanceState) {
42     super.onCreate(savedInstanceState);
43     setContentView(R.layout.activity_main);
44
45     recyclerView = findViewById(R.id.recyclerView);
46     recyclerView.setLayoutManager(new LinearLayoutManager(context, this));
47
48     dbHelper = new DatabaseHelper(context, this);
49     noContactTextView = findViewById(R.id.noContactTextView);
50     addContactButton = findViewById(R.id.addContactButton);
51
52     addContactLauncher = registerForActivityResult(new ActivityResultContracts.StartActivityForResult(),
53         result -> {
54             if (result.getResultCode() == RESULT_OK) {
55                 setupRecyclerViewAdapter();
56             }
57         });
58
59     updateContactLauncher = registerForActivityResult(new ActivityResultContracts.StartActivityForResult(),
60         result -> {
61             if (result.getResultCode() == RESULT_OK) {
62                 setupRecyclerViewAdapter();
63             }
64         });
65
66     addContactButton.setOnClickListener(view -> openAddContactActivity());
67
68     setupRecyclerViewAdapter();
69 }
```

Figure 33: The code snippet of MainActivity.java (code line 38-68) in Exercise 3

The following explains the interior parts: The **onCreate** method sets up the main screen of an app, configuring the **RecyclerView** to display a list of contacts using a **LinearLayoutManager**. A **DatabaseHelper** is created to manage the contact database. Two UI elements, **noContactTextView** and **addContactButton**, are linked for a user-friendly experience. Activity Result Launchers, **addContactLauncher** and **updateContactLauncher**, handle adding and updating contacts. A click listener is attached to the **addContactButton** to open the Add Contact screen. The contact list is initialized using **setupRecyclerViewAdapter()**.

```
69
70
71 3 usages
72 private void setupRecyclerViewAdapter() {
73     Cursor cursor = dbHelper.getContactList();
74     adapter = new AvatarNameAdapter(cursor);
75     recyclerView.setAdapter(adapter);
76
77     updateNoContactMessageVisibility();
78 }
79
80 1 usage
81 private void openAddContactActivity() {
82     Intent intent = new Intent(packageContext, this, AddContactActivity.class);
83     addContactLauncher.launch(intent);
84 }
85
86 1 usage
87 private void updateNoContactMessageVisibility() {
88     if (adapter.getItemCount() == 0) {
89         noContactTextView.setVisibility(View.VISIBLE);
90     } else {
91         noContactTextView.setVisibility(View.GONE);
92     }
93 }
94 }
```

Figure 34: The code snippet of MainActivity.java (code line 69-89) in Exercise 3

The following explains the interior parts: The **setupRecyclerViewAdapter()** method retrieves the database's contact list and creates a custom **AvatarNameAdapter** class to populate the **RecyclerView** with contact data. The **openAddContactActivity()** method opens an intent to add a new contact to the database, and the **updateNoContactMessageVisibility()** method determines whether to display or hide the "No contacts available" message based on the number of items in the **RecyclerView**. This method ensures the user interface remains informative and responsive to changes in the contact list.

```

90
91 3 usages
92 public class AvatarNameAdapter extends RecyclerView.Adapter<AvatarNameAdapter.ViewHolder> {
93
94     11 usages
95     private final Cursor cursor;
96
97     1 usage
98     public AvatarNameAdapter(Cursor cursor) { this.cursor = cursor; }
99
100     @NonNull
101     @Override
102     public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
103         View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.item, parent, attachToRoot: false);
104         return new ViewHolder(view);
105     }

```

Figure 35: The code snippet of MainActivity.java (code line 90-105) in Exercise 3

The following explains the interior parts: The **AvatarNameAdapter** class is a method that populates data from a database cursor into a **RecyclerView**. It takes a **Cursor** object representing a list of contacts from the database. The **onCreateViewHolder** method inflates the layout for a single item, while the **onBindViewHolder** handles binding data to views for each item. Inside **onBindViewHolder** checks the cursor's validity, moves it to the correct position, extracts contact information, displays the user's avatar image, and sets the contact's name to the corresponding **TextView**. The **getItemCount** method determines the total number of items in the **RecyclerView**.

```

106
107 @Override
108 public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
109     if (cursor != null && cursor.moveToPosition(position)) {
110         @SuppressWarnings("Range") String avatarUri = cursor.getString(cursor.getColumnIndex(s: "picture_uri"));
111         @SuppressWarnings("Range") String name = cursor.getString(cursor.getColumnIndex(s: "name"));
112         @SuppressWarnings("Range") long contactId = cursor.getLong(cursor.getColumnIndex(s: "_id"));
113
114         if (!TextUtils.isEmpty(avatarUri)) {
115             Bitmap userBitmap = loadImageFromStorage(avatarUri);
116             if (userBitmap != null) {
117                 holder.avatarImageView.setImageBitmap(userBitmap);
118             }
119         } else {
120             holder.avatarImageView.setImageResource(R.drawable.placeholder_image);
121         }
122
123         holder.nameTextView.setText(name);
124         holder.itemView.setOnClickListener(view -> openUpdateContactActivity(contactId));
125     }
126 }
127
128 @Override
129 public int getItemCount() {
130     return cursor != null ? cursor.getCount() : 0;
131 }

```

Figure 36: The code snippet of MainActivity.java (code line 106-131) in Exercise 3

The following explains the interior parts: The **onBindViewHolder** method is used to perform data binding for each item in the **RecyclerView**, ensuring correct contact data is loaded and displayed. It extracts contact information from the cursor, including avatar URI, name, and contact ID. The **@SuppressWarnings("Range")**

annotations suppress potential lint warnings. The user's avatar image is loaded from storage using the **loadImageFromStorage** method, and a default placeholder image is set if the URI is empty. A click listener is set for the item view, triggering the **openUpdateContactActivity** method when a contact item is clicked. The **getItemCount** method calculates and returns the total number of items in the **RecyclerView**, ensuring efficient contact data loading and display.

```

132
133 4 usages
    public class ViewHolder extends RecyclerView.ViewHolder {
134      3 usages
        ImageView avatarImageView;
135      2 usages
        TextView nameTextView;
136
137      1 usage
        public ViewHolder(@NonNull View itemView) {
138          super(itemView);
139          avatarImageView = itemView.findViewById(R.id.avatarImageView);
140          nameTextView = itemView.findViewById(R.id.nameTextView);
141      }
142  }
143
144  1 usage
    private Bitmap loadImageFromStorage(String filePath) {
145      try {
146          File file = new File(filePath);
147          if (file.exists()) {
148              return BitmapFactory.decodeStream(new FileInputStream(file));
149          }
150      } catch (FileNotFoundException e) {
151          e.printStackTrace();
152      }
153      return null;
154  }

```

Figure 37: The code snippet of MainActivity.java (code line 132-154) in Exercise 3

The following is an explanation of the interior parts: The **ViewHolder** class defines the structure of each item view in the **RecyclerView**, extending **RecyclerView.ViewHolder**. It includes two member variables: **avatarImageView** and **nameTextView**, which reference the **ImageView** for displaying avatars and the **TextView** for displaying contact names in each item view. The constructor initializes these member variables using their respective IDs, ensuring easy access and manipulation. The **loadImageFromStorage** method loads a **Bitmap** image from a specified file path, using a try-catch block to handle potential **FileNotFoundException**s. If an exception occurs, the method returns null to indicate that the image loading has failed. This method is used within the **onBindViewHolder** method of the **AvatarNameAdapter** to load and set the user's avatar image when binding data to each item in the **RecyclerView**, ensuring efficient and graceful display of contact avatars in the user interface.

```

155
156 1 usage
    private void openUpdateContactActivity(long contactId) {
157      Intent intent = new Intent( packageContext: this, UpdateContactActivity.class);
158      intent.putExtra( name: "contact_id", contactId);
159      updateContactLauncher.launch(intent);
160  }
161

```

Figure 38: The code snippet of MainActivity.java (code line 155-end) in Exercise 3

The following explains the interior parts: The **openUpdateContactActivity** method initiates a transition from the current **MainActivity** to the **UpdateContactActivity** when a user clicks on a contact item in the **RecyclerView**. A new intent is created, specifying the target activity and attaching **contactId** for display and updating. The **updateContactLauncher** is used to execute the transition, triggering the opening of the **UpdateContactActivity** with selected contact data.



- File **AddContactActivity.java**: The ContactDatabase Android app uses this code. It is a contact management tool that displays contact names and avatars on a RecyclerView. At the beginning of the code are the libraries and variables required for importation and declaration.

```

29     private final ActivityResultLauncher<String> requestPermissionLauncher =
30         registerForActivityResult(new ActivityResultContracts.RequestPermission(), isGranted -> {
31             if (!isGranted) {
32                 Toast.makeText(context, this, text: "Permission denied. You cannot access photos.", Toast.LENGTH_SHORT).show();
33             }
34         });
35
36     private final ActivityResultLauncher<Intent> pickImageLauncher =
37         registerForActivityResult(new ActivityResultContracts.StartActivityForResult(), result -> {
38             if (result.getResultCode() == RESULT_OK && result.getData() != null) {
39                 Intent data = result.getData();
40                 selectedImageUri = data.getData();
41                 imageView.setImageURI(selectedImageUri);
42             }
43         });

```

Figure 39: The code snippet of AddContactActivity.java (code line 29-42) in Exercise 3

The following explains the interior parts: This code snippet defines two **ActivityResultLaunchers**, **requestPermissionLauncher** and **pickImageLauncher**, to handle specific actions in an activity. **RequestPermissionLauncher** requests runtime permissions, specifically accessing photos, using the **ActivityResultContracts.RequestPermission** contract. If denied, a toast message is displayed. **PickImageLauncher** initiates the image picker activity using the **ActivityResultContracts.StartActivityForResult** contract. These launchers improve user experience by organizing permission requests and activity results.

```

43     @RequiresApi(api = Build.VERSION_CODES.TIRAMISU)
44     @Override
45     protected void onCreate(Bundle savedInstanceState) {
46         super.onCreate(savedInstanceState);
47         setContentView(R.layout.activity_add_contact);
48         dbHelper = new DatabaseHelper(context, this);
49         nameEditText = findViewById(R.id.textInputEditText1);
50         dobEditText = findViewById(R.id.editTextDate1);
51         emailEditText = findViewById(R.id.editTextTextEmailAddress1);
52         imageView = findViewById(R.id.imageView1);
53         imageView.setOnClickListener(view -> openImagePicker());
54         if (!checkPermissions()) {
55             requestPermissions();
56         }
57         Button saveButton = findViewById(R.id.buttonSave);
58         Button backButton = findViewById(R.id.buttonBack);
59         backButton.setOnClickListener(view -> backToMain());
60         saveButton.setOnClickListener(view -> saveUserData());
61     }

```

Figure 40: The code snippet of AddContactActivity.java (code line 43-61) in Exercise 3

The following explains the interior parts: The **onCreate** method is a crucial component in managing the user interface and interactions when adding a new contact to an application. It initializes the content view, database, **EditText** fields, and an **ImageView** for the new contact's avatar. The **checkPermissions** method checks if the app has the necessary permissions to access photos and if not, the **requestPermissions** method is used. Two buttons, **saveButton** and **backButton**, are initialized for saving the new contact's information and returning to the main activity. The **onCreate** method sets up the initial state of the **AddContactActivity**, including layout, input fields, image selection, permission handling, and button interactions.

```

62
63 1 usage
64  @RequiresApi(api = Build.VERSION_CODES.TIRAMISU)
65  private boolean checkPermissions() {
66      String[] permissions = {
67          android.Manifest.permission.READ_MEDIA_IMAGES
68      };
69      for (String permission : permissions) {
70          if (ContextCompat.checkSelfPermission( context: this, permission) != PackageManager.PERMISSION_GRANTED) {
71              return false;
72          }
73      }
74      return true;
75  }
76
77 1 usage
78  @RequiresApi(api = Build.VERSION_CODES.TIRAMISU)
79  private void requestPermissions() {
80      requestPermissionLauncher.launch(android.Manifest.permission.READ_MEDIA_IMAGES);
81  }
82
83 1 usage
84  @RequiresApi(api = Build.VERSION_CODES.TIRAMISU)
85  private void openImagePicker() {
86      if (ContextCompat.checkSelfPermission( context: this, android.Manifest.permission.READ_MEDIA_IMAGES)
87          == PackageManager.PERMISSION_GRANTED) {
88          launchImagePicker();
89      } else {
90          requestPermissionLauncher.launch(Manifest.permission.READ_MEDIA_IMAGES);
91      }
92  }

```

Figure 41: The code snippet of AddContactActivity.java (code line 62-89) in Exercise 3

The following explains the interior parts: This code snippet manages permission management for accessing media images. It uses functions to check if the app has the necessary permissions, such as **android.Manifest.permission.READ\_MEDIA\_IMAGES**, and **requestPermissions**. If not granted, it returns false. The **requestPermissions** function requests the permission, initiating the Android system's permission dialog. The **openImagePicker** function accesses the image picker functionality, checking if the app already has the permission. If not, it initiates a permission request. These functions work together to ensure the app has the necessary permissions to read media images and enable smooth functionality.

```

90
91 1 usage
92  private void launchImagePicker() {
93      Uri imageUri = MediaStore.Images.Media.INTERNAL_CONTENT_URI;
94      Intent intent = new Intent(Intent.ACTION_PICK, imageUri);
95      pickImageLauncher.launch(intent);
96  }
97
98  @Override
99  protected void onActivityResult(int requestCode, int resultCode, Intent data) {
100      super.onActivityResult(requestCode, resultCode, data);
101      if (requestCode == IMAGE_PICKER_REQUEST_CODE && resultCode == RESULT_OK && data != null) {
102          selectedImageUri = data.getData();
103          imageView.setImageURI(selectedImageUri);
104      }
105  }
106
107 2 usages
108  private void backToMain() {
109      Intent intent = new Intent( packageContext: AddContactActivity.this, MainActivity.class);
110      startActivity(intent);
111  }

```

Figure 42: The code snippet of AddContactActivity.java (code line 90-109) in Exercise 3

The explanation of this part: The **launchImagePicker** function is responsible for launching an image picker for the user, setting the image URI to **MediaStore.Images.Media.INTERNAL\_CONTENT\_URI**. It creates an Intent with the action **Intent.ACTION\_PICK** and the specified image URI, and launches the image picker using **pickImageLauncher.launch(intent)**. The **onActivityResult** method handles the result of the image picker, checking if the request code matches the **IMAGE\_PICKER\_REQUEST\_CODE**. If a valid URI is obtained, it is set to the **imageView** for display. The **backToMain** function navigates back to the main activity.



```

110
111     1 usage
112     private void saveUserData() {
113         String name = nameEditText.getText().toString().trim();
114         String dob = dobEditText.getText().toString().trim();
115         String email = emailEditText.getText().toString().trim();
116         String pictureUri = (selectedImageUri != null) ? getRealPathFromUri(selectedImageUri) : "";
117         try {
118             if (!name.isEmpty() && !dob.isEmpty() && !email.isEmpty()) {
119                 long newRowId = dbHelper.insertUserData(name, dob, email, pictureUri);
120                 if (newRowId != -1) {
121                     nameEditText.setText("");
122                     dobEditText.setText("");
123                     emailEditText.setText("");
124                     imageView.setImageResource(R.drawable.placeholder_image);
125                     Toast.makeText(context: AddContactActivity.this, text: "Data saved successfully", Toast.LENGTH_SHORT).show();
126                     backToMain();
127                 } else {
128                     Toast.makeText(context: AddContactActivity.this, text: "Failed to save data", Toast.LENGTH_SHORT).show();
129                 }
130             } else {
131                 Toast.makeText(context: AddContactActivity.this, text: "Please enter all fields", Toast.LENGTH_SHORT).show();
132             }
133         } catch (Exception e) {
134             e.printStackTrace();
135             Toast.makeText(context: AddContactActivity.this, text: "Error saving data", Toast.LENGTH_SHORT).show();
136         }
137     }

```

Figure 43: The code snippet of *AddContactActivity.java* (code line 110-136) in Exercise 3

The **SaveUserData** method in **AddContactActivity** saves user data by extracting and trimming user text into fields like name, dob, and email, and checking if a profile picture has been selected. If a picture is selected, the method retrieves the associated file path from the image URI. Within a try-catch block, it checks if mandatory fields are not empty and attempts to save the user's data. If successful, it performs actions like clearing text fields, resetting the profile picture, and navigating the user back to the main activity. If failure occurs, the method prints error details for debugging and displays a Toast message.

```

137
138     16 usages
139     @Override
140     public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
141         super.onRequestPermissionsResult(requestCode, permissions, grantResults);
142         if (requestCode == IMAGE_PICKER_REQUEST_CODE) {
143             if (grantResults.length > 0 && grantResults[0] != PackageManager.PERMISSION_GRANTED) {
144                 Toast.makeText(context: this, text: "Permission denied. You cannot access photos.", Toast.LENGTH_SHORT).show();
145             }
146         }
147     }
148
149     1 usage
150     private String getRealPathFromUri(Uri uri) {
151         String[] projection = {MediaStore.Images.Media.DATA};
152         Cursor cursor = getContentResolver().query(uri, projection, selection: null, selectionArgs: null, sortOrder: null);
153         if (cursor != null && cursor.moveToFirst()) {
154             int columnIndex = cursor.getColumnIndexOrThrow(MediaStore.Images.Media.DATA);
155             String realPath = cursor.getString(columnIndex);
156             cursor.close();
157             return realPath;
158         }
159         return uri.getPath();
160     }

```

Figure 44: The code snippet of *AddContactActivity.java* (code line 137-end) in Exercise 3

The **onRequestPermissionsResult** method is called when the app receives a response regarding a permission request, specifically for accessing photos. It checks if the request code matches the predefined **IMAGE\_PICKER\_REQUEST\_CODE** and examines the **grantResults** array to determine if permission has been granted. If not, a **Toast** message is displayed, indicating denied access. The **getRealPathFromUri** method converts an image object into its corresponding file path, queries the content resolver, defines a projection, retrieves the cursor, extracts the real file path, and returns the real path or the **Uri**'s path if no path is found.

- **Overview Explanation Of UpdateContactActivity:**

Everything are the same with **AddContactActivity** except some of these:

- + The user receives an intent extra called "contact\_id" to update an existing contact's details, which are then inserted into the **EditText** fields and **ImageView**, ensuring the user can view and make modifications to the current data.

```

73      contactId = intent.getLongExtra( name: "contact_id", defaultValue: -1);
74
75      if (contactId != -1) {
76          loadContactDetails();
77      }

```

Figure 44: The code snippet of UpdateContactActivity.java (code line 73-77) in Exercise 3

- + The **loadContactDetails** method retrieves contact data from the database, populating EditText fields and ImageView with the current contact's information for user viewing and editing.

```

82      private void loadContactDetails() {
83          Cursor cursor = dbHelper.getContactById(contactId);
84
85          if (cursor != null && cursor.moveToFirst()) {
86              @SuppressWarnings("Range") String name = cursor.getString(cursor.getColumnIndex( s: "name"));
87              @SuppressWarnings("Range") String dob = cursor.getString(cursor.getColumnIndex( s: "dob"));
88              @SuppressWarnings("Range") String email = cursor.getString(cursor.getColumnIndex( s: "email"));
89              @SuppressWarnings("Range") String pictureUri = cursor.getString(cursor.getColumnIndex( s: "picture_uri"));
90
91              nameEditText.setText(name);
92              dobEditText.setText(dob);
93              emailEditText.setText(email);
94
95              if (!TextUtils.isEmpty(pictureUri)) {
96                  selectedImageUri = Uri.parse(pictureUri);
97                  imageView.setImageURI(selectedImageUri);
98              }
99              cursor.close();
100          }
101      }

```

Figure 44: The code snippet of UpdateContactActivity.java (code line 82-101) in Exercise 3

- + The **updateContact** method updates a contact's database information based on user changes. If successful, the result is "RESULT\_OK", and the updated avatar URI is sent back to the calling activity.

```

138      private void updateContact() {
139          String name = nameEditText.getText().toString().trim();
140          String dob = dobEditText.getText().toString().trim();
141          String email = emailEditText.getText().toString().trim();
142
143          String pictureUri = (selectedImageUri != null) ? getRealPathFromUri(selectedImageUri) : "";
144
145          try {
146              if (!name.isEmpty() && !dob.isEmpty() && !email.isEmpty()) {
147                  int rowsUpdated = dbHelper.updateUserData(contactId, name, dob, email, pictureUri);
148
149                  if (rowsUpdated > 0) {
150                      Toast.makeText( context: UpdateContactActivity.this, text: "Data updated successfully", Toast.LENGTH_SHORT).show();
151                      Intent resultIntent = new Intent();
152                      resultIntent.putExtra( name: "updated_avatar_uri", selectedImageUri.toString());
153                      setResult(RESULT_OK, resultIntent);
154                      backToMain();
155                  } else {
156                      Toast.makeText( context: UpdateContactActivity.this, text: "Failed to update data", Toast.LENGTH_SHORT).show();
157                  }
158              } else {
159                  Toast.makeText( context: UpdateContactActivity.this, text: "Please enter all fields", Toast.LENGTH_SHORT).show();
160              }
161          } catch (Exception e) {
162              e.printStackTrace();
163              Toast.makeText( context: UpdateContactActivity.this, text: "Error updating data", Toast.LENGTH_SHORT).show();
164          }
165      }

```

Figure 44: The code snippet of UpdateContactActivity.java (code line 138-165) in Exercise 3

- + The method **openDetailsActivity** will open **DetailsActivity** for a user to view a contact's complete details, triggered when the user clicks the "View Details" button.
- + These differences work with the method I describe above, changing a little bit of the target. For example, the **openDetailsActivity** is the same as **openUpdateActivity** from **MainActivity**, but the difference is that the target is not **UpdateActivity**; now, the target is **DetailsActivity**.
- File **DetailsActivity.java**: The supplied code is for an Android app's **DetailsActivity**. This activity displays the details for a specific contact, including their name, date of birth (dob), email, and profile picture (if available).

At the beginning of the code is the libraries and variables required for importation and declaration.

```

23  @Override
24  protected void onCreate(Bundle savedInstanceState) {
25      super.onCreate(savedInstanceState);
26      setContentView(R.layout.activity_details);
27      dbHelper = new DatabaseHelper( context: this);
28      Button backButton = findViewById(R.id.backButton);
29      TextView dataEntryTextView = findViewById(R.id.dataEntryTextView);
30      ImageView dataEntryImageView = findViewById(R.id.dataEntryImageView);
31      Intent intent = getIntent();
32      long contactId = intent.getLongExtra( name: "contact_id",  defaultValue: -1);
33      if (contactId != -1) {
34          Cursor cursor = dbHelper.getContactById(contactId);
35          if (cursor != null && cursor.moveToFirst()) {
36              @SuppressWarnings("Range") String name = cursor.getString(cursor.getColumnIndex( s: "name"));
37              @SuppressWarnings("Range") String dob = cursor.getString(cursor.getColumnIndex( s: "dob"));
38              @SuppressWarnings("Range") String email = cursor.getString(cursor.getColumnIndex( s: "email"));
39              @SuppressWarnings("Range") String pictureUri = cursor.getString(cursor.getColumnIndex( s: "picture_uri"));
40              String formattedData = getString(R.string.name_label, name) + "\n"
41                  + getString(R.string.dob_label, dob) + "\n"
42                  + getString(R.string.email_label, email) + "\n\n";
43              dataEntryTextView.setText(formattedData);
44              if (!TextUtils.isEmpty(pictureUri)) {
45                  Bitmap userBitmap = loadImageFromStorage(pictureUri);
46                  if (userBitmap != null) {
47                      dataEntryImageView.setImageBitmap(userBitmap);
48                  }else {
49                      dataEntryImageView.setImageResource(R.drawable.placeholder_image);
50                  }
51              }else{
52                  dataEntryImageView.setImageResource(R.drawable.placeholder_image);
53              }
54              cursor.close();
55          }
56      }
57      backButton.setOnClickListener(view -> backToUpdate(contactId));
58  }

```

Figure 45: The code snippet of **DetailstActivity.java** (code line 23-58) in Exercise 3

The **onCreate** method in the "**DetailsActivity**" configures the user interface to display contact details. It initializes the activity, sets the content view to the "activity\_details" layout, and creates a database helper object to manage interactions with the SQLite database. The activity retrieves the contact ID from the intent, retrieves corresponding contact details, formats them into a readable string, sets it as text content for **dataEntryTextView**, and loads the contact's image.

```

59     private void backToUpdate(long contactId){
60         Intent intent = new Intent( packageContext, this, UpdateContactActivity.class);
61         intent.putExtra( name, "contact_id", contactId);
62         startActivity(intent);
63     }
64
65     1 usage
66     @ private Bitmap loadImageFromStorage(String filePath) {
67         try {
68             File file = new File(filePath);
69             if (file.exists()) {
70                 return BitmapFactory.decodeStream(new FileInputStream(file));
71             }
72         } catch (FileNotFoundException e) {
73             e.printStackTrace();
74         }
75         return null;
76     }

```

Figure 46: The code snippet of *DetailstActivity.java* (code line 59-76) in Exercise 3

The **backToUpdate** method returns to the "**UpdateContactActivity**" with the contact ID as an extra parameter, creating intent and starting the activity when the user clicks the back button in the details view. The **loadImageFromStorage** method loads an image from the device's storage based on the provided file path, displaying the contact's image in the "**dataEntryImageView**" when viewing contact details in the "DetailsActivity".

- File **DatabaseHelper.java**: To manage a local SQLite database in an Android application, the provided code builds a DatabaseHelper class. This database's main objective is to save and retrieve contact information.

At the beginning of the code are the libraries and variables required for importation and declaration.

```

21     public DatabaseHelper(Context context) {
22         super(context, DATABASE_NAME, factory: null, DATABASE_VERSION);
23     }
24
25     @Override
26     @ public void onCreate(SQLiteDatabase db) {
27         String createTableQuery = "CREATE TABLE IF NOT EXISTS " + TABLE_NAME + " (" +
28             COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
29             COLUMN_NAME + " TEXT, " +
30             COLUMN_DOB + " TEXT, " +
31             COLUMN_EMAIL + " TEXT, " +
32             COLUMN_PICTURE_URI + " TEXT)";
33         db.execSQL(createTableQuery);
34     }
35
36     no usages
37     @Override
38     public void onUpgrade(SQLiteDatabase sqLiteDatabase, int i, int i1) {
39         // Handle database upgrade if needed
40     }

```

Figure 47: The code snippet of *DatabaseHelper.java* (code line 21-40) in Exercise 3

The **DatabaseHelper** class manages the SQLite database through a constructor and **onCreate** method. The constructor initializes the database helper and takes a **Context** object as a parameter. The **onCreate** method executes a **SQL CREATE TABLE** query to create a table named **TABLE\_NAME** variables declared with columns for ID, name, date of birth, email, and picture URI. The **onUpgrade** function is not used instead, I will use another one will describe below.



```

41 public long insertUserData(String name, String dob, String email, String pictureUri) {
42     SQLiteDatabase db = this.getWritableDatabase();
43     ContentValues values = new ContentValues();
44     values.put(COLUMN_NAME, name);
45     values.put(COLUMN_DOB, dob);
46     values.put(COLUMN_EMAIL, email);
47     values.put(COLUMN_PICTURE_URI, pictureUri);
48
49     try {
50         long newRowId = db.insert(TABLE_NAME, nullColumnHack, null, values);
51         db.close();
52         return newRowId;
53     } catch (Exception e) {
54         e.printStackTrace();
55         return -1;
56     }
57 }
58
59 1 usage
60 public int updateUserData(long userId, String name, String dob, String email, String pictureUri) {
61     SQLiteDatabase db = this.getWritableDatabase();
62     ContentValues values = new ContentValues();
63     values.put(COLUMN_NAME, name);
64     values.put(COLUMN_DOB, dob);
65     values.put(COLUMN_EMAIL, email);
66     values.put(COLUMN_PICTURE_URI, pictureUri);
67
68     try {
69         int rowsUpdated = db.update(TABLE_NAME, values, whereClause: COLUMN_ID + " = ?",
70             new String[]{String.valueOf(userId)});
71         db.close();
72         return rowsUpdated;
73     } catch (Exception e) {
74         e.printStackTrace();
75         return 0;
76     }
77 }

```

Figure 48: The code snippet of DatabaseHelper.java (code line 41-76) in Exercise 3

The **insertUserData** method inserts a new user's data into a database using four parameters: name, dob, email, and pictureUri. A writable database instance is obtained using **getWritableDatabase()**, and a **ContentValues** object is created to hold the values. A try-catch block handles exceptions. If successful, the ID of the newly inserted row is returned, and the database is closed. The **updateUserData** method updates an existing user's data with five parameters: userId, name, dob, email, and pictureUri. The method creates a **ContentValues** object and attempts to update the database table with the new values, returning the number of successfully updated rows.

```

77 public Cursor getContactById(long userId) {
78     SQLiteDatabase db = this.getReadableDatabase();
79     String[] projection = {
80         COLUMN_ID,
81         COLUMN_NAME,
82         COLUMN_DOB,
83         COLUMN_EMAIL,
84         COLUMN_PICTURE_URI
85     };
86     String selection = COLUMN_ID + " = ?";
87     String[] selectionArgs = {String.valueOf(userId)};
88     return db.query(TABLE_NAME, projection, selection, selectionArgs,
89         null, null, null, null, null);
90 }
91
92 1 usage
93 public Cursor getContactList() {
94     SQLiteDatabase db = this.getReadableDatabase();
95     String[] projection = {
96         COLUMN_ID,
97         COLUMN_NAME,
98         COLUMN_PICTURE_URI
99     };
100     return db.query(TABLE_NAME, projection,
101         selection: null, selectionArgs: null,
102         groupBy: null, having: null, orderBy: null);
103 }

```

Figure 49: The code snippet of DatabaseHelper.java (code line 77-end) in Exercise 3

The **getContactById** method retrieves detailed information about a contact based on their unique user ID. It creates a database instance, defines a projection for desired columns, and filters queries based on the **userId**.

The method returns a **Cursor** with the requested contact details. The **getContactList** method retrieves a list of contacts based on user ID, name, and picture URI, but does not specify a specific selection criteria, making it suitable for a comprehensive list.

- File **activity\_main.xml**: The XML layout for the "Main Activity" interface includes a header, scrollable contacts, an "Add Contact" button, and a hidden message for unavailable contacts.

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:tools="http://schemas.android.com/tools"
4      android:layout_width="match_parent"
5      android:layout_height="match_parent"
6      xmlns:app="http://schemas.android.com/apk/res-auto">
7
8      <TextView
9          android:id="@+id/textView2"
10         android:layout_width="412dp"
11         android:layout_height="60dp"
12         android:background="#673AB7"
13         android:fontFamily="sans-serif-condensed"
14         android:paddingLeft="10sp"
15         android:paddingTop="12sp"
16         android:text="@string/contact_database"
17         android:textAppearance="@style/TextAppearance.AppCompat.Body1"
18         android:textColor="@color/white"
19         android:textSize="24sp"
20         android:textStyle="bold"
21         app:layout_constraintBottom_toBottomOf="parent"
22         app:layout_constraintEnd_toEndOf="parent"
23         app:layout_constraintHorizontal_bias="0.043"
24         app:layout_constraintStart_toStartOf="parent"
25         app:layout_constraintTop_toTopOf="parent"
26         app:layout_constraintVertical_bias="0.0" />
27
28     <androidx.recyclerview.widget.RecyclerView
29         android:id="@+id/recyclerView"
30         android:layout_width="match_parent"
31         android:layout_height="600dp"
32         android:layout_marginTop="65dp"
33         android:padding="16dp"
34         app:layout_constraintTop_toTopOf="parent"
35         tools:layout_editor_absoluteX="1dp" />

```

Figure 50: The code snippet of activity\_main.xml (Code line 1-35) in Exercise 3

- File **item.xml**: The item.xml XML layout is a horizontal LinearLayout for individual items within a horizontal list, arranged horizontally from left to right, displaying images and text side by side.

```

1  <!-- res/layout/horizontal_item.xml -->
2  <LinearLayout
3      xmlns:android="http://schemas.android.com/apk/res/android"
4      android:layout_width="match_parent"
5      android:layout_height="wrap_content"
6      android:orientation="horizontal"
7      android:padding="10dp"
8      android:layout_marginTop="10dp">
9
10     <ImageView
11         android:id="@+id/avatarImageView"
12         android:layout_width="60dp"
13         android:layout_height="60dp"
14         android:scaleType="centerCrop" />
15
16     <TextView
17         android:id="@+id/nameTextView"
18         android:layout_width="wrap_content"
19         android:layout_height="wrap_content"
20         android:layout_gravity="center_vertical"
21         android:layout_marginStart="16dp"
22         android:textSize="16sp"
23         android:textColor="@android:color/black" />
24 </LinearLayout>

```

Figure 52: The code snippet of item.xml in Exercise 3

- File **activity\_add\_contact.xml**: The XML layout for the "AddContactActivity" utilizes **ConstraintLayout** to organize fields for inputting contact information, navigation buttons, and a placeholder for the profile image. This choice of layout facilitates optimal alignment and responsiveness.

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      tools:context=".AddContactActivity">
8
9      <TextView
10         android:id="@+id/textView2"
11         android:layout_width="412dp"
12         android:layout_height="60dp"
13         android:background="#673A87"
14         android:fontFamily="sans-serif-condensed"
15         android:paddingLeft="10sp"
16         android:paddingTop="12sp"
17         android:text="ContactDatabase"
18         android:textAppearance="@style/TextAppearance.AppCompat.Body1"
19         android:textColor="@color/white"
20         android:textSize="24sp"
21         android:textStyle="bold"
22         app:layout_constraintBottom_toBottomOf="parent"
23         app:layout_constraintEnd_toEndOf="parent"
24         app:layout_constraintHorizontal_bias="0.043"
25         app:layout_constraintStart_toStartOf="parent"
26         app:layout_constraintTop_toTopOf="parent"
27         app:layout_constraintVertical_bias="0.0" />
28

```

Figure 54: The code snippet of activity\_add\_contact.xml (code line 1-28) in Exercise 3

- File **activity\_update\_contact.xml**: The "UpdateContactActivity" XML layout incorporates a profile image placeholder, buttons for viewing and saving, fields for modifying contact information, and utilizes **ConstraintLayout** for optimal alignment and responsiveness.

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      tools:context=".UpdateContactActivity">
8
9      <TextView
10         android:id="@+id/textView2"
11         android:layout_width="412dp"
12         android:layout_height="60dp"
13         android:background="#673A87"
14         android:fontFamily="sans-serif-condensed"
15         android:paddingLeft="10sp"
16         android:paddingTop="12sp"
17         android:text="ContactDatabase"
18         android:textAppearance="@style/TextAppearance.AppCompat.Body1"
19         android:textColor="@color/white"
20         android:textSize="24sp"
21         android:textStyle="bold"
22         app:layout_constraintBottom_toBottomOf="parent"
23         app:layout_constraintEnd_toEndOf="parent"
24         app:layout_constraintHorizontal_bias="0.043"
25         app:layout_constraintStart_toStartOf="parent"
26         app:layout_constraintTop_toTopOf="parent"
27         app:layout_constraintVertical_bias="0.0" />
28

```

Figure 55: The code snippet of activity\_update\_contact.xml (code line 1-28) in Exercise 3

- File **activity\_details.xml**: This layout shows contact information in a vertically aligned **LinearLayout** that includes a header, an image, text, and navigation buttons.

```
1 <!-- res/layout/activity_details.xml -->
2
3 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
4     android:layout_width="match_parent"
5     android:layout_height="match_parent"
6     xmlns:app="http://schemas.android.com/apk/res-auto"
7     android:orientation="vertical">
8
9     <TextView
10         android:id="@+id/textView2"
11         android:layout_width="412dp"
12         android:layout_height="60dp"
13         android:background="#673AB7"
14         android:fontFamily="sans-serif-condensed"
15         android:paddingLeft="10sp"
16         android:paddingTop="12sp"
17         android:layout_marginBottom="50dp"
18         android:text="ContactDatabase"
19         android:textAppearance="@style/TextAppearance.AppCompat.Body1"
20         android:textColor="@color/white"
21         android:textSize="24sp"
22         android:textStyle="bold"
23         app:layout_constraintBottom_toBottomOf="parent"
24         app:layout_constraintEnd_toEndOf="parent"
25         app:layout_constraintHorizontal_bias="0.043"
26         app:layout_constraintStart_toStartOf="parent"
27         app:layout_constraintTop_toTopOf="parent"
28         app:layout_constraintVertical_bias="0.0" />
29
```

Figure 56: The code snippet of *activity\_details.xml* (code line 1-29) in Exercise 3



## References

Smyth, N., 2020. *Android Studio 4.0 Development Essentials - Java Edition: Build Android Apps with Android Studio 4. 0 and Java*. Limited 编辑 Birmingham: Packt.