

**A Mobile Application Design and Development**

**Logbook**

Course: **COMP1786**

Submitted by: **Chi Trien Nguyen**

Student ID: **001353546**

Date of Submission: September**, 28th 2023**

**Table Of Content**

[I. INTRODUCTION 3](#_Toc17536)

[II. BASIC INFORMATION 3](#_Toc22305)

[III. EXERCISE ANSWER 3](#_Toc13807)

[1. Screenshots demonstrating what you achieved 3](#_Toc22242)

[2. Code snippets and explanation 7](#_Toc30631)

[References 24](#_Toc5995)

**Table Of Figures**

[Figure 1 Main Interface (No contact). 5](#_Toc31832)

[Figure 2 Main Interface (has contact). 5](#_Toc9927)

[Figure 3 Add Contact Interface (Blank). 6](#_Toc4549)

[Figure 4 Add Contact Interface (Full fill available fields). 6](#_Toc29906)

[Figure 5 Update Contact interface when we access from an available contact (example: Harry contact). 7](#_Toc12331)

[Figure 6 Main Interface (After update name of contact Harry to HarryBodison). 7](#_Toc6051)

[Figure 7 Details Interface (example: HarryBodison contact). 8](#_Toc15767)

[Figure 8 Main Activity (Code line 1 - 25 ). 8](#_Toc1617)

[Figure 9 Main Activity (Code line 26 - 37 ). 9](#_Toc20359)

[Figure 10 Main Activity (Code line 38 - 68). 9](#_Toc5344)

[Figure 11 Main Activity (Code line 69 - 89). 10](#_Toc426)

[Figure 12 Main Activity (Code line 90 - 105). 10](#_Toc8478)

[Figure 13 Main Activity (Code line 106 - 131). 11](#_Toc10728)

[Figure 14 Main Activity (Code line 132 - 154). 11](#_Toc1740)

[Figure 15 Main Activity (Code line 155 - 161) End Main Activity. 12](#_Toc3490)

[Figure 16 Add Contact Activiy (Code line 1 - 21). 12](#_Toc7934)

[Figure 17 Add Contact Activity (Code line 22 - 28). 13](#_Toc31803)

[Figure 18 Add Contact Activity (Code line 29 - 42). 13](#_Toc20165)

[Figure 19 Add Contact Activity (Code line 43 - 61). 13](#_Toc29399)

[Figure 20 Add Contact Activity (Code line 62 - 89). 14](#_Toc26890)

[Figure 21 Add Contact Activity (Code line 90 - 109). 15](#_Toc11618)

[Figure 22 Add Contact Activity (Code line 110 - 136). 15](#_Toc26557)

[Figure 23 Add Contact Activity (Code line 137 - 159) End Add Contact Activity. 16](#_Toc18645)

[Figure 24 Update Contact Activity (Code line 1-30). 17](#_Toc16432)

[Figure 25 Update Contact Activity (Code line 31 - 65). 17](#_Toc23220)

[Figure 26 Update Contact Activity (Code line 66 -101). 18](#_Toc22205)

[Figure 27 Update Contact Activity (Code line 102 - 131). 18](#_Toc8667)

[Figure 28 Update Contact Activity (Code line 132 - 165). 19](#_Toc21935)

[Figure 29 Update Contact Activity (Code line 166 - 197) End Update Contact Activity. 19](#_Toc13435)

[Figure 30 Details Activity (Code line 1 - 22). 20](#_Toc27488)

[Figure 31 Details Activity (Code line 23 -58). 21](#_Toc20320)

[Figure 32 Details Activity (Code line 59 - 76) End Details Activity. 21](#_Toc13214)

[Figure 33 Database Helper (Code line 1-20). 22](#_Toc10869)

[Figure 34 Database Helper (Code line 21 - 40). 22](#_Toc20563)

[Figure 35 Database Helper (Code line 41 - 76). 23](#_Toc30345)

[Figure 36 Database Helper (Code line 77 - 99) End Database Helper. 24](#_Toc30053)

# INTRODUCTION

# BASIC INFORMATION

|  |  |
| --- | --- |
| 1. Student Name & ID | Chi Trien Nguyen - 001353546 |
| 1. Who did you work with? | Phat Dat Truong - 001353261  Minh Nghia Nguyen - 001353656 |
| 1. Which Exercise is this? Tick as appropriate. | * Exercise 1 □ * Exercise 2 □ * Exercise 3 ☑ |
| 1. How well did you complete the exercise? Tick as appropriate. | * I tried but couldn’t complete it □ * I did it, but I feel I should have done better □ * I did everything that was asked ☑ * I did more than was asked for □ |
| 1. Briefly explain your answer to question 4 | I have met all the requirements outlined in the logbook for this task. |

# EXERCISE ANSWER

## Screenshots demonstrating what you achieved

This is Contact Database application, user can use it to store their contact information such as name, date of birth, email, and their 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 right leads to the Add Contact Interface, and clicking on an available contact leads to the Update Contact Interface for updating new information).

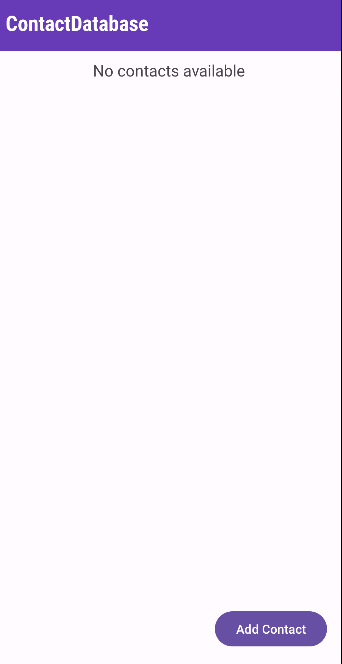


Figure 1 Main Interface (No contact).

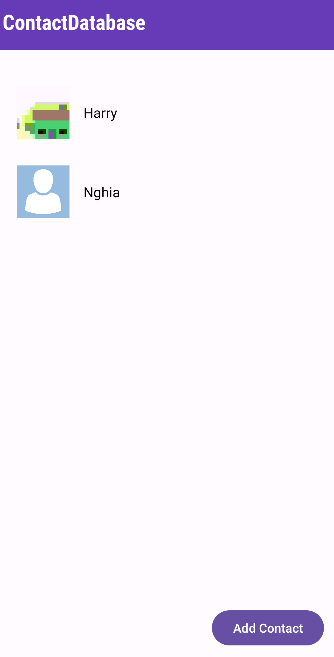


Figure 2 Main Interface (has contact).

* **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.

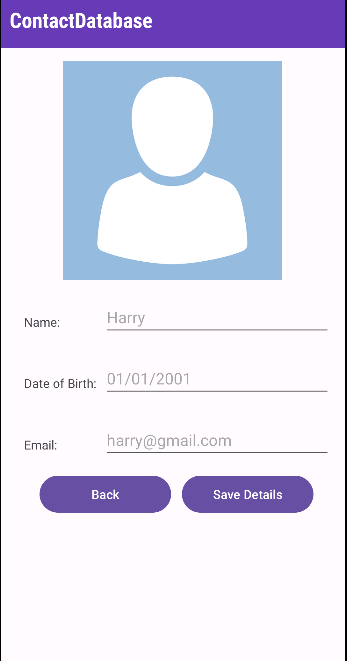


Figure 3 Add Contact Interface (Blank).



Figure 4 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 is clicked, 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".

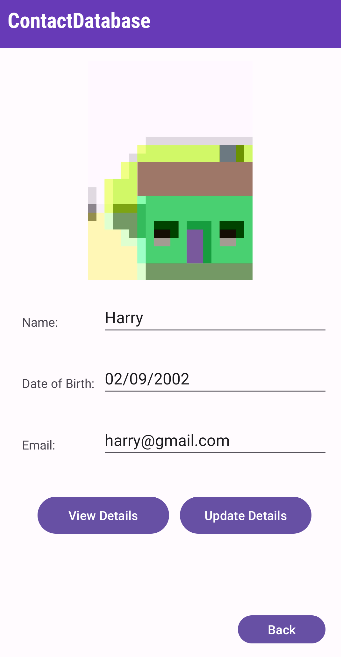


Figure 5 Update Contact interface when we access from an available contact (example: Harry contact).

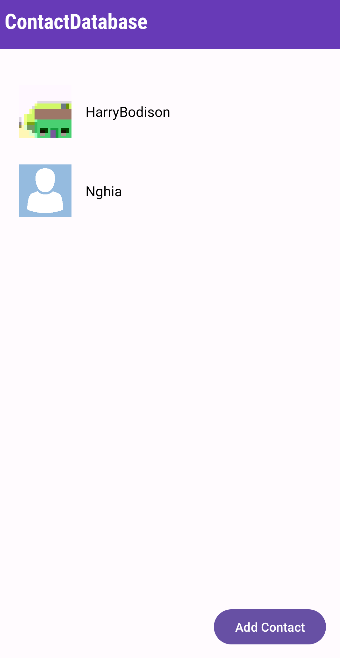


Figure 6 Main Interface (After update name of contact Harry to HarryBodison).

* **Details Interface:** The Details Interface displays all contact information, accessible only from the Update Contact Interface. This interface is an expansion of 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.

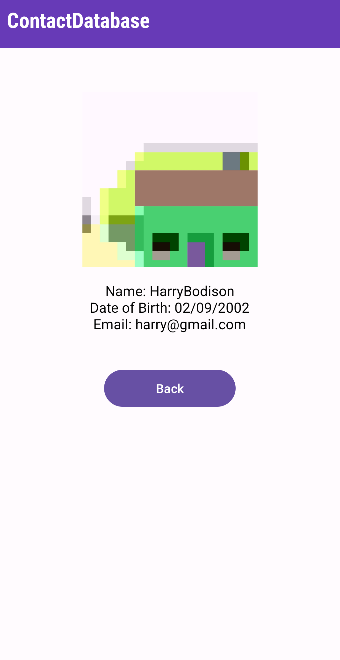


Figure 7 Details Interface (example: HarryBodison contact).

## Code snippets and explanation

* **Main Activity:**

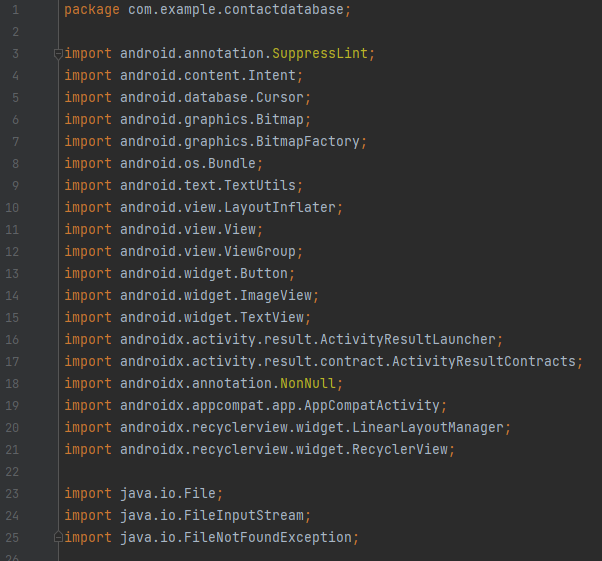


Figure 8 Main Activity (Code line 1 - 25 ).

**Explanation**: Import necessary.

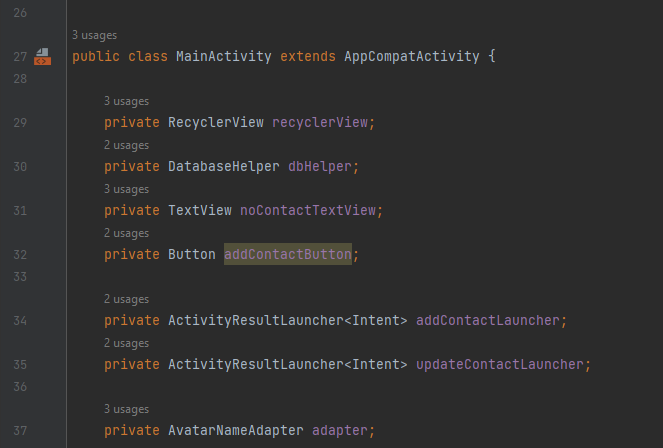


Figure 9 Main Activity (Code line 26 - 37 ).

**Explanation**: Declare variables.



Figure 10 Main Activity (Code line 38 - 68).

**Explanation:** 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 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**().



Figure 11 Main Activity (Code line 69 - 89).

**Explanation:** 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.

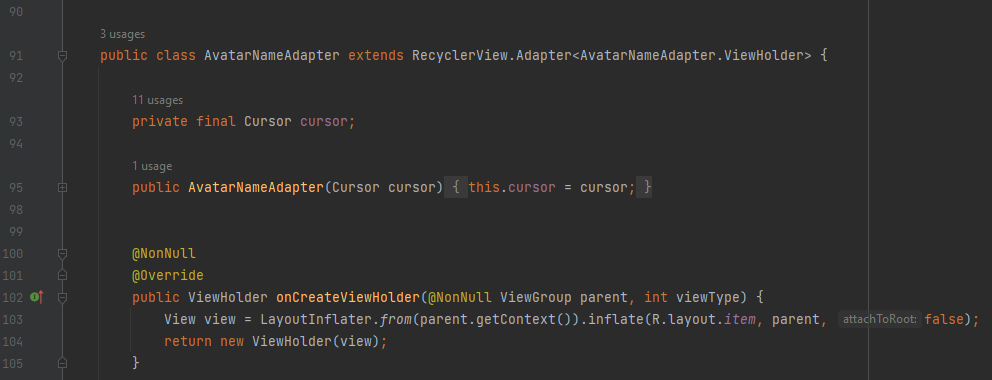


Figure 12 Main Activity (Code line 90 - 105).

**Explanation**: 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**, it 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**.



Figure 13 Main Activity (Code line 106 - 131).

**Explanation**: 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 **@SuppressLint("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.



Figure 14 Main Activity (Code line 132 - 154).

**Explanation**: 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 are used to 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 **FileNotFoundExceptions**. If an exception occurs, the method returns null to indicate 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.



Figure 15 Main Activity (Code line 155 - 161) End Main Activity.

**Explanation:** 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.

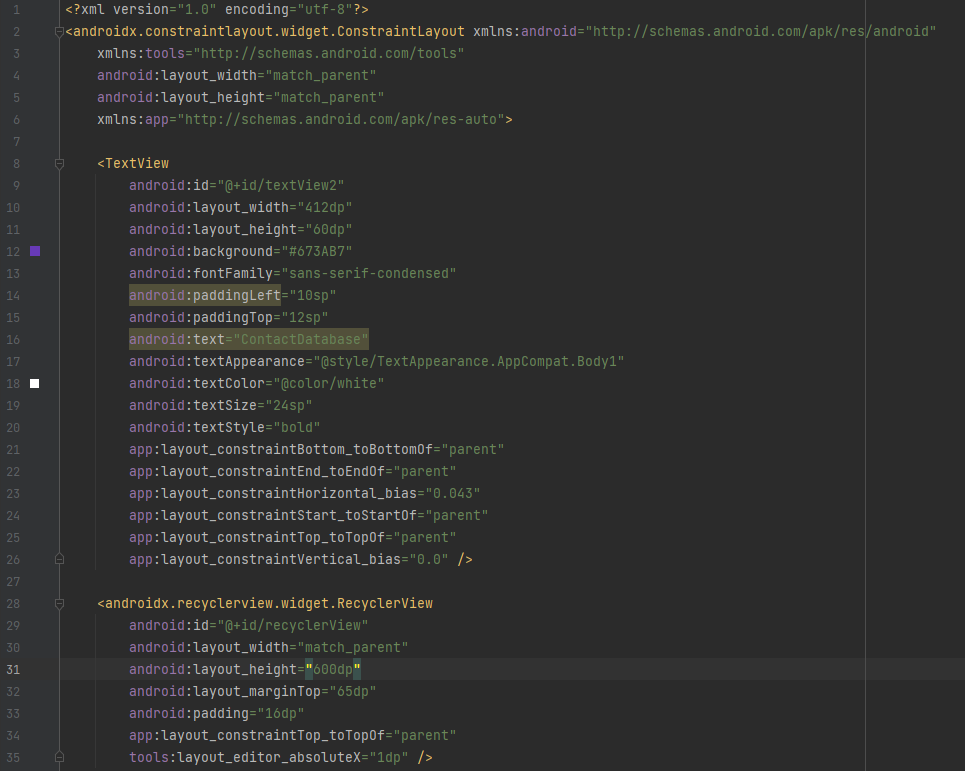


Figure 16 Main Activity XML (Code line 1 - 35) In Exercise 3

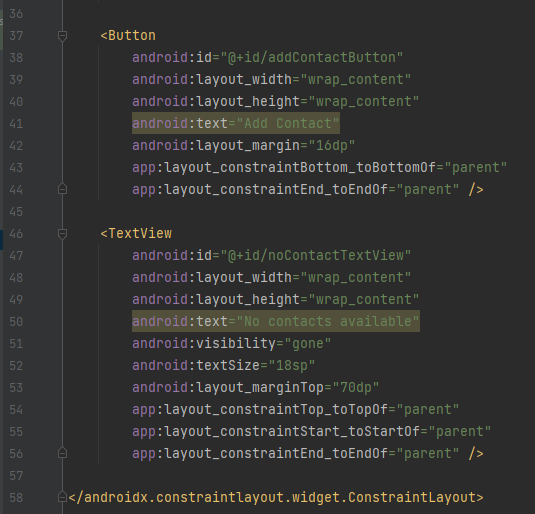


Figure 17 Main Activity XML (Code line 36 - 58) In Exercise 3

**Explanation**: The XML layout for the "Main Activity" interface includes a header, scrollable contacts, an "Add Contact" button, and a hidden message for unavailable contacts.

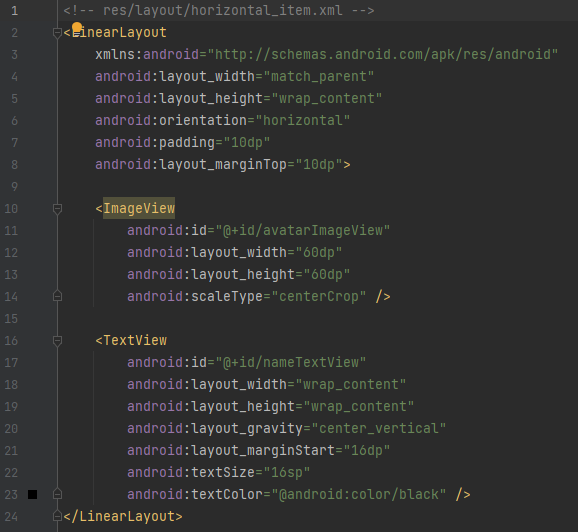


Figure 18 Item XML In Exercise 3

**Explanation:** The horizontal\_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.

* **Add Contact Activity:**

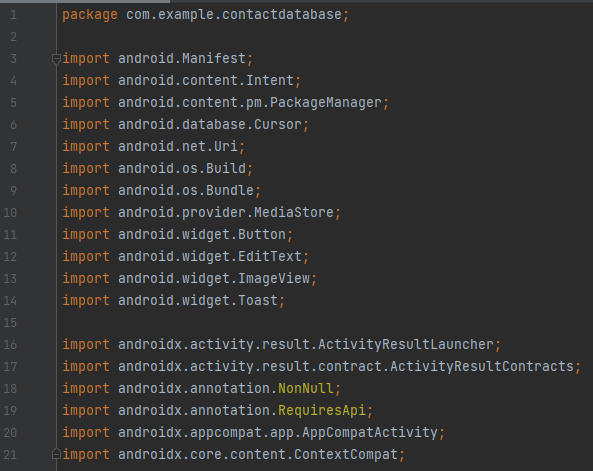


Figure 19 Add Contact Activiy (Code line 1 - 21).

**Explanation**: Import necessary.

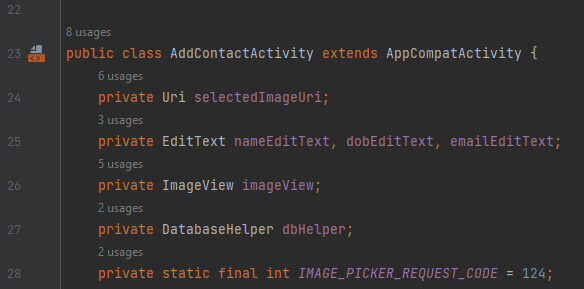


Figure 20 Add Contact Activity (Code line 22 - 28).

**Explanation:** Declare variables.



Figure 21 Add Contact Activity (Code line 29 - 42).

**Explanation:** 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.



Figure 22 Add Contact Activity (Code line 43 - 61).

**Explanation:** 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.

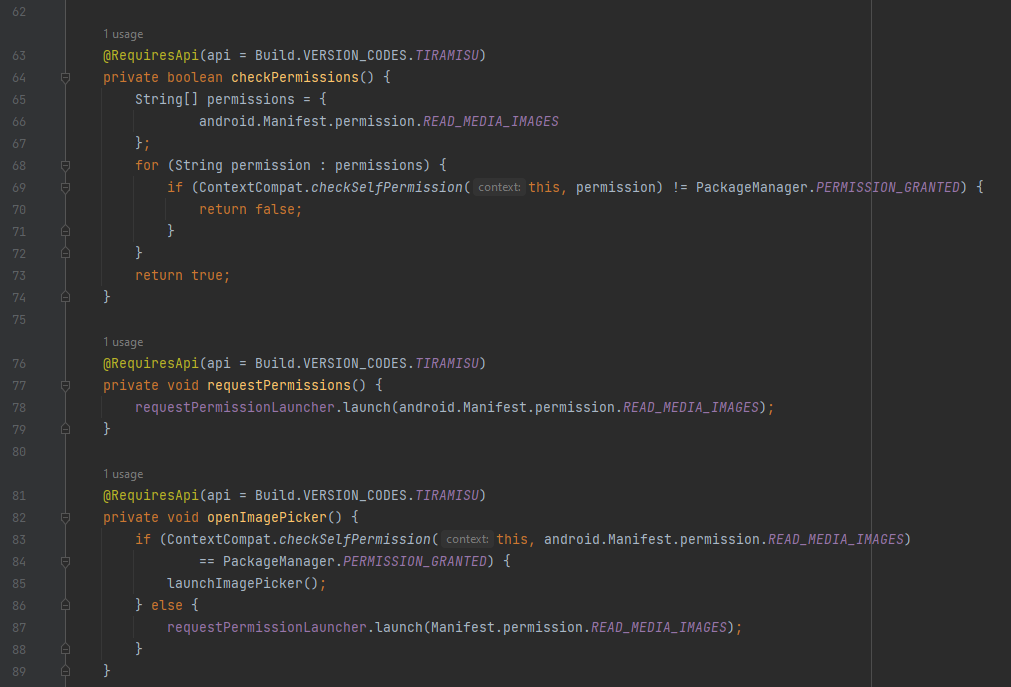


Figure 23 Add Contact Activity (Code line 62 - 89).

**Explanation:** 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.



Figure 24 Add Contact Activity (Code line 90 - 109).

**Explanation:** 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.



Figure 25 Add Contact Activity (Code line 110 - 136).

**Explanation:** 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.

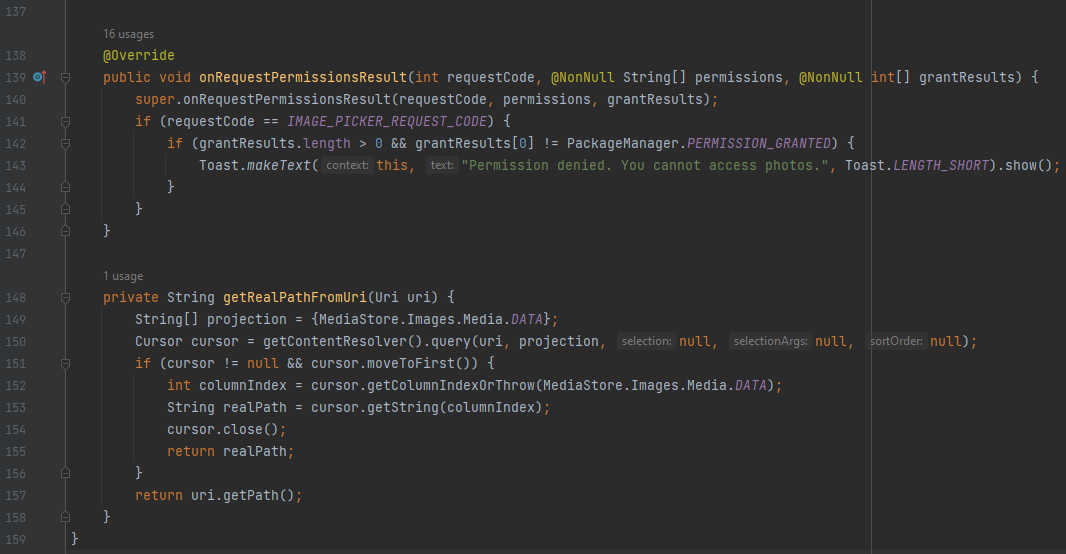


Figure 26 Add Contact Activity (Code line 137 - 159) End Add Contact Activity.

**Explanation:** 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.

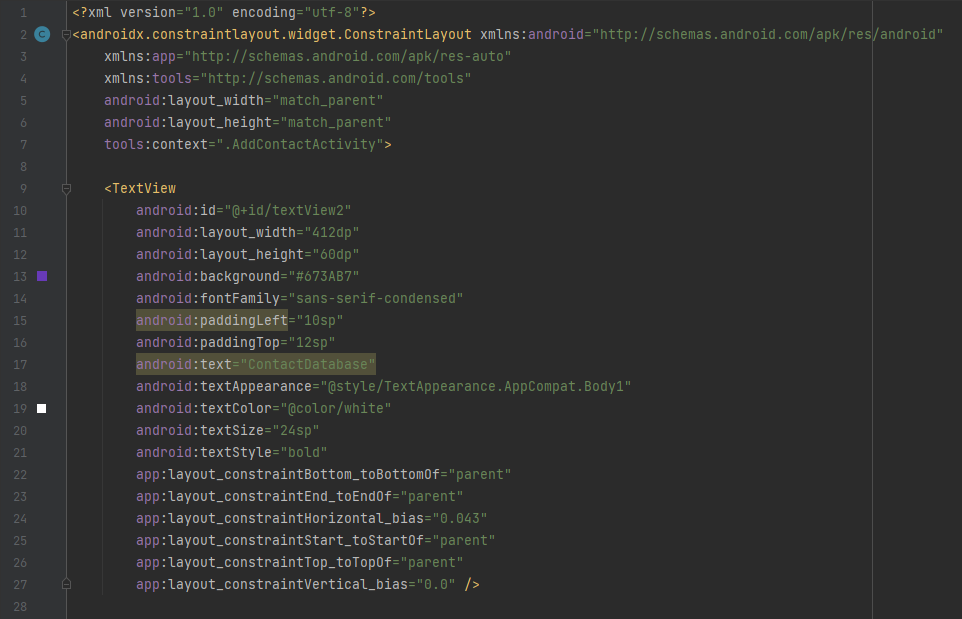


Figure 27 Add Contact Activity XML (Code line 1 - 28) In Exercise 3



Figure 28 Add Contact Activity XML (Code line 29 - 63) In Exercise 3



Figure 29 Add Contact Activity XML (Code line 64 - 98) In Exercise 3

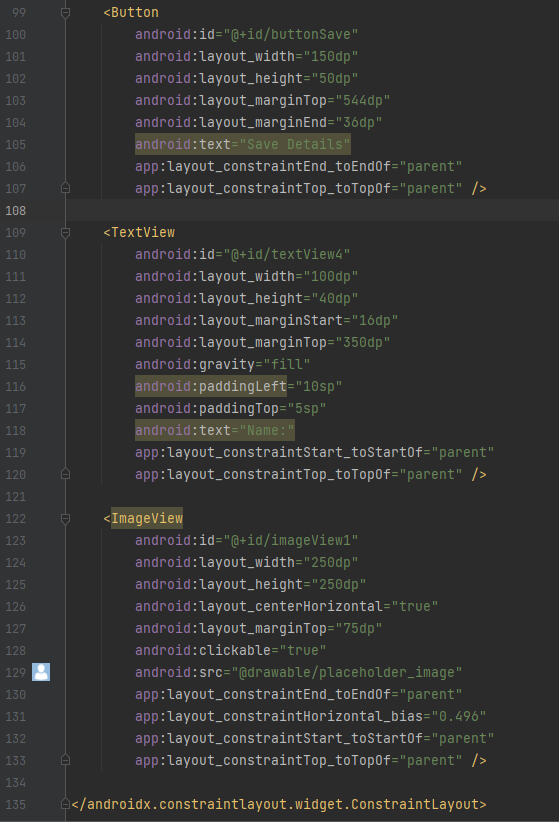


Figure 30 Add Contact Activity XML (Code line 99 -135) In Exercise 3

**Explaination:** The XML layout for "AddContactActivity" includes fields for entering personal information, navigation buttons, a profile image placeholder, and is organized using ConstraintLayout for optimal alignment and responsiveness.

* **Update Contact Activity:**

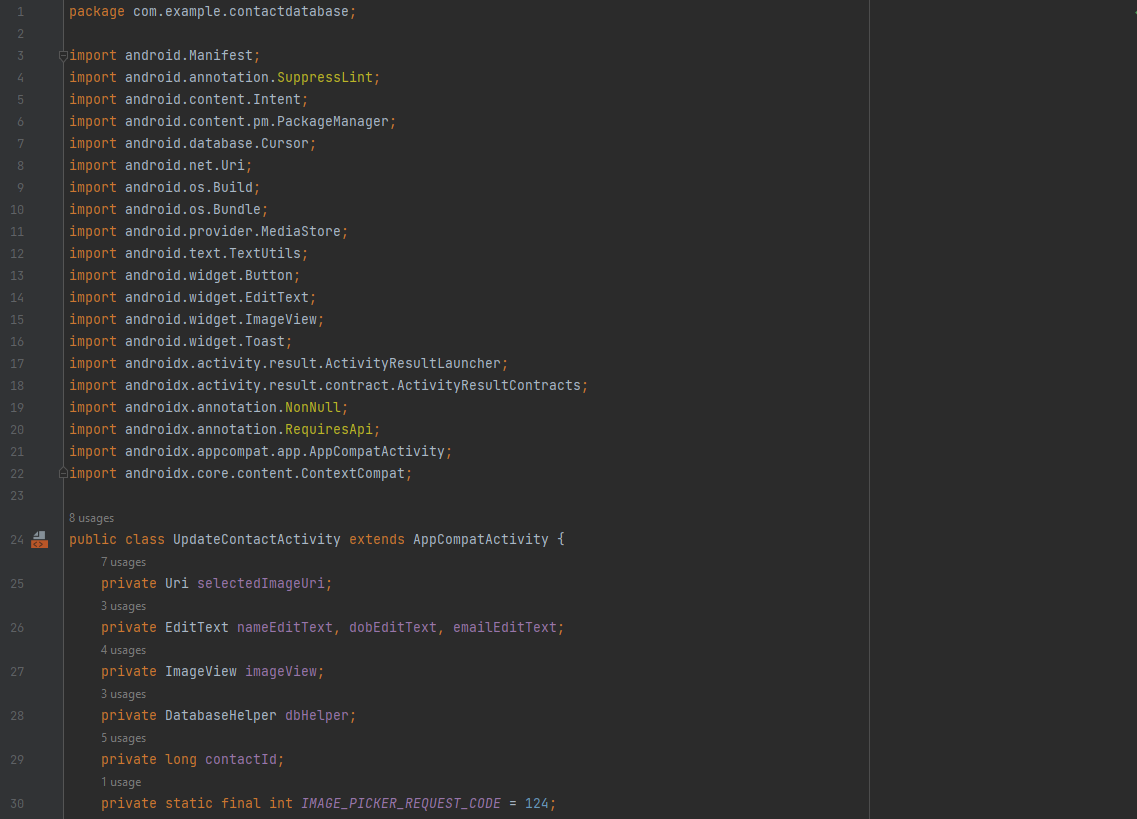


Figure 31 Update Contact Activity (Code line 1-30).

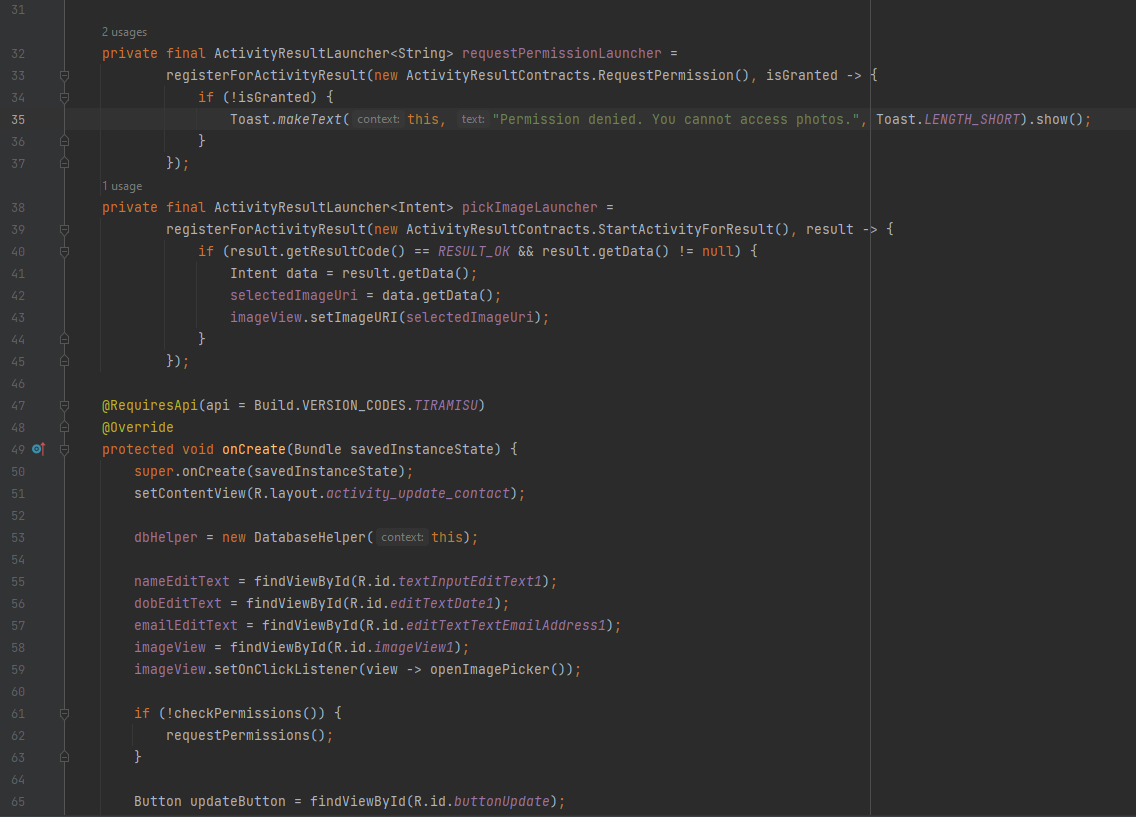


Figure 32 Update Contact Activity (Code line 31 - 65).



Figure 33 Update Contact Activity (Code line 66 -101).



Figure 34 Update Contact Activity (Code line 102 - 131).

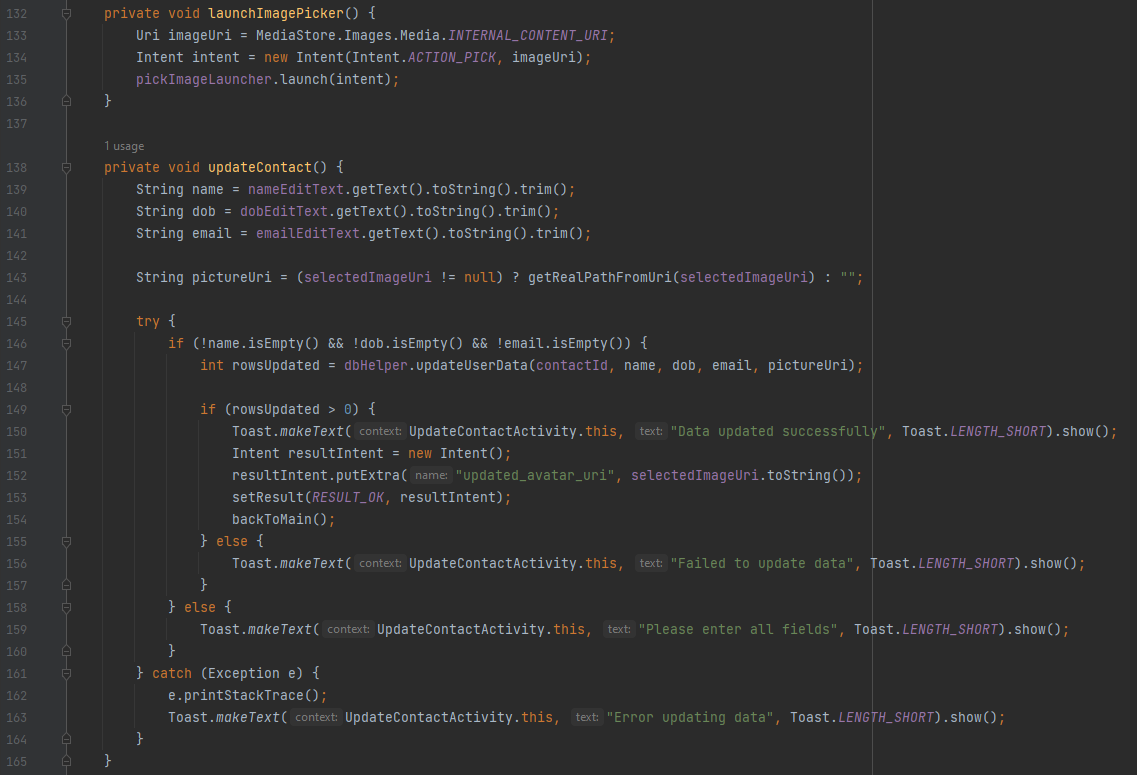


Figure 35 Update Contact Activity (Code line 132 - 165).

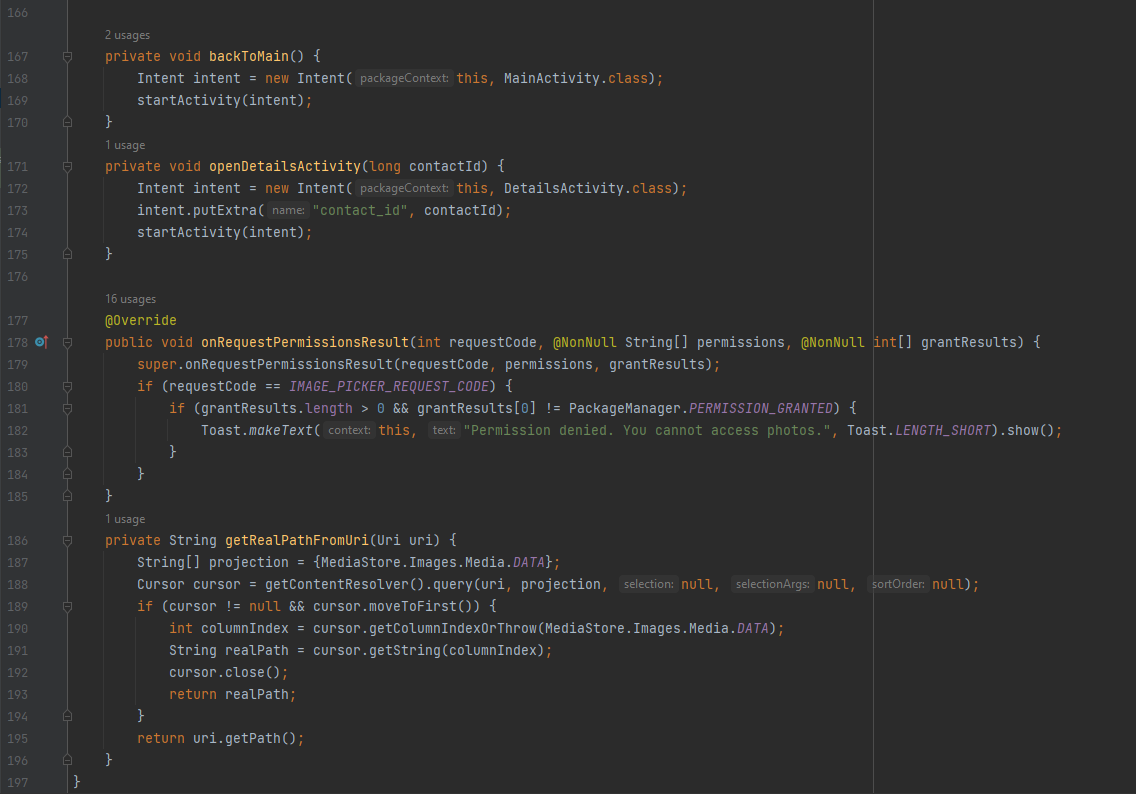


Figure 36 Update Contact Activity (Code line 166 - 197) End Update Contact Activity.

**Overview Explanation**:

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

1. 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.
2. 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.
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.
4. The method opens a "**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 same method I describe above just change a little bit of the target. Example the **openDetailsActivity** are the same with **openUpdateActivity** from **MainActivity** but the differences is the target is not **UpdateActivity** now the target is **DetailsActivity**.

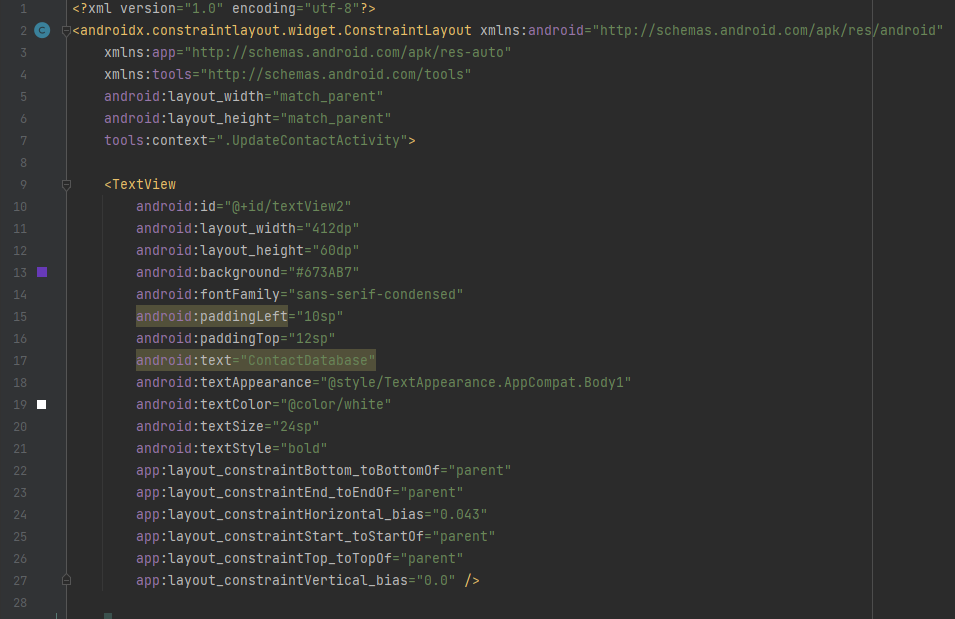


Figure 37 Update Contact Activity XML (Code line 1-28) In Exercise 3

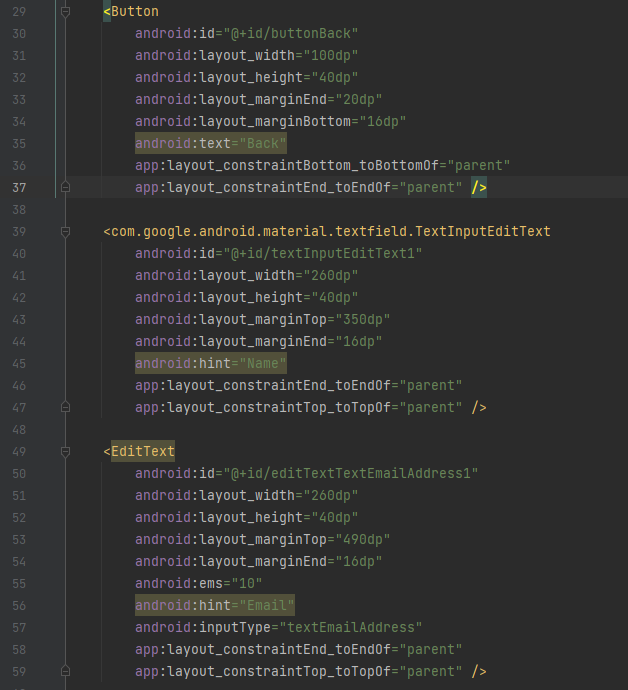


Figure 38 Update Contact Activity XML (Code line 29 - 59) In Exercise 3



Figure 39 Update Contact Activity XML (Code line 60 - 92) In Exercise

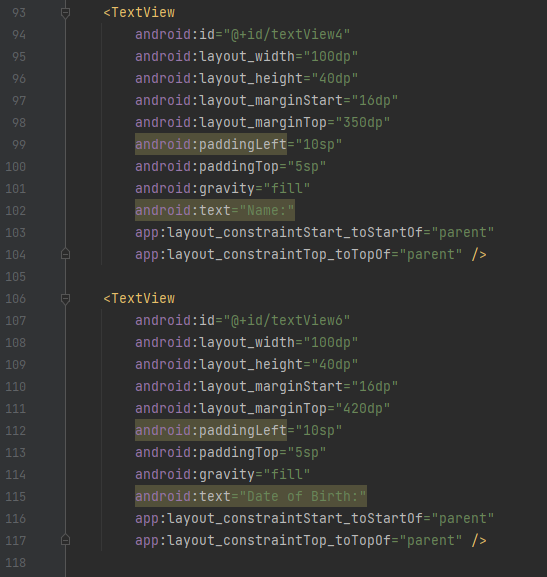


Figure 40 Update Contact Activity XML (Code line 93 - 118) In Exercise 3

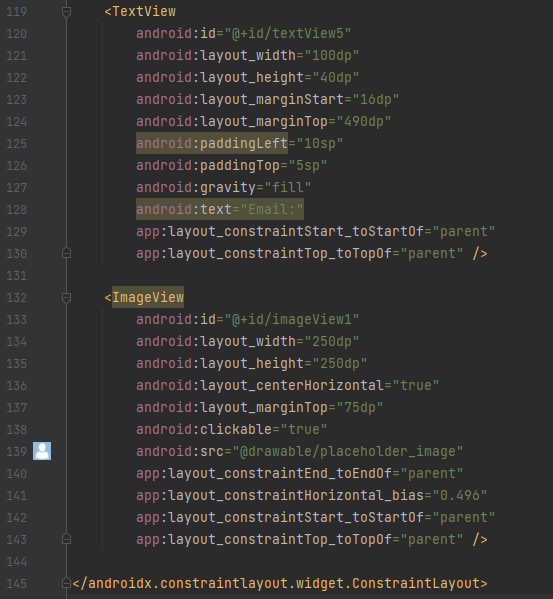


Figure 41 Update Contact Activity (Code line 119 - 145) In Exercise 3

**Explanation:** The XML layout for "UpdateContactActivity" includes fields for updating contact details, buttons for viewing and saving, a profile image placeholder, and ConstraintLayout for optimal alignment and responsiveness.

* **Details Activity:**

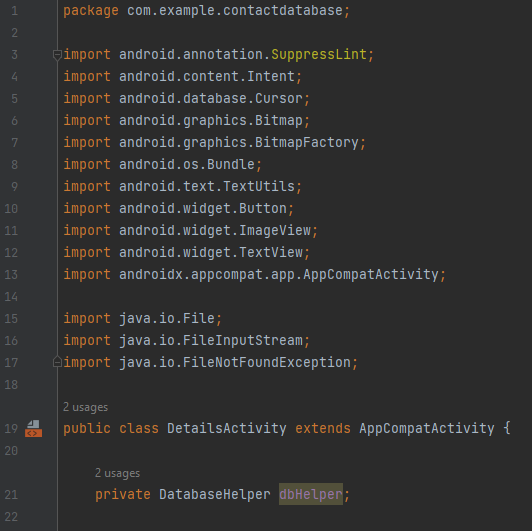


Figure 42 Details Activity (Code line 1 - 22).

**Explanation**: Import necessary and declare variables.

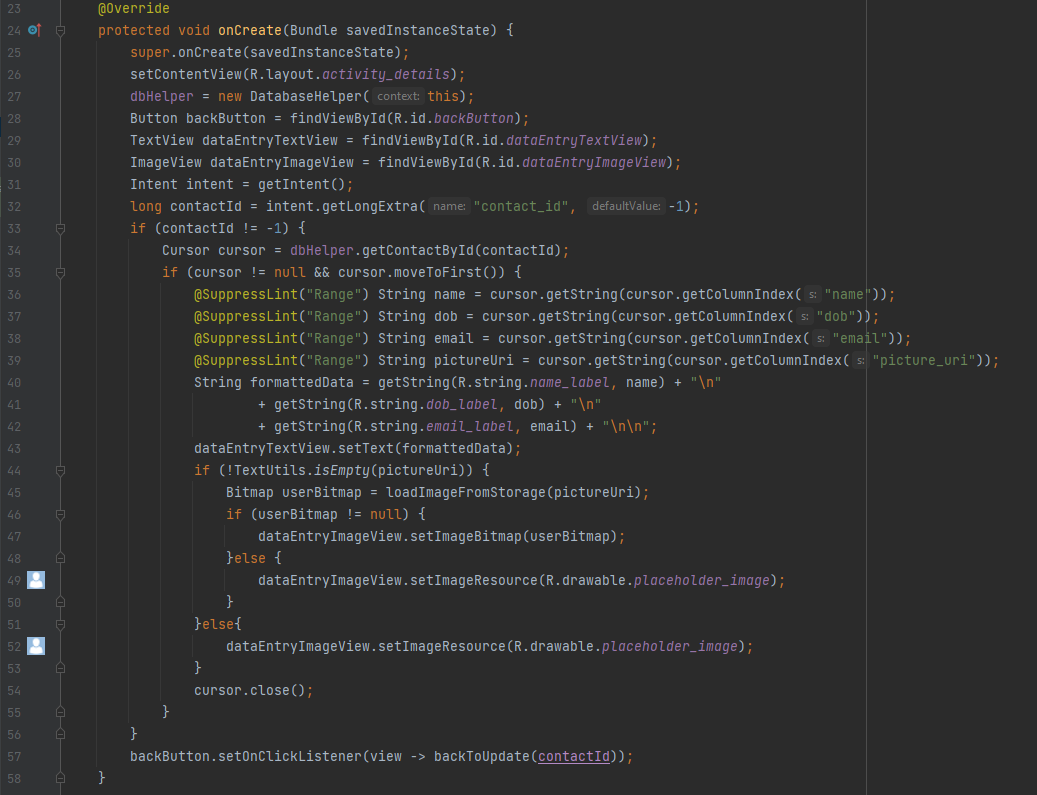


Figure 43 Details Activity (Code line 23 -58).

**Explanation:** 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.

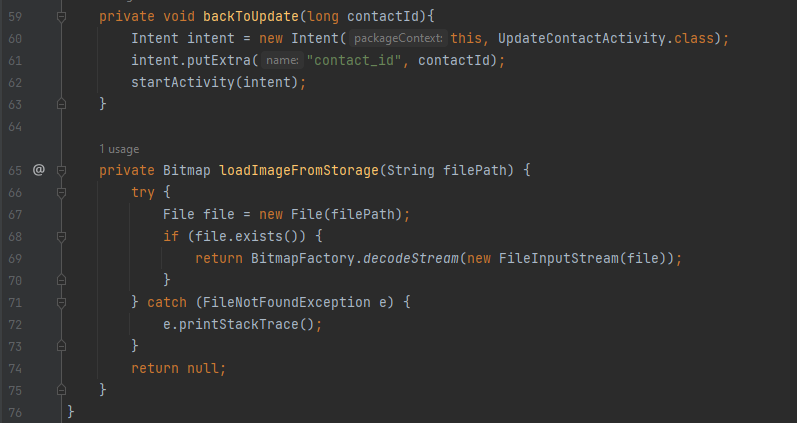


Figure 44 Details Activity (Code line 59 - 76) End Details Activity.

**Explanation:** The **backToUpdate** method returns to the "**UpdateContactActivity**" with the contact ID as an extra parameter, creating an 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".

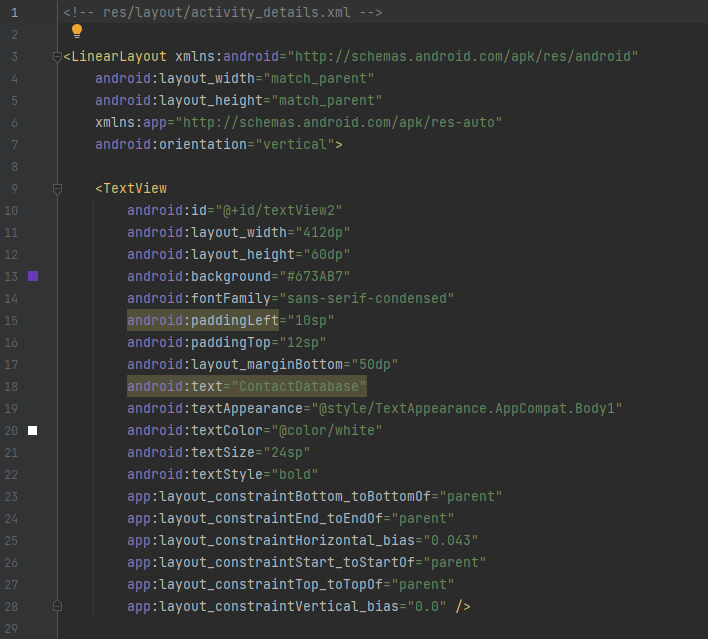


Figure 45 Details Activity XML (Code line 1 - 29) In Exercise 3

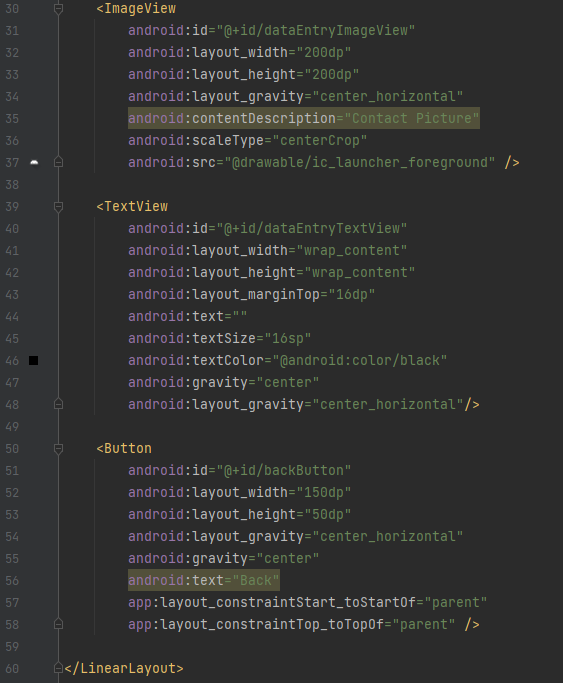


Figure 46 Details Activity XML In Exercise 3

**Explanation**: The activity\_details.xml XML layout displays contact details, including a header, image, text, and navigation buttons, organized vertically using a LinearLayout.

* **Database Helper:**

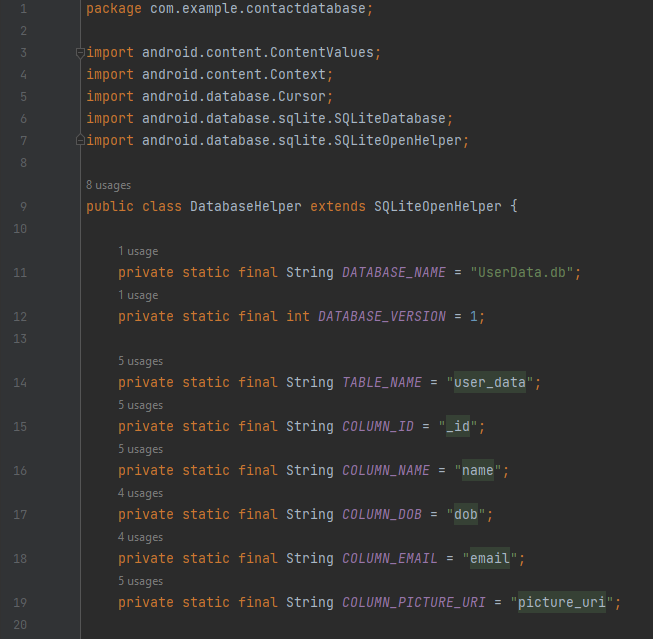


Figure 47 Database Helper (Code line 1-20).

**Explanation:** Import necessary and declare variables.

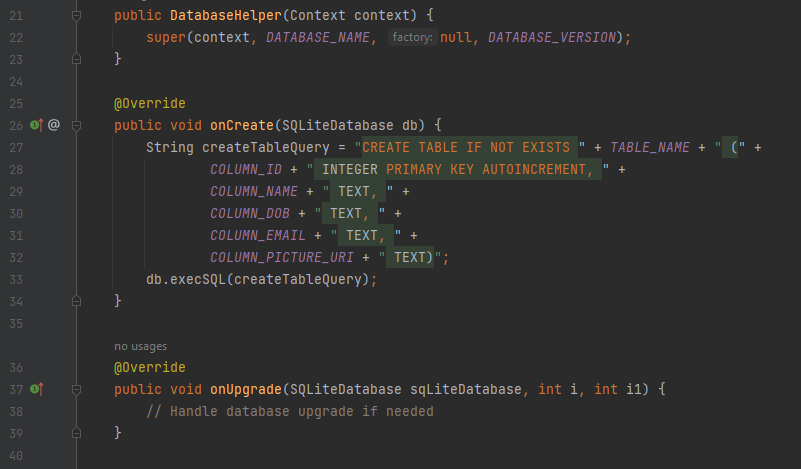


Figure 48 Database Helper (Code line 21 - 40).

**Explanation:** 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.

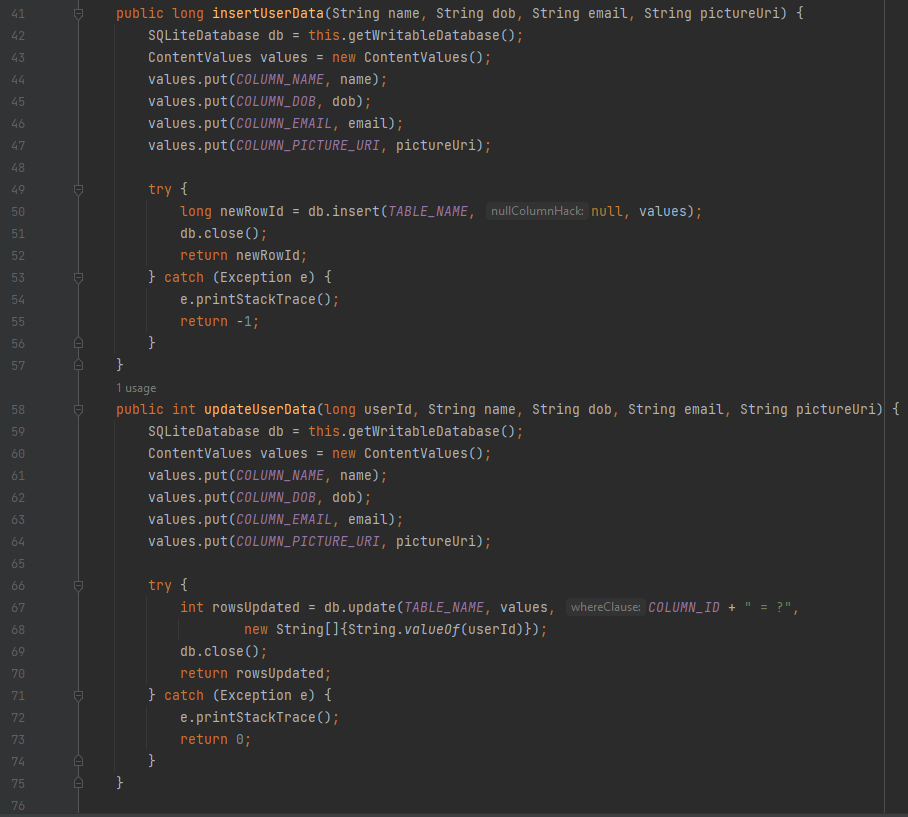


Figure 49 Database Helper (Code line 41 - 76).

**Explanation:** 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.



Figure 50 Database Helper (Code line 77 - 99) End Database Helper.

**Explanation:** 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.

# References