Name: Sean Baker

ID: sbake021

Class Name: Application Development for Smart Devices

Assignment Name: Address App Assignment With

Database

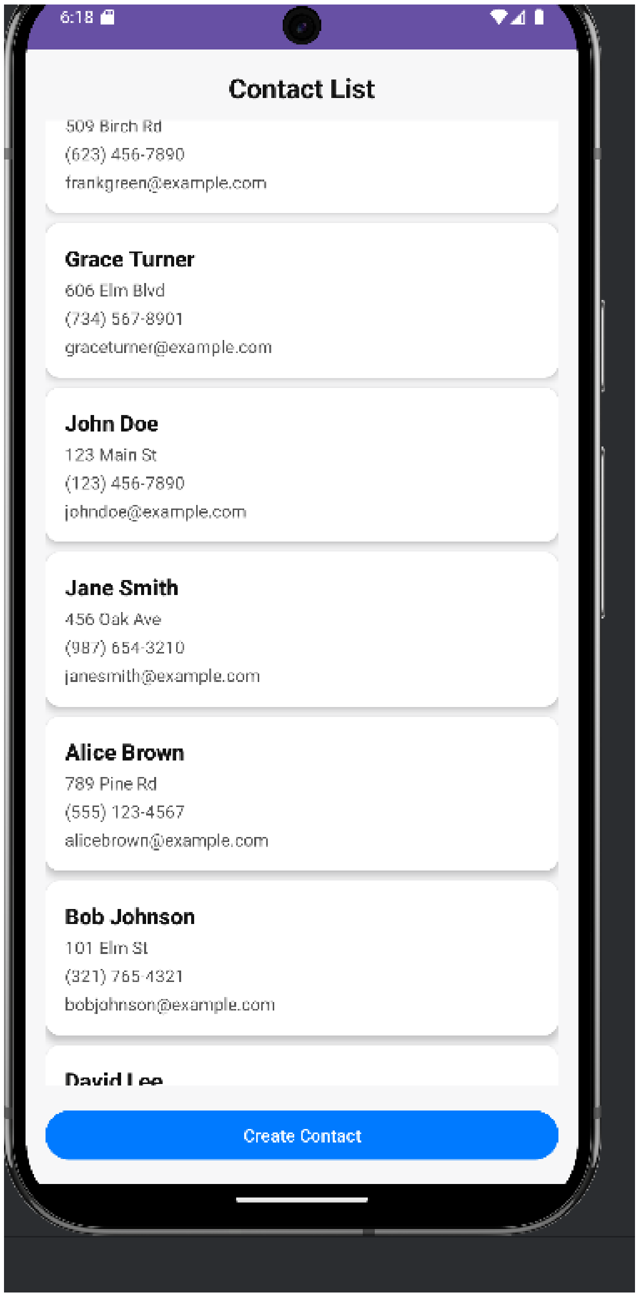
Computer: Ubuntu Linux 24.04

IDE: Android Studio LadyBug 2024.1.2 Patch 1 Build #AI 241.19072.14.2412.12360217, built on October 31, 2024

Runtime version: openjdk 23.0.1 20241015

Purpose

The AddressBooks app is a contact management application built with the MVVM architecture on Android, enabling users to create, view, edit, and delete contact information seamlessly. Using a Room database, the app securely stores contact details like name, phone number, email, and address. The main screen displays a scrollable list of contacts, allowing users to select a contact to view details or add new contacts. Through LiveData and ViewModel integration, the app provides a responsive UI, with automatic data updates and smooth navigation, making it an efficient tool for managing personal or professional contacts.


6:18

Contact List

509 Birch Rd
(623) 456-7890
frankgreen@example.com

Grace Turner
606 Elm Blvd
(734) 567-8901
graceturner@example.com

John Doe
123 Main St
(123) 456-7890
johndoe@example.com

Jane Smith
456 Oak Ave
(987) 654-3210
janesmith@example.com

Alice Brown
789 Pine Rd
(555) 123-4567
alicebrown@example.com

Bob Johnson
101 Elm St
(321) 765-4321
bobjohnson@example.com

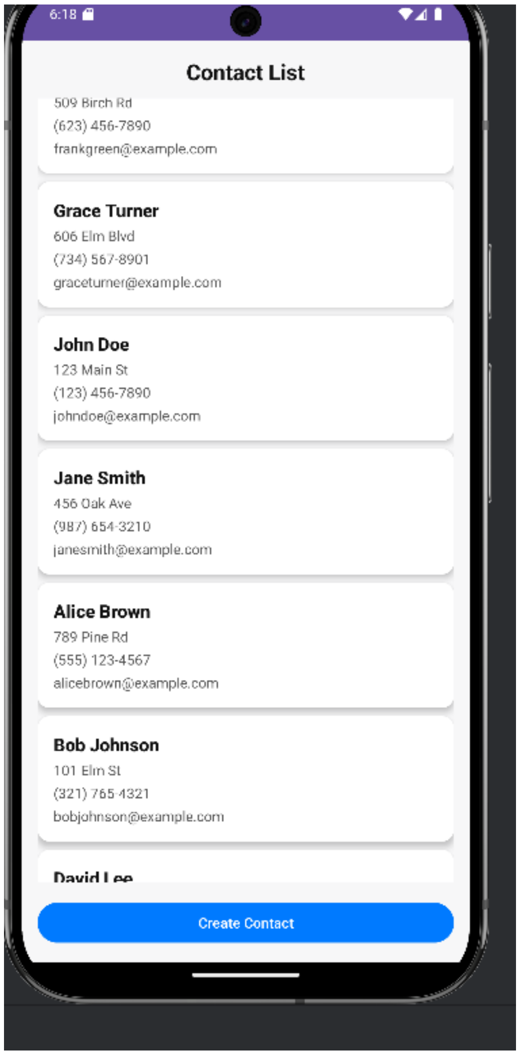
David Lee

ContactListActivity.java

Purpose: It's a major component that displays a dynamic list of saved contacts. A

RecyclerView was used in the application to present contacts in a beautiful, scrollable view. A user can select any contact to view or edit the details.

Main Elements: It sets up the RecyclerView and binds it with ContactAdapter; it observes a contact list with ContactViewModel to ensure that the user displays the latest information. And a scroll bar each of the contacts is clickable and underneath there is a button that allows you to create contacts.



509 Birch Rd
(623) 456-7890
frankgreen@example.com

Grace Turner
606 Elm Blvd
(734) 567-8901
graceturner@example.com

John Doe
123 Main St
(123) 456-7890
johndoe@example.com

Jane Smith
456 Oak Ave
(987) 654-3210
janesmith@example.com

Alice Brown
789 Pine Rd
(555) 123-4567
alicebrown@example.com

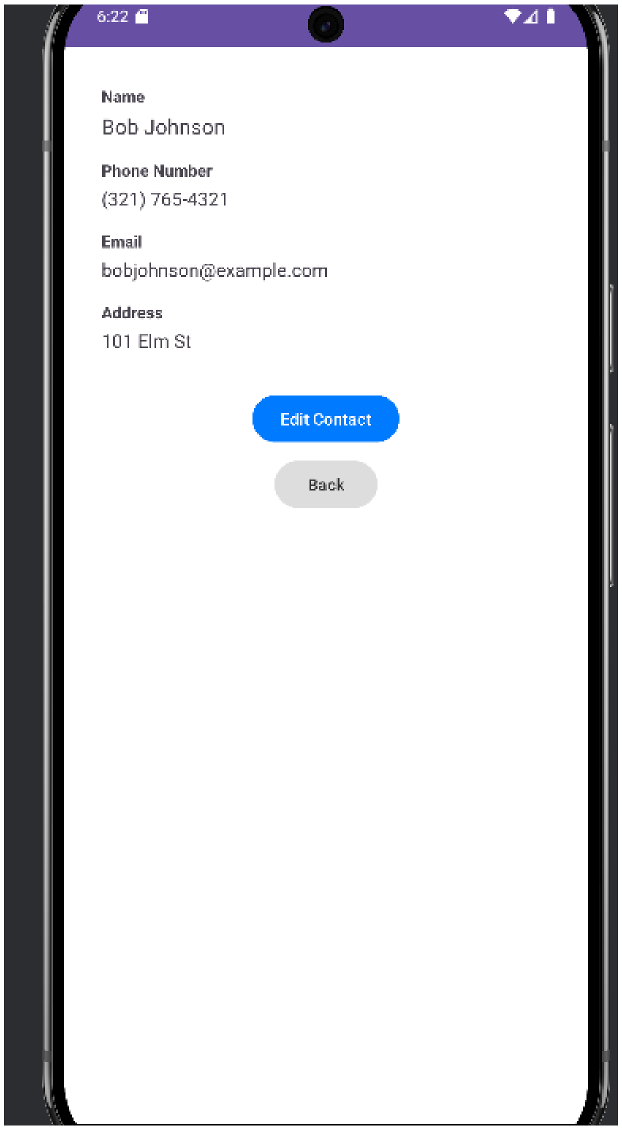
Bob Johnson
101 Elm St
(321) 765-4321
bobjohnson@example.com

David Lee

ContactViewActivity.java

Intent: will enable the user to focus on a particular contact's details. The user should be able to view details such as the contact's name, phone number, email ID, and address, and modify these details in case of necessity.

Main Components: The screen fetches the ID of the contact from an intent and displays the relevant information, with convenience buttons like "Back" and "Edit" included in the design for good user experience.

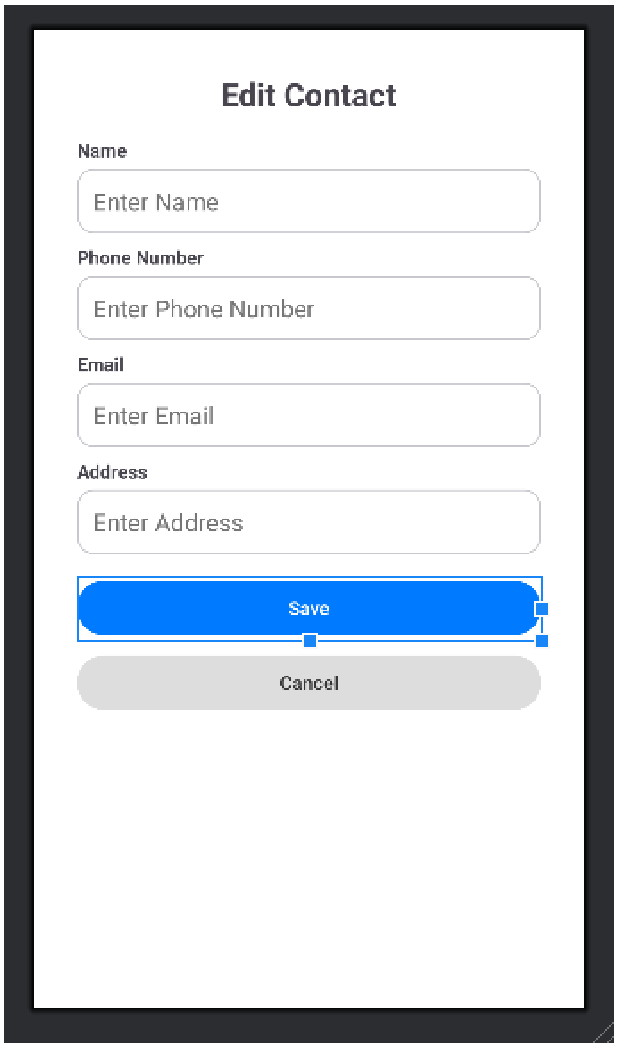


Name
Bob Johnson
Phone Number
(321) 765-4321
Email
bobjohnson@example.com
Address
101 Elm St

EditContactActivity.java

Purpose: In this, allow the user to either edit any of the existing contacts or create any new contact. In this screen, embed userfriendly form fields for updating details like name, phone, email, and address.

Major Components: This contains fields that give input for contact details and has a Save button; after any modification, this updates the database.



ContactAdapter.java

Purpose: This is like a bridge between the RecyclerView in ContactListActivity and the actual contact data, ensuring that every particular item view is correctly populated.

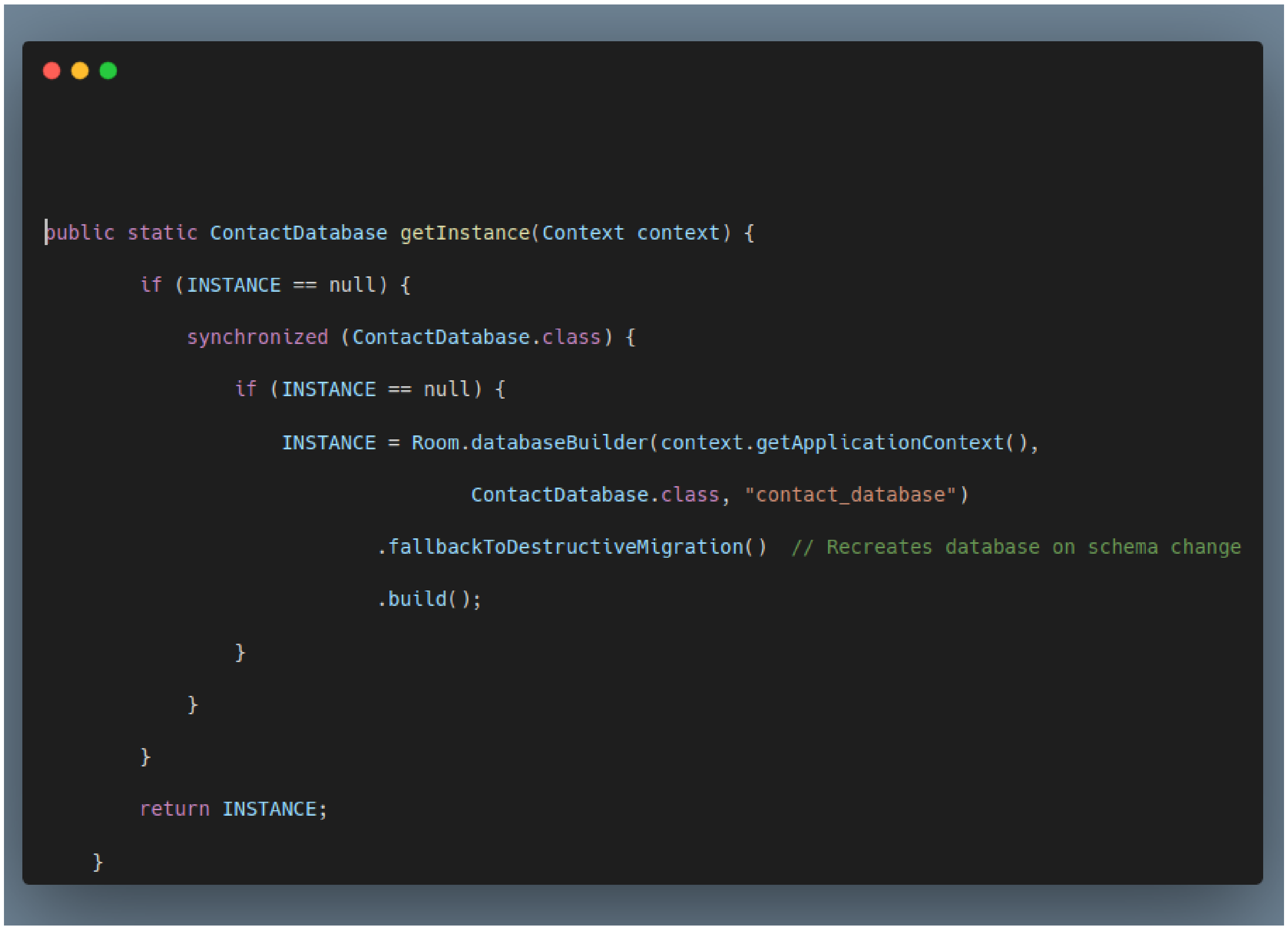
Major elements: uses.ViewHolder to define how every list elementcontactis going to look. It does the binding of data in the Contact objects to views inside the ViewHolder.

This thus helps in the dynamic and fluent user experience.

None

Package

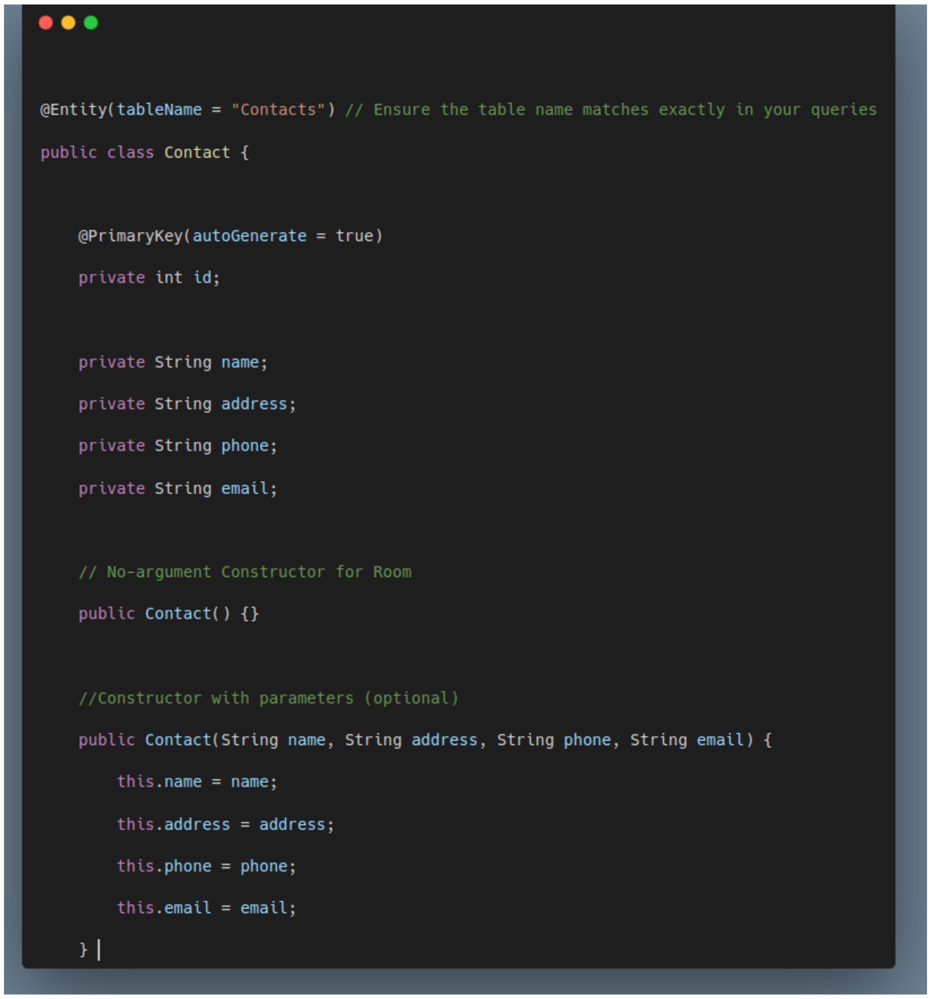
This will be handling database operations; it's the package important to handle appropriate data in the application.



```
public static ContactDatabase getInstance(Context context) {
        if (INSTANCE == null) {
            synchronized (ContactDatabase.class) {
                if (INSTANCE == null) {
                    INSTANCE = Room.databaseBuilder(context.getApplicationContext(),
                                    ContactDatabase.class, "contact_database")
                            .fallbackToDestructiveMigration()  // Recreates database on schema change
                            .build();
                }
            }
        }
        return INSTANCE;
    }
```

Contact.java

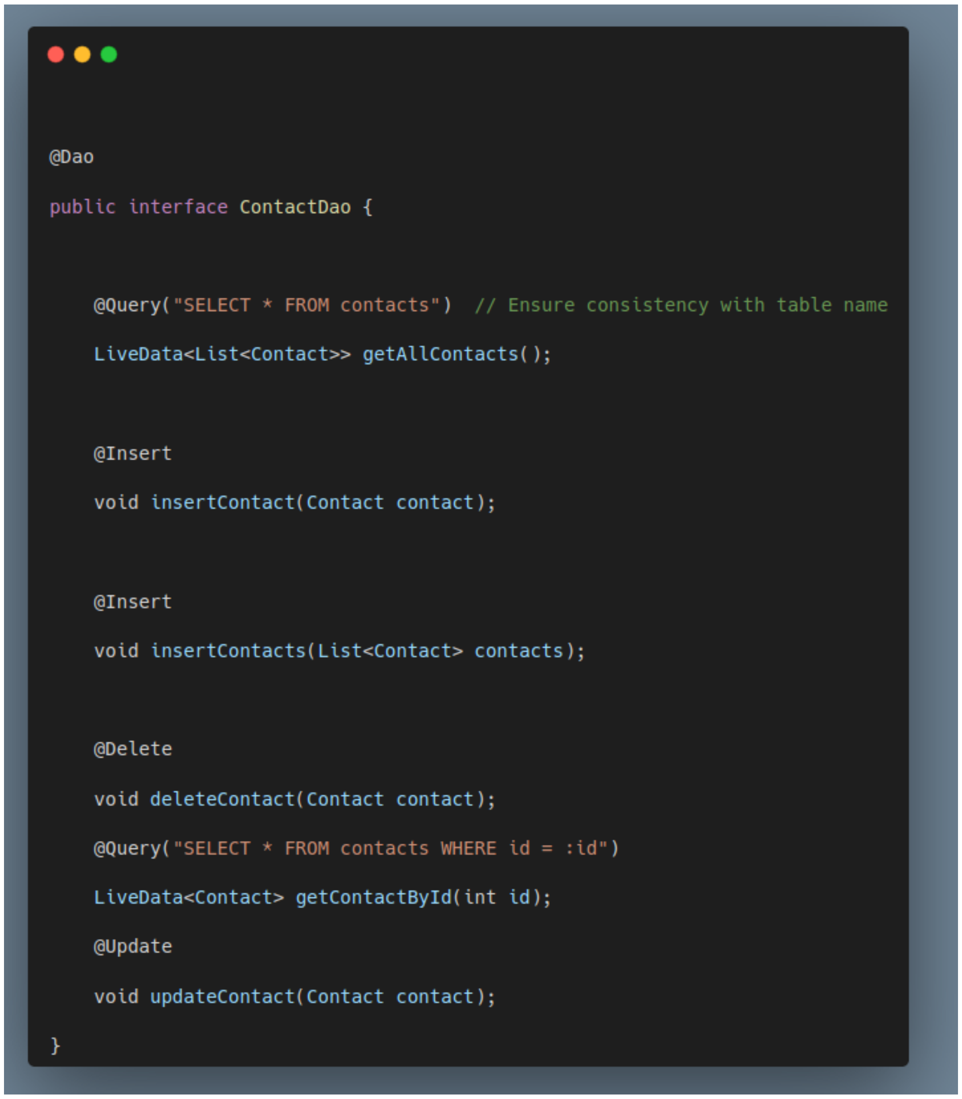
Purpose: This class persists Contact Entity using Room. The fields here have been designed for id, name, phone, email, and address. It includes some annotations which make Room consider this class as a database table.



```java
@Entity(tableName = "Contacts") // Ensure the table name matches exactly in your queries
public class Contact {

    @PrimaryKey(autoGenerate = true)
    private int id;

    private String name;
    private String address;
    private String phone;
    private String email;

    // No-argument Constructor for Room
    public Contact() {}

    //Constructor with parameters (optional)
    public Contact(String name, String address, String phone, String email) {
        this.name = name;
        this.address = address;
        this.phone = phone;
        this.email = email;
    }
```

ContactDao.java

Purpose: This is an interface describing the database operations - insert, update, delete, select among others - which can be performed on Contact entities. Thus, it declares methods for inserting, updating or deleting contacts, as well as methods which return all or only one Contact objects, like getAllContacts() and getContactById(). Room generates automatically the code that will perform these operations.



```
@Dao
public interface ContactDao {

    @Query("SELECT * FROM contacts")  // Ensure consistency with table name
    LiveData<List<Contact>> getAllContacts();

    @Insert
    void insertContact(Contact contact);

    @Insert
    void insertContacts(List<Contact> contacts);

    @Delete
    void deleteContact(Contact contact);
    @Query("SELECT * FROM contacts WHERE id = :id")
    LiveData<Contact> getContactById(int id);
    @Update
    void updateContact(Contact contact);
}
```

ContactViewModel.java

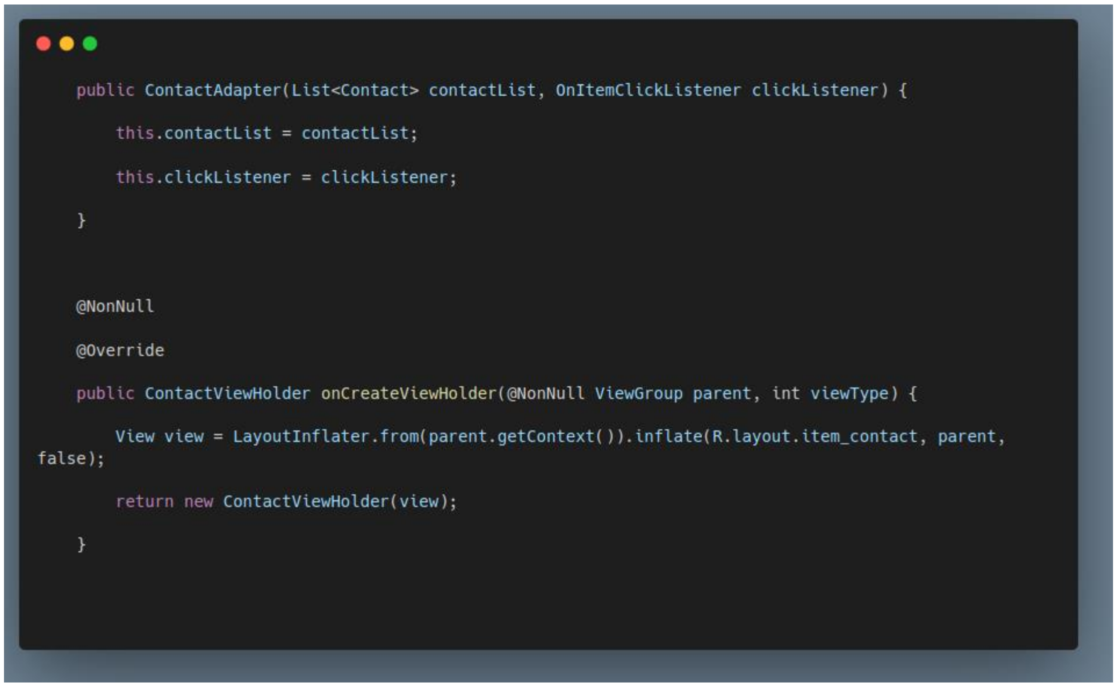
Purpose: This is the ViewModel serving as a kind of hub between the UI and the repository. ContactViewModel manages UI related data for Contact objects in life cycle aware fashion. It observes data from Contact Repository and exposes that data to the UI components that require it, such as ContactListActivity and ContactViewActivity. This ensures the data will be saved upon configuration changes such as screen rotations in a user-friendly fashion.

ContactRepository.java

Purpose: This repository is the unified source of Contact data, abstracts data sources, and provides a clean API for the ViewModel. Hence, it decouples the data handling logic from the UI layer, making the testing of the same along with data management easier.

ContactAdapter.java

Acts as a bridge between the RecyclerView (in ContactListActivity) and the contact data. It binds each Contact's data to the individual item views within the RecyclerView.The adapter binds each Contact to its corresponding view in the RecyclerView.



```
    public ContactAdapter(List<Contact> contactList, OnItemClickListener clickListener) {

        this.contactList = contactList;

        this.clickListener = clickListener;

    }


    @NonNull

    @Override

    public ContactViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {

        View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.item_contact, parent,
false);

        return new ContactViewHolder(view);

    }
```

Summary of the Structure

This architecture maintains a clean separation of concerns:

Model: Defines Contact data and manages database access.

Repository: Provides a unified data access point.

ViewModel: Manages UI data and handles data retrieval in a lifecycle-aware manner.

View (UI): Displays data to the user and reacts to changes, enabling smooth interaction through RecyclerView, adapters, and activities.

Activities: The main components users interact with, such as viewing, editing, and listing contacts.

Adapter: Manages and binds contact data to the UI within a list.

Database: Holds classes for data storage, retrieval, and management, including the Room database setup and the entity definition.

Repository: Provides a clean API for data access and abstracts data handling logic.

ViewModel: Manages UI-related data and lifecycle-aware data handling for a smoother user experience and better separation of concerns. This structure follows **MVVM (Model-View-ViewModel)**, which organizes code for maintainability and scalability. It separates data handling, UI logic, and the UI itself, making the app easier to test and extend.

Basic View Classes are

1. Google Developers. (n.d.). *Guide to app architecture*. Retrieved October 15, 2024, from https://developer.android.com/jetpack/guide
2. Harwani, B. M. (2018). *Android Programming Unleashed*. Sams Publishing.
3. Shilpashree, M., & Padmaja, K. (2020). "A Comprehensive Study on Android Mobile Application Development." *International Journal of Engineering Research and Technology*, 9(7), 100-107.
4. Nayak, R. & Sahoo, B. (2018). "Android Mobile Application Building Approach and Components Analysis." *International Journal of Computer Science and Mobile*

Computing, 7(6), 14–21.

5. Parhi, K., & Satapathy, S. (2019). "Architecture Patterns and MVVM Design Pattern in Android Development." *International Journal of Recent Technology and Engineering*, 8(4), 1669–1675.

Android Developers. (n.d.). *Android Studio: The official IDE for Android*. Retrieved October 15, 2024, from https://developer.android.com/studio

Android Developers. (n.d.). *Room Persistence Library*. Retrieved October 15, 2024, from https://developer.android.com/training/data-storage/room

Google Developers. (n.d.). *LiveData Overview*. Retrieved October 15, 2024, from https://developer.android.com/topic/libraries/architecture/livedata

[redacted]