**1. Menu Management**

This feature will allow the restaurant admin to add, edit, and delete menu items.

**Implementation:**

1. **Menu Manager Class:** Create a MenuManager class in the Business Logic Layer that will encapsulate all the menu management operations.
2. **Methods:** Define methods like addMenuItem(MenuItem item), editMenuItem(MenuItem item), and deleteMenuItem(int itemId) to perform the desired actions.
3. **Data Access:** Use a DAO (Data Access Object) class to interact with the database (SQLite) and store/retrieve menu item data.
4. **Admin Interface:** In the admin interface, provide input fields and buttons for adding, editing, and deleting menu items. Connect these UI elements to the MenuManager methods.

**Example Code:**

public class MenuManager {

private DAO dao;

public MenuManager(DAO dao) {

this.dao = dao;

}

public void addMenuItem(MenuItem item) {

dao.createMenuItem(item);

}

public void editMenuItem(MenuItem item) {

dao.updateMenuItem(item);

}

public void deleteMenuItem(int itemId) {

dao.deleteMenuItem(itemId);

}

}

**2. Order Placement**

This feature allows customers to place orders, selecting items from the menu and providing their delivery information.

**Implementation:**

1. **Order Manager Class:** Create an OrderManager class in the Business Logic Layer to handle order creation and related logic.
2. **Methods:** Define methods like createOrder(Order order) and placeOrder(Customer customer, List<MenuItem> items, Address address) to handle the order placement process.
3. **UI:** In the customer-facing UI, provide a menu browsing interface with item selection and quantity controls. Include input fields for delivery address and contact information. Connect these UI elements to the OrderManager methods.

**Example Code:**

public class OrderManager {

private DAO dao;

public OrderManager(DAO dao) {

this.dao = dao;

}

public void createOrder(Order order) {

dao.createOrder(order);

}

public void placeOrder(Customer customer, List<MenuItem> items, Address address) {

Order order = new Order(customer, items, address);

createOrder(order);

}

}

**3. Order Status Management**

The restaurant admin should be able to view and update the status of orders.

**Implementation:**

1. **Order Manager:** Enhance the OrderManager class with methods to retrieve orders, update their status, and manage the order lifecycle.
2. **Admin Interface:** In the admin interface, display a list of incoming orders with their current status. Provide buttons or controls to update the status (e.g., "preparing," "out for delivery," "completed").

**Example Code:**

public class OrderManager {

private DAO dao;

public OrderManager(DAO dao) {

this.dao = dao;

}

public List<Order> getOrders() {

return dao.getAllOrders();

}

public void updateOrderStatus(Order order, OrderStatus status) {

order.setStatus(status);

dao.updateOrder(order);

}

}

**4. Customer Notifications**

Customers should receive notifications about their order status, such as when the order is placed, confirmed, prepared, and out for delivery.

**Implementation:**

1. **Notification Service:** Create a notification service that can send push notifications to users.
2. **Order Manager:** In the OrderManager, integrate the notification service to send notifications at appropriate stages of the order lifecycle.

I'll design a comprehensive structure suitable for a course project that demonstrates your understanding of Software Engineering principles, Android development, Java, and databases.

```mermaid

classDiagram

class MainActivity {

+BottomNavigationView navigation

+setupNavigation()

}

class MenuFragment {

-MenuViewModel viewModel

-RecyclerView menuList

+onCreateView()

+displayMenu()

}

class OrderFragment {

-OrderViewModel viewModel

-RecyclerView cartItems

+onCreateView()

+placeOrder()

}

class AdminFragment {

-AdminViewModel viewModel

+onCreateView()

+manageMenu()

+manageOrders()

}

class MenuViewModel {

-MenuRepository repository

+getAllItems()

+addMenuItem()

+updateMenuItem()

+deleteMenuItem()

}

class OrderViewModel {

-OrderRepository repository

+placeOrder()

+getOrders()

+updateOrderStatus()

}

class MenuItem {

-String id

-String name

-double price

-String description

-String imageUrl

+getters()

+setters()

}

class Order {

-String id

-String userId

-List~MenuItem~ items

-OrderStatus status

-String address

+getters()

+setters()

}

class MenuRepository {

-FirebaseDatabase firebaseDb

-SQLiteDatabase localDb

+syncData()

+saveMenuItem()

+fetchMenuItems()

}

class OrderRepository {

-FirebaseDatabase firebaseDb

-SQLiteDatabase localDb

+syncOrders()

+saveOrder()

+fetchOrders()

}

class LocalDatabase {

+getMenuItems()

+saveMenuItem()

+getOrders()

+saveOrder()

}

MainActivity --> MenuFragment

MainActivity --> OrderFragment

MainActivity --> AdminFragment

MenuFragment --> MenuViewModel

OrderFragment --> OrderViewModel

AdminFragment --> MenuViewModel

AdminFragment --> OrderViewModel

MenuViewModel --> MenuRepository

OrderViewModel --> OrderRepository

MenuRepository --> MenuItem

OrderRepository --> Order

MenuRepository --> LocalDatabase

OrderRepository --> LocalDatabase

```

# Restaurant App Project Structure

## 1. Project Organization

```

app/

├── java/

│ └── com.example.restaurantapp/

│ ├── activities/

│ │ └── MainActivity.java

│ ├── fragments/

│ │ ├── MenuFragment.java

│ │ ├── OrderFragment.java

│ │ └── AdminFragment.java

│ ├── viewmodels/

│ │ ├── MenuViewModel.java

│ │ └── OrderViewModel.java

│ ├── models/

│ │ ├── MenuItem.java

│ │ └── Order.java

│ ├── repositories/

│ │ ├── MenuRepository.java

│ │ └── OrderRepository.java

│ ├── database/

│ │ ├── LocalDatabase.java

│ │ └── DatabaseHelper.java

│ ├── adapters/

│ │ ├── MenuAdapter.java

│ │ └── OrderAdapter.java

│ └── utils/

│ ├── Constants.java

│ └── FirebaseUtils.java

└── res/

├── layout/

│ ├── activity\_main.xml

│ ├── fragment\_menu.xml

│ ├── fragment\_order.xml

│ └── fragment\_admin.xml

└── values/

├── strings.xml

├── colors.xml

└── styles.xml

```

## 2. Key Components

### Activities

- \*\*MainActivity\*\*: Single activity architecture with navigation between fragments

### Fragments

- \*\*MenuFragment\*\*: Displays menu items for customers

- \*\*OrderFragment\*\*: Handles cart and order placement

- \*\*AdminFragment\*\*: Menu and order management for admin

### ViewModels

- \*\*MenuViewModel\*\*: Menu-related business logic

- \*\*OrderViewModel\*\*: Order processing logic

### Models

- \*\*MenuItem\*\*: Menu item data class

- \*\*Order\*\*: Order data class

### Repositories

- \*\*MenuRepository\*\*: Handles menu data operations

- \*\*OrderRepository\*\*: Handles order data operations

### Database

- \*\*LocalDatabase\*\*: SQLite implementation

- \*\*FirebaseDatabase\*\*: Cloud database integration

## 3. Implementation Highlights

### Firebase Integration

```java

public class FirebaseUtils {

private FirebaseDatabase database;

private DatabaseReference menuRef;

private DatabaseReference ordersRef;

public void initializeDatabase() {

database = FirebaseDatabase.getInstance();

menuRef = database.getReference("menu");

ordersRef = database.getReference("orders");

}

public void syncWithLocal(LocalDatabase localDb) {

menuRef.addValueEventListener(new ValueEventListener() {

@Override

public void onDataChange(@NonNull DataSnapshot snapshot) {

// Sync with SQLite

}

});

}

}

```

### Local Database

```java

public class LocalDatabase extends SQLiteOpenHelper {

private static final String DATABASE\_NAME = "RestaurantDB";

private static final int DATABASE\_VERSION = 1;

// Table creation queries

private static final String CREATE\_MENU\_TABLE =

"CREATE TABLE menu (id TEXT PRIMARY KEY, name TEXT, price REAL, description TEXT, imageUrl TEXT)";

private static final String CREATE\_ORDERS\_TABLE =

"CREATE TABLE orders (id TEXT PRIMARY KEY, userId TEXT, status TEXT, address TEXT)";

}

```

### Repository Pattern

```java

public class MenuRepository {

private LocalDatabase localDb;

private FirebaseDatabase firebaseDb;

public LiveData<List<MenuItem>> getMenuItems() {

// Check local DB first

List<MenuItem> localItems = localDb.getMenuItems();

if (localItems.isEmpty()) {

// Fetch from Firebase if local is empty

fetchFromFirebase();

}

return new MutableLiveData<>(localItems);

}

}

```

## 4. Key Features Demonstration

1. \*\*Software Engineering Principles\*\*

- MVVM Architecture

- Repository Pattern

- Single Responsibility Principle

- Dependency Injection

2. \*\*Android Studio & Java\*\*

- Fragments

- RecyclerViews

- LiveData

- ViewModels

- Navigation Component

3. \*\*Database Management\*\*

- SQLite local storage

- Firebase real-time database

- Data synchronization

- CRUD operations

## 5. Data Flow

1. \*\*Online-Offline Sync\*\*

- Local SQLite for offline access

- Firebase for real-time updates

- Automatic synchronization when online

- Conflict resolution strategies

2. \*\*User Data Flow\*\*

- Authentication state management

- Order status updates

- Menu updates

- Real-time notifications

This structure demonstrates:

1. \*\*Software Engineering Skills\*\*

- Clean Architecture

- MVVM pattern

- Repository pattern

- Separation of concerns

2. \*\*Android/Java Proficiency\*\*

- Fragment management

- ViewModel usage

- RecyclerView implementations

- LiveData observables

3. \*\*Database Knowledge\*\*

- Local SQLite implementation

- Firebase integration

- Data synchronization

- CRUD operations

The key advantages of this structure:

1. \*\*Offline Capability\*\*: SQLite ensures the app works without internet

2. \*\*Real-time Updates\*\*: Firebase enables instant updates across devices

3. \*\*Scalability\*\*: Easy to add new features

4. \*\*Maintainability\*\*: Clear separation of concerns

5. \*\*Code Organization\*\*: Logical project structure