For the **Smart Shop Management System** project, it's assumed to be useful to use the **Layered Architecture** (n-Tier Architecture) with some elements of **MVVM (Model-View-ViewModel)**. Here's why:

Recommended Architecture: Layered Architecture + MVVM

Layered Architecture

This approach breaks the application into logical layers, making it scalable and easy to maintain. have distinct layers for different components of the system can be done.

1. **Presentation Layer (UI)**
    - Separate interfaces for the manager, data entry officer, and salesman.
    - Each user type has a different role in the system with specific screens (data entry, billing, customer management, offers, etc.).
2. **Business Logic Layer**

- Manages core functions like customer profile updates, tier assignments, billing logic, stock management, and offer generation. Handles complex logic, such as assigning customer tiers based on their spending, updating stock information, and notifying customers of special offers.

3. **Data Access Layer**

- Responsible for managing the database and storing information such as customer profiles, product data, sales records, and notifications.
- Interacts with the database for CRUD operations (Create, Read, Update, Delete) without impacting the rest of the system.

4. **Notification/Communication Layer**

- Manages the sending of SMS notifications when a customer has purchased a product on special offer.
- This layer can interact with a third-party service or API to send text messages.

MVVM for Presentation Layer

MVVM can be implemented in the presentation layer to manage the complexity of the UI interactions. The **Model-View-ViewModel** architecture will keep the UI separate from the logic that drives it, making the system easier to maintain and extend in the future.

- **Model**: Represents data (e.g., customer data, product stock, sales data).
- **View**: The UI elements (different user roles, screens for data entry, billing, customer profiles, etc.).
- **ViewModel**: Manages the interaction between the UI and the underlying data, making sure that data is updated in the UI in real time (e.g., billing amounts, customer profiles).

Advantages:

- **Modular Design**: Each layer can be developed and updated independently.

- **Scalability**: Easily add more features such as tier systems, promotions, or more users in the future.
- **Maintenance**: Codebase is easier to maintain since responsibilities are well-separated.
- **Offline System**: It suits the need for offline operations while allowing features like customer notifications to work without needing an online shopping interface.

Potential Tools:

- **SQLite**: For local database management.
- **SMS API**: For text messaging to notify customers (e.g., Twilio or Nexmo).
- **Jetpack Compose** or **XML Layouts**: For UI development in Android.

This architecture will support smooth role-based operations (Manager, Salesman, Data Entry Officer), track customer profiles, manage sales, and handle notifications effectively.