Medication Reminder App

Mobile Application Development

Final Project Report

American International University-Bangladesh

October 2025

Project:	Medication Reminder App
Course:	Mobile Application Development
Author:	Md Mochaddec Hossain
Instructor:	Md. Khairul Alam Mazumder
Department:	Computer Science and Engineering
Submission Date:	October 2025

Contents

1 Abstract

The **Medication Reminder App** is an Android application written in Kotlin (Android Studio). It helps users manage medication schedules: add, edit, delete medicines; view grouped schedules (Morning/Noon/Night); maintain inventory (cabinet); and prepare for notifications. The app follows the MVVM architecture, uses Room (SQLite) for persistence, LiveData for reactive UI, and SharedPreferences for lightweight settings and draft-saving.

2 Introduction

People often miss medication doses. This app provides a simple, robust solution to schedule and track medication intake. It demonstrates Android fundamentals taught in the course: Activities, Fragments, ConstraintLayout, Intents, SharedPreferences, AlertDialog, Snackbar, Room, Repositories, ViewModels, and notification helper patterns.

3 Objectives

- Build a UI-first medication reminder app using only course-covered topics.
- Store structured data locally using SQLite (Room).
- Demonstrate MVVM pattern and LiveData-driven UI updates.
- Provide add/edit/delete (CRUD) operations for medicines and inventory.
- Provide a NotificationHelper to display reminders.

4 System Overview

The system is a multi-fragment Android app with a single activity host and modular components:

- **UI:** Activities and Fragments (Dashboard, Schedule, Cabinet, Settings).
- **ViewModels:** Expose LiveData and run repository operations.
- **Repositories:** Encapsulate DAO/Room logic.
- **Room:** Entities, DAOs, and database instance.
- **Utils:** NotificationHelper and TimeShiftManager.

5 Architecture Diagram

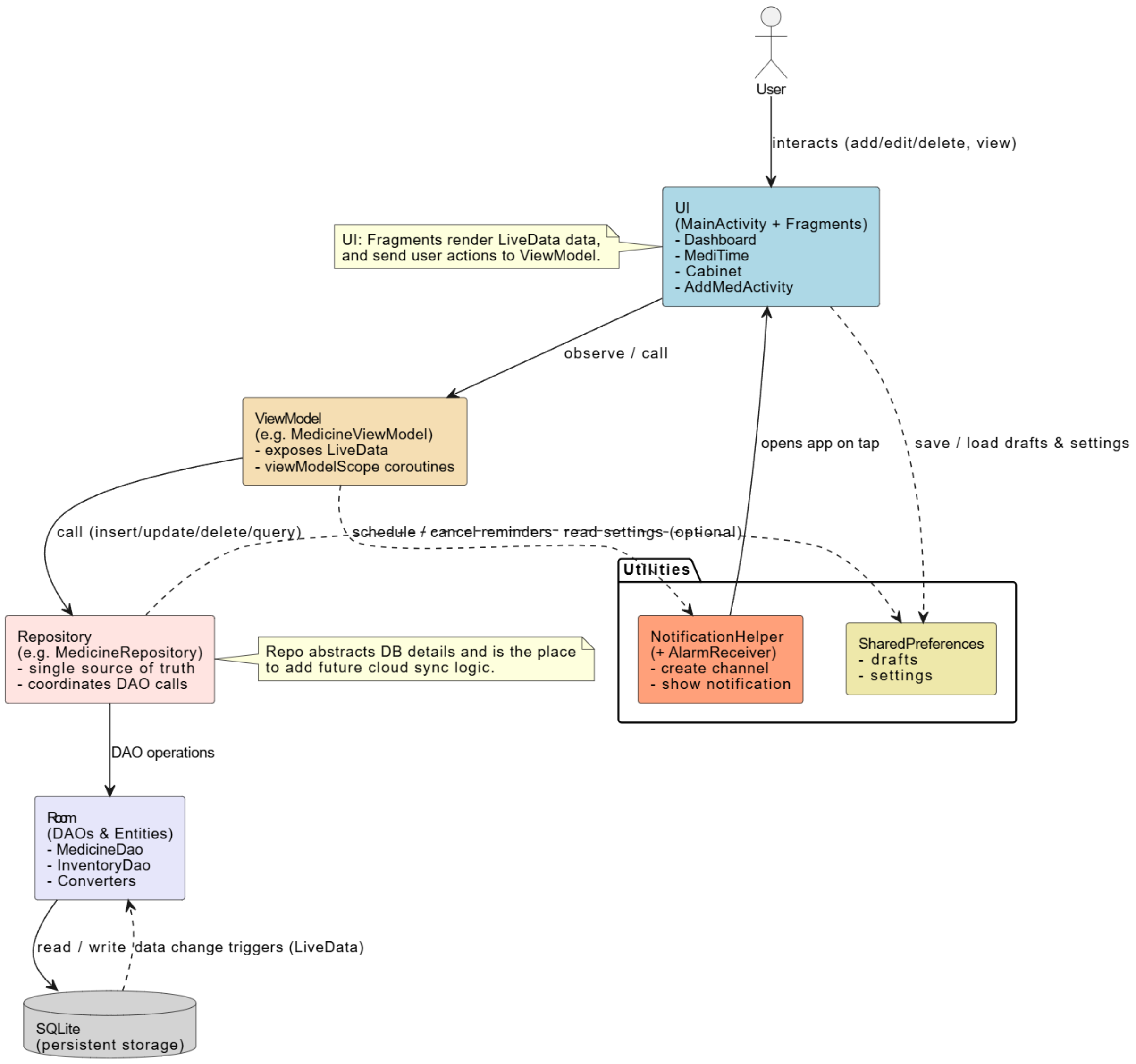

User
interacts (add/edit/delete, view)
UI (MainActivity + Fragments) - Dashboard - MediTime - Cabinet - AddMedActivity
UI: Fragments render LiveData data, and send user actions to ViewModel.
observe / call
opens app on tap
save / load drafts & settings
ViewModel (e.g. MedicineViewModel) - exposes LiveData - viewModelScope coroutines
call (insert/update/delete/query)
schedule / cancel reminders
read settings (optional)
Utilities
NotificationHelper (+ AlarmReceiver) - create channel - show notification
SharedPreferences - drafts - settings
Repository (e.g. MedicineRepository) - single source of truth - coordinates DAO calls
Repo abstracts DB details and is the place to add future cloud sync logic.
DAO operations
Room (DAOs & Entities) - MedicineDao - InventoryDao - Converters
read / write
data change triggers (LiveData)
SQLite (persistent storage)

Figure 1: High-level system architecture (MVVM + Room + utilities)

6 Dataflow Diagram

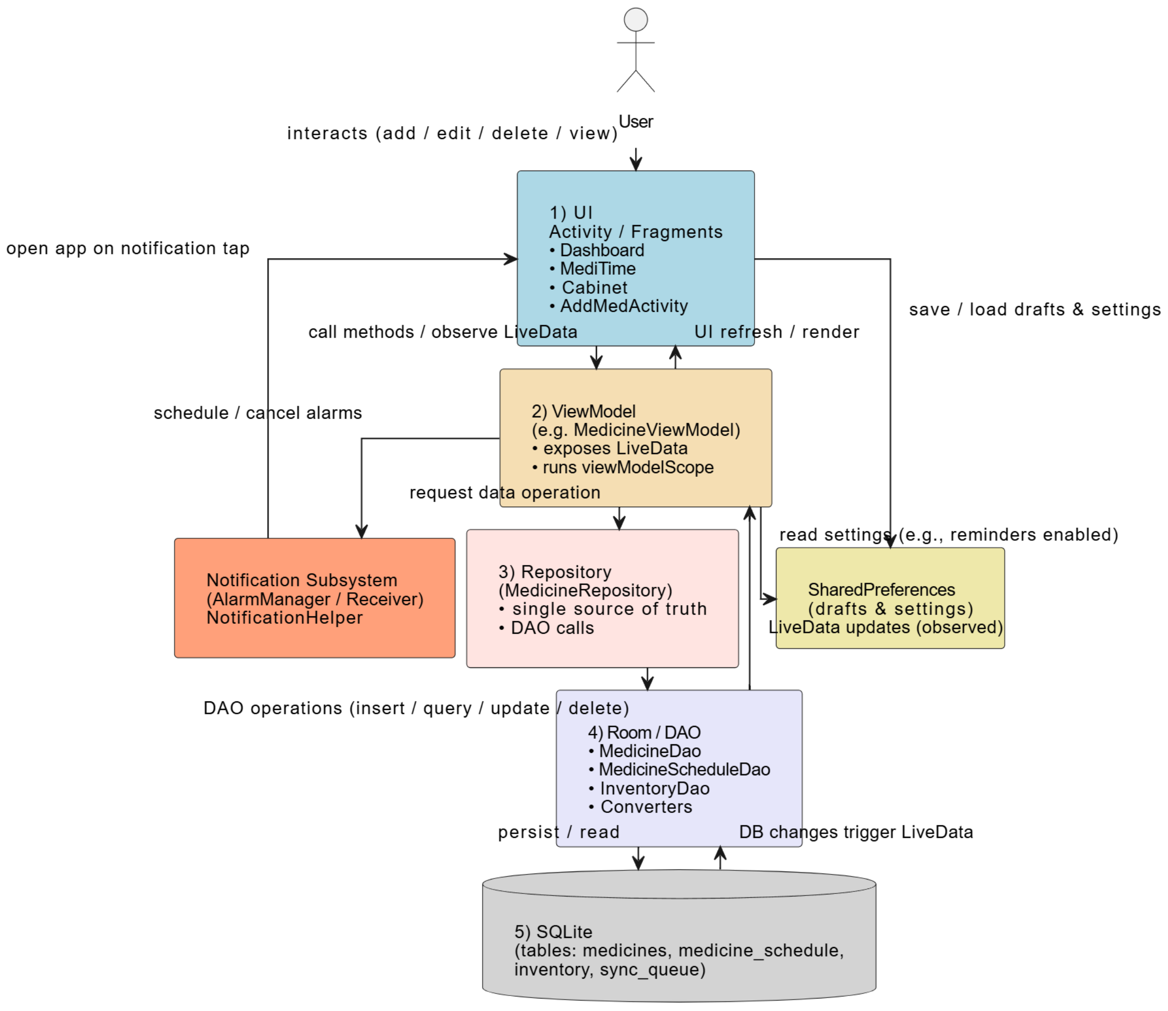


Figure 2: Data flow from UI to DB and reactive update back to UI

7 Database Design

7.1 Entities

- **Medicine** (id: TEXT PK, name: TEXT, dosage: TEXT, times: TEXT, mealType: TEXT, ifMissed: TEXT, status: TEXT, inventoryId: TEXT, lastModified: INTEGER)
- **MedicineSchedule** (id: TEXT PK, medicineId: TEXT FK, shift: TEXT, timeOfDay: TEXT, lastModified: INTEGER)
- **Inventory** (id: TEXT PK, name: TEXT, unit: TEXT, stock: INTEGER, lastModified: INTEGER)

7.2 ER Diagram

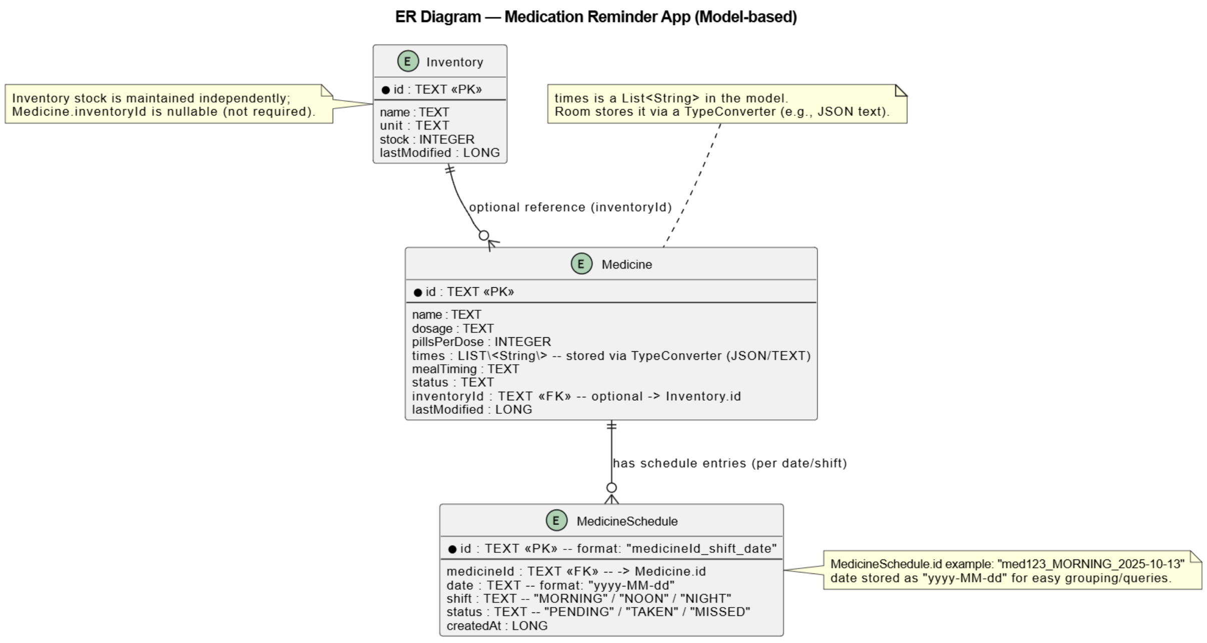

ER Diagram — Medication Reminder App (Model-based)
E Inventory
● id : TEXT «PK»
name : TEXT
unit : TEXT
stock : INTEGER
lastModified : LONG
Inventory stock is maintained independently;
Medicine.inventoryId is nullable (not required).
times is a List<String> in the model.
Room stores it via a TypeConverter (e.g., JSON text).
optional reference (inventoryId)
E Medicine
● id : TEXT «PK»
name : TEXT
dosage : TEXT
pillsPerDose : INTEGER
times : LIST\<String\> -- stored via TypeConverter (JSON/TEXT)
mealTiming : TEXT
status : TEXT
inventoryId : TEXT «FK» -- optional -> Inventory.id
lastModified : LONG
has schedule entries (per date/shift)
E MedicineSchedule
● id : TEXT «PK» -- format: "medicineId_shift_date"
medicineId : TEXT «FK» -- -> Medicine.id
date : TEXT -- format: "yyyy-MM-dd"
shift : TEXT -- "MORNING" / "NOON" / "NIGHT"
status : TEXT -- "PENDING" / "TAKEN" / "MISSED"
createdAt : LONG
MedicineSchedule.id example: "med123_MORNING_2025-10-13"
date stored as "yyyy-MM-dd" for easy grouping/queries.

Figure 3: Entity Relationship Diagram

8 Implementation Details

8.1 Project Structure

```
app/src/main/java/com/yourpackage/
├── model/
│   ├── Medicine.kt
│   ├── MedicineSchedule.kt
│   └── Inventory.kt
├── room/
│   ├── AppDatabase.kt
│   ├── Converters.kt
│   └── dao/
│       ├── MedicineDao.kt
│       ├── MedicineScheduleDao.kt
│       └── InventoryDao.kt
├── repository/
│   ├── MedicineRepository.kt
│   ├── MedicineScheduleRepository.kt
│   └── InventoryRepository.kt
├── viewModel/
│   ├── MedicineViewModel.kt
│   ├── MedicineScheduleViewModel.kt
│   ├── InventoryViewModel.kt
│   └── (factories)
├── view/
│   ├── activities/
│   │   ├── MainActivity.kt
│   │   └── AddMedActivity.kt
│   └── fragments/
│       ├── Dashboard.kt
│       ├── MediTime.kt
│       ├── Cabinet.kt
│       └── Settings.kt
└── utils/
    ├── NotificationHelper.kt
    └── TimeShiftManager.kt
```

8.2 Key Implementation Notes

- **Room/Dao/Repository:** Entities annotated with `@Entity`, DAOs annotated with `@Dao` providing `@Insert`, `@Update`, `@Delete` and `@Query` methods. Repositories provide a thin layer over DAOs.
- **ViewModel:** Exposes LiveData lists and wraps repository calls inside `viewModelScope.launch({...})` to avoid blocking UI.
- **SharedPreferences:** Used only for UI drafts and settings (small key-value pairs).
- **Notifications:** NotificationHelper class creates notification channel and builds notifications; integration via AlarmManager/WorkManager is recommended.

9 Functional Modules

9.1 Add Medication

- **Screen:** AddMedActivity (XML: `activity_add_med.xml`)
- **Fields:** Name, Dosage, Times (Morning/Noon/Night checkboxes), Before/After meal (radio), Pills per dose.
- **Flow:** Validate → build `Medicine` object → ViewModel.add → Repository.insert → DAO.insert → Room writes to DB.

9.2 Dashboard

- **Screen:** Dashboard Fragment (XML: `fragment_dashboard.xml`)
- **Components:** ScrollView with card items for upcoming meds, horizontal cabinet preview, FAB to add new medication.
- **Actions:** Edit (open AddMedActivity with extra), Delete (AlertDialog → ViewModel.delete), Mark Taken (update status and optionally decrement inventory).

9.3 Schedule (MediTime)

- **Screen:** MediTime Fragment (XML: `fragment_mediTime.xml`)
- **Behavior:** Group meds by shift (Morning/Noon/Night) and show subdivided cards by meal timing.

9.4 Cabinet (Inventory)

- **Screen:** Cabinet Fragment
- **Actions:** Increment/Decrement stock, Edit/Delete inventory item.

10 CRUD Operations

10.1 Create

When the user adds a new medicine, input data is validated and inserted into the SQLite database using a repository pattern through `MedicineRepository`.

10.2 Read

Data is retrieved using a `Cursor` or `LiveData<List<Medicine»` from the database and displayed dynamically in ScrollView inside DashboardFragment.

10.3 Update

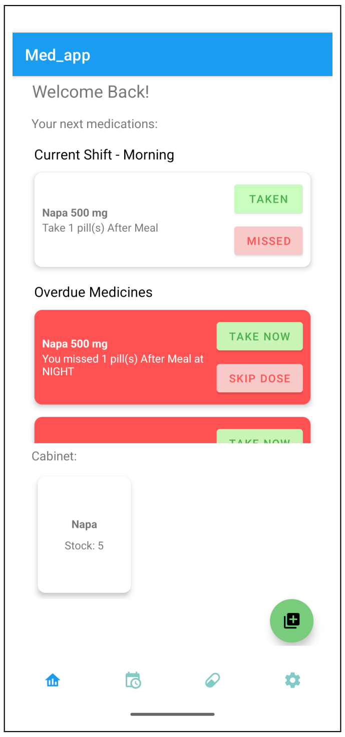
Edit button on each card allows modification of existing medicine entries, updating the SQLite database record and refreshing the list adapter.

10.4 Delete

Delete button removes the selected record from the database and updates the ScrollView immediately.

11 User Interface

11.1 Screenshots (Emulator output)



Med_app

Welcome Back!

Your next medications:

Current Shift - Morning

Napa 500 mg
Take 1 pill(s) After Meal

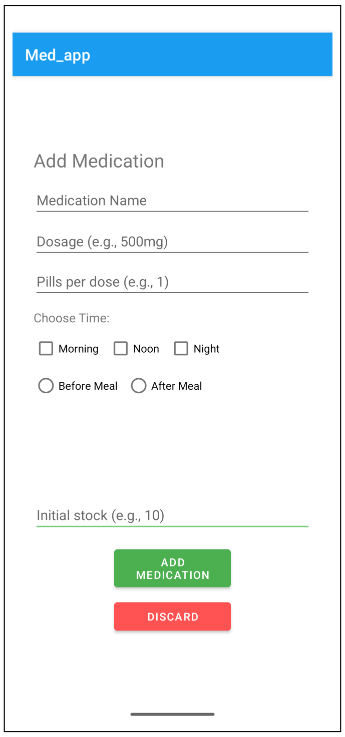
Overdue Medicines

Napa 500 mg
You missed 1 pill(s) After Meal at NIGHT

Cabinet:

Napa
Stock: 5

(a) Dashboard



Med_app

Add Medication

Medication Name

Dosage (e.g., 500mg)

Pills per dose (e.g., 1)

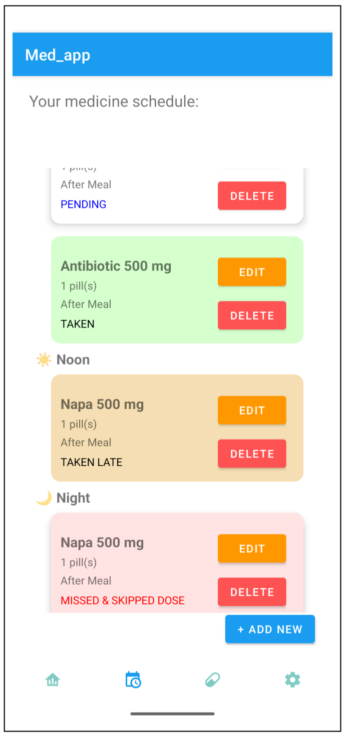
Choose Time:

☐ Morning ☐ Noon ☐ Night

○ Before Meal ○ After Meal

Initial stock (e.g., 10)

(b) Add/Edit Medication Form



Med_app

Your medicine schedule:

After Meal
PENDING

Antibiotic 500 mg
1 pill(s)
After Meal
TAKEN

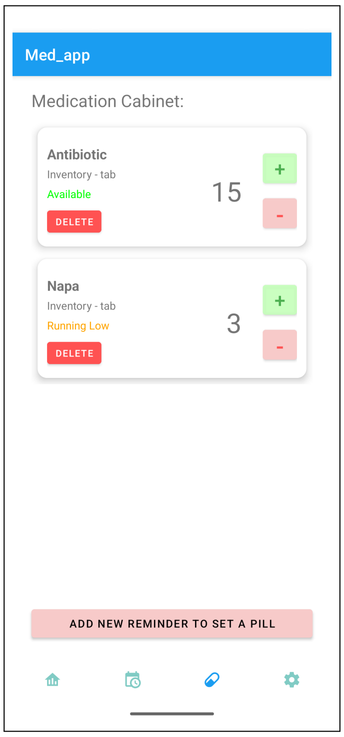
Noon

Napa 500 mg
1 pill(s)
After Meal
TAKEN LATE

Night

Napa 500 mg
1 pill(s)
After Meal
MISSED & SKIPPED DOSE

(a) Full Schedule



Med_app

Medication Cabinet:

Antibiotic
Inventory - tab
Available
15

Napa
Inventory - tab
Running Low
3

(b) Cabinet

12 Testing and Validation

12.1 Test Cases

Test ID	Scenario	Expected Result
T1	Add medicine with valid inputs	New medicine appears immediately in Dashboard and Schedule.
T2	Edit medicine	Updated values displayed; DB updated.
T3	Delete medicine	Row removed; UI updated.
T4	Decrement inventory below 0	Prevent negative stock; show warning.

13 Future Work

- Cloud sync using Firebase Realtime Database / Firestore.
- Scheduling accurate alarms using AlarmManager or WorkManager and linking to NotificationHelper.
- Add user authentication and multi-profile support.
- Export/import data (CSV / JSON).

14 Conclusion

The project demonstrates a complete, maintainable Android app implementing course-covered features in a production-like architecture. It is ready for demonstration, extension to cloud sync, and addition of scheduled notifications.

15 References

- Mobile Application Development - `https://github.com/Robinak47/Mobile_App_Dev`
- Android Developers — `https://developer.android.com`
- Kotlin Documentation — `https://kotlinlang.org/docs`
- Android Knowledge - `https://www.youtube.com/@android_knowledge`

16 Source Code

GitHub - `https://github.com/hossainGit/MedApp-RoomDB-Final`