Health Care Management System

Technical Documentation

Olar Tudor
Mandoiu Vlad Stefan
Jemal Ovezdurdyyeva

February 3, 2026

Abstract

This document presents the design and usage of a native Android Health Care Management System implemented in Kotlin with Jetpack Compose. The application supports two roles (Patients and Doctors) and provides authentication, appointment booking and management, medical record handling, and medication reminders implemented via background notifications. The text combines an article-like narrative with a concise technical sketch of the system structure, allowing the reader to understand both the intent and the concrete implementation layout.

Contents

1 Introduction

The *Health Care Management System* is a native Android application that brings together two perspectives in one codebase: Patients, who primarily need a simple way to schedule care and keep track of personal health information, and Doctors, who need a lightweight interface for managing requests and documenting outcomes. The application therefore adopts role-based routing: after a standard registration and login flow, the user is directed to a role-specific home screen and feature set.

From a software engineering perspective, the project is structured in a conventional layered style aligned with MVVM. Compose screens remain focused on presentation; ViewModels handle UI state and orchestrate operations; repositories mediate between UI logic and stored data; and Room provides persistence over SQLite. Time-dependent medication alerts are implemented through WorkManager, which integrates background execution with Android OS constraints.

2 Functional Overview (What the App Does)

In everyday use, the Patient experience resembles a personal health companion. After authentication, a Patient can navigate from the dashboard to appointment booking, appointment tracking, medical record viewing, medication reminder management, and health tips. The core workflow starts with creating an appointment request; subsequent screens allow the Patient to follow the request as it progresses through statuses.

Doctors use the application in a more operational manner. They can review pending appointment requests, approve or reject them, and mark appointments as completed. Following a consultation, a Doctor can create a medical record entry, which becomes visible to the Patient as part of their medical history. Health tips are shared across both roles as a general informational module.

2.1 Feature Sketch by Role

The following overview lists the implemented features in a compact form.

Patient features:

- Register and login
- Dashboard (central navigation hub)
- Book appointments and track appointment status
- View medical records
- Create and manage medication reminders
- Browse health tips

Doctor features:

- Register and login
- Dashboard (doctor control panel)
- View appointments (including pending requests)
- Approve/reject appointments; mark appointments complete
- Add medical records after consultation
- Browse health tips

3 Technology Stack and Configuration (How it is Built)

The implementation uses a modern Android stack emphasizing maintainability and clear separation of concerns.

3.1 Core Technologies

- Kotlin and Jetpack Compose (declarative UI)
- MVVM (state and UI logic separation)
- Room (local persistence with DAOs and entities)
- WorkManager (background execution for reminders/notifications)
- Navigation Compose (route-based navigation and role routing)

3.2 Build Configuration (as documented)

- Gradle: 8.13.2
- Kotlin: 2.0.21
- Compose BOM: 2024.09.00
- Room: 2.6.1
- Navigation Compose: 2.7.7
- WorkManager: 2.9.0
- Gson: 2.10.1
- Min SDK: 26 (Android 8.0)
- Target SDK: 36

4 Architecture (A Guided Tour)

The system can be read as a data flow from UI interaction to persistence and back. Compose screens render state and collect input; ViewModels hold state and define user actions; repositories provide a stable API for data operations; DAOs implement database queries; and entities define the stored schema. Navigation is defined centrally through route definitions and a navigation graph, enabling role-based routing and a coherent user journey from login to feature screens.

Medication reminders are the main element that extends beyond foreground UI. Instead of relying on timers tied to UI lifecycles, the project uses WorkManager and a dedicated worker to trigger notifications, improving reliability across device states (foreground/background) and OS policies.

4.1 Layered Sketch

- **Application entry:** `MainActivity.kt` (initialization and navigation host)
- **Navigation:** `Screen.kt` (routes), `AppNavigation.kt` (navigation graph)
- **UI:** Compose screens (patient/doctor flows)
- **ViewModels:** auth, appointments, medical records, reminders

- **Data:** entities, DAOs, repositories, Room database configuration
- **Background:** reminder worker for notifications

5 Codebase Organization (Where Things Are)

The package structure mirrors the architectural responsibilities:

```
app/src/main/java/com/example/healthcaremanager/
 data/
    entity/          # User, Appointment, MedicalRecord, MedicationReminder
    dao/             # UserDao, AppointmentDao, MedicalRecordDao, MedicationReminderDao
    repository/      # Repository layer
    Converters.kt    # Enum/string conversion for Room
    HealthCareDatabase.kt
    UserPreferences.kt
 viewmodel/
    AuthViewModel.kt
    AppointmentViewModel.kt
    MedicalRecordViewModel.kt
    MedicationReminderViewModel.kt
 ui/
    screens/
    theme/
 navigation/
    Screen.kt
    AppNavigation.kt
 worker/
    MedicationReminderWorker.kt
 MainActivity.kt
```

A representative screen set includes login/registration, role-specific home screens, booking and appointment list screens, medical record screens, reminder management screens, and a health tips screen.

6 Setup and Execution (How to Run It)

The project is intended to be opened and executed through Android Studio.

6.1 Prerequisites

- Android Studio (Hedgehog or later; latest recommended)
- JDK 11 or higher
- Android SDK API 26+

6.2 Practical Setup Steps

1. Open Android Studio and verify the configured JDK path (File → Project Structure → SDK Location).
2. Open the project directory.

3. Sync Gradle when prompted and wait for dependencies to download.
4. Run on an emulator (API 26+) or a physical device (USB debugging enabled).

If you encounter issues, typical fixes include cleaning/rebuilding the project, installing missing SDK components, or invalidating IDE caches for Gradle inconsistencies.

6.3 What "Working" Looks Like

A correct setup is indicated by (i) successful Gradle sync, (ii) successful build, and (iii) an app launch that reaches the login screen and allows role-based registration and login.

7 Operational Validation (Suggested Test Scenarios)

A compact validation sequence is:

1. Register two accounts: one Patient and one Doctor.
2. With the Patient, create an appointment request.
3. With the Doctor, approve (or reject) the request and observe status changes.
4. Mark the appointment as completed.
5. Add a medical record entry from the Doctor account and verify that it appears for the Patient.
6. Create a medication reminder and verify that notifications can trigger (Android 8.0+ channels required).