Group 2

MediPal
Software Architecture Document

Version 2.0

Revision History

Date	Version	Description	Author
07/07/2025	1.0	Introduction, Goals & Constraints	Nguyễn Kiến Hào
08/07/2025	1.0	Use-Case Model	Nguyễn Công Toàn
08/07/2025	1.0	Component: Database	Trần Hữu Lượng & Nguyễn Kiến Hào
10/07/2025	1.0	Architecture Choice, Architecture Diagram, Component Overview	Trần Anh Khoa
12/07/2025	1.0	Component: View Layer, ViewModel Layer, Domain Layer, Data Layer, External Services	Trần Anh Khoa
19/07/2025	2.0	Deployment Diagram	Nguyễn Kiến Hào
21/07/2025	2.0	Implementation View	Trần Anh Khoa

Table of Contents

Software Architecture Document

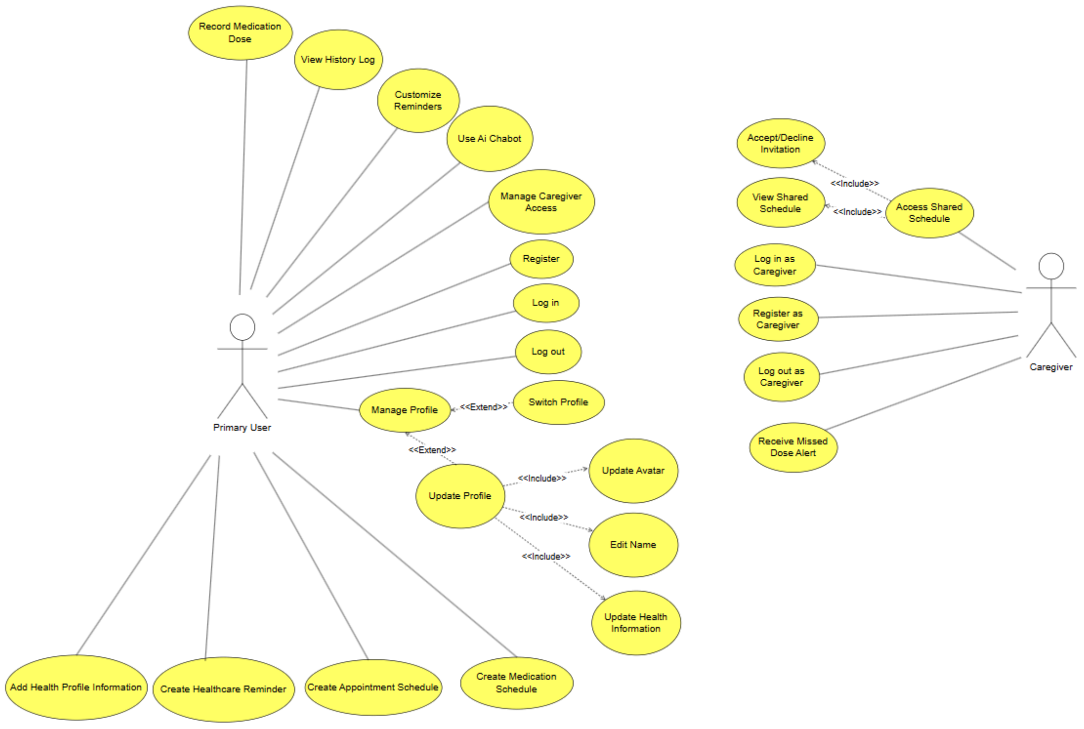
1. Introduction

- This architecture document presents the functions, attributes and features of the project at hand in a systematic and structured way
- This document is intended to be used as a reference for all team members working on the project
- Below will be the components of the software MediPal, specifically the purpose of each component within the software and their relationships with each other, as well as each component's specific attributes and how they should be implemented

2. Architectural Goals and Constraints

- The application will be developed for **Android OS (version 9.0 and above).**
- The user interface will be designed to be **simple and intuitive**, ensuring that users with no technical background can navigate without guidance.
- The app must operate reliably, with **less than 1% crash rate** during normal usage. In case of failure (e.g., app crash), unsaved user data should be preserved or auto-recovered.
- User data (medications, schedules, profiles) must be securely stored using **local encryption** on the device, and multi-user profiles must be **separated and protected** by PIN or biometric authentication (if supported by the device).
- Designed for use on smartphones in varying conditions, including **offline** environments (reminders should still function without internet).

3. Use-Case Model



4. Logical View

a) Architecture choice

MediPal's architecture is built by combining **Model-View-ViewModel (MVVM)** with **Clean Architecture** principles, allowing for a clear separation of concerns, modularity, and testability across the app.

- **Model-View-ViewModel (MVVM):**
  - To ensure maintainability, testability, and a clean separation between UI and business logic, MediPal adopts the **MVVM architecture pattern** in its Presentation Layer. This pattern divides the app's front-end logic into **three distinct components**, each with a focused responsibility.
  - **View Layer:**
    - **Location**: On-device, implemented using **Jetpack Compose**
    - **Technology: Composable functions (e.g.,** MedicationScreen**,** LoginScreen**)**
    - **Responsibilities:**

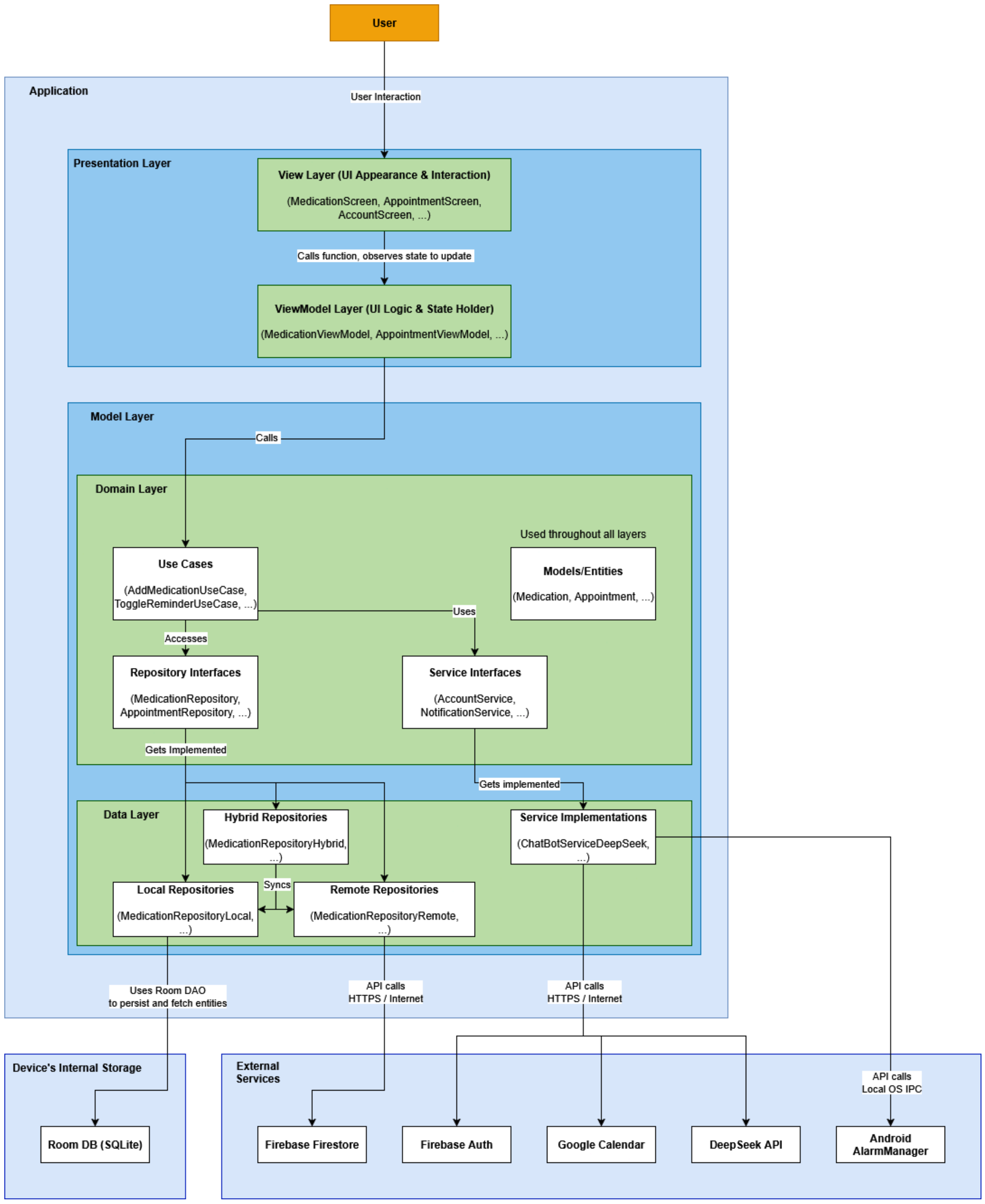
        - Renders UI to the user
        - Observes reactive state (StateFlow or LiveData) from the ViewModel
        - Sends user events (e.g., clicks, input changes) to the ViewModel
    - **Notes:**
        - Views are completely stateless; they render **based on state**
        - No business logic or direct repository access
        - Tied closely to the UI toolkit (Compose), but easily testable via previews and instrumentation
- **ViewModel Layer:**
    - **Location**: On-device, sits between the View and the domain logic
    - **Technology**: ViewModel classes using StateFlow, Kotlin coroutines
    - **Responsibilities:**
        - Holds and manages UI state (e.g., medications: StateFlow<List<Medication>>)
        - Responds to user events from the View
        - Calls **UseCase** classes (application logic) to perform actions
        - Acts as the **middle layer** that knows how to talk to the domain, but not how things are displayed
- **Model Layer:**
    - **Location**: On-device, part of Clean Architecture's domain + data layers
    - **Technology**: Domain Models (Medication, User), UseCases, Repositories
    - **Responsibilities:**
        - Provide the data and business logic to the ViewModel
        - Contain reusable logic (e.g., scheduling reminders, validating input)
    - **Components:**
        - **UseCases** (e.g., AddMedicationUseCase) → called by ViewModel
        - **Repositories** (e.g., MedicationRepository) → injected into UseCases
        - **Entities / DTOs** → represent structured data from database or network

- **Clean Architecture**:
    - To maximize testability, maintainability, and independence from frameworks, the system is organized into **concentric layers**. Each inner layer defines what the software must do, while every outer layer decides how it will be done. **All dependencies point inward**—outer layers know about, and depend on, the inner layers, never the reverse.

b) Means of communication

Service	Protocol	Notes
FirebaseAuth	HTTPS / Internet	Credential management for login / register
FirebaseFirestore	HTTPS / Internet	Remote medication, appointment, profile sync
Google Calendar API	HTTPS / Internet	Optional appointment export
DeepSeek API	HTTPS / Internet	Medication Q&A chatbot
Android Notification System	Local OS IPC	Offline scheduling of local reminder notifications

c) Architecture Diagram



User
Application
User Interaction
Presentation Layer
View Layer (UI Appearance & Interaction)
(MedicationScreen, AppointmentScreen, AccountScreen, ...)
Calls function, observes state to update
ViewModel Layer (UI Logic & State Holder)
(MedicationViewModel, AppointmentViewModel, ...)
Model Layer
Calls
Domain Layer
Used throughout all layers
Use Cases
(AddMedicationUseCase, ToggleReminderUseCase, ...)
Models/Entities
(Medication, Appointment, ...)
Uses
Accesses
Repository Interfaces
(MedicationRepository, AppointmentRepository, ...)
Service Interfaces
(AccountService, NotificationService, ...)
Gets Implemented
Gets implemented
Data Layer
Hybrid Repositories
(MedicationRepositoryHybrid, ...)
Service Implementations
(ChatBotServiceDeepSeek, ...)
Syncs
Local Repositories
(MedicationRepositoryLocal, ...)
Remote Repositories
(MedicationRepositoryRemote, ...)
Uses Room DAO to persist and fetch entities
API calls HTTPS / Internet
API calls HTTPS / Internet
Device's Internal Storage
Room DB (SQLite)
External Services
API calls Local OS IPC
Firebase Firestore
Firebase Auth
Google Calendar
DeepSeek API
Android AlarmManager

d) Component Overview

- **View Layer:**
  - **Description:**
    - The View Layer is responsible for **what the user sees and interacts with** — all the **screens**, **buttons**, **inputs**, and how they visually **react to data changes**.
    - In Jetpack Compose, the View Layer is made up of **composable functions**. These are **Kotlin functions** that describe **UI structure and behavior**, and automatically **react to state updates**.
  - **Language used:** Kotlin
  - **Components:** MedicationScreen, HistoryLogScreen, …
  - **Example flow: Toggle a reminder**
    - User taps a toggle in the MedicationItem composable.
    - That composable calls **view_model.toggle_reminder(id).**
    - ViewModel calls the appropriate **UseCase**, which hits the repository.
    - State is updated → **medications StateFlow** changes.
    - The UI re-renders automatically — no manual refresh needed.
- **ViewModel Layer:**
  - **Description:**
    - The ViewModel Layer **manages UI state**, handles **user interactions**, and **talks to the domain layer** (via use cases).
    - It connects the **View** with the **business logic**, without knowing how that logic is implemented (local DB, Firebase, ...).
  - **Language used:** Kotlin
  - **Components:** MedicationViewModel, …
  - **Example flow: Add a medication**
    - User fills out a form and taps **Save**.
    - The composable calls **MedicationViewModel.add_medication(form)**.
    - ViewModel:
      - Converts form to domain model
      - Calls **AddMedicationUseCase.invoke()**
      - Updates internal state after success
    - UI automatically updates because **medications: StateFlow<List<Medication>>** changed.
- **Model Layer**
  - **Domain Layer**:
    - **Description:** The Domain Layer is the core of the MediPal system. It defines the **business rules**, **entities**, and **use cases** that govern the application's logic, entirely **independent of platform, framework, or infrastructure**.
    - **Language used:** Kotlin
    - **Responsibilities:**
      - Enforces **business rules** independently of the presentation and data layers.
      - Enables **unit testing** of all logic without requiring Android or I/O dependencies.
      - Serves as the boundary between **ViewModel logic** and **infrastructure details**.

- **Components:**
  - **Entities (Medication, Reminder, Appointment, …):** data classes representing core business concepts. They include only fields and essential behavior, with no annotations or dependencies. These are used across multiple layers (ViewModel, UseCases, Repositories).
  - **Use Cases (AddMedicationUseCase, ExportHistoryUseCase, …):** encapsulates a specific, isolated action. It orchestrates calls to repositories and services while enforcing any business rules.
  - **Repository Interfaces (MedicationRepository, ReminderRepository, …):** Abstract interfaces that define what data access should look like.
  - **Service Interfaces (NotificationService, ChatBotService, …):** Contracts for interacting with side systems in a decoupled way.
- **Typical Workflow:**
  - A **ViewModel** receives a user action (e.g., “add medication”).
  - It calls a **UseCase** from the Domain Layer (e.g., **AddMedicationUseCase**).
  - The use case applies business rules (e.g., checking for duplicate names).
  - It delegates persistence to a **MedicationRepository** interface.
  - The actual data saving happens in the **Data Layer** (local/remote), but the Domain Layer is unaware of how.

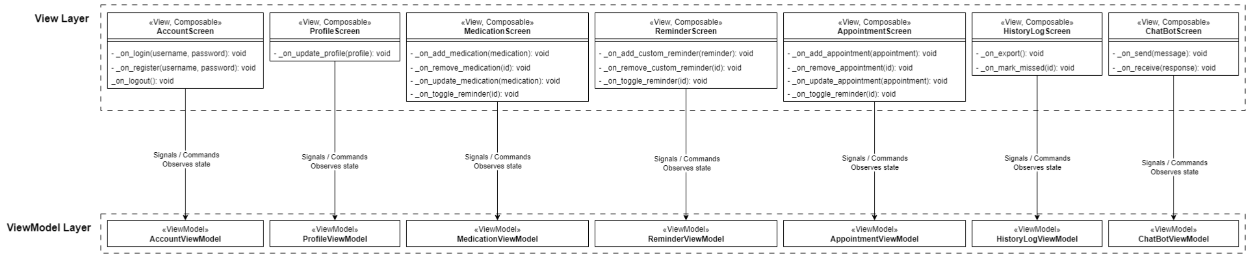
- Data Layer:
  - **Description:**
    - The Data Layer is responsible for **implementing the contracts** (repository and service interfaces) defined in the Domain Layer. It handles all interactions with **external systems** — including **local databases**, **cloud storage**, and **third-party APIs**.
    - In Clean Architecture, this layer is **infrastructure-aware** and often includes:
      - Data access (Room, Firebase, etc.)
      - API calls and DTO conversion
      - Local/remote repository implementations
      - Mappers between entity formats
  - **Language used:** Kotlin
  - **Components:**
    - **Repository Implementations**
      - **Local Repositories (MedicationRepositoryLocal, …):** Accesses Room database via DAO.
      - **Remote Repositories (MedicationRepositoryRemote, …):** Accesses Firebase Firestore and maps DTOs.
      - **Hybrid Repositories (MedicationRepositoryHybrid, …):** Combines local-first and remote-sync strategies.
    - **Service Implementations (NotificationServiceImpl, …):** These classes are tightly coupled with the Android SDK or external APIs.
    - **DAOs (MedicationDao, …):** Interface for Room queries,

marked with **@Query, @Insert**, etc.
- **Entities (MedicationEntity, …):** Database schema representation
- **Mappers:** Convert between Room Entity ↔ Domain Model and DTO ↔ Domain Model.

- **Data flow example: Get all medications**
  - **MedicationViewModel** calls **GetAllMedicationsUseCase**
  - Use case calls **MedicationRepositoryHybrid.getAllMedications()**
  - Hybrid repo returns from:
    - **MedicationRepositoryLocal**: pulls **MedicationEntity** from **MedicationDao**
    - **MedicationRepositoryRemote**: queries Firebase Firestore
    - Mapper converts **MedicationEntity → Medication** (domain)
    - Use case returns list to ViewModel
    - UI re-renders via StateFlow update

- **RoomDB:**
  - **Description:**
    - RoomDB is the **local database** layer — it stores and retrieves your data **on the device**, using SQLite under the hood.
    - It's where your app persists data **offline**, especially when:
      - The user has no internet
      - You want faster access
      - You want to avoid hitting Firebase constantly
  - **Example flow: Add a medication**
    - The user adds a medication via the UI.
    - **MedicationViewModel** calls **AddMedicationUseCase**.
    - The use case calls **MedicationRepositoryHybrid**, which delegates to **MedicationRepositoryLocal**.
    - **MedicationRepositoryLocal** calls **medicationDao.insert(medication_entity)**.
    - Room converts the entity into a SQL **INSERT** and sends it to SQLite.
    - SQLite stores it in the **.db** file on device storage.
- External Services:
  - **Description:** External services provide **infrastructure** and **specialized functionality** that MediPal **doesn't implement itself**
  - **Components:** FirebaseAuth, FirebaseFirestore, DeepSeek API, …

4.1 Component: View Layer


View Layer
«View, Composable»
AccountScreen
- _on_login(username, password): void
- _on_register(username, password): void
_on_logout(): void
«View, Composable»
ProfileScreen
- _on_update_profile(profile): void
«View, Composable»
MedicationScreen
- _on_add_medication(medication): void
- _on_remove_medication(id): void
- _on_update_medication(medication): void
- _on_toggle_reminder(id): void
«View, Composable»
ReminderScreen
- _on_add_custom_reminder(reminder): void
- _on_remove_custom_reminder(id): void
- _on_toggle_reminder(id): void
«View, Composable»
AppointmentScreen
- _on_add_appointment(appointment): void
- _on_remove_appointment(id): void
- _on_update_appointment(appointment): void
- _on_toggle_reminder(id): void
«View, Composable»
HistoryLogScreen
- _on_export(): void
- _on_mark_missed(id): void
«View, Composable»
ChatBotScreen
- _on_send(message): void
- _on_receive(response): void
Signals / Commands
Observes state
ViewModel Layer
«ViewModel»
AccountViewModel
«ViewModel»
ProfileViewModel
«ViewModel»
MedicationViewModel
«ViewModel»
ReminderViewModel
«ViewModel»
AppointmentViewModel
«ViewModel»
HistoryLogViewModel
«ViewModel»
ChatBotViewModel

The View Layer consists of **Composable functions** responsible for the display of different screens. While they are **not** classes on a technical level, they have triggers / actions that are included in the class diagram.

AccountScreen	
Serves as the main user interface for account-related actions. Handles user login, registration, and logout, dynamically showing appropriate forms based on authentication state.	
Triggers / Actions	
_on_login(username, password)	Triggered when the user submits login information. Forwards the credentials to the AccountViewModel for authentication.
_on_register(username, password)	Triggered during registration. Sends the input data to the ViewModel to create a new account.
_on_logout()	Called when the user clicks logout. Delegates to the ViewModel to clear session and auth data.

ProfileScreen	
Displays and allows editing of the user's personal profile, including full name, avatar, birth date, and gender.	
Triggers / Actions	
_on_update_profile(profile)	Triggered when the user confirms profile edits. Sends the updated info to the ProfileViewModel.

MedicationScreen
Displays the list of medications and allows users to add, remove, and toggle reminder status for each medication.

Triggers / Actions	
_on_add_medication(medication)	Requests the ViewModel to add the medication.
_on_remove_medication(id)	Requests the ViewModel to remove the medication.
_on_update_medication(medication)	Requests the ViewModel to update the medication.
_on_toggle_reminder(id)	Enables or disables the medication's reminder.

ReminderScreen	
Displays a unified list of active reminders across medications and appointments, with options to enable, disable, or delete them.	
Triggers / Actions	
_on_add_custom_reminder(reminder)	Requests the ViewModel to add a custom reminder.
_on_remove_custom_reminder(id)	Triggered when the user deletes a reminder. Informs the ViewModel to remove it.
_on_toggle_reminder(id)	Called when a user toggles a reminder switch. Updates the reminder state via the ReminderViewModel.

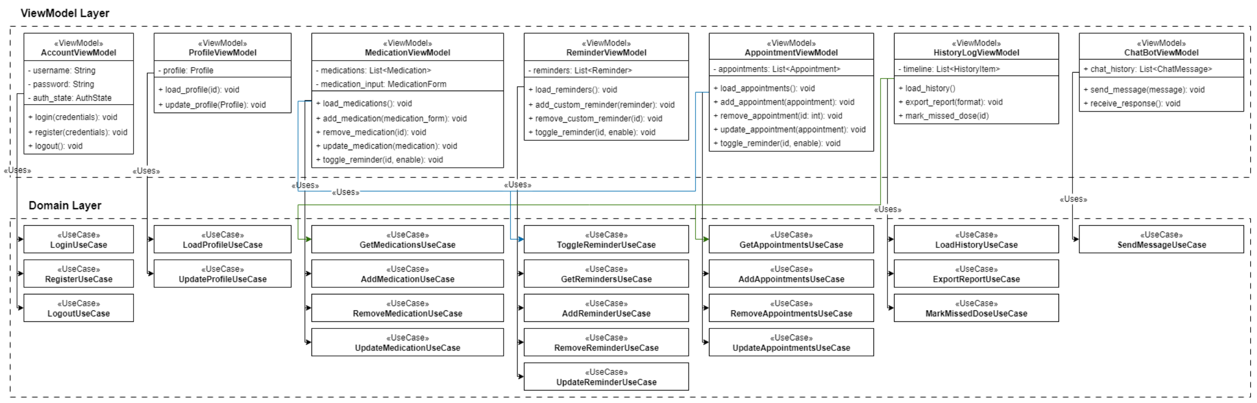
AppointmentScreen	
Allows the user to view, add, edit, and delete appointments, and optionally set reminders for them.	
Triggers / Actions	
_on_add_appointment(appointment)	Triggered when adding a new appointment. Sends the details to the AppointmentViewModel.
_on_remove_appointment(appointment)	Deletes the selected appointment.
_on_update_appointment(appointment)	Called when editing an existing appointment.
_on_toggle_reminder(id)	Enables or disables reminder notifications for the appointment.

HistoryLogScreen
Presents a timeline view of past medication doses and completed appointments. Supports export and tracking of missed doses.

Triggers / Actions	
_on_export()	Triggered when the user wants to export history data. Calls HistoryLogViewModel to generate a report in the selected format.
_on_mark_missed(id)	Marks a missed medication dose in the log.

ChatBotScreen	
Chat interface where the user can ask health-related questions. Communicates with an AI chatbot through the ChatViewModel.	
Triggers / Actions	
_on_send(message)	Called when the user submits a message. Sends it to the ViewModel, which communicates with ChatBotService.
_on_receive(response)	Triggered by the ViewModel when a chatbot reply is received, rendering it in the chat feed.

4.2 Component: ViewModel Layer



ViewModel Layer
«ViewModel» AccountViewModel
- username: String
- password: String
- auth_state: AuthState
+ login(credentials): void
+ register(credentials): void
+ logout(): void
«ViewModel» ProfileViewModel
- profile: Profile
+ load_profile(id): void
+ update_profile(Profile): void
«ViewModel» MedicationViewModel
- medications: List<Medication>
- medication_input: MedicationForm
+ load_medications(): void
+ add_medication(medication_form): void
+ remove_medication(id): void
+ update_medication(medication): void
+ toggle_reminder(id, enable): void
«ViewModel» ReminderViewModel
- reminders: List<Reminder>
+ load_reminders(): void
+ add_custom_reminder(reminder): void
+ remove_custom_reminder(id): void
+ toggle_reminder(id, enable): void
«ViewModel» AppointmentViewModel
- appointments: List<Appointment>
+ load_appointments(): void
+ add_appointment(appointment): void
+ remove_appointment(id: int): void
+ update_appointment(appointment): void
+ toggle_reminder(id, enable): void
«ViewModel» HistoryLogViewModel
- timeline: List<HistoryItem>
+ load_history()
+ export_report(format): void
+ mark_missed_dose(id)
«ViewModel» ChatBotViewModel
+ chat_history: List<ChatMessage>
+ send_message(message): void
+ receive_response(): void
«Uses»
Domain Layer
«UseCase» LoginUseCase
«UseCase» RegisterUseCase
«UseCase» LogoutUseCase
«UseCase» LoadProfileUseCase
«UseCase» UpdateProfileUseCase
«UseCase» GetMedicationsUseCase
«UseCase» AddMedicationUseCase
«UseCase» RemoveMedicationUseCase
«UseCase» UpdateMedicationUseCase
«UseCase» ToggleReminderUseCase
«UseCase» GetRemindersUseCase
«UseCase» AddReminderUseCase
«UseCase» RemoveReminderUseCase
«UseCase» UpdateReminderUseCase
«UseCase» GetAppointmentsUseCase
«UseCase» AddAppointmentsUseCase
«UseCase» RemoveAppointmentsUseCase
«UseCase» UpdateAppointmentsUseCase
«UseCase» LoadHistoryUseCase
«UseCase» ExportReportUseCase
«UseCase» MarkMissedDoseUseCase
«UseCase» SendMessageUseCase

AccountViewModel	
Handles all authentication-related logic: logging in, registering users, logging out, and tracking authentication state. It serves as a bridge between the account UI and the use cases related to user credentials.	
Methods	
login(credentials)	Calls LoginUseCase with the provided credentials and updates authState.

register(credentials)	Uses RegisterUseCase to attempt account creation.
logout()	Executes LogoutUseCase to clear session info.

ProfileViewModel	
Manages user profile data and updates, including full name, birth date, avatar, and gender.	
Methods	
load_profile()	Fetches the current user profile using LoadProfileUseCase.
update_profile(profile)	Calls UpdateUserUseCase to persist new profile data.

MedicationViewModel	
Handles all operations around managing medications: listing, adding, deleting, and toggling reminders.	
Methods	
load_medications()	Loads list from GetMedicationsUseCase.
add_medication(medication)	Triggers AddMedicationUseCase.
remove_medication(id)	Triggers RemoveMedicationUseCase.
update_medication(id)	Triggers UpdateMedicationUseCase.
toggle_reminder(id)	Calls ToggleReminderUseCase.

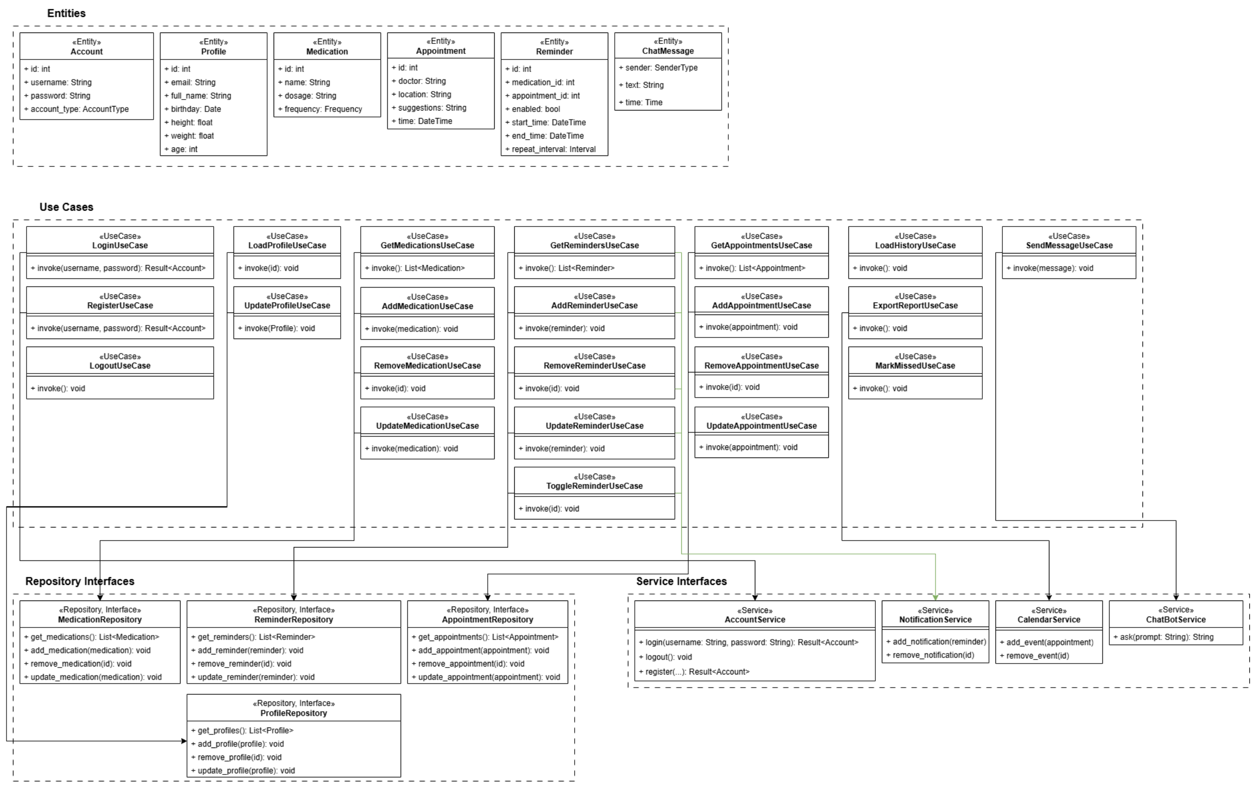
ReminderViewModel	
Central logic hub for managing active reminders, including medication and appointment reminders.	
Methods	
load_reminders()	Fetches current reminders.
add_custom_reminder(reminder)	Calls UpdateUserUseCase to persist new profile data.
remove_custom_reminder()	Uses RemoveReminderUseCase.
toggle_reminder(id)	Calls ToggleReminderUseCase with updated status.

AppointmentViewModel	
Handles all appointment-related data and user actions, including scheduling and managing appointment reminders.	
Methods	
load_appointments()	Loads list from GetAppointmentsUseCase.
add_appointment(appointment)	Triggers AddAppointmentUseCase.
remove_appointment(id)	Calls RemoveAppointmentUseCase.
update_appointment(appointment)	Triggers AddAppointmentUseCase.
toggle_reminder(id)	Updates reminder setting.

HistoryLogViewModel	
Manages history of medication and appointment actions, exposing a timeline-style dataset and allowing report exports.	
Methods	
load_history()	Aggregates logs from Medication and Appointment history sources.
export_report()	Calls ExportHistoryUseCase to generate a file.
mark_missed_dose(id)	Marks medication as missed via MarkDoseMissedUseCase.

ChatBotViewModel	
Manages user-chatbot interactions, sending user queries and handling bot replies via ChatBotService.	
Methods	
send_message(message)	Sends input to the AI and adds message to history.
receive_response(response)	Receives chatbot response and appends it to the chat feed.

4.3 Component: Domain Layer


Entities
«Entity» Account
+ id: int
+ username: String
+ password: String
+ account_type: AccountType
«Entity» Profile
+ id: int
+ email: String
+ full_name: String
+ birthday: Date
+ height: float
+ weight: float
+ age: int
«Entity» Medication
+ id: int
+ name: String
+ dosage: String
+ frequency: Frequency
«Entity» Appointment
+ id: int
+ doctor: String
+ location: String
+ suggestions: String
+ time: DateTime
«Entity» Reminder
+ id: int
+ medication_id: int
+ appointment_id: int
+ enabled: bool
+ start_time: DateTime
+ end_time: DateTime
+ repeat_interval: Interval
«Entity» ChatMessage
+ sender: SenderType
+ text: String
+ time: Time
Use Cases
«UseCase» LoginUseCase
+ invoke(username, password): Result<Account>
«UseCase» RegisterUseCase
+ invoke(username, password): Result<Account>
«UseCase» LogoutUseCase
+ invoke(): void
«UseCase» LoadProfileUseCase
+ invoke(id): void
«UseCase» UpdateProfileUseCase
+ invoke(Profile): void
«UseCase» GetMedicationsUseCase
+ invoke(): List<Medication>
«UseCase» AddMedicationUseCase
+ invoke(medication): void
«UseCase» RemoveMedicationUseCase
+ invoke(id): void
«UseCase» UpdateMedicationUseCase
+ invoke(medication): void
«UseCase» GetRemindersUseCase
+ invoke(): List<Reminder>
«UseCase» AddReminderUseCase
+ invoke(reminder): void
«UseCase» RemoveReminderUseCase
+ invoke(id): void
«UseCase» UpdateReminderUseCase
+ invoke(reminder): void
«UseCase» ToggleReminderUseCase
+ invoke(id): void
«UseCase» GetAppointmentsUseCase
+ invoke(): List<Appointment>
«UseCase» AddAppointmentUseCase
+ invoke(appointment): void
«UseCase» RemoveAppointmentUseCase
+ invoke(id): void
«UseCase» UpdateAppointmentUseCase
+ invoke(appointment): void
«UseCase» LoadHistoryUseCase
+ invoke(): void
«UseCase» ExportReportUseCase
+ invoke(): void
«UseCase» MarkMissedUseCase
+ invoke(): void
«UseCase» SendMessageUseCase
+ invoke(message): void
Repository Interfaces
«Repository, Interface» MedicationRepository
+ get_medications(): List<Medication>
+ add_medication(medication): void
+ remove_medication(id): void
+ update_medication(medication): void
«Repository, Interface» ReminderRepository
+ get_reminders(): List<Reminder>
+ add_reminder(reminder): void
+ remove_reminder(id): void
+ update_reminder(reminder): void
«Repository, Interface» AppointmentRepository
+ get_appointments(): List<Appointment>
+ add_appointment(appointment): void
+ remove_appointment(id): void
+ update_appointment(appointment): void
«Repository, Interface» ProfileRepository
+ get_profiles(): List<Profile>
+ add_profile(profile): void
+ remove_profile(id): void
+ update_profile(profile): void
Service Interfaces
«Service» AccountService
+ login(username: String, password: String): Result<Account>
+ logout(): void
+ register(...): Result<Account>
«Service» NotificationService
+ add_notification(reminder)
+ remove_notification(id)
«Service» CalendarService
+ add_event(appointment)
+ remove_event(id)
«Service» ChatBotService
+ ask(prompt: String): String

- **Entities**

Account	
Stores credentials for an account	
Attributes	
id	Identifier for the account
account_type	Account type (customer, caretaker)
profile_id	Identifier of the profile

Profile	
Stores user profile information relevant to the account and personalization.	
Attributes	
id	Identifier for the profile

email	User’s email address
full_name	User’s full name
birthday	User’s birthday
height	User’s height
weight	User’s weight

Medication	
Represents a medication in the domain — includes the information needed for scheduling and display.	
Attributes	
id	Identification for the medication
name	Medication’s name
dosage	Medication’s dosage
frequency	Medication’s intake frequency (daily, twice a day, …)

Reminder	
Encapsulates a time-based reminder tied to a medication or appointment.	
Attributes	
id	Identifier for the reminder
medication_id	Identifier for the medication / appointment
appointment_id	
enabled	Whether or not the reminder is enabled / active
start_time	The date that the reminder becomes active
end_time	The date that the reminder becomes inactive
repeat_interval	Interval between each reminder

ChatMessage

Represents a single exchange in the chatbot conversation.	
Attributes	
sender	The sender of the message
text	The content of the message
time	The time the message was sent

- Use Cases: Each Use Case class implements the logic for their corresponding use case
- Repository Interfaces

MedicationRepository / ReminderRepository / AppointmentRepository / ProfileRepository	
Defines the contract for data access.	
Methods	
get_items()	Gets the list of all the items in the database
add_item(item)	Adds a new item to the database
remove_item(id)	Remove an item from the database by its ID
update_item(item)	Update the data of an item in the database

- Service Interfaces

AccountService	
Handles user authentication	
Methods	
login(...)	Attempts to log the user in with the given credentials
register(...)	Attempts to register with the given credentials
logout()	Logs the user out of the account

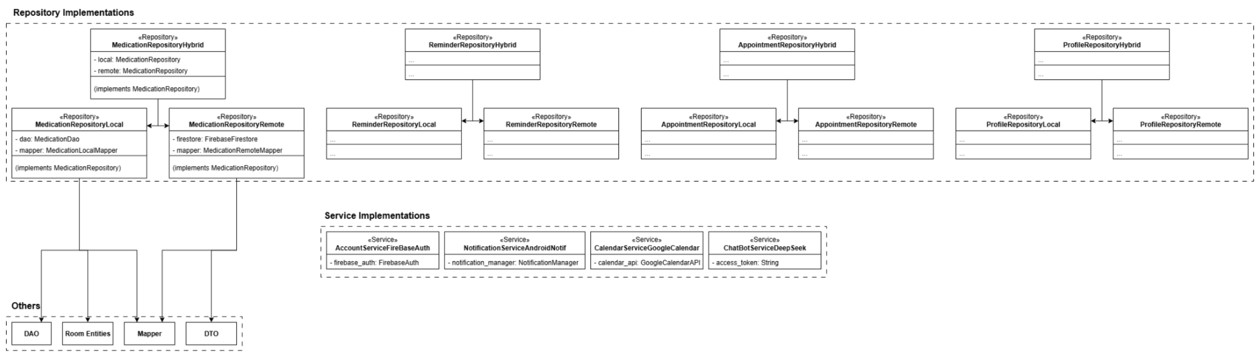
NotificationService
Pushes notifications to the user

Methods	
add_notification(reminder)	Schedules a notification
remove_notification(id)	Cancels a notification by ID

CalendarService	
Schedules events on the calendar	
Methods	
add_event(appointment)	Adds an event to the calendar
remove_event(id)	Removes an event from the calendar by ID

ChatBotService	
Provides a ChatBot for medical inquiries	
Methods	
ask(prompt)	Sends a prompt to the ChatBot

4.4 Component: Data Layer



Repository Implementations
«Repository» MedicationRepositoryHybrid
- local: MedicationRepository
- remote: MedicationRepository
(implements MedicationRepository)
«Repository» MedicationRepositoryLocal
- dao: MedicationDao
- mapper: MedicationLocalMapper
(implements MedicationRepository)
«Repository» MedicationRepositoryRemote
- firestore: FirebaseFirestore
- mapper: MedicationRemoteMapper
(implements MedicationRepository)
«Repository» ReminderRepositoryHybrid
«Repository» ReminderRepositoryLocal
«Repository» ReminderRepositoryRemote
«Repository» AppointmentRepositoryHybrid
«Repository» AppointmentRepositoryLocal
«Repository» AppointmentRepositoryRemote
«Repository» ProfileRepositoryHybrid
«Repository» ProfileRepositoryLocal
«Repository» ProfileRepositoryRemote
Service Implementations
«Service» AccountServiceFireBaseAuth
- firebase_auth: FirebaseAuth
«Service» NotificationServiceAndroidNotif
- notification_manager: NotificationManager
«Service» CalendarServiceGoogleCalendar
- calendar_api: GoogleCalendarAPI
«Service» ChatBotServiceDeepSeek
- access_token: String
Others
DAO
Room Entities
Mapper
DTO

- **Repository implementations**

MedicationRepositoryLocal / ReminderRepositoryLocal / …
Accesses databases on the device's local storage, backed by RoomDB

Attributes	
dao	Stores / retrieves data from RoomDB
mapper	Converts between Room Entities and Domain Models (Entities)

MedicationRepositoryRemote / ReminderRepositoryRemote / …	
Uses Firebase Firestore to sync or fetch medication documents	
Attributes	
firestore	Firestore object
mapper	Maps between DTOs and Domain Models (Entities)

MedicationRepositoryHybrid / ReminderRepositoryHybrid / …	
Combines both the local and remote repository, falls back to the local repository when Internet is not available, and attempts to sync it to the remote repository when Internet is available.	
Attributes	
local	Reference to the local repository
remote	Reference to the remote repository

- **Service Implementations**

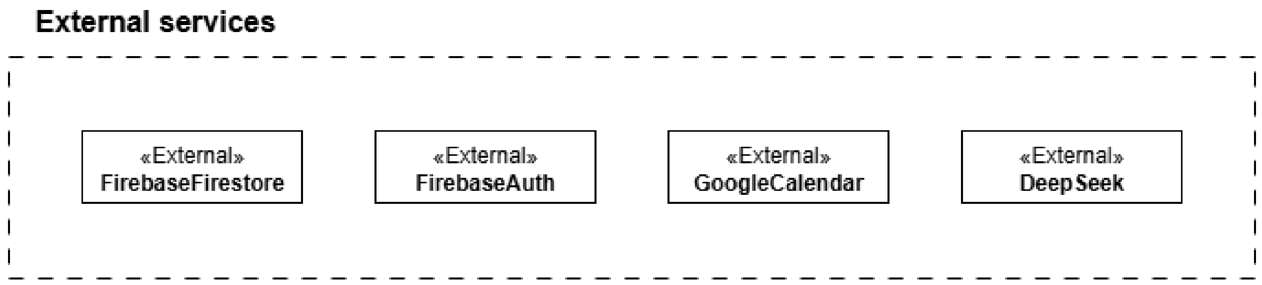
AccountServiceFirebaseAuth	
Uses Firebase Auth for authentication	
Attributes	
firebase_auth	Firebase Authentication Service

NotificationServiceAndroidNotif	
Uses android notification system to push notifications	
Attributes	
notification_manager	Android's Notification Manager

CalendarServiceGoogleCalendar	
Uses Google Calendar to schedule appointments	
Attributes	
calendar_api	Google Calendar's API

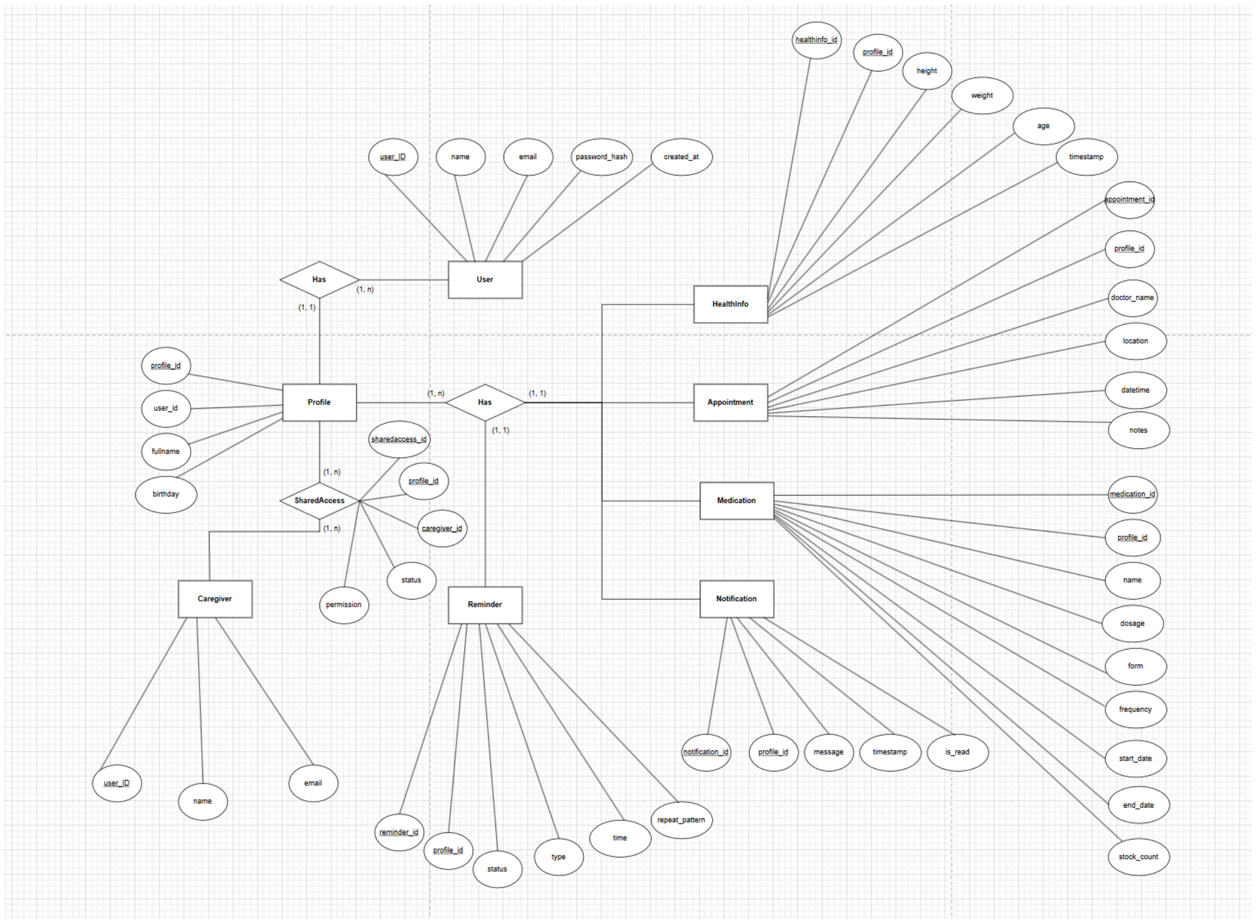
ChatBotServiceDeepSeek	
Uses DeepSeek as ChatBot	
Attributes	
access_token	Access token for DeekSeek's API

4.5 Component: External Services



Service	**Used by**	**Description**
FirebaseFirestore	MedicationRepository, ProfileRepository, …	Provides Non-SQL databases for remote access from multiple devices
FirebaseAuth	AccountService	Handles user authentication via credentials, Google account, …
GoogleCalendar	CalendarService	Allows scheduling events (medication intake, appointments, …)
DeepSeek	ChatBotService	Provides chatbot services related to the user's medical problems

4.6 Component: Database



The agreed upon code language for the app's database is SQL.

Entity explanation:

- **User:**

Attribute	Description
user_id (PK)	Unique identifier for each registered user
name	Full name of the account owner
email	Email address used for login and communication
password_hash	Encrypted password for authentication
created_at	Timestamp indicating when the account was created

- **Profile:**

Attribute	Description
profile_id (PK)	Unique identifier for a profile under a user account
user_id (FK)	References the user who owns this profile
name	Name of the person being tracked (e.g., child, parent)
birthday	Date of birth
relationship	Relationship to the user (e.g., self, father, daughter)

- **HealthInfo:**

Attribute	Description
healthinfo_id (PK)	Unique identifier for each health record
profile_id (FK)	References the profile to which the health data belongs
height	Height of the person (in cm)
weight	Weight of the person (in kg)
age	Age of the person (can be derived from DOB)
timestamp	Date and time when the health data was recorded

- **Medication:**

Attribute	Description
medication_id(PK)	Unique identifier for each medication
profile_id (FK)	References the profile taking the medication
name	Name of the medication
dosage	Dosage information (e.g., "1 pill")
form	Form of medication (pill, syrup, injection, etc.)
frequency	Frequency of use (e.g., "twice a day")
start_date	Start date of the medication
end_date	End date of the medication
stock_count	Number of remaining doses or pills

- **Reminder:**

Attribute	Description
reminder_id (PK)	Unique identifier for each reminder
profile_id (FK)	References the profile receiving the reminder
Type	Type of reminder (medication, appointment, health check, etc.)
Time	Time to trigger the reminder
Repeat_pattern	Pattern of repetition (e.g., daily, every 2 days, weekly)
status	Current status (e.g., scheduled, missed, completed)

- **Appointment:**

Attribute	Description
appointment_id (PK)	Unique identifier for each appointment
profile_id (FK)	References the profile that created the appointment
Doctor_name	The name of the doctor
Location	Location of the appointment
Datetime	Time of the appointment
notes	Optional extra notes

- **Caregiver:**

Attribute	Description
caregiver_id (PK)	Unique identifier for each caregiver
email	Email of caregiver
name	Name of caregiver

- **SharedAccess:**

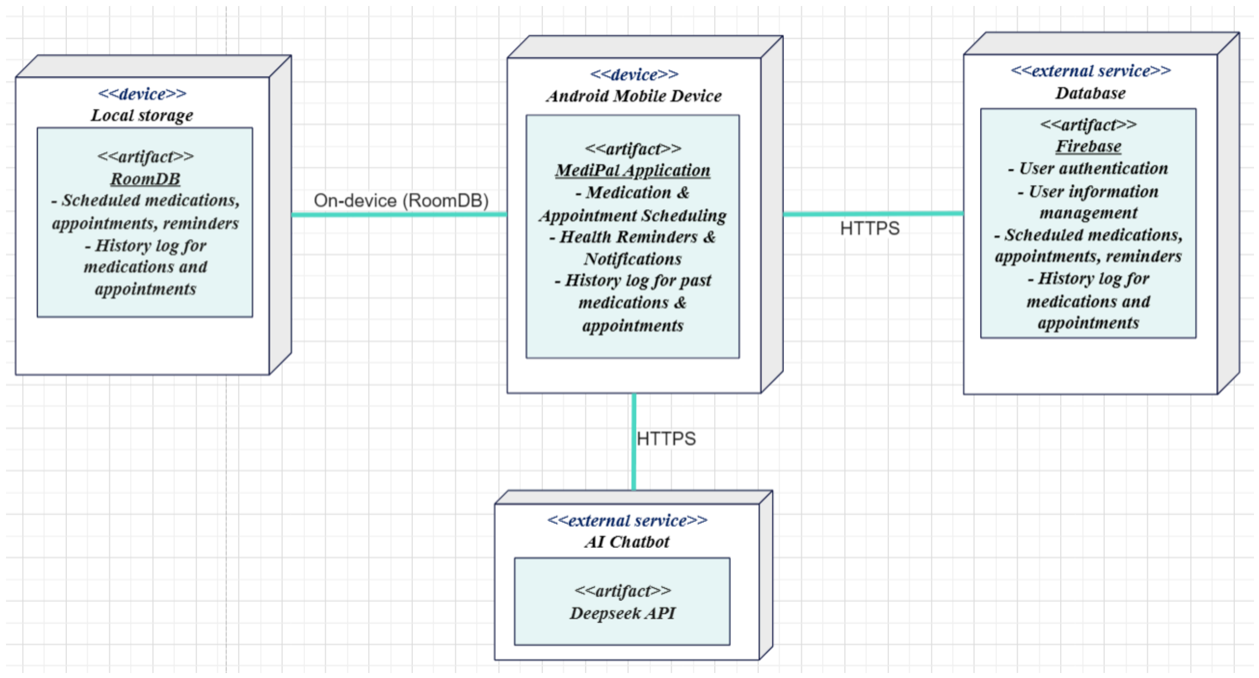
Attribute	Description
access_id (PK)	Unique identifier for each sharing record
caregiver_id (FK)	References the caregiver that has shared access
profile_id (FK)	References the client that has shared access
permission	Permission to modify (view, edit, full access,...)
status	Status include: pending, accepted, rejected

- **Notification:**

Attribute	Description
notification_id (PK)	Unique identifier for each notification
profile_id (FK)	References the client that is being sent the notification
message	Displayed message within notification
timestamp	Time sent of notification
is_read	Checks if notification has been read (true, false)

5. Deployment

- The deployment diagram below illustrates the structure of the MediPal application via physical and external nodes:


<<device>>
Local storage
<<artifact>>
RoomDB
- Scheduled medications, appointments, reminders
- History log for medications and appointments
On-device (RoomDB)
<<device>>
Android Mobile Device
<<artifact>>
MediPal Application
- Medication & Appointment Scheduling
- Health Reminders & Notifications
- History log for past medications & appointments
HTTPS
<<external service>>
Database
<<artifact>>
Firebase
- User authentication
- User information management
- Scheduled medications, appointments, reminders
- History log for medications and appointments
HTTPS
<<external service>>
AI Chatbot
<<artifact>>
Deepseek API

- **Android Mobile Device:**
  - Hosts the MediPal application, which handles:
    - ➢ Medication, appointment & reminder scheduling
    - ➢ Notifications regarding medications, appointments & reminders
    - ➢ History log of past medications & reminders

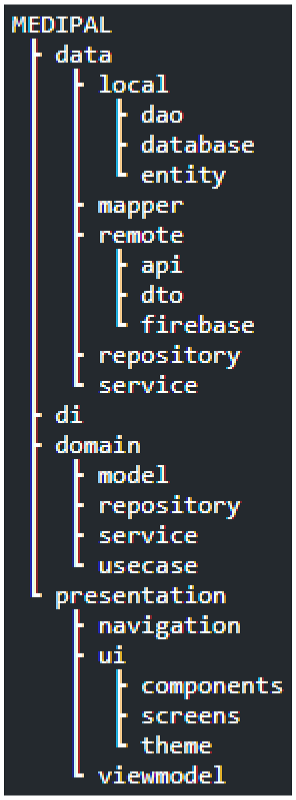
- **Local storage:**
  - Stores medication, appointment, reminder & history log data locally on user's device
  - Accessed from the user's device via RoomDB

- **Database:**
  - An external database accessed via HTTPS which:
    - ➢ Authenticates, stores and manages user data
    - ➢ Stores medication, appointment, reminder and history log data

- **AI Chatbot:**
  - Uses an external service, specifically Deepseek API, to provide users with an AI chatbot for inquiries about medications
  - Accessed via HTTPS

6. Implementation View

a) Folder Structure



```
MEDIPAL
├ data
│ ├ local
│ │ ├ dao
│ │ ├ database
│ │ └ entity
│ ├ mapper
│ ├ remote
│ │ ├ api
│ │ ├ dto
│ │ └ firebase
│ ├ repository
│ └ service
├ di
├ domain
│ ├ model
│ ├ repository
│ ├ service
│ └ usecase
└ presentation
  ├ navigation
  ├ ui
  │ ├ components
  │ ├ screens
  │ └ theme
  └ viewmodel
```

b) Folder Overview

- **data:** Concrete implementation of data sources. Everything here deals with how and where data is fetched or stored — SQLite, Firebase, …
  - **mapper:** Mappers that convert between: entity ↔ domain model, dto ↔ domain model
  - **local:** Data that comes from the device (e.g., Room DB, shared prefs).
    - **dao:** Interfaces to access local DB tables (e.g., Room DAOs).
    - **entity:** DB table models (Room @Entity classes).
    - **database:** RoomDB.
  - **remote:** API calls and networking
    - **api:** Retrofit interfaces or Firebase functions. Network contracts.
    - **dto:** Data Transfer Objects for API payloads. Mirrors API schema.
    - **firebase:** Firebase-specific logic (Auth, Firestore)
  - **repository:** Concrete implementations of domain.repository.* interfaces (Hybrid implementation). This is the glue layer that talks to local, remote.
  - **service:** implementations of service interfaces
- **domain:** Core business logic
  - **model:** Pure Kotlin data classes representing core app concepts (e.g., User, Medication, …). Shared across UseCases and ViewModels.
  - **repository:** Interfaces only. Describe what operations are available (fun getUser(id): User), but not how they're done.
  - **service:** Service interfaces.
  - **usecase:** One class per business rule or operation (AddMedication, RemoveAppointment, …).
- **presentation:** Everything the user interacts with.

- ○ **navigation:** Defines the full navigation graph using NavHost and composable() destinations. Each screen is connected here.
- ○ **viewmodel:** ViewModels that expose StateFlow, handle UI logic, and call UseCases.
- ○ **ui:** All the Compose-based UI code.
  - ■ **components:** Reusable, dumb UI widgets (e.g., buttons, cards, dialogs).
  - ■ **screens:** One folder per screen/feature (Home, Login, Settings).
  - ■ **theme:** Color.kt, Theme.kt, Type.kt — all theming stuff for Compose Material3.
- **di:** Dependency injection configuration. Wires up UseCases, ViewModels, Repositories, etc.