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

[**1. Introduction 4**](#_heading=h.e3k61qssjxxh)

[**2. Architectural Goals and Constraints 4**](#_heading=h.15vymeyypt1j)

[**3. Use-Case Model 5**](#_heading=h.7q44vkrq2l4i)

[**4. Logical View 5**](#_heading=h.itcc9lkri4fo)

[4.1 Component: View Layer 11](#_heading=h.x9mmjuutaoiu)

[4.2 Component: ViewModel Layer 14](#_heading=)

[4.3 Component: Domain Layer 17](#_heading=)

[4.4 Component: Data Layer 20](#_heading=)

[4.5 Component: External Services 22](#_heading=)

[4.6 Component: Database 23](#_heading=)

[**5. Deployment 26**](#_heading=h.8b0sn5wyh2ob)

[**6. Implementation View 26**](#_heading=h.niyixbfq1zt)

Software Architecture Document

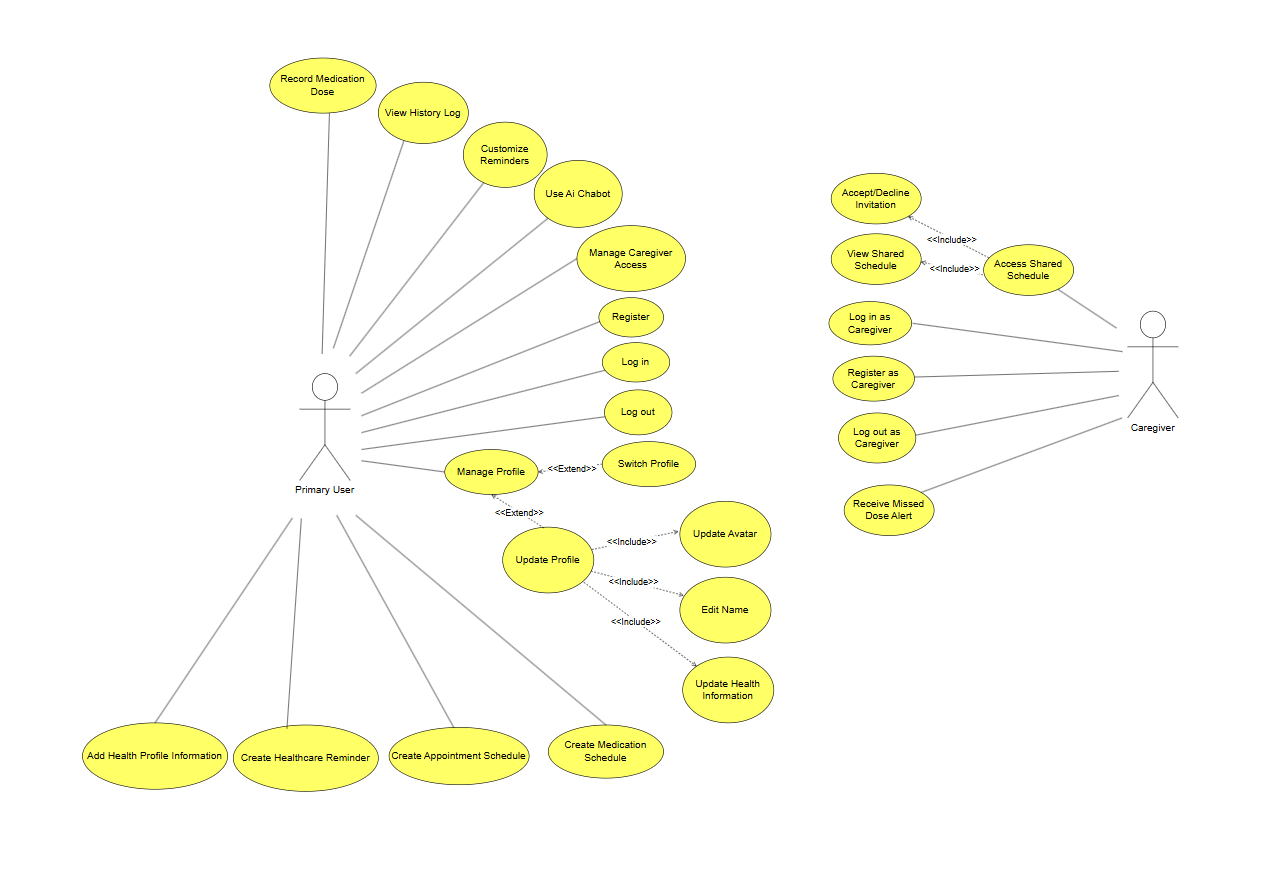
# Introduction

* 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

# 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).

# Use-Case Model

**

# Logical View

1. **Architecture choice**

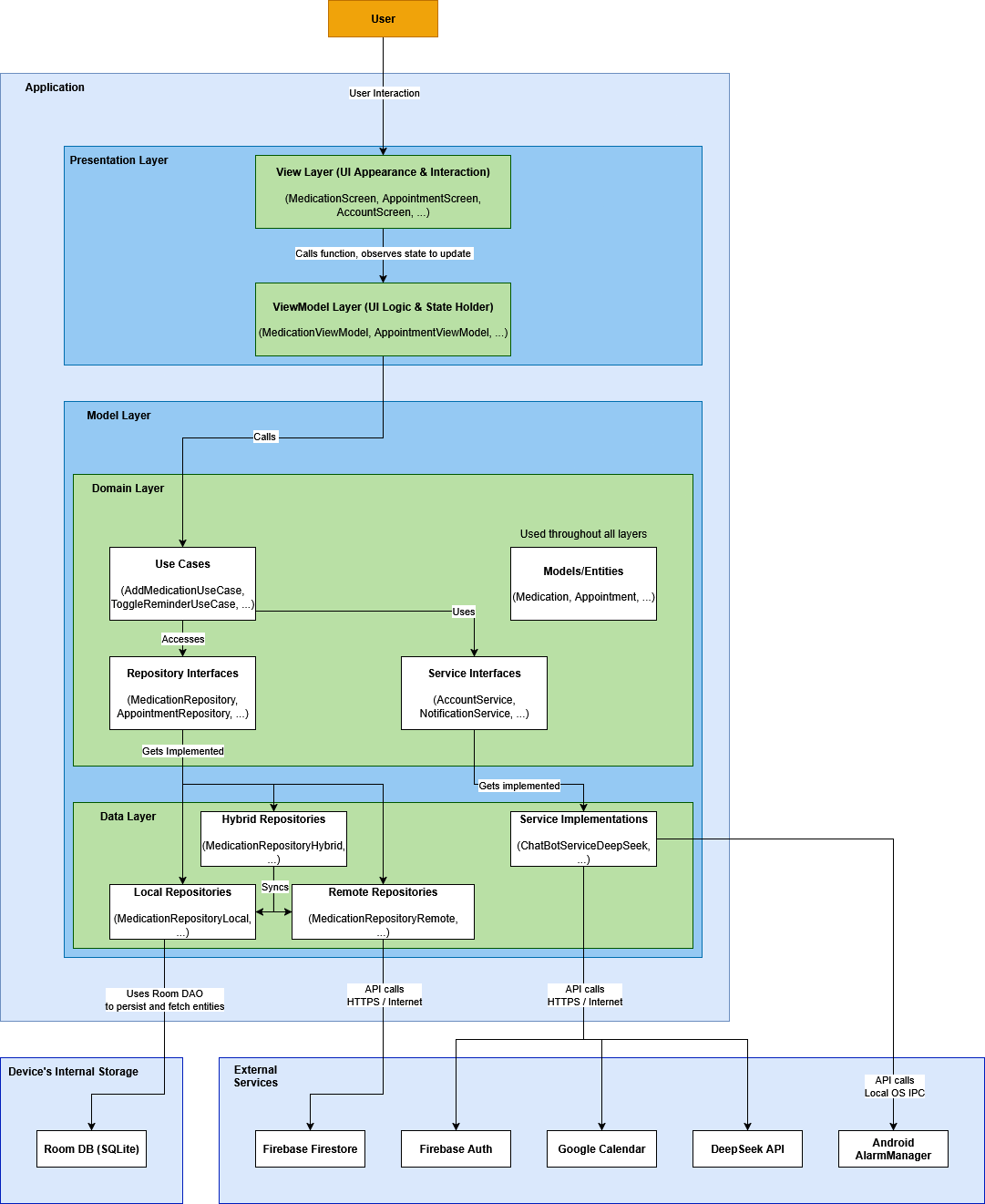
MediPal’s architecture is built by combining **Model-View-ViewModel (MVVM)** with **Clean Architecture** principles, allowing for a clear separation of concerns, modularity, and testability across the app.

* **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 (StateFloworLiveData) 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.

1. **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 |

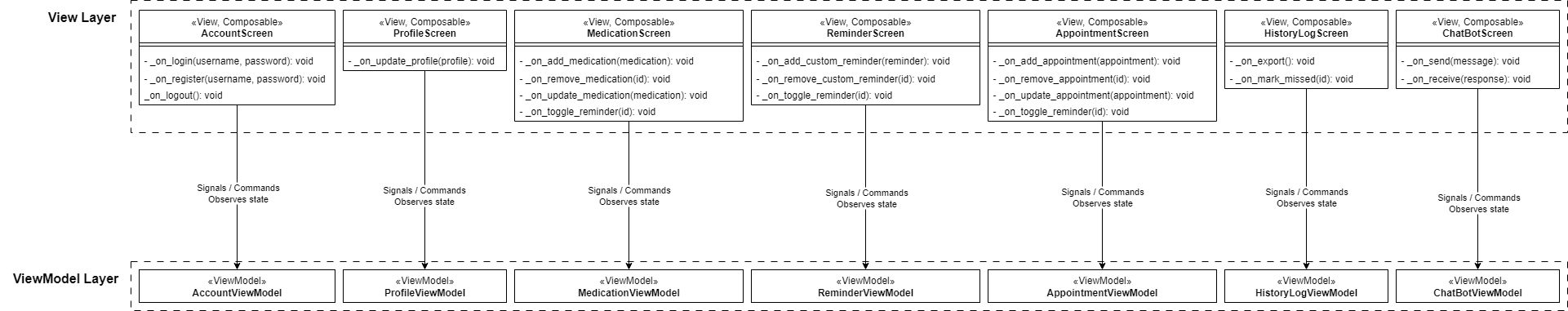
1. **Architecture Diagram**



1. **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, …

## Component: View Layer



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. |

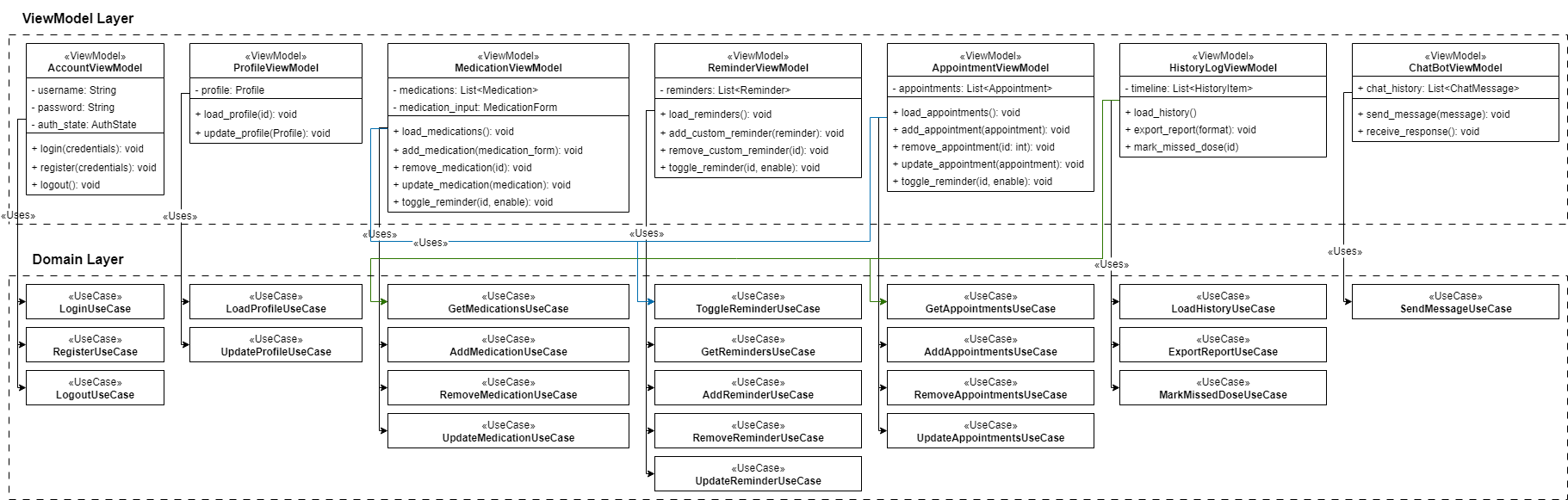
| **ReminderScreen** | |
| --- | --- |
| Displays a unified list of active reminders across medications and appointments, with options to enable, disable, or delete them. | |
| **Triggers / Actions** | |
| \_on\_add\_custom\_reminder(reminder) | Requests the ViewModel to add a custom reminder. |
| \_on\_remove\_custom\_reminder(id) | 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. |

## Component: ViewModel Layer



| **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. |

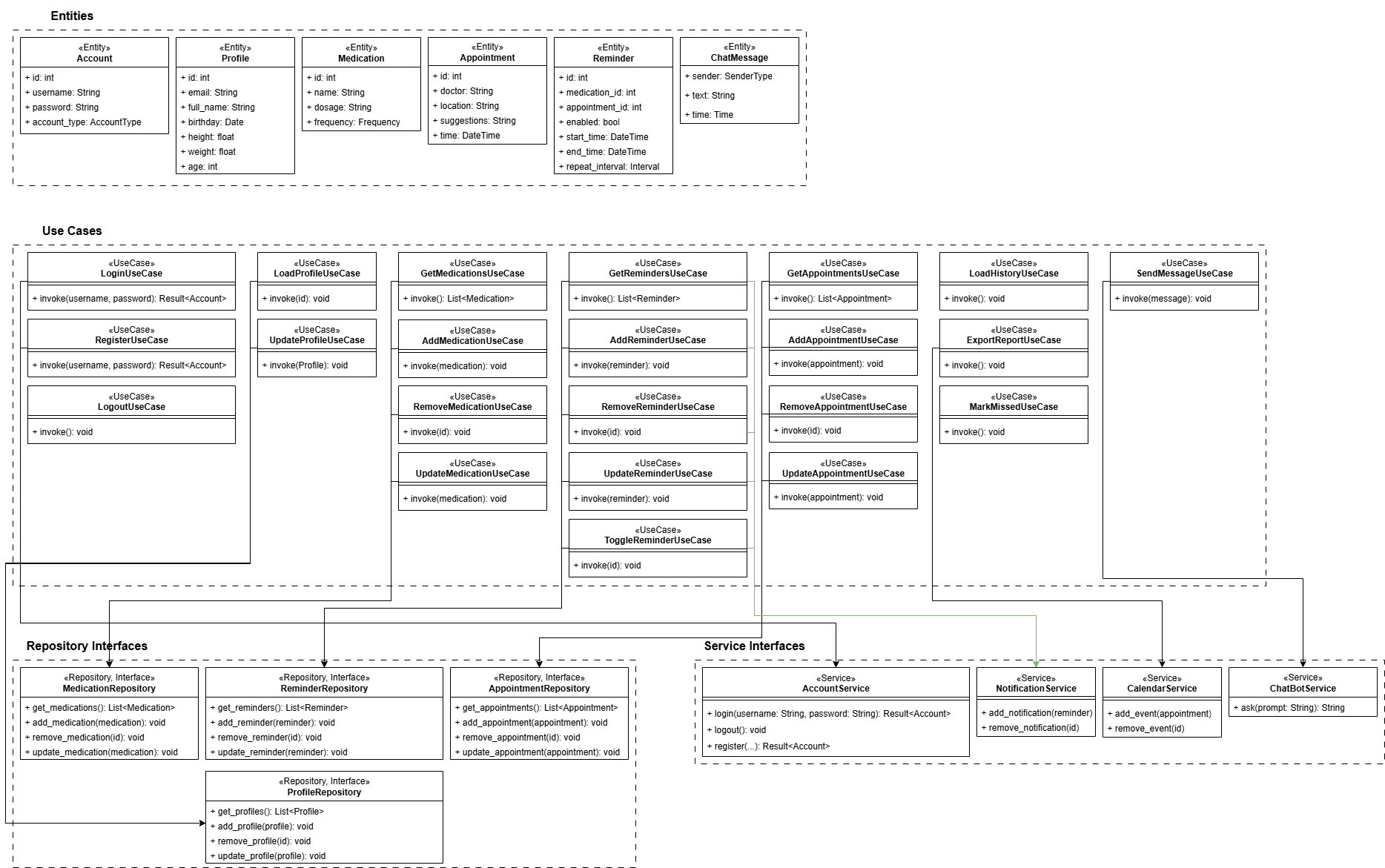
| **ReminderViewModel** | |
| --- | --- |
| Central logic hub for managing active reminders, including medication and appointment reminders. | |
| **Methods** | |
| load\_reminders() | Fetches current reminders. |
| add\_custom\_reminder(reminder) | Calls UpdateUserUseCase to persist new profile data. |
| 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. |

## Component: Domain Layer



* **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

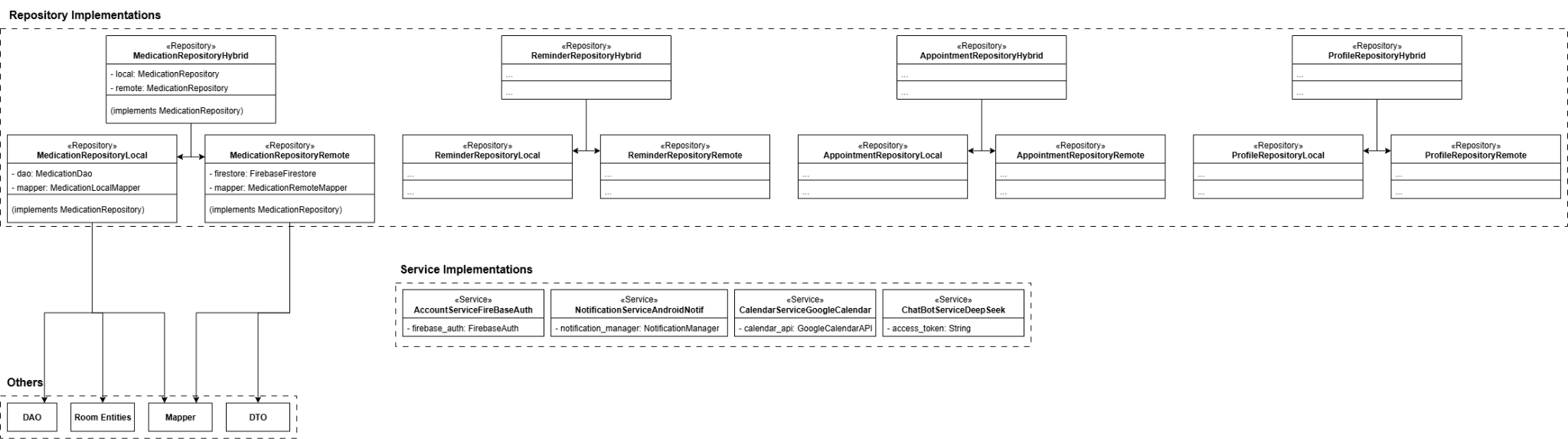
| **AccountService** | |
| --- | --- |
| Handles user authentication | |
| **Methods** | |
| login(...) | Attempts to log the user in with the given credentials |
| register(...) | Attempts to register with the given credentials |
| logout() | Logs the user out of the account |

| **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 |

## Component: Data Layer



* **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**

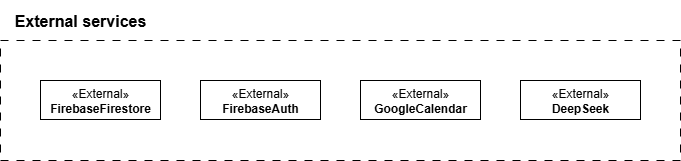
| **AccountServiceFirebaseAuth** | |
| --- | --- |
| Uses Firebase Auth for authentication | |
| **Attributes** | |
| firebase\_auth | Firebase Authentication Service |

| **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 |

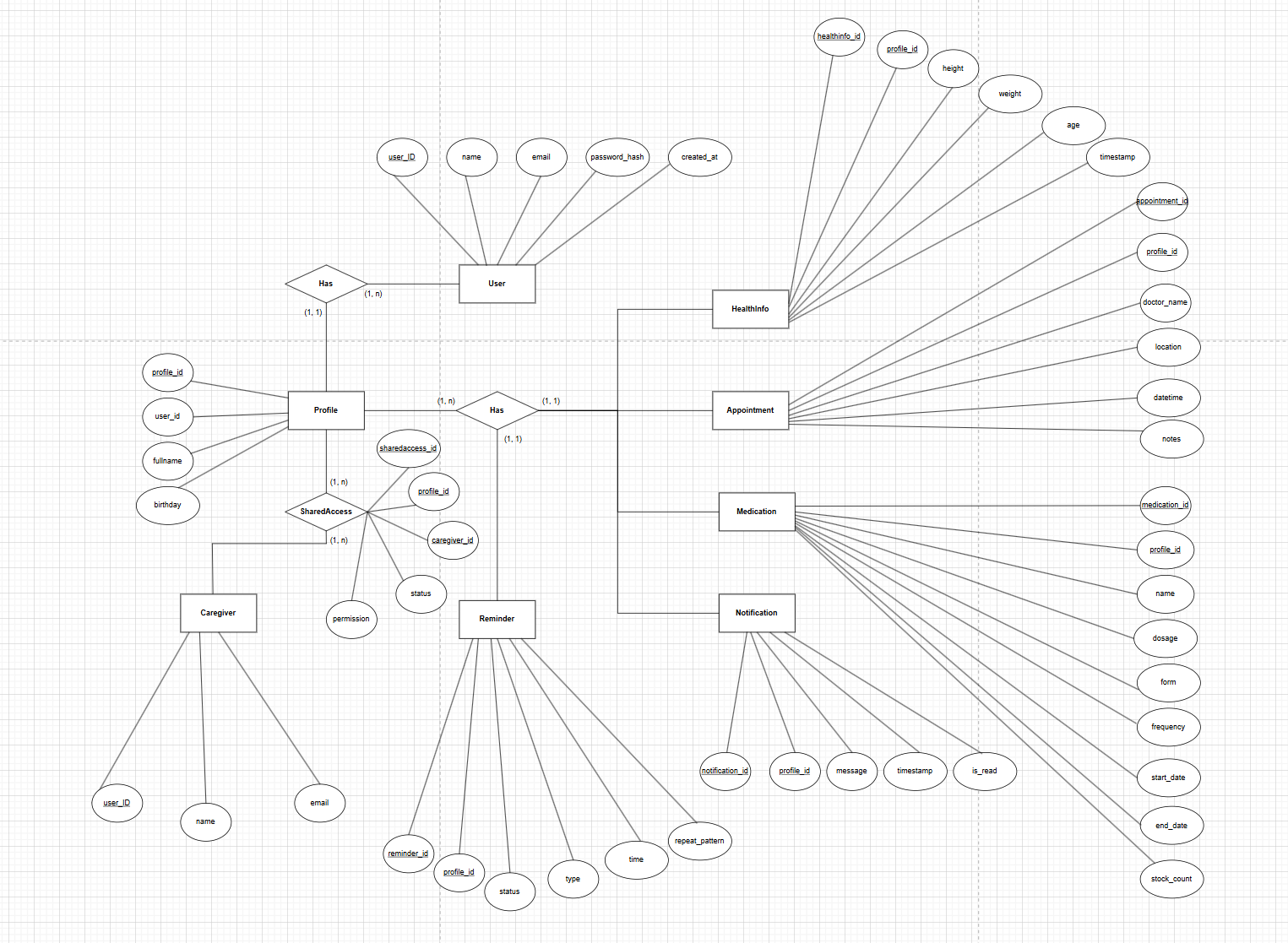
| **ChatBotServiceDeepSeek** | |
| --- | --- |
| Uses DeepSeek as ChatBot | |
| **Attributes** | |
| access\_token | Access token for DeekSeek’s API |

## 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 |

## Component: Database

**

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

**Entity explanation:**

* **User:**

| **Attribute** | **Description** |
| --- | --- |
| user\_id (PK)  name  email  password\_hash  created\_at | Unique identifier for each registered user  Full name of the account owner  Email address used for login and communication  Encrypted password for authentication  Timestamp indicating when the account was created |

* **Profile:**

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

* **HealthInfo:**

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

* **Medication:**

| **Attribute** | **Description** |
| --- | --- |
| medication\_id(PK)  profile\_id (FK)  name  dosage  form  frequency  start\_date  end\_date  stock\_count | Unique identifier for each medication  References the profile taking the medication  Name of the medication  Dosage information (e.g., "1 pill")  Form of medication (pill, syrup, injection, etc.)  Frequency of use (e.g., "twice a day")  Start date of the medication  End date of the medication  Number of remaining doses or pills |

* **Reminder:**

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

* **Appointment:**

| **Attribute** | **Description** |
| --- | --- |
| appointment\_id (PK)  profile\_id (FK)  Doctor\_name  Location  Datetime  notes | Unique identifier for each appointment  References the profile that created the appointment  The name of the doctor  Location of the appointment  Time of the appointment  Optional extra notes |

* **Caregiver:**

| **Attribute** | **Description** |
| --- | --- |
| caregiver\_id (PK)  email  name | Unique identifier for each caregiver  Email of caregiver  Name of caregiver |

* **SharedAccess:**

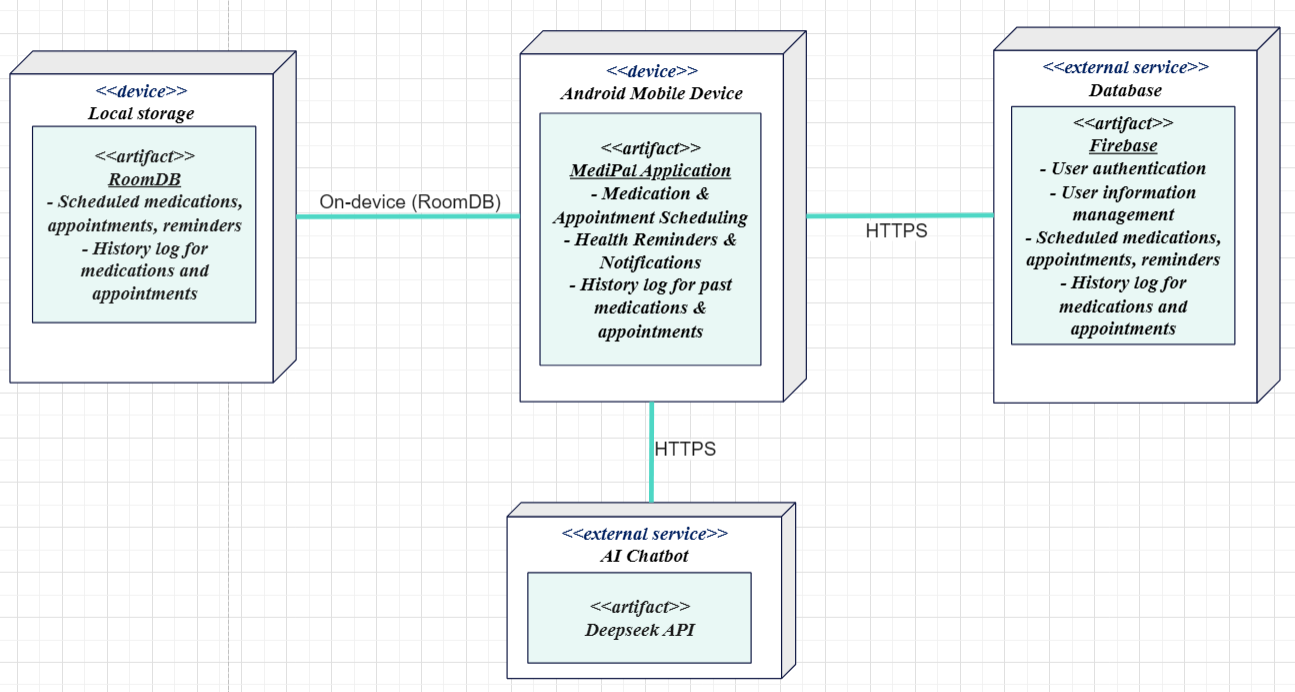
| **Attribute** | **Description** |
| --- | --- |
| access\_id (PK)  caregiver\_id (FK)  profile\_id (FK)  permission  status | Unique identifier for each sharing record  References the caregiver that has shared access  References the client that has shared access  Permission to modify (view, edit, full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

* **Notification:**

| **Attribute** | **Description** |
| --- | --- |
| notification\_id (PK)  profile\_id (FK)  message  timestamp  is\_read | Unique identifier for each notification  References the client that is being sent the notification  Displayed message within notification  Time sent of notification  Checks if notification has been read (true, false) |

# Deployment

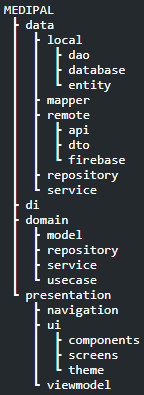
* The deployment diagram below illustrates the structure of the MediPal application via physical and external nodes:



* **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

# Implementation View

1. Folder Structure



1. 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.