People's Democratic Republic of Algeria

Ministry of Higher Education and Scientific Research

Martyr Hamma Lakhdar University of El Oued

Faculty of Exact Sciences

Department of Computer Science

Graduation Project Report

License 3rd And Final Year

Hospital Finder Mobile App

Prepared by:

- Gouder Hicham
- Guedda Alla
- Ayoub Zekri

Supervised by:

- Saci Medileh

Academic Year: 2024/2025

Contents

Abstract

In regions like El Oued, private medical clinics, healthcare complexes, and hospitals are widespread, attracting patients from both within and outside the state. However, locating these facilities, determining the shortest routes, or finding accurate addresses remains a significant challenge, especially for visitors or those unfamiliar with the area. This difficulty often delays access to timely medical care.

To address this issue, we propose the development of a mobile application designed to provide the geographic locations of healthcare institutions in **El Oued**, with the potential for future expansion to other cities.

The app will offer detailed information about these facilities and integrate mapping services to assist users in navigation. Additionally, it may include a direct communication feature for inquiries and appointment scheduling.

By combining real-time location-based services with a user-friendly interface, the application aims to simplify the process of finding healthcare facilities, improve patient navigation, and enhance overall access to medical services.

Keywords: Mobile Application, Android, Geolocation, Google Maps.

Acknowledgment

I would like to express my gratitude to everyone who contributed to the completion of this project. Their support and guidance played a significant role in its successful development.

First, I extend my sincere appreciation to my supervisor, **Saci Medileh**, for his valuable guidance, feedback, and support throughout this work. his expertise and constructive advice have been essential in refining the project and addressing challenges effectively.

Furthermore, thank the faculty members of the **Faculty of Exact Sciences** for their insights and recommendations, which have helped improve the quality of this work.

I also like to acknowledge the **administrative and technical staff** at the faculty for providing necessary resources and assistance throughout the process.

I also, I appreciate the efforts of my colleagues, for their collaboration and commitment. Their contributions were crucial in different stages of the project, and their teamwork helped ensure its completion.

Finally, I am grateful to my family for their continuous support and encouragement. Their patience and understanding allowed me to stay focused on this project.

1. General Introduction

1.1 Project Presentation

In today's fast-paced world, **access to healthcare services** is a fundamental need. However, finding the right hospital at the right time remains a **challenge** for many individuals. Whether it is for **emergency cases, routine check-ups, or specialized consultations**, people often struggle to locate nearby healthcare facilities that match their needs.
Traditional methods of searching for hospitals—such as **word-of-mouth recommendations or general internet searches**—are often **inefficient, time-consuming, and unreliable**. They rarely provide crucial details such as:

- **Hospital specialties**
- **Available doctors and their expertise**
- **Operating hours and emergency services**
- **Real-time availability of services**

To address these challenges, we propose the development of a **Hospital Finder Mobile Application**. This app aims to help users **quickly and efficiently** locate nearby hospitals based on various criteria such as **location, specialty, available services, and real-time availability**. By integrating **modern technologies like geolocation, search filtering, and live data updates**, our system provides an **intelligent, user-friendly, and accessible solution** for patients and healthcare professionals.

1.2 Application Objectives

The primary goal of this application is to **simplify the process of finding hospitals and healthcare facilities** while ensuring users receive the most relevant and **real-time** information. Specifically, the application aims to:

- **Improve Accessibility to Healthcare Services:** Provide an intuitive platform for users to locate hospitals, clinics, and medical specialists.
- **Enhance Patient Decision-Making:** Offer users hospital **ratings, available doctors, specialization areas, and real-time service availability**.

- **Reduce Search Time:** Optimize search and filtering functionalities to help users find the best hospital quickly.
- **Integrate Geolocation Services:** Enable real-time location tracking to suggest **the closest and most relevant medical facilities**.
- **Facilitate Patient-Hospital Communication:** Provide direct contact options such as **phone calls, appointment booking, and navigation assistance**.

By implementing these objectives, the **Hospital Finder Mobile Application** seeks to **enhance healthcare accessibility** and **reduce the time required to locate medical facilities**.

1.3 Methodology and Adopted Formalisms

To develop the **Hospital Finder Mobile Application**, we followed a structured approach, combining:

- **Theoretical Study**
- **Technical Analysis**
- **Practical Implementation**

Our methodology consists of the following key steps:

1. **Preliminary Study and Research**
   - Analyze existing hospital-finder applications and their **limitations**.
   - Identify key **user needs and expectations** through research and surveys.
2. **Requirement Specification and System Analysis**
   - Define the **functional and non-functional requirements** of the system.
   - Develop **use case diagrams, system architecture, and database design** to ensure a structured implementation.
3. **Design and Development**
   - Implement a **cross-platform mobile application**.
   - Utilize **Flutter for the front-end** to ensure compatibility with both Android and iOS.

- Store and manage data using **Firebase and a structured database modify later**.

4. **Testing and Validation**

    - Conduct rigorous **functionality, performance, and security** testing.
    - Gather **user feedback** and refine the app based on real-world usage.

5. **Deployment and Future Enhancements**

    - Deploy the application for public use, ensuring **a smooth user experience**.
    - Plan for **future updates**, including:
        - **AI-based hospital recommendations modify later**
        - **Telemedicine features**
        - **Integration with electronic health records (EHR)**

Chapter 1

Theoretical Study and Technical Choices

1.1 Introduction

In today's digital age, access to precise and reliable healthcare information is essential. Many existing applications provide hospital location services, but they often lack **detailed and accurate clinic information**. This chapter explores the theoretical aspects of our **Hospital Finder Mobile Application**, analyzes existing solutions, and presents our improved approach.

1.2 Preliminary Study

1.2.1 Study of Existing Solutions: Geolocation, Search on Google Maps

Several solutions exist for finding hospitals and clinics, but they have **significant limitations**:

- **Google Maps:** Google Maps is the most widely used mapping service, offering general hospital searches. However, it **lacks precise information on clinics**, including doctors' availability, specialties, and operating hours.
- **Hospital Finder by Darshan University:** This global application **only fetches hospital data from the Google Maps API** without allowing clinics to **add or**

modify their information. Additionally, its **user interface is outdated**, making navigation difficult.

- **Local Availability:** Currently, **no local application** provides a **dedicated hospital and clinic management system** that allows clinics to **register themselves, update details, and display real-time doctor availability**.

1.2.2 Critique of Existing Solutions and Proposed Solution

Limitations of Existing Solutions:

- No **custom clinic registration** or modifications
- Outdated UI and poor **user experience**
- No **detailed doctor profiles and schedules**
- Google Maps data is **too general** and lacks **localized clinic information**
- **Navigation issues** due to the absence of clear or guided paths to clinic locations, especially in certain areas

Proposed Solution: To overcome these limitations, our **Hospital Finder Mobile Application** introduces:

- **Enhanced Geolocation** $\rightarrow$ Provides **precise clinic locations** added by clinic administrators themselves.
- **Clinic Registration & Management** $\rightarrow$ Clinics can **create, modify, and update their information**.
- **Detailed Doctor Profiles** $\rightarrow$ Displays **doctor specialties, working hours, availability, and schedules**.
- **Nearby Hospital & Clinic Recommendations** $\rightarrow$ Suggests hospitals and clinics **based on real-time location**.
- **Integrated GPS & Path Navigation** $\rightarrow$ Provides **step-by-step directions and map-based paths to each clinic**, addressing the issue of poor accessibility in some areas.
- **Modern UI & Better User Experience** $\rightarrow$ A **new, intuitive, and visually appealing interface**.

1.3 Conclusion

The existing hospital location solutions fail to provide **detailed, precise, and user-friendly** hospital and clinic information. Our **Hospital Finder Mobile Application** bridges this gap by offering:

- A **better geolocation system** for **more accurate clinic placement**.
- The ability for **clinics to register and update their details**.
- **Comprehensive doctor profiles**, making hospital visits more efficient.
- **Map-based navigation with GPS**, enhancing access and ease of finding clinics.
- **A modern UI** with an intuitive design for easy navigation.

This enhanced system will significantly improve **healthcare accessibility** and provide a **better user experience** compared to existing solutions.

Chapter 2

Analysis and Specification of Requirements

2.1 Introduction

To develop an efficient and user-friendly **Hospital Finder Mobile Application**, it is crucial to analyze the key actors involved, their roles, and the functional and non-functional requirements. This chapter presents an in-depth study of the application's requirements and specifications.

2.2 Global Analysis of the Application

This section defines the main actors interacting with the system, along with their respective tasks and permissions. The application has four primary actors:

- **User** – Searches for hospitals and doctors.
- **Clinic** – Registers doctors and manages clinic information.
- **Doctor** – Manages schedules and appointments.
- **Admin** – Approves new clinics and ensures system integrity.

2.2.1 User Definition

A **user** is a person who enters the application to access services such as searching for nearby clinics and doctors. Users do not need special registration and can quickly access relevant information.

2.2.1.1 Main Tasks of a User

- **Find Nearby Clinics:** View their locations on the map.
- **Find Available Doctors:** Check their specialties and schedules.
- **View Clinic Schedules:** Check the availability of clinics and doctors.
- **Contact Clinics:** Get phone numbers for appointments and inquiries.

2.2.2 Clinic Definition

A **clinic** is an entity that registers in the application to add doctors and manage their data. Registration is not automatic; an admin must approve the clinic before it becomes active.

2.2.2.1 Main Tasks of a Clinic

- **Enter Clinic Information:** Name, address, phone number, specialties.
- **Wait for Admin Approval:** The clinic remains inactive until approval.
- **Manage Doctors:** Once approved, the clinic can:
  - Add doctors to the system.
  - Provide login credentials (email and password) to doctors.

2.2.3 Doctor Definition

A **doctor** is a user who receives login credentials from the clinic they work at. Doctors cannot register independently but are added by their respective clinics.

2.2.3.1 Doctor Tasks

- **Login:** Use the email and password provided by the clinic.
- **Manage Schedule:** Set availability and update consultation hours.
- **Receive Appointments:** Accept patient calls and schedule visits.

2.2.4 Admin Definition

The **admin** is responsible for approving or rejecting newly registered clinics. They have a separate admin application to manage the system efficiently.

2.2.4.1 Admin Tasks

- **Review Clinic Registration Requests:** Check clinic information for validity.
- **Approve or Reject Clinics:** Grant or deny access based on verification.
- **Ensure System Integrity:** Monitor and manage the overall platform.

2.2.5 App Users Information Table

The application collects specific information for each user type to support functionality, access control, and communication. Each user type—whether a patient, clinic, doctor, or admin—has a tailored set of required details that align with their role in the system. The table below outlines the key types of users in the application and the information stored for each to facilitate registration, authentication, and system interaction.

User Type	Stored Information
Patient / Guest	Full name, email, password, Google Login id, notification preferences.
Clinic	Clinic name, email, password, phone number, address, location coordinates, registration number and images.
Doctor	Full name, email, phone number, associated clinic and specialty, availability schedule.
Admin	Email and password only, used for platform management access.

Figure 2.2.5: App user types and the information they store.

2.2.6 App Users Roles Table

The application consists of multiple user roles, each with distinct responsibilities and functionalities. Understanding these roles is essential for ensuring smooth operation and proper management within the system. Below is a breakdown of the key roles and their main tasks:

Role	Definition	Main Tasks
User (Patient)	A person accessing the app to find clinics and doctors without registration.	• Search for nearby clinics and doctors. • View doctor specialties and clinic details. • Check clinic and doctor schedules. • Contact clinics for appointments.
Clinic	An entity that registers on the app to manage doctors and provide services. Approval by admin is required.	• Register and provide clinic details. • Wait for admin approval. • Add and manage doctors. • Provide login credentials to doctors.
Doctor	A medical professional registered under a clinic who cannot independently sign up.	• Log in with clinic-provided credentials. • Manage and update their schedule. • Receive and manage patient appointments.
Admin	The system manager responsible for approving clinics and ensuring smooth operation.	• Review and approve/reject clinic registrations. • Monitor system performance and security. • Ensure proper functionality of the platform.

Figure 2.2.6: App user roles and desception.

2.2.7 Conclusion

This document has provided an overview of the system, focusing on the roles and responsibilities of users. The Hospital Finder Mobile App ensures smooth interaction between patients, clinics, and administrators.

2.3 Functional Requirements Specification

The functional requirements define the expected behavior of the application. These include:

- **User Authentication:** Clinics, doctors, and admins must log in securely.
- **Geolocation Services:** Users can find nearby clinics and doctors with precise locations.
- **Clinic and Doctor Management:** Clinics can add doctors, and doctors can update their availability.
- **Appointment Booking:** Users can call clinics and book appointments.
- **Admin Panel:** The admin can approve or reject clinic registrations.

2.4 Non-Functional Requirements Specification

The non-functional requirements define the application's quality attributes, such as performance, security, and usability.

- **Performance:** The application should provide quick search results with minimal load time.
- **Security:** User data must be encrypted, and authentication should be secure.
- **Scalability:** The system should handle a growing number of users and clinics efficiently.
- **User-Friendly Interface:** The UI should be intuitive and responsive across all devices.

2.5 Use Case Diagrams

2.5.1 Definition of Use Case Diagrams

A **Use Case Diagram** is a visual representation that describes the interactions between users (actors) and the system. It outlines the different functionalities provided by the application and the roles of each user in relation to those functionalities.

These diagrams are part of **Unified Modeling Language (UML)** and help in understanding how users engage with the application. The primary components of a use case diagram include:

- **Actors:** Represent users or external systems interacting with the application.
- **Use Cases:** Define specific actions that users can perform in the system.
- **Associations:** Indicate the relationship between actors and their use cases.
- **System Boundary:** Defines the scope of the application by enclosing all the use cases.

The goal of these diagrams is to **provide a clear, high-level understanding** of how different actors interact with the system's functionalities, making it easier for developers, stakeholders, and designers to visualize and refine the system requirements.

2.5.2 Use Case Diagrams for the Application

Below are the key **Use Case Diagrams** representing interactions for different roles in the **Hospital Finder Mobile Application**.

2.5.2.1 Doctor Interactions

The following diagram illustrates how a doctor interacts with the application. Doctors have limited access and can only perform actions related to their schedules and their data.

Observation: In this and other diagrams, the element `<<abstract>> Do/ask Server` acts as a unified layer for all communication between the app and backend.”**Do**” refers to sending commands to perform actions, while ”**Ask**” refers to requesting information. This abstraction centralizes operations like in this diagram example: View Data which is a sending a request to our **App’s Server/Database** and managing into one scalable points, keeping diagrams clean and consistent.

- **Use Case Diagram: Doctor Using the App**

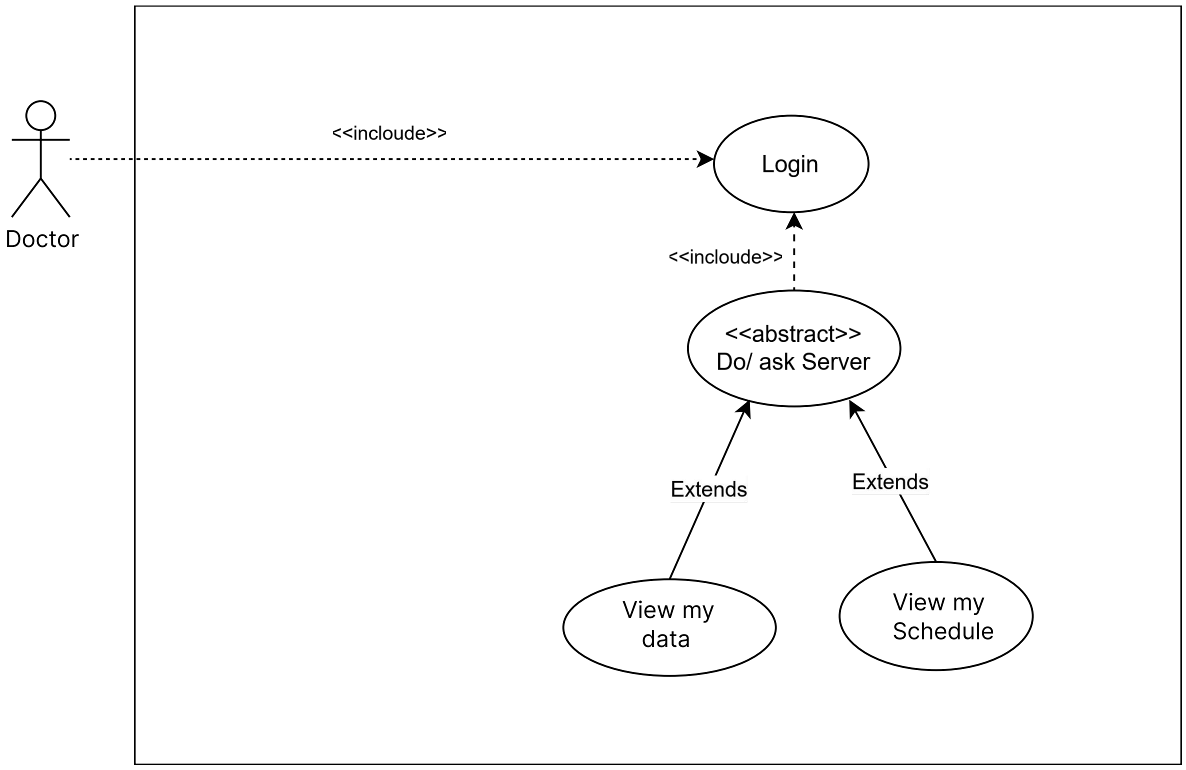


Figure 2.5.2.1: Doctor Using the App Case Diagram

2.5.2.2 User Interactions (Patients)

Users or patients use the application to find nearby clinics and doctors, check their schedules, and obtain contact details for appointments.

- **Use Case Diagram: Patient or User Using the App**

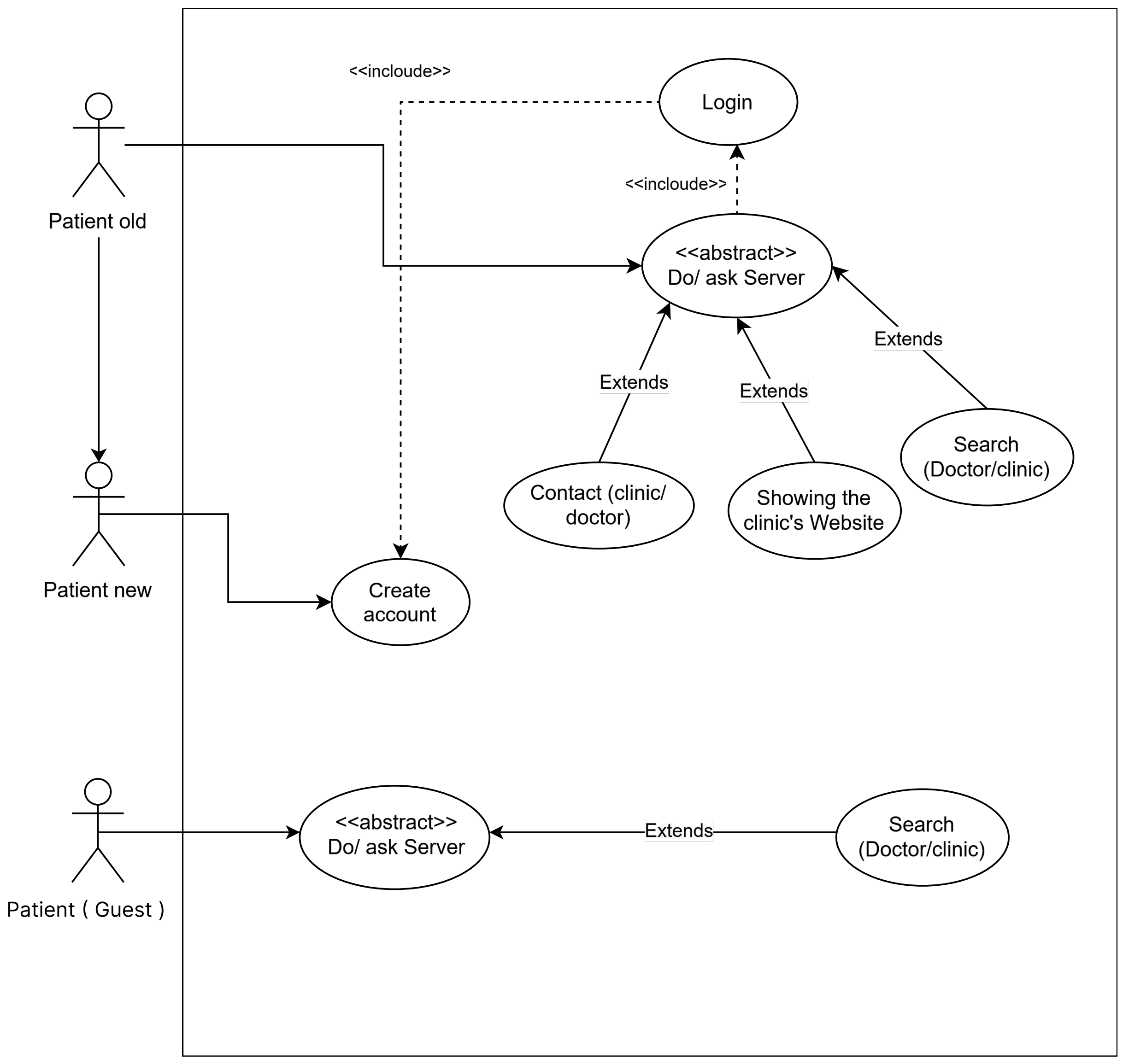


Figure 2.5.2.2: User Interactions Case Diagram

2.5.2.3 Clinic Interactions

Clinics are responsible for registering in the application, adding doctors, and managing their clinic details. They must be approved by an admin before becoming active.

- **Use Case Diagram: Clinic Using the App**

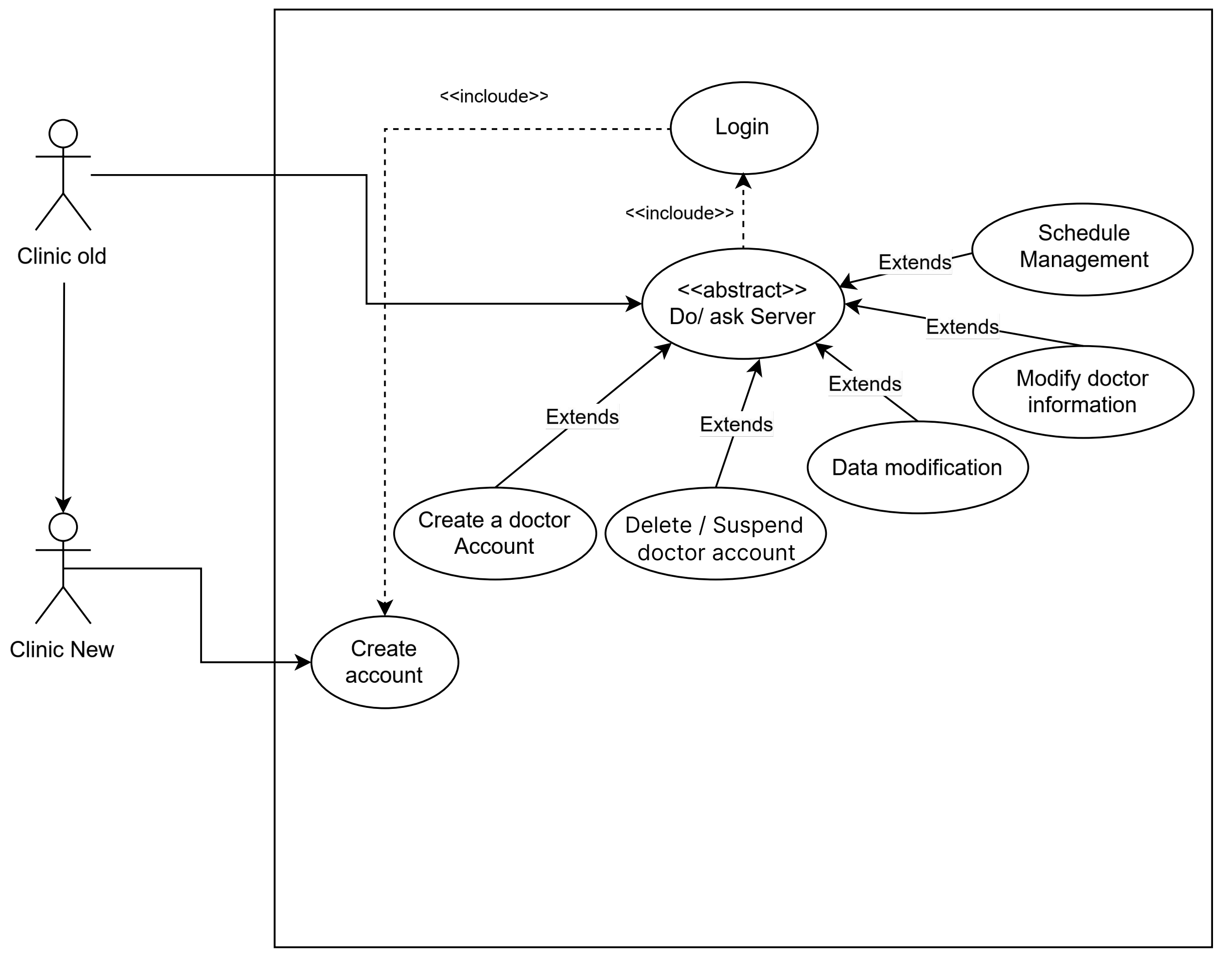


Figure 2.5.2.3: Clinic Using the App Case Diagram

2.5.2.4 Admin Interactions

Admins ensure the integrity of the platform by reviewing clinic registration requests and either accepting or rejecting them. They also monitor system performance and maintain security.

- **Use Case Diagram: Admin Using the App**

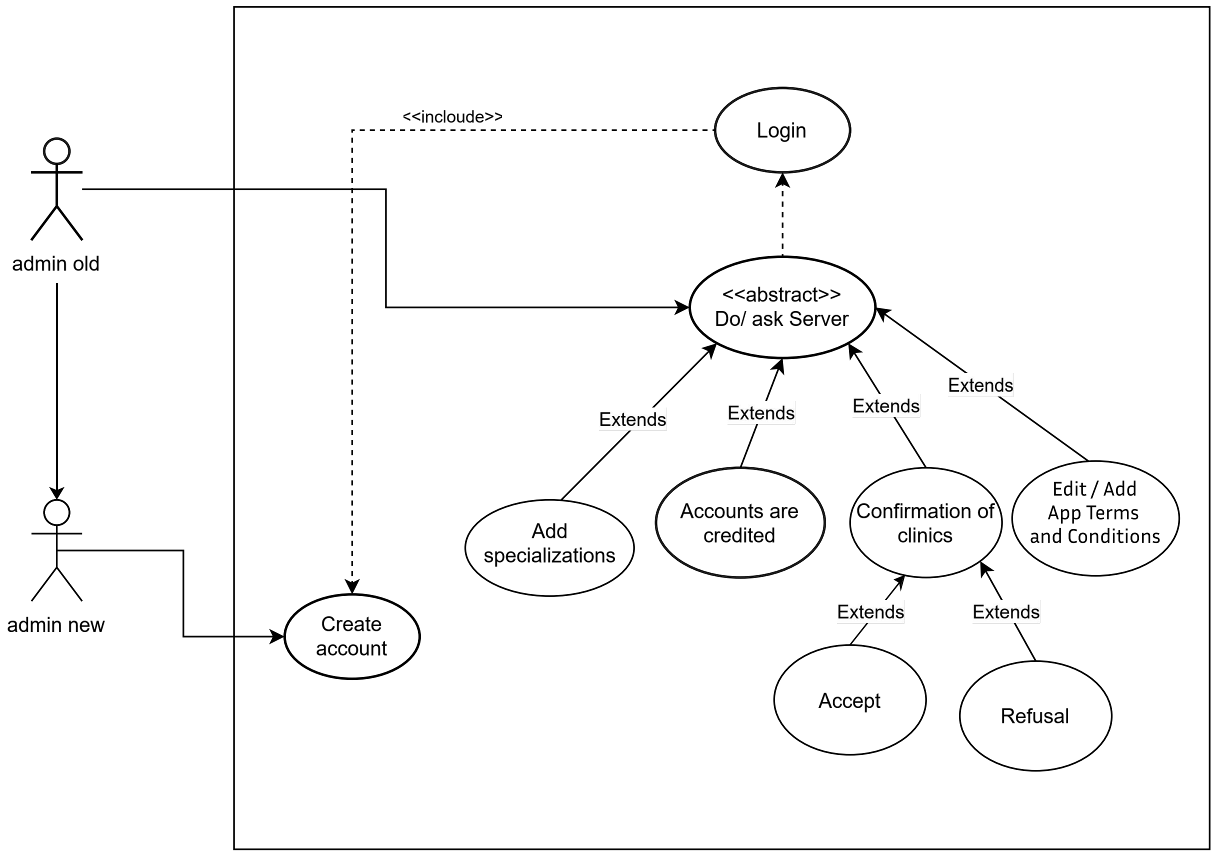


Figure 2.5.2.4: Admin Using the App Case Diagram

2.6 Conclusion

This section has provided a detailed overview of **Use Case Diagrams** and their role in defining user interactions within the **Hospital Finder Mobile Application**.

By mapping out different functionalities for doctors, users, clinics, and admins, these diagrams offer a structured approach to understanding the system's behavior.

The next steps involve designing the app structure and implementing the system based on these well-defined interactions.

In the next chapter, we will dive deeper into the app and its conception through additional diagrams, further refining the system's design and architecture.

Chapter 3

La Conception d'Application

3.1 Introduction

In this chapter, we present the design phase of our mobile application *Hospital Finder*. Following the analysis and specification of requirements, this stage is essential to define the structural and behavioral aspects of the application.
The design phase translates the identified functionalities into technical diagrams that guide implementation. These include use case diagrams, class diagrams, and sequence diagrams. Each diagram offers a different perspective, illustrating how components interact and how the system is structured internally.

3.1.1 Use Case Actors Overview

The application involves multiple actors such as:

- **User**: the main end-user searching for hospitals or clinics.
- **Clinic**: responsible for managing their own clinic profile and associated doctors.
- **Admin**: oversees the system and manages reported data or misuse.
- **Doctor**: while present in the system, the doctor does not interact directly with the application and is managed entirely by the clinic.

We will observe their interactions more thoroughly in the diagrams presented in the following sections.
A summarized view of the main actors is illustrated in the diagram below :

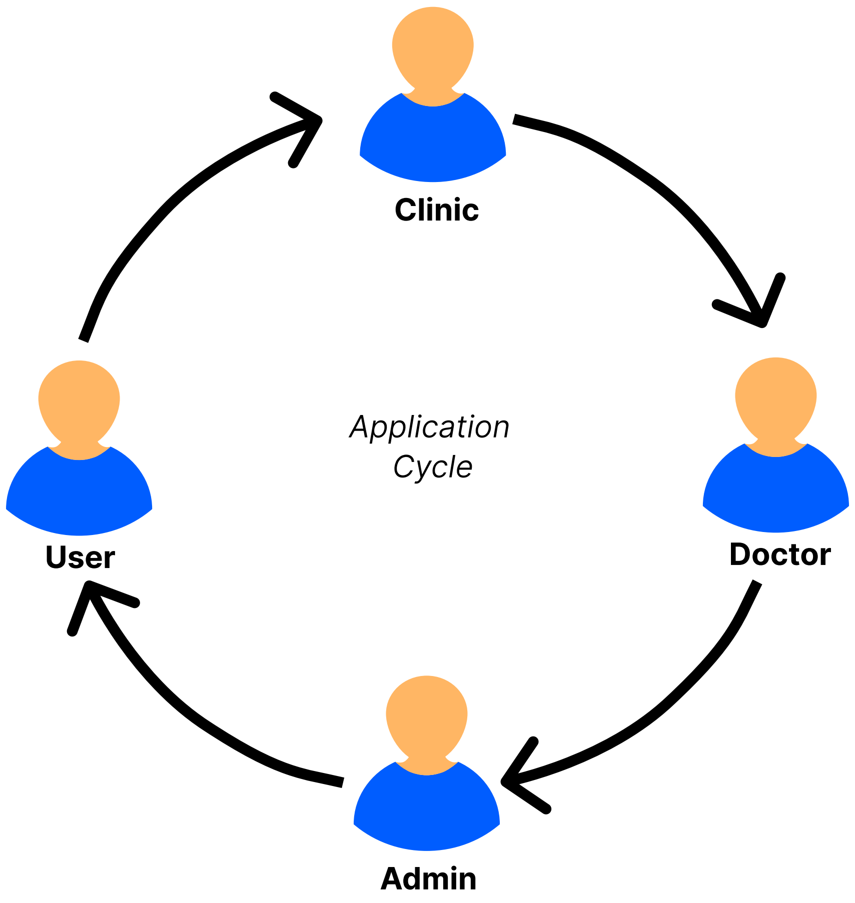


Figure 3.1: Application Cycle between App actors .

3.2 Detailed Design

This section presents the core structural design of the *Hospital Finder* application. After analyzing the system's requirements and actors, we now focus on how these elements are implemented internally.
The detailed design phase provides a clear **technical blueprint** by describing the key components, their responsibilities, and how they interact. This allows for a smooth transition from the functional analysis to the actual development.
UML (Unified Modeling Language) offers various diagrams to describe a software system from different perspectives. These diagrams are generally grouped into two main categories:

- **Structural diagrams** - describe the static aspects of the system, such as classes and their relationships.
- **Behavioral diagrams** - illustrate the dynamic behavior of the system, such as interactions and workflows between components.

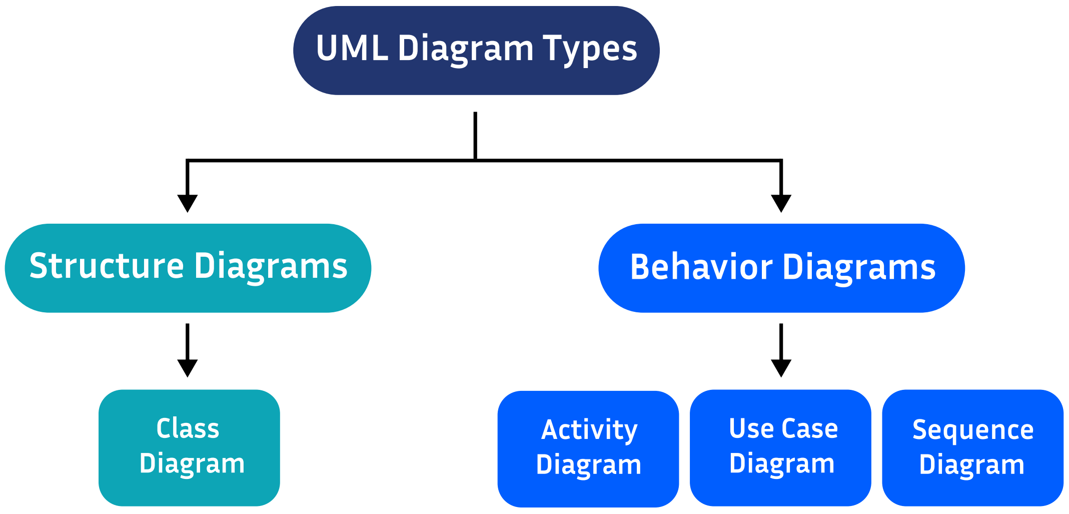


Figure 3.1: Classification of UML diagrams (structure vs behavior).

In Chapter 2, we presented a **Use Case Diagram**, which belongs to the behavioral category. In this chapter, we will explore two additional diagrams:

- The **Class Diagram**, selected from the structural diagrams.
- The **Sequence Diagram**, selected from the behavioral diagrams.

These diagrams allow us to model both the architecture and the interactions in our application in a clear and structured manner. We begin with the class diagram in the next section.

3.2.1 Class Diagram

A class diagram plays a key role in the object-oriented design of software systems. In this section, we will define what a class diagram is, explain its importance in the development process, and finally present the class diagram of our *Hospital Finder* application.

3.2.1.1 What is a Class Diagram ?

A **class diagram** is a static structural diagram that describes the types of objects in the system and the various kinds of **relationships** that exist among them. It shows the system's **classes**, their **attributes**, **methods**, and the **associations** between objects. It is one of the most commonly used diagrams in **UML** (Unified Modeling Language) and serves as a **blueprint** for the **implementation phase**.
Each class is represented as a rectangle divided into compartments. The top compartment shows the **class name**, the middle one lists the **attributes**, and the bottom lists the **operations** or **methods**.

3.2.1.2 Why Use a Class Diagram ?

The **class diagram** is essential in the software development lifecycle for the following reasons:

- It provides a visual overview of the system **structure**.
- It helps developers understand how **data** is organized and managed.
- It facilitates the identification of responsibilities and roles of each class.
- It reflects both the logical architecture and the design of the application's **database**.
- It helps in detecting potential **design flaws** early in the development process.

In the context of our project, the **class diagram** acts as a bridge between the **actors** (users, clinics, doctors, and admins) and the data handled internally. It helps clarify how the **components** of the system interact with one another and ensures a consistent and maintainable code structure.

3.2.1.3 Class Diagram of the Application

The class diagram of the *Hospital Finder* application was developed based on the functional and non-functional requirements outlined during the analysis phase. It models the following key classes:

- **users**: Stores core information for every system user. This includes:
  - `id`: Primary key.
  - `google_id`: Used when logging in with Google.

  - `name`, `email`, `password`: Basic user credentials.
  - `email_verified_at`: Timestamp indicating when the user's email was verified.
  - `user_notify_status`: Boolean indicating if notifications are enabled.
  - `fcm_token`: Used for Firebase push notifications.
  - `user_role`: Small integer representing the role type.
  - `created_at`: Account creation date.

- **role**: Defines all possible roles within the app (e.g., admin, doctor, clinic, general user). Supports role-based access control.
- **roles_user**: A pivot table allowing users to have multiple roles. Fields include:
  - `user_id`: References `users.id`.
  - `role_id`: References `role.id`.
- **specialties**: Stores medical specialties such as dermatology or pediatrics. Includes:
  - `name`: The default name (usually in Arabic).
  - `name_fr`: The name written in Latin (French) characters.
  - `specialy_img`: Optional image representing the specialty.
- **municipalities**: Represents cities or regions where clinics are located. Includes:
  - `name`: Default name.
  - `name_fr`: Latin-character version of the name.
- **clinic_schedules**: Contains working hours of clinics. Includes:
  - `clinic_id`: References the clinic.
  - `day`, `opening_time`, `closing_time`: Working day and hours.
- **Clinics**: Represents registered health facilities. Each clinic:
  - Belongs to a `user` (usually the clinic owner or admin).
  - Is located in a `municipality`.
  - Includes details like multilingual name (`name`, `name_fr`), contact info, location (latitude and longitude), and media (profile and cover images).

  - Has a `type` (e.g., clinic or hospital).
  - Has a `status` (active/inactive), registration number, and creation timestamp.

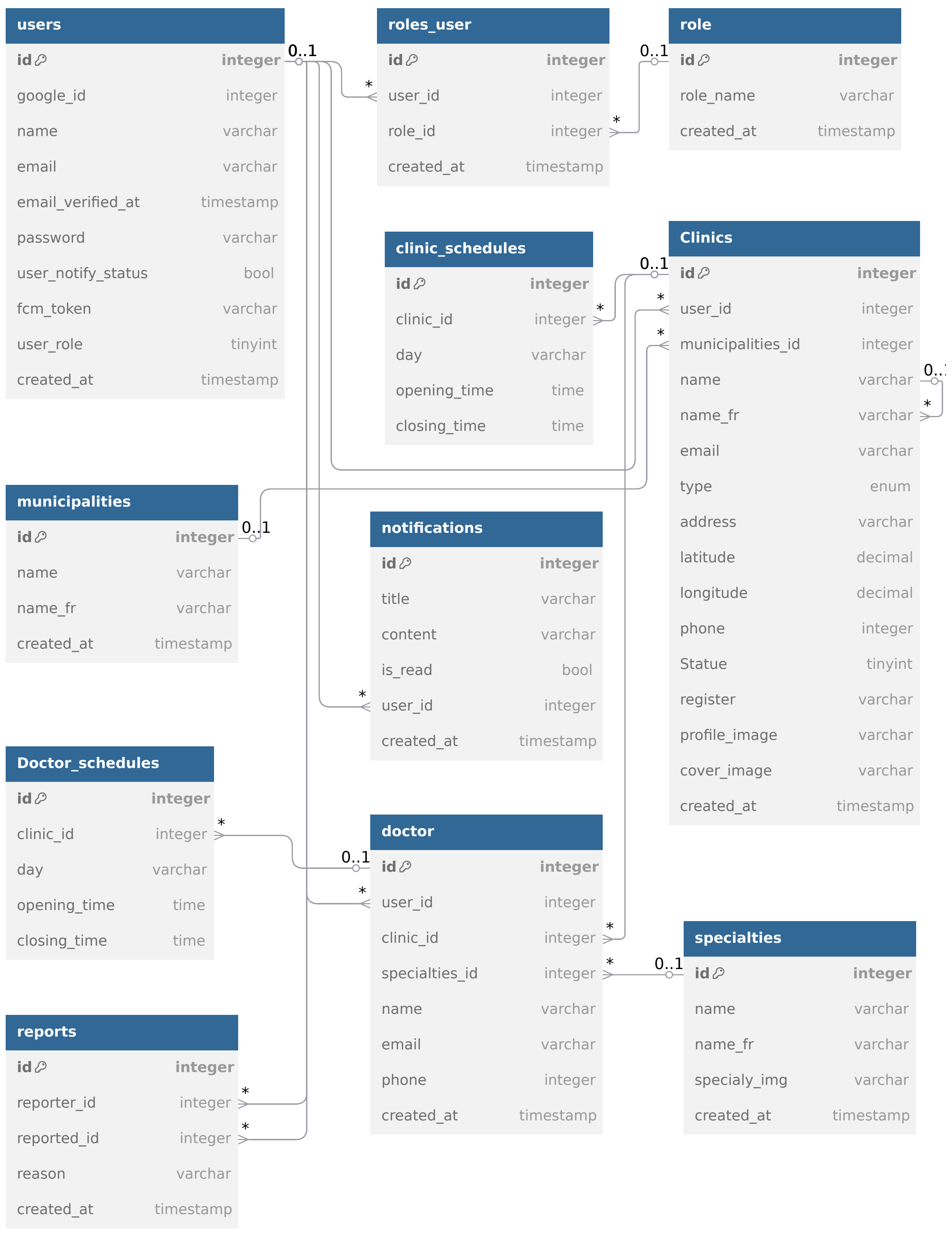
- **doctor**: Contains profiles of medical professionals. Each doctor:
  - Is a registered `user`.
  - Works at a specific `clinic`.
  - Belongs to a specific `specialty`.
  - Has personal details such as `name`, `email`, `phone`, and `created_at`.

- **Doctor_schedules**: Specifies working hours for doctors. (Note: `clinic_id` in this table seems to be misnamed and likely refers to `doctor_id`.)

- **notifications**: Stores push or in-app notifications sent to users. Includes:
  - `title`, `content`: Notification text.
  - `is_read`: Boolean indicating if the user has seen the notification.
  - `user_id`: Target recipient.

- **reports**: Enables users to report other users for various reasons. Includes:
  - `reporter_id`: The user submitting the report.
  - `reported_id`: The user being reported.
  - `reason`, `created_at`: Reason for reporting and time of submission.

The class diagram is shown below :



3.2.1.3 Figure : Class diagram of the Hospital Finder application.

3.2.2 Sequence Diagrams

3.2.2.1 Definition

Sequence diagrams are a type of UML behavioral diagram that show how objects or components in a system interact with each other over time. They describe the flow of messages and the order of operations between actors and system parts in specific scenarios.

Each sequence diagram illustrates how different entities collaborate in a particular process, using lifelines, messages, and activation bars to convey their roles and responsibilities.

3.2.2.2 Purpose and Importance

Sequence diagrams are essential for modeling **dynamic behavior** in a system. They help developers understand the exact **flow of operations**, communication between objects, and timing of events.

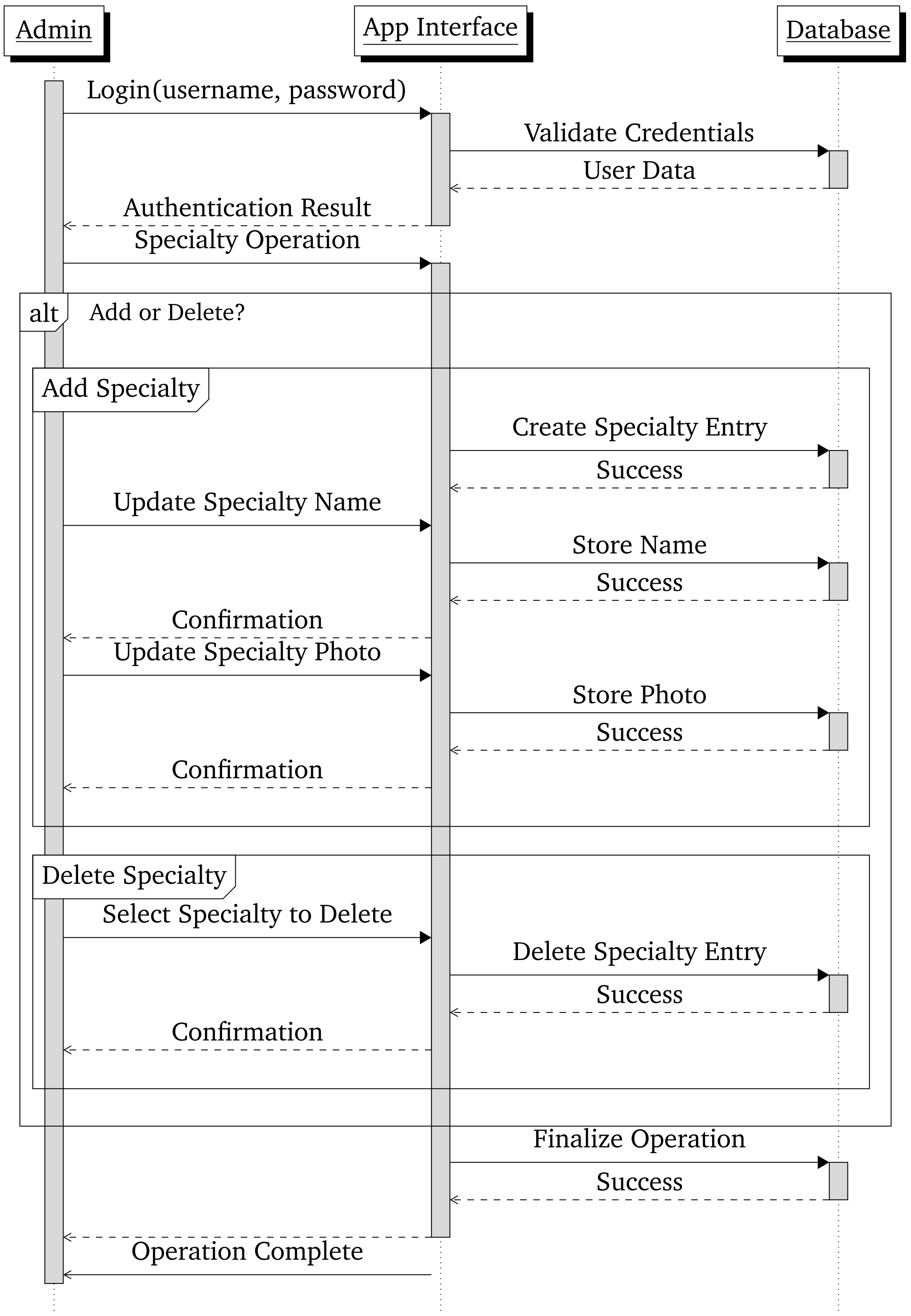
In our case, these diagrams allow us to:

- Clearly visualize **interactions** between the application interface, users (like admin), and the backend (**database**).
- Ensure that the designed processes are logically consistent and complete.

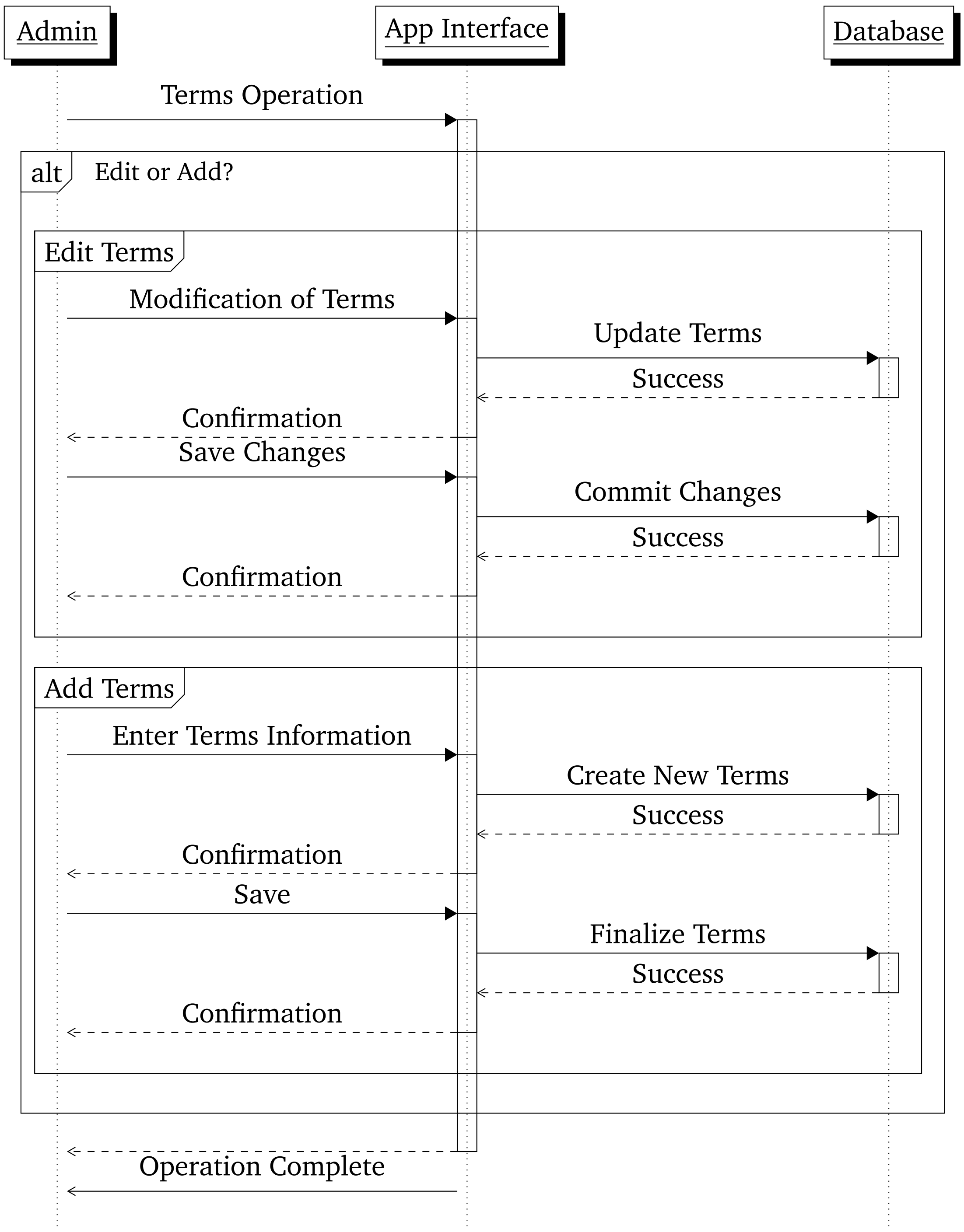
3.2.2.3 Application Sequence Diagrams

Below, we present a set of sequence diagrams that demonstrate different operations carried out by the admin user. Each diagram focuses on a specific use case and models the message flow involved.

- **Admin Managing Specialties (Clinics):** This diagram below illustrates how the admin adds or deletes medical specialties, including names and photos and addresses... through the app interface, with all changes saved to the database.



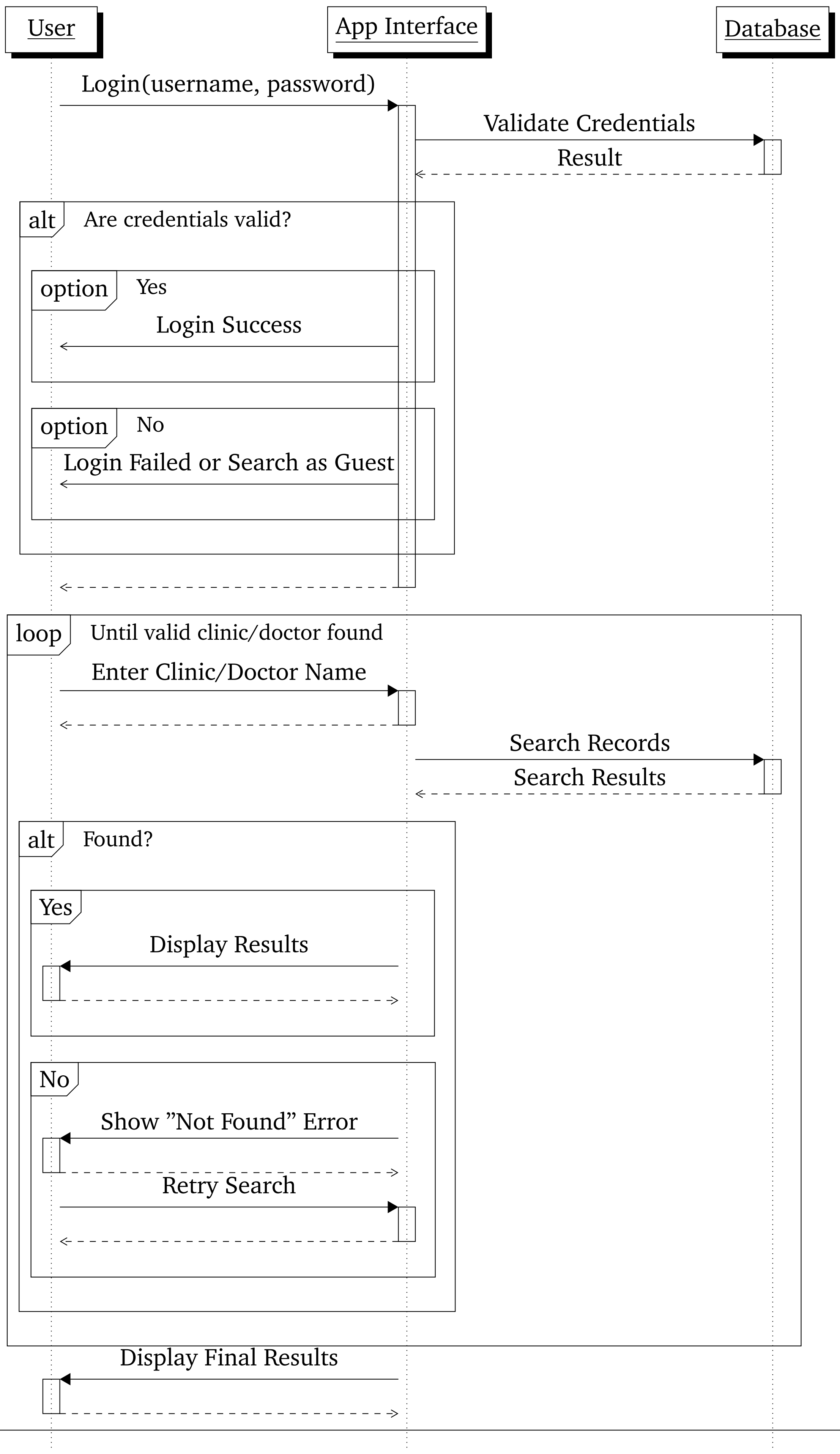
• **Admin Managing Terms and Conditions of the App:** This sequence diagram models how the admin adds or edits the Terms and Conditions of the application. Admin actions pass through the interface, updating the data in the database accordingly.



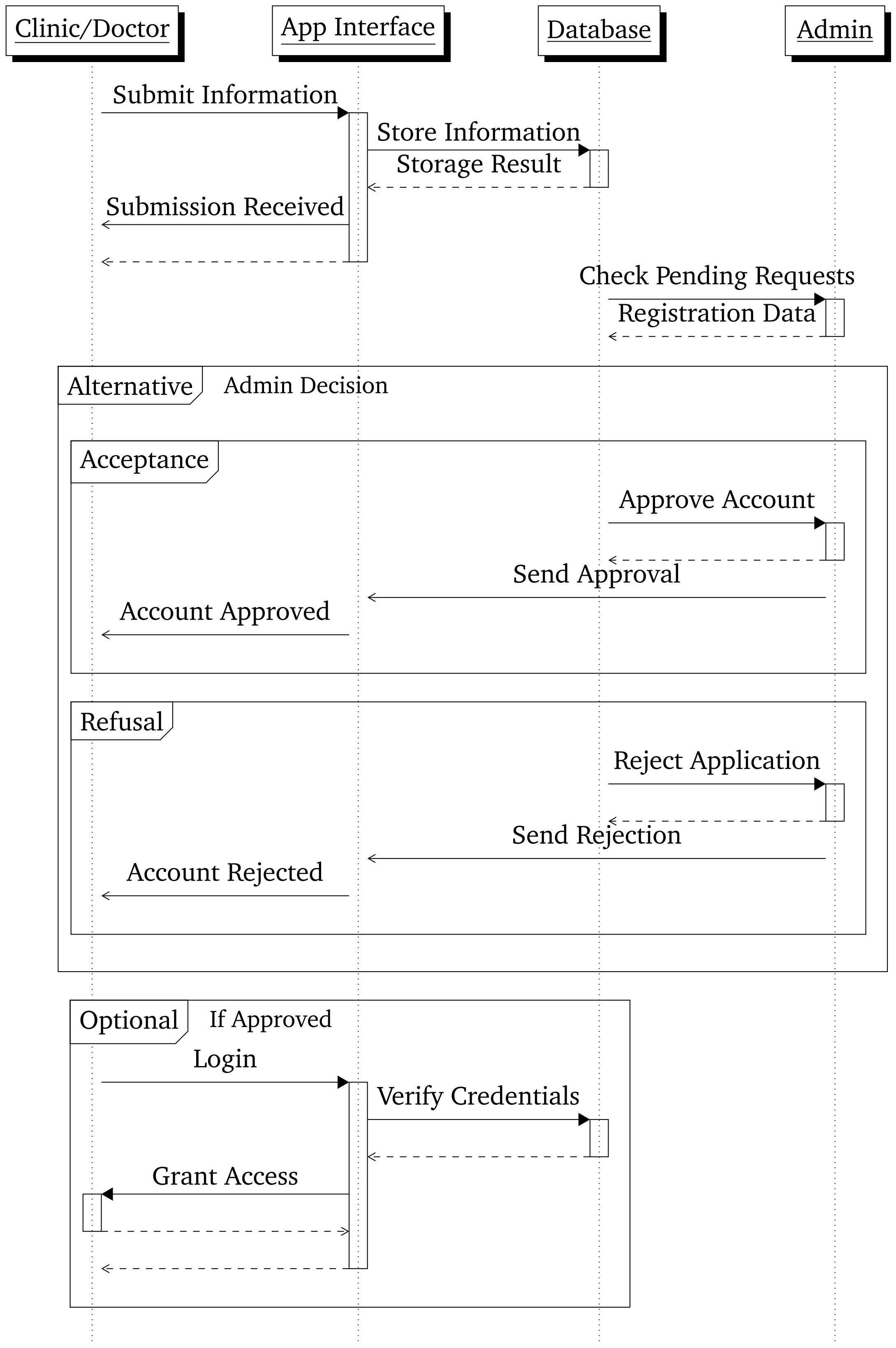
• **Searching for a Clinic or Doctor:**

This sequence diagram models the process in which the user searches for a clinic or doctor.

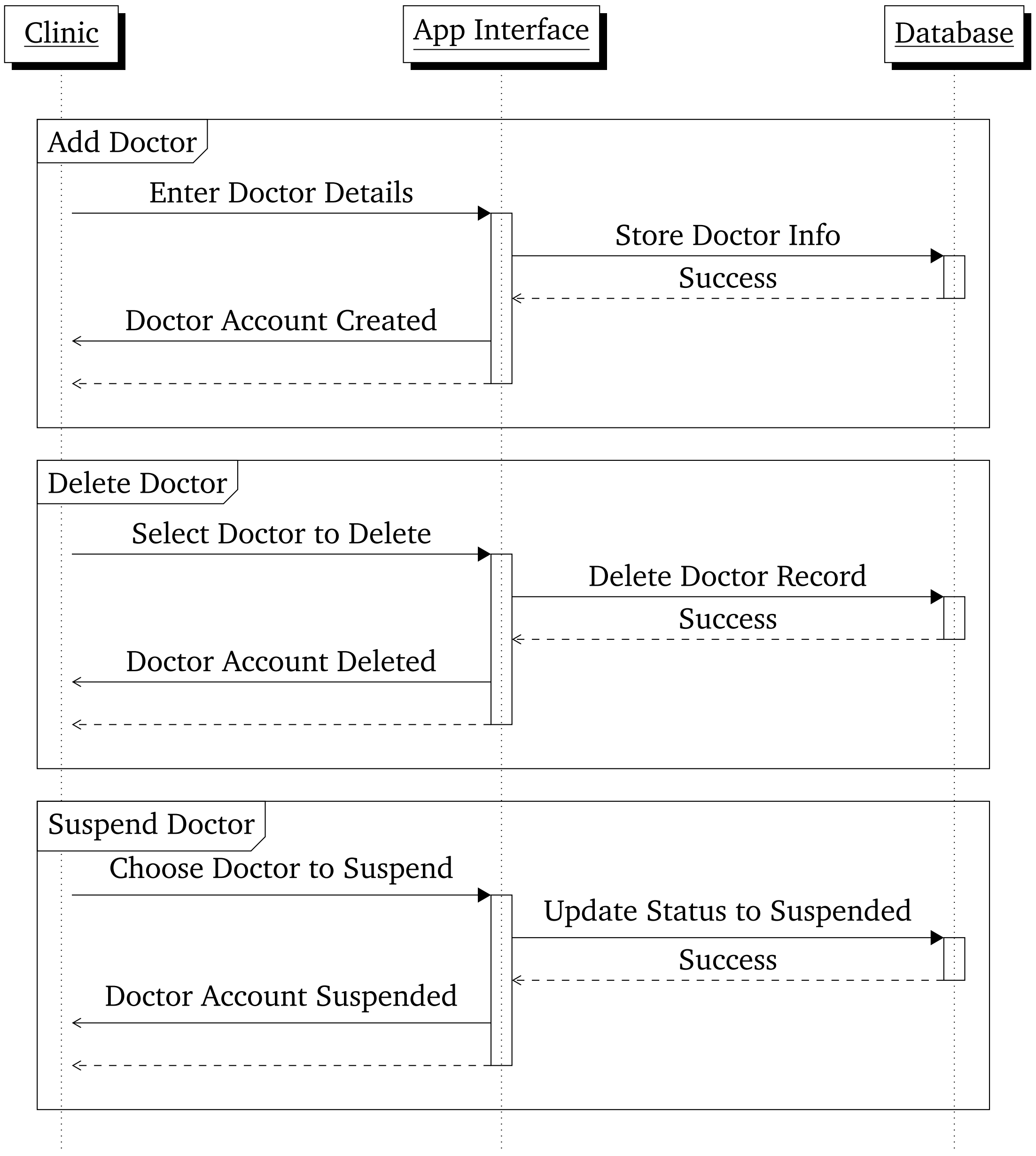
The user logs into the app, performs a search, and the app displays results or error messages if the search does not yield results.



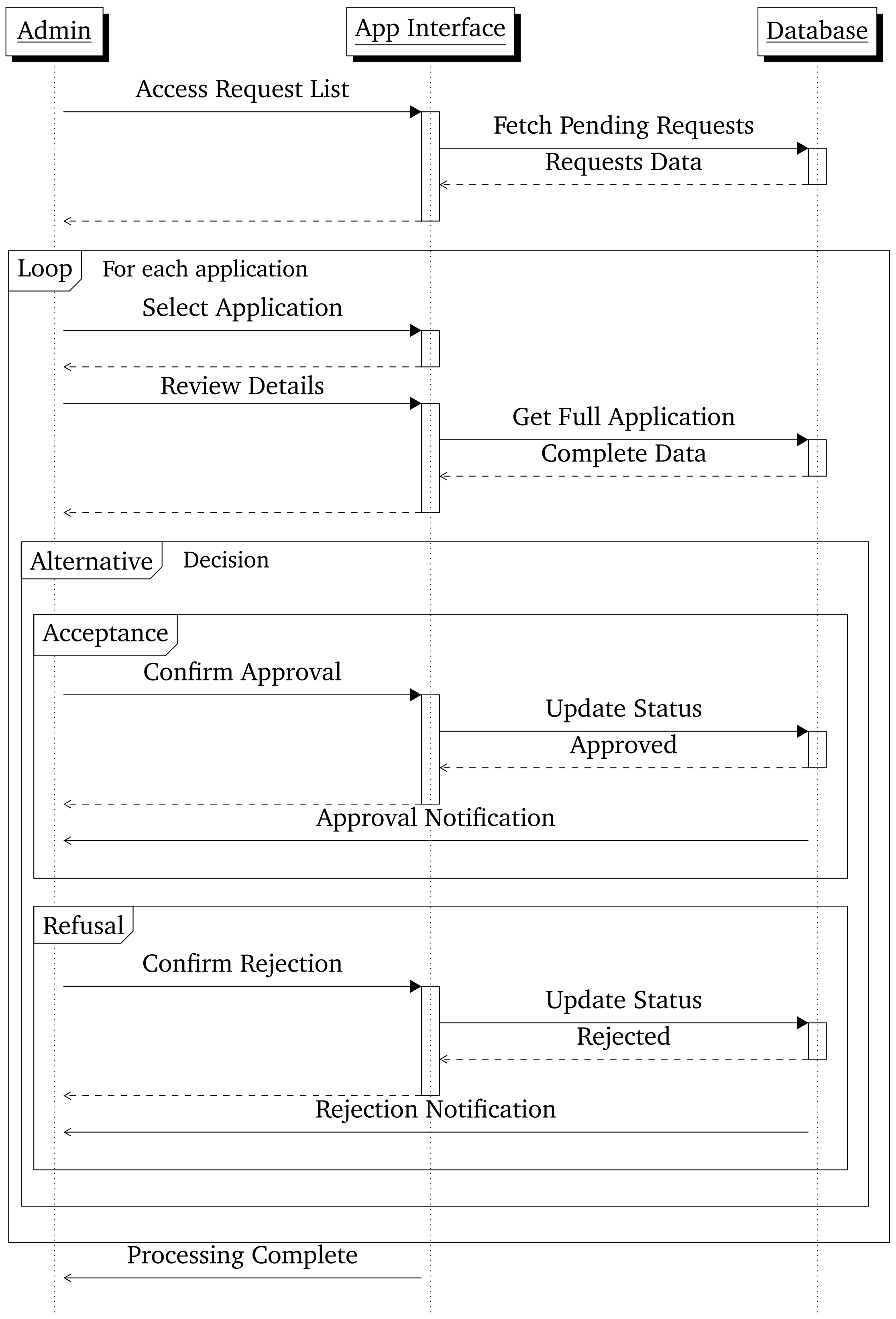
• **Clinic Registering In The Application :** This sequence diagram illustrates the process by which a clinic registers on the app. The clinic submits information through the app, which is then stored in the database. Admin verifies the submission and either approves or rejects the registration.



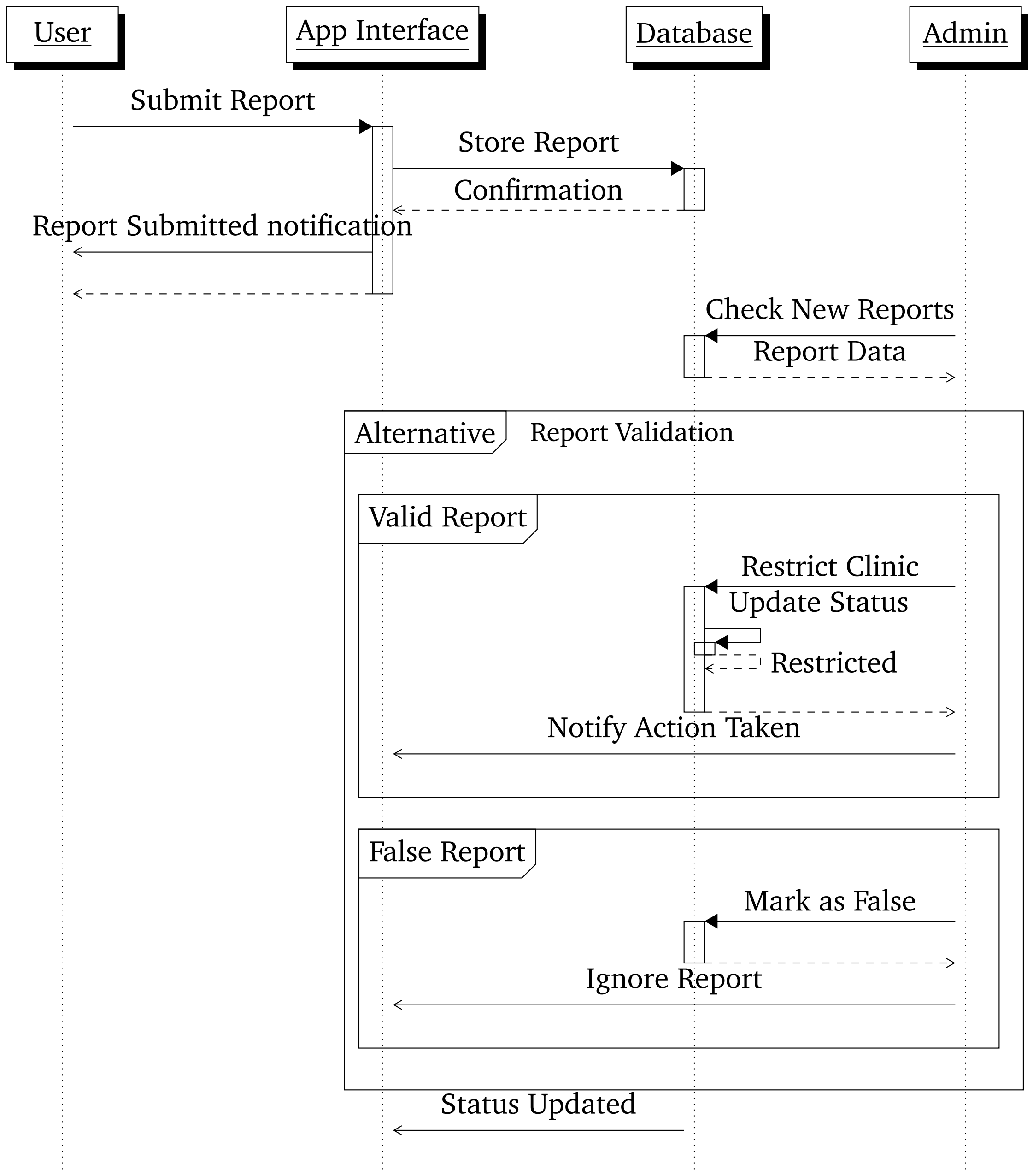
• **Clinic Managing Doctor Accounts (Add, Delete, Suspend) :** This sequence diagram illustrates how a clinic directly manages doctor accounts—adding, deleting, or suspending—through the application interface. All operations interact with the database without any admin involvement.



• **Admin Managing Clinics Requests:** This sequence diagram models the process by which the admin handles clinic registration requests. The admin reviews the applications, makes a decision, and updates the database accordingly.



• **User Report Activity:** This sequence diagram models the process when a user submits a report. After submission, the admin reviews the report and takes action, which may result in a notification being sent to the user.



3.3 Conclusion

The design phase of our *Hospital Finder* mobile application has been carefully structured to ensure clarity, consistency, and extensibility. Throughout this chapter, we laid the foundation of the application by modeling its components and interactions using essential UML diagrams.

We began with the class diagram, which defined the main entities such as users, clinics, doctors, and the admin, along with their relationships. This provided a blueprint for structuring the database and guiding the backend development. Each class was tailored to support the application's functionality, ensuring that both user and system needs are represented.

Subsequently, we introduced a series of sequence diagrams to illustrate the dynamic behavior of the system. These diagrams covered key interactions including user registration, clinic requests, login, searching, and admin management operations. Particular focus was placed on administrator features—such as reviewing clinic applications, editing terms and conditions, and handling user reports—emphasizing the application's flexibility and administrative control.

Each interaction was modeled to ensure that data flows seamlessly through the app interface and is consistently stored or retrieved from the database. The logical flow within these diagrams also helps in pinpointing edge cases, user roles, and error-handling scenarios.

In summary, this conception phase has transformed abstract requirements into detailed structural and behavioral models. These models not only support the system's current requirements but also anticipate future enhancements. The groundwork laid here will facilitate a smoother transition to the implementation phase, ensuring that development is aligned with the overall system design.

Chapter 4

The Implementation of our App

1 Introduction

This chapter is dedicated to the practical realization of our mobile application. It provides a detailed overview of the technical environment in which the project was developed and tested, including both hardware and software components.

We will begin by presenting the development and testing equipment used, followed by the tools and technologies adopted throughout the implementation process. In addition, we will present the application's key interfaces and explain how they interact.

Finally, this chapter will showcase selected code snippets and logic responsible for handling certain events and functionalities in the application — offering a clear view of the internal mechanisms that drive the user experience.

2 Work Environment

2.1 Hardware Environment

For development and testing, the following hardware was used:

- **Development Machines:**
  - **HP EliteBook 840 G4** – Intel Core i5-7200U, 16GB DDR4 RAM (2444MHz), 256GB SSD.
  - **Dell Laptop 1** – Intel Core i5 9th Gen, 16GB RAM.
  - **Dell Laptop 2** – Intel Core i5 6th Gen, 16GB RAM.

- **Test Devices:**
  - Samsung Galaxy S21
  - Samsung Galaxy S21 FE
  - Redmi Note 8 Pro

2.2 Software Environment

2.2.1 Technologies Used

It presents the main technologies adopted, such as the programming language, development platform, target mobile environment, and the chosen database management system.

2.2.1.1 Presentation

For the development of our mobile application, we adopted a set of modern tools and technologies to ensure the quality, performance, and maintainability of the project. Below is an overview of the main components of our software environment:

- **Programming Language:** We used **Dart**, through the **Flutter** framework, to develop a cross-platform mobile application (Android and iOS) from a single codebase.
- **Development Environments:** We primarily used **Visual Studio Code** and **Android Studio** as our code editors and development environments, which are widely recognized and powerful for mobile app development.
- **Backend:** We chose the **Laravel** PHP framework for creating the backend API. It provided us with a clear MVC structure, enhanced security, and easy management of routes and controllers.
- **Database:** The application uses a **MySQL** database, which we designed and hosted on a **Hostinger** server. The backend files were also deployed on this server to handle API communication with the mobile app.
- **Firebase:** Integrated for backend services, particularly **authentication**. We used **Google Console** to enable patients to log in using their Google accounts.
- **Testing Tools:** We used **Postman** to test various API requests and endpoints before integrating them into the mobile app.

- **Version Control:** Our source code was managed using **Git** and hosted on **GitHub**, which facilitated team collaboration and version tracking.
- **Design Tools:** We used **Figma** to design the user interface of the app and create user flow diagrams. For database modeling and class diagrams, we used **dbdiagram.io**.
- **Additional Tools:** *[This section is reserved for any additional tools or technologies we may have used during the development process.]*

2.2.1.2 Database Management System Used

In this project, special focus was given to designing and implementing the database using Laravel, which provides an efficient system for migrations, models, and seeders. This section describes the workflow, tools, and structure adopted for managing data throughout the application development.

The database schema was initially designed using **dbdiagram.io**, a web-based tool that allows for clear visualization of tables, their fields, and relationships. This step served as a blueprint, ensuring a well-structured and relational database design before implementation.

After finalizing the design, the schema was implemented using Laravel's **migration system**. Each table was defined in a separate migration file, outlining columns, data types, primary and foreign keys, and relevant constraints. Laravel's migration mechanism supports version control for the database, allowing for easy maintenance and scalability over time.

Following the migration setup, a dedicated **Eloquent model** was created for each table. These models act as an abstraction layer between the application and the database, handling operations such as data insertion, updates, deletion, and retrieval. Laravel automatically maps models to their corresponding tables and supports defining inter-model relationships (e.g., `hasMany`, `belongsTo`, `hasOne`, `belongsToMany`) that mirror the relational schema.

To initialize the database with essential application data, **seeders** were created. Seeders insert predefined data into the tables, making them particularly useful during development, testing, or on first-time deployments.

Throughout the development process, the database was managed and inspected using **phpMyAdmin**, a popular web-based tool for administering MySQL databases. phpMyAdmin facilitated direct interaction with the database for purposes such as data verification, relationship checks, and query testing, complementing Laravel's command-line and code-driven operations.

Although backend authentication **(using Firebase And Laravel)** was not the focus of this section, the system does implement strict user authentication and role-based access control. Each user is assigned a specific role upon registration (e.g., doctor, clinic, admin, or general user), and strict permission logic ensures that a user of one type cannot access another role's interface or perform unauthorized actions. For example, a doctor account cannot log in as a normal user or admin, and vice versa.

The authentication system integrates **Firebase Authentication** (via email/password and Google Sign-In), managed through the **Google Cloud Console**. Upon successful authentication on the client side (using Flutter), the user receives a Firebase token, which is then sent to the backend. On the Laravel side, we verify this token using Firebase's Admin SDK and issue a Laravel session token using **Laravel Sanctum**.

Sanctum provides a simple and secure way to manage API tokens and session authentication. We implemented custom middleware that checks the user's Firebase UID, retrieves their corresponding role from the database, and enforces access rules based on that role. Unauthorized access attempts result in a 403 Forbidden error, with clear JSON responses for frontend handling.

The following figures illustrate two key components of the system's security architecture. The first diagram explains how authentication is handled using Firebase and Laravel Sanctum, ensuring only verified users can access the system. The second diagram focuses on how user roles and permissions are enforced to control access to specific resources within the application.

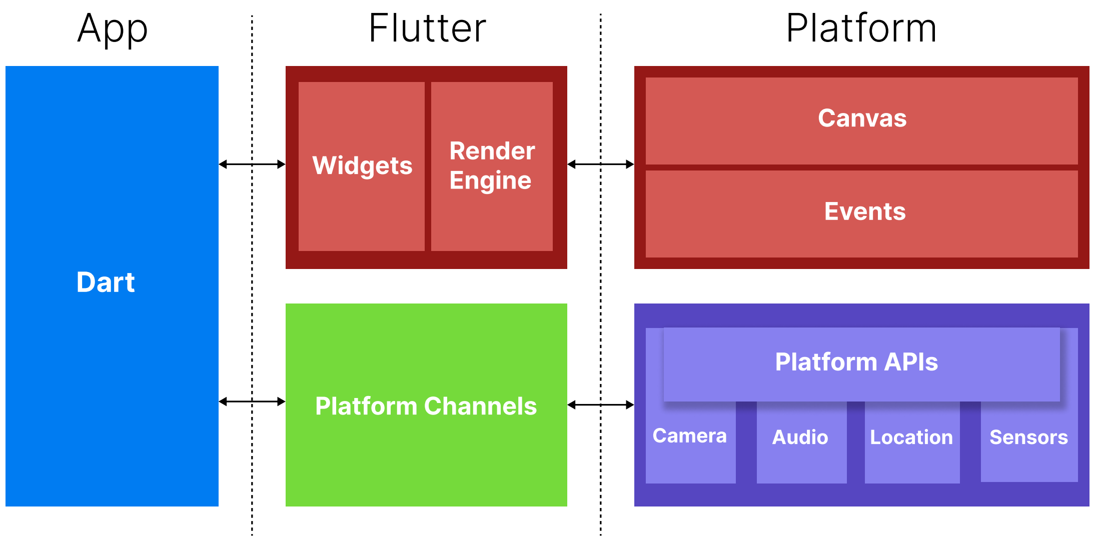


Figure 4.1: Authentication Flow: Firebase Authentication integrated with Laravel Sanctum

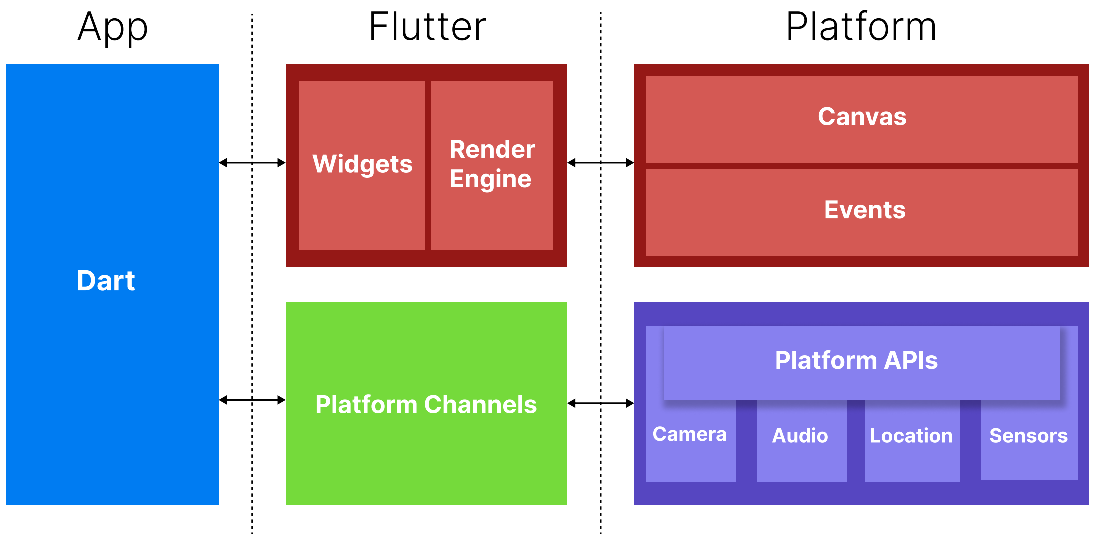


Figure 4.2: Role-Based Access Control: Enforcing Permissions for Users, Doctors, Clinics, and Admins

With a secure authentication and access control system in place, the structure of our database becomes the backbone for managing and organizing user-related data. To illustrate how different types of users and their interactions are handled within the system, we now highlight the most important tables—such as `users`, `clinics`, `doctors`, and `appointments`. These tables represent the core entities and relationships that power the application's functionality.

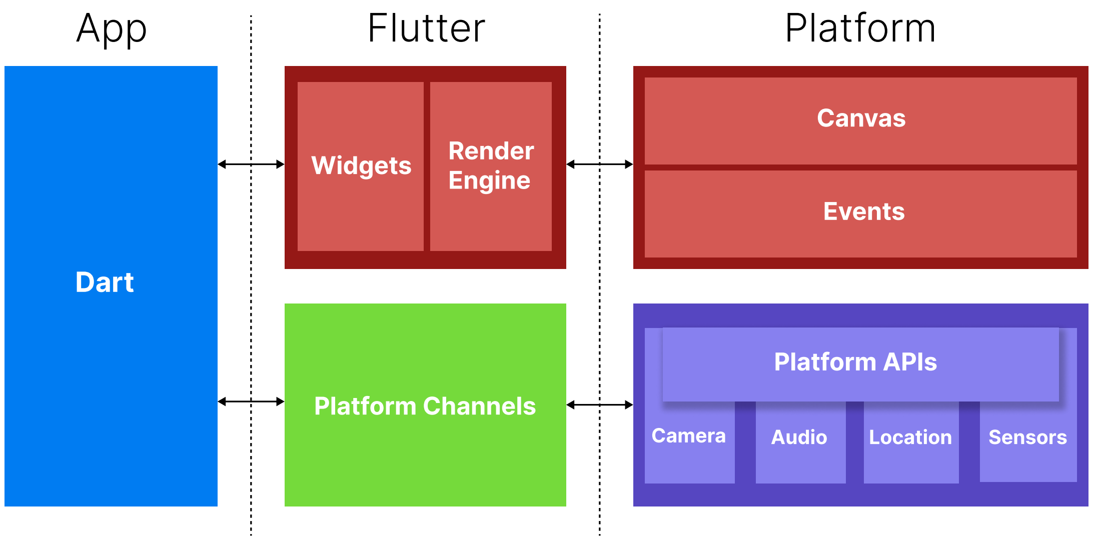


Figure 4.3: Users Table Schema

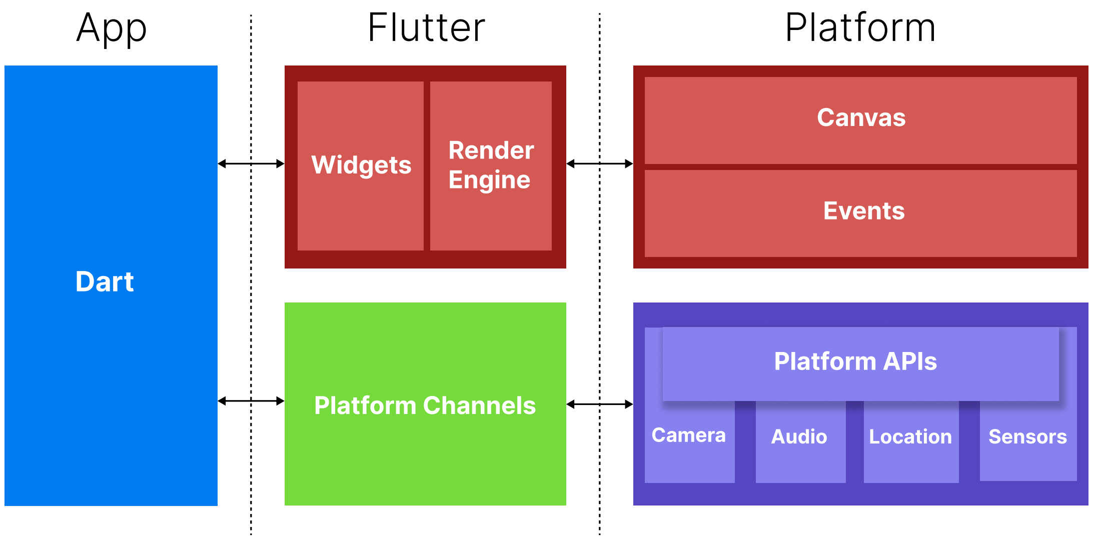


Figure 4.4: Clinics Table Schema

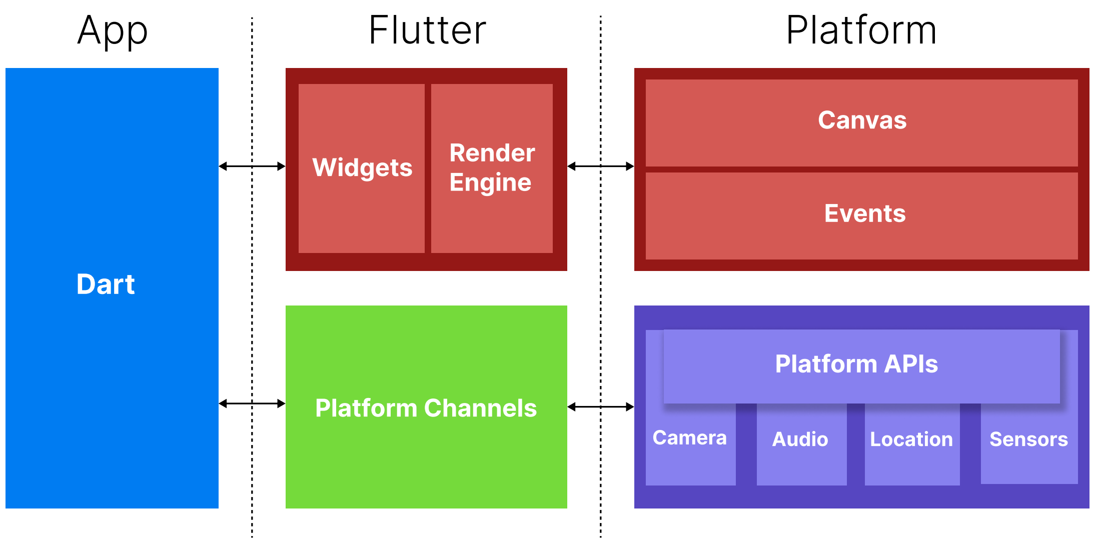


Figure 4.5: Doctors Table Schema

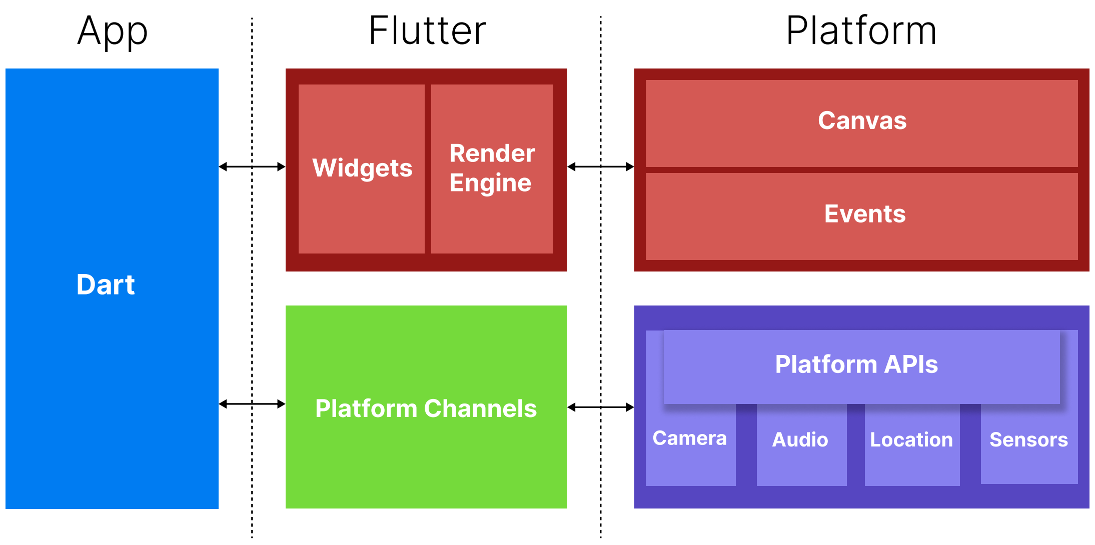


Figure 4.6: Appointments Table Schema

Finally, the complete database structure and its initial data were deployed using the following Artisan command:

Listing 4.1: Laravel command to run migrations and seeders

```
php artisan migrate --seed
```

This command runs all migration files to create the defined schema and immediately executes the seeders to populate the tables with required data. It ensures the database is fully prepared for integration with the rest of the application.

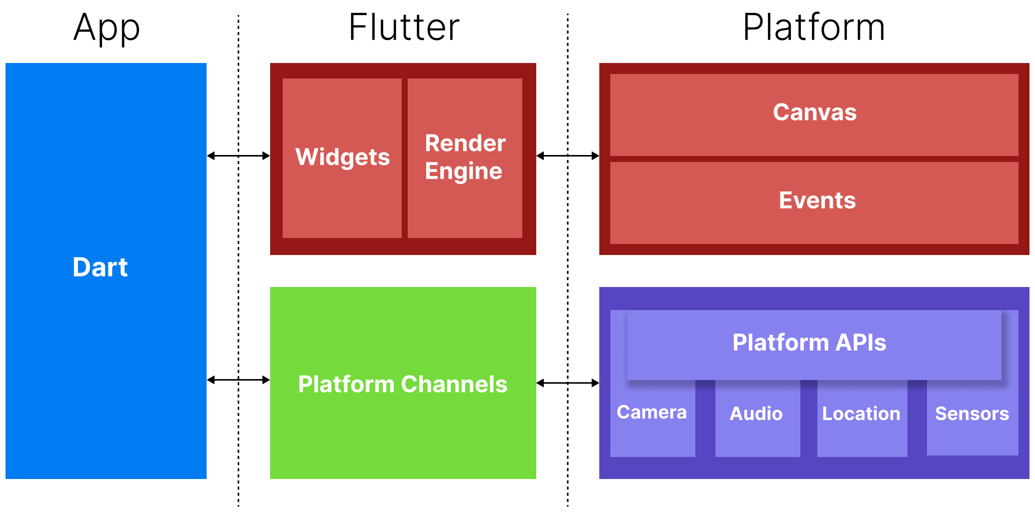


Figure 4.7: Laravel MVC Architecture: Model-View-Controller Structure