Architecture Report

Project : Medical Record Management of the Preventive Medicine Unit of ESI Sidi Bel Abbès

VERSION TABLE

Version	Date	Author
1.0	05-03-2025	Mohamed Haouari
1.1	11-03-2025	Mohamed Haouari
1.2	15-03-2025	Mohamed Haouari
1.3	01-04-2025	Mohamed Haouari
1.4	20-04-2025	Mohamed Haouari

TABLE OF CONTENTS

TABLE OF FIGURES

1. INTRODUCTION:

In the healthcare sector, efficient and secure management of medical records is essential for delivering quality patient care. The Medical Record Management System for the Preventive Medicine Unit of ESI Sidi Bel Abbès aims to modernize healthcare operations by replacing manual processes with a robust, web-based platform. This system will streamline record-keeping, enhance data accessibility, and ensure compliance with regulatory standards, all while prioritizing data security and user-friendliness.

This document outlines the project's objectives, scope, technical architecture, and implementation strategy. By leveraging cutting-edge technologies and best practices, we aim to deliver a scalable solution that meets the unit's current needs and supports future growth. Through this initiative, we strive to empower healthcare professionals and improve patient care at ESI Sidi Bel Abbès.

2.GLOBAL OVERVIEW:

2.1 System Actor :

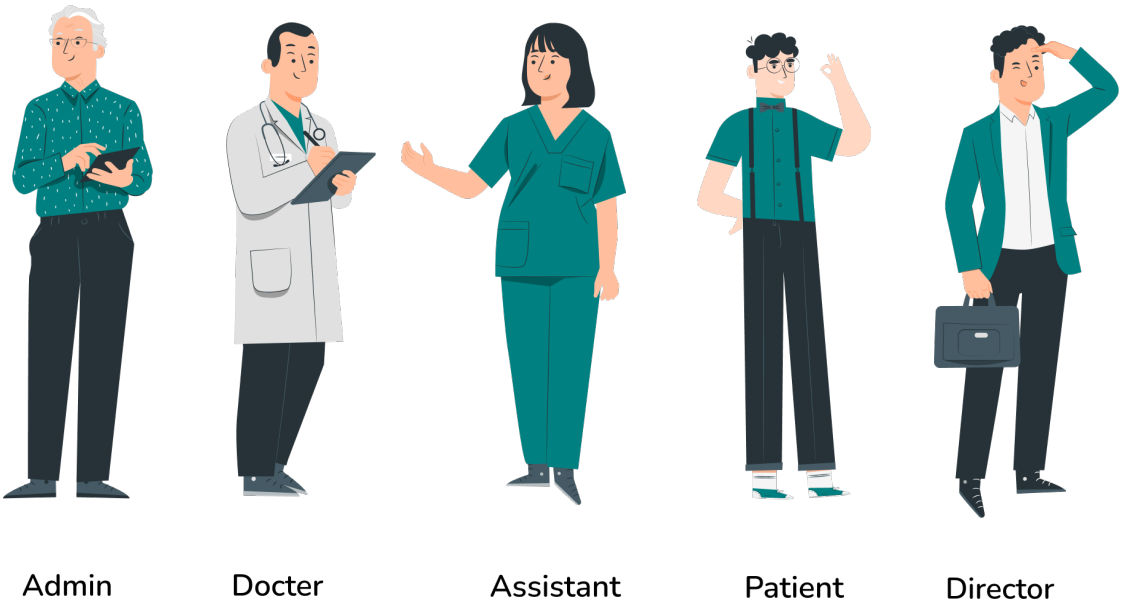
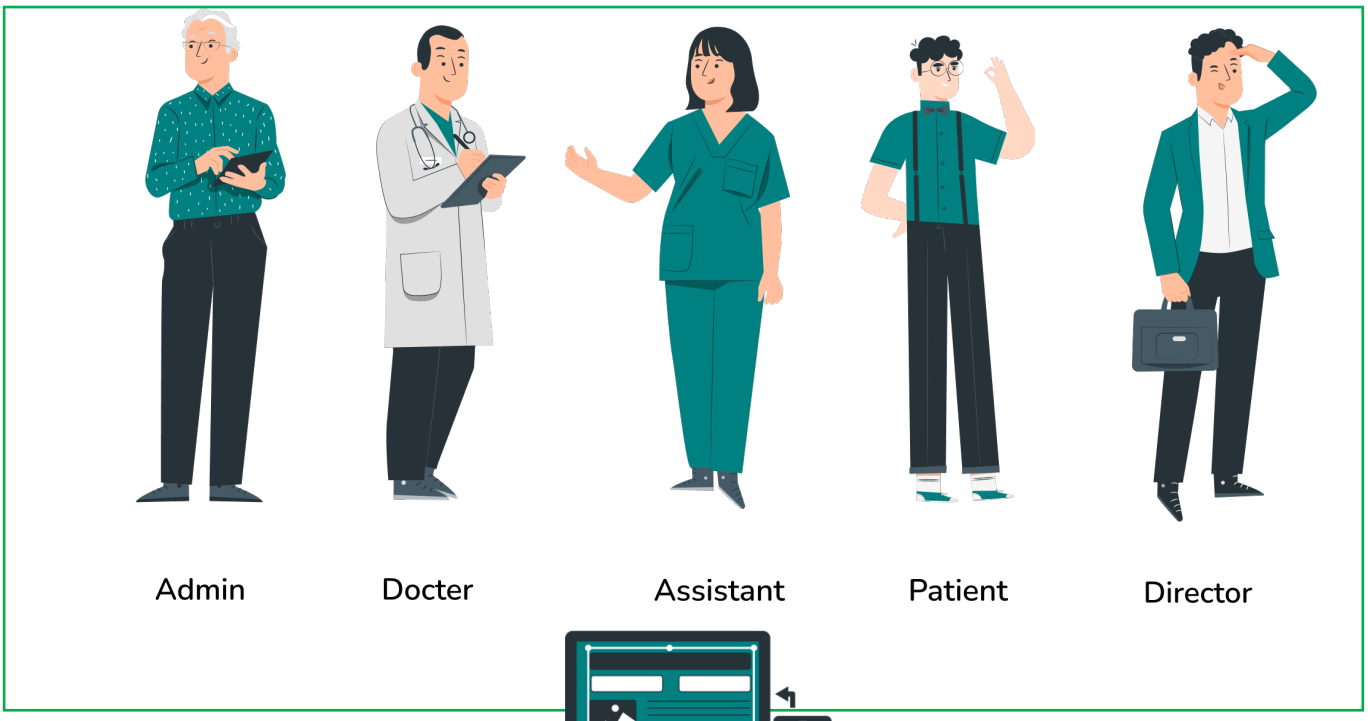


Figure 1 : THE SYSTEM'S ACTORS

- **Admin:** Manages user accounts, system settings, and ensures data security.
- **Doctor:** Handles patient records, appointments, and medical examinations.
- **Doctor's Assistant:** Manages appointments and supports the doctor.
- **Patient:** Books appointments and views medical records.

- **School Director:** Monitors health statistics for decision-making.

2.2 Global Schema:



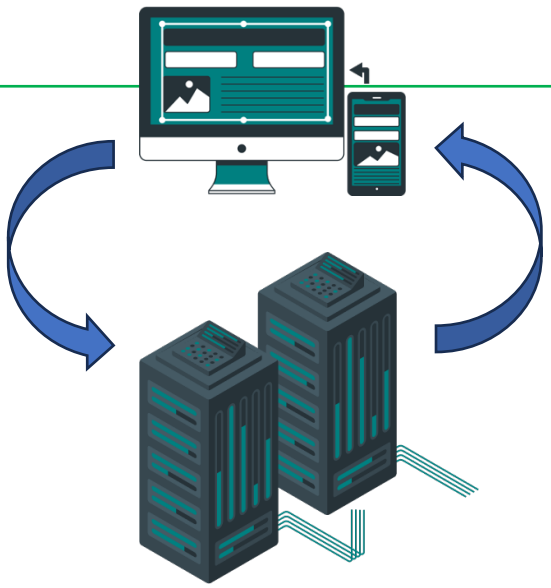


Figure 2:Global System diagram

3.DETAILED SCHEMA FOR EACH MODEL:

3.1 Account Management and Roles:

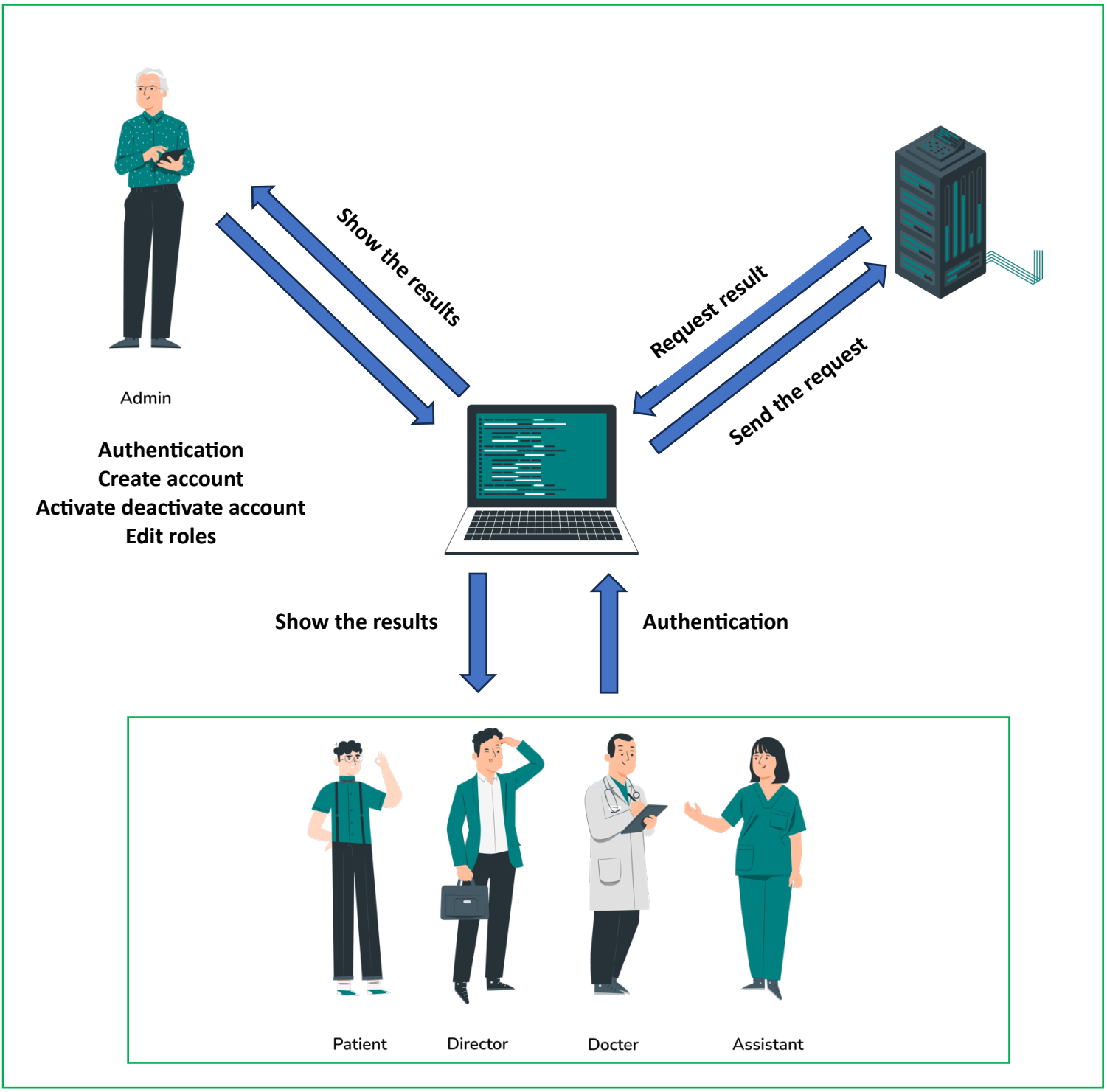


Figure 3:ACCOUNT/ROLE MANAGEMENT SCHEME

3.2 Medical record Management:

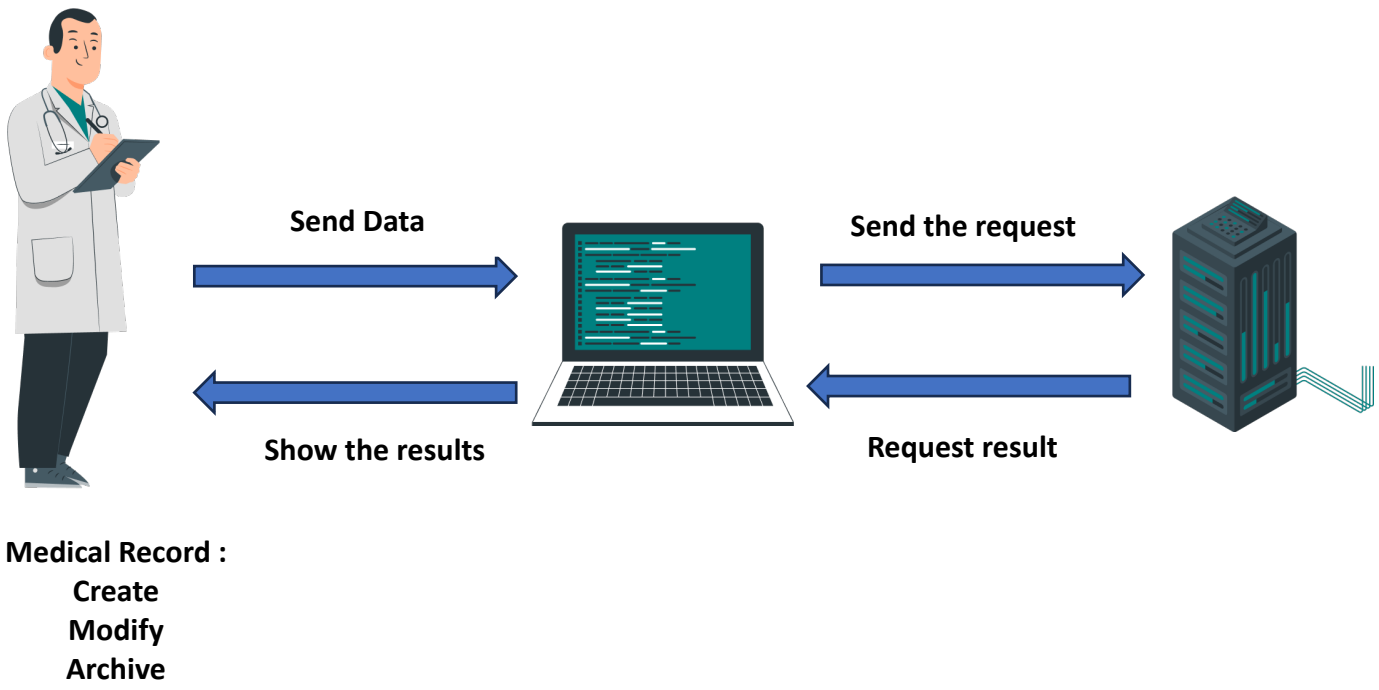
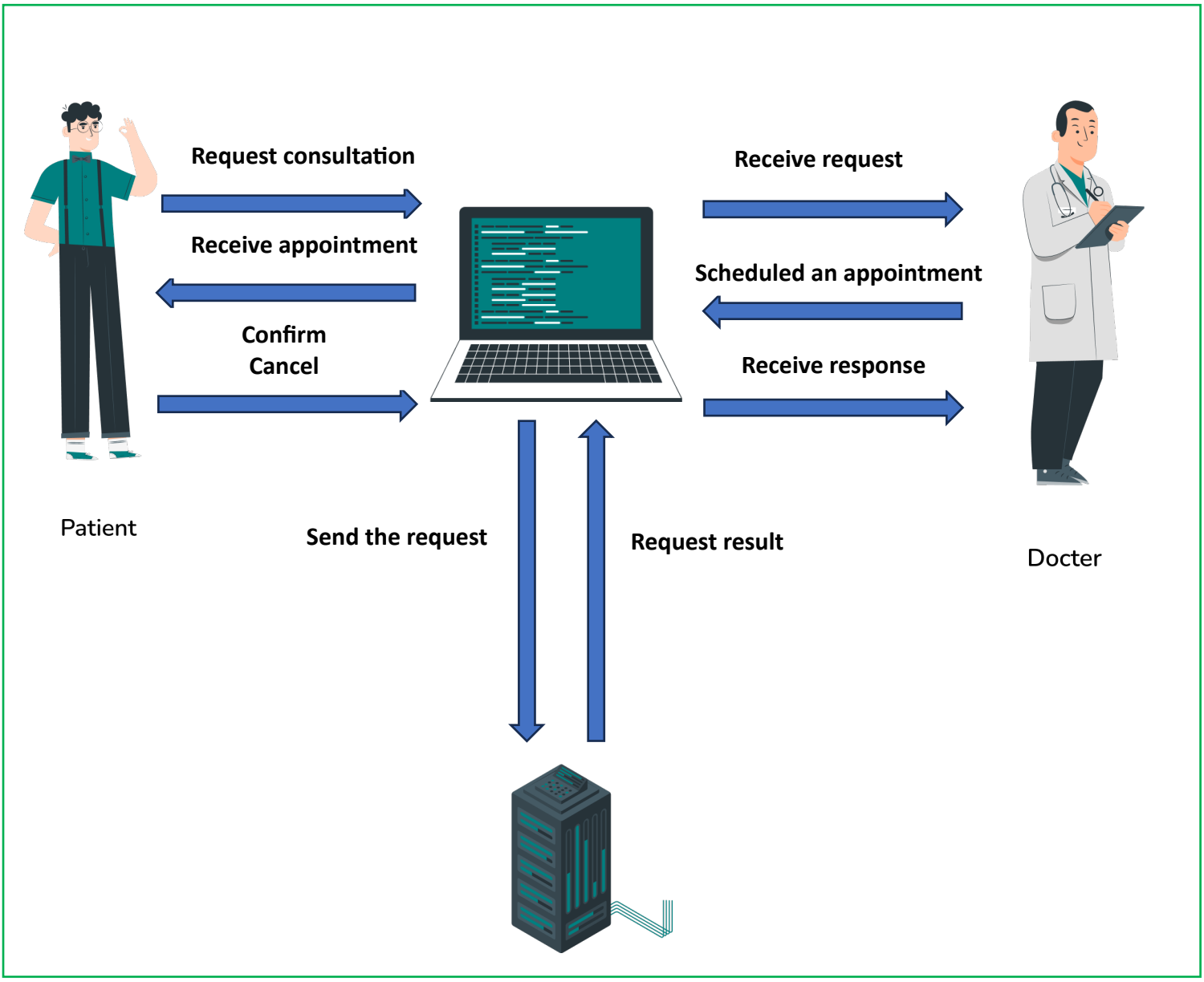


Figure 4: Medical record Management

3.3 Appointment Management:



4. ARCHITECTURE USED:

4.1 Frontend-Backend Architecture:

The system is built using a **frontend-backend architecture**, where the frontend (user interface) and backend (server-side logic) are separated for scalability, flexibility, and ease of maintenance.

4.1.1 Frontend (React.js and Flutter):

1. **Web Frontend (React.js)**:
    - The web frontend is developed using **React.js**, a popular JavaScript library for building dynamic and responsive user interfaces.
    - It handles user interactions and displays data fetched from the backend.
    - Communication with the backend is facilitated through **RESTful APIs**.
2. **Mobile Frontend (Flutter)**:
    - The mobile frontend is developed using **Flutter**, a cross-platform framework for building native-like mobile applications for iOS and Android.
    - It provides a responsive and visually appealing interface for mobile users.
    - Communication with the backend is facilitated through **RESTful APIs**.

4.1.2 Backend (Laravel - MVC):

The backend is built using **Laravel (PHP)**, which follows the **Model-View-Controller (MVC)** architecture. This design pattern ensures a clear separation of concerns, making the backend codebase modular, maintainable, and scalable.

4.2 MVC Architecture:

4.2.1 Definition:

1. **Model**:
    - The **Model** represents the data and business logic of the application.
    - It interacts with the **MySQL database** to perform operations such as fetching, updating, or deleting data.
    - The Model also manages relationships between different data entities, such as patients, appointments, and medical records.

- **Example**:
  - A **Patient** model handles all operations related to patient data, such as retrieving a patient's medical history or updating their contact information.

2. **View**:
   - In traditional MVC architecture, the **View** is responsible for rendering the user interface.
   - In this system, since the frontend is built with **React.js (web)** and **Flutter (mobile)**, the View layer in Laravel is minimal. Instead, Laravel acts as an **API backend**, returning data in **JSON format** to the frontend.
   - **Example**:
     - When a request is made to fetch a patient's medical records, Laravel processes the request and returns the data in JSON format, which the frontend then displays.
3. **Controller**:
   - The **Controller** acts as the intermediary between the Model and the View.
   - It handles **HTTP requests** from the frontend, such as booking an appointment or updating a record.
   - The Controller processes the request, interacts with the **Model** to fetch or update data, and returns the appropriate response to the frontend.
   - It also implements application logic, such as ensuring only authorized users can access certain data.
   - **Example**: An **AppointmentController** handles requests related to appointments, such as creating a new appointment or fetching a list of upcoming appointments.

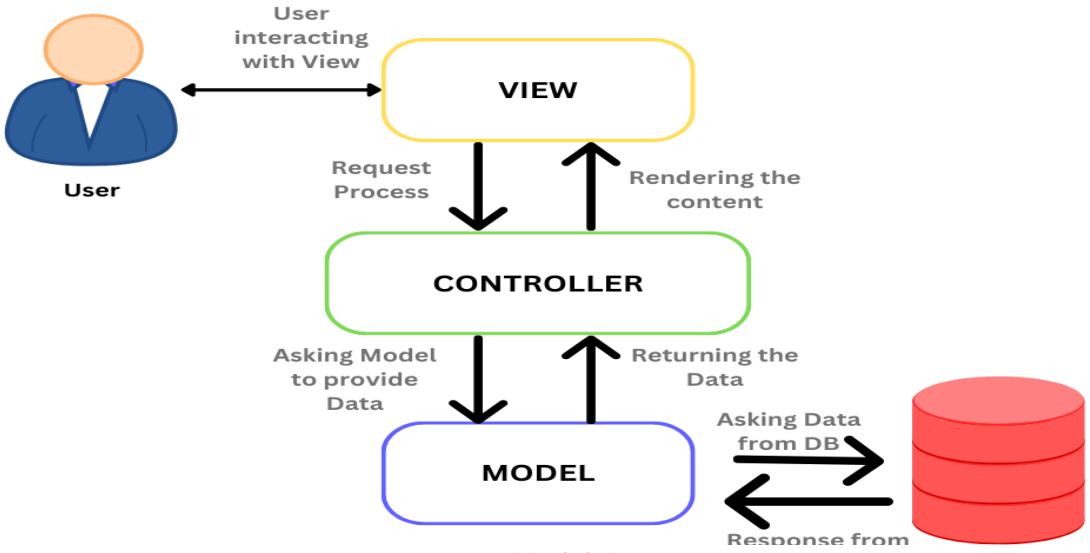


Figure 5: MVC Scheme

4.2.2 ADVANTAGES OF MVC ARCHITECTURE:

- **Separation of Concerns:** Each component (Model, View, Controller) has a specific responsibility, making the code easier to maintain and debug.
- **Reusability:** Components like the Model can be reused across different parts of the application (e.g., the same patient data model can be used for appointments and medical records).
- **Scalability:** Adding new features or modifying existing ones is easier because changes are isolated to specific components.
- **Improved Collaboration:** Developers can work on different components (e.g., UI designers on the View, backend developers on the Model) simultaneously without conflicts.
- **Enhanced Testability:** Each component can be tested independently, ensuring higher code quality and reliability.

5.COMPONENT DIAGRAM:

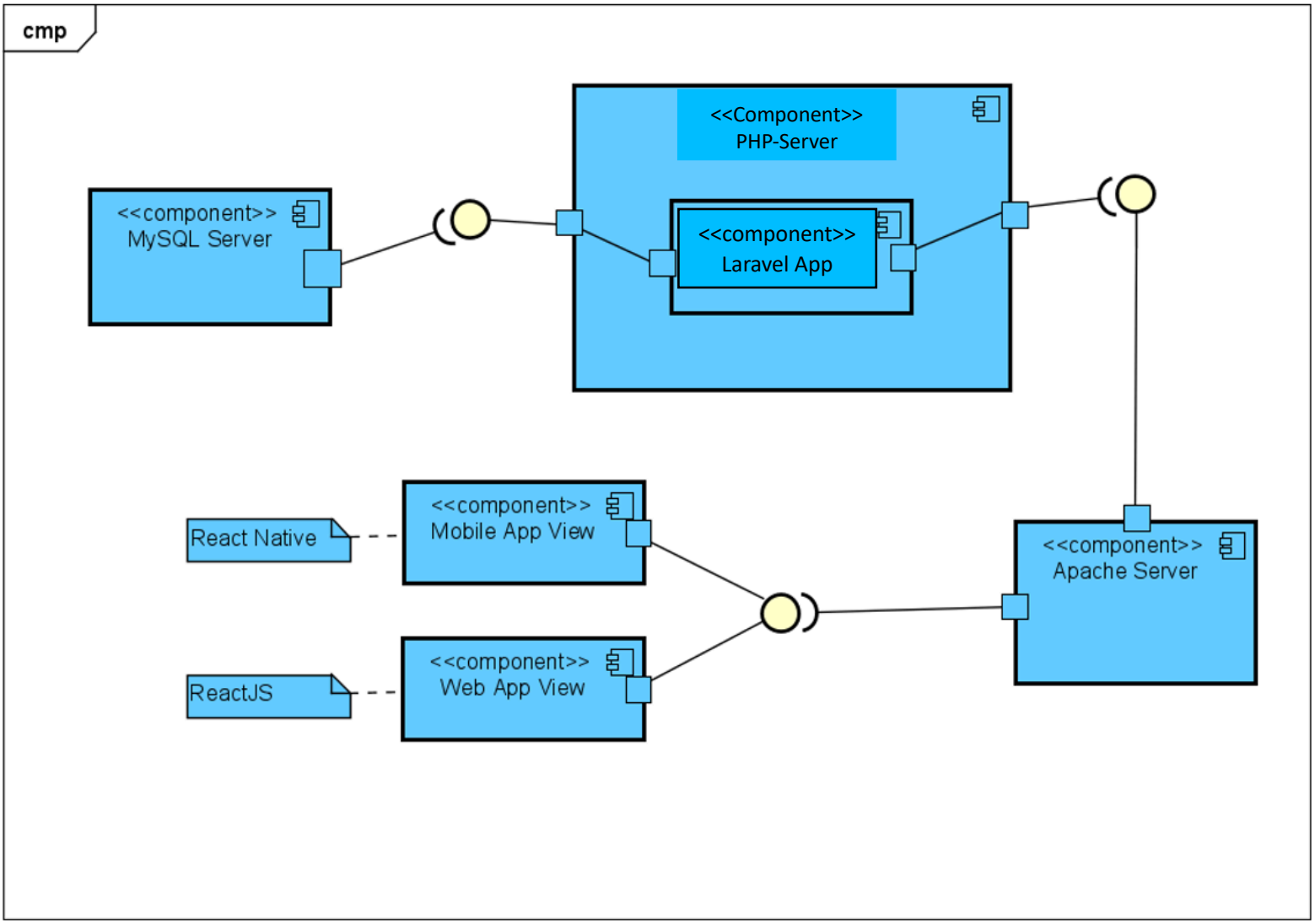


Figure 6: Component Diagram

6.DEPLOYMENT DIAGRAM:

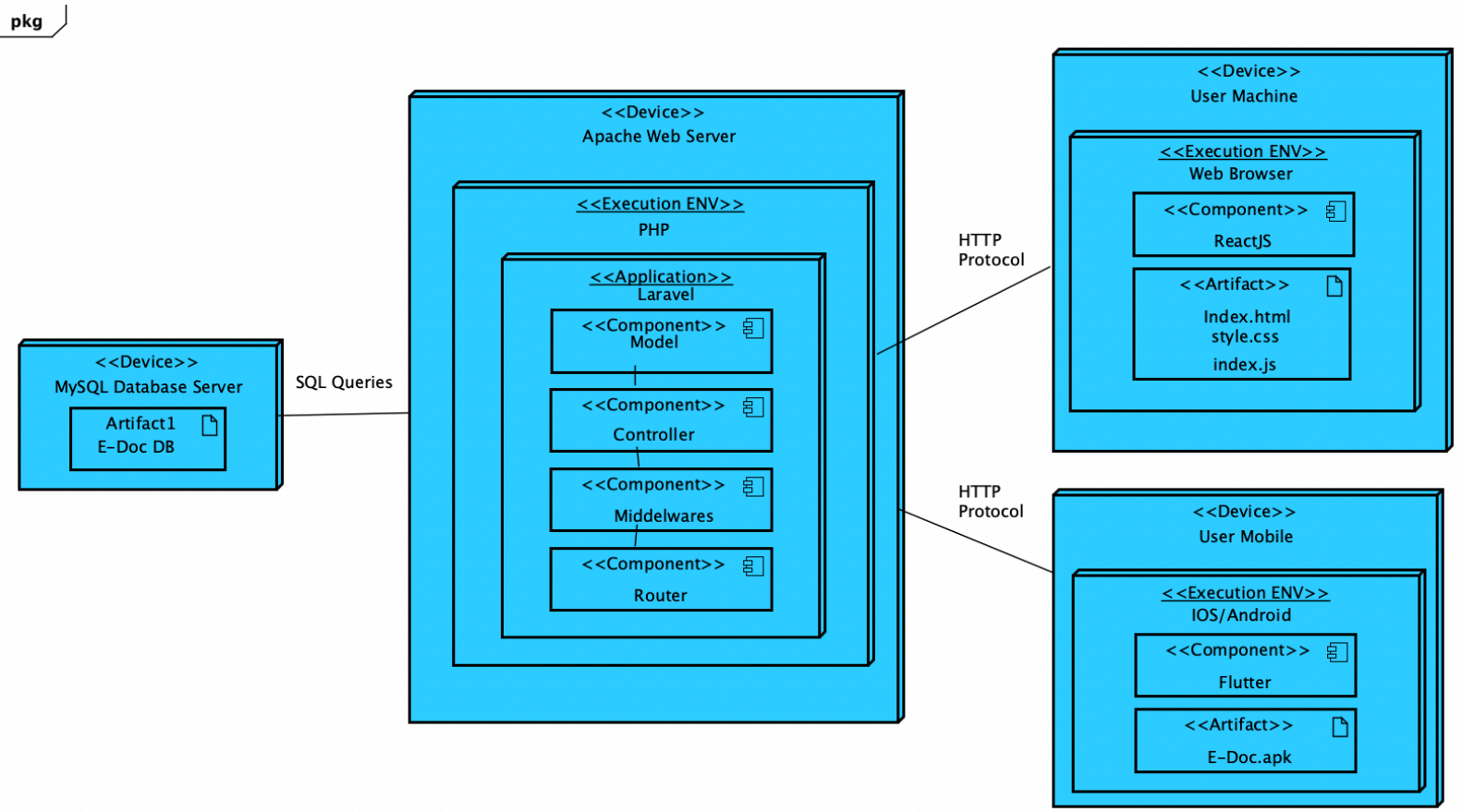


Figure 7: Deployment Diagram