الجمهورية الجزائرية الديمقراطية الشعبية
The People's Democratic Republic of Algeria
وزارة التعليم العالي والبحث العلمي
Ministry of Higher Education and Scientific Research
جامعة محمد بوضياف بالمسيلة
University Mohamed Boudiaf of M'sila

كلية الرياضيات والإعلام الآلي
Faculty of Mathematics and Informatics

قسم الإعلام الآلي
Department of Computer Science

Domain: Mathematics and Computer Science

Thesis Presented to Fulfill the Partial Requirement
for the **Master's Degree** in Computer Science

Specialty: Information System and Software Engineering

Prepared By: Fardjoui Mounir Islem

Supervised By:
Hichem Debbi

Entitled

Healthcare Managment System

Jury Members

Name and Surname	Function
SALAH GUESMIA	President
Ahlem Bouzaroura	Examiner
Hichem Debbi	Supervisor

Academic Year **2024/2025**

إهداء

لى من كان لهم أكبر دعم في حياتي ومسيرتي الدراسية...

إلى والدي، **أحمد فرجاوي**، قدوتي في الصبر والاجتهاد والتضحية. كان تشجيعكم الدائم هو الدافع وراء نجاحي.

إلى والدتي، **فضيلة بن داود**، مصدر حبي اللامشروط ودعواتي الصادقة، التي منحني دعمها القوة للاستمرار.

إلى أخي، **وليد فرجاوي**، سنداً لي، لولاه لما حققتُ شيئاً، كان دائماً بجانبي، وكان دائماً سندي الأكبر.

إلى روح أخي العزيز **محمد**، الذي رحل عنا باكراً، لكنه سيبقى حاضراً في قلبي، نوراً يُنير دربي ويُلهمني في هذه الرحلة.

إلى شقيقتي **أمينة ونسرين**، وزوجيهما **أحمد بورنان وطيب القالي**، على دعمهم وتشجيعهم الدائم.

إلى أبنائهم الأعزاء: **محمد ودانيا وسجى**، الذين أدخلوا الفرح والأمل إلى قلبي بابتساماتهم.

إلى صديق طفولتي، **عمر بورحلة**، رفيق دربي في بداياتي.

إلى صديقي العزيز في الجامعة، **أيمن قاسمي**، الذي شاركني مشقة هذه الرحلة الدراسية وجهدها.

إلى كل من ساندني بكلماته الطيبة، ودعواته الصادقة، وتشجيعه...

إلى كل من ساهم في هذا الطريق...

أهدي هذا العمل بكل حب وامتنان وتقدير.

تشكرات

أحمد الله تعالى حمدًا يليق بجلاله، وأشكره على ما أنعم به عليّ من توفيق وتسديد طيلة مراحل هذا العمل الأكاديمي.

أتقدم بجزيل الشكر والامتنان إلى مشرفي الفاضل، الأستاذ **هشام دبي**، على ما أبداه من توجيه علمي دقيق، ومتابعة مستمرة، ودعم لا محدود. لقد ساهمت ملاحظاته العميقة، ونصائحه الثمينة، وحرصه على الجودة الأكاديمية في تطوير هذا البحث، وإثرائه بالمعرفة والتقنيات الحديثة.

كما أتوجه بخالص الشكر والامتنان إلى الأستاذة **بوزعرورة أحلام** التي رافقتنا خلال مرحلة شهادة الليسانس، وقدمّت لنا الدعم والتوجيه المستمر، وكان لها أثر بالغ في إنجاح هذه الرحلة العلمية.

كما أخص بالشكر السادة أعضاء لجنة المناقشة الكرام على وقتهم وجهودهم، وتفضلهم بقراءة هذا العمل وتقديم ملاحظاتهم العلمية القيّمة، والتي كان لها أثرٌ كبير في تجويد المحتوى العلمي لهذا البحث.

ولا يفوتني أن أتوجه بخالص الامتنان لكل من قدم لي يد العون والمساعدة، سواء من خلال الدعم الفني، أو المشورة العلمية، أو النقاشات المثمرة، التي أسهمت جميعها في تذليل العقبات وإثراء تجربتي البحثية.

لكل من ساهم في هذا العمل، أقول: شكرًا جزيلًا، وبارك الله في جهودكم.

Contents

List of Figures

General Introduction

With the rapid advancements in modern technology and its adoption in various areas of life, it has become imperative to develop smart digital solutions that keep pace with this development, especially in the healthcare sector, given its importance and direct impact on human life. [1] This framework, however, is not sufficient for this project, which has contributed to the development of a bilingual medical platform (Arabic/English) aimed at facilitating communication between doctors, patients, and pharmacists. The platform, supported by a group of advanced investors, aims to facilitate communication between doctors, patients, and pharmacists. The most prominent features include providing prescriptions using QR codes, scheduling appointments, managing tests and analyses, and chatting for real-time organization.

This platform focuses on providing an effective solution that addresses a number of real-world challenges facing the healthcare sector, with an emphasis on security, usability, and partial data separation based on a multi-tenant architecture (multi-tenancy), where each doctor or pharmacist owns their own database **arabadmin2020healthit**.

This main memorandum is divided into four chapters:

- **Chapter One:** This chapter provides an overview of the modern healthcare sector, including the transformation and introduction to digital concepts and definitions of their principles directly related to our project, such as health information systems, digital medicine, and the importance of communication between the various actors in the healthcare system.
- **Chapter Two:** This chapter introduces the system analysis and design phase, identifying the building requirements (functional and non-functional), exploring various usage scenarios, and the planning implications that define the system's structure and interaction.
- **Chapter Three:** This chapter focuses on explaining the most important technologies used during the project's development, such as Artificial Intelligence Models (LLM), Multi-Tenancy technology, WebSocket for real-time communication, and other enhancements.
- **Chapter Four:** This chapter presents the interfaces developed for each user category, explains the background, and clarifies the tools, languages, and libraries used to build the platform, both in the interface and on the platform itself. It also provides extensive details on documentation, security, and multilingualism.

Through this project, we combine digital and applied perspectives to provide a practical solution for the local healthcare sector, utilizing advanced programming and coding techniques.

Chapter 1

Healthcare Management Systems

1 introduction

The healthcare sector is undergoing a profound digital transformation worldwide, driven by the need for efficient, accessible, and patient-centered care. In Algeria, like many other countries, the healthcare system faces significant challenges due to fragmented and paper-based medical records. These outdated systems lead to inefficiencies, medical errors, and difficulties in accessing patient information, ultimately compromising the quality of care. The advent of Electronic Health Records (EHRs) and Healthcare Management Systems (HMS) has revolutionized healthcare delivery by digitizing patient data and streamlining workflows. These systems not only enhance operational efficiency but also improve patient outcomes by ensuring that healthcare providers have real-time access to comprehensive medical histories

2 Healthcare Management Systems:

2.1 Definition of health care management system:

The Health Management System - HMS is an integrated digital system that aims to organize and coordinate administrative, medical, and financial operations within health institutions such as hospitals, clinics, and medical centers. This system includes a set of software tools that facilitate the management of electronic medical records (EHR), scheduling appointments, following up on treatments, managing resources, preparing invoices, and ensuring compliance with health and legal standards, which contributes to improving the quality of health services and increasing the efficiency of medical and administrative work alike.whether. **ali2019hospital** [2]

2.2 Types of HMS

- **Paper-Based Medical Records (PBMR)**
  Before the digital era, healthcare facilities relied on paper-based records, which included handwritten notes, test results, and prescriptions
  Advantages : Familiarity: Healthcare professionals are accustomed to using paper records. No Technology Dependency: Does not require internet connectivity or electronic devices.
  Disadvantages:
  - Limited Accessibility: Records are stored in a single location, making it difficult for multiple providers to access them simultaneously.
  - Risk of Loss or Damage: Physical records can be misplaced, damaged by environmental factors (e.g., fire, water), or deteriorate over time.
  - Inefficient Data Retrieval: Searching for specific patient information is time-consuming and prone to errors.
  - Lack of Interoperability: Difficult to share records across different healthcare facilities or integrate with other systems

- **Electronic Health Records (EHRs)**
  Electronic Health Records (EHRs) are digital versions of patient records that offer a more comprehensive and integrated approach to healthcare documentation.

  Components of EHRs: begin

  - Electronic Medical Records (EMRs): Digital patient histories, diagnoses, treatments, and test results.
  - Personal Health Records (PHRs): Patient-managed health data, empowering individuals to take an active role in their
  - Practice Management Tools: Features for scheduling appointments, processing billing, and managing administrative tasks.
  - Clinical Decision Support: Alerts for potential drug interactions, evidence-based treatment guidelines, and diagnostic suggestions

  Advantages:

  - Improved Accessibility: Authorized providers can access patient records remotely, facilitating timely care.
  - Enhanced Data Management: Efficient storage, retrieval, and analysis of patient data support better decision-making.
  - Reduced Errors: Eliminates issues like illegible handwriting and transcription mistakes.
  - Patient Engagement: Patients can view their records online, track treatments, and communicate with providers.

  Challenges:

  - High Implementation Costs: Initial setup and maintenance can be expensive, especially for small clinics.
  - Training Requirements: Staff need training to use EHR systems effectively.
  - Data Privacy and Security: Protecting sensitive patient information from breaches is critical.
  - Interoperability Issues: Ensuring compatibility between different EHR systems remains a challenge.

2.3 Functions of the healthcare management system

- **Centralizing Patient Information:**

  Electronic Health Records (EHRs) securely store medical histories, test results, and treatment plans. Patient portals allow patients to access their data and schedule appointments.

- **Optimizing Scheduling and Appointments:**

  Appointment scheduling tools reduce wait times and streamline provider schedules. Resource allocation efficiently manages facilities, equipment, and personnel.

- **Automating Billing and Financial Management:**

  Billing systems handle insurance claims and patient invoicing. Financial reporting tools provide valuable insights for better financial planning.

- **Enhancing Clinical Workflow:**

  Order entry systems streamline lab tests, medications, and other services. Clinical decision support aids providers in making evidence-based decisions.

- **Ensuring Regulatory Compliance:**

  HMS helps organizations comply with healthcare regulations like HIPAA. Automated reports are generated for regulatory bodies.

- **Data Analytics and Reporting:**

  Tools analyze clinical and operational data to identify trends, measure performance, and inform decision-making. Customizable reports provide insights for internal and external stakeholders.

- **Facilitating Communication and Collaboration:**

  communication among healthcare staff. HMS can also connect with external providers, labs, and pharmacies.

2.4 Evolution of Healthcare Management Systems

Healthcare management systems (HMS) have undergone accelerated development across the decades, driven by technological development and the health sector's growing needs **arabadmin2020healthit**. This development can be divided into several distinct stages, each of which represented a turning point in the way health information is managed:

- **Paper stage (Pre-Digital Era)**

  At this point, all medical and administrative processes were done manually using paper records. This includes recording medical history, test results, and prescriptions in paper files stored in lockers .

  Characteristics :

  - Difficulty archiving and retrieving information.
  - Limited access to files by more than one user at the same time.
  - Increased possibility of loss or damage due to fire, moisture or time.
  - Complete reliance on the human factor in recording and reviewing data.

- **Mainframe stage (1960s and 1970s)**

  Some large hospitals, especially in developed countries, have begun using mainframes to perform limited tasks such as billing and payroll management.

  Characteristics :

  - Closed, expensive systems need a special operating environment.
  - It does not directly include clinical or medical aspects.
  - Difficult to modify or expand to fit the requirements of small clinics

- **The personal computer stage (1980 s)**

  With the advent and spread of personal computers, clinics and hospitals began adopting local software systems to manage some administrative and medical functions.

  Characteristics:

  - Use office software to manage appointments, billing, and patients.
  - Store data locally on computers.
  - Lower cost compared to mainframes.
  - Limited interaction between systems and difficulty of integration between them.

- **The electronic medical records stage (1990 s)**

  This stage witnessed a qualitative shift in the medical field through the shift to the use of electronic medical records (EMR), which improved the quality of care and facilitated access to patient information.

  Characteristics:

  - Digitizing medical files and unifying them in an electronic system.
  - Reducing errors resulting from handwriting.
  - Improving communication between healthcare
  - providers within the same institution.
  - Beginning to use clinical decision support tools.

- **Cloud Systems and Analytics Phase (2000–2010)**

  In this decade, health organizations began to rely on cloud architecture to store data and expand access to it, with the introduction of big data analysis tools to improve medical and administrative performance.

  Characteristics:

  - Access to information from anywhere and anytime.
  - Reduce costs related to local infrastructure. Use analytical tools to detect trends and improve quality.
  - The beginning of the adoption of integrated electronic systems (HIS) that include all medical and administrative aspects.

- **Smart Phase (from 2010 to today)**

  This phase represents a comprehensive development in HMS systems in terms of intelligence and integration, integrating advanced technologies such as artificial intelligence (AI), Internet of Things (IoT), telemedicine, and blockchain.

  Characteristics:

- Artificial Intelligence: Improving diagnosis, predicting diseases, supporting decision making.
- Telemedicine: Providing medical consultations over the Internet, especially in remote areas.
- Blockchains: enhancing data security and preventing tampering.
- Wearable devices: monitoring vital signs and automatically sending them to the health system.
- Integrated comprehensive systems: patient management, resources, processes, reporting, and communication within and outside the health system.

3 Healthcare Management System: More than just an information project

Some believe that adopting a healthcare management system is limited to installing software and creating databases. But the truth is that this type of project profoundly transforms the daily way of working within health institutions. It is a culture and organizational change project, not just a digital project.

4 The importance of involving medical teams in the process:

Doctors, nurses, technicians, receptionists, and pharmacists are the end users of the system, therefore:

- **They should be involved from the beginning of the project:**

  through interviews, questionnaires, or brainstorming sessions.

- **Their feedback is taken into account:**

  when designing interfaces, defining functionality, and adjusting workflow.

- **Their training prior to official launch:**

  is necessary to ensure proper use. This early interaction enhances their sense of belonging to the system and reduces resistance to change.

5 Change Management

One of the biggest reasons for the failure of digital projects in the health field is the absence of a clear plan for managing change.

Change management is a set of activities and techniques aimed at facilitating the transition from the traditional system to the new system. Include:

- **Effective communication:**

  Explain the project objectives and schedule clearly to all concerned.

- **Motivation:**

  Highlighting the benefits of the new system, such as speeding up procedures and improving the quality of care.

- **Practical training :**

  Providing training courses according to the role of each user.

- **Post-launch support:**

  Allocate a technical support team during the first weeks.

6 Use of experts and specialists :

Many health institutions are moving to:

- Hiring consultants specialized in healthcare management systems to guide the implementation process.
- Request legal expertise to ensure system compliance with regulatory standards such as:
- HIPAA (United States Patient Information Protection Act)
- HITECH (Law to Support the Use of Information Technology in Health)

7 Expected challenges and solutions

Challenge	**Impact**	**Proposed Solution**
User resistance	Delays in adoption or system rejection	Continuous communication, user involvement, motivation
Lack of training	Misuse or misunderstanding of the system	Practical, role-specific training sessions
Weak infrastructure	Technical issues or poor system performance	Preliminary infrastructure audit before implementation
Privacy concerns	Fear of data breaches	Apply high security standards and ensure legal compliance

Table 1.1: Challenges, Impacts, and Proposed Solutions

8 4 Healthcare Management Systems in the word

8.1 Athenahealth

Athenahealth is a start-up company founded in 1997, which has grown to become one of the world's leading providers of healthcare services and systems based on cloud computing. athenahealth offers an integrated platform that combines medical records management, revenue cycle, patient interaction, and care coordination into one cloud system. With the development of cloud computing technologies, athenahealth has become among the leading providers of cloud solutions for medical practices and health systems. During the first three quarters of 2022, the company registered more than 2,200 new users, reflecting growing confidence in its services.

Main advantages:

- A unified platform that includes clinical and administrative functions.
- It relies on cloud computing, making it easy to access data anytime, anywhere.
- Effective tools for patient interaction and care coordination between stakeholders.

ossible limitations:

- The need for a permanent internet connection to access the system.
- Challenges in customizing the system for practices with special requirements.
- Subscription costs may be relatively high for some small clinics

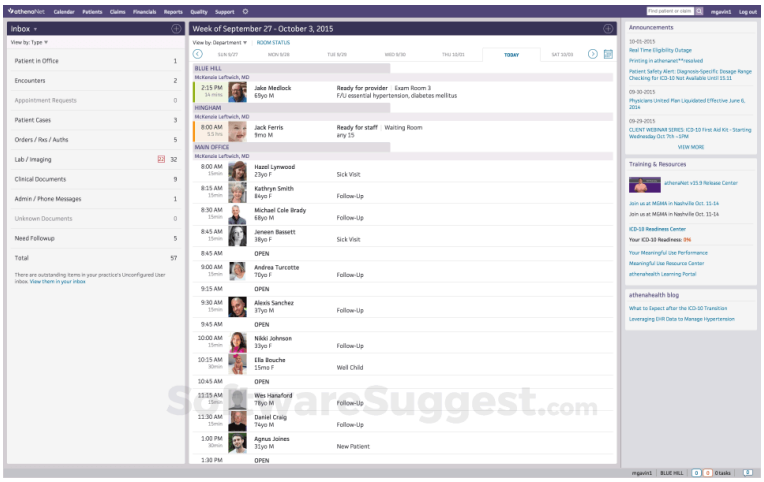

Figure 1.1: Athenahealth

8.2 Epic Systems

Epic Systems is one of the largest and most famous medical software companies in the world, founded in 1979 in the United States. Its systems are used in thousands of hospitals and health

institutions around the world, and are known for their focus on delivering comprehensive and robust electronic health records. Epic currently manages data for more than 250 million patients, and is one of the first companies to embrace the integration of clinical and financial data, as well as its support for medical analytics and artificial intelligence.

Main advantages: Provides integrated electronic medical records (EHR) covering clinical and administrative aspects. Supports interaction between caregivers through data sharing systems (such as Patient MyChart). Provides advanced tools for data analysis and evidence-based clinical decision making.

Possible limitations:

- The cost of licensing and implementation is high, making it more suitable for large hospitals.
- The allocation process is complex and needs lengthy staff training.
- Lack of flexibility in some modifications compared to open source or small-sized systems.

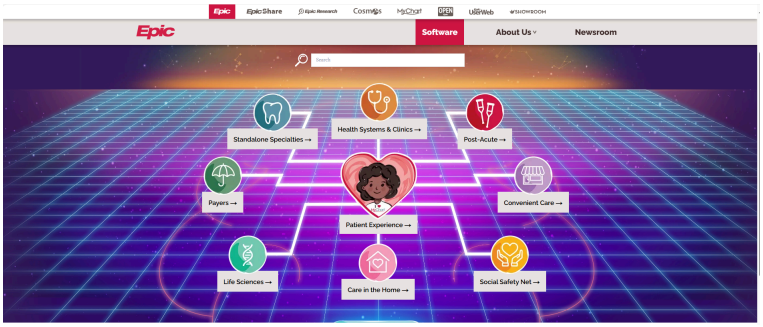

Figure 1.2: Epic Systems

9 Conclusion

The transition from fragmented, paper-based records to Electronic Medical Records is a critical step in modernizing Algeria's healthcare system. While challenges such as data security, interoperability, and user adoption must be addressed, the potential benefits—improved patient care, operational efficiency, and cost savings—are substantial. The proposed system, tailored to Algeria's infrastructure, offers a scalable solution to unify patient data and enhance healthcare delivery. By learning from global examples like Athenahealth and Nordic EHRs, Algeria can leverage technology to overcome current limitations and build a more resilient, patient-centered healthcare system.

Chapter 2

System Design

1 Introduction :

In this chapter, we present a detailed explanation of the system's architecture and internal structure. The objective is to provide a clear understanding of how the system is built, how its components interact, and how it meets the functional and non-functional requirements. We begin by introducing the overall system architecture, highlighting the core components and the relationships between them. This includes the Front-end Application, the Back-end Server, and the Admin Dashboard, each serving a distinct role in delivering the system's functionalities. Each of these components will be discussed in a dedicated section, covering the following aspects:

- **Component Responsibilities:** : What each component is responsible for
- **Internal Elements:** The main classes/modules or services within each component.
- **Data Models** The database structure used to manage and store data efficiently
- **API Endpoints :** The communication interfaces between the frontend and backend.
- **User Interfaces (UI):** Key user-facing views and how users interact with the system.

To formalize and visualize the system structure, we adopted the C4 Model, which consists of the System Context Diagram, Container Diagram, and Component Diagram (for backend). This model helps illustrate both the high-level architecture and the detailed organization of the system components. For modeling system behavior and interactions between actors and system elements, we used Unified Modeling Language (UML) diagrams, including Use Case, Sequence, and Class Diagrams. These diagrams provide a behavioral and structural perspective of the system's workflow and data organization. By the end of this chapter, the reader will have a comprehensive understanding of how the system is architected, how its parts collaborate, and how the technical design aligns with the goals of the project.

2 C4 model

2.1 System Context Diagram :

A context diagram is the top level of system representation within a C4 model. This diagram shows the relationships between the central system and external users or other systems that interact with it. This chart is important for understanding the big picture without going into technical details. Explanation of the components of the chart: MediLink System: The system that facilitates the creation, sharing, and management of prescriptions between doctors, patients, pharmacists, and nurses.

- Doctor: He can create examinations, electronic prescriptions, analyses, schedule appointments, and communicate with pharmacists.
- Patient: He can view his prescriptions, book appointments, and view his medical file.

- Pharmacist: Accesses, verifies and executes prescriptions coming from doctors
- Nurse: Helps carry out medical observations and follow-up
- System Administrator (Admin): Has powers to manage users, permissions, and monitor activities within the system.

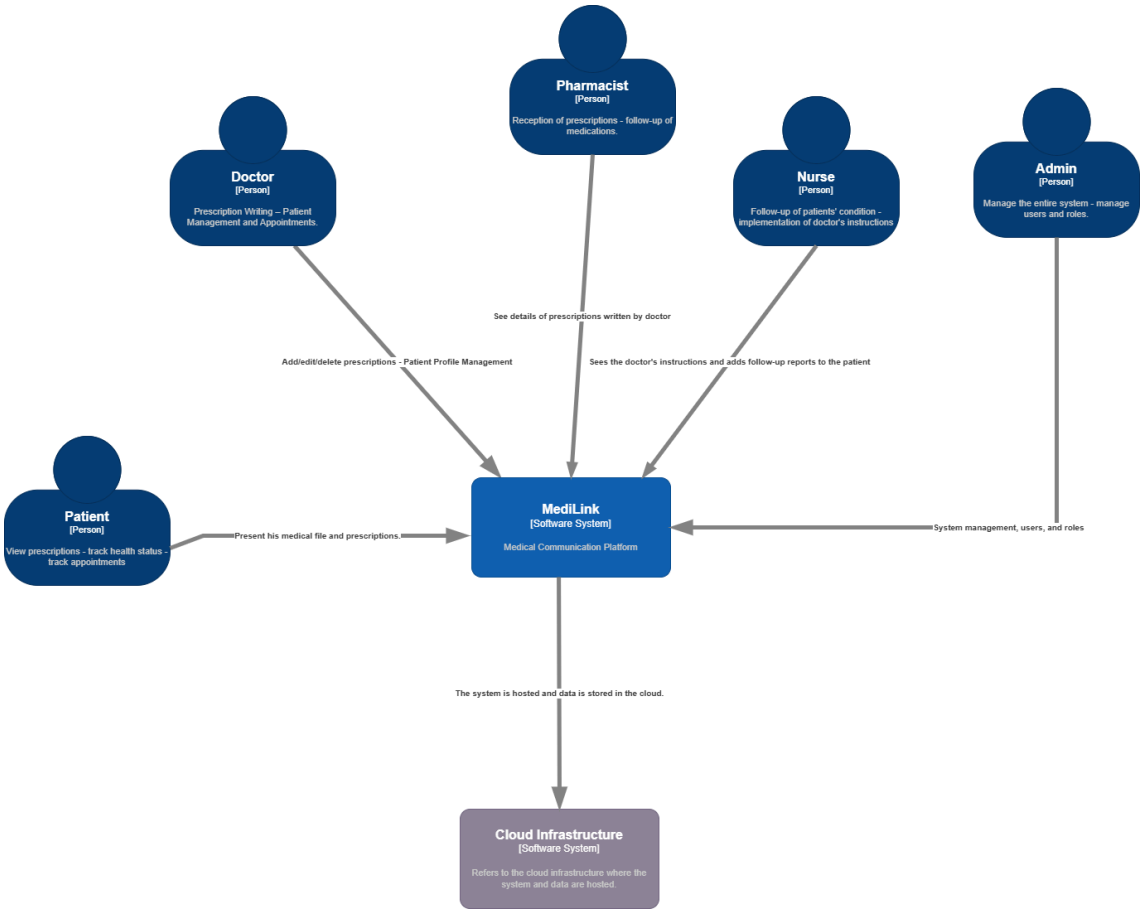

Figure 2.1: C4 Model – System Context Diagram

2.2 Container Diagram

After defining the overall context of the system, I moved on to designing the Container Diagram, which shows how to divide the system into independent logical units, such as the back-end server, user interface, and databases. Through this scheme, I aimed to clarify the distribution of technical responsibilities between these containers, as well as to define the methods of communication and interaction between them using REST APIs, in accordance with the requirements of a multi-tenant architecture.

basic components

Front-end Application

- Developed with React.js

- It represents the user interface with which the doctor, patient, pharmacist, and nurse interact.
- Connects to the back-end server via the REST API.

Back-end Server

- Developed with Node.js and Express.js
- Processes requests, implements business logic, verifies permissions, and generates codes (JWT).
- Manages databases using Sequelize ORM.

Master Database

- It contains a table of tenants that stores information for each institution/doctor.
- Used to define the database for each user.

Multiple Databases (Tenant Databases)

- Each doctor has an independent database.
- Contains patient schedules, appointments, prescriptions, etc.

Admin Dashboard

- Separate interface for full administrative access and system administration.

Cloud Storage (Cloudinary or alternative)

- To store images such as pictures of recipes or tests.

Interaction Components

- The front-end application sends requests to the server via HTTP.
- The server interacts with the database specified according to the Tenant ID.
- Tokens (JWT and Refresh Token) used to secure interaction between components.

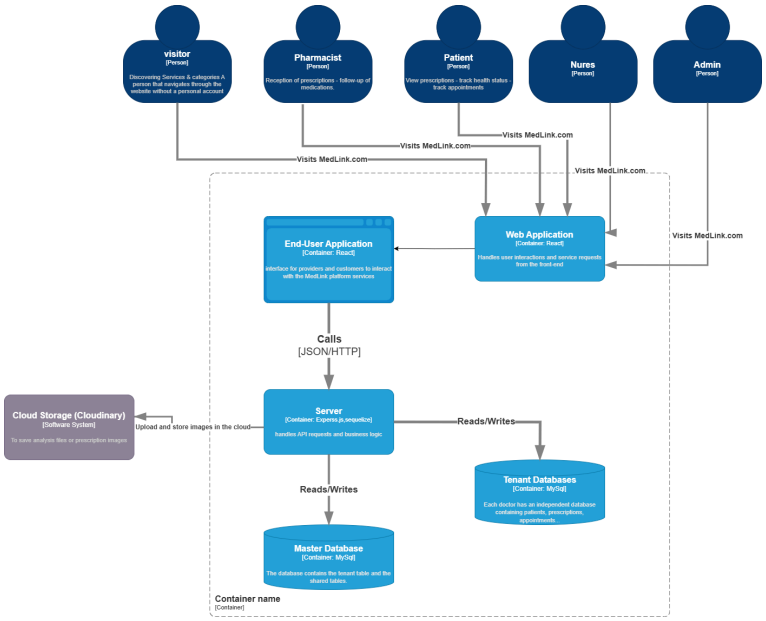

Figure 2.2: C4 Model – Container Diagram

3 Back-end Server

Through this C4 Model, I provided an in-depth visualization of the back-end system design for a medical application that relies on a multi-tenant architecture. This diagram shows how responsibilities are distributed among the various technical components, in addition to the mechanism of their interaction and integration to ensure efficient performance and ease of expansion.

3.1 Main components of the system

Routing (Routes - Express.js)

Within the back-end architecture of the system, I designed the routing layer (Routes) using the Express.js framework. This layer is responsible for receiving incoming requests via the HTTP protocol, and routing them to the appropriate processors. It selects the appropriate endpoints for each request based on the URL and its type (GET, POST, PUT, DELETE). Through this layer, the API structure can be organized and the various functions provided by the system can be facilitated.

Middleware layer (Middleware - Express.js)

The back-end architecture also includes the Middleware layer, which acts as a link between routing and executing business logic. I designed this layer to initially process requests before passing them to controllers. Among the basic tasks of this layer are: verifying the validity of the entered data, performing authentication operations using JWT, and verifying the user's permissions before allowing him to perform some sensitive operations. This layer is necessary to ensure security and organization within the system.[3]

Controller (Controllers - Express.js)

Controllers are one of the most important components of the back-end architecture, as they contain the underlying business logic. I created a group of Controllers to process the different types of requests that arrive from clients (Front-end or Dashboard). These treatments include patient management processes, reservations, prescriptions, and more. It also coordinates between other layers such as the database and the authentication layer.

3.2 Database layer

Master Database (Master Database - MySQL)

This database contains the general data of the system, such as the Tenants table, which is used to determine the database for each user (such as a doctor or pharmacist). In addition to other schedules common to all tenants. Through this rule, registration and login information is managed, determining the settings of each tenant individually.

Tenant Databases - MySQL

I adopted a "Database per Tenant" strategy, so that each tenant has a separate database, containing only his or her medical records, appointments, and prescriptions. This approach allows data isolation, improves performance, and makes it easier to customize the system to each tenant's requirements.

ORM (Sequelize - Node.js)

To facilitate interaction with databases, an ORM library known as Sequelize has been adopted. With them, database queries can be executed in a structured and secure software way, and also enable the transformation of data into processable objects in JavaScript code. This contributes to reducing errors and improving development productivity.

3.3 Cloud storage services

Cloud storage To handle medical files such as test images and prescriptions, the Cloudinary service is integrated into the system. This service allows files to be stored in a secure and easy-to-access manner, with the ability to organize files by tenant, and control access permissions. Cloudinary also supports image compression and improves overall system performance.

3.4 Interaction of components

The system follows a logical sequence in processing requests, where the process begins with the request being received by the Express Router, which routes the request to Middleware. This layer checks the validity and validity of the request. The request then goes to the appropriate

Controller, which implements the required logic and interacts with Sequelize to retrieve or modify the data in the appropriate base. If the request includes files, they are sent to Cloudinary. At the end of the chain, a final response is sent to the client with the required information.

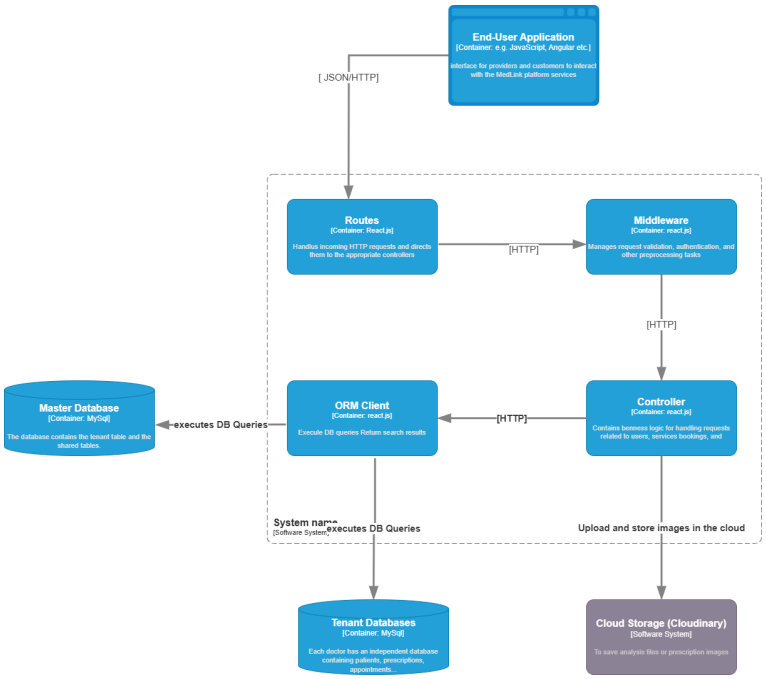

Figure 2.3: C4 Model – Back-end Server Components

4 2. Entity Relationship Diagram

4.1 Master Database

The master database represents the core of the Multi-Tenant Architecture system. It contains tables that relate to general registration, account management, and shared data.

1. **Tenants table**
   This table contains information for each tenant and identifies their contact information for the sub-database associated with them.
   `db_name`, `db_user`, `db_password`, `db_host` are used for dynamic communication with each tenant's database.
   This table is the basis of the principle of isolation between users.

2. **Users table**
   It includes all primary users in the system, whether they are doctors, patients, pharmacists or system administrators, with roles saved via the `role` field.
   Relationship: Both patients and doctors are linked to this table by user ID (`user_id`).

3. **Patients table**
   It stores patients' general personal and medical information such as gender, blood type, and address, and is directly linked to the users table.

4. **Doctors table**
   It contains additional information for doctors only, such as specialty, license number, and number of years of experience.

5. **Message table**
   It allows communication between different parties (doctor, patient, pharmacist, or administrator), specifying the type of sender, receiver, and text of the message.
   It contains the `is_urgent` urgent messages feature to alert parties in critical situations.

6. **Medication schedule**
   Stores information on all drugs, based on international nomenclature, trade names, dosages, packaging and price. This table can be used while writing prescriptions.

7. **Medical records table (medical_records)**
   Records patient-related medical events such as consultations, prescriptions, tests.
   It has a `db_name` field to specify the base where the linked data resides, which facilitates proper routing within a multi-tenant environment.

4.2 Tenant Databases

Each tenant has an independent database that has been carefully designed to ensure complete isolation between tenant data.

1. **Appointments table**
   Saves patient appointment data, specifying appointment status (scheduled, completed, cancelled).

2. **Consultation table**
   It is related to appointments and patients, and contains the diagnosis of the condition, the price of the consultation, and the doctor's notes.

3. **Analysis table**
   It is associated with a consultation and is used to store laboratory test results, such as the name and value of the test.

4. **Prescription table**
   It is linked to the consultation and enables the doctor to issue a prescription containing medications, with the status of the prescription (pending, disbursed, cancelled).

5. **PrescriptionItems table**
   Memorizes the details of the medications prescribed in each prescription, such as the name of the medication, dosage, duration of treatment, and method of use.

6. **Nurses table**
   It saves the data of nurses associated with the doctor, and is used when expanding the system to include nurse tasks such as following up or assisting with examinations.

7. **Doctor's working hours schedule (doctor_working_hours)**
   Specifies the days and hours that the doctor is available to receive appointments, with the ability to activate or disable the specified day.

4.3 Logical Relationship Between Databases

There is a table of tenants in the master database (`master_db`) containing the complete contact information for each tenant database.
When a user logs in or uses the system, their database is dynamically loaded depending on `db_name`.

Figure 2.4: C4 Model – Container Diagram

5 Analyze the system using a Use Case diagram

After providing a comprehensive overview of the system architecture through the C4 diagram, and detailing the database using ERD, we now move on to analyze the system behavior from the user's perspective. The Use Case Diagram is used to identify the main interactions between

users (actors) and the system, and shows the basic functions provided by the application. This diagram helps to understand how the system is used by each actor (such as doctor, pharmacist, patient, or supervisor), and contributes to accurately determining functional requirements.

5.1 Actors and Use Cases

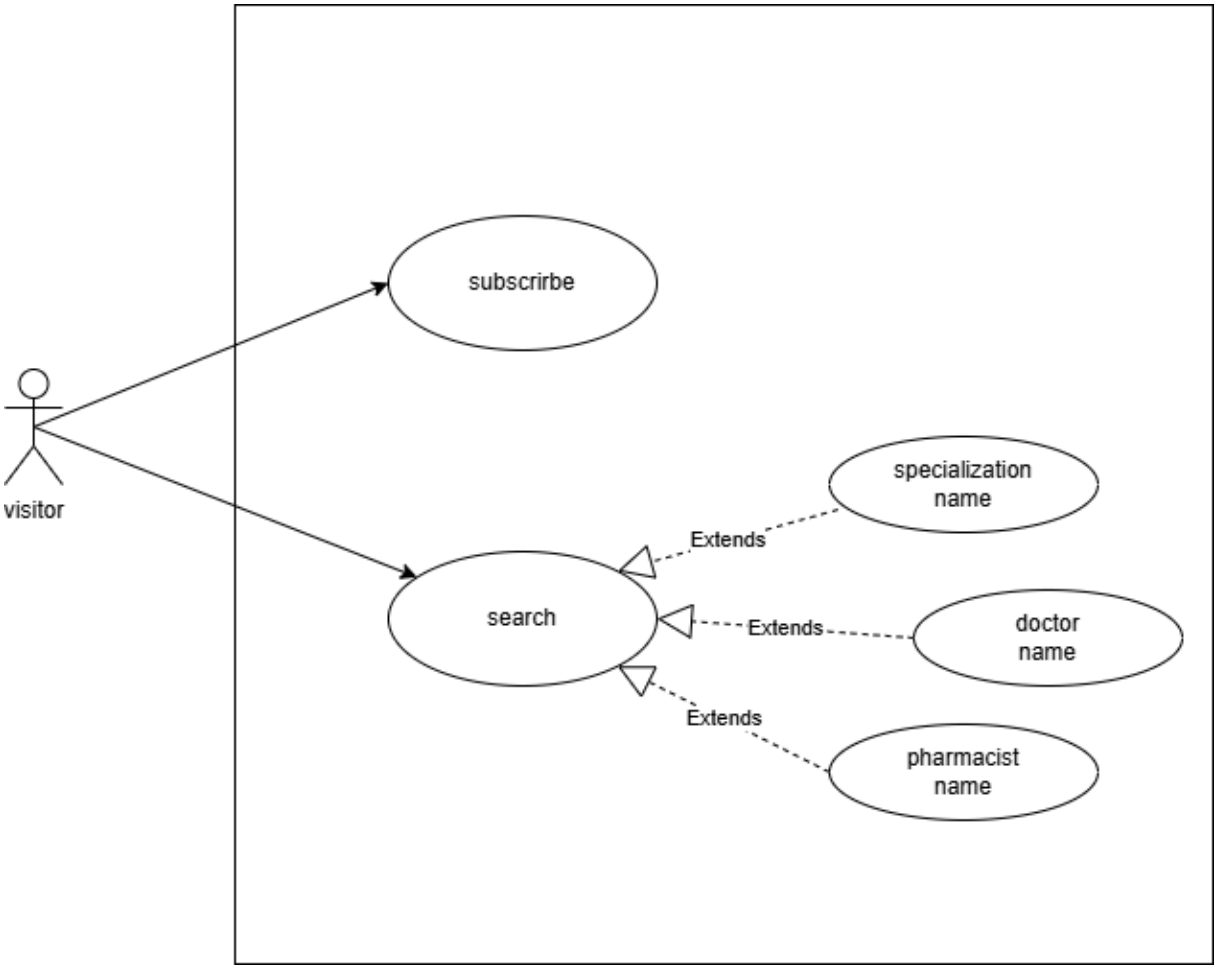

Figure 2.5: Visitor Use Case Diagram

Actors:

- **Visitor:** Any user who has not created an account or is not logged in.

Use Cases:

- **Subscribe:** The visitor can register a new account in the system.
- **Search:** The visitor can search for:
  1. Doctors by name.
  2. Specialties.
  3. Pharmacists by name.

5.2 Patient use case diagram

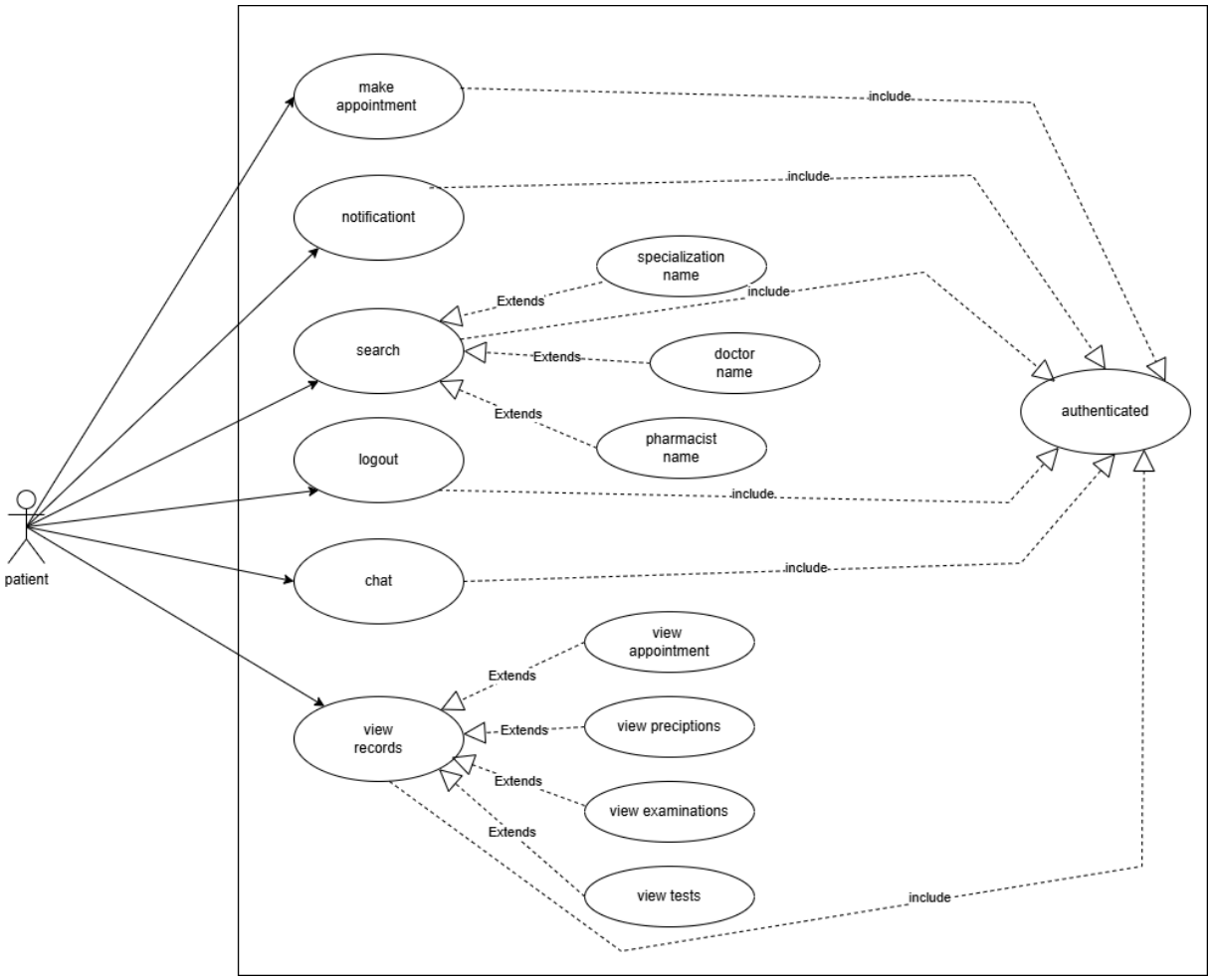

Figure 2.6: Patient Use Case Diagram

Actors:

- **The Patient:** A specific type of user who can perform actions such as consulting their profile, viewing their medical history, and making appointments using their registered information.

Use Cases:

- **Make Appointment:** The patient can schedule an appointment by selecting a preferred date and doctor. This use case includes the following steps:
  1. Searching for a doctor based on specialization or name.
  2. Selecting an available time slot.
  3. Confirming the appointment details.
- **View Records:** The patient can review their personal medical history, which includes:
  1. Viewing past appointments.
  2. Accessing examination records.
  3. Checking test results.

- **Additional Use Cases:**
  - **Search:** The patient can search for doctors or pharmacists by name or specialization.
  - **Chat:** The patient can communicate with healthcare providers (doctors or pharmacists) through a chat feature.
  - **Logout:** The patient can securely log out of their account.
  - **Notification:** The patient receives notifications related to appointments or other important updates.

5.3 Doctor use case diagram

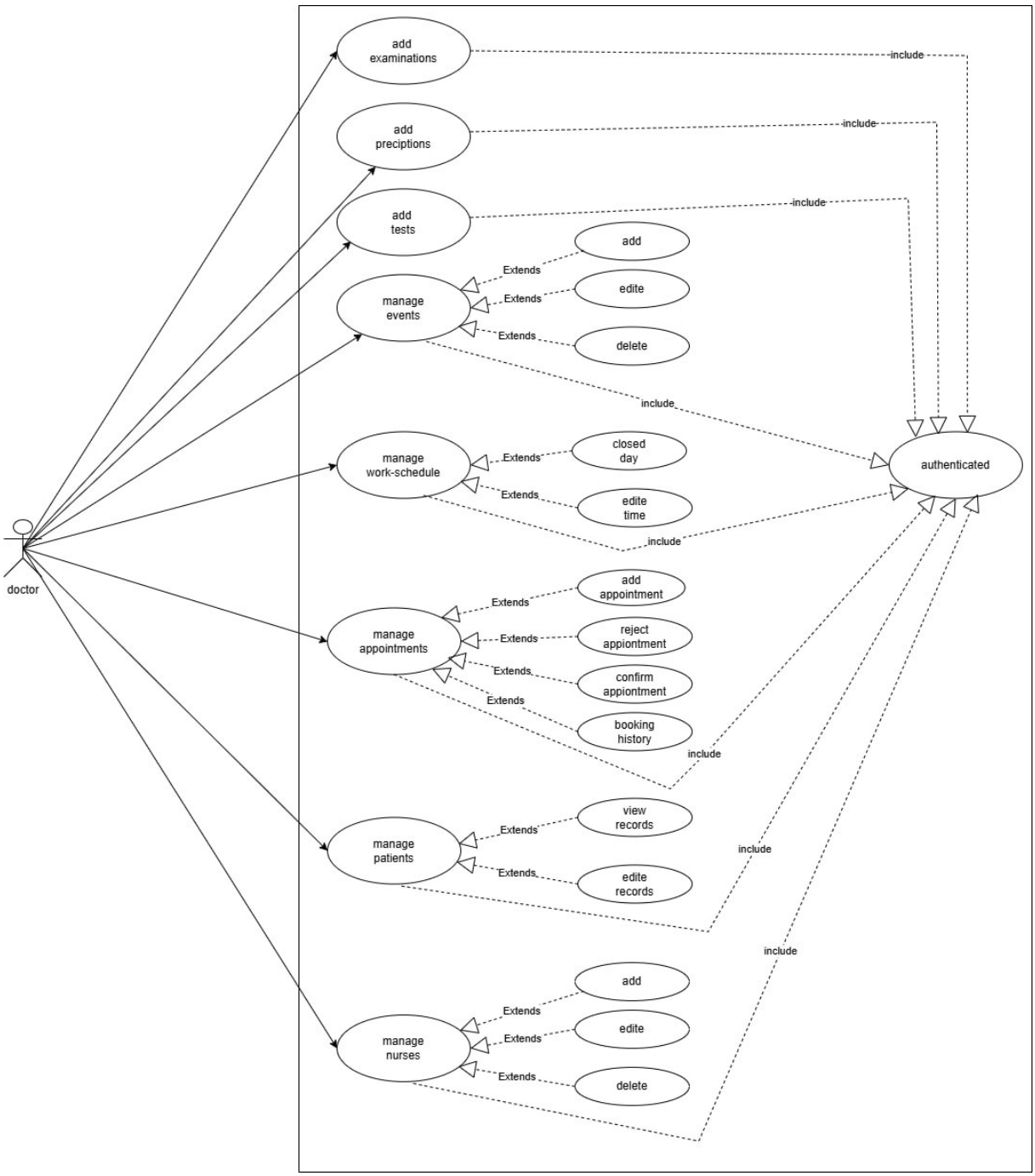

Figure 2.7: Doctor Use Case Diagram

Actors:

- **The Doctor:** A registered user with a role of *"doctor"* who has access to all medical and administrative operations in the system.

Use Cases:

- **Add Examinations:** The doctor can create a new examination report for a patient.
- **Add Prescriptions:** The doctor can prescribe medication to a patient and store it in the system.
- **Add Tests:** The doctor can request medical tests for patients.
- **Manage Appointments:** The doctor can:
  - View all appointments.
  - Confirm or reject them.
  - See the booking history.
- **Manage Patients:** The doctor can:
  - Access patient profiles.
  - Edit their medical data or personal info.
- **Manage Nurses:** The doctor can:
  - Add, edit, or remove nurses under their supervision.
- **Manage Work Schedule:** The doctor can:
  - Set availability, working hours, and holidays.
- **Manage Events:** The doctor can:
  - Create, update, or delete events related to their schedule.

5.4 Nurse Use Case Diagram

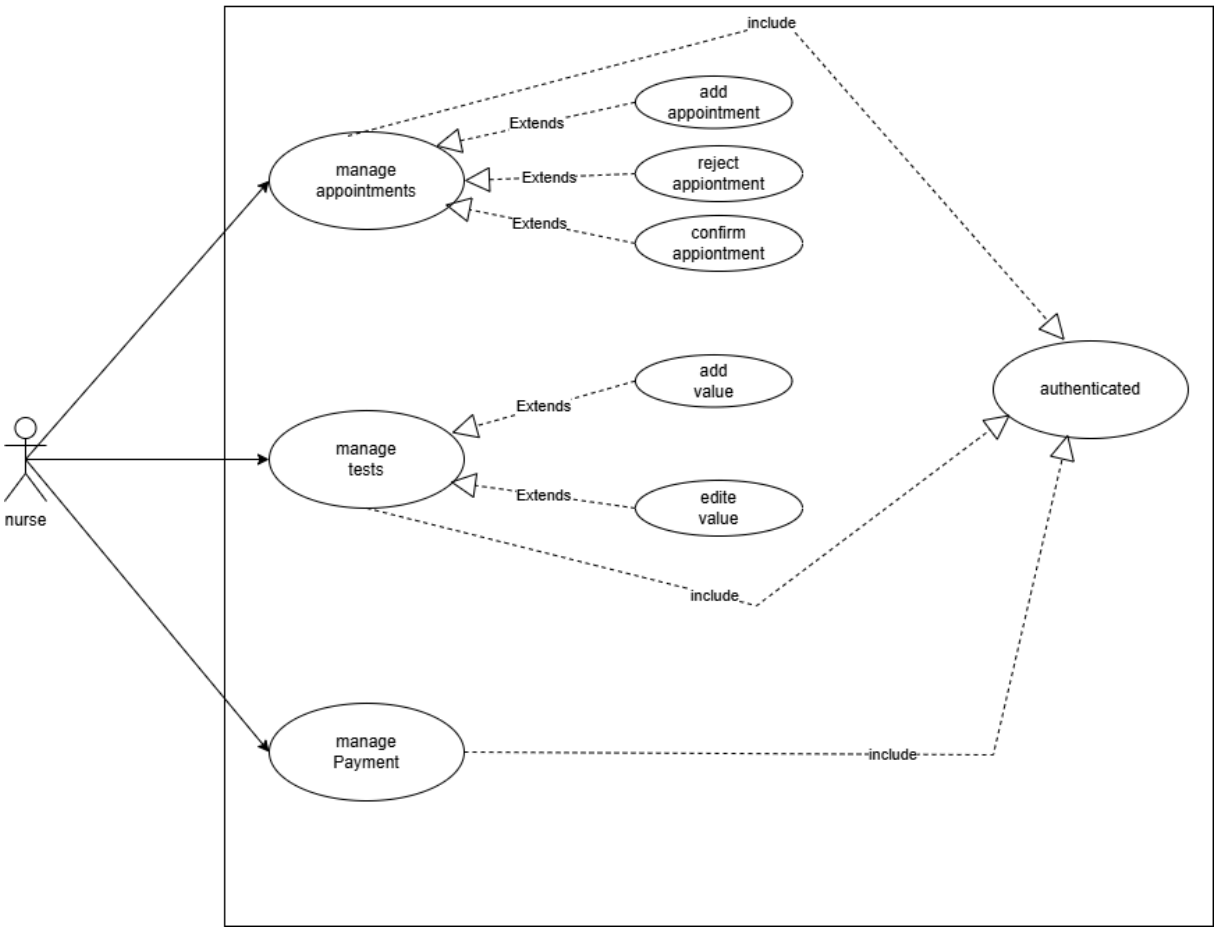

Figure 2.8: Nurse Use Case Diagram

Actors:

- **Nurse**: A user added by a doctor.

Use Cases:

- **Appointment Management:**
  - Schedule an appointment for a patient.
  - Accept or decline appointments.
- **Test Management:**
  - Enter or modify test results assigned to patients.
- **Payment Management:**
  - Process patient payments.

5.5 Admin Use Case List

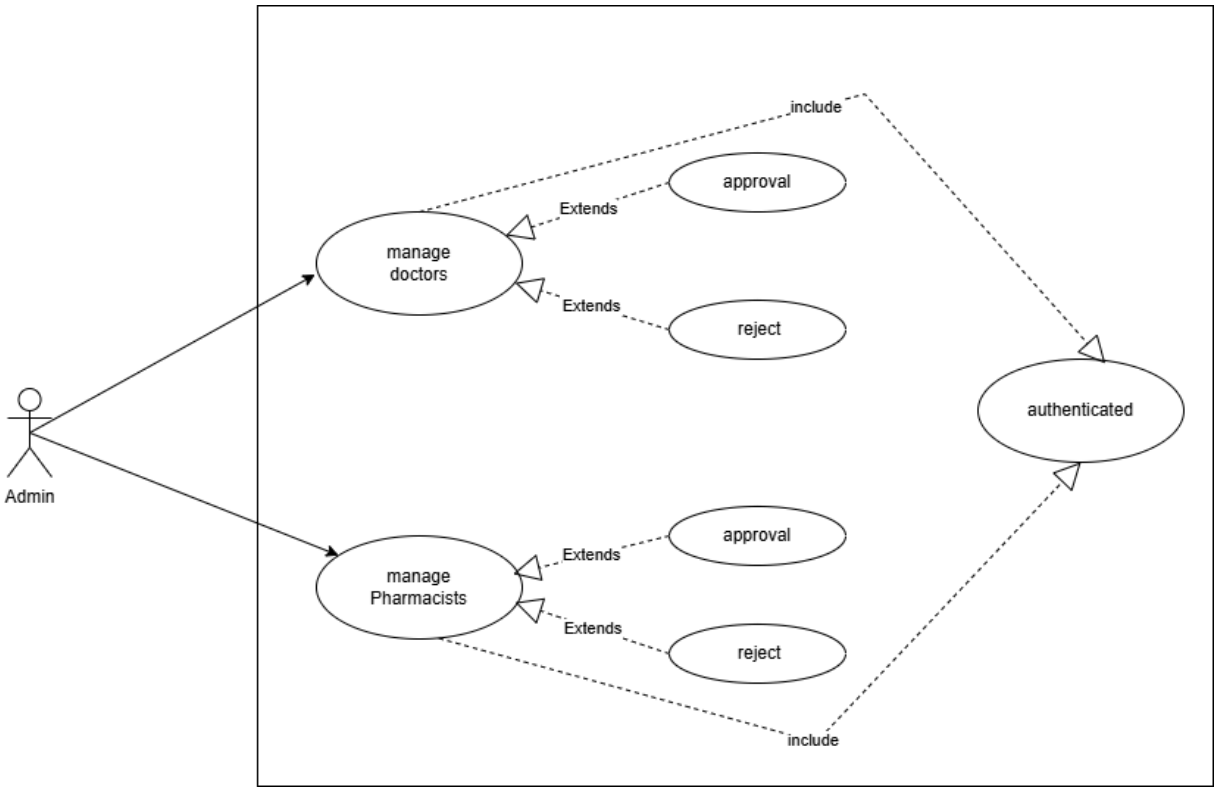

Figure 2.9: Admin Use Case

Actors:

- **Admin**: The user who has full permissions to manage the system.

Use Cases:

- **Doctors Management:**
  - Add, accept, reject, or delete doctors' accounts in the system.
- **Pharmacist Management:**
  - Review pharmacist registration requests.
  - Approve or reject them.
  - Delete pharmacist accounts.

5.6 Pharmacist Use Case List

Actors:

- **Pharmacist**: A user who has an account in the system and can perform some tasks related to prescriptions.

Use Cases:

- **Scan the QR code:** The pharmacist can scan the QR code on the prescription to show the details of the medication prescribed to the patient.

- **Confirm dispensing:** After reviewing the prescription, the pharmacist can record that the medication has been dispensed.
- **Chat:** The pharmacist can use the messaging system to communicate with the doctor in the event of an inquiry about the prescription.

6 Analyze the System Using Sequence Diagrams

After presenting the system architecture using the C4 model, the database structure through the Entity-Relationship Diagram (ERD), and analyzing user interactions via Use Case Diagrams, we now utilize Sequence Diagrams to explore how the system operates step by step.

Each diagram illustrates how the system components — such as the frontend, backend (API layer), and databases (Master and Tenant DBs) — interact in real time during key operations. These diagrams help visualize the flow of requests and responses, clarifying the internal logic of user actions like logging in, booking appointments, or viewing medical records.

6.1 Doctor Login – Sequence Diagram

This sequence diagram demonstrates how a doctor securely logs in to the system:

1. The doctor accesses the system's login page and inputs their email and password via the website's frontend.
2. The frontend sends a `POST` request to the authentication API endpoint (`/login`) with the provided credentials.
3. The backend receives the request and queries the Master Database to locate the doctor’s record using the entered email.
4. The system performs two verification steps:
    - It checks whether the hashed password matches the stored password.
    - It verifies that the account is active by checking if `status = 1`.
5. If both checks pass, the backend retrieves the name of the tenant database associated with the doctor.
6. The backend then generates two tokens:
    - **Access Token**: used for authenticated communication between the frontend and backend.
    - **Refresh Token**: used to obtain a new access token when the current one expires.
7. Finally, the tokens are returned to the frontend, allowing the doctor to securely access protected resources.

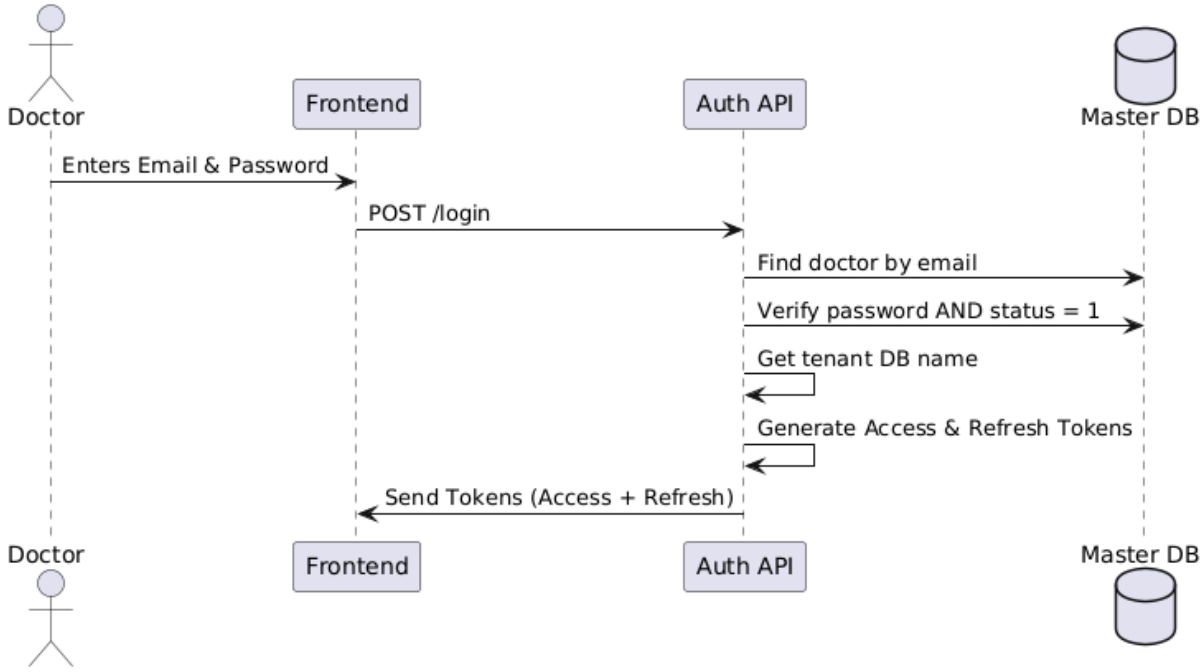

Figure 2.10: Sequence Diagram for login doctor

6.2 Appointment Booking – Sequence Diagram

This diagram explains how a patient books an appointment:

1. The patient chooses a doctor and a time from the frontend.
2. The frontend sends a request to the Appointment API on `/appointments`.
3. The API looks in the Master DB to find the doctor's tenant database.
4. Then it checks if the time is free in the Tenant DB.
5. If the time is available, the appointment is saved.
6. A confirmation message is sent to the patient.

6.3 Appointment Booking – Sequence Diagram

This diagram explains how a patient books an appointment:

1. The patient chooses a doctor and a time from the frontend.
2. The frontend sends a request to the Appointment API on `/appointments`.
3. The API looks in the Master DB to find the doctor's tenant database.
4. Then it checks if the time is free in the Tenant DB.
5. If the time is available, the appointment is saved.
6. A confirmation message is sent to the patient.

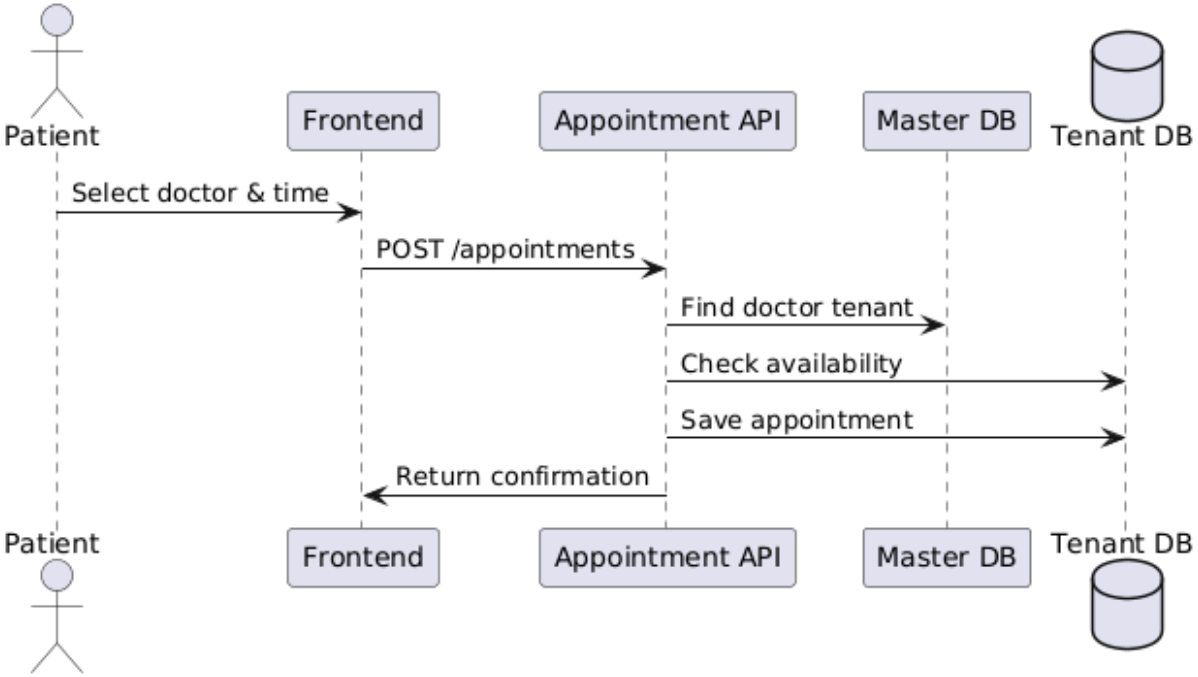

Figure 2.11: Sequence Diagram for Appointment Booking

6.4 View Examinations – Sequence Diagram

This diagram shows how a patient sees their examinations:

1. The patient clicks on `"My Examinations"`.
2. A request is sent to `/examinations` with the Access Token.
3. The Auth Middleware:
    - Checks if the token is valid.
    - Extracts the patient's ID from the token.
    - Allows the request to continue.
4. The API queries the Master DB for the patient's medical records of type `"examination"`.
5. The API receives from Master DB:
    - `related_id` (the consultation ID),
    - `tenant_name` (doctor's database).
6. For each examination, the API fetches consultation details from the doctor's Tenant DB.
7. The API returns the list of examination details to the patient.

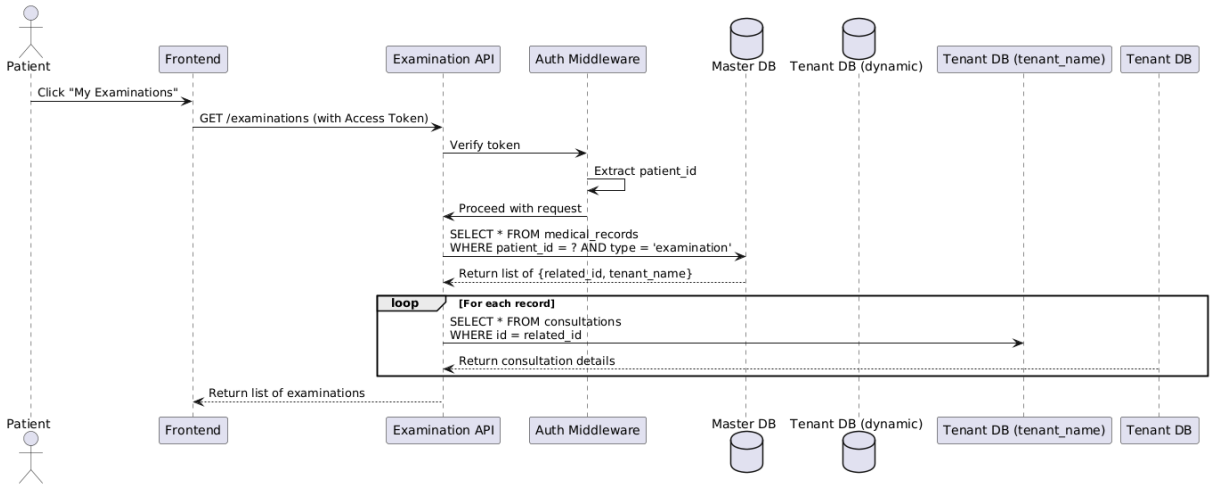

Figure 2.12: Sequence Diagram for Viewing Examinations

6.5 Prescription – Sequence Diagram (Doctor)

This diagram shows how a doctor creates a new prescription with a QR code:

1. Doctor fills the prescription form on the website.
2. The Frontend sends a request `POST /prescriptions` to the Prescription API.
3. The API inserts the prescription into the Tenant Database.
4. The API generates a QR code for this prescription.
5. The API saves the QR code in the Tenant Database.
6. The Frontend receives a success message with the QR code.

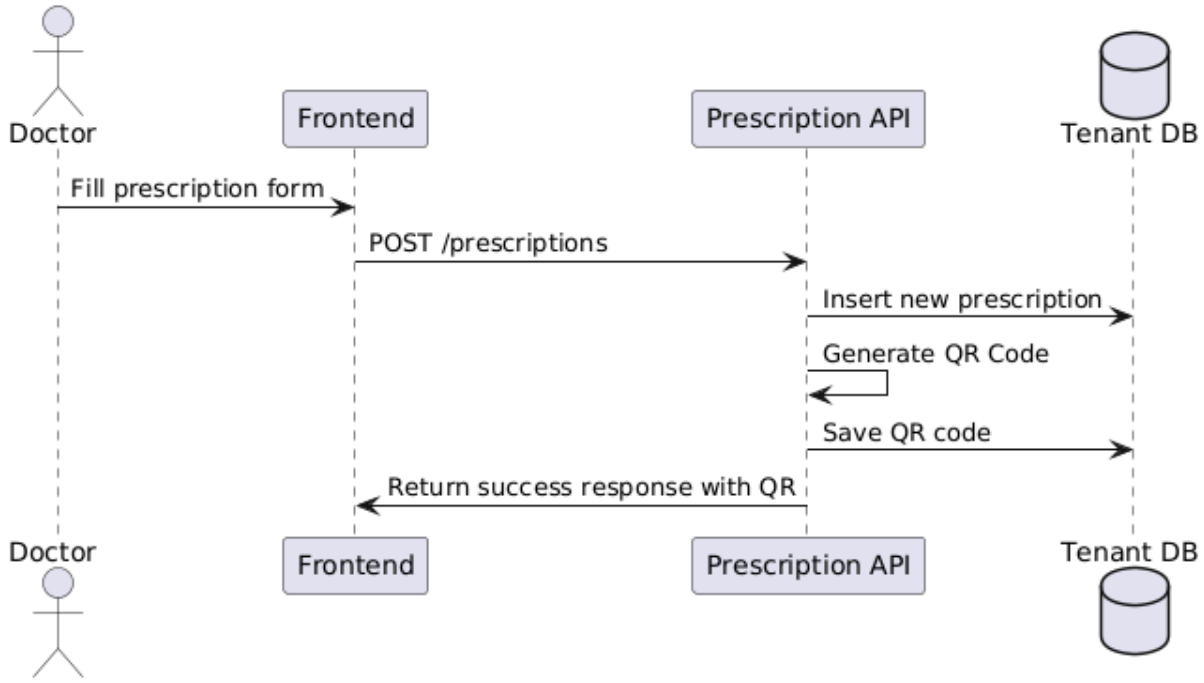

Figure 2.13: Sequence Diagram for Prescription Creation

7 System Analysis Using Class Diagrams

After creating the main structure of the system with the C4 model, designing the database with an ERD (Entity-Relationship Diagram), and showing user actions with Use Case Diagrams, we now use Class Diagrams to show the system's fixed structure. These diagrams provide information about:

- Main entities like User, Doctor, and Appointment, and their details (data fields).
- Relationships between classes, such as inheritance and connections.
- How data moves between parts of the system, like linking a Prescription to a Consultation.

Using these diagrams, we change real-world ideas, like a patient booking an appointment, into a clear software design that developers and stakeholders can understand and use easily.

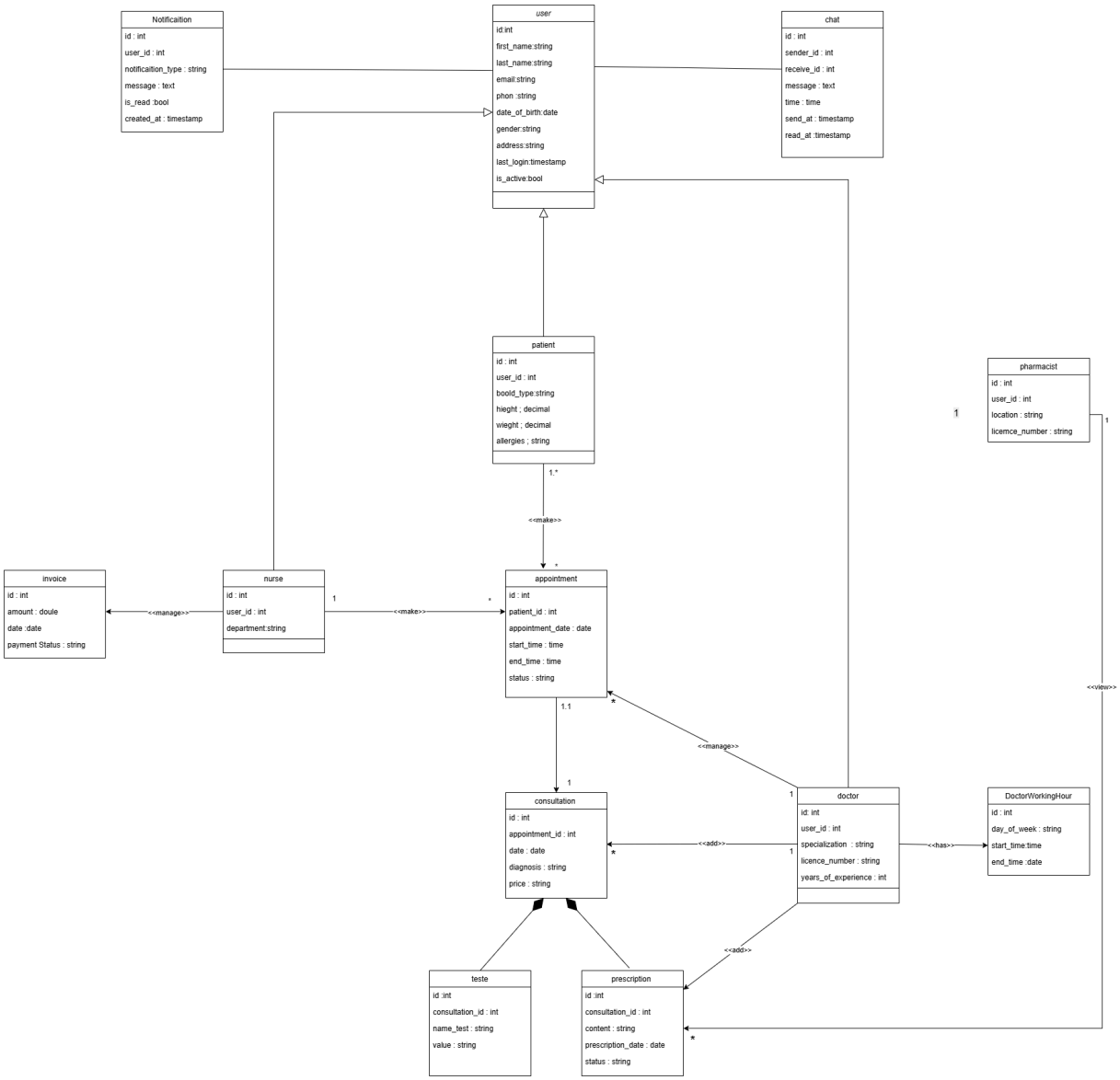

Figure 2.14: class Diagram

1. **User**
   The base class inherited by all other users (Doctor, Patient, Pharmacist, Nurse). It contains general information such as:

   - First Name
   - Last Name
   - Email
   - Gender
   - Address
   - Phone
   - Role

2. **Doctor, Patient, Pharmacist, Nurse**
   These are subclasses of the `User` class:

- **Doctor:**
  - Specialization
  - License Number
  - Years of Experience
- **Patient:**
  - Weight
  - Height
  - Blood Type
- **Pharmacist:**
  - Pharmacy Location
  - License Number
- **Nurse:**
  - Department they work in

3. **Appointment**
   Represents the booking between a Patient and a Doctor. Includes:
   - Date
   - Time
   - Status (Completed, Rejected)
4. **Consultation**
   Occurs after an Appointment. Includes:
   - Diagnosis
   - Notes
   - May include:
     - Prescription
     - Medical Tests
5. **Prescription**
   Linked to a Consultation. Contains:
   - Prescription Details
   - Date
6. **Tests**
   Associated with a Consultation. Includes:
   - Test Name
   - Result

7. **Chat**
   Enables communication between Doctor-Pharmacist or Doctor-Patient. Includes:
   - Sender
   - Receiver
   - Message
   - Time
8. **Notification**
   Alerts users about:
   - New Appointments
   - Messages
   - Medicine Status
9. **Invoice**
   Linked to a Patient. Used for:
   - Consultation Payment
   - Billing

8 Conclusion

This chapter provided a structured and in-depth overview of the system's design by integrating both architectural and modeling approaches. Through the use of the C4 Model, we illustrated the system at multiple levels of abstraction, starting with the System Context Diagram to show the system's environment and external interactions, followed by the Container Diagram to break down internal components, and finally detailing the Back-end Server Components to demonstrate internal logic and responsibilities. To further support the understanding of the system's structure and behavior, we employed several UML diagrams—including Use Case Diagrams to capture user interactions, Sequence Diagrams to depict dynamic flows, and Class Diagrams to define the system's static structure. Additionally, the Entity-Relationship Diagram (ERD) was used to represent the logical data model and the relationships between entities within the database. Together, these models and diagrams offer a comprehensive view of the system's architecture, interactions, and data organization. They serve as a solid foundation for development, testing, and future maintenance, ensuring that the system design aligns closely with both functional and non-functional requirements.

Chapter 3

Smart Medical System: AI, Multi-Tenancy, and Real-Time Integration

1 Introduction:

In recent years, the world has witnessed an unprecedented digital revolution represented by the tremendous progress in artificial intelligence technologies, most notably Large Language Models (LLMs), which have become the cornerstone of the development of intelligent systems, especially in areas that require natural language processing and human interaction. These models have become able to understand complex texts, generate content, and provide accurate and natural answers, which has provided broad opportunities to improve the user experience in many vital sectors, including the health sector. In this chapter, we highlight how modern artificial intelligence technologies, especially large language models such as the Gemini model, are employed within the developed medical system, with the aim of improving the quality of services provided, facilitating the interaction process between patients, doctors and pharmacists, and providing intelligent and immediate support without the need for direct human intervention.every time. We begin by defining large language models, and clarifying their structure based on the Transformer architecture, which represents a qualitative shift compared to traditional models. Then we address ways to use these models within software systems, whether through direct interfaces or through programming interfaces (APIs). Furthermore, we discuss how to integrate the Gemini model into the medical system through the use of a special library in the Node.js environment, and employ it in developing an intelligent medical assistant (Chatbot) that interacts with patients in simplified Arabic, and contributes to analyzing questions and converting them into understandable or actionable queries.On the database.The chapter also reviews the architectural challenges and solutions that have been adopted in the system architecture using the Multi-Tenancy model with "database for each tenant" technology, which ensures the privacy and confidentiality of the data of each doctor and their patients. We show the benefits of this model, the mechanism to create dynamic databases when registering, and the management of their contact when logging in.Finally, we focus on the aspect of real-time interaction through the use of the WebSocket protocol instead of traditional methods such as HTTP Polling, as WebSocket has been adopted to ensure immediate and smooth communication between the various

users of the platform (doctors, pharmacists, and patients) in real time, which enhances the effectiveness of the system and provides an advanced user experience.Meets the requirements of contemporary medical reality.

2 Large language models

2.1 Definition

Large Language Models (LLMs) are artificial intelligence models based on deep learning techniques, previously trained on huge amounts of text data with the aim of understanding natural language understanding and generating human-like texts.Text Generation LLMs are based on Transformer Architecture, which consists of two main components:

- **Encoder** to analyze and understand context.
- **Decoder** To produce texts based on what has been understood

These models are characterized by their ability to process the entire input sequence at once via a mechanism known as self-attention, making them more efficient and accurate than traditional models such as recurrent neural networks (RNNs).

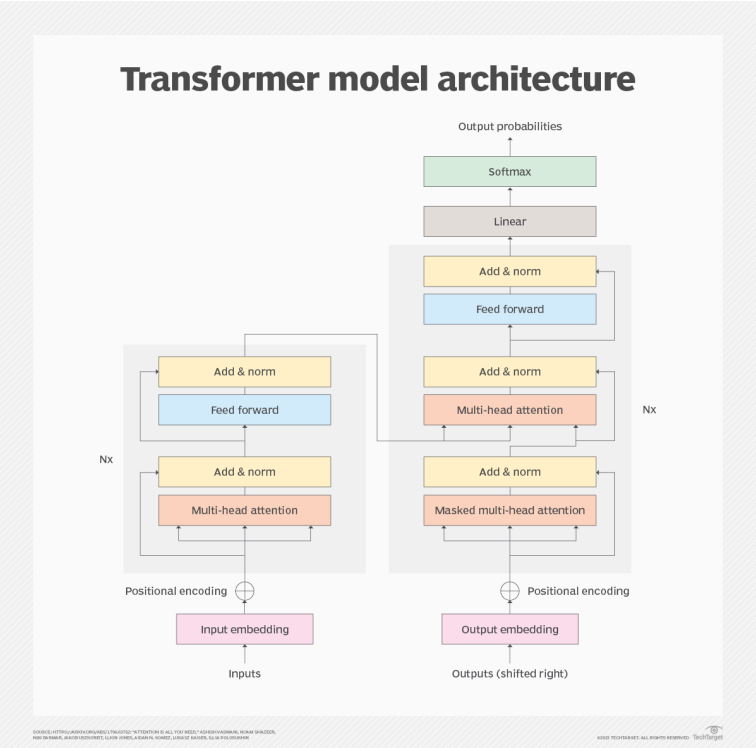

Figure 3.1: Transformers model Architecture

LLMs also rely on the self-supervised learning method, where they learn from the texts themselves without the need for direct human supervision, which enables them to acquire language rules, expand vocabulary, and understand general information.

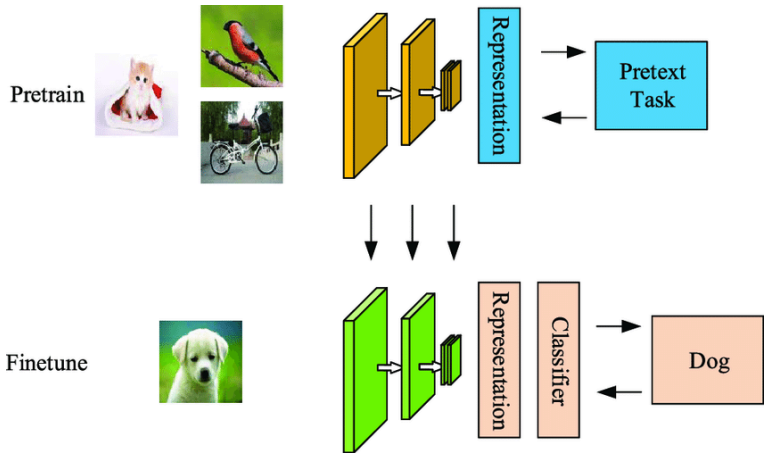

Figure 3.2: Self-supervised learning method

These models range in size from millions to hundreds of billions of transactions, and are trained on huge resources such as: Common Crawl project (over 50 billion web pages) Wikipedia encyclopedia (about 57 million pages)

2.2 Methods of using LLM in software systems

- **Direct use** The user handles the form via a ready-made interface such as the ChatGPT or Gemini website, without the need for programming
- **Use via API** The model is employed within a software project through an API key, allowing the application to intelligently interact with users on a built-in basis, whether to generate scripts, aids, or intelligent decisions.

3 Using the Gemini API in the project

3.1 Introducing Gemini

The Gemini model is one of the large language models (LLMs) developed by Google.It is designed to process multiple types of data (text, images, codes) and supports more than one language, including Arabic. What distinguishes it is its ability to understand complex context, produce accurate responses, and analyze unstructured inputs and transform them into meaningful structured outputs.[4]

3.2 Method of integration within the system

After registering on the Google AI Studio platform, I obtained Gemini's API Key and then integrated it into the system I develop using JavaScript and the Node.js environment. Steps to use:

- 1- Install Gemini library: npm install @google/generative-ai
- 2- Form initialization and invocation:

```
const { ChatGoogleGenerativeAI } = require("@langchain/google-genai");

const chatModel = new ChatGoogleGenerativeAI({
  apiKey: process.env.GOOGLE_API_KEY,
  model: "gemini-1.5-flash",
});
```

Figure 3.3: Gemini Model Configuration

3.3 Areas in which artificial intelligence was used in the platform

Currently, integrating artificial intelligence into the system is a vital step to provide more effective and intelligent health services. It has been employed in several central areas aimed at improving interaction with the user, facilitating access to medical information, and reducing pressure on the medical staff by providing smart support in real time.This is our goal to integrate some functions that rely on artificial intelligence

Create an intelligent medical assistant (Chatbot)

Within the objectives of this project, an intelligent medical assistant (Medical Chatbot) was developed that relies on artificial intelligence technologies, especially Google's Gemini model, to act as an interactive intermediary between the patient and the medical system, and helps provide general medical answers, in understandable Arabic, and in a simplified and practical manner. **Basic idea** The assistant's work is based on two main stages: Language Normalization stage: Many patients express their symptoms or questions in colloquial language or in non-medical terms. The question is first analyzed and its true meaning determined, then reformulated into a formal and clear medical question.

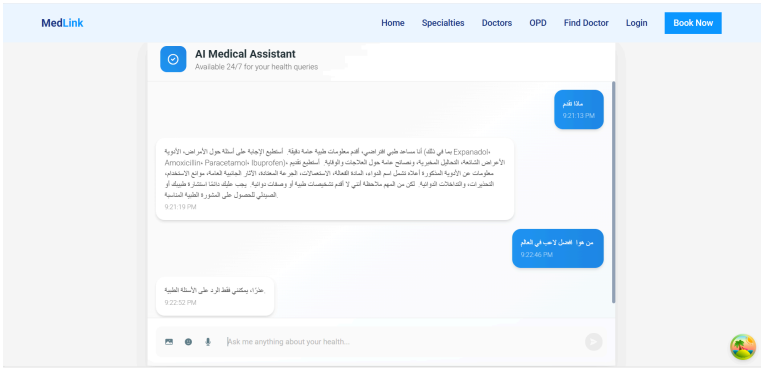

Figure 3.4: Medical assistant (Chatbot)

Answer Generation stage:

After converting the question into a suitable form, it is passed to the Gemini form. The model generates an answer based on general knowledge, taking care to be: Clear and in classical Arabic. Non-diagnostic (does not identify the disease or recommend a specific medication without medical supervision). Directed for general guidance only.

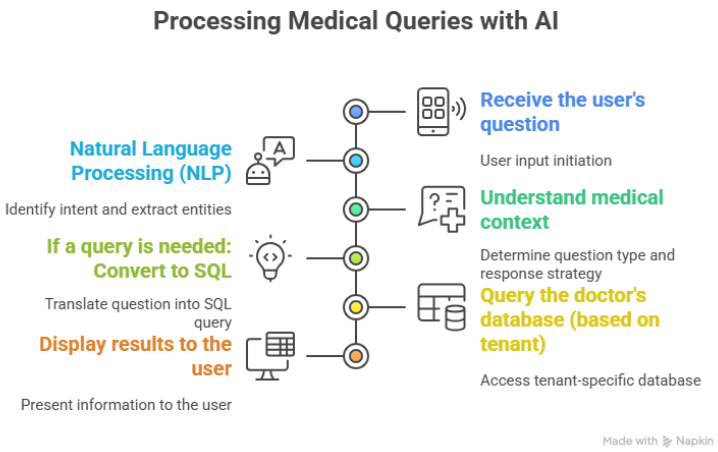

Figure 3.5: Processing medical Queries with AI

Converting questions into queries (Query Understanding)

This feature is one of the most prominent applications of artificial intelligence within the system, as it enables the user to interact in his natural language without the need for technical knowledge in a manner similar to Natural Language to SQL systems. **General idea:**

Converts a patient or physician question written in natural language into a SQL query that executes directly on the database, taking into account medical identity and context within the Multi-Tenancy environment (where each physician has an independent database).

4 Multi-Database Management

In modern applications, especially those dealing with sensitive medical data and multiple user roles such as doctors, pharmacists, and patients, it's essential to ensure both data isolation and security. To meet these requirements, this project adopts a Multi-Tenancy model using the Database-per-Tenant strategy. In this approach, each doctor has a separate database that contains their patients' records, prescriptions, appointments, and messages. This setup ensures data privacy, scalability, and ease of management.[5]

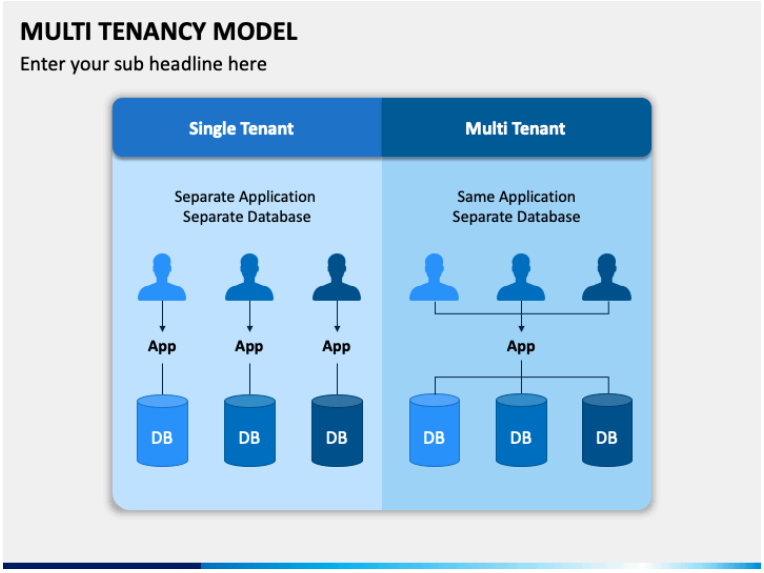

Figure 3.6: Single tenant vs multi tenant

4.1 Benefits of Using Multi-Tenant Architecture

- **Cost Efficiency:** Multi-tenant architecture enables resource sharing across tenants using a single instance of the application. This reduces the need for duplicate hardware, software, and maintenance, resulting in lower operational costs. These savings can be passed on to customers through more competitive pricing.
- **Scalability:** The architecture allows the application to easily scale by allocating additional resources to the shared instance as needed. There's no need to create separate environments for each tenant, making it easier to handle growth and fluctuations in demand.[6]
- **Simplified Maintenance and Updates:** Since all tenants use the same codebase, updates, bug fixes, and new features can be deployed once and made available to everyone instantly. This streamlines maintenance and ensures all users benefit from the latest improvements and security patches.
- **Faster Deployment and Onboarding:** New tenants can be quickly added to the existing system without setting up new installations. This reduces setup time and allows faster time-to-market for new clients.
- **Centralized Management:** Multi-tenancy provides centralized control over the application, making it easier to monitor performance, manage configurations, and troubleshoot issues—all from a single interface.
- **Enhanced Collaboration and Data Sharing:** In certain use cases, tenants can benefit from controlled data sharing or collaborative features. This creates opportunities for interaction between tenants, fostering a community-like environment when needed.
- **Improved Security:** Using separate databases for each tenant helps isolate data and reduce the risk of large-scale breaches. Despite the complexity of cloud environments,

major cloud providers offer robust security tools that can be easily enabled to protect customer data.

4.2 Overview of Multi-Tenancy Models

Model	Description	Pros	Cons
Single DB, Shared Tables	All tenants share the same tables using a `tenant_id` column	Easy to set up	Low security, harder to scale
Single DB, Separate Tables	Each tenant has their own set of tables in the same database	Moderate logical isolation	Hard to manage tables
Separate DB per Tenant	Each tenant has a completely separate database	Highest level of isolation and security	More complex and costly to manage

Table 3.1: Multi-Tenant Database Models Comparison

4.3 Project Architecture

The system is built with the following structure:

- **Central Master Database (master-db)** This database stores information about all registered doctors in a tenants table
- **ndividual Databases for Each Doctor** Examples: doctor-omar-db, doctor-samira-db Each database contains the same core tables: appointments

  nurses

  consultation

  prescription

  prescriptionItems

  doctor-working-hours

4.4 Dynamic Database Connection

When a doctor logs in:

- The system extracts their identity from the token.
- It queries the master-db to find the name of the doctor's database.
- A dynamic connection is established using Sequelize to access the doctor's specific database.
- The system then handles actions such as viewing patients or adding prescriptions.

Figure 3.7: Tenant database connection

4.5 Automatically Creating a New Database

When a new doctor registers:

- A new entry is added to the tenants table in master-db
- A new database is automatically created (e.g., tenant-db-40).
- The necessary base tables are initialized (Appointment, prescriptions, etc.).

Figure 3.8: Gemini Model Configuration

4.6 Challenges of This Model

While this model offers many benefits, it also presents some challenges:

- **Managing Multiple Connections:** Keeping multiple database connections active without hurting performance.
- **Unified Schema:** Ensuring all databases share the same table structure and logic.
- **Deployment Complexity:** Automating database creation and updates.
- **Security Risks:** Avoiding mistakes that could result in data leaks or wrong database access.

5 Using WebSocket in the System

In any system that requires fast and direct communication between users, such as our medical platform that connects doctors, pharmacists, and patients, having a real-time communication channel is essential. For this reason, we chose to implement WebSocket instead of traditional techniques like HTTP Polling, due to its superior speed and efficiency in message exchange.[7]

5.1 What is HTTP Polling?

HTTP Polling is an older method where the browser or application sends repeated HTTP requests at regular intervals (e.g., every 5 seconds) to ask the server: "Are there any new messages?" The server then responds with new data if available, or with an empty response if nothing has changed.

How it works:

- The client sends a GET request to the server
- The server responds with messages or an empty response.
- After a few seconds, the client sends another request

Drawbacks:

- High server load (due to repeated requests)
- Delays in receiving messages (based on polling interval).
- Not practical for real-time chat.

5.2 - What is WebSocket?

WebSocket is a protocol that allows a persistent, bidirectional connection between the client (browser or app) and the server. Once the connection is established, both sides can send and receive messages instantly without needing repeated requests.[8]

How it works :

- A WebSocket connection is opened after login.
- The connection stays open during the entire session.
- When a user sends a message, it reaches the other party instantly

Benefits:

- Real-time interaction.
- Low resource usage.
- Perfect for instant messaging applications.

6 Conclusion:

In concluding this chapter, it is clear that integrating artificial intelligence models, especially large language models such as Gemini, into digital health systems represents a strategic step towards smart transformation in the provision of medical services. Leveraging the capabilities of these models to understand natural language and generate responses enhances the system's ability to provide immediate and personalized support to the user, whether through a smart medical assistant or by converting questions into actionable queries. Adopting a multi-tenant model using a database for each doctor achieves the highest levels of privacy and ensures flexible and secure data management, taking into account scalability and speed of deployment. From an interactive standpoint, the use of WebSocket allowed building a real-time communication channel that enhances users' interaction with the platform quickly and effectively. All of these technologies together form an integrated and advanced structure for a smart health system that seeks to improve the patient experience, facilitate the tasks of medical staff, and contribute to the development of the health care system based on the latest technology.

Chapter 4

IMPLEMENTATION AND REALIZATION

1 Introduction

In this chapter, I have made a comprehensive presentation of the platform that I have developed, focusing on the most outstanding functions that it offers to each category of users (doctor, pharmacist, patient). It also reviewed the technologies and programming languages that it relied on in building the front and back ends, in addition to the development environment and tools that it used during the stages of developing the system and organizing its components.

2 Presentation of the System

2.1 Login

The login page allows the doctor, pharmacist and patient to access their accounts after verifying their email and password.

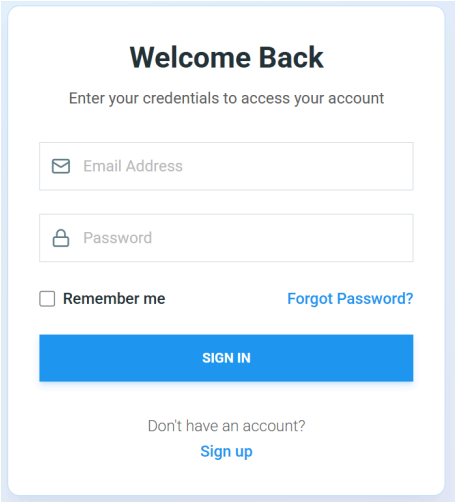

Figure 4.1: login page

2.2 Doctor registration

The doctor registration page consists of four stages, including: personal information, account details, professional information that explains the role of the doctor, and uploading a professional personal photo.

Personal information

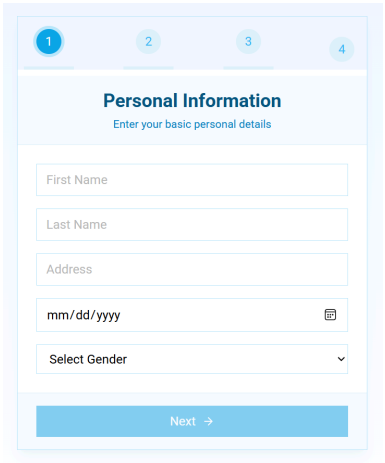

Figure 4.2: doctor registration-personal information

Account details

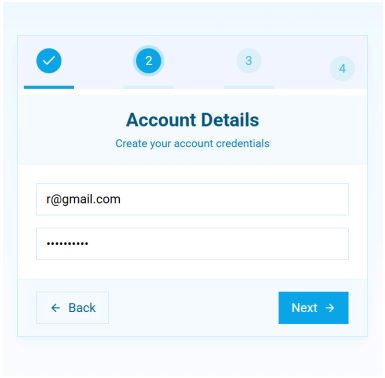

Figure 4.3: doctor registration-account details

Vocational information

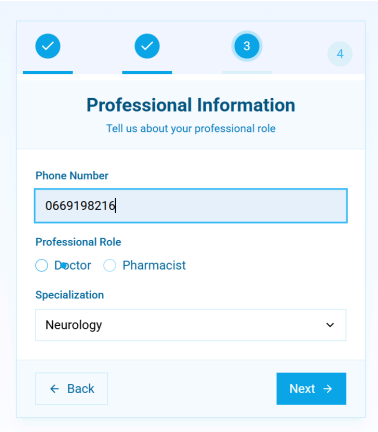

Figure 4.4: ddoctor registration-vocational information

Uploading a professional personal photo

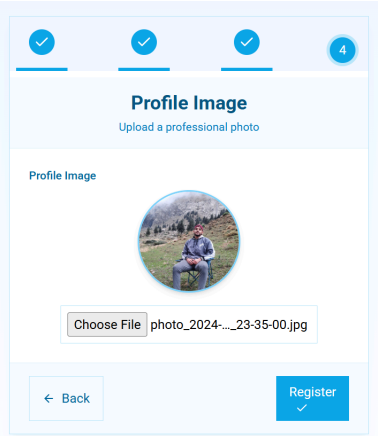

Figure 4.5: Profile image

After completing the four stages, the doctor moves to the verification stage, which requires raising the medical ID and personal ID to verify the validity of the data.

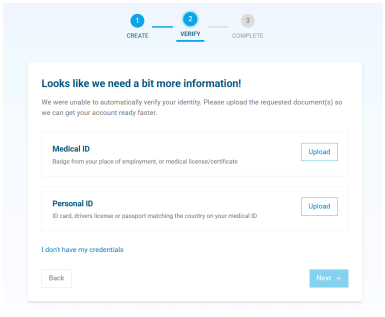

Figure 4.6: Doctor registration-verification

2.3 Doctor Dashboard

home page

The main doctor's control panel page displays important statistics and latest activities to facilitate follow-up of cases and appointments.

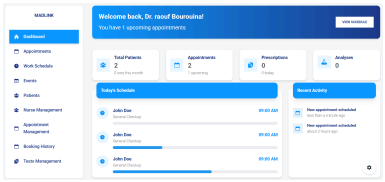

Figure 4.7: Doctor-Dashboard dashboar

Appointments

The appointment booking page displays all appointments, with labels for today's appointments and completed appointments for easy follow-up and organization.

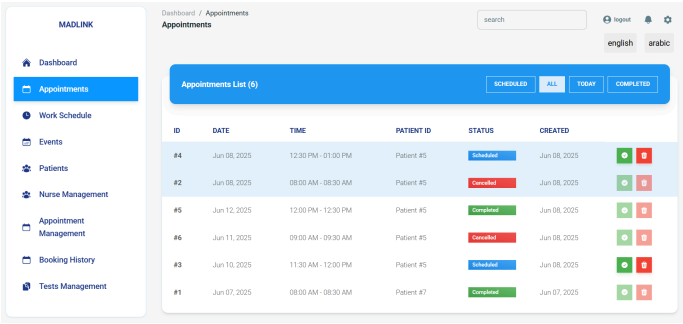

Figure 4.8: Appointments

Patient Review page

The Patient Review page appears to the doctor after accepting the appointment, through which he can add a medical report, prescription, or request tests for the patient.

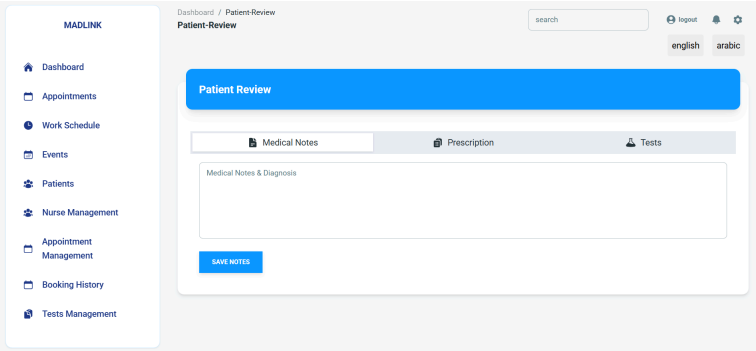

Figure 4.9: Doctor-Dashboard Patient Review

Work schedule

The work schedule page enables the doctor to specify work days, start and end times for each day, with the ability to disable any day in the event of unavailability

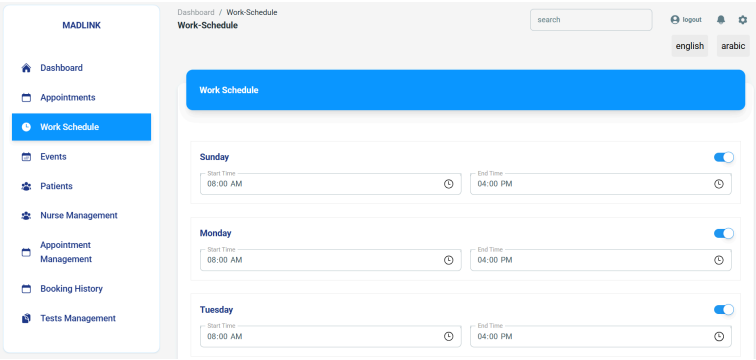

Figure 4.10: Doctor-Dashboard Patient Review

Events page

The Events page allows the doctor to add a new event, edit or delete previous events to easily organize his schedule.

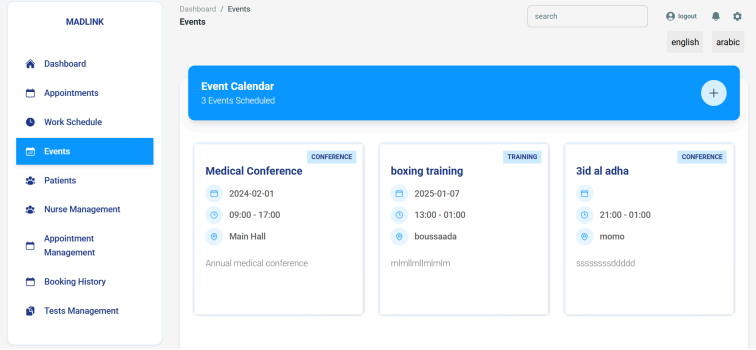

Figure 4.11: Doctor-Dashboard Events

patients page

The patients page displays only the patients whom the doctor has examined, with the ability to view their files, including prescriptions, examinations, and tests,without the ability to access the patient files of other doctors.

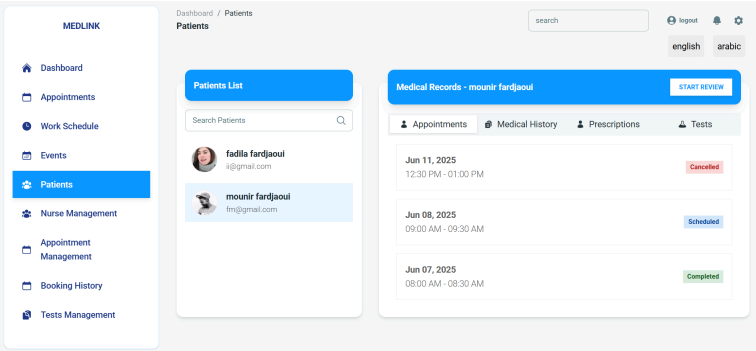

Figure 4.12: Doctor-Dashboard Patients

Nurse Management

The Nurse Management page allows the doctor to add a new nurse, modify nurse information, or delete them as needed

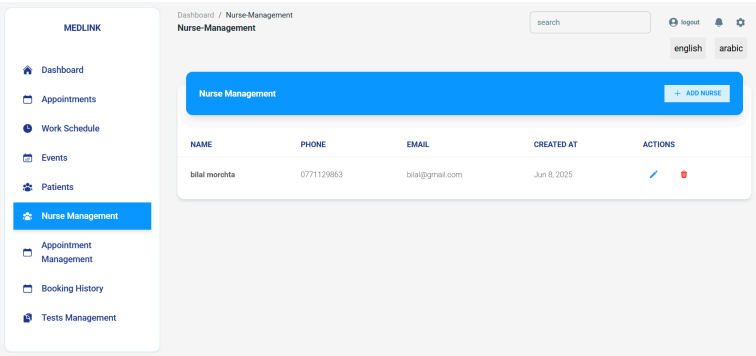

Figure 4.13: Doctor-Dashboard Nurse Management

Booking History Page

The Booking History page displays all appointments that have been previewed or deleted, with the ability to view the details of each appointment, including reports, medications, or tests that the doctor provided to the patient.

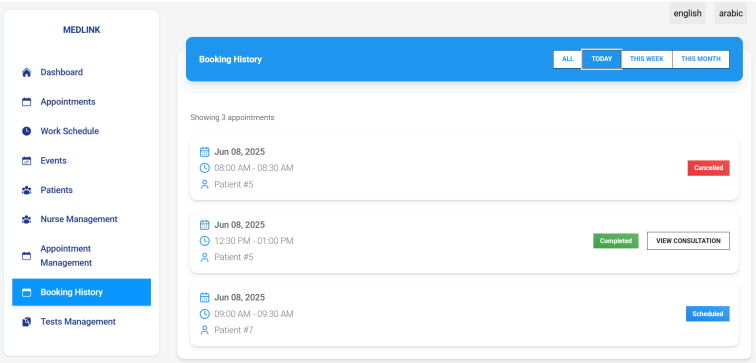

Figure 4.14: Doctor-Dashboard Booking History

Tests Management

The Tests Management page displays all the tests that the doctor has added, with the ability to modify the test values. This page is also available for the nurse to view and help manage

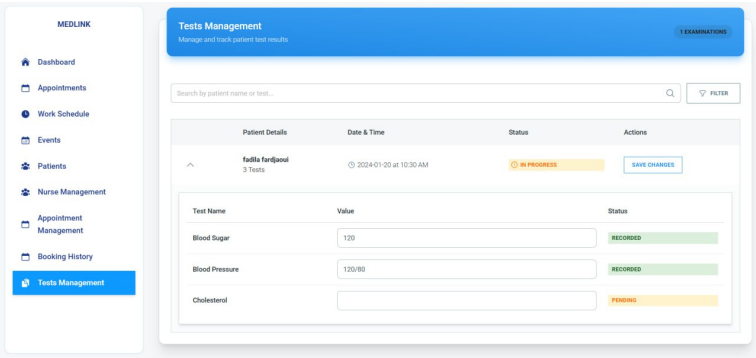

Figure 4.15: Tests Management

2.4 Patient registration Page

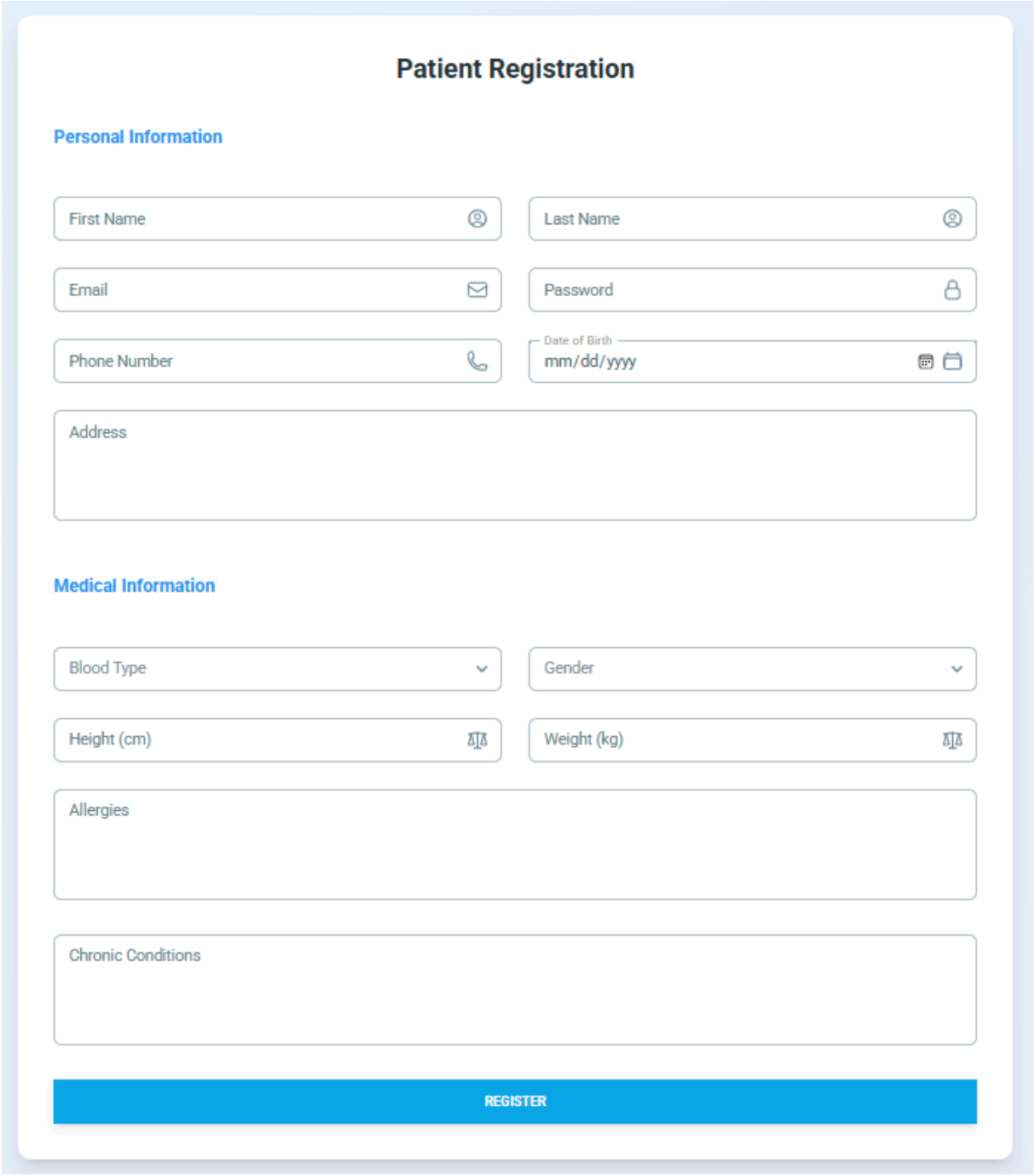

Figure 4.16: Patient registration Page

2.5 Patient Dashboard

The patient's Dashboard page displays his complete profile, including his personal information, medical history, all medical reports, prescriptions, tests, and previous and upcoming appointments

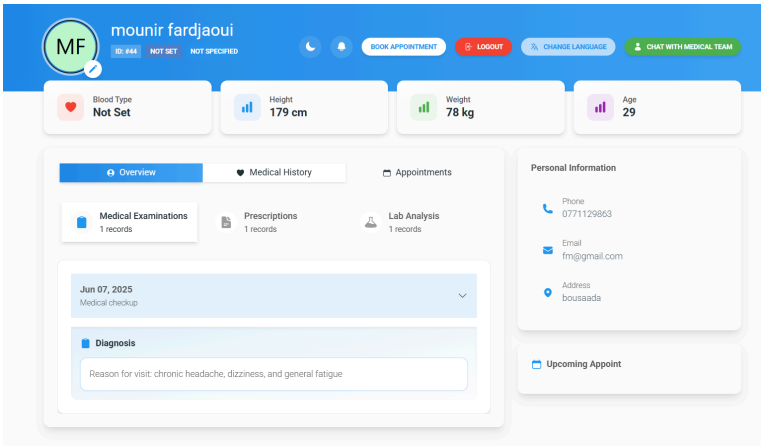

Figure 4.17: Patient-Dashboard Medical Examinations

2.6 Find Doctor Page

The Find a Doctor page allows the patient to search for a doctor by name, specialty, or location, and also displays a list of all doctors available in the system.

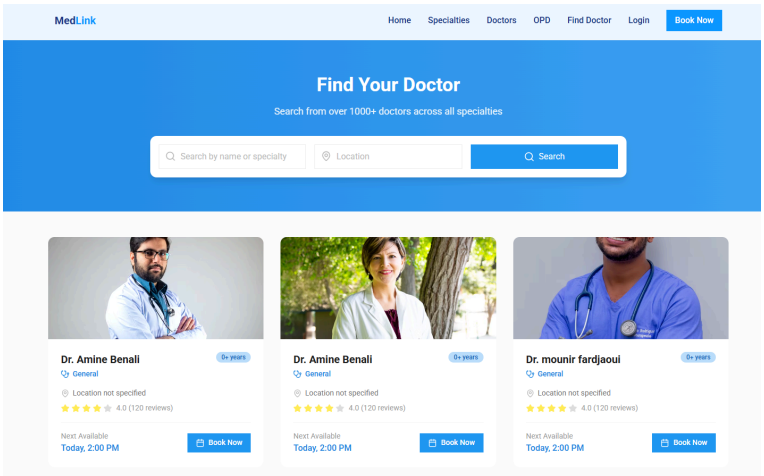

Figure 4.18: Find Doctor Page

2.7 Appointment booking page

The appointment booking page displays a doctor's identification card, including his basic information, displaying the days available for booking, and each day contains the time periods (slots) available for choosing the appropriate appointment.

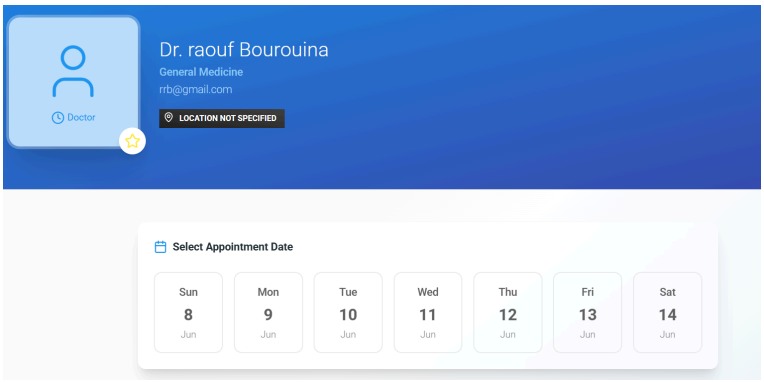

Figure 4.19: booking page

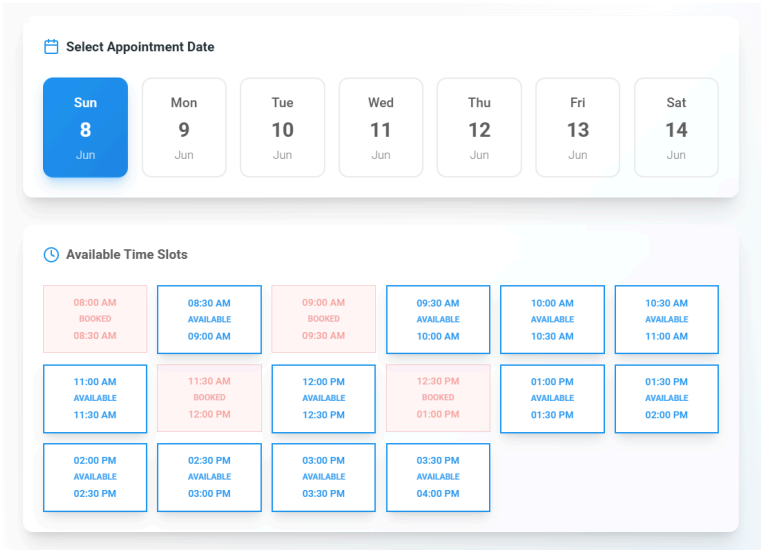

Figure 4.20: booking page (slots)

2.8 Pharmacist Dashboard

The pharmacist's dashboard page contains a QR Code scanner to display the prescription sent by the doctor, with the ability to confirm or reject the dispensing of medications, depending on the case.

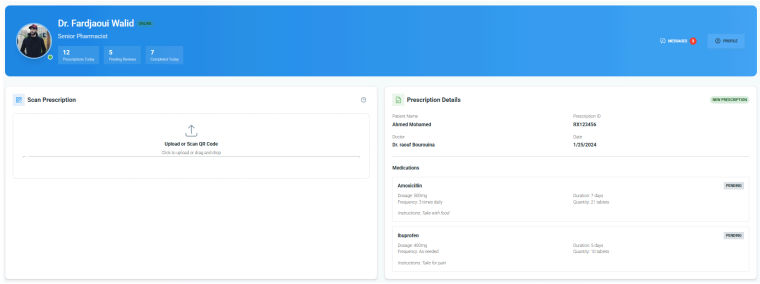

Figure 4.21: Pharmacist -Dashboard

2.9 Chat page

The chat page is available to the doctor, pharmacist, and patient, and allows real-time conversation to facilitate quick and effective communication between them.

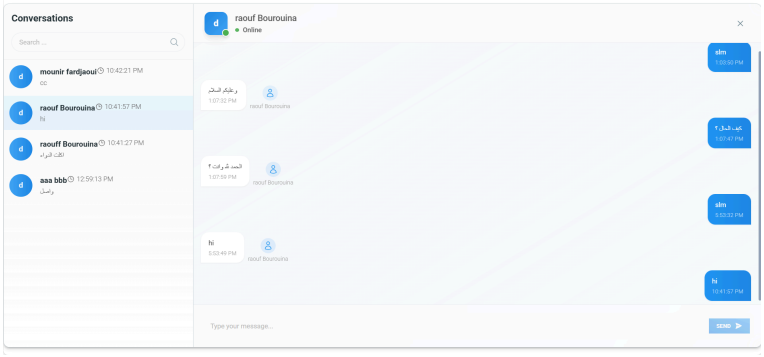

Figure 4.22: chat page

2.10 Chatbot page

A Chatbot page containing a smart medical assistant that users can interact with to get instant answers and initial medical advice

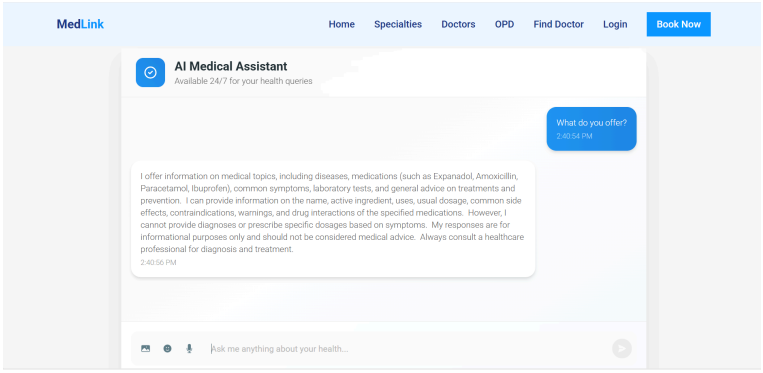

Figure 4.23: chatbot page

2.11 Medical History Assistant page

The Medical History Assistant page allows the user to ask questions in human slang, where the question is converted into an accurate medical query and the appropriate answer is retrieved from the user's medical record.

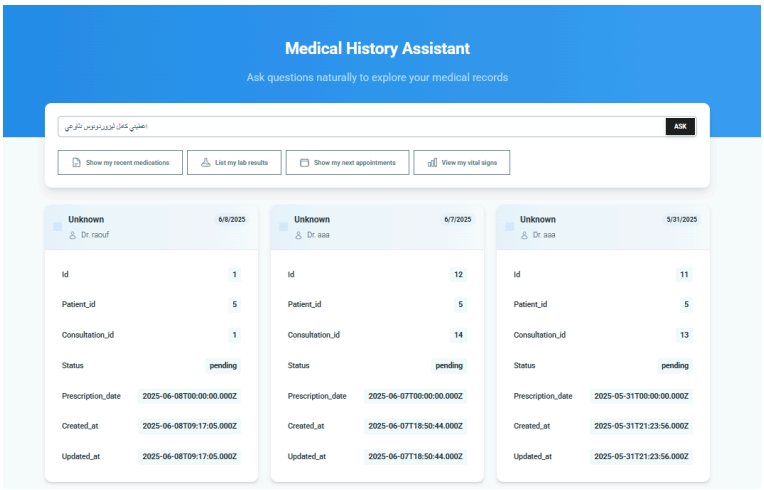

Figure 4.24: Medical History Assistant Page

3 Used Technologies

This section outlines the various modern web technologies and tools utilized in the development of our system. It covers the chosen technology stack, development environment, authentication mechanisms, and deployment solutions that contributed to building a secure, efficient, and scalable application.

3.1 Development Environment

The development environment employed in building the system integrates a set of essential tools that streamline and optimize the software development workflow:

- **Visual Studio Code (VS Code)**: A powerful, open-source code editor developed by Microsoft. It offers an integrated development experience with features such as syntax highlighting, code auto-completion, real-time error detection, formatting tools, and seamless version control integration.
- **Google Chrome**: A widely-used web browser developed by Google, recognized for its performance and intuitive interface. Chrome includes a set of built-in developer tools that allow real-time editing and debugging of web pages, facilitating rapid development and testing.
- **Postman**: A collaborative platform designed for API development. It simplifies the process of designing, testing, and documenting APIs, enabling efficient communication between frontend and backend components.

- **Git**: A distributed version control system that supports efficient management of code changes. Git is especially valuable for collaborative development, allowing multiple developers to track changes, revert to previous versions, and manage code branches effectively.
- **GitHub**: A cloud-based platform built on top of Git, offering source code hosting, collaboration features, issue tracking, task management, and continuous integration. It enhances team productivity and supports scalable, collaborative software development.

3.2 Development Stack

Back-end Stack

The back-end of our system was developed using the following technologies:

- **Node.js**: A JavaScript runtime environment on the server, used to run code outside the browser. It provides high performance when processing a large number of requests and serves as the foundation for the server application.[9]
- **Express.js**: A lightweight framework built on Node.js, used to facilitate the creation of servers, define routes, and handle requests and responses. It provides a simple and flexible architecture for developing REST APIs.**<empty citation>**
- **MySQL**: A relational database management system used to store data related to users, patients, appointments, and prescriptions.[6]
- **Sequelize**: An Object-Relational Mapping (ORM) tool that simplifies interactions with the MySQL database by allowing developers to use JavaScript instead of SQL.
- **JWT with Refresh Token**: JSON Web Tokens are used to secure authentication between the frontend and backend. Refresh Tokens support session renewal without requiring the user to log in again.[10]
- **Qrcode**: A library used to generate QR codes that contain prescription information, allowing doctors to send data to pharmacists quickly and securely.

Front-end Stack

The front-end of the system was implemented using the following technologies:

- **HTML**: A markup language used to structure the content of web pages, including forms, buttons, and text elements.
- **CSS**: A styling language used to define the visual presentation of web pages, including colors, fonts, spacing, and layout.
- **JavaScript**: The primary programming language used to add dynamic behavior to the interface, such as form validation and real-time server communication.

- **React.js**: A JavaScript library used to build component-based user interfaces. It allows for the modular and reusable development of web pages.[11]
- **TailwindCSS & Shadcn/ui**: TailwindCSS is a utility-first CSS framework that accelerates interface development with consistent styling. Shadcn/ui is a component library built on Tailwind, offering professional and customizable UI components.[12]
- **React Query**: A library used for fetching, caching, and updating server data in React applications. It ensures real-time data updates without the need for manual page reloads.[11]
- **i18next**: A powerful internationalization library that supports multilingualism within the application. It was used to provide dynamic translation for both Arabic and English interfaces.
- **Axios**: A promise-based HTTP client used to interact with the backend API, handling operations such as login, appointment scheduling, and data retrieval.
- **Vite**: A modern build tool that significantly improves development speed and efficiency through features such as fast module replacement and optimized performance.

4 Conclusion

Through this chapter, I provided a detailed explanation of the platform that was developed, reviewing the basic functions of each user, and the most important interfaces that were designed to meet the needs of each category. The software tools and techniques that were relied upon to build a modern, secure, multilingual web application were also explained. The choice of these technologies was not random, but rather came to ensure high performance, organization, and ease of expansion in the future

General Conclusion

At the conclusion of this work, I can say that this memorandum was a rich experience that allowed me to apply much of the theoretical knowledge I acquired during my university career in the field of automated media. Through this project, I designed and developed a digital medical platform that brings together the doctor, the patient, and the pharmacist in a unified space, with a focus on improving the quality of health services and facilitating communication processes between them.

Through this work, I have sought to provide a practical solution to a real problem that many health institutions suffer from, which is the absence of direct communication between the various actors in the medical process. To achieve this goal, I relied on modern technologies such as the Multi-Tenancy architecture and WebSocket technologies for real-time communication, in addition to taking advantage of artificial intelligence capabilities in some ancillary services.

On the other hand, I faced several challenges during the completion of the project, especially with regard to managing multiple databases and ensuring the security and protection of user data. However, these challenges contributed to developing my skills further and pushed me to research and continuous learning to reach effective solutions.

Finally, I would like to emphasize that this project represents only the first step in a long path of research and development, as there remain many aspects that can be improved and enriched in the future, whether in terms of functionality, integration with other health systems, or even expanding the use of artificial intelligence to support medical decisions.

I ask God for success and hope that this work will be a useful addition to the field of developing digital systems in the health sector.

References

[1] World Health Organization, *Global Diffusion of eHealth: Making Universal Health Coverage Achievable*. Geneva, Switzerland: World Health Organization, 2016.

[2] K. Häyrinen, K. Saranto, and P. Nykänen, ``Definition, structure, content, use and impacts of electronic health records: A review of the research literature," *International Journal of Medical Informatics*, vol. 77, no. 5, pp. 291–304, 2008.

[3] *Json web tokens introduction*, https://jwt.io/introduction/, 2023.

[4] G. A. Blog, *Gemini: A multimodal ai model for healthcare applications*, https://ai.google/research, 2023.

[5] M. Docs, *Introduction to multi-tenant architecture*, https://learn.microsoft.com/en-us/azure/architecture/multitenant/, 2023.

[6] *Mysql 8.0 reference manual*, https://dev.mysql.com/doc/, 2024.

[7] I. Fette and A. Melnikov, *The websocket protocol*, https://tools.ietf.org/html/rfc6455, RFC 6455, IETF, 2011.

[8] T. Axelrod and S. Kalenda, *Building Real-Time Applications with WebSockets*. O'Reilly Media, 2019.

[9] *Building scalable backend services with node.js*, https://nodejs.org/docs/latest/api/, 2024.

[10] A. M. Al-Aswad *et al.*, ``Electronic health record (ehr) system: A review of the literature," *Journal of Healthcare Information Management*, vol. 27, no. 2, pp. 1–10, 2013.

[11] *React – a javascript library for building user interfaces*, https://reactjs.org/docs/getting-started.html, 2024.

[12] *Tailwind css documentation*, https://tailwindcss.com/docs, 2024.

ملخص

شهد قطاع الرعاية الصحية في السنوات الأخيرة تحوّلًا جذريًا بفضل اعتماد الأنظمة الرقمية بدلاً من السجلات الورقية، مما ساهم في تحسين جودة الخدمات الصحية من خلال تمكين الأطباء من الوصول الفوري إلى الملفات الطبية للمرضى. كما أصبح بإمكان المرضى الاستفادة من خدمات إلكترونية متعددة، مثل حجز المواعيد والاطلاع على معلوماتهم الصحية بشكل آمن وسلس.

ومع هذا التطور المتسارع، ظهرت الحاجة إلى اعتماد هياكل برمجية أكثر مرونة وكفاءة لمعالجة التحديات المرتبطة بالأنظمة التقليدية. من بين هذه الهياكل تبرز بنية متعددة المستأجرين (Multi-Tenancy)، التي تُمكّن من تخصيص قاعدة بيانات مستقلة لكل طبيب، بما يضمن عزل بيانات كل طبيب ومرضاهم بشكل كامل، مع الحفاظ على خصوصية المعلومات الطبية وحمايتها، وهو ما يمثل قيمة مضافة للمطورين ومهندسي البرمجيات في بناء أنظمة مرنة وقابلة للتوسع.

تركز هذه المذكرة على تصميم وتنفيذ منصة إلكترونية لإدارة العيادات والخدمات الطبية باستخدام بنية متعددة المستأجرين بقاعدة بيانات منفصلة لكل طبيب. تم تطوير النظام باستخدام تقنيات حديثة تشمل React.js في الواجهة الأمامية وNode.js مع MySQL في الواجهة الخلفية. توفر المنصة ميزات متقدمة، مثل: إدارة المواعيد، إنشاء وصفات طبية عبر رموز QR، نظام دردشة بين الأطباء والصيادلة، إدارة التحاليل الطبية، وإمكانية إشراك الممرضين في سير العمل الطبي الخاص بكل طبيب.

يهدف هذا العمل إلى تقديم نظام رقمي متكامل يسهم في تحسين جودة الخدمات الصحية، وضمان حماية البيانات الطبية، وتسهيل عملية التفاعل بين مختلف الأطراف الفاعلة في المجال الطبي.

Abstract

In recent years, the healthcare sector has undergone a significant transformation by adopting digital systems instead of paper records. This has improved the quality of care by enabling doctors to access patients' medical files quickly. Patients can also benefit from online services such as booking appointments and securely viewing their health information. With this fast development, there was a need to use more flexible and efficient software architectures to address the challenges of traditional systems. Among these architectures, multi-tenancy allows for a separate database for each doctor, ensuring full isolation and protection of each doctor's data and their patients' data. This approach provides great value to developers in building flexible and scalable systems. This thesis focuses on designing and implementing an electronic platform for managing clinics and medical services using a multi-tenancy architecture with a separate database for each doctor. The system was developed using modern technologies such as React.js for the frontend, and Node.js with MySQL for the backend. The platform offers advanced features including appointment management, creating medical prescriptions with QR codes, real-time chat between doctors and pharmacists, medical test management, and involving nurses in the medical workflow. The goal is to deliver a complete digital system that improves healthcare quality, ensures data privacy, and facilitates interaction between different stakeholders in the medical field.

Résumé

Ces dernières années, le secteur de la santé a connu une transformation importante grâce à l'adoption des systèmes numériques au lieu des dossiers papier. Cela a amélioré la qualité des soins en permettant aux médecins d'accéder rapidement aux dossiers médicaux des patients. Les patients peuvent aussi profiter de services en ligne, comme la prise de rendez-vous et la

consultation sécurisée de leurs informations médicales. Avec cette évolution rapide, il est devenu nécessaire d'utiliser des structures logicielles plus flexibles et efficaces pour répondre aux défis des systèmes traditionnels. Parmi ces structures, la multi-location (Multi-Tenancy) permet d'attribuer une base de données séparée pour chaque médecin, ce qui assure l'isolement et la protection des données médicales de chaque médecin et ses patients. Cela offre une grande valeur aux développeurs pour créer des systèmes flexibles et évolutifs. Ce mémoire présente la conception et la réalisation d'une plateforme électronique pour gérer les cliniques et les services médicaux, en utilisant une architecture multi-location avec une base de données séparée pour chaque médecin. Le système est développé avec des technologies modernes comme React.js pour l'interface utilisateur, et Node.js avec MySQL pour le serveur. La plateforme propose plusieurs fonctionnalités avancées : gestion des rendez-vous, création d'ordonnances avec codes QR, chat en temps réel entre médecins et pharmaciens, gestion des analyses médicales, et intégration des infirmiers dans le workflow médical. L'objectif est de fournir un système numérique complet qui améliore la qualité des soins, protège la confidentialité des données médicales et facilite la communication entre les différents acteurs du secteur de la santé.