

**Faculty of Artificial Intelligence  
Department of Software Engineering  
and Information Systems**



## **Voice Based System For Medical Clinics**

Senior Project- Completed the requirements for obtaining  
a bachelor's degree in Informatics Engineering - Software  
Engineering and Information Systems

### **Prepared By**

Thamer AlSaiad

Ahmad AlGhothani

### **Supervised By**

Eng. Anas Abdulaziz

# Voice Based System For Medical Clinics

مشروع تخرج - قدم لاستكمال متطلبات الحصول على درجة البكالوريوس  
في هندسة المعلوماتية

هندسة البرمجيات ونظم المعلومات

إعداد

أحمد أيمن الغوثاني

ثامر محمد سهيل الصياد

إشراف

المهندس أنس عبد العزيز

# ABSTRACT

This project presents a comprehensive clinical management system designed to revolutionize healthcare documentation and streamline clinical workflows through artificial intelligence. Traditional clinical documentation methods are time-consuming, prone to errors, and detract from patient-clinician interaction time. To address these challenges, the system integrates voice-to-text transcription and automated clinical data handling.

The platform enables clinicians to register securely, manage patient records, and conduct patient visits with real-time voice recording and transcription capabilities. It supports structured data entry and organization to reduce documentation time while improving accuracy.

Core features include comprehensive patient and visit management, appointment scheduling for both clinicians and patients, and medical terminology recognition during transcription. The system also provides a patient portal where individuals can book appointments, access their medical records, and export their health data in standard formats. Administrative capabilities allow healthcare organizations to manage clinician accounts, generate clinical reports, and maintain system oversight.

By combining speech recognition technology with medical AI, the platform transforms the clinical documentation process, allowing healthcare providers to focus more on patient care while ensuring accurate, comprehensive, and timely medical records. The system supports the complete patient care cycle from registration and appointment booking through visit documentation and long-term record management.

## **الملخص**

يقدم هذا المشروع نظاماً شاملاً لإدارة العيادات، مصمماً لإحداث ثورة في توثيق الرعاية الصحية وتبسيط سير العمل السريري من خلال الذكاء الاصطناعي. تعتبر طرق التوثيق السريري التقليدية مستهلكة للوقت وعرضة للأخطاء، كما أنها تتطلب من وقت التفاعل بين المريض والطبيب. ومعالجة هذه التحديات، يدمج النظام تقنية النسخ الصوتي إلى نص ومعالجة البيانات السريرية بشكل آلي.

تمكّن المنصة الأطباء من التسجيل بشكل آمن، وإدارة سجلات المرضى، وإجراء الزيارات الطبية مع إمكانيات التسجيل الصوتي والنسخ في الوقت الفعلى. يدعم النظام تنظيم البيانات السريرية بطريقة مبسطة لقليل وقت التوثيق وتحسين الدقة.

تشمل الميزات الأساسية الإدارية الشاملة للمرضى والزيارات، وجدولة المواعيد لكل من الأطباء والمرضى، والتعرف على المصطلحات الطبية أثناء النسخ، والتوليد الآلي للوثائق السريرية. يوفر النظام أيضاً بوابة للمرضى حيث يمكنهم حجز مواعيد، والوصول إلى سجلاتهم الطبية، وتصدير بياناتهم الصحية بصيغة قياسية. تتيح القدرات الإدارية للمؤسسات الصحية إدارة حسابات الأطباء، وإنشاء التقارير السريرية، والحفظ على الإشراف على النظام.

من خلال الجمع بين تقنية التعرف على الكلام والذكاء الاصطناعي الطبي، تحوّل المنصة عملية التوثيق السريري، مما يسمح لمقدمي الرعاية الصحية بالتركيز أكثر على رعاية المرضى مع ضمان سجلات طيبة دقيقة وشاملة وفي الوقت المناسب. يدعم النظام دورة رعاية المريض الكاملة من التسجيل وحجز المواعيد حتى توثيق الزيارات وإدارة السجلات طويلة الأجل.

# **SUPERVISION CERTIFICATION**

I certify that the preparation of this project entitle  
**"Voice Based System For Medical Clinics"**

Prepared by **Thamer Al-Saiad and Ahmad Al-Ghothani**, was  
made under my supervision at the Faculty of Informatics  
Engineering in partial fulfillment of the requirements for the  
Degree of Bachelor of Software Engineering and Information  
Systems.

**Name:**.....

**Signature:**.....

<b>ABSTRACT .....</b>	3
الملخص .....	4
<b>SUPERVISION CERTIFICATION .....</b>	5
<b>CHAPTER 1 - INTRODUCTION .....</b>	9
1.1 Introduction .....	10
1.2 Problem Statement.....	10
1.4 Project Objectives .....	12
1.5 Report Organization .....	13
1.6 Summary.....	13
<b>Chapter 2 - Fundamental Concepts and Literature Review .....</b>	15
2.1 Introduction .....	15
2.2 Fundamental Concepts .....	16
2.2.1 Automatic Speech Recognition (ASR) in Healthcare.....	16
2.2.2 Voice-Enabled Healthcare Interaction.....	16
2.2.3 Speech-to-Text as a Clinical Tool .....	16
2.2.4 Role-Based Access and Functionality.....	17
2.2.5 Integrated Healthcare Management.....	17
2.2.6 Real-Time Processing and Web-Based Architecture .....	17
2.3 Literature Review: Comparative Analysis of Existing Platforms .....	18
2.3.1 Nuance Dragon Medical One.....	18
2.3.2 Epic MyChart.....	18
2.3.3 Athenahealth .....	18
2.3.4 Practice Fusion.....	19
2.3.5 Kareo.....	19
2.4 Gap Analysis and Innovation.....	20
2.5 Comparative Table: Feature-by-Feature Analysis.....	21
2.6 Summary.....	22
<b>Chapter 3 - Project Management and Initialization .....</b>	23
3.1 Introduction .....	23
3.2 Project Management Documents .....	24
3.2.1 Project Charter .....	24
Project Objectives.....	24
Approach .....	24
Roles and Responsibilities.....	25
3.2.2 Statement Of Work (SOW) .....	25
1. Project Title.....	26
2. Introduction.....	26
3. Purpose .....	26
4. Scope .....	26
5. Project Goals .....	26
6. Deliverables.....	27

7. Technical Requirements .....	27
8. Assumptions .....	27
9. Project Resources .....	28
10. Project Approach .....	28
11. Schedule.....	28
3.2.3 Risk Management.....	28
3.2.4 Gantt Chart.....	31
3.3 Initial System Study.....	31
3.3.1 Introduction.....	31
3.3.2 High-Level Analysis .....	31
3.3.2.1 Actors and Interactions Overview.....	32
3.3.2.2 Use Case Diagram.....	33
3.3.3 Development Process.....	34
3.3.3.1 Sprint 0 (Preplanning and Initialization) .....	34
3.3.3.2 Sprint 1 (Account Management & Authentication) .....	35
3.3.3.3 Sprint 2 (Patient Visit Management & Medical Records) .....	35
3.3.3.4 Sprint 3 (Speech-to-Text Integration) .....	36
3.3.3.5 Sprint 4 (Appointment Management & Patient Portal) .....	36
3.3.3.6 Sprint 5 (Administration Module & Final Integration) .....	37
3.3.3.7 Sprint 6 (Final Testing & Documentation Review) .....	37
3.3.4 High-Level System Design.....	38
3.3.4.1 High-Level System Architecture .....	38
3.3.4.2 High-Level System Decomposition .....	38
3.3.5 Progress Monitoring and Report .....	39
3.3.6 Summary .....	39
3.4 Initial RTM (Requirements Traceability Matrix) .....	39
3.5 Initial Test Cases.....	50
<b>3.6 Project Management Using Jira (Scrumban Methodology).....</b>	71
3.6.1 Scrumban Methodology Overview .....	72
3.6.2 Jira Board Structure and Workflow .....	72
3.6.3 Jira-Based Sprint Planning and Monitoring.....	73
3.6.4 Jira Board Figure Description .....	74
<b>3.7 Version Control and Configuration Management Using Git and GitHub.....</b>	75
3.7.1 Git-Based Version Control Strategy .....	75
3.7.2 Branching Model .....	75
3.7.3 GitHub Workflow and Collaboration.....	76
3.7.4 Commit Conventions and Traceability .....	76
3.7.5 Quality Assurance and Deployment Control .....	77
<b>Chapter 4 - System Analysis .....</b>	78
4.1 Introduction .....	79
4.2 Purpose.....	79

4.3 Project Scope.....	79
4.4 Requirements Elicitation.....	79
4.5 Requirements Table .....	80
4.6 Analysis.....	87
4.6.1 Requirements Modeling.....	88
4.7 RTM V.2.....	294
Table 73 - RTM V.2.....	306
<b>Chapter 5 - System Design.....</b>	<b>307</b>
5.1 Introduction: .....	308
5.2 Architecture Layers .....	308
5.2.1 Client Side .....	308
5.2.2 Server-Side .....	308
5.2.3 Database .....	308
5.3 Sprint 1 Design.....	309
5.3.1 System Architecture & Component Diagram.....	309
5.3.2 Class Diagram .....	313
Figure 137, Sprint 1 Class Diagram.....	313
5.5 RTM V.3.....	316
Table 74- RTM V.3 .....	328
<b>Chapter 6 - Implementation and Testing .....</b>	<b>329</b>
6.1 Introduction .....	330
6.2 Technologies Used.....	330
6.3 System Interfaces.....	336
6.4 Test Cases.....	355
6.5 RTM V.4.....	464
Table 76- RTM V.4 .....	476
<b>Chapter 7 - Conclusion, future directions and references .....</b>	<b>477</b>
7.1 Introduction .....	478
7.2 Summary of Achievements.....	478
7.3 Addressing Key Challenges .....	478
7.4 System Reliability and Performance .....	479
7.5 Future Vision: Proposed Enhancements .....	479
7.6 Conclusion .....	481
7.7 References.....	481

## **CHAPTER 1 - INTRODUCTION**

## 1.1 Introduction

In this chapter, we introduce the Voice-Based Electronic Medical Records (EMR) System, a healthcare platform designed to improve the efficiency and accessibility of medical documentation and patient care management. We begin by presenting the problems that currently hinder effective healthcare record management and clinical workflows. Then, we outline the system's goals and objectives, and provide a high-level view of the platform. Finally, we present the structure of this report and how the rest of the document is organized.

## 1.2 Problem Statement

Electronic Medical Records and healthcare management systems are becoming essential components of modern healthcare delivery, yet the way medical information is documented and accessed often lacks efficiency and intuitiveness. One of the most significant gaps lies in the time-consuming nature of manual data entry and the complexity of traditional EMR interfaces, particularly in clinical settings where healthcare providers need to focus on patient care rather than navigating complex software systems.

Traditional healthcare management platforms rely on text-based input and complex navigation structures that require significant time and attention from medical professionals. As a result, many healthcare providers struggle to maintain comprehensive documentation while delivering quality patient care, leading to incomplete records or reduced patient interaction time. Additionally, many platforms don't include intuitive voice-based interaction capabilities, and patients often face barriers in accessing their own health information or managing appointments, which creates a gap between healthcare providers, patients, and efficient care delivery.

To address these challenges, there is a need for an interactive, voice-enabled, and user-friendly platform that supports both clinical documentation and patient engagement, with features that bridge the gap between healthcare providers and patients through natural language interaction.

### **1.3 Proposed System**

The proposed system is envisioned as a comprehensive healthcare management platform tailored to support clinical operations and patient engagement, with a focus on voice-based interaction for medical record management. It combines structured healthcare workflows with voice-enabled features to create an efficient and accessible healthcare experience.

Users will be able to access different functionalities based on their role in the system. Patients will have the ability to manage their accounts, view and track their medical visit history, schedule and manage appointments, and access a dedicated patient portal for viewing health information and communicating with their healthcare providers. Clinics and healthcare providers will be able to manage institutional accounts, document and track patient visits, utilize speech-to-text functionality for efficient clinical documentation, maintain comprehensive medical records, and manage appointment schedules across their practice. System administrators will have access to account management tools, medical record oversight capabilities, and administrative functions to ensure smooth system operations.

The system will support user registration and authentication for all user types, allowing patients, clinics, and administrators to securely manage their profiles and access role-appropriate features. Voice interaction capabilities will be integrated throughout the platform to streamline documentation and improve accessibility, while traditional input methods will remain available for user preference and flexibility.

## 1.4 Project Objectives

The primary objective of the Voice-Based EMR System is to develop a healthcare-focused platform that enhances clinical efficiency and patient engagement through voice-enabled documentation, structured workflows, and comprehensive record management capabilities.

The key goals include:

- Delivering role-based functionality that serves the distinct needs of patients, healthcare providers, and system administrators through intuitive interfaces tailored to each user type.
- Providing voice-enabled medical documentation that allows healthcare providers to efficiently record patient information using speech-to-text technology, reducing documentation time and improving patient interaction.
- Offering comprehensive patient visit management and medical record tracking to ensure continuity of care and easy access to patient health history.
- Implementing robust appointment management that enables both patients and clinics to schedule, modify, and track appointments efficiently.
- Creating a patient portal that empowers patients to access their health information, view visit summaries, and engage with their healthcare providers.
- Implementing a secure system for user registration, authentication, and role-based access control to protect sensitive medical information.
- Ensuring an intuitive and accessible interface that supports both voice and traditional interaction methods across desktop and mobile devices, making healthcare management accessible to users with varying technical abilities and accessibility needs.

## 1.5 Report Organization

The remainder of the report is organized into the following chapters:

Chapter 1: Introduction

Chapter 2: Basic Concepts and Literature Review

Chapter 3: Project Management and Initial Study

Chapter 4: System Analysis

Chapter 5: System Design

Chapter 6: Practical Implementation

Chapter 7: System Testing

Chapter 8: Conclusion

## 1.6 Summary

This chapter has introduced the motivation and vision behind the Voice-Based System for Medical Clinics. It identified the current issues within clinical documentation workflows, particularly the reliance on manual data entry, fragmented patient information, and reduced clinician efficiency. The proposed solution leverages voice-enabled automation and intelligent EMR processing to simplify documentation and improve overall accuracy. The next chapters will expand on the technical and operational aspects of the system, moving toward the development of a streamlined, reliable, and user-centered clinical platform.



## **Chapter 2 - Fundamental Concepts and Literature Review**

### **2.1 Introduction**

In the realm of modern healthcare and medical information technology, there exists an ever-increasing need for platforms that not only manage medical records but do so through intuitive and accessible interfaces. The Voice-Based Electronic Medical Records System stands at the intersection of healthcare delivery and technology, offering a system that streamlines clinical documentation and patient engagement through voice-enabled interaction and comprehensive management features. This chapter explores the fundamental concepts that drive this project, alongside a rigorous literature review comparing similar platforms. It aims to identify both technical and operational gaps that this system addresses.

## 2.2 Fundamental Concepts

### 2.2.1 Automatic Speech Recognition (ASR) in Healthcare

Automatic Speech Recognition converts spoken language into written text using acoustic and language models. In healthcare settings, ASR systems must handle:

- **Medical terminology and jargon:** Accurate recognition of complex medical terms, drug names, and anatomical references
- **Varied accents and speaking speeds:** Adaptation to diverse healthcare providers with different speaking patterns
- **Background noise from clinical environments:** Robust performance in busy clinical settings with ambient noise
- **Real-time processing requirements:** Immediate transcription to support clinical workflows without delays

Modern ASR systems leverage deep learning architectures, including transformer-based models, to achieve high accuracy in medical contexts. These systems enable healthcare providers to document patient information efficiently without interrupting the natural flow of clinical encounters.

### 2.2.2 Voice-Enabled Healthcare Interaction

Modern healthcare delivery emphasizes the importance of efficiency and accessibility. The Voice-Based EMR System centers its functionality around speech-to-text technology that allows healthcare providers to document patient information naturally while maintaining focus on patient care. Rather than navigating complex menu systems or typing lengthy notes, clinicians engage with the system through an intuitive voice interface that enhances productivity by allowing hands-free documentation and faster data entry.

### 2.2.3 Speech-to-Text as a Clinical Tool

Voice-enabled input has been shown to improve documentation efficiency and reduce clinician burnout. By enabling natural spoken documentation that is automatically converted to text, the

system creates a streamlined workflow that reduces the time spent on administrative tasks and enhances the quality of patient interactions. Healthcare providers can focus on patient care while the system captures information accurately and efficiently.

#### **2.2.4 Role-Based Access and Functionality**

The platform introduces three distinct user roles with appropriate access levels:

- **Patient Role:** Provides access to personal health information, appointment scheduling, visit history tracking, and a dedicated patient portal for healthcare engagement
- **Clinic/Provider Role:** Offers comprehensive tools for patient visit management, medical record maintenance, speech-to-text documentation, and appointment scheduling across the practice
- **Administrator Role:** Enables system-wide account management, medical record oversight, and administrative functions to ensure secure and efficient operations

Each role contains appropriate permissions, security measures, and interface customization, encouraging efficient workflows while maintaining data security and privacy compliance.

#### **2.2.5 Integrated Healthcare Management**

To support comprehensive clinical operations, the system features integrated management tools that healthcare providers and patients can utilize. Here, clinics can manage patient visits, maintain medical records, and coordinate appointments, while patients can access their health information and communicate with providers. This approach mirrors modern healthcare delivery models, promoting better coordination and patient engagement.

#### **2.2.6 Real-Time Processing and Web-Based Architecture**

Modern web-based clinical systems utilize:

- **WebRTC (Web Real-Time Communication):** Enables peer-to-peer audio streaming with low latency for voice input
- **WebSocket Streaming:** Provides bi-directional, real-time data transmission between browser and backend for immediate feedback
- **Incremental Transcription:** Delivers partial text conversion during speech, allowing clinicians to monitor accuracy in real-time
- **Responsive Web Design:** Ensures accessibility across desktop and mobile devices without software installation requirements

This architecture ensures minimal delay between speech and transcription, critical for clinical workflow efficiency, while maintaining the flexibility and accessibility of a web-based platform.

## 2.3 Literature Review: Comparative Analysis of Existing Platforms

Several platforms support healthcare documentation and management through varying degrees of voice integration, user accessibility, and comprehensive features. Below is a deeper comparative analysis including strengths, weaknesses, similarities, and how our system addresses the limitations:

### 2.3.1 Nuance Dragon Medical One

**Strengths:** Cloud-based speech recognition platform with high accuracy (99%+) for medical terminology; deep integration with major EMR systems (Epic, Cerner, AllScripts); extensive medical vocabulary coverage; voice commands for navigation.

**Weaknesses:** Limited to transcription functionality without comprehensive patient management; no patient-facing portal or appointment scheduling; requires integration with separate EMR systems; primarily clinician-focused without patient engagement features.

**Similarities:** Provides robust speech-to-text capabilities for clinical documentation, demonstrating the viability of voice-enabled medical documentation.

**Our Contribution:** We expand beyond pure transcription to provide a complete healthcare management ecosystem that includes patient engagement, appointment scheduling, and integrated medical record management, all accessible through both voice and traditional interfaces.

### 2.3.2 Epic MyChart

**Strengths:** Comprehensive patient portal with appointment scheduling; secure messaging with providers; access to medical records and test results; integration with Epic EMR system; mobile application support.

**Weaknesses:** No voice-enabled documentation for clinicians; requires Epic EMR implementation (expensive and complex); limited customization for smaller practices; no integrated speech-to-text functionality.

**Similarities:** Effective for patient engagement and appointment management, demonstrating the value of patient-facing healthcare portals.

**Our Contribution:** We combine patient portal functionality with voice-enabled clinical documentation in a unified platform that doesn't require expensive EMR system implementation, making comprehensive healthcare management accessible to practices of all sizes.

### 2.3.3 Athenahealth

**Strengths:** Cloud-based practice management with scheduling, billing, and patient records; patient portal with appointment booking; mobile accessibility; comprehensive administrative tools.

**Weaknesses:** No voice-enabled documentation features; complex interface requiring significant training; high cost barrier for small practices; limited speech-to-text integration.

**Similarities:** Provides integrated practice management demonstrating the importance of unified healthcare platforms.

**Our Contribution:** We introduce voice-enabled documentation as a core feature while maintaining comprehensive practice management capabilities, with an intuitive interface designed for immediate adoption without extensive training requirements.

#### 2.3.4 Practice Fusion

**Strengths:** Free cloud-based EMR system; appointment scheduling; patient portal; prescription management; customizable templates for documentation.

**Weaknesses:** Ad-supported model raises privacy concerns; no voice recognition capabilities; limited advanced features; basic interface without modern UX design.

**Similarities:** Demonstrates the need for accessible EMR solutions for smaller practices but lacks modern interaction methods.

**Our Contribution:** We provide a modern, voice-enabled platform without compromising privacy through ad-based models, offering intuitive interfaces and advanced features accessible to practices of all sizes.

#### 2.3.5 Kareo

**Strengths:** Practice management and EMR combined; patient scheduling and portal; billing integration; mobile apps for providers; designed for small practices.

**Weaknesses:** No voice-enabled documentation; requires manual text entry for all documentation; limited real-time collaboration features; separate modules can feel disconnected.

**Similarities:** Targets the right audience (small to medium practices) but lacks modern input methods that improve efficiency.

**Our Contribution:** We adapt comprehensive practice management to include voice-enabled documentation, creating a unified experience that reduces documentation time while maintaining all essential practice management features in an integrated platform.

## 2.4 Gap Analysis and Innovation

The Voice-Based EMR System fills several identified voids in current healthcare management technologies:

- **Unified Management and Documentation:** Most platforms segregate clinical documentation from practice management and patient engagement. Our system integrates all three in one seamless user experience.
- **Voice-Enabled Accessibility:** Existing comprehensive EMR systems lack integrated speech-to-text functionality, while voice-focused tools lack complete practice management features.
- **Lowered Entry Barrier:** No expensive EMR implementation or complex software installation required, broadening access to practices of all sizes and technical capabilities.
- **Patient Engagement + Clinical Efficiency Fusion:** Our system integrates patient-facing features with clinician-focused documentation tools and administrative capabilities, bridging the gap between all stakeholders in the healthcare delivery process.
- **Multilingual Support:** Unlike most existing platforms that support only English, our system provides multilingual capabilities to serve diverse patient populations and international healthcare settings.

## 2.5 Comparative Table: Feature-by-Feature Analysis

Feature / Platform	Dragon Medical One	Epic MyChart	Athenahealth	Practice Fusion	Kareo	Voice-Based EMR System
<b>Voice-Enabled Documentation</b>	✓	✗	✗	✗	✗	✓
<b>Real-Time Speech-to-Text</b>	✓	✗	✗	✗	✗	✓
<b>Patient Portal Access</b>	✗	✓	✓	✓	✓	✓
<b>Appointment Scheduling</b>	✗	✓	✓	✓	✓	✓
<b>Medical Record Management</b>	✗	✓	✓	✓	✓	✓
<b>Patient Visit Tracking</b>	✗	✓	✓	✓	✓	✓

Table 1: Similar Systems Analysis

## 2.6 Summary

The Voice-Based EMR System is more than just a documentation tool; it is a complete healthcare management environment built around accessibility, efficiency, and comprehensive functionality. Unlike existing platforms that separate clinical documentation from practice management or cater only to large healthcare systems with significant resources, our system unifies voice-enabled documentation, patient engagement, and administrative capabilities into a singular healthcare ecosystem. With integrated speech-to-text functionality, comprehensive patient and visit management, intuitive role-based interfaces, and multilingual support, it bridges the gap between clinical efficiency and patient accessibility for modern healthcare practices. This chapter lays the groundwork for how these concepts translate into the system's architecture and features in the following chapters.

# **Chapter 3 - Project Management and Initialization**

## **3.1 Introduction**

In this chapter, we will explore the project management phase, a crucial element in ensuring the project's success. We will focus on scope determination, and setting the right goals for the project and requirements, also we will focus on breaking down the project into small tasks and estimating the time and effort needed for the project.

## **3.2 Project Management Documents**

### **3.2.1 Project Charter**

**Project Title:** Voice-Based System For Medical Records Management

**Project Start Date:** Nov 1st, 2025

**Project Finish Date:** Jan 21st, 2026

**Project Manager:** Eng. Anas Abdulaziz

### **Project Objectives**

The objective of the Voice-Based EMR System is to develop a healthcare management web platform providing users with role-based access focused on:

- Medical record management and documentation
- Voice-enabled clinical documentation (Speech-to-Text)
- Patient visit tracking and management
- Appointment scheduling and management

The platform will deliver a comprehensive healthcare experience by combining:

- Patient portal for health information access
- Speech-to-text functionality for efficient clinical documentation
- Comprehensive medical record management
- Appointment scheduling system
- Patient visit tracking and history
- Secure user authentication and authorization

### **Approach**

- Define the project's scope and objectives
- Break the project down into well-defined sprints
- Use Scrum methodology to ensure iterative delivery and feedback
- Maintain thorough documentation at each stage of the project's lifecycle

### Roles and Responsibilities

Name	Role	Responsibility
Eng. Anas Abdulaziz	Project Manager and Supervisor	Project management and monitoring
Ahmad Algothani	Software Engineer	Frontend development, Project documentation, analysis and design phase team leader, Configuration manager
Thamer AlSaiad	Software Engineer	Backend development, Project documentation, analysis, design and testing phase leader

**Table 2: Roles and Responsibilities**

#### 3.2.2 Statement Of Work (SOW)

## **1. Project Title**

Voice-Based System For Medical Records Management

## **2. Introduction**

This Statement of Work (SOW) outlines the objectives, scope, deliverables, requirements, assumptions, resources, and schedule for the development of a voice-enabled healthcare management platform aimed at enhancing clinical efficiency and patient engagement through integrated medical record management, speech-to-text documentation, and comprehensive appointment scheduling.

## **3. Purpose**

Our purpose is to design and implement a comprehensive healthcare platform that enables patients to access their medical information and schedule appointments, clinics to efficiently document patient visits using voice input and manage medical records, and administrators to oversee system operations, through role-based interfaces, voice-enabled documentation, and integrated management tools.

## **4. Scope**

The Voice-Based EMR System aims to deliver comprehensive healthcare management functionality with a particular focus on voice-enabled clinical documentation using speech-to-text technology. The platform includes role-based access for patients, clinics, and administrators, appointment scheduling, patient visit management, medical record tracking, and a patient portal. The specification covers high-level requirements and outlines the actors within the system.

## **5. Project Goals**

- **G1:** Empower patients through a dedicated portal for accessing health information
- **G2:** Enable efficient clinical documentation through integrated speech-to-text functionality
- **G3:** Facilitate comprehensive patient visit tracking and medical record management
- **G4:** Support appointment scheduling and management for both patients and clinics
- **G5:** Provide role-based healthcare management for patients, clinics, and administrators
- **G6:** Ensure accessibility and usability with intuitive UI/UX and mobile responsiveness

## 6. Deliverables

- **D1:** Project Plan
- **D2:** SRS Document
- **D3:** Software Design Document
- **D4:** Final Project Report
- **D5:** Functional Web Application

## 7. Technical Requirements

### Technical Stack:

- Front-End: ReactJs
- Backend: NestJs
- Database: PostgreSQL
- Server: NodeJs

## 8. Assumptions

- Availability of project team members and supervisors
- Modern user interface with good design appropriate for healthcare settings
- The system's features should be aligned with healthcare industry standards and privacy regulations
- Access to reliable speech-to-text service

- Users have access to devices with microphone capabilities for voice input

## 9. Project Resources

- Eng. Anas Abdulaziz – Project Manager & Supervisor
- Ahmad Alghothani – Software Engineer
- Thamer AlSaiad – Software Engineer

## 10. Project Approach

- Define the project's scope and objectives
- Break the project down into well-defined sprints
- Use Scrum methodology to ensure iterative delivery and feedback
- Maintain thorough documentation at each stage of the project's lifecycle

## 11. Schedule

**Project Start:** Nov 1, 2025

**Project Finish:** Jan 21, 2026

**First Seminar:** Nov 15, 2025

**Second Seminar:** Nov 27, 2025

**Final Seminar:** Jan 31, 2026

**Technical Interview:** Jan 10, 2026

### 3.2.3 Risk Management

Risk ID	Risk Title	Description	State	Impact	Mitigation Plan	Likelihood	Reported Date	Tracking Frequency
VEMR_R_SK_1024	Speech Recognition Accuracy	Speech-to-text functionality may have accuracy issues with medical terminology or accents	Open	High	Implement medical vocabulary enhancement, provide editing interface for corrections, conduct extensive testing with medical terminology	Medium	26/3/2025	Weekly
VEMR_R_SK_1025	Data Privacy Compliance	Healthcare data requires strict compliance with privacy regulations (HIPAA-like standards)	Open	Critical	Implement encryption, secure authentication, role-based access control, conduct security audits	High	26/3/2025	Weekly
VEMR_R_SK_1026	Time Constraints	The timeline of the project is really short and compact, any issue will lead to a delay	Open	High	Prioritizing key features with strict timeline, focusing on core	Medium	6/4/2025	Weekly

					functionalit y first			
VEMR_R SK_1027	Integration Complexity	Integrating multiple modules (voice, appointments, records) may create technical challenges	Open	Medium	Modular architecture design, clear API definitions, incremental integration testing	Medium	29/3/2025	Weekly
VEMR_R SK_1028	User Adoption	Healthcare providers may resist adopting voice-based documentation	Open	Medium	Provide traditional input alternatives, user training documentation, intuitive interface design	Low	6/4/2025	Bi-weekly

**Table 3: Risk Management**

### 3.2.4 Gantt Chart

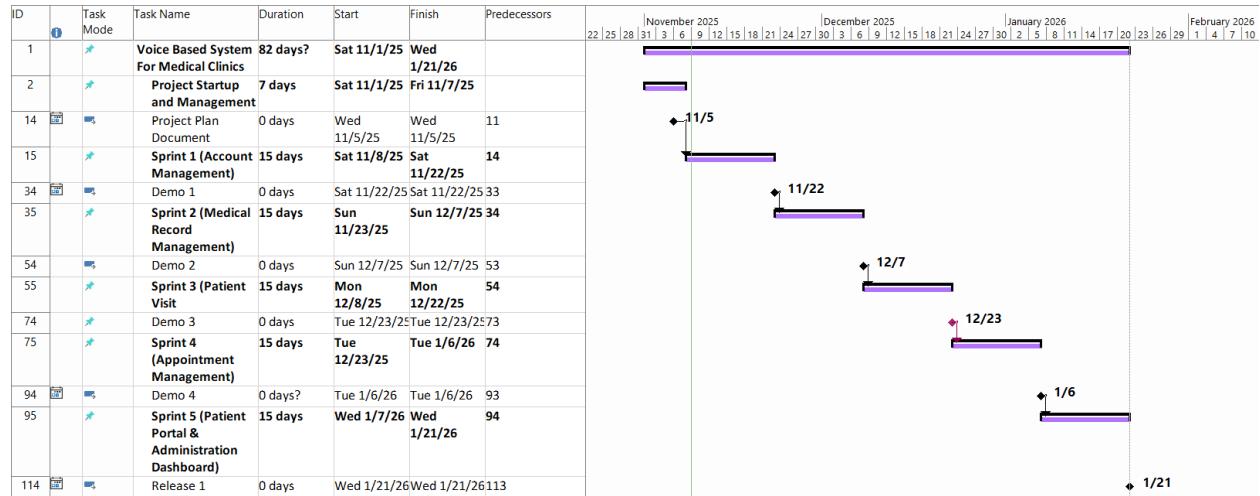


Figure 1: Gantt Chart

## 3.3 Initial System Study

### 3.3.1 Introduction

This chapter presents the initial analysis and system design of the Voice-Based EMR System, focusing on how the system architecture and functional specifications align with healthcare needs, user roles, and technical efficiency. This chapter provides high-level functional requirements, system decomposition, and the use of Unified Modeling Language (UML) diagrams to document early design decisions.

### 3.3.2 High-Level Analysis

The Voice-Based EMR System is designed as an integrated healthcare management platform with a focus on voice-enabled clinical documentation. This way healthcare providers can document patient information efficiently while maintaining focus on patient care, patients can easily access their health

information and manage appointments, and administrators can oversee system operations effectively.

### 3.3.2.1 Actors and Interactions Overview

This section provides an overview of the system's actors and their core interactions before introducing the formal use case diagram. It defines each actor's role in the healthcare system and usage, offering contextual grounding for the UML representation.

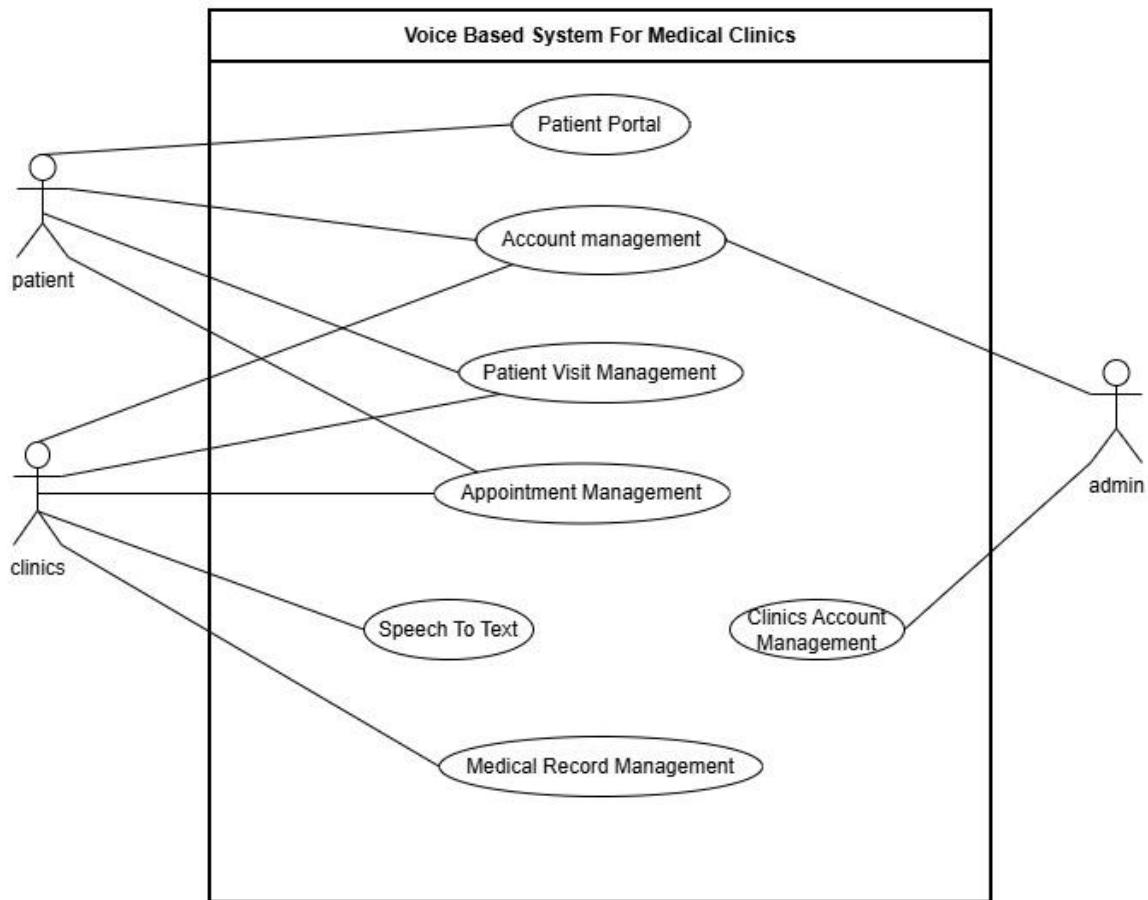
<b>Actor Type</b>	<b>Actor</b>	<b>Description</b>	<b>Goals</b>
Primary Actor	Patient	Registered patient user	Access medical records, schedule appointments, view visit history, manage profile
Primary Actor	Clinic	Healthcare provider or clinic staff	Document patient visits, manage medical records, use speech-to-text, schedule appointments, track patient history
Primary Actor	Admin	System administrator	Manage user accounts, oversee medical records, perform administrative functions, ensure system security
Supporting Actor	Guest	Unregistered user	Explore the website and register
External System	Email Service	Email service provider (NodeMailer)	Send verification, password reset, and appointment confirmation emails

External System	Speech-to-Text Service	OpenAI Whisper	Convert voice input to text for clinical documentation
-----------------	------------------------	----------------	--

**Table 4: System Actors**

### 3.3.2.2 Use Case Diagram

Use case diagrams model the behavior of a system and help to capture the interactions between system actors and the requirements.



**Figure 2: High-Level Use Case Diagram**

### 3.3.3 Development Process

#### 3.3.3.1 Sprint 0 (Preplanning and Initialization)

**Objective:** Conduct high-level system analysis, gather healthcare requirements, select technologies, and estimate timelines.

**Activities:**

- Study project scope and purpose
- Define high-level system requirements
- Select development tools and frameworks
- Create a project timeline
- Study healthcare data privacy requirements
- Research speech-to-text integration options

**Deliverables:**

- Project scope
- Technical scope
- High-level system requirements
- High-level system architecture
- Project management documents

### **3.3.3.2 Sprint 1 (Account Management & Authentication)**

**Objective:** Implement secure user authentication and role-based account management for patients, clinics, and administrators.

**Activities:**

- Sprint analysis
- Sprint design
- Implementation
- Testing
- Documentation

**Deliverables:**

- Functional micro release
- Document for the activities made

### **3.3.3.3 Sprint 2 (Patient Visit Management & Medical Records)**

**Objective:** Develop patient visit tracking and medical record management functionality.

**Activities:**

- Sprint analysis
- Sprint design
- Implementation
- Testing
- Documentation

**Deliverables:**

- Functional micro release
- Document for activities made

### **3.3.3.4 Sprint 3 (Speech-to-Text Integration)**

**Objective:** Integrate voice-enabled documentation using speech-to-text technology.

**Activities:**

- Sprint analysis
- Sprint design
- Implementation
- Testing with medical terminology
- Documentation

**Deliverables:**

- System second release
- Document for activities made

### **3.3.3.5 Sprint 4 (Appointment Management & Patient Portal)**

**Objective:** Implement appointment scheduling system and patient portal features.

**Activities:**

- Sprint analysis
- Sprint design
- Implementation
- Testing
- Documentation

**Deliverables:**

- System third release
- Document for activities made

**3.3.3.6 Sprint 5 (Administration Module & Final Integration)**

**Objective:** Develop administrative functions and integrate all system modules.

**Activities:**

- Sprint analysis
- Sprint design
- Implementation
- Integration testing
- Documentation

**Deliverables:**

- Complete system integration
- Document for activities made

**3.3.3.7 Sprint 6 (Final Testing & Documentation Review)**

**Objective:** Ensure system stability, security, and complete documentation.

**Activities:**

- System testing
- Security testing
- Acceptance testing
- Project closure and documentation

**Deliverables:**

- Final system report

**3.3.4 High-Level System Design**

**3.3.4.1 High-Level System Architecture**

The system adopts a hybrid architecture (Client-Server architecture) and the server adopts the three-tier architecture:

**Client:** Contains user interfaces, and what the user sees (patient portal, clinic dashboard, admin panel).

**Server:** Contains the logic, and what is invisible from the user:

- **Presentation Layer:** Entry point to the server, handles HTTP requests and responses
- **Business Logic Layer:** Contains the core logic of the platform (authentication, medical records, appointments, speech-to-text processing)
- **Data Access Layer:** Responsible for the communication with the database (PostgreSQL)

**External Services:**

- Speech-to-Text Service (Whisper)
- Email service for notifications and verification (Nodemailer)

Each of these layers is designed to be extensible, secure, testable and modular, which exactly meets our healthcare system needs and privacy requirements.

**3.3.4.2 High-Level System Decomposition**

Our system is made of 4 main cores:

1. **Account Management Module:** Contains how users (patients, clinics, admins) manage their profiles and handles authentication and authorization with role-based access control
2. **Medical Records Module:** Contains patient visit management, medical record storage and retrieval, and visit history tracking
3. **Speech-to-Text Module:** Contains voice input capture, speech recognition processing, and text output for clinical documentation
4. **Appointment Module:** Contains appointment scheduling, calendar management, and appointment notifications for patients and clinics

### **3.3.5 Progress Monitoring and Report**

Progress was tracked and evaluated through:

**Sprint Review:** We examine completed features during a sprint and we identify the weak areas that need to be improved for future sprints.

**Daily Standups:** A daily meeting that's duration is 5-15 minutes, we talk about what we have done, what we will do, and what are the problems that we're facing.

**Milestone Reports:** Formal presentations that happen during seminars to showcase the progress we've made, also happen when we reach a milestone during project life cycle, it can be used to see if the milestone date differs from the plan date.

### **3.3.6 Summary**

In the previous chapter, we've done activities like project management which is crucial to see how the project will be done, what's the effort needed for this project, what are the problems that we may face and we've made the project formal, then we looked at the system and studied it from a high-level perspective, looked at the main functional requirements and non-functional requirements, the system's architecture and the plan to achieve the non-functional requirements.

## **3.4 Initial RTM (Requirements Traceability Matrix)**

ID	Title	Analysis Section	Design Section	Code	Integration Test	Unit Test
VEMR-FR-PM-01	The system should allow patients to create a new account by providing email, password, first name, and last name.					
VEMR-FR-PM-02	The system should allow patients to login to the system using their email and password credentials.					
VEMR-FR-PM-03	The system should allow patients to verify their email address before accessing the system.					
VEMR-FR-PM-04	The system should allow users to request a password reset link via email and set a new password.					
VEMR-FR-PM-05	The system should allow users to view their profile information including personal details.					

VEMR-FR-PM-06	The system should allow users to update their profile information such as name, phone number, and other details.					
VEMR-FR-PM-07	The system should allow users to change their current password to a new one.					
VEMR-FR-AU-08	The system should allow administrators to login to the system using their credentials.					
VEMR-FR-AU-09	The system should allow clinicians/doctors to login to the system using their email and password.					
VEMR-FR-VM-10	The system should allow clinicians to create a new patient visit with date, type, reason, and chief complaint.					
VEMR-FR-VM-11	The system should allow clinicians to search visits by patient name or reason for visit.					

VEMR-FR-VM-12	The system should allow clinicians to edit visit details when the visit is in progress.					
VEMR-FR-VM-13	The system should allow clinicians to save changes made to a visit including medical record data.					
VEMR-FR-VM-14	The system should allow clinicians to delete a visit from the system.					
VEMR-FR-VM-15	The system should allow clinicians to view complete details of a visit including medical record, vitals, and allergies.					
VEMR-FR-VM-16	The system should allow voice transcription to appear in real-time as the clinician speaks using Whisper.					
VEMR-FR-VM-17	The system should allow clinicians to view a paginated list of all patients in the system.					

VEMR-FR-VM-18	The system allows clinicians to search for patient.					
VEMR-FR-VM-19	The system should allow clinicians to view a patient's profile with their visits and allergies.					
VEMR-FR-VM-20	The system should allow clinicians to view detailed patient information including contact and demographic data.					
VEMR-FR-VM-21	The system should allow clinicians to change visit status: start (planned to in-progress), complete, or cancel.					
VEMR-FR-VM-22	The system should allow clinicians to view a paginated list of all visits with filtering options.					
VEMR-FR-VS-23	The system should allow clinicians to view vital signs recorded for a specific visit.					

VEMR-FR-VS-24	The system should allow clinicians to record and edit vital signs (BP, HR, temp, etc.) during an active visit.					
VEMR-FR-AM-25	The system should allow clinicians to view a patient's recorded allergies with severity and reaction details.					
VEMR-FR-AM-26	The system should allow clinicians to add a new allergy record for a patient with type, allergen, severity, and reaction.					
VEMR-FR-AM-27	The system should allow clinicians to update existing allergy information for a patient.					
VEMR-FR-AM-28	The system should allow clinicians to delete an allergy record from a patient's profile.					
VEMR-FR-CM-29	The system should allow the admin to create clinicians' accounts.					

VEMR-FR-CM-30	The system should allow the clinicians and admins to update their account.					
VEMR-FR-CM-31	The system should allow the admin to view the clinicians accounts list.					
VEMR-FR-CM-32	The system should allow the admin to view the clinicians account details.					
VEMR-FR-CM-33	The system should allow the admin to delete the clinician's accounts.					
VEMR-FR-CM-34	The system should allow the admin to search the clinician's accounts.					
VEMR-FR-OM-35	The system should allow the admin to create organizations.					
VEMR-FR-OM-36	The system should allow the admin to update organization information.					
VEMR-FR-OM-37	The system should allow the admin to view all organizations.					

VEMR-FR-OM-38	The system should allow the admin to view detailed information about a specific organization.					
VEMR-FR-OM-39	The system should allow the admin to delete an organization.					
VEMR-FR-OM-40	The system should allow the admin to search for organizations.					
VEMR-FR-AN-41	The system should allow administrators to view system-wide analytics and reports.					
VEMR-FR-AP-42	The system should allow clinicians to view their appointment schedule.					
VEMR-FR-AP-43	The system should allow clinicians to view their daily appointment schedule.					
VEMR-FR-AP-44	The system should allow clinicians to filter appointments by specific date.					

VEMR-FR-AP-45	The system should allow clinicians to filter appointments by status.					
VEMR-FR-AP-46	The system should allow the doctor to check in a patient's appointment.					
VEMR-FR-AP-47	The system should allow the doctor to cancel a patient's appointment.					
VEMR-FR-AP-48	The system should allow the doctor to set an appointment as no show.					
VEMR-FR-DA-49	The system should allow clinicians to view their personal analytics and performance metrics.					
VEMR-FR-SM-50	The system should allow clinicians to create their work schedule.					
VEMR-FR-SM-51	The system should allow clinicians to edit their existing work schedule.					

VEMR-FR-SM-52	The system should allow clinicians to delete their work schedule.					
VEMR-FR-SM-53	The system should allow clinicians to automatically generate available visit slots based on their schedule.					
VEMR-FR-SM-54	The system should allow clinicians to mark appointments as completed, no-show, or cancelled.					
VEMR-FR-SM-55	The system should allow clinicians to view detailed information about a specific appointment.					
VEMR-FR-SM-56	The system should allow clinicians to define their available time slots for appointments.					
VEMR-FR-PP-57	The system should allow patients to view available healthcare organizations.					

VEMR-FR-PP-58	The system should allow patients to view doctors within specific organizations.					
VEMR-FR-PP-59	The system should allow patients to view available appointment slots.					
VEMR-FR-PP-60	The system should allow patients to book an appointment with a clinician.					
VEMR-FR-PP-61	The system should allow patients to view their own appointment history and upcoming appointments.					
VEMR-FR-PP-62	The system should allow patients to filter their appointments by status.					
VEMR-FR-PP-63	The system should allow patients to view their own medical records.					
VEMR-FR-PP-64	The system should allow patients to view their own allergy information.					

VEMR-FR-PP-65	The system should allow patients to view their complete visit history.					
---------------	--	--	--	--	--	--

**Table 5: Initial RTM****3.5 Initial Test Cases**

Test Case ID	Test Name	Purpose
TC-001	Register new user successfully	Verify that a new user can successfully register with valid credentials
TC-002	Throw ConflictException if email exists	Verify that the system prevents duplicate email registration
TC-003	Hash password before storing	Verify that passwords are properly hashed before storage
TC-004	Login successfully with valid credentials	Verify that a user can login with correct email and password
TC-005	Throw UnauthorizedException for invalid email	Verify that login fails with non-existent email
TC-006	Throw UnauthorizedException for invalid password	Verify that login fails with incorrect password
TC-007	Throw UnauthorizedException if email not verified	Verify that unverified users cannot login
TC-008	Throw UnauthorizedException if account not active	Verify that suspended accounts cannot login

TC-009	Verify email successfully	Verify that email verification works with a valid token
TC-010	Throw BadRequestException for invalid token	Verify that invalid tokens are rejected
TC-011	Throw BadRequestException for expired token	Verify that expired verification tokens are rejected
TC-012	Throw BadRequestException for already used token	Verify that used tokens cannot be reused
TC-013	Resend verification email successfully	Verify that verification email can be resent
TC-014	Throw BadRequestException if user not found	Verify error handling for non-existent user
TC-015	Throw BadRequestException if email already verified	Verify that verified users cannot request resend
TC-016	Send password reset email	Verify that password reset email is sent for valid user
TC-017	Return generic message if user not found	Verify security by not revealing user existence
TC-018	Reset password successfully	Verify that password can be reset using valid reset token
TC-019	Throw BadRequestException for invalid reset token	Verify that invalid reset tokens are rejected
TC-020	Throw BadRequestException for expired reset token	Verify that expired reset tokens are rejected

TC-021	Calculate expiration for seconds	Verify expiration calculation for seconds format
TC-022	Calculate expiration for minutes	Verify expiration calculation for minutes format
TC-023	Calculate expiration for hours	Verify expiration calculation for hours format
TC-024	Calculate expiration for days	Verify expiration calculation for days format
TC-025	AdminAuthService should be defined	Verify AdminAuthService is properly instantiated
TC-026	Have userType set to ADMIN	Verify correct user type configuration for admin
TC-027	Extend BaseAuthService	Verify inheritance and method availability for admin
TC-028	DoctorAuthService should be defined	Verify DoctorAuthService is properly instantiated
TC-029	Have userType set to DOCTOR	Verify correct user type configuration for doctor
TC-030	Extend BaseAuthService	Verify inheritance and method availability for doctor
TC-031	Register a new patient successfully	Verify patient registration with all required fields
TC-032	Throw ConflictException if email already exists	Verify duplicate email prevention for patients

TC-033	Login successfully with valid credentials	Verify patient can login with correct credentials
TC-034	Throw UnauthorizedException for invalid credentials	Verify login fails with wrong email
TC-035	Throw UnauthorizedException for wrong password	Verify login fails with incorrect password
TC-036	Throw UnauthorizedException if email not verified	Verify unverified patients cannot login
TC-037	Throw UnauthorizedException if account not active	Verify suspended patients cannot login
TC-038	Verify email successfully	Verify patient email verification process
TC-039	Throw BadRequestException for invalid token	Verify invalid verification tokens are rejected
TC-040	Throw BadRequestException for expired token	Verify expired tokens are rejected
TC-041	Send password reset email for existing user	Verify password reset flow for patients
TC-042	Return success message even for non-existing user	Verify security by not revealing user existence
TC-043	Reset password successfully	Verify password reset with valid token
TC-044	Throw BadRequestException for invalid token	Verify invalid reset tokens are rejected
TC-045	Service should be defined	Verify PatientAuthService is properly configured

TC-046	Use PatientRepository	Verify repository injection
TC-047	Use VerificationTokenRepository	Verify token repository injection
TC-048	Use EmailService	Verify email service injection
TC-049	Use JwtService	Verify JWT service injection
TC-050	Use ConfigService	Verify config service injection
TC-051	Create an allergy successfully	Verify allergy record creation with patient validation
TC-052	Throw NotFoundException if patient not found	Verify patient existence validation
TC-053	Throw ConflictException if duplicate allergy exists	Verify duplicate allergen prevention
TC-054	Return allergy by id	Verify allergy retrieval by ID
TC-055	Throw NotFoundException if allergy not found	Verify error handling for missing allergy
TC-056	Return allergies for a patient	Verify patient allergy list retrieval
TC-057	Return active allergies by default	Verify default filtering to active allergies only
TC-058	Return all allergies when activeOnly is false	Verify ability to retrieve all allergies including inactive
TC-059	Return paginated allergies	Verify pagination support for allergies
TC-060	Filter by patient id when provided	Verify patient-specific filtering
TC-061	Update allergy successfully	Verify allergy update functionality
TC-062	Throw NotFoundException if allergy not found	Verify update error handling

TC-063	Check for duplicates when allergen is changed	Verify duplicate validation on update
TC-064	Throw ConflictException if new allergen is duplicate	Verify duplicate prevention on update
TC-065	Not check duplicates if allergen unchanged	Verify optimization when allergen not changed
TC-066	Not check duplicates if allergen is same value	Verify duplicate check optimization
TC-067	Soft delete allergy	Verify soft delete functionality with timestamp
TC-068	Throw NotFoundException if allergy not found	Verify delete error handling
TC-069	Create an encounter successfully	Verify medical encounter creation with PLANNED status
TC-070	Throw NotFoundException if patient not found	Verify patient validation for encounter
TC-071	Return existing encounter if idempotency key matches	Verify idempotency key prevents duplicate encounters
TC-072	Return encounter by id	Verify encounter retrieval by ID
TC-073	Throw NotFoundException if encounter not found	Verify error handling for missing encounter
TC-074	Start a planned encounter	Verify status transition from PLANNED to IN_PROGRESS
TC-075	Throw BadRequestException if encounter not planned	Verify status validation before starting
TC-076	Complete an in-progress encounter	Verify status transition from IN_PROGRESS to COMPLETED
TC-077	Throw BadRequestException if encounter not in progress	Verify status validation before completing
TC-078	Cancel a planned encounter	Verify cancellation of PLANNED encounter

TC-079	Throw BadRequestException if encounter already completed	Verify completed encounters cannot be cancelled
TC-080	Cancel an in-progress encounter	Verify cancellation of IN_PROGRESS encounter
TC-081	Update encounter successfully	Verify encounter update functionality
TC-082	Throw ForbiddenException if not the assigned doctor	Verify access control for encounter updates
TC-083	Throw BadRequestException if encounter completed	Verify completed encounters cannot be updated
TC-084	Throw BadRequestException if encounter cancelled	Verify cancelled encounters cannot be updated
TC-085	Allow admin to update any encounter	Verify admin override for access control
TC-086	Soft delete encounter	Verify soft delete functionality
TC-087	Throw ForbiddenException if not the assigned doctor	Verify access control for encounter deletion
TC-088	Allow admin to delete any encounter	Verify admin override for deletion
TC-089	Return paginated encounters	Verify pagination support for encounters
TC-090	Apply filters when provided	Verify filtering by various criteria
TC-091	Return encounters for patient	Verify patient-specific encounter list
TC-092	Return active encounters for doctor	Verify doctor's active encounter list
TC-093	Not check idempotency if key not provided	Verify optional idempotency key
TC-094	Handle encounter status transitions correctly	Verify state machine transitions
TC-095	Validate encounter data integrity	Verify data validation rules

TC-096	Create a medical record successfully	Verify medical record creation linked to encounter
TC-097	Throw NotFoundException if patient not found	Verify patient validation for medical record
TC-098	Throw NotFoundException if encounter not found	Verify encounter validation
TC-099	Throw BadRequestException if record already exists	Verify duplicate prevention for encounter records
TC-100	Return medical record by id	Verify medical record retrieval
TC-101	Throw NotFoundException if record not found	Verify error handling for missing records
TC-102	Update medical record successfully	Verify medical record update
TC-103	Throw ForbiddenException if not the assigned doctor	Verify access control for updates
TC-104	Throw BadRequestException if record is finalized	Verify finalized records cannot be updated
TC-105	Finalize medical record successfully	Verify record finalization
TC-106	Throw BadRequestException if already finalized	Verify records cannot be re-finalized
TC-107	Soft delete medical record	Verify soft delete functionality
TC-108	Throw BadRequestException if record is finalized	Verify finalized records cannot be deleted
TC-109	Create a vital sign successfully	Verify vital signs recording for patient
TC-110	Throw NotFoundException if patient not found	Verify patient validation for vital signs
TC-111	Create multiple vital signs successfully	Verify bulk creation support

TC-112	Throw NotFoundException if patient not found	Verify bulk creation validation
TC-113	Return vital sign by id	Verify vital sign retrieval
TC-114	Throw NotFoundException if vital sign not found	Verify error handling for missing vital signs
TC-115	Return vital signs for a patient	Verify patient's vital sign history
TC-116	Return vital signs for an encounter	Verify encounter-specific vital signs
TC-117	Return latest vital sign of a type	Verify retrieval of most recent value by type
TC-118	Return null if no vital sign found	Verify handling of no data scenario
TC-119	Return latest vital signs of all types	Verify retrieval of latest of each vital sign type
TC-120	Soft delete vital sign	Verify soft delete functionality
TC-121	Throw NotFoundException if vital sign not found	Verify delete error handling
TC-122	Validate vital sign data ranges	Verify data integrity validation
TC-123	Create a past medical procedure successfully	Verify past procedure documentation
TC-124	Throw NotFoundException if patient not found	Verify patient validation for procedures
TC-125	Create procedure with minimal data	Verify support for optional fields
TC-126	Return procedure by id	Verify procedure retrieval
TC-127	Throw NotFoundException if procedure not found	Verify error handling for missing procedures
TC-128	Return all procedures for a patient	Verify patient's procedure history

TC-129	Return empty array if no procedures found	Verify handling of no data
TC-130	Return paginated procedures	Verify pagination support
TC-131	Filter by patient id when provided	Verify patient-specific filtering
TC-132	Update procedure successfully	Verify procedure update
TC-133	Throw NotFoundException if procedure not found	Verify update error handling
TC-134	Update with partial data	Verify partial update support
TC-135	Soft delete procedure successfully	Verify soft delete functionality
TC-136	Throw NotFoundException if procedure not found	Verify delete error handling
TC-137	Create a review of systems successfully	Verify system review documentation
TC-138	Throw NotFoundException if patient not found	Verify patient validation for reviews
TC-139	Create review without encounter id	Verify support for standalone reviews
TC-140	Create review with minimal data	Verify support for optional fields
TC-141	Create multiple reviews successfully	Verify bulk creation support
TC-142	Throw NotFoundException if patient not found	Verify bulk creation validation
TC-143	Create bulk reviews without encounter id	Verify bulk standalone reviews
TC-144	Return review by id	Verify review retrieval
TC-145	Throw NotFoundException if review not found	Verify error handling for missing reviews

TC-146	Return all reviews for a patient	Verify patient's review history
TC-147	Return empty array if no reviews found	Verify handling of no data
TC-148	Return all reviews for an encounter	Verify encounter-specific reviews
TC-149	Return empty array if no reviews found	Verify handling of no encounter data
TC-150	Return reviews filtered by category	Verify category-based filtering
TC-151	Return empty array if no reviews in category	Verify handling of empty category
TC-152	Update review successfully	Verify review update
TC-153	Throw NotFoundException if review not found	Verify update error handling
TC-154	Update with partial data	Verify partial update support
TC-155	Soft delete review successfully	Verify soft delete functionality
TC-156	Throw NotFoundException if review not found	Verify delete error handling
TC-157	Create a schedule successfully	Verify doctor availability schedule creation
TC-158	Throw ConflictException if schedule overlaps	Verify prevention of overlapping schedules
TC-159	Throw BadRequestException if end time before start time	Verify time validation
TC-160	Throw BadRequestException if day of week is invalid	Verify day validation (0-6)
TC-161	Throw BadRequestException if day of week is negative	Verify negative day validation

TC-162	Throw BadRequestException if slot duration is too small	Verify minimum 5 minutes duration
TC-163	Throw BadRequestException if slot duration is too large	Verify maximum 120 minutes duration
TC-164	Use default slot duration if not provided	Verify default 30 minutes duration
TC-165	Set effectiveFrom to current date if not provided	Verify default effective date
TC-166	Use provided effectiveFrom and effectiveUntil dates	Verify custom date ranges
TC-167	Return schedule by id	Verify schedule retrieval
TC-168	Throw NotFoundException if schedule not found	Verify error handling for missing schedules
TC-169	Return schedules for a doctor	Verify doctor's schedule list
TC-170	Update schedule successfully	Verify schedule update
TC-171	Throw ConflictException if update causes overlap	Verify conflict prevention on update
TC-172	Throw BadRequestException if day of week is invalid	Verify update day validation
TC-173	Throw BadRequestException if day of week is negative	Verify update negative day validation
TC-174	Throw BadRequestException if new times are invalid	Verify update time validation
TC-175	Throw BadRequestException if slot duration is too small	Verify update duration validation
TC-176	Throw BadRequestException if slot duration is too large	Verify update max duration validation
TC-177	Not check conflicts if day and times are not changing	Verify optimization for unchanged values

TC-178	Check conflicts when day is changed	Verify conflict check on day change
TC-179	Check conflicts when start time is changed	Verify conflict check on start change
TC-180	Check conflicts when end time is changed	Verify conflict check on end change
TC-181	Return active schedules for doctor	Verify active schedule filtering
TC-182	Check for conflicts without exclusion	Verify conflict detection
TC-183	Check for conflicts with exclusion	Verify conflict detection excluding current
TC-184	Return schedules for date range	Verify date range filtering
TC-185	Create an appointment successfully	Verify appointment booking with time slot
TC-186	Throw NotFoundException if time slot not found	Verify time slot validation
TC-187	Throw ConflictException if time slot not available	Verify prevention of double booking
TC-188	Return appointment by id	Verify appointment retrieval
TC-189	Throw NotFoundException if appointment not found	Verify error handling for missing appointments
TC-190	Cancel a scheduled appointment	Verify cancellation of SCHEDULED appointment
TC-191	Throw BadRequestException if appointment already completed	Verify completed appointments cannot be cancelled
TC-192	Not release time slot for late cancellations	Verify <24 hours cancellation policy
TC-193	Release time slot for early cancellations	Verify >24 hours cancellation policy

TC-194	Throw BadRequestException for past appointments	Verify past appointments cannot be cancelled
TC-195	Check in a scheduled appointment	Verify status transition to CHECKED_IN
TC-196	Throw BadRequestException if appointment not scheduled	Verify check-in validation
TC-197	Start a visit for checked-in appointment	Verify status transition to IN_PROGRESS
TC-198	Complete an in-progress appointment	Verify status transition to COMPLETED
TC-199	Mark appointment as no-show	Verify no-show handling
TC-200	Allow valid transitions	Verify state machine transitions
TC-201	Reject invalid transitions	Verify prevention of invalid state changes
TC-202	Allow cancellation from SCHEDULED	Verify SCHEDULED to CANCELLED transition
TC-203	Allow cancellation from CHECKED_IN	Verify CHECKED_IN to CANCELLED transition
TC-204	Allow NO_SHOW from SCHEDULED	Verify SCHEDULED to NO_SHOW transition
TC-205	Allow NO_SHOW from CHECKED_IN	Verify CHECKED_IN to NO_SHOW transition
TC-206	Not allow transitions from COMPLETED	Verify terminal state validation
TC-207	Not allow transitions from CANCELLED	Verify terminal state validation
TC-208	Not allow transitions from NO_SHOW	Verify terminal state validation
TC-209	Return true for past appointments	Verify past date detection

TC-210	Return false for future appointments	Verify future date detection
TC-211	Return true for cancellations more than 24 hours before	Verify early cancellation detection
TC-212	Return false for cancellations less than 24 hours before	Verify late cancellation detection
TC-213	Return paginated appointments for patient	Verify patient appointment list
TC-214	Return paginated appointments for doctor	Verify doctor appointment list
TC-215	Return valid transitions map	Verify state machine configuration
TC-216	Generate slots for matching days	Verify time slot generation from schedule
TC-217	Throw NotFoundException if schedule not found	Verify schedule validation
TC-218	Throw BadRequestException if schedule is inactive	Verify active schedule validation
TC-219	Not generate slots for non-matching days	Verify day matching logic
TC-220	Return time slot by id	Verify time slot retrieval
TC-221	Throw NotFoundException if slot not found	Verify error handling for missing slots
TC-222	Mark available slot as booked	Verify status change from AVAILABLE to BOOKED
TC-223	Throw BadRequestException if slot is not available	Verify booking validation
TC-224	Mark booked slot as available	Verify status change from BOOKED to AVAILABLE
TC-225	Throw BadRequestException if slot is not booked	Verify release validation

TC-226	Return available slots	Verify filtering by AVAILABLE status
TC-227	Block slots in range	Verify slot blocking for exceptions
TC-228	Expire old slots	Verify marking past slots as EXPIRED
TC-229	Calculate correct slot count	Verify slot count calculation
TC-230	Handle different durations	Verify duration-based calculation
TC-231	Create exception and block slots	Verify exception creation and slot blocking
TC-232	Throw BadRequestException if end time before start time	Verify time validation for exceptions
TC-233	Throw BadRequestException for past dates	Verify past date validation
TC-234	Identify affected appointments	Verify finding appointments in exception range
TC-235	Return exception by id	Verify exception retrieval
TC-236	Throw NotFoundException if not found	Verify error handling for missing exceptions
TC-237	Delete exception and unblock slots	Verify exception removal and slot unblocking
TC-238	Return affected appointments	Verify listing affected appointments
TC-239	Check if time is blocked	Verify time blocking validation
TC-240	Return upcoming exceptions	Verify future exception list
TC-241	Create an organization successfully	Verify organization creation with all details
TC-242	Throw ConflictException if code already exists	Verify unique code validation

TC-243	Throw BadRequestException if name is empty	Verify name required validation
TC-244	Throw BadRequestException if name is only whitespace	Verify whitespace validation
TC-245	Create organization without code	Verify optional code field
TC-246	Trim whitespace from all fields	Verify input sanitization
TC-247	Return organization by id	Verify organization retrieval
TC-248	Throw NotFoundException if organization not found	Verify error handling for missing organizations
TC-249	Return paginated organizations	Verify pagination support
TC-250	Filter by search term when provided	Verify search functionality
TC-251	Not include search filter when search is undefined	Verify optional search
TC-252	Return active organizations	Verify active organization filtering
TC-253	Update organization successfully	Verify organization update
TC-254	Throw NotFoundException if organization not found	Verify update error handling
TC-255	Throw ConflictException if new code already exists	Verify code uniqueness on update
TC-256	Not check code conflict if code unchanged	Verify optimization for unchanged code
TC-257	Update all fields with trimmed values	Verify update sanitization
TC-258	Handle undefined optional fields	Verify optional field handling
TC-259	Activate an organization	Verify setting isActive to true

TC-260	Deactivate an organization	Verify setting isActive to false
TC-261	Soft delete organization	Verify soft delete functionality
TC-262	Throw NotFoundException if organization not found	Verify delete error handling
TC-263	Return organization statistics	Verify calculation of doctor and appointment counts
TC-264	Throw NotFoundException if organization not found	Verify statistics error handling
TC-265	Calculate total appointments correctly	Verify appointment count aggregation
TC-266	Return doctors for organization	Verify organization's doctor list
TC-267	Throw NotFoundException if organization not found	Verify doctor list error handling
TC-268	Additional organization validation tests	Verify comprehensive validation rules
TC-269	Create a new doctor successfully	Verify doctor account creation with organization
TC-270	Throw ConflictException if email already exists	Verify duplicate email prevention
TC-271	Throw BadRequestException if email is empty	Verify email required validation
TC-272	Throw NotFoundException if organization not found	Verify organization validation
TC-273	Throw BadRequestException if organization is inactive	Verify active organization validation
TC-274	Throw BadRequestException if firstName is empty	Verify first name required
TC-275	Throw BadRequestException if lastName is empty	Verify last name required

TC-276	Throw BadRequestException if organizationId is missing	Verify organization required
TC-277	Generate temporary password if not provided	Verify auto-generation of secure password
TC-278	Trim whitespace from fields	Verify input sanitization
TC-279	Handle email service error gracefully	Verify email failure handling
TC-280	Return doctor with organizations	Verify doctor retrieval with org relationships
TC-281	Throw NotFoundException if doctor not found	Verify error handling for missing doctors
TC-282	Update doctor successfully	Verify doctor update
TC-283	Throw NotFoundException if doctor not found	Verify update error handling
TC-284	Suspend doctor and cancel future appointments	Verify account suspension and appointment cancellation
TC-285	Throw BadRequestException if already suspended	Verify prevention of re-suspension
TC-286	Activate doctor successfully	Verify activation of suspended account
TC-287	Throw BadRequestException if no organizations assigned	Verify organization requirement
TC-288	Throw BadRequestException if already active	Verify prevention of re-activation
TC-289	Throw NotFoundException if doctor not found	Verify activation error handling
TC-290	Assign doctor to organization successfully	Verify organization assignment creation
TC-291	Throw NotFoundException if doctor not found	Verify assignment error handling

TC-292	Throw NotFoundException if organization not found	Verify organization validation
TC-293	Throw BadRequestException if organization is inactive	Verify active org validation
TC-294	Default isPrimary to false	Verify default primary flag
TC-295	Remove doctor from organization successfully	Verify organization assignment removal
TC-296	Throw BadRequestException if only one organization	Verify requirement of at least one org
TC-297	Throw NotFoundException if assignment not found	Verify assignment validation
TC-298	Throw NotFoundException if doctor not found	Verify removal error handling
TC-299	Return doctors for organization	Verify organization's doctor list
TC-300	Throw NotFoundException if organization not found	Verify list error handling
TC-301	Delete doctor and cancel appointments	Verify soft delete and appointment cancellation
TC-302	Return doctor statistics	Verify calculation of appointment and patient counts
TC-303	Return user profile	Verify user profile retrieval without password
TC-304	Throw NotFoundException if user not found	Verify error handling for missing users
TC-305	Update user profile successfully	Verify profile update
TC-306	Throw NotFoundException if user not found	Verify update error handling
TC-307	Change password successfully	Verify password update with validation

TC-308	Throw BadRequestException if current password is incorrect	Verify password verification
TC-309	Throw NotFoundException if user not found	Verify password change error handling
TC-310	Delete account successfully	Verify soft delete of user account
TC-311	DoctorProfileService should be defined	Verify service instantiation
TC-312	Extend BaseProfileService	Verify inheritance
TC-313	Use DoctorRepository	Verify repository injection
TC-314	Get doctor profile with organizations	Verify inclusion of organization relationships
TC-315	Update doctor-specific fields	Verify doctor-specific data updates
TC-316	PatientProfileService should be defined	Verify service instantiation
TC-317	Extend BaseProfileService	Verify inheritance
TC-318	Use PatientRepository	Verify repository injection
TC-319	Get patient profile	Verify patient-specific profile retrieval
TC-320	Update patient-specific fields	Verify patient-specific data updates
TC-321	Create transporter with SMTP config	Verify email transporter initialization
TC-322	Throw error if SMTP config is missing	Verify configuration validation
TC-323	Throw error if FRONTEND_URL is missing	Verify frontend URL validation
TC-324	Send verification email successfully	Verify email sending with verification link

**Table 6: Initial Test Cases**

### **3.6 Project Management Using Jira (Scrumban Methodology)**

To ensure effective planning, monitoring, collaboration, and timely delivery of the Voice-Based EMR System, Jira is used as the primary project management tool. The project adopts the **Scrumban methodology**, a hybrid approach that combines Scrum-based planning and milestones with Kanban-based task visualization and workflow management.

This methodology enables structured sprint planning while maintaining flexibility in task execution and continuous progress tracking. It is particularly suitable for projects with evolving requirements, such as healthcare systems, where adaptability and quality assurance are critical.

### **3.6.1 Scrumban Methodology Overview**

The Scrumban approach used in this project integrates key Scrum practices, including:

- Sprint planning and defined sprint goals
- Incremental and iterative development
- Sprint reviews and milestone evaluations

These practices are combined with Kanban principles, such as:

- Visualizing work using a task board
- Managing work-in-progress (WIP)
- Continuous task flow across defined stages

This hybrid methodology ensures disciplined project management while allowing flexibility in task execution and prioritization.

### **3.6.2 Jira Board Structure and Workflow**

The Jira board is designed to represent the complete lifecycle of project requirements and tasks. Each Jira issue (requirement, feature, or defect) progresses through clearly defined workflow stages, ensuring traceability, transparency, and quality control.

The Jira board consists of the following columns:

- **Requirements Backlog**  
Contains high-level functional and non-functional requirements derived from the Software Requirements Specification (SRS) and Requirements Traceability Matrix (RTM). These items represent the initial scope of the project.
- **Sprint Backlog**  
Includes selected requirements and tasks planned for implementation during the current sprint, as determined during sprint planning meetings.

- **Analysis**

Represents tasks under requirement analysis, including functional clarification, feasibility assessment, and validation of healthcare-related constraints.

- **Design**

Includes system design activities such as architectural design, database modeling, API specification, and user interface design.

- **Development**

Represents active implementation of features in the frontend and backend components of the system.

- **Testing**

Includes functional testing, integration testing, and validation against acceptance criteria and requirements.

- **2 Eyes QA**

A peer-review quality assurance stage where another team member reviews the implemented functionality to ensure correctness, code quality, and compliance with system requirements.

- **To Push to Production**

Tasks that have successfully passed testing and quality assurance and are approved for deployment.

- **Pushed to Production**

Features that have been deployed to the production environment.

- **Done**

Completed tasks that fully satisfy acceptance criteria and require no further action.

This structured workflow ensures disciplined progress tracking and supports continuous delivery with high quality standards.

### **3.6.3 Jira-Based Sprint Planning and Monitoring**

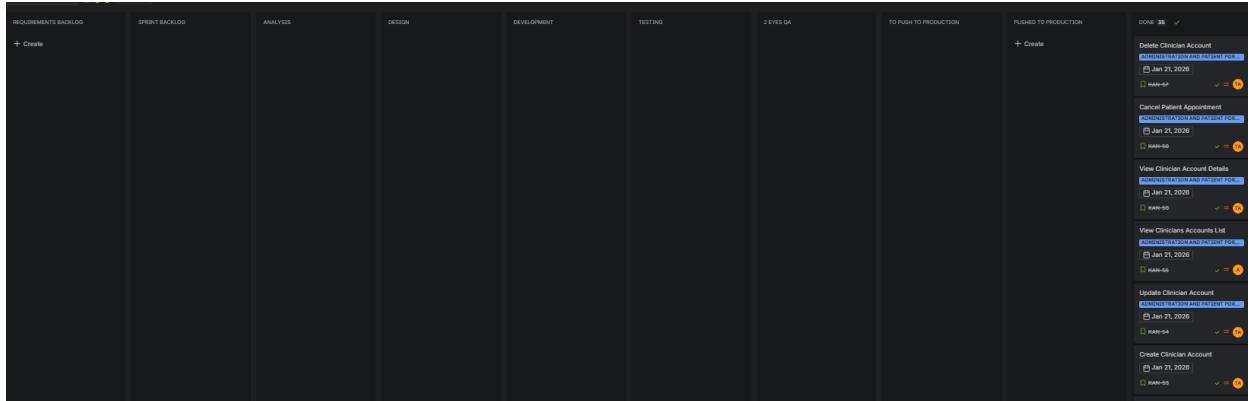
Sprint planning sessions are conducted to select and prioritize requirements from the Requirements Backlog into the Sprint Backlog. Each sprint has a clearly defined objective aligned with project milestones.

Progress is monitored through:

- Daily stand-up meetings
- Continuous task movement across Jira board stages
- Sprint review meetings to evaluate completed work
- Milestone and seminar presentations

This approach ensures early identification of risks, timely issue resolution, and alignment with the project timeline.

### **3.6.4 Jira Board Figure Description**



**Figure 3: Jira Scrumban Board Workflow**

### 3.7 Version Control and Configuration Management Using Git and GitHub

To manage source code evolution, ensure configuration control, and support collaborative development, the project uses **Git** as the version control system and **GitHub** as the centralized code repository and collaboration platform.

This approach ensures traceability between system requirements, implementation, testing, and delivered functionality while maintaining code integrity and stability.

#### 3.7.1 Git-Based Version Control Strategy

Git is used to track all changes to the source code throughout the development lifecycle. It enables multiple developers to work concurrently while maintaining a complete history of modifications.

The project follows a structured branching strategy to support parallel development and controlled releases.

#### 3.7.2 Branching Model

The branching strategy adopted for this project includes:

- **Main Branch**  
Contains stable, production-ready versions of the system.

- **Develop Branch**

Serves as the integration branch for completed features before production deployment.  
the develop branch after review and testing.

This model ensures system stability and controlled integration of new features.

### **3.7.3 GitHub Workflow and Collaboration**

GitHub is used to host the repository and facilitate collaboration among team members. The workflow includes:

- Creating feature branches linked to Jira issues
- Submitting Pull Requests before merging code
- Performing peer code reviews as part of the **2 Eyes QA** process
- Resolving review comments and retesting before approval
- Merging approved code into the develop or main branches

This process ensures high code quality and accountability.

### **3.7.4 Commit Conventions and Traceability**

Commit messages follow a clear and consistent convention and reference Jira issue identifiers. This practice provides traceability between:

- System requirements
- Development activities
- Testing and quality assurance

- Delivered features

Traceability is essential for auditing, maintenance, and academic evaluation.

### **3.7.5 Quality Assurance and Deployment Control**

Before deployment:

- Code must pass functional and integration testing
- Peer review must be completed
- Tasks must reach the **To Push to Production** stage in Jira

Only verified and approved code is deployed to production, ensuring reliability and compliance with system requirements.

## **Chapter 4 - System Analysis**

## 4.1 Introduction

This chapter provides a comprehensive analysis of the Voice-Based Medical System. It defines the system's goals, capabilities, and technical specifications by examining clinician needs and the healthcare workflow. The chapter outlines both functional and non-functional requirements and presents the key actors and their interaction processes. Together, these elements form the foundation for designing and developing a modern, intelligent clinical platform powered by voice-enabled documentation and automated patient management.

## 4.2 Purpose

Our purpose is to design and implement a structured educational platform that enables learners to gain foundational and advanced knowledge in AI algorithms, particularly in clustering and exploratory data analytics, through guided lessons, interactive visualizations, and providing structured projects.

## 4.3 Project Scope

The Voice-Based Medical System aims to streamline clinical workflows by providing voice-enabled documentation, intelligent patient management, and automated appointment handling. The system focuses on enhancing clinical efficiency through AI-powered features that support tasks such as generating medical notes, retrieving patient information, and organizing clinical data. The specification defines the high-level requirements of the platform and identifies the main actors interacting with the system.

## 4.4 Requirements Elicitation

While analyzing existing healthcare systems and conducting our feasibility study, we examined the strengths and weaknesses of other voice-enabled and EMR solutions. Our goal was to extract the best requirements from all evaluated systems in order to build a comprehensive, efficient, and user-friendly clinical platform. This ensures that the system supports clinicians at different experience levels and makes clinical documentation faster, smoother, and more intuitive through voice-driven interaction.

## 4.5 Requirements Table

Req_ID	Requirement Title	Category	Type
VEMR-FR-PM-01	The system should allow patients to create a new account by providing email, password, first name, and last name.	Patient Management	Functional
VEMR-FR-PM-02	The system should allow patients to login to the system using their email and password credentials.	PatientManagement	Functional
VEMR-FR-PM-03	The system should allow patients to verify their email address before accessing the system.	Patient Management	Functional
VEMR-FR-PM-04	The system should allow users to request a password reset link via email and set a new password.	Patient Management	Functional
VEMR-FR-PM-05	The system should allow users to view their profile information including personal details.	Patient Management	Functional
VEMR-FR-PM-06	The system should allow users to update their profile information such as name, phone number, and other details.	Patient Management	Functional
VEMR-FR-PM-07	The system should allow users to change their current password to a new one.	Patient Management	Functional

## Chapter 4 - System Analysis

VEMR-FR-AU-08	The system should allow administrators to login to the system using their credentials.	Authentication	Functional
VEMR-FR-AU-09	The system should allow clinicians/doctors to login to the system using their email and password.	Authentication	Functional
VEMR-FR-VM-10	The system should allow clinicians to create a new patient visit with date, type, reason, and chief complaint.	Visit Management	Functional
VEMR-FR-VM-11	The system should allow clinicians to search visits by patient name or reason for visit.	Visit Management	Functional
VEMR-FR-VM-12	The system should allow clinicians to edit visit details when the visit is in progress.	Visit Management	Functional
VEMR-FR-VM-13	The system should allow clinicians to save changes made to a visit including medical record data.	Visit Management	Functional
VEMR-FR-VM-14	The system should allow clinicians to delete a visit from the system.	Visit Management	Functional
VEMR-FR-VM-15	The system should allow clinicians to view complete details of a visit including medical record, vitals, and allergies.	Visit Management	Functional
VEMR-FR-VM-16	The system should allow voice transcription to appear in real-time as the clinician speaks using Whisper.	Visit Management	Functional

## Chapter 4 - System Analysis

VEMR-FR-VM-17	The system should allow clinicians to view a paginated list of all patients in the system	Visit Management	Functional
VEMR-FR-VM-18	The system allows clinicians to search for patient	Visit Management	Functional
VEMR-FR-VM-19	The system should allow clinicians to view a patient's profile with their visits and allergies.	Visit Management	Functional
VEMR-FR-VM-20	The system should allow clinicians to view detailed patient information including contact and demographic data.	Visit Management	Functional
VEMR-FR-VM-21	The system should allow clinicians to change visit status: start (planned to in-progress), complete, or cancel.	Visit Management	Functional
VEMR-FR-VM-22	The system should allow clinicians to view a paginated list of all visits with filtering options.	Visit Management	Functional
VEMR-FR-VS-23	The system should allow clinicians to view vital signs recorded for a specific visit.	Vital Signs Management	Functional
VEMR-FR-VS-24	The system should allow clinicians to record and edit vital signs (BP, HR, temp, etc.) during an active visit.	Vital Signs Management	Functional
VEMR-FR-AM-25	The system should allow clinicians to view a patient's recorded allergies with severity and reaction details.	Allergy Management	Functional

## Chapter 4 - System Analysis

VEMR-FR-AM-26	The system should allow clinicians to add a new allergy record for a patient with type, allergen, severity, and reaction.	Allergy Management	Functional
VEMR-FR-AM-27	The system should allow clinicians to update existing allergy information for a patient.	Allergy Management	Functional
VEMR-FR-AM-28	The system should allow clinicians to delete an allergy record from a patient's profile	Allergy Management	Functional
VEMR-FR-CM-29	The system should allow the admin to create clinicians' accounts.	Clinician Management	Functional
VEMR-FR-CM-30	The system should allow the clinicians and admins to update their account.	Clinician Management	Functional
VEMR-FR-CM-31	The system should allow the admin to view the clinicians accounts list	Clinician Management	Functional
VEMR-FR-CM-32	The system should allow the admin to view the clinicians account details.	Clinician Management	Functional
VEMR-FR-CM-33	The system should allow the admin to delete the clinician's accounts.	Clinician Management	Functional
VEMR-FR-CM-34	The system should allow the admin to search the clinician's accounts.	Clinician Management	Functional
VEMR-FR-OM-35	The system should allow the admin to create organizations.	Organization Management	Functional

## Chapter 4 - System Analysis

VEMR-FR-OM-36	The system should allow the admin to update organization information.	Organization Management	Functional
VEMR-FR-OM-37	The system should allow the admin to view all organizations.	Organization Management	Functional
VEMR-FR-OM-38	The system should allow the admin to view detailed information about a specific organization.	Clinician Management	Functional
VEMR-FR-OM-39	The system should allow the admin to delete an organization	Clinician Management	Functional
VEMR-FR-OM-40	The system should allow the admin to search for organizations.	Clinician Management	Functional
VEMR-FR-AN-41	The system should allow administrators to view system-wide analytics and reports.	Analytics	Functional
VEMR-FR-AP-42	The system should allow clinicians to view their appointment schedule.	Appointment Management	Functional
VEMR-FR-AP-43	The system should allow clinicians to view their daily appointment schedule.	Appointment Management	Functional
VEMR-FR-AP-44	The system should allow clinicians to filter appointments by specific date.	Appointment Management	Functional
VEMR-FR-AP-45	The system should allow clinicians to filter appointments by status.	Appointment Management	Functional

VEMR-FR-AP-46	The system should allow the doctor to check in a patient's appointment	Appointment Management	Functional
VEMR-FR-AP-47	The system should allow the doctor to cancel a patient's appointment	Appointment Management	Functional
VEMR-FR-AP-48	The system should allow the doctor to set an appointment as no show	Appointment Management	Functional
VEMR-FR-DA-49	The system should allow clinicians to view their personal analytics and performance metrics.	Doctor Analytics	Functional
VEMR-FR-SM-50	The system should allow clinicians to create their work schedule.	Schedule Management	Functional
VEMR-FR-SM-51	The system should allow clinicians to edit their existing work schedule.	Schedule Management	Functional
VEMR-FR-SM-52	The system should allow clinicians to delete their work schedule.	Schedule Management	Functional
VEMR-FR-SM-53	The system should allow clinicians to automatically generate available visit slots based on their schedule.	Schedule Management	Functional
VEMR-FR-SM-54	The system should allow clinicians to mark appointments as completed, no-show, or cancelled.	Schedule Management	Functional
VEMR-FR-SM-55	The system should allow clinicians to view detailed	Schedule Management	Functional

	information about a specific appointment.		
VEMR-FR-SM-56	The system should allow clinicians to define their available time slots for appointments.	Schedule Management	Functional
VEMR-FR-PP-57	The system should allow patients to view available healthcare organizations..	Patient Portal	Functional
VEMR-FR-PP-58	The system should allow patients to view doctors within specific organizations	Patient Portal	Functional
VEMR-FR-PP-59	The system should allow patients to view available appointment slots.	Patient Portal	Functional
VEMR-FR-PP-60	The system should allow patients to book an appointment with a clinician..	Patient Portal	Functional
VEMR-FR-PP-61	The system should allow patients to view their own appointment history and upcoming appointments.	Patient Portal	Functional
VEMR-FR-PP-62	The system should allow patients to filter their appointments by status.	Patient Portal	Functional
VEMR-FR-PP-63	The system should allow patients to view their own medical records.	Patient Portal	Functional
VEMR-FR-PP-64	The system should allow patients to view their own allergy information.	Patient Portal	Functional

VEMR-FR-PP-65	The system should allow patients to view their complete visit history.	Patient Portal	Functional
---------------	--	----------------	------------

Table 7, Requirements Table

## 4.6 Analysis

In this Section, we will introduce the analytical study for the first sprint (Account Management and User Authentication), The analysis includes identifying and modeling the primary functional

requirements through Unified Modeling Language (UML) diagrams, such as use case, activity and sequence diagrams, to visualize system behavior and interactions. This structured approach ensures alignment between user needs, functional goals, and software components, providing a clear roadmap for implementation and testing during this sprint.

**- The list of requirements that we will complete for this sprint:**

- ❖ **VEMR-FR-AM-01 :** The system should allow clinicians to register for an account using their email and password
- ❖ **VEMR-FR-AM-02:** The system should allow clinicians to log in to their accounts using their credentials
- ❖ **VEMR-FR-AM-03:** The system should allow users to verify their accounts via email verification link
- ❖ **VEMR-FR-AM-04:** The system should allow users to reset their password using the emailed link
- ❖ **VEMR-FR-AM-05:** The system should allow users to view their profile data
- ❖ **VEMR-FR-AM-06:** The system should allow users to update their profile information
- ❖ **VEMR-FR-AM-07:** The system should allow users to change their

#### **4.6.1 Requirements Modeling**

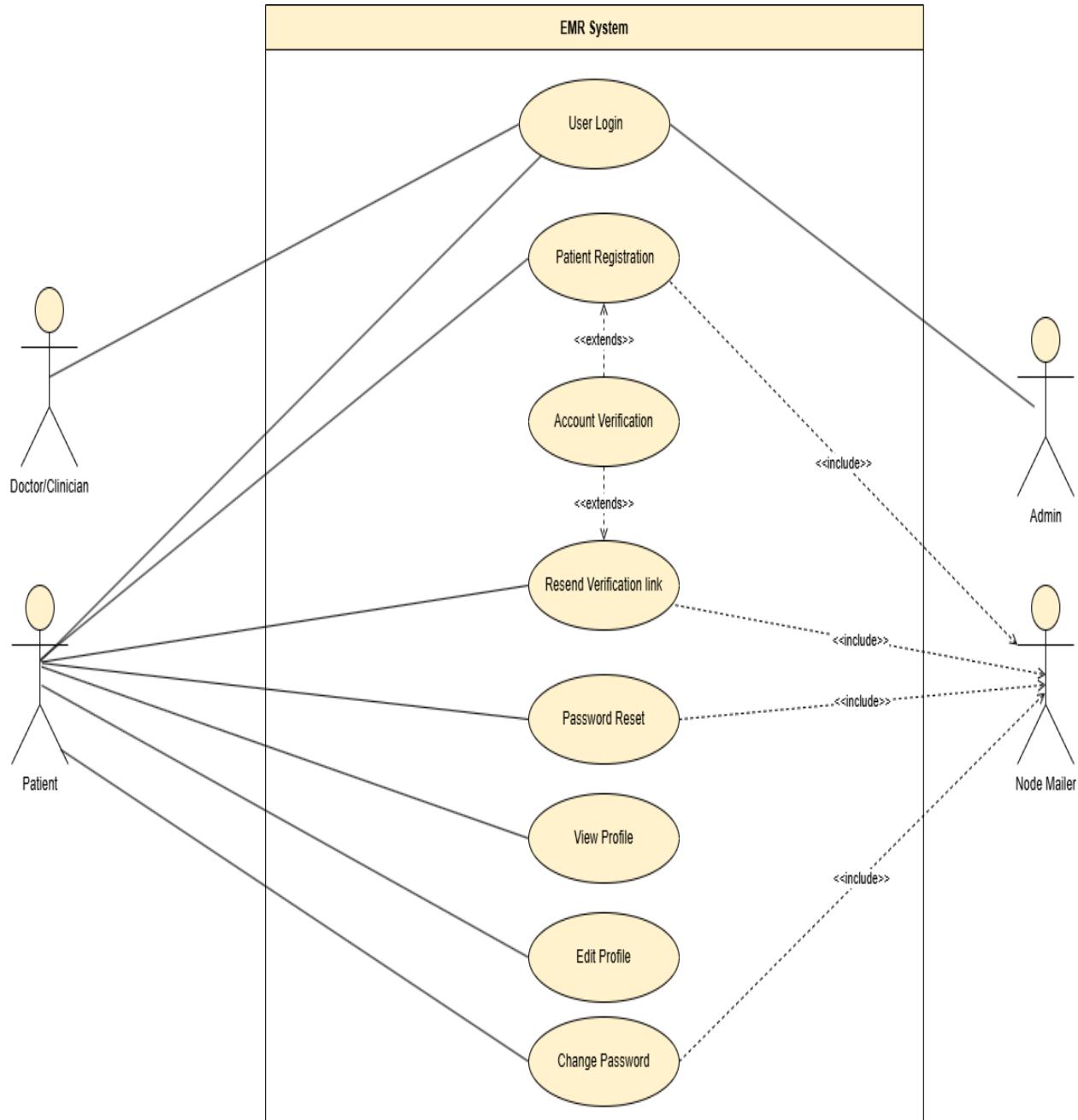


Figure 4, Sprint 1 Use Case Diagram

- **Sign up :**

<b>Use case ID</b>	<b>VEMR-FR-AM-01</b>
<b>Use case name</b>	Patient Registration
<b>Description</b>	The system allow patient to register for an account using their email and password
<b>Actor</b>	Patient
<b>Pre-conditions</b>	Patient does not have an exist
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. Patient navigates to the registration page</li> <li>2. Patient enters registration information: (First name , Last name, Email ,Password)</li> <li>3. System validates the input data</li> <li>4. System checks if email is already registered</li> <li>5. System creates patient account</li> <li>6. System generates verification token</li> <li>7. System send verification email to patient</li> <li>8. System return success message with patu</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Email Already Registered</b></p> <ul style="list-style-type: none"> <li>- At step 4 , if email already exists</li> <li>-System returns Error ."Email already registered"</li> </ul> <p><b>A2: Invalid Input Data</b></p> <ul style="list-style-type: none"> <li>- At step 3, if validation fails</li> <li>-System return validation error</li> </ul> <p><b>A3:Email Sending Failure</b></p> <ul style="list-style-type: none"> <li>- At step 8 , if email fails to send</li> <li>-System logs the error</li> <li>-Account is still created</li> <li>-Patient can request resend verification email later</li> </ul>
<b>Post condition</b>	Patient is authenticated

Table 8 , Use Case Specification ( Sign up )

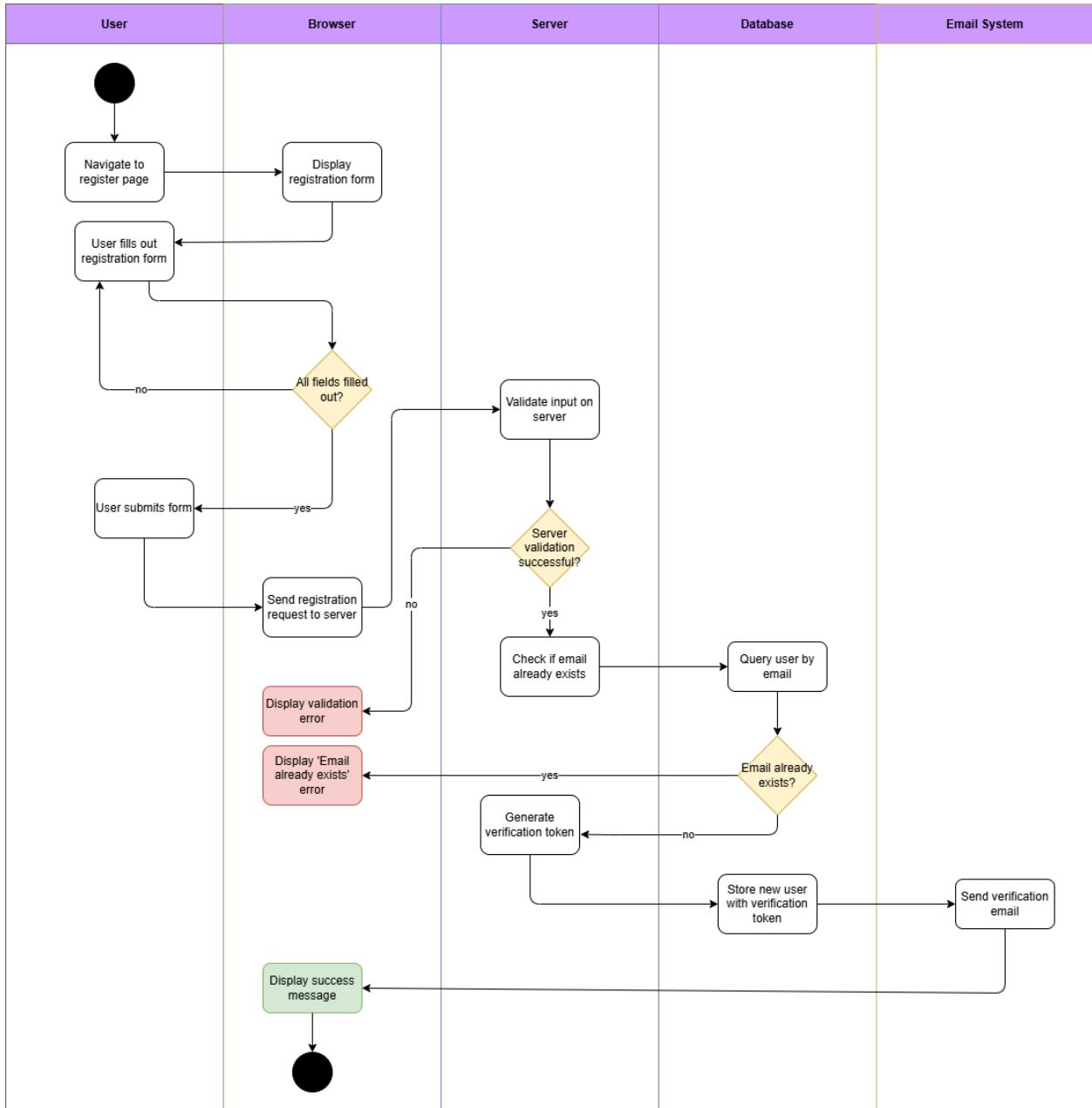


Figure 5, Activity Diagram ( Sign up )

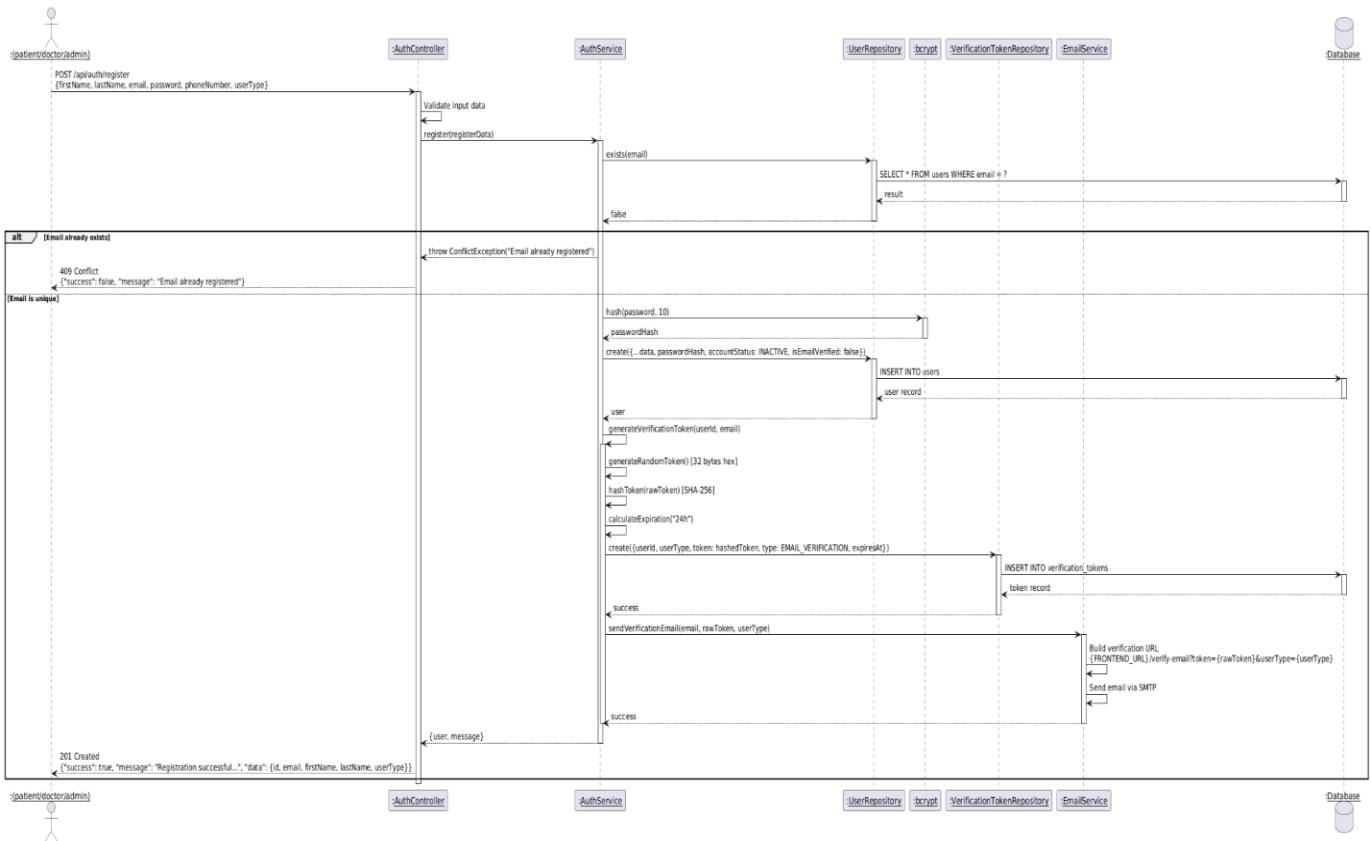


Figure 6, Sequence Diagram ( Sign up )

● Log in :

Use case ID	VEMR-FR-AM-02
-------------	---------------

<b>Use case name</b>	Patient Login
<b>Description</b>	This use case allows the Patient to login into his account
<b>Actor</b>	Patient
<b>Pre-conditions</b>	- User has Registered account -User email is verified
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. User navigate to login page</li> <li>2. User enters credentials ( Email , Password )</li> <li>3. System validates the input data</li> <li>4. System retrieves User record by email</li> <li>5. System compares password with stored hash</li> <li>6. System verifies email is verified</li> <li>7. System checks account status is ACTIVE</li> <li>8. System updates last login timestamp</li> <li>9. System generate JWT access token and refresh token</li> <li>10. System returns tokens and User information</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Invalid Credentials</b></p> <ul style="list-style-type: none"> <li>- At step 5 , if email not found or password doesn't match</li> <li>-System returns error : "Invalid credentials"</li> </ul> <p><b>A2: Email not verified</b></p> <ul style="list-style-type: none"> <li>- At step 5 , if email is not verified</li> <li>-System return error :"Please verify your email before login"</li> </ul> <p><b>A3: Account Not active</b></p> <ul style="list-style-type: none"> <li>- At step 7,if account status is not ACTIVE</li> <li>-System return error: " Account is not active"</li> </ul>
<b>Post condition</b>	User is authenticated

Table 9 , Use Case Specification ( Log in )

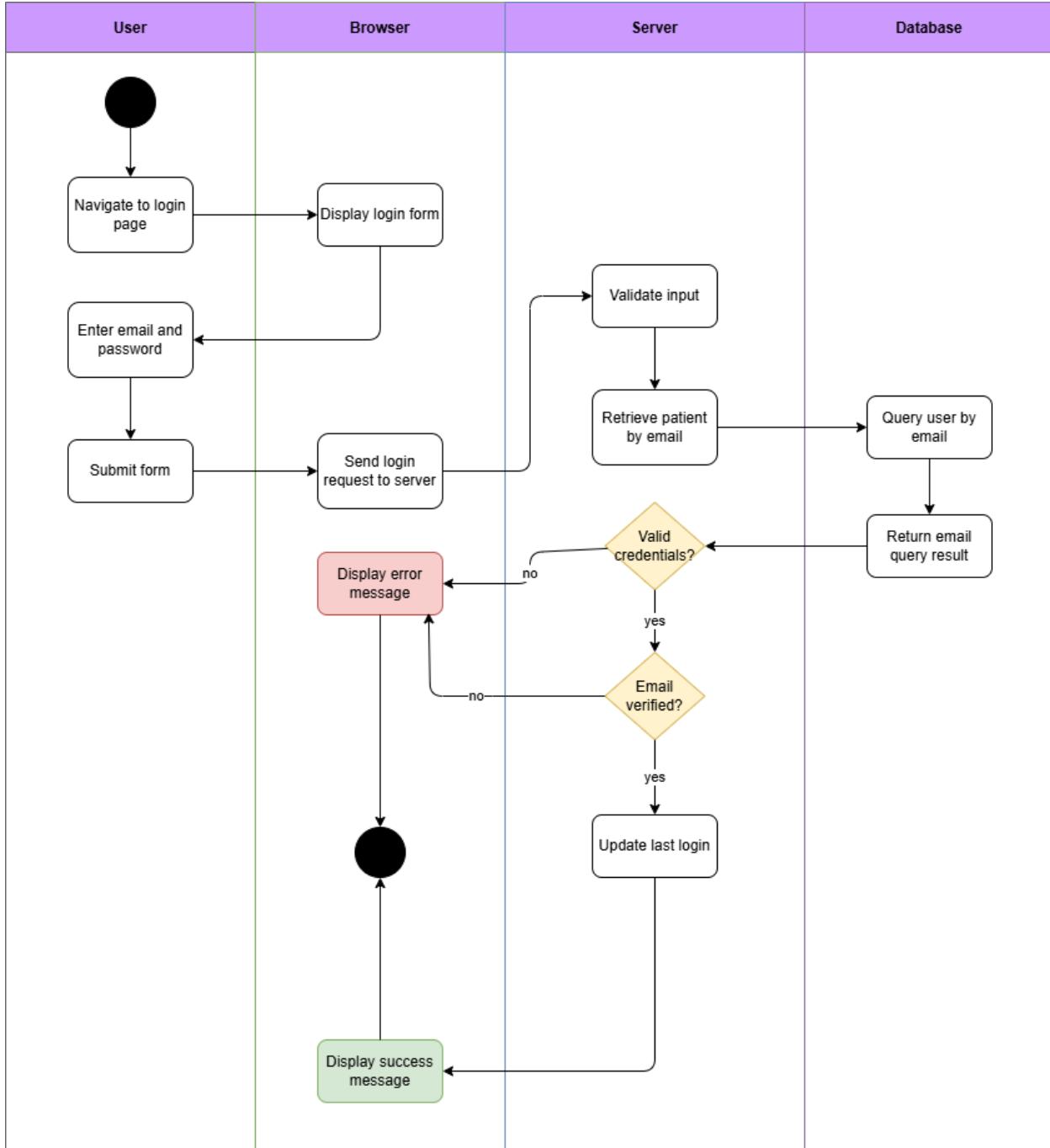


Figure 7, Activity Diagram ( Log in )

## Chapter 4 - System Analysis

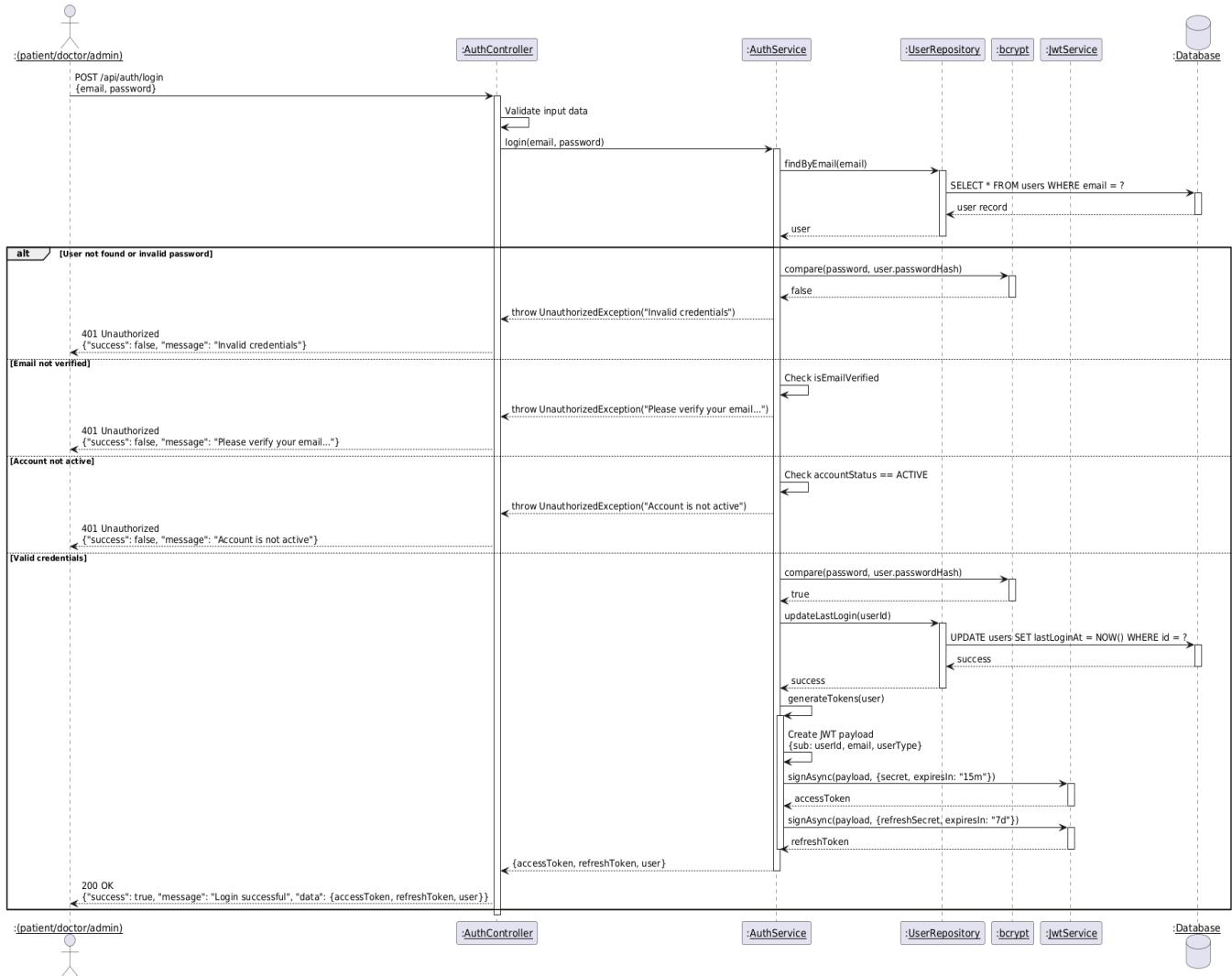


Figure 8, Sequence Diagram ( Log in )

## ● Account Verification :

<b>Use case ID</b>	<b>VEMR-FR-AM-03</b>
<b>Use case name</b>	Account Verification
<b>Description</b>	The system allows users to verify their account via email verification link
<b>Actor</b>	Patient
<b>Pre-conditions</b>	Verification email has been sent
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. User receives a verification email</li> <li>2. User clicks the verification link</li> <li>3. System validates the token</li> <li>4. System verifies token type , user type , expiration , and usage</li> <li>5. System verifies the users email</li> <li>6. System activates the account</li> <li>7. System returns a success message</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Invalid Token</b></p> <ul style="list-style-type: none"> <li>- At step 5 , if the token is not found in the database</li> <li>- System returns error : "Invalid or expired verification token"</li> </ul> <p><b>A2: Token already used</b></p> <ul style="list-style-type: none"> <li>- At step 9 , if the token is already marked used</li> <li>- System return error :"Verification token has already been used"</li> </ul>
<b>Post condition</b>	User account status is changed to ACTIVE

Table 10 , Use Case Specification ( Account Verification )

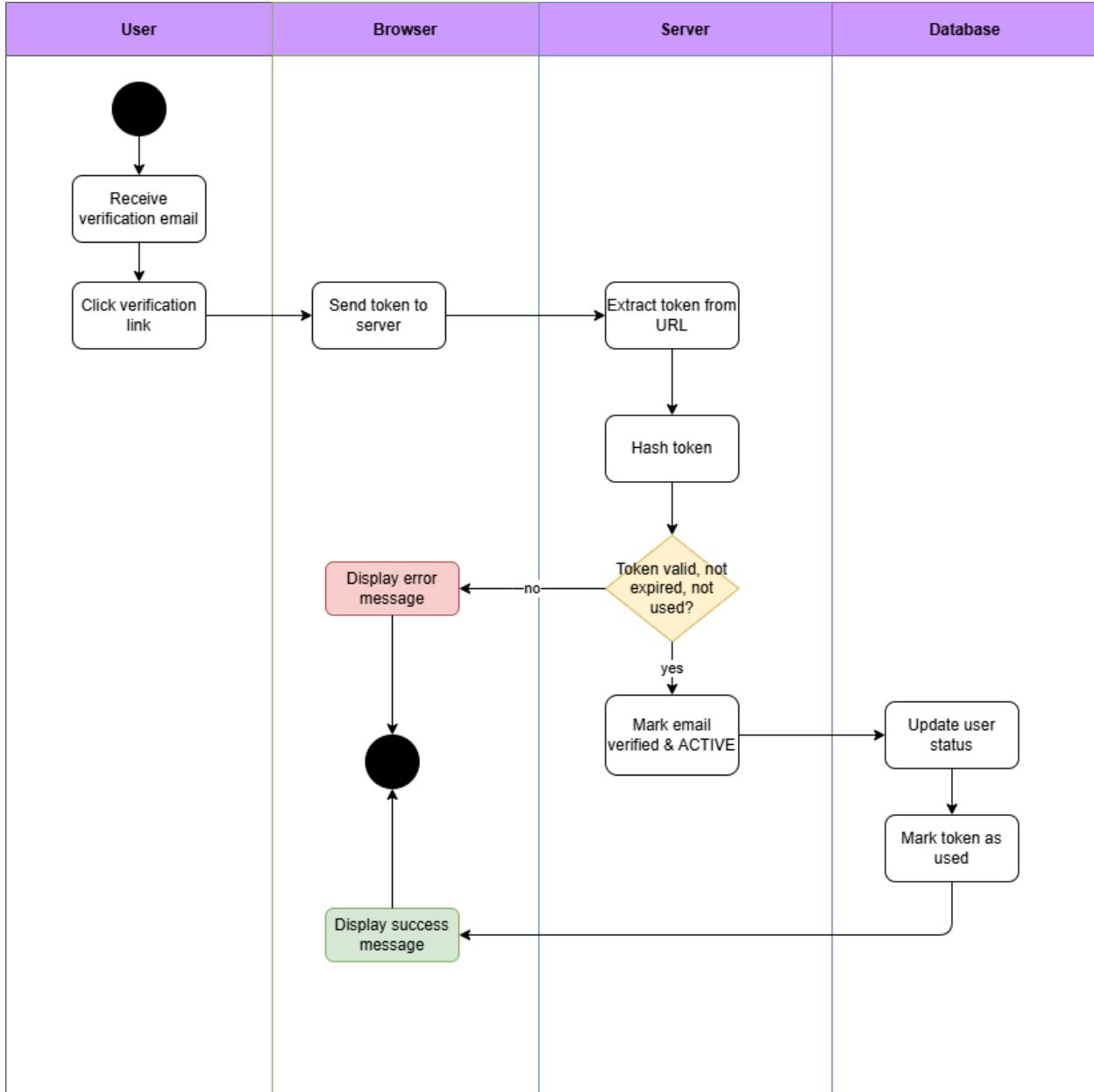


Figure 9, Activity diagram ( Account Verification )

## Chapter 4 - System Analysis

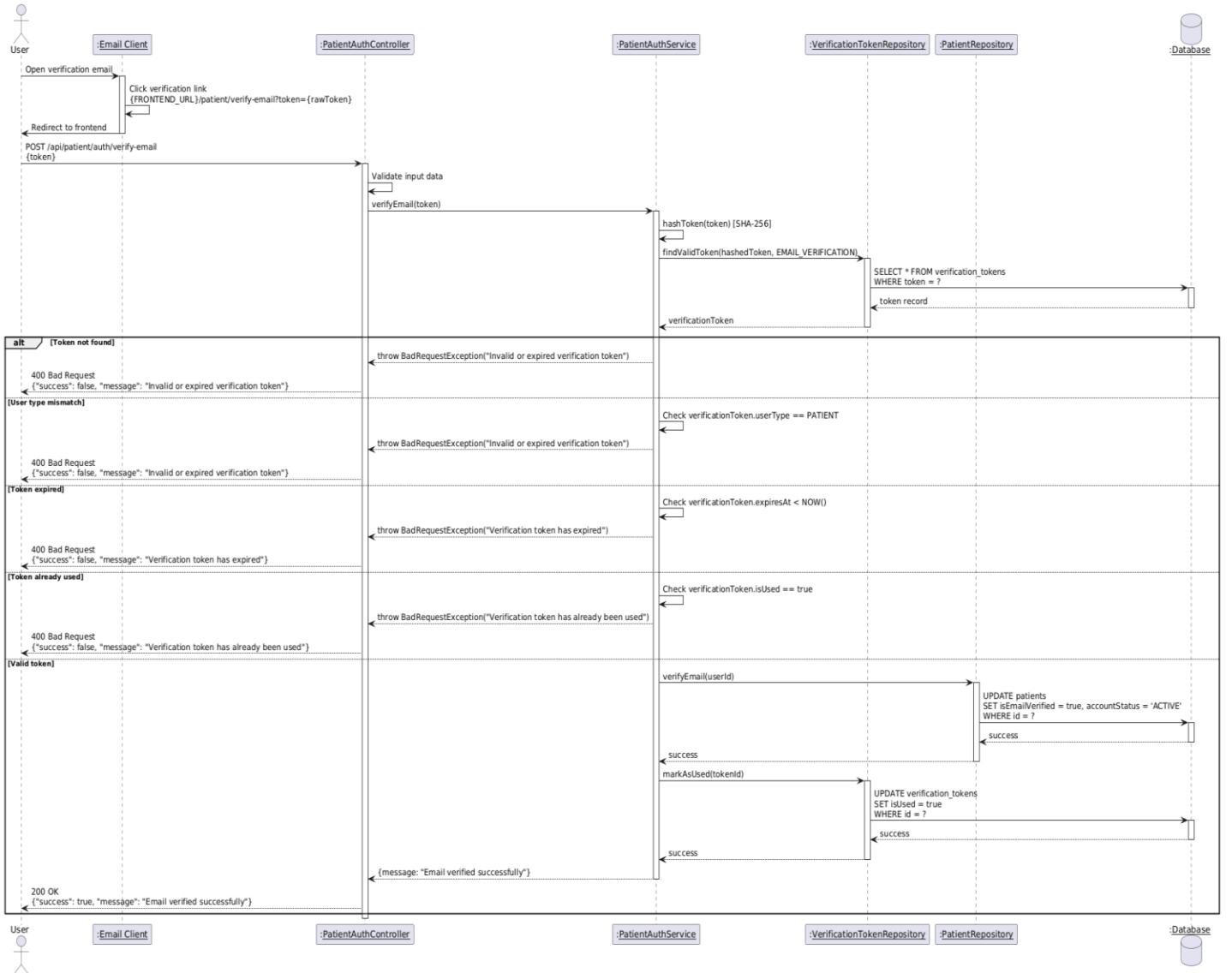


Figure 10, Sequence Diagram ( Account Verification )

## ● Reset Password :

<b>Use case ID</b>	<b>VEMR-FR-AM-04</b>
<b>Use case name</b>	Reset password
<b>Description</b>	The system allows users to reset their password using an email link
<b>Actor</b>	Patient
<b>Pre-conditions</b>	Users has a registered account
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. User opens the Forgot Password page</li> <li>2. Users enter their email address</li> <li>3. System sends a password reset link to the email</li> <li>4. User opens the reset link</li> <li>5. User enters a new password</li> <li>6. System saves the new password and shows a success message</li> </ol>
<b>Alternative scenario</b>	<p><b>A1:Email Not Found</b></p> <ul style="list-style-type: none"> <li>- At step 3, if the email is not found in the database</li> <li>- System return error: "Invalid or expired verification token"</li> </ul> <p><b>A2:Token Already used</b></p> <ul style="list-style-type: none"> <li>- At step 5, if the token is already marked as used</li> <li>- System return error : "Verification token has already been used"</li> </ul>
<b>Post condition</b>	User password is updated

Table 11 , Use Case Specification (Reset Password )

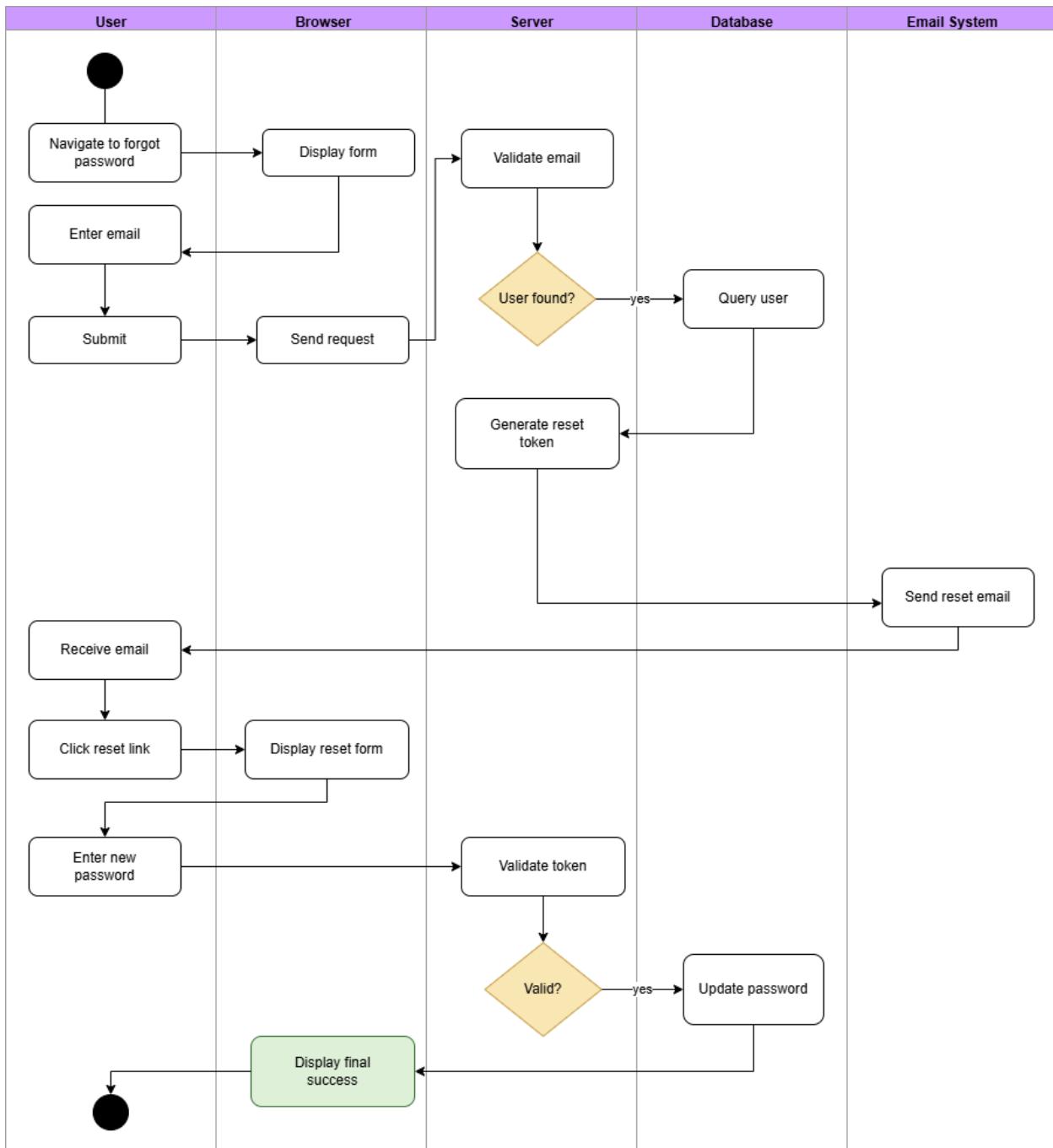


Figure 11, Activity diagram ( Reset Password )

## Chapter 4 - System Analysis

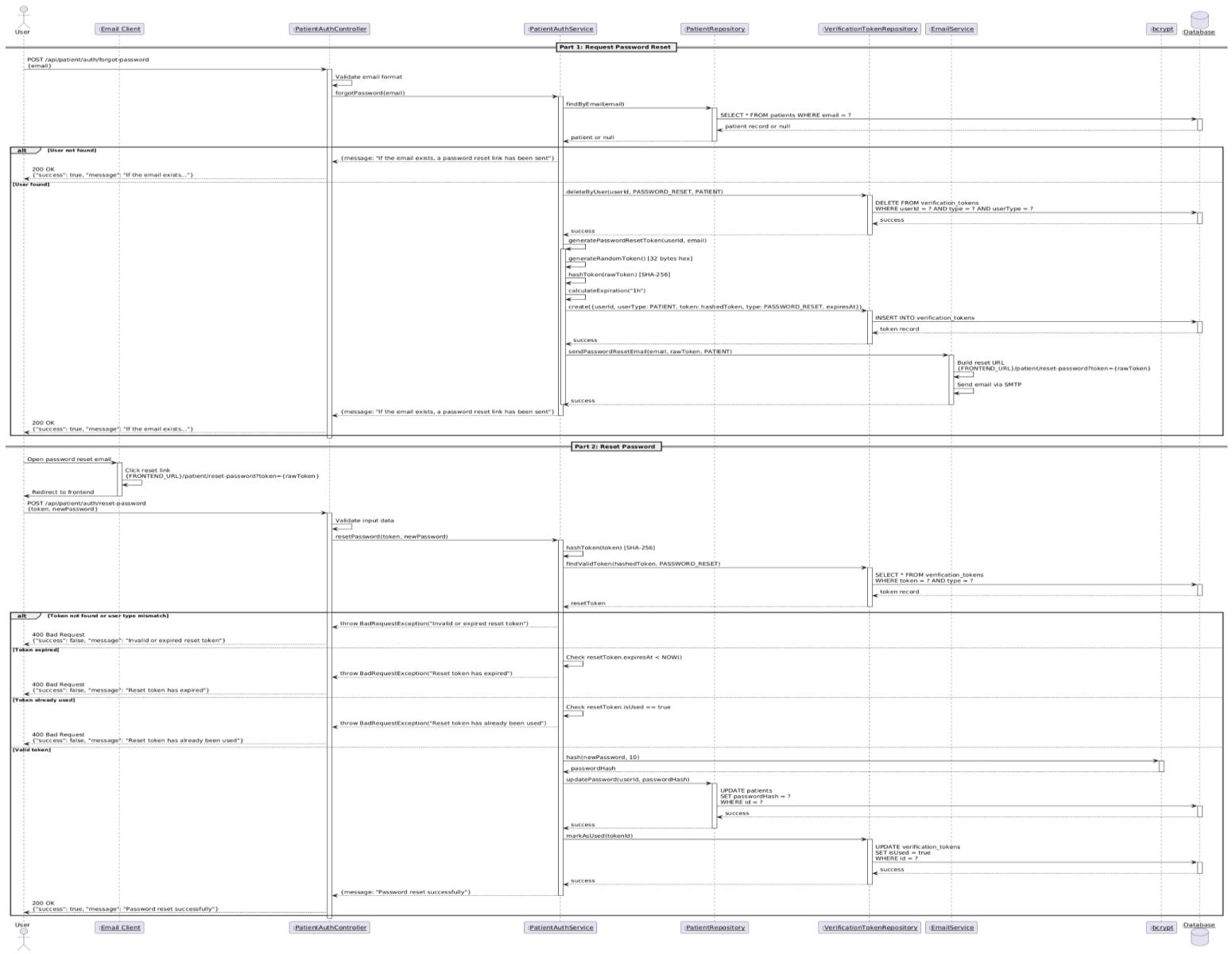


Figure 12, Sequence Diagram ( Reset Password )

- **View Profile :**

<b>Use case ID</b>	<b>VEMR-FR-AM-05</b>
<b>Use case name</b>	View profile
<b>Description</b>	The system allows users to view profile data.
<b>Actor</b>	Patient
<b>Pre-conditions</b>	User is logged in
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. User navigates to profile page</li> <li>2. System extracts JWT token from request header.</li> <li>3. System validate JWT token</li> <li>4. System extract user ID from token payload</li> <li>5. System retrieves user profile from database by ID</li> <li>6. System returns user profile information</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Invalid token</b></p> <ul style="list-style-type: none"> <li>- At step 3, if token is invalid or expired</li> <li>- System returns 401 Unauthorized error</li> </ul> <p><b>A2:User not found</b></p> <ul style="list-style-type: none"> <li>- At step 5,if user ID does not exist</li> <li>- System returns 404 Not Found error</li> </ul>
<b>Post condition</b>	Users profile information is displayed

Table 12, Use Case Specification ( View Profile )

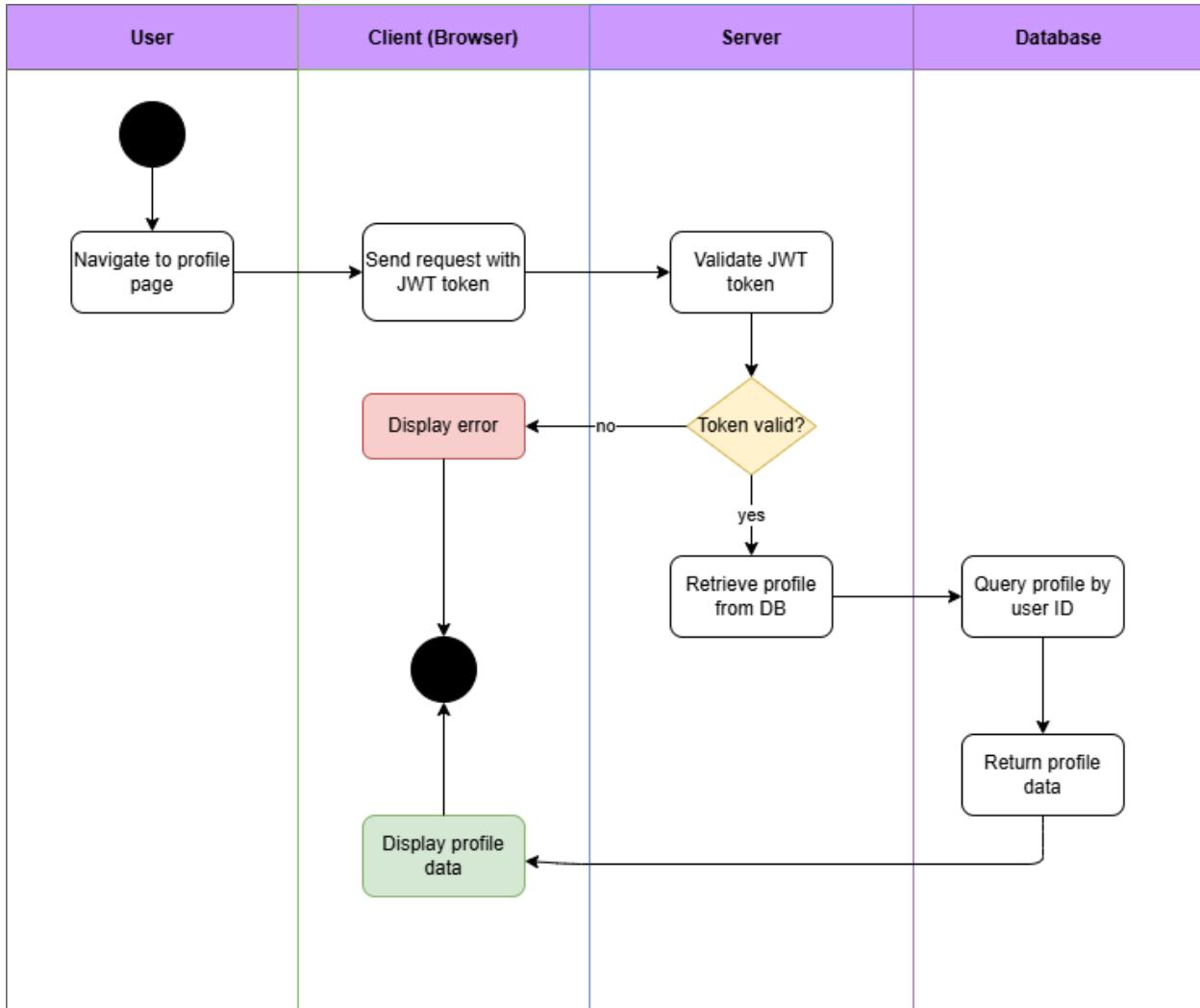


Figure 13, Activity Diagram ( View Profile )

## Chapter 4 - System Analysis

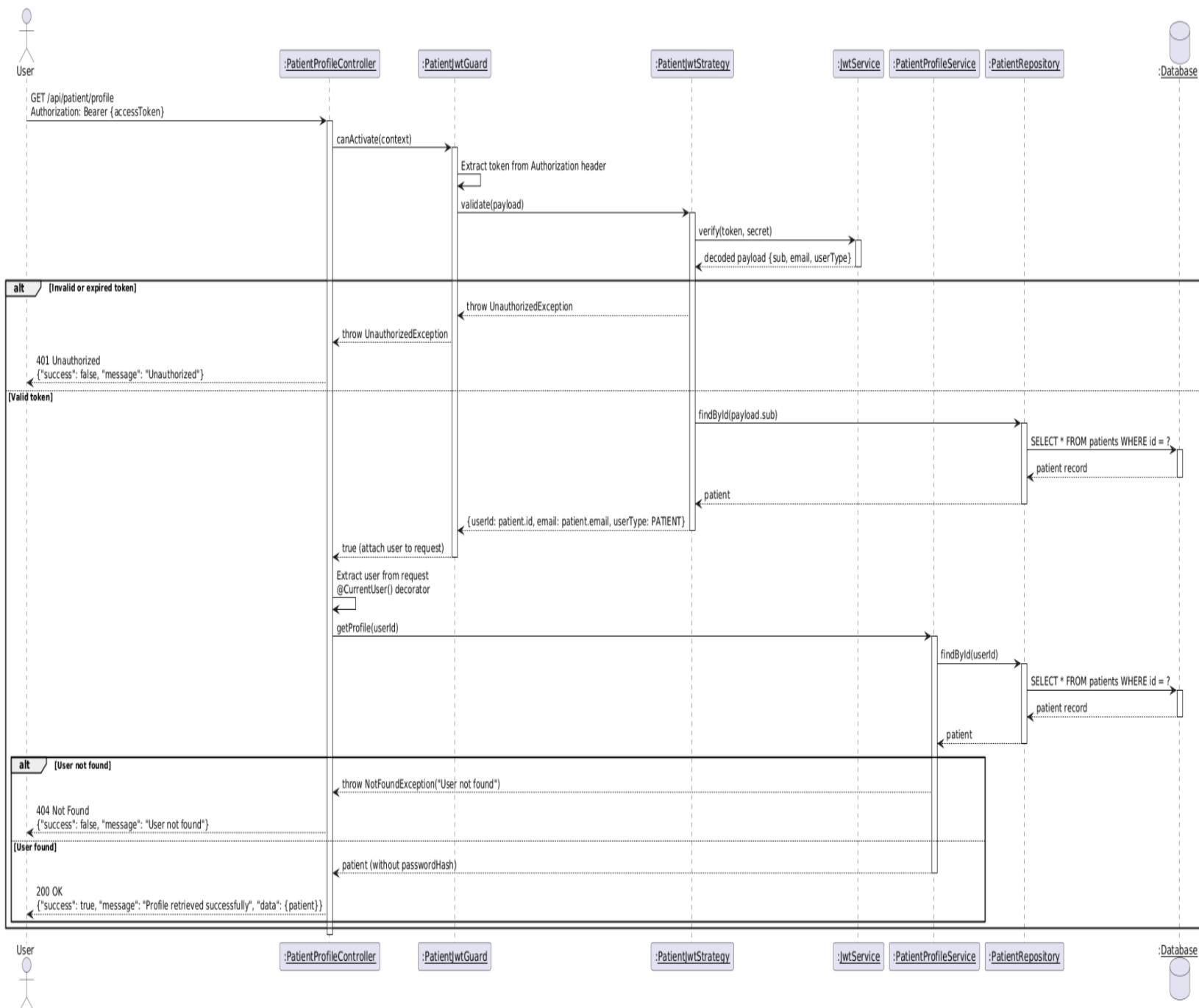


Figure 14, Sequence Diagram ( View Profile )

## ● Edit Profile

<b>Use case ID</b>	<b>VEMR-FR-AM-06</b>
<b>Use case name</b>	Edit profile
<b>Description</b>	The system allows users to update their profile information
<b>Actor</b>	Patient
<b>Pre-conditions</b>	User us logged in
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. User navigates to edit profile page</li> <li>2. User views current profile information</li> <li>3. User modifies desired fields</li> <li>4. User submits the update form</li> <li>5. System extracts JWT token from request header</li> <li>6. System validates JWT token</li> <li>7. System extracts user ID from token payload</li> <li>8. System validates the input data</li> <li>9. System updates user profile in database</li> <li>10. System retrieves updated profile</li> <li>11. System return updated profile information</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Invalid Token</b></p> <ul style="list-style-type: none"> <li>- At step 6, if token is invalid or expired</li> <li>- System return 401 Unauthorized error</li> </ul> <p><b>A2: Invalid Input Data</b></p> <ul style="list-style-type: none"> <li>- At step 8, if validation fails</li> <li>- System return validation error</li> </ul> <p><b>A3: User Not Found</b></p> <ul style="list-style-type: none"> <li>- At step 9, if user ID does not exist</li> <li>- System return 404 Not Found error</li> </ul>
<b>Post condition</b>	User profile information is updated in the database

Table 13, Use Case Specification ( Edit Profile )

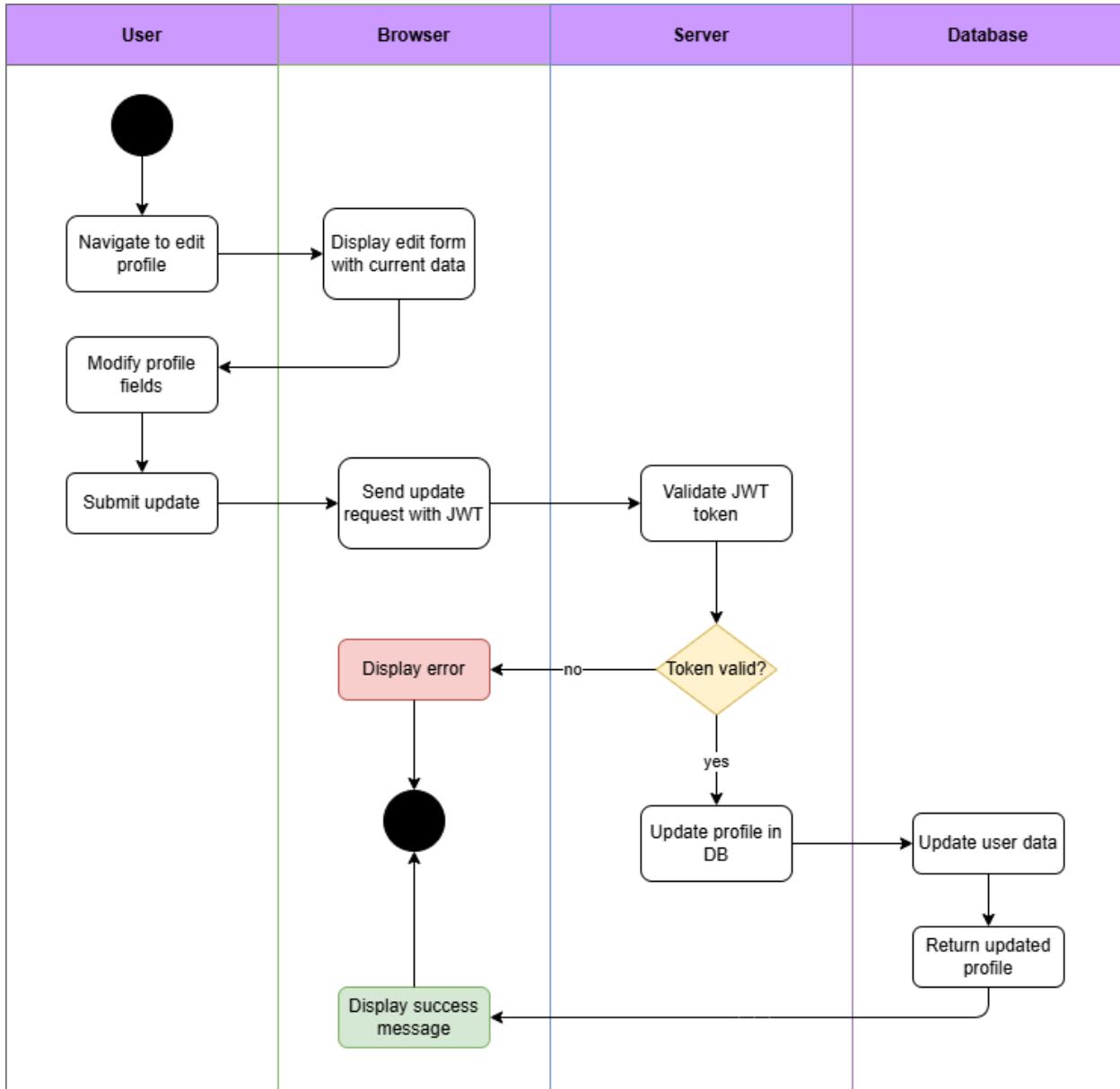


Figure 15, Activity Diagram ( Edit Profile )

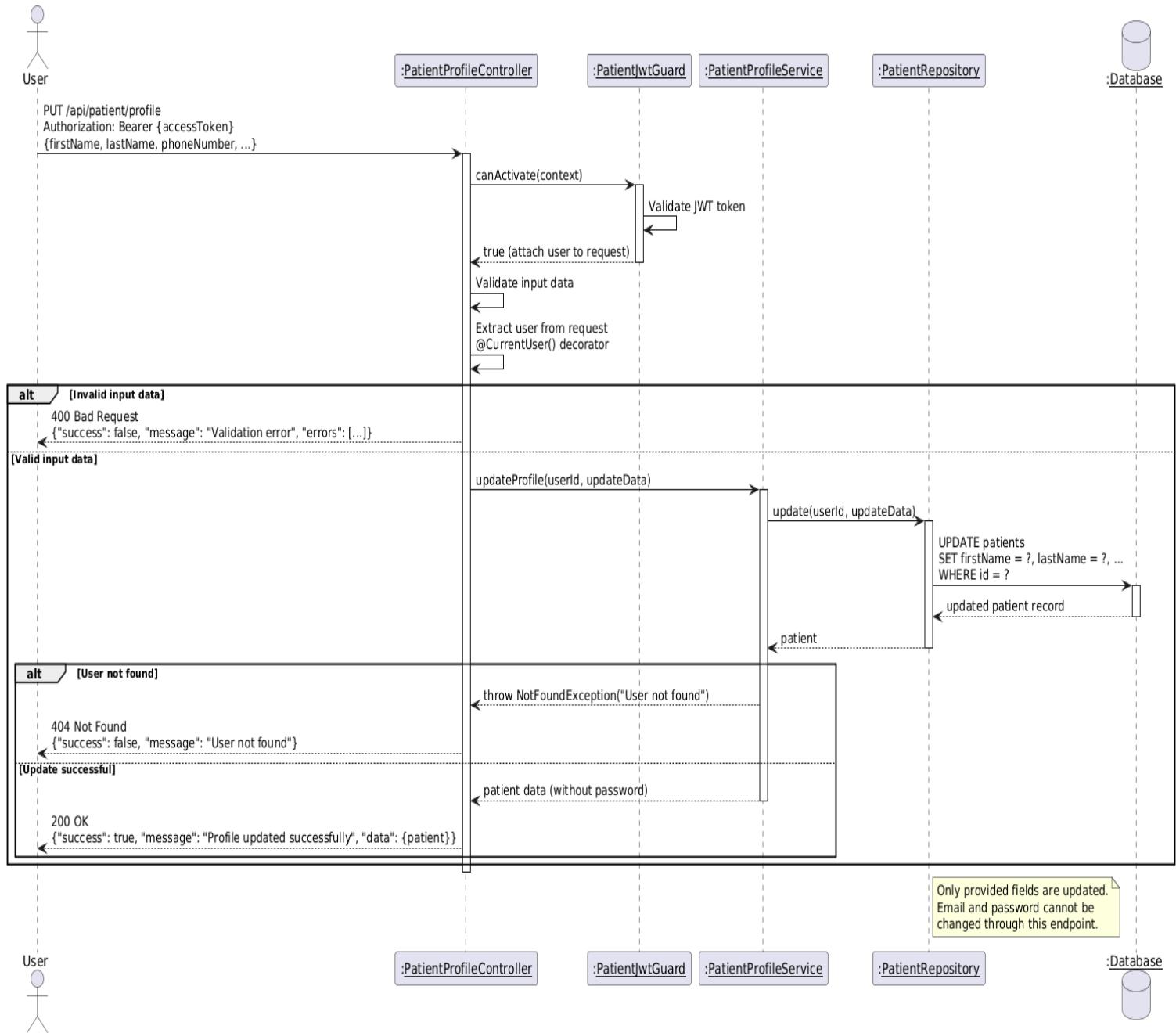


Figure 16, Sequence Diagram ( Edit Profile )

- Change Password :

<b>Use case ID</b>	<b>VEMR-FR-AM-07</b>
<b>Use case name</b>	Change Password
<b>Description</b>	The system allows users to change their password
<b>From</b>	Patient
<b>Pre-conditions</b>	User is logged in
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. User navigates to change password page</li> <li>2. User enters current password then new password</li> <li>3. User confirms new password</li> <li>4. User submits the form</li> <li>5. System extracts JWT token from request header and validate it</li> <li>6. System extracts user ID from token payload</li> <li>7. System validates input data</li> <li>8. System retrieves user profile from data</li> <li>9. System verifies current password</li> <li>10. System hashes new password</li> <li>11. System updates password in database</li> <li>12. System returns success message</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Invalid Token</b></p> <ul style="list-style-type: none"> <li>- At step 5, if token is invalid or expired</li> <li>- System returns 401 Unauthorized error</li> </ul> <p><b>A2: Invalid Input Data</b></p> <ul style="list-style-type: none"> <li>- At step 7, if validation fails</li> <li>- System returns validation errors</li> </ul> <p><b>A3: Incorrect Current Password</b></p> <ul style="list-style-type: none"> <li>- At step 9, if current password does not match</li> <li>- System returns error : "Current password is incorrect"</li> </ul>
<b>Post condition</b>	User password is updated in the database

Table 14, Use Case Specification (Change Password )

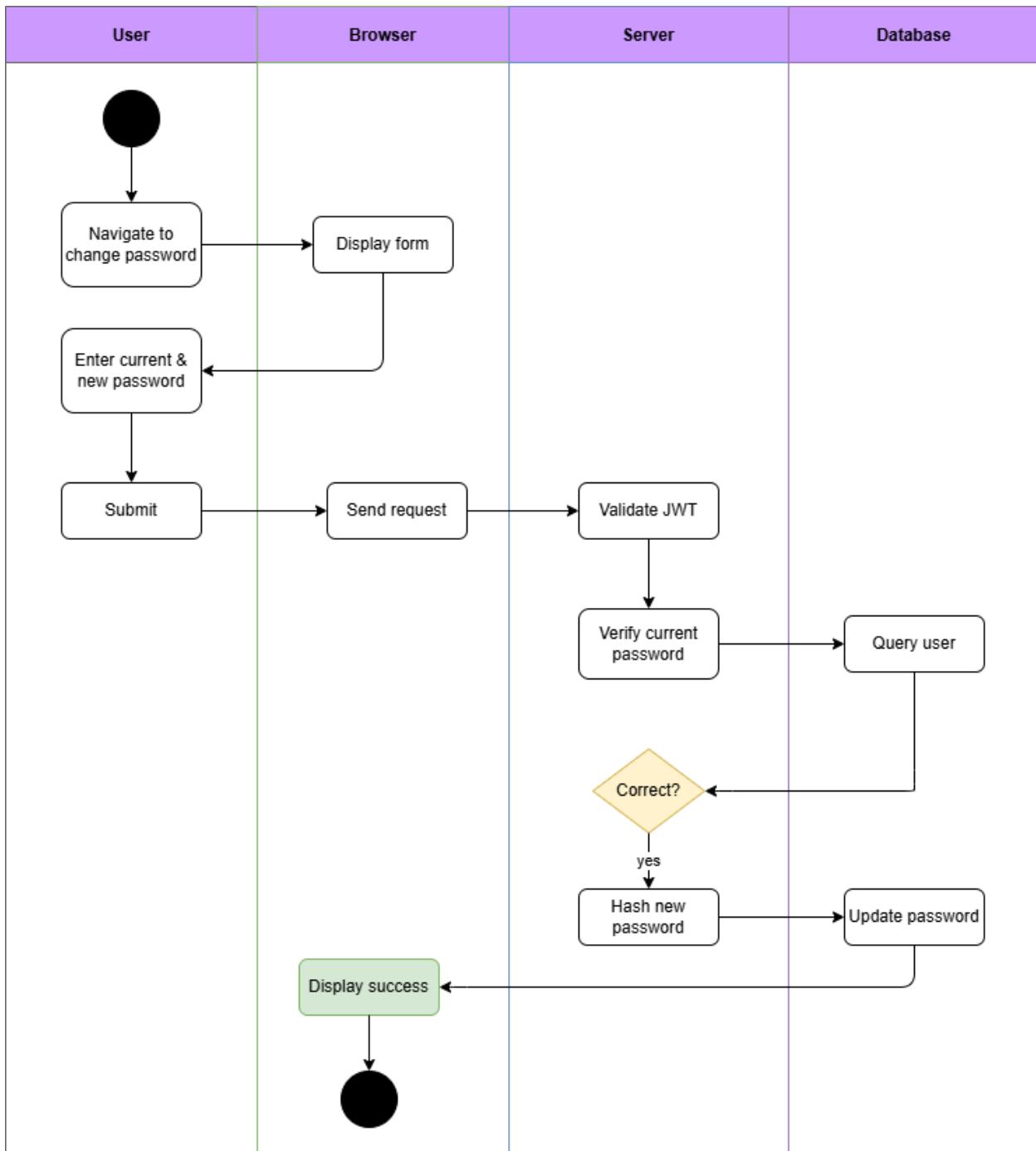


Diagram 17, Activity Diagram ( Change Password )

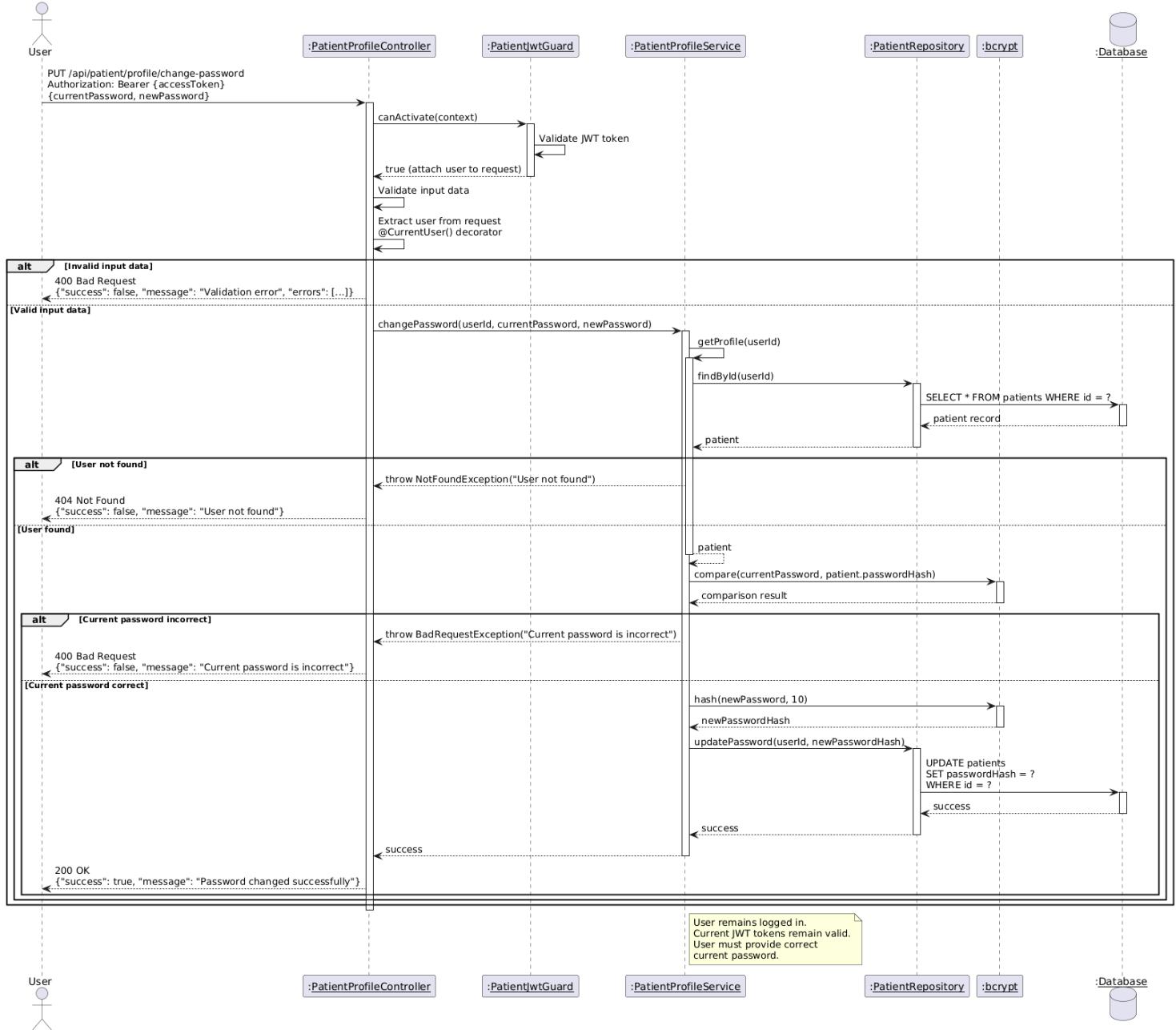


Figure 18, Sequence Diagram ( Change Password )

## ● Admin Login :

<b>Use case ID</b>	<b>VEMR-FR-AU-08</b>
<b>Use case name</b>	Admin Login
<b>Description</b>	The system allows an administrator to log in using an email and password to access the admin dashboard.
<b>Actor</b>	Admin
<b>Pre-conditions</b>	Administrator account exists
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. Administrator navigates to the admin login page.</li> <li>2. Administrator enters email and password.</li> <li>3. Administrator clicks the Login button.</li> <li>4. System validates the input data.</li> <li>5. System verifies the credentials and generates a JWT token.</li> <li>6. System stores the JWT token and redirects the administrator to the admin dashboard.</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Invalid Credentials</b></p> <ul style="list-style-type: none"> <li>- At step 5 , If the email or password is incorrect</li> <li>- System displays "Invalid email or password"</li> </ul> <p><b>A2: Validation Error</b></p> <ul style="list-style-type: none"> <li>- At step 4, If the email format is invalid or the password is empty</li> <li>- System displays a validation error message</li> </ul> <p><b>A3: Already Logged In</b></p> <ul style="list-style-type: none"> <li>- At step 1 ,If a valid JWT token already exists</li> <li>- System redirects the administrator directly to the admin dashboard</li> </ul>
<b>Post condition</b>	Administrator is authenticated

Table 15 , Use Case Specification ( Admin Login )

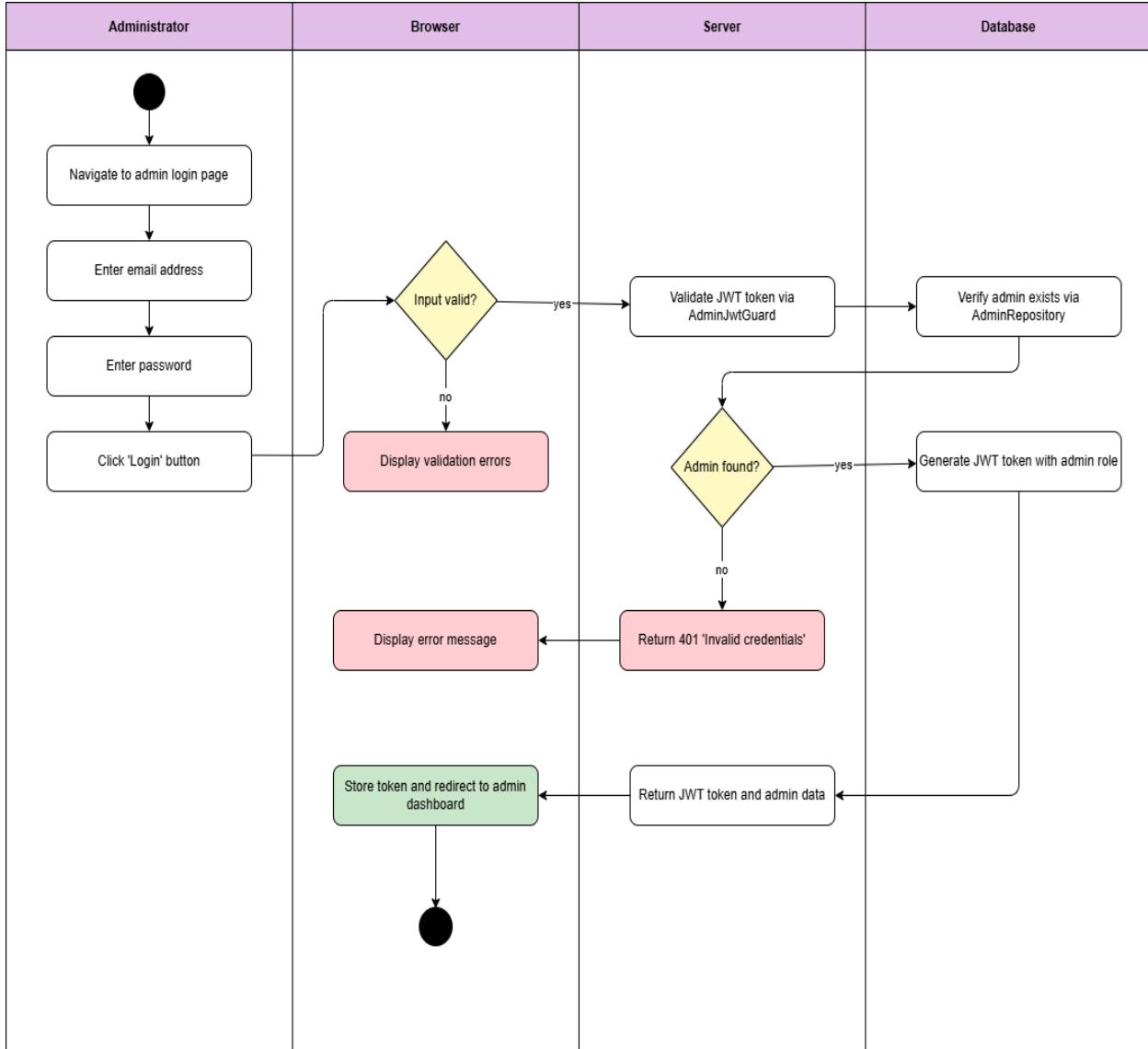


Diagram 19, Activity Diagram ( Admin Login )

## Chapter 4 - System Analysis

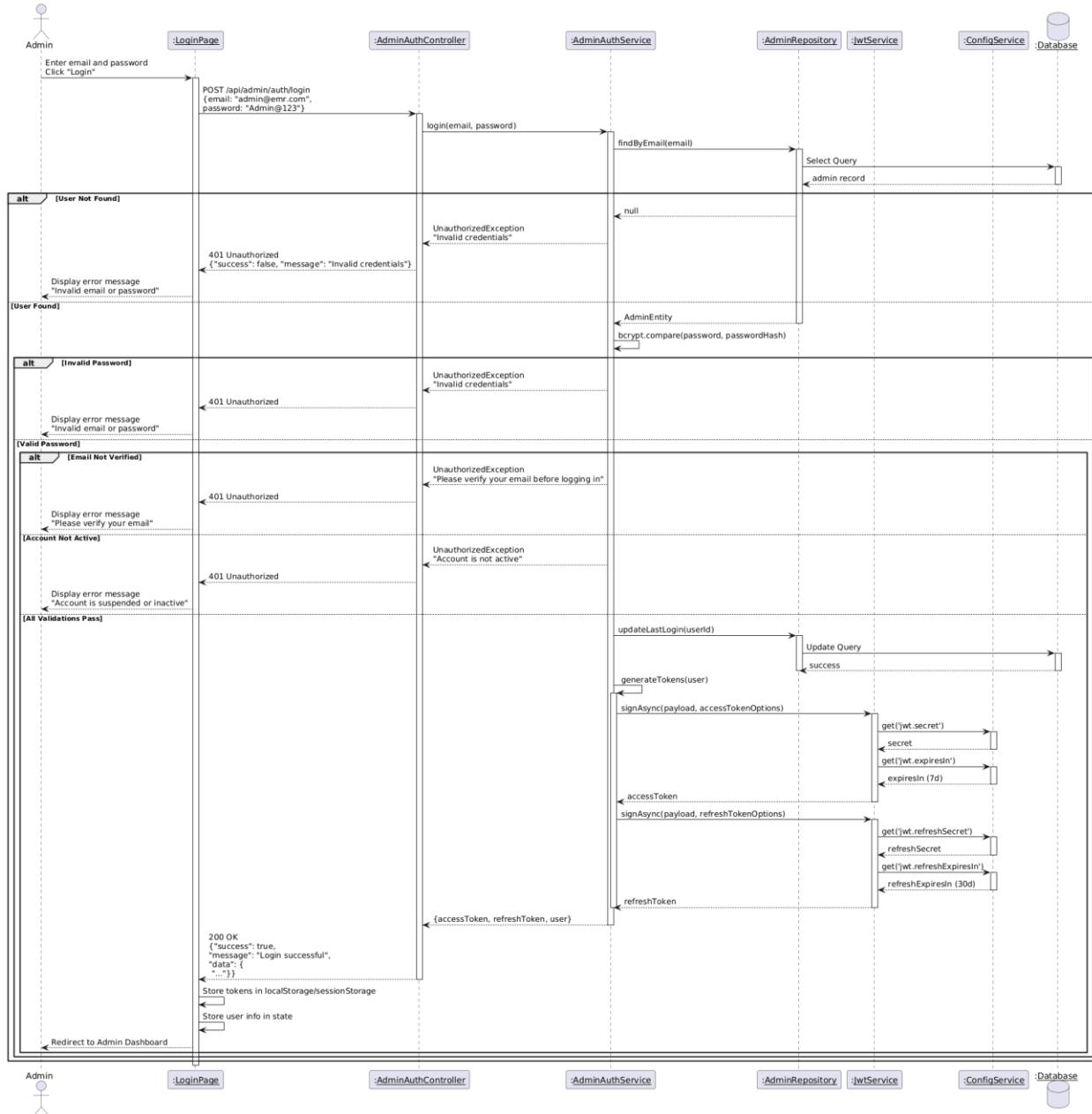


Diagram 20, Sequence Diagram ( Admin Login )

## ● Clinician Login :

<b>Use case ID</b>	<b>VEMR-FR-AU-09</b>
<b>Use case name</b>	ClinicianLogin
<b>Description</b>	The system allows an administrator to log in using an email and password to access the Clinician dashboard.
<b>Actor</b>	Clinician
<b>Pre-conditions</b>	Clinician account exists in the system
<b>Main scenario</b>	<p>1.Clinician navigates to clinician login page      2.Clinician enters email address and password.      3. Clinician clicks "Login" button.      4. System validates the input data.      5.System verifies the credentials and generates a JWT token.      6.System stores the JWT token and redirects the administrator to the Clinician dashboard.</p>
<b>Alternative scenario</b>	<p><b>A1: Invalid Input</b>        - At step 4 , if the email format is invalid or the password is empty        -System displays a validation error message</p> <p><b>A2: Account Not Found</b>        - At step 5, if the email or password is incorrect, or the account does not exist        -System displays "Invalid email or password"</p> <p><b>A3: Already Logged In</b>        - At step 1 ,If a valid JWT token already exists        -System redirects the clinician directly to the clinician dashboard</p>
<b>Post condition</b>	Administrator is authenticated

Table 16 , Use Case Specification ( Clinician Login )

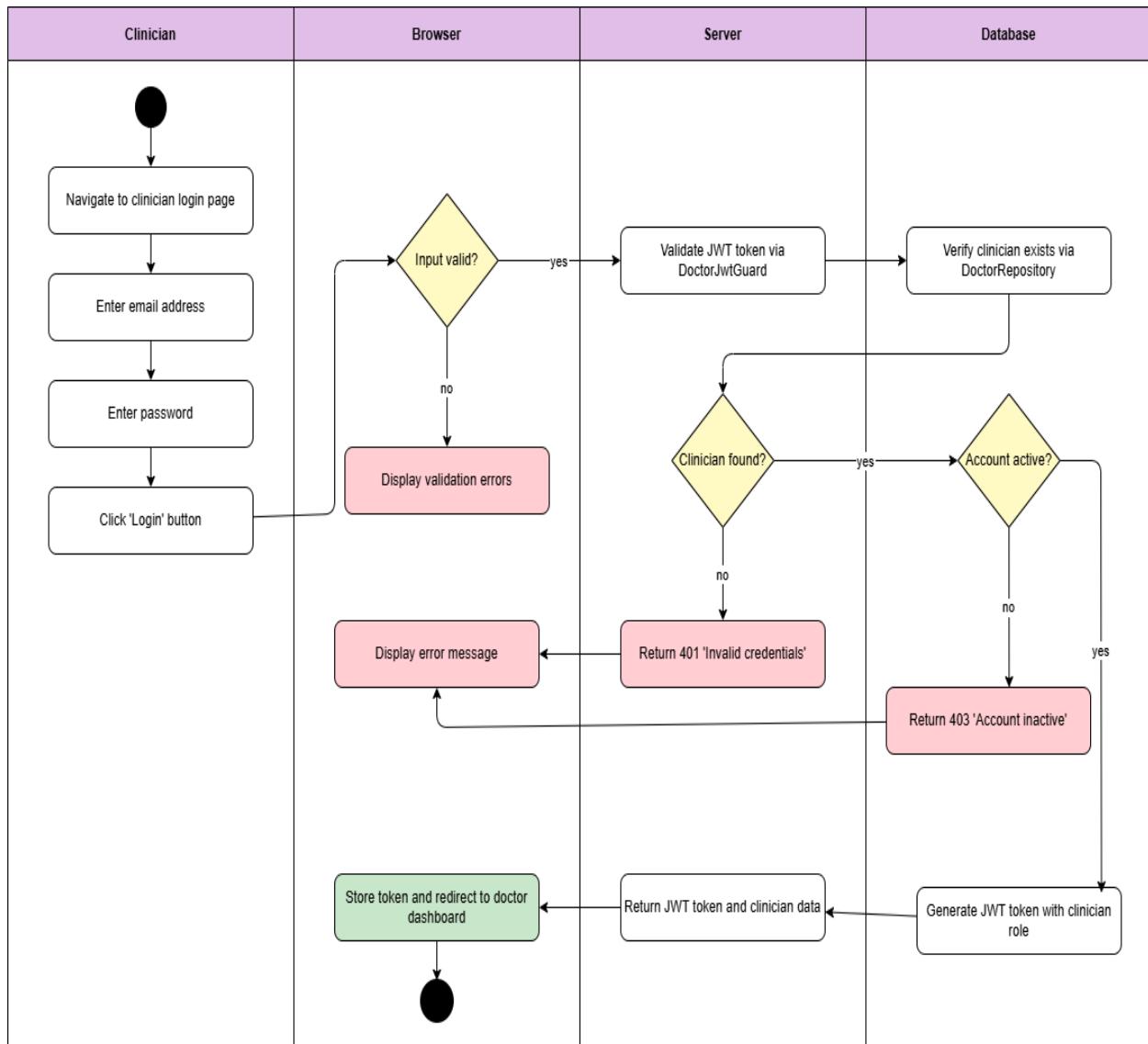


Diagram 21, Activity Diagram ( Clinician Login )

## Chapter 4 - System Analysis

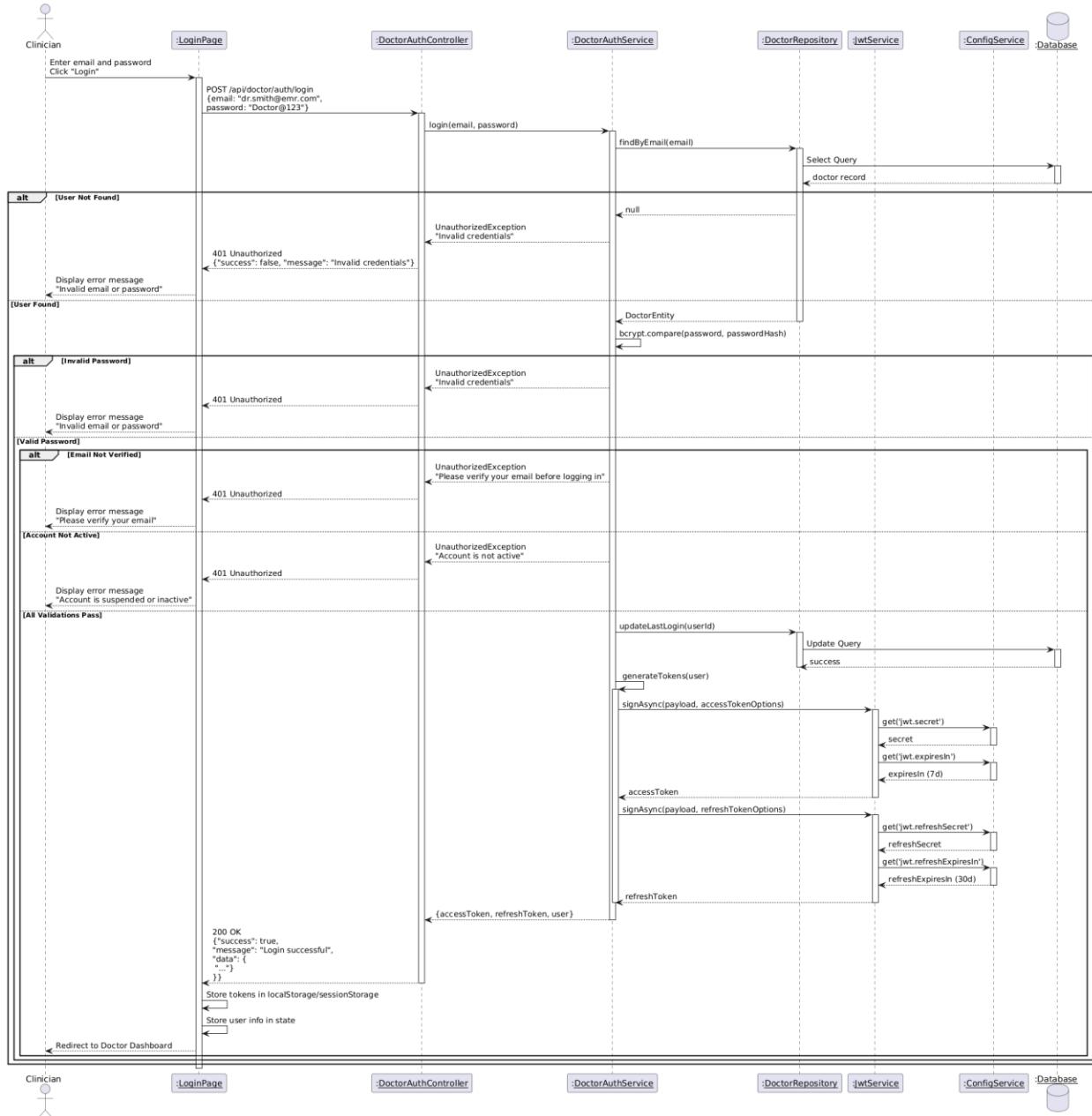


Diagram 22, Sequence Diagram ( Clinician Login )

- **Create New Visit :**

<b>Use case ID</b>	<b>VEMR-FR-AM-10</b>
<b>Use case name</b>	Create New Visit
<b>Description</b>	The system allows clinicians to create a new visit
<b>From</b>	Clinician
<b>Pre-conditions</b>	Clinician is authenticated
<b>Main scenario</b>	<p>1.clinician click “new visit” button from visit page      2.System displays EncounterForm model      3.Clinician selects patient and fills required field      4.Clinican optionally enters reason for visit,chief complaint , location ,and notes (support voice input).      5.Clinician submits the form      6.System validates input,authenticates user,and verifies patient exists.      7.System creates encounter with status “planned”      8.System Displays success message and closes modal</p>
<b>Alternative scenario</b>	<p><b>A1:Validation Error</b></p> <ul style="list-style-type: none"> <li>- At step 6, if required fields are missing or patient not found</li> <li>-System displays error message on form</li> </ul> <p><b>A2: Authentication Error</b></p> <ul style="list-style-type: none"> <li>- At step 7, if JWT token is invalid or expired</li> <li>-System returns existing encounter without creating duplicate</li> </ul>
<b>Post condition</b>	New visit is created with status “planned”

Table 17, Use Case Specification (Create New Visit )

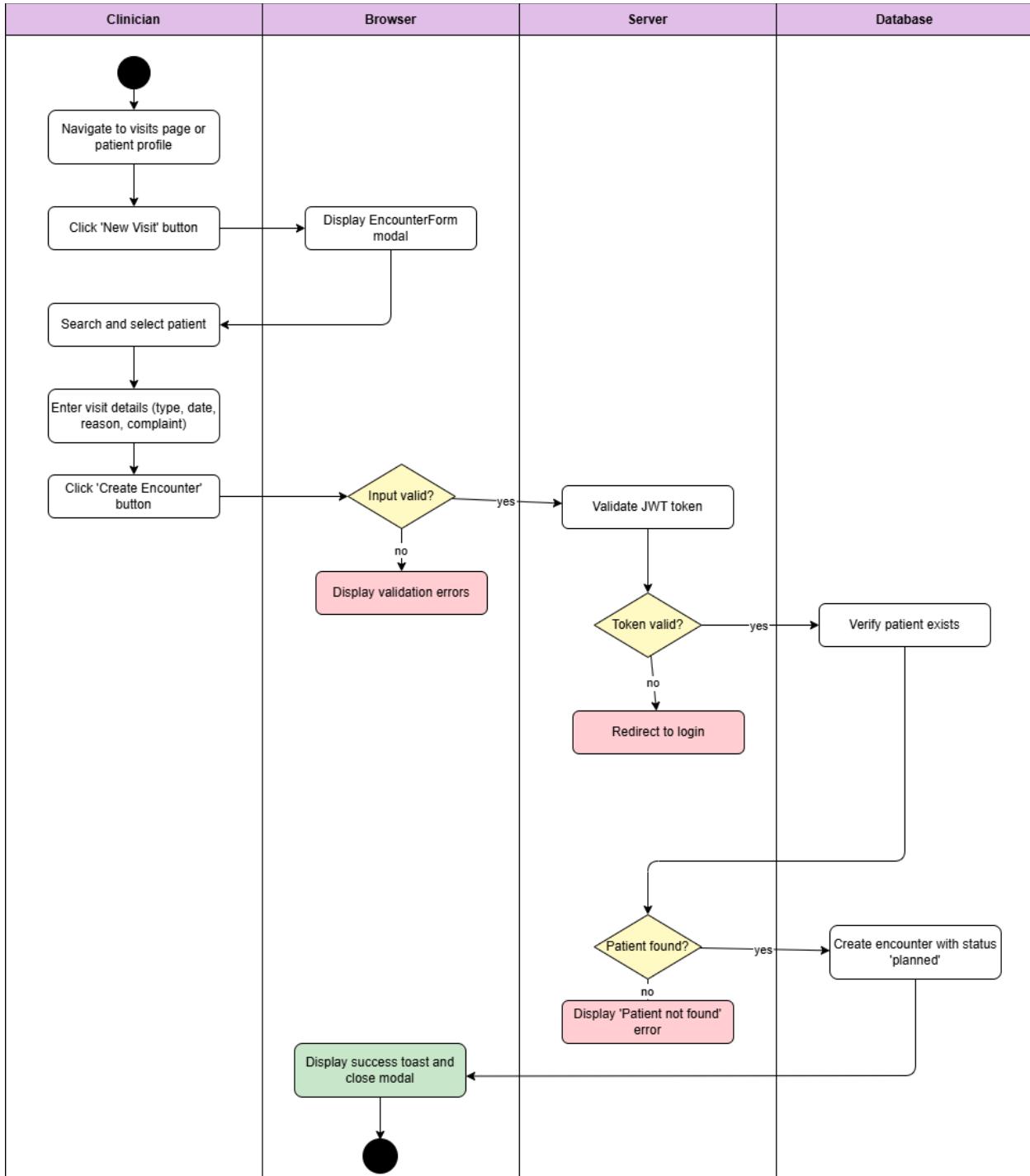


Diagram 23, Activity Diagram ( Create New Visit )

## Chapter 4 - System Analysis

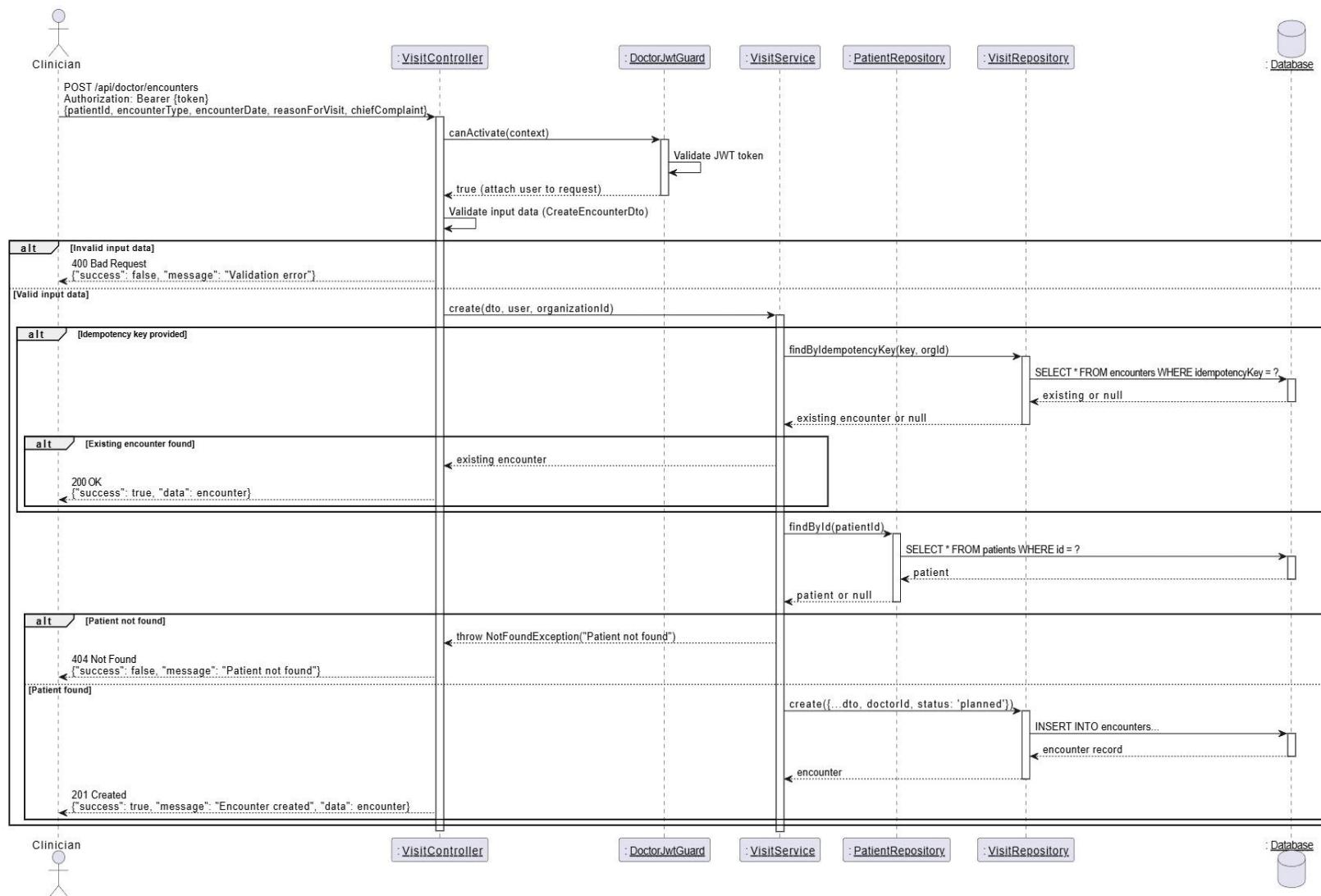


Diagram 24, Sequence Diagram ( Create New Visit)

## ● Search Visit :

<b>Use case ID</b>	<b>VEMR-FR-AM-11</b>
<b>Use case name</b>	Search visit
<b>Description</b>	The system allows clinicians to search for visit
<b>From</b>	Clinician
<b>Pre-conditions</b>	Clinician is authenticated
<b>Main scenario</b>	<p>1.Clinician enters search query in search input field on visits page.</p> <p>2.System authenticated user and searches visits by patient name</p> <p>3.System returns paginated results with matching visits</p> <p>4.System displays search results in visits table</p>
<b>Alternative scenario</b>	<p><b>A1:No Result Found</b></p> <ul style="list-style-type: none"> <li>- At step 3, if no visit match the query</li> <li>-System displays “No visits found” message</li> </ul> <p><b>A2: Authentication Error</b></p> <ul style="list-style-type: none"> <li>- At step 7, if JWT token is invalid or expired</li> <li>-System redirects to login page</li> </ul>
<b>Post condition</b>	Matching visits are displayed in the table

Table 18, Use Case Specification (Search Visit )

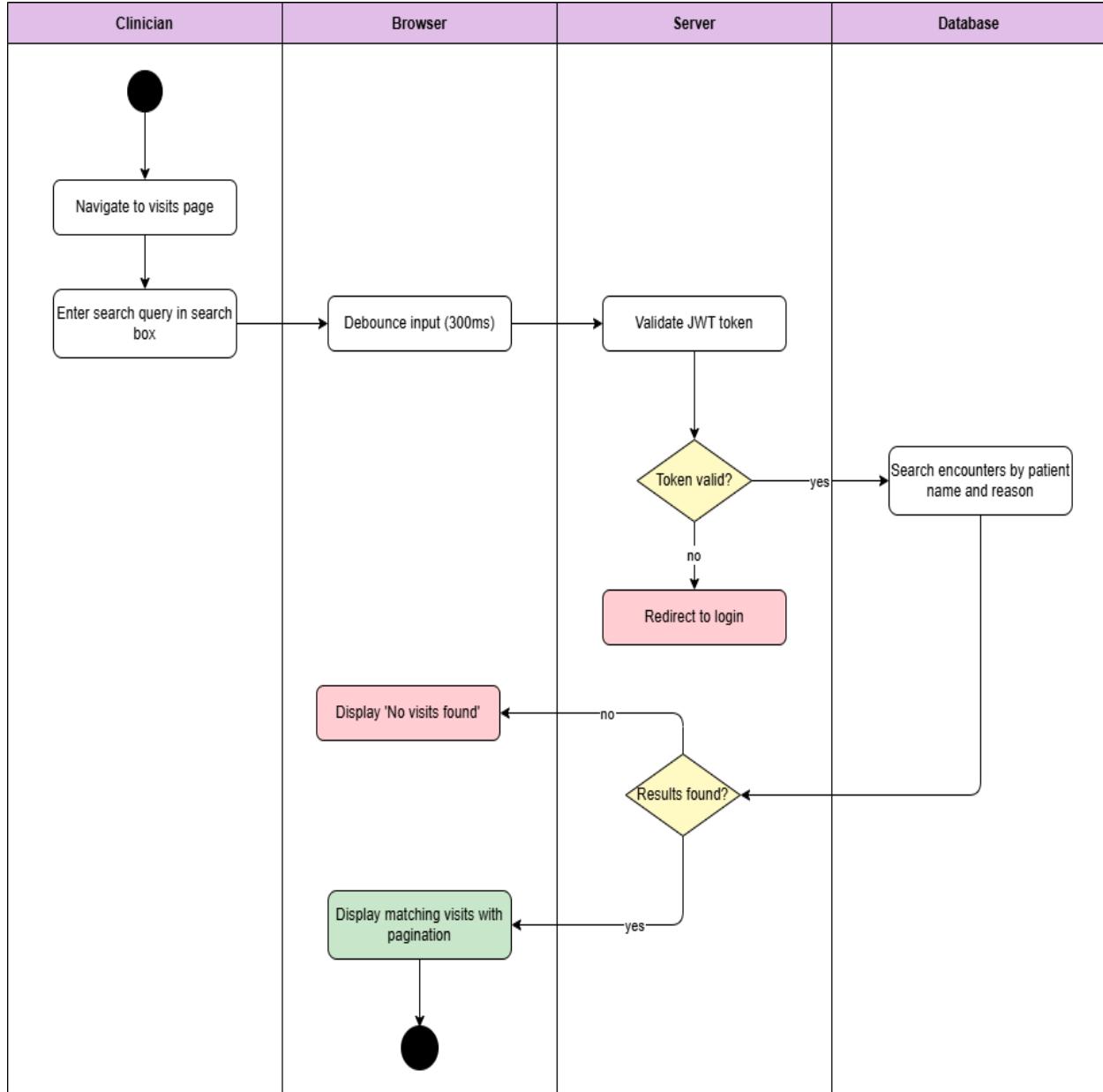


Diagram 25, Activity Diagram ( Search Visit )

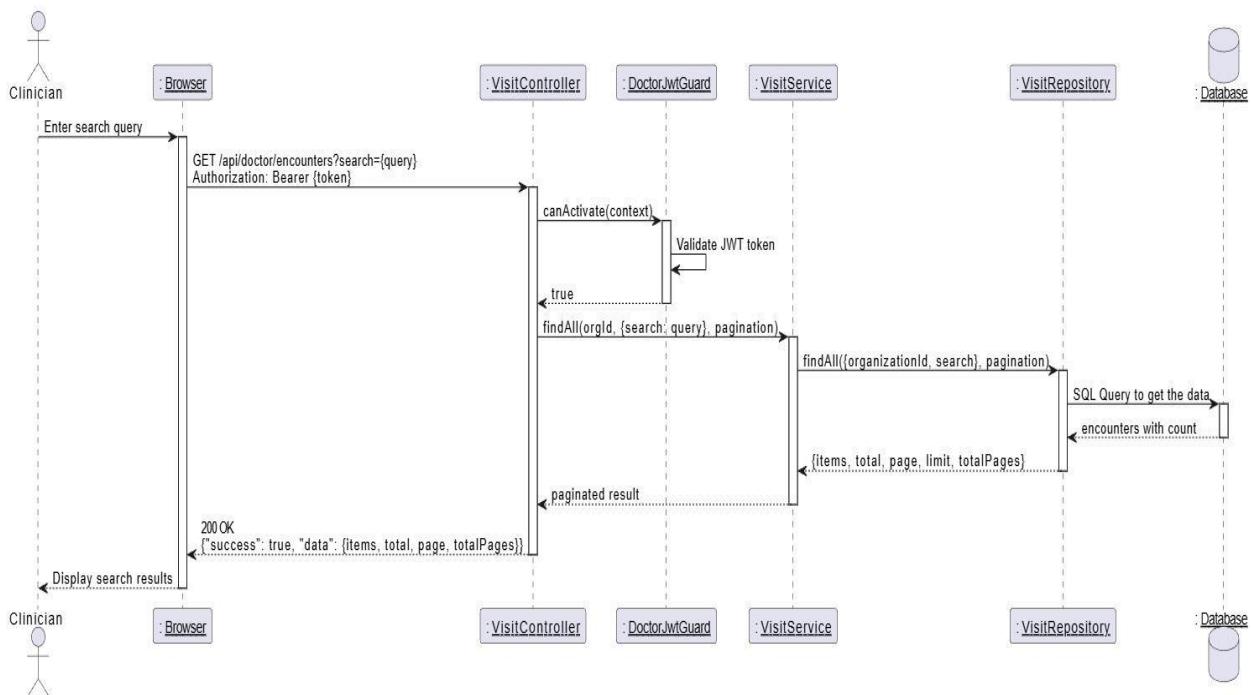


Diagram 26, Sequence Diagram ( Search Visit )

- **Edit Visit :**

<b>Use case ID</b>	<b>VEMR-FR-AM-12</b>
<b>Use case name</b>	Edit visit
<b>Description</b>	The system allows clinicians to edit visit details .
<b>From</b>	Clinician
<b>Pre-conditions</b>	Clinician is authenticated
<b>Main scenario</b>	<p>1.Clinician open visit detail page and clicks edit button</p> <p>2.System updates visit information (type , reason , chief , complaint , notes , location)</p> <p>3.Clinician submits changes</p> <p>4.System validates input , authenticates user , and verifies visit is in progress</p> <p>5.System verifies clinician is the assigned doctor</p> <p>6.System updates encounter in database</p> <p>7.System displays success message</p>
<b>Alternative scenario</b>	<p><b>A1:Visit Not Editable</b></p> <ul style="list-style-type: none"> <li>- At step 4, if visit is not “in progress” or does not exist</li> <li>-System displays error message</li> </ul> <p><b>A2: Authentication Error</b></p> <ul style="list-style-type: none"> <li>- At step 4, if JWT token is invalid or expired</li> <li>-System redirects to login page</li> </ul>
<b>Post condition</b>	Visit is updated with new information

Table 19, Use Case Specification (Edit Visit )

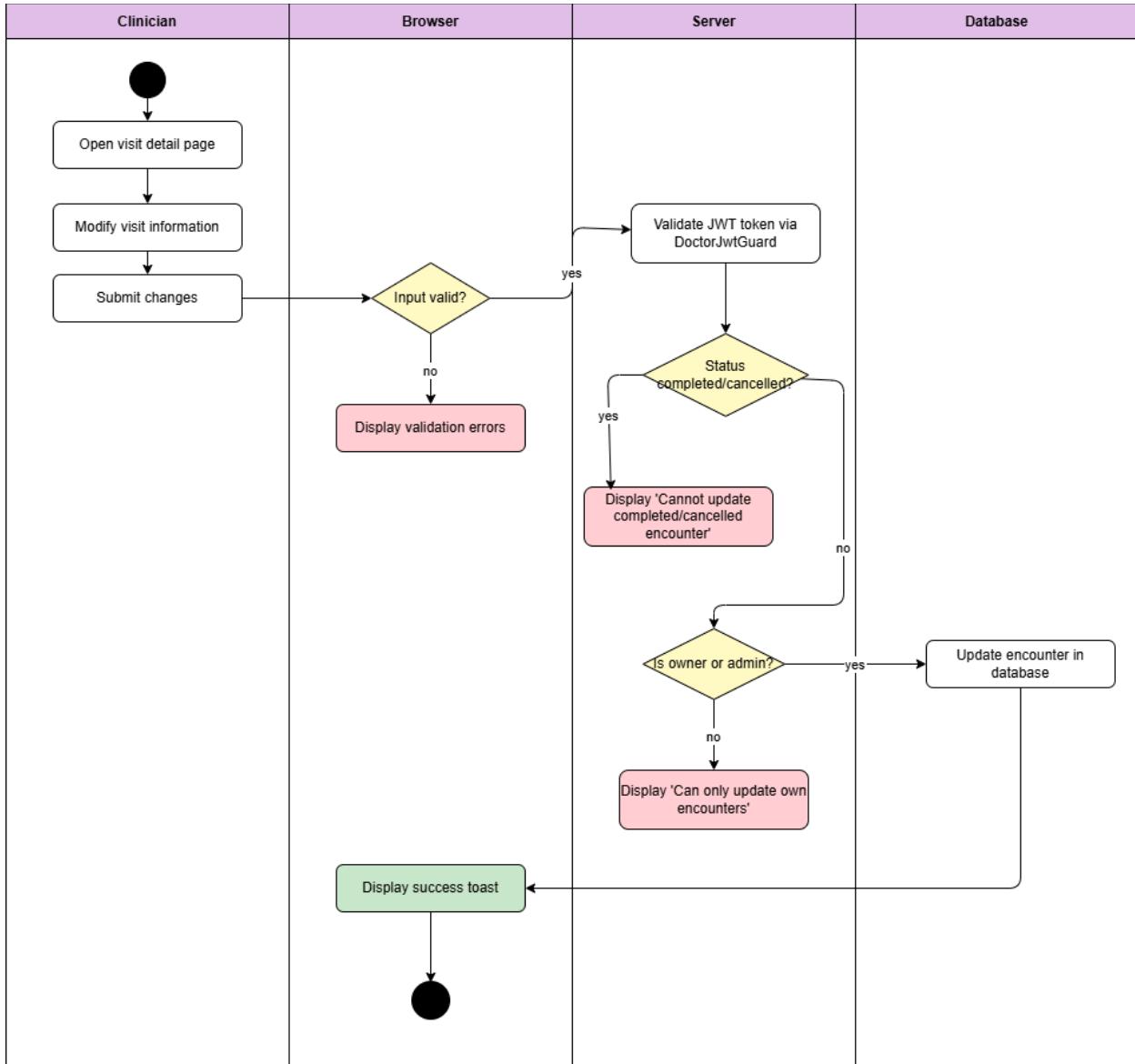


Diagram 27, Activity Diagram ( Edit Visit )

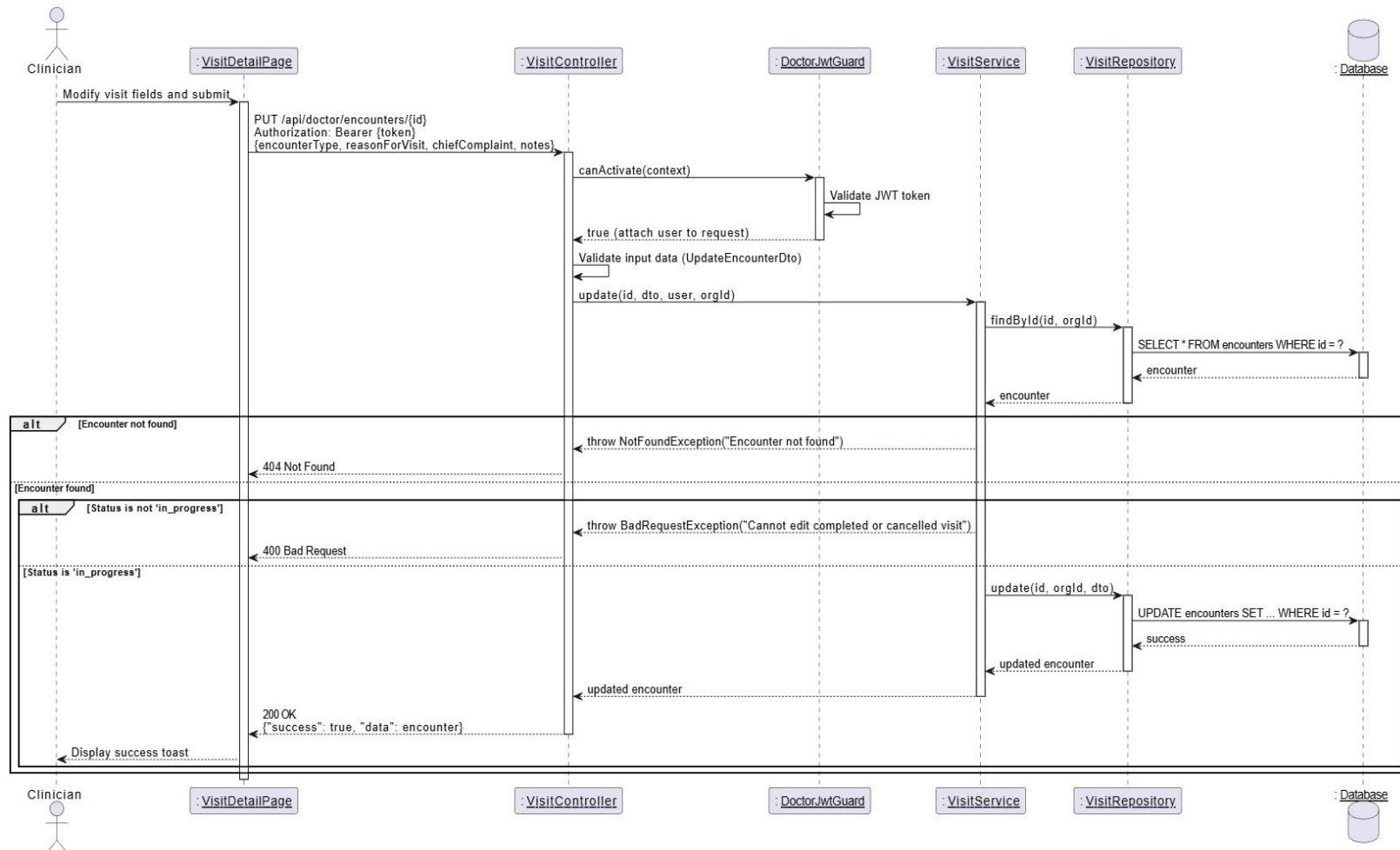


Diagram 28, Sequence Diagram (Edit Visit)

## ● Save Visit :

<b>Use case ID</b>	<b>VEMR-FR-AM-13</b>
<b>Use case name</b>	Save visit
<b>Description</b>	The system allows clinicians to save visit details .
<b>From</b>	Clinician
<b>Pre-conditions</b>	Clinician is authenticated
<b>Main scenario</b>	<p>1.Clinician makes changes to visit or medical record fields</p> <p>2.Clinician click save button</p> <p>3.System validates input , authenticates user .</p> <p>4.System verifies visit clinician is in progress and not finalized</p> <p>5.System updates encounter/medical record in database</p> <p>6.System displays success message.</p>
<b>Alternative scenario</b>	<p><b>A1:Validation or Status Error</b></p> <ul style="list-style-type: none"> <li>- At step 3-4, if validation fails , visit not in progress , or record finalized</li> <li>-System redirects to login page</li> </ul> <p><b>A2: Authentication Error</b></p> <ul style="list-style-type: none"> <li>- At step 4, if JWT token is invalid or expired</li> <li>-System redirects to login page</li> </ul>
<b>Post condition</b>	Visit / medical record changes are persisted

Table 20, Use Case Specification (Save Visit )

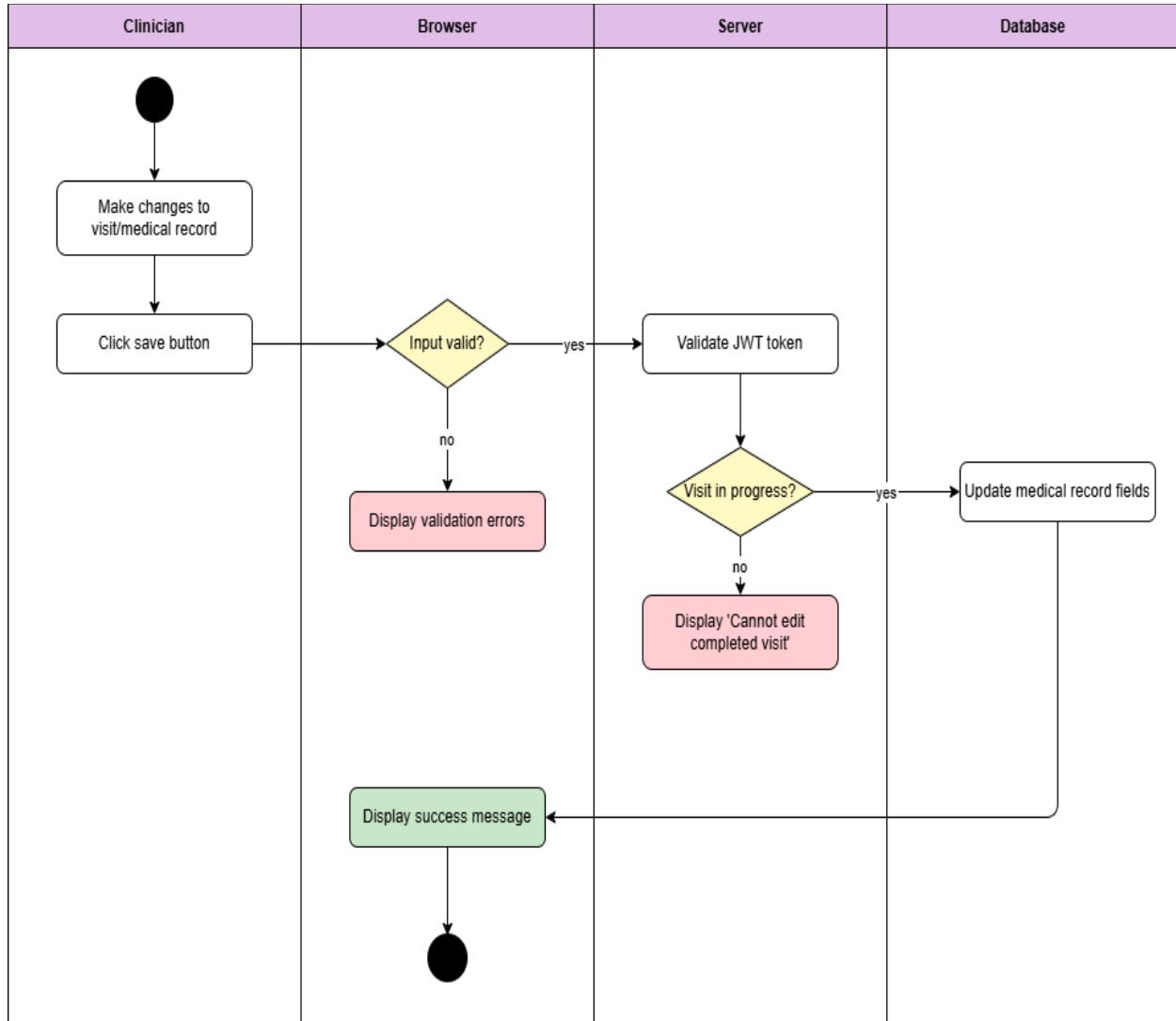


Diagram 29, Activity Diagram ( Save Visit )

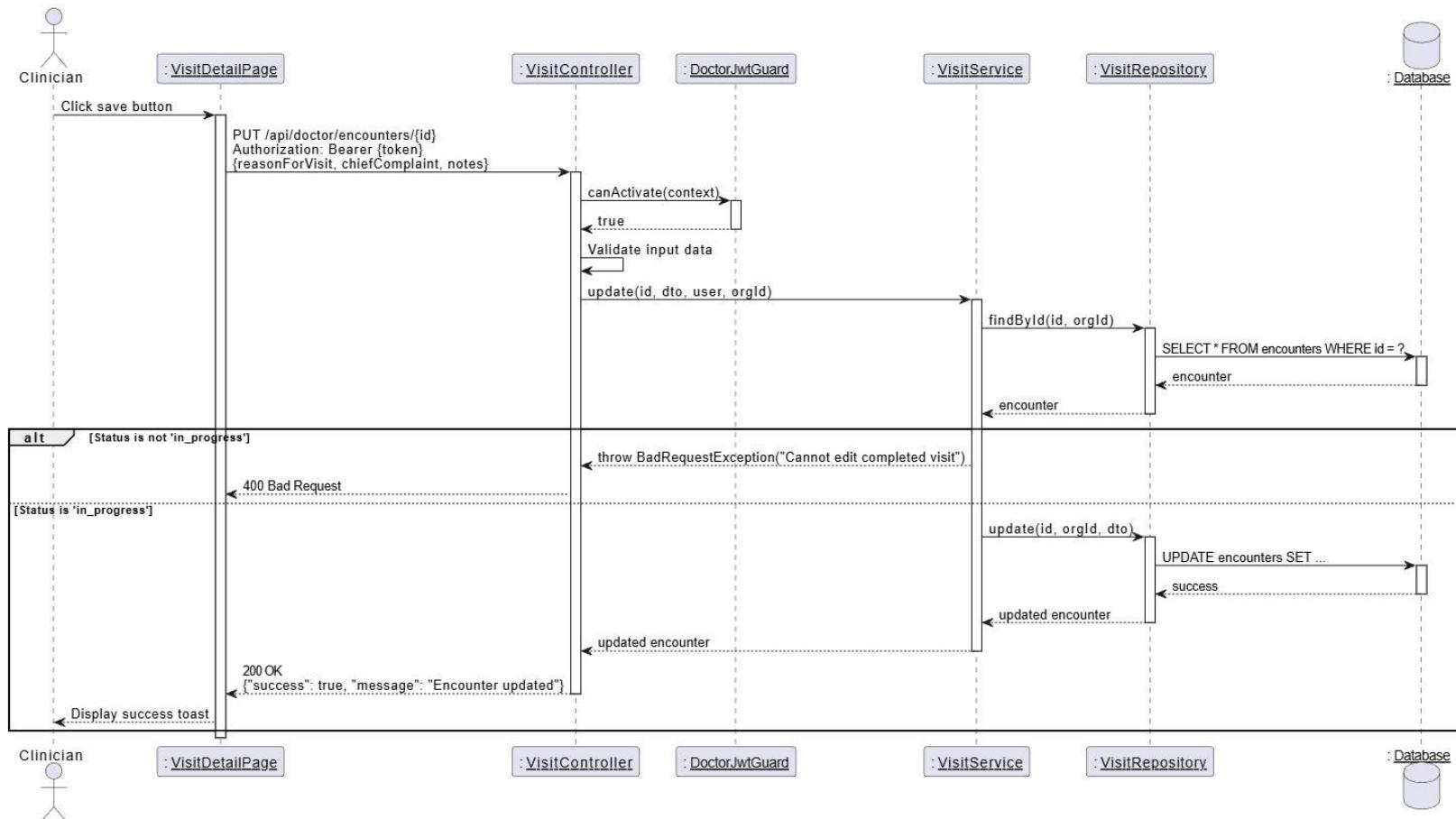


Diagram 30, Sequence Diagram ( Save Visit )

## ● Delete Visit :

<b>Use case ID</b>	<b>VEMR-FR-AM-14</b>
<b>Use case name</b>	Delete visit
<b>Description</b>	The system allows clinicians to delete a visit .
<b>From</b>	Clinician
<b>Pre-conditions</b>	Clinician is authenticated
<b>Main scenario</b>	<p>1.Clinician selects visit to delete from visits list or details page</p> <p>2.System prompts for confirmation dialog</p> <p>3.Clinician confirms deletion</p> <p>4.System authenticates user and verifies clinician is the assigned doctor</p> <p>5.System soft-deletes the encounter ( sets deletedAt timestamp )</p> <p>6.System displays success message and removes visit from list</p>
<b>Alternative scenario</b>	<p><b>A1:User Cancels</b></p> <ul style="list-style-type: none"> <li>- At step 3, if clinician cancels confirmation</li> <li>-No action is taken</li> </ul> <p><b>A2: Authentication Error</b></p> <ul style="list-style-type: none"> <li>- At step 4, if JWT token is invalid or expired</li> <li>-System redirects to login page</li> </ul>
<b>Post condition</b>	Visit is soft delete

Table 21, Use Case Specification (Delete Visit )

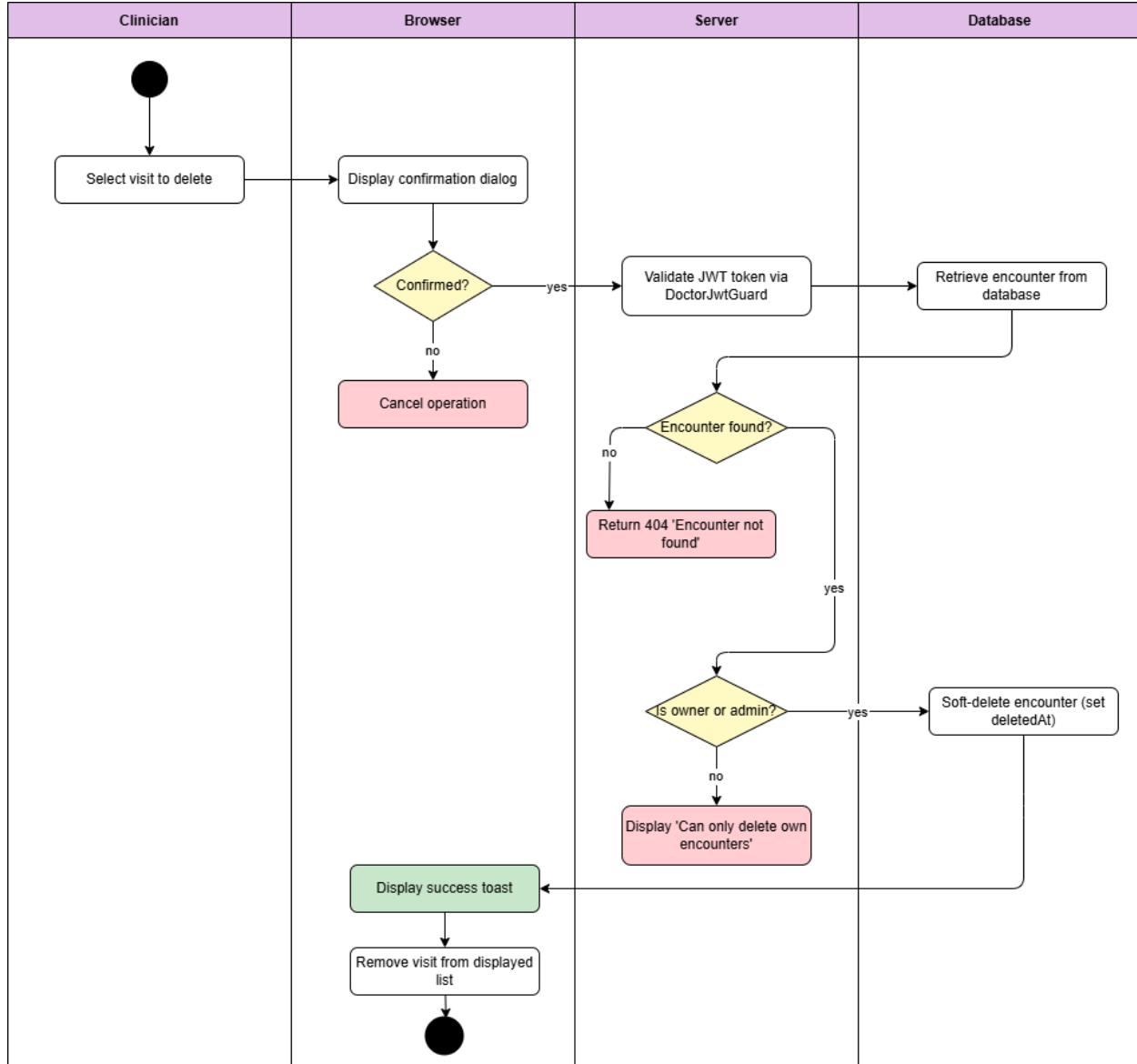


Diagram 31, Activity Diagram ( Delete Visit )

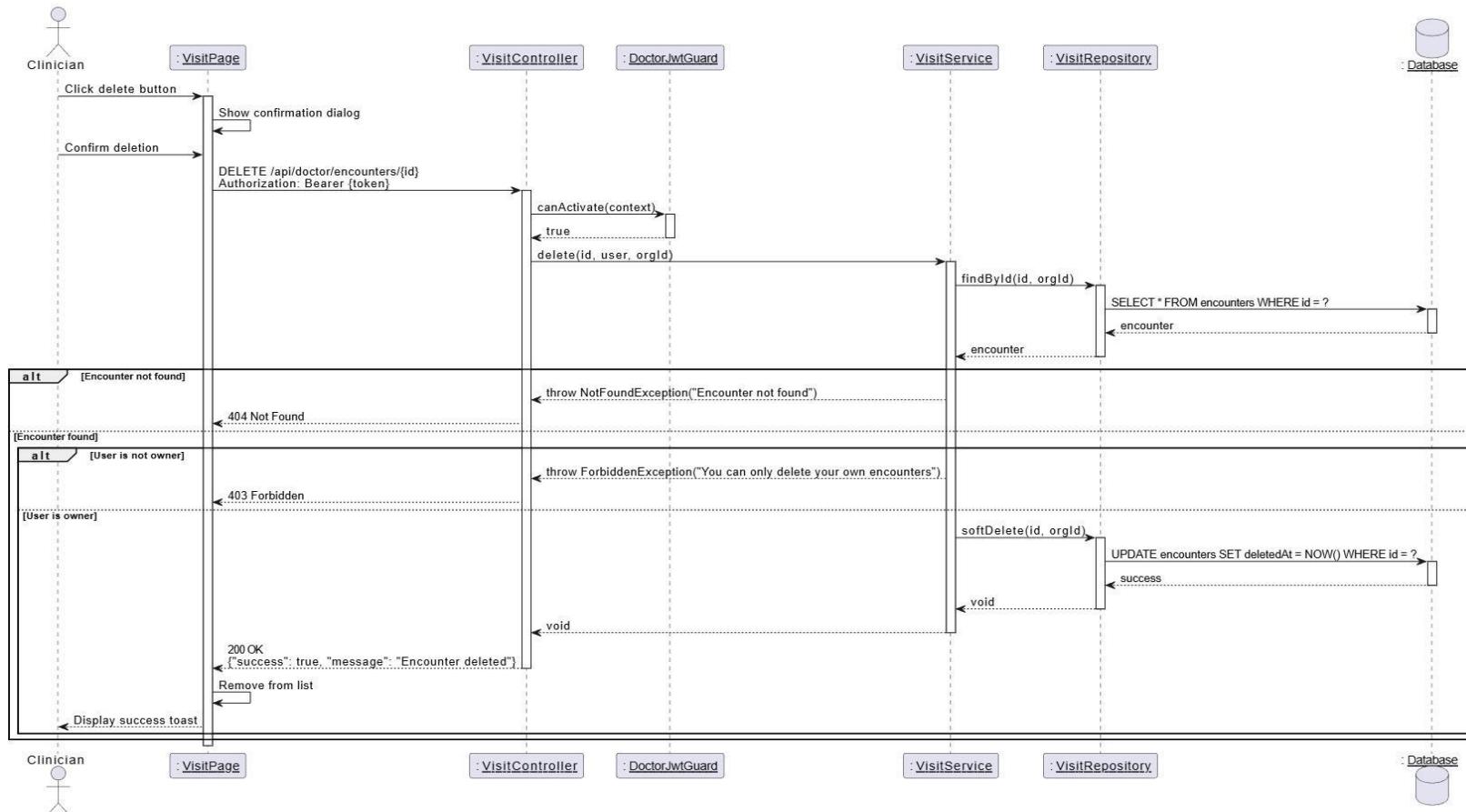


Diagram 32, Sequence Diagram ( Delete Visit )

## ● View Visit Details :

<b>Use case ID</b>	<b>VEMR-FR-AM-15</b>
<b>Use case name</b>	View visit details
<b>Description</b>	The system allows clinicians to view complete details of a visit
<b>From</b>	Clinician
<b>Pre-conditions</b>	Clinician is authenticated
<b>Main scenario</b>	<p>1.Clinician click on a visit form the visits list      2.System navigates to visit detail page      3.System authenticates user and retrieves encounter details      4.System retrieves associated medical record ,vital signs, and patient allergies      5.System displays visit details page with tabs (details ,vitals,Allergies )</p>
<b>Alternative scenario</b>	<p><b>A1: Visit Not Found</b></p> <ul style="list-style-type: none"> <li>- At step 3, if visit does not exist</li> <li>-System displays error page or redirects</li> </ul> <p><b>A2: Authentication Error</b></p> <ul style="list-style-type: none"> <li>- At step 3, if JWT token is invalid or expired</li> <li>-System redirects to login page</li> </ul>
<b>Post condition</b>	Visit details are displayed with all associated data

Table 22, Use Case Specification (View Visit Details )

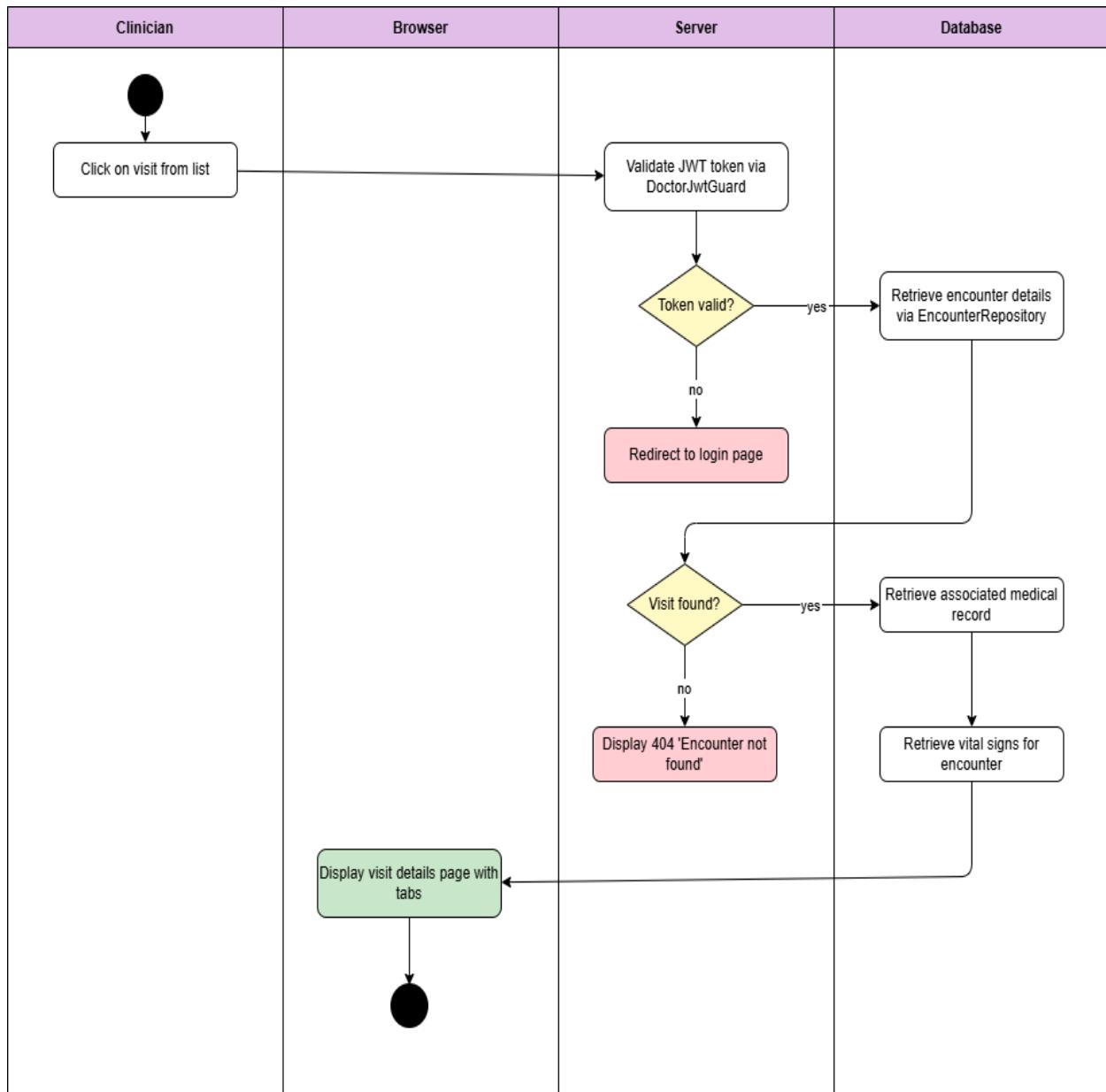


Diagram 33, Activity Diagram ( View Visit Details )

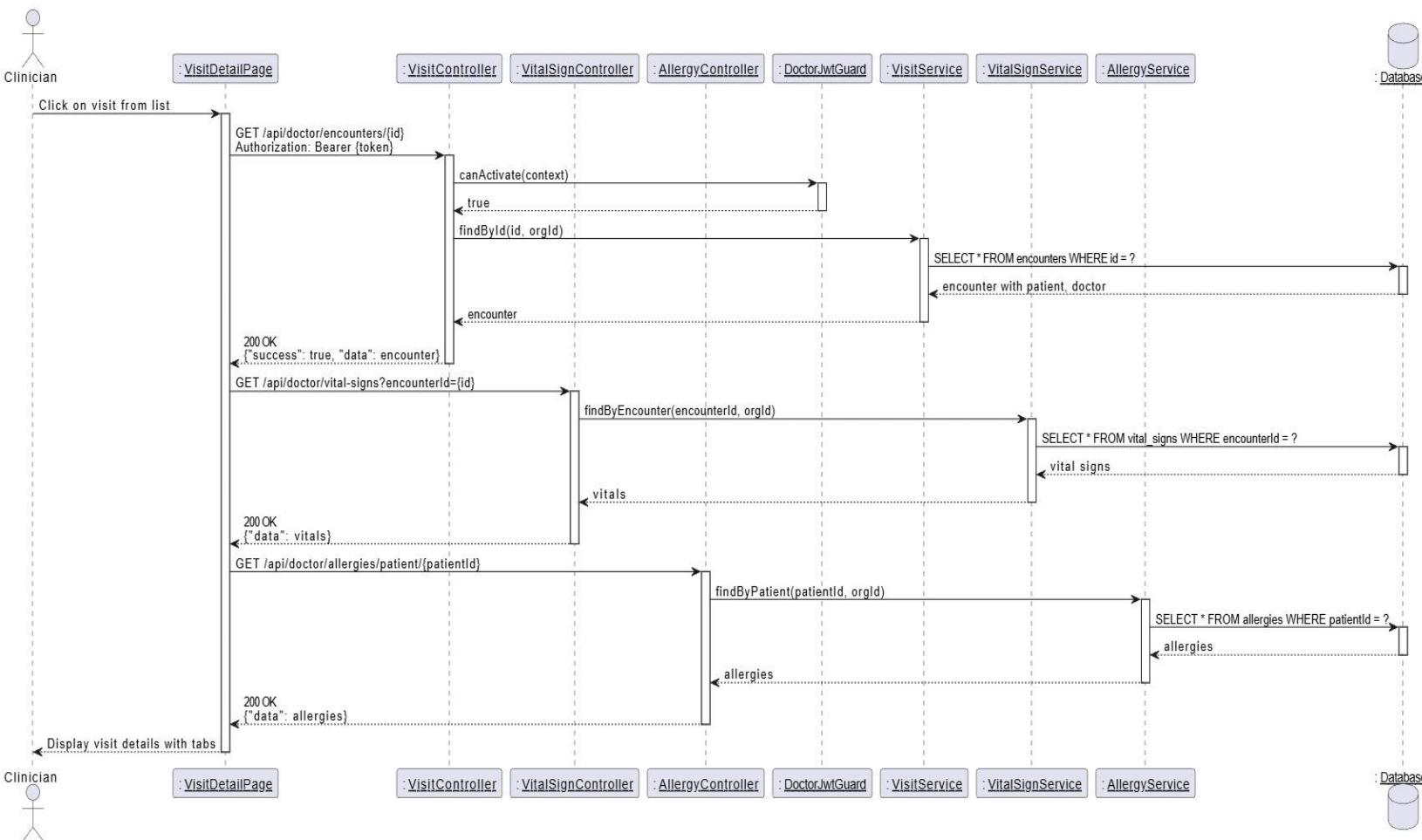


Diagram 34, Sequence Diagram ( View Visit Details )

## ● Real-time Transcription Display :

<b>Use case ID</b>	<b>VEMR-FR-AM-16</b>
<b>Use case name</b>	Real-time Transcription Display
<b>Description</b>	Voice transcription appears in real-time as the clinician speaks using whisper
<b>From</b>	Clinician
<b>Pre-conditions</b>	Clinician is authenticated
<b>Main scenario</b>	<p>1.Clinicians clicks microphone button on form field      2.Browser requests and user grants microphone permission      3.System establishes WebSocket connection to whisper server and starts recording      4.System sends audio chunks every 3 seconds to server      5.Whisper server transcribes audio and send partial text back      6.System displays transcription in real-time in the form fields      7.Clinician click stop button      8.System sends final audio chunk and receives final transcription      9.System closes connection and stop recording</p>
<b>Alternative scenario</b>	<p><b>A1: Microphone Access Denied</b></p> <ul style="list-style-type: none"> <li>- At step 2,if user denies permission</li> <li>-System displays error and recording remains inactive</li> </ul> <p><b>A2: Server Connection Failed</b></p> <ul style="list-style-type: none"> <li>- At step 3, if webSocket connection fails or server not running</li> <li>-System display error message</li> </ul>
<b>Post condition</b>	<ul style="list-style-type: none"> <li>- Transcribed text is added to the form field</li> <li>- Audio recording is stopped</li> <li>- WebSocket connection is closed</li> </ul>

Table 23, Use Case Specification (Real-time Transcription Display )

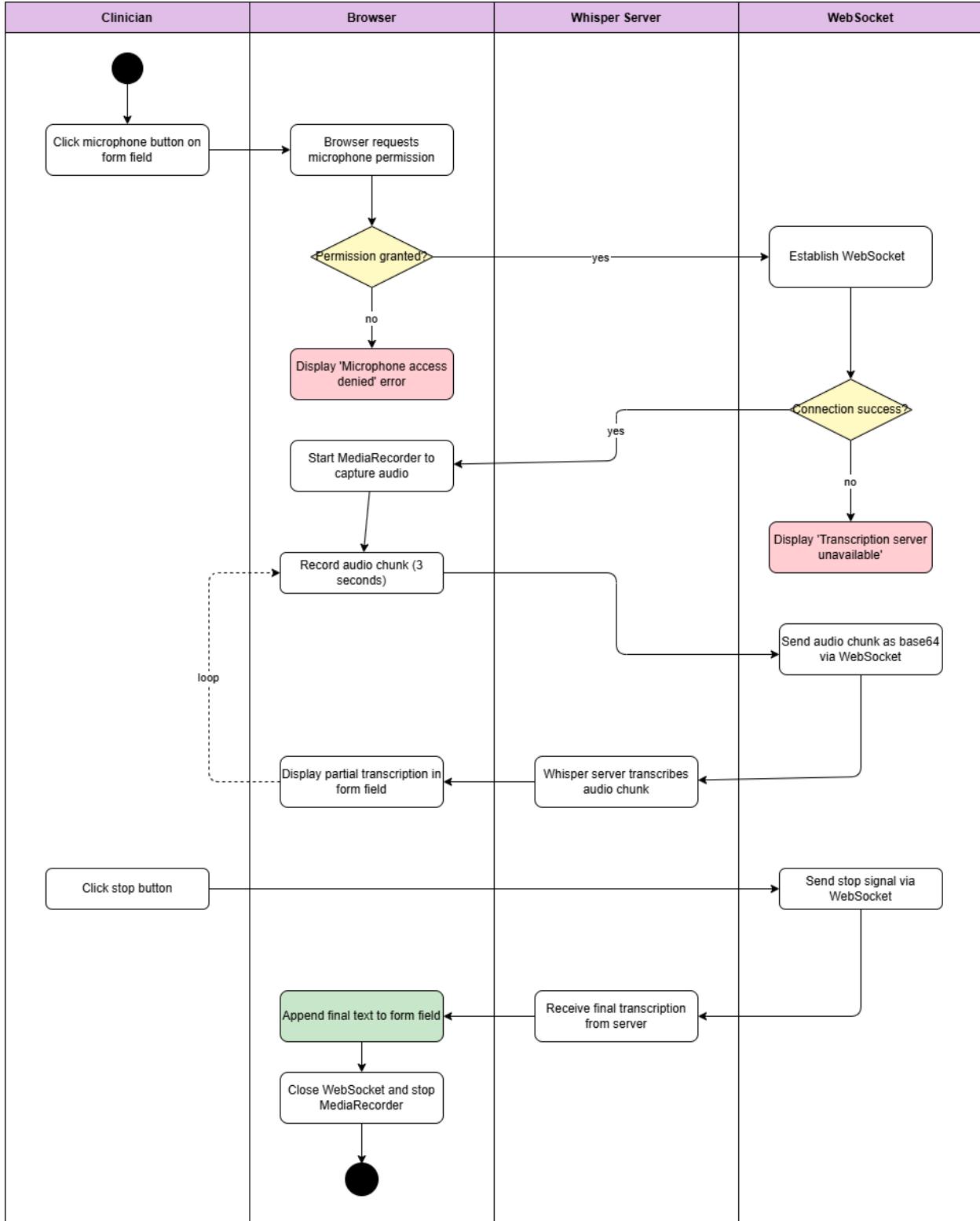


Diagram 35, Activity Diagram (Real-time Transcription Display )

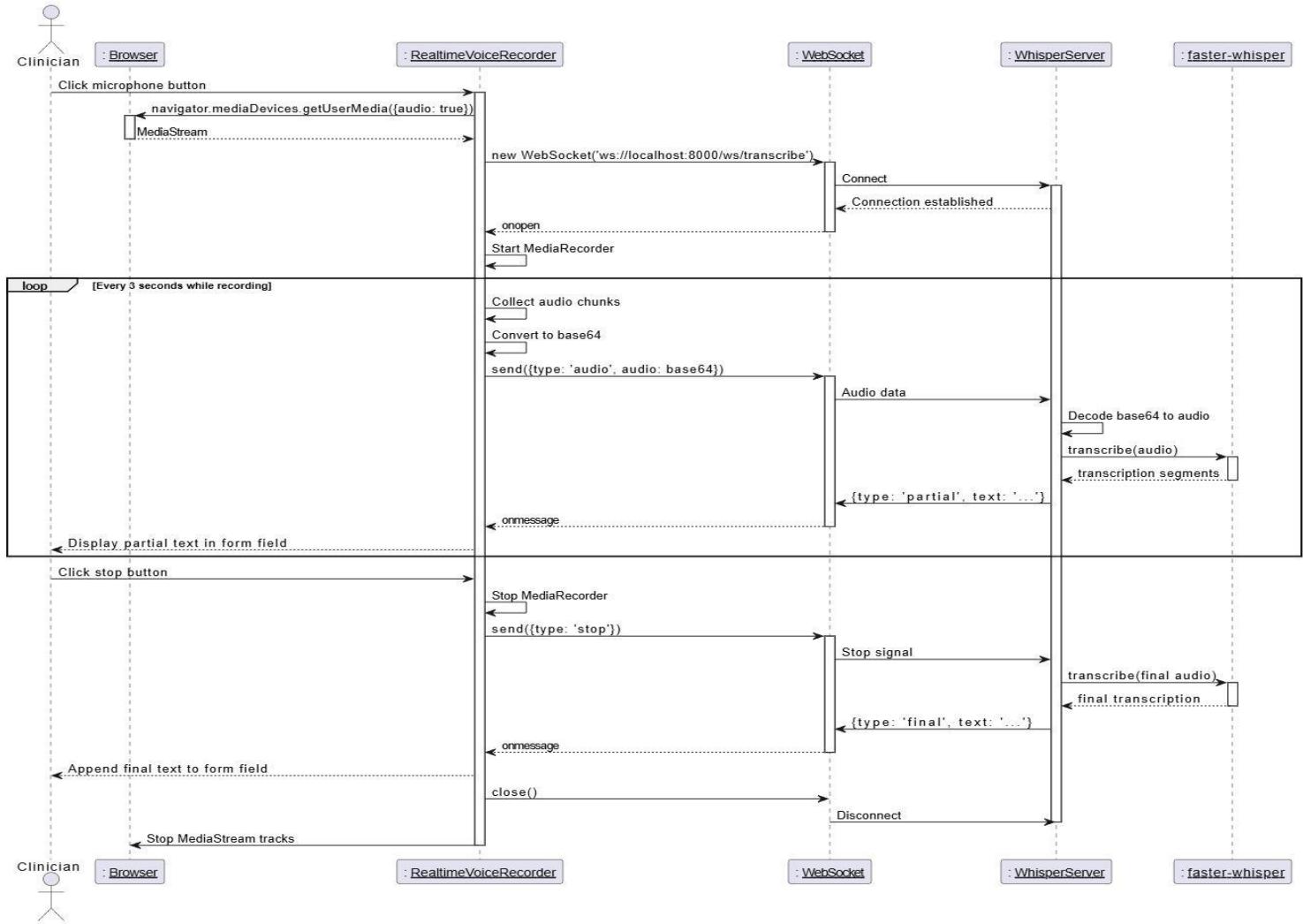


Diagram 36, Sequence Diagram (Real-time Transcription Display )

- **View Patient List:**

<b>Use case ID</b>	<b>VEMR-FR-AM-17</b>
<b>Use case name</b>	View patient list
<b>Description</b>	The system allows clinicians to view a paginated list of all patients
<b>From</b>	Clinician
<b>Pre-conditions</b>	Clinician is authenticated
<b>Main scenario</b>	<p>1.Clinician navigates to the patients page .</p> <p>2.System authenticates user and retrieves paginated patient list</p> <p>3.System displays patients in a table with basic info ( name , email , phone , date of birth).</p> <p>4.Clinician can navigate between pages using pagination controls</p>
<b>Alternative scenario</b>	<p><b>A1: No Patients</b></p> <ul style="list-style-type: none"> <li>- At step 2, No patient exist</li> <li>-System displays “ No patients found “ message</li> </ul> <p><b>A2: Authentication Error</b></p> <ul style="list-style-type: none"> <li>- At step 3, if JWT token is invalid or expired</li> <li>-System return 401 Unauthorized</li> </ul>
<b>Post condition</b>	Patient list is displayed with pagination

Table 24, Use Case Specification (View Patient List)

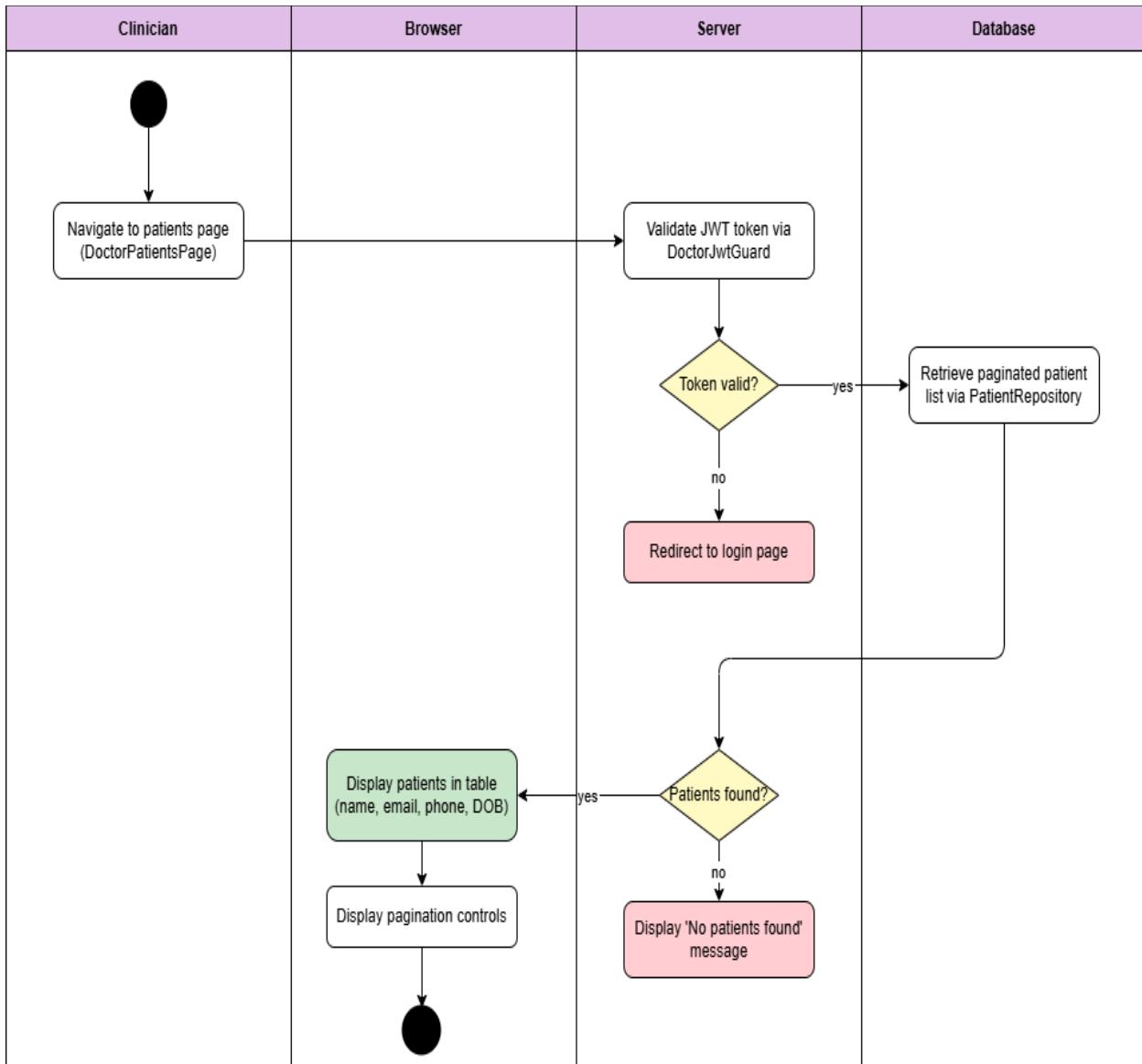


Diagram 374, Activity Diagram (View Patient List)

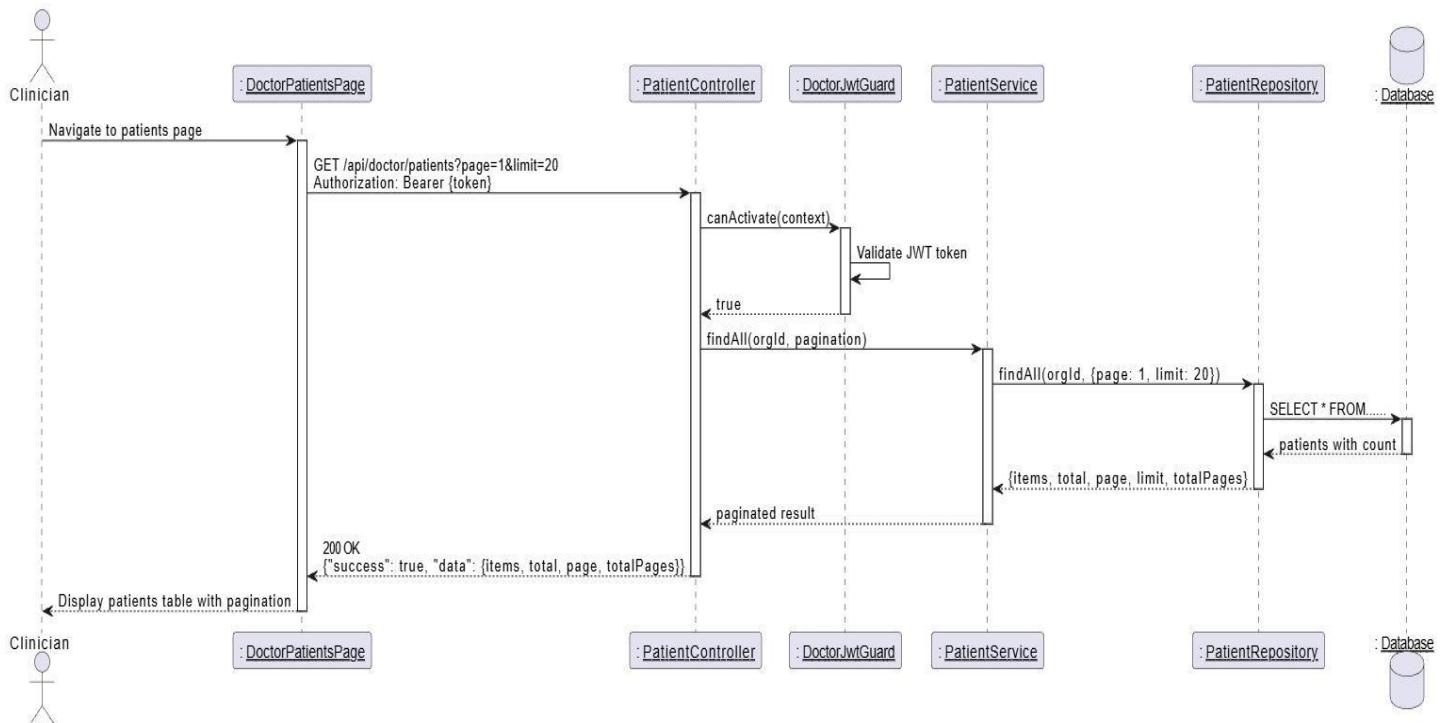


Diagram 38, Sequence Diagram (View Patient List)

## ● Search Patient :

<b>Use case ID</b>	<b>VEMR-FR-AM-18</b>
<b>Use case name</b>	Search patient
<b>Description</b>	The system allows clinicians to search for patient
<b>From</b>	Clinician
<b>Pre-conditions</b>	Clinician is authenticated
<b>Main scenario</b>	<p>1.Clinician enters search query in search input field      2.System authenticates user and searches patients by name      3.System returns paginated results with matching patients      4.System displays search results in patients table</p>
<b>Alternative scenario</b>	<p><b>A1: No Results Found</b></p> <ul style="list-style-type: none"> <li>- At step 3,if no patients match the query</li> <li>-System displays "No patients found" message</li> </ul> <p><b>A2: Empty Search Query</b></p> <ul style="list-style-type: none"> <li>- At step 2,if search query is cleared</li> <li>-System returns all patients with pagination</li> </ul> <p><b>A3: Authentication Error</b></p> <ul style="list-style-type: none"> <li>- At step 3,if JWT token is invalid or expired</li> <li>-System redirects to login page</li> </ul>
<b>Post condition</b>	Matching patient are displayed in the table

Table 25, Use Case Specification (Search Patient )

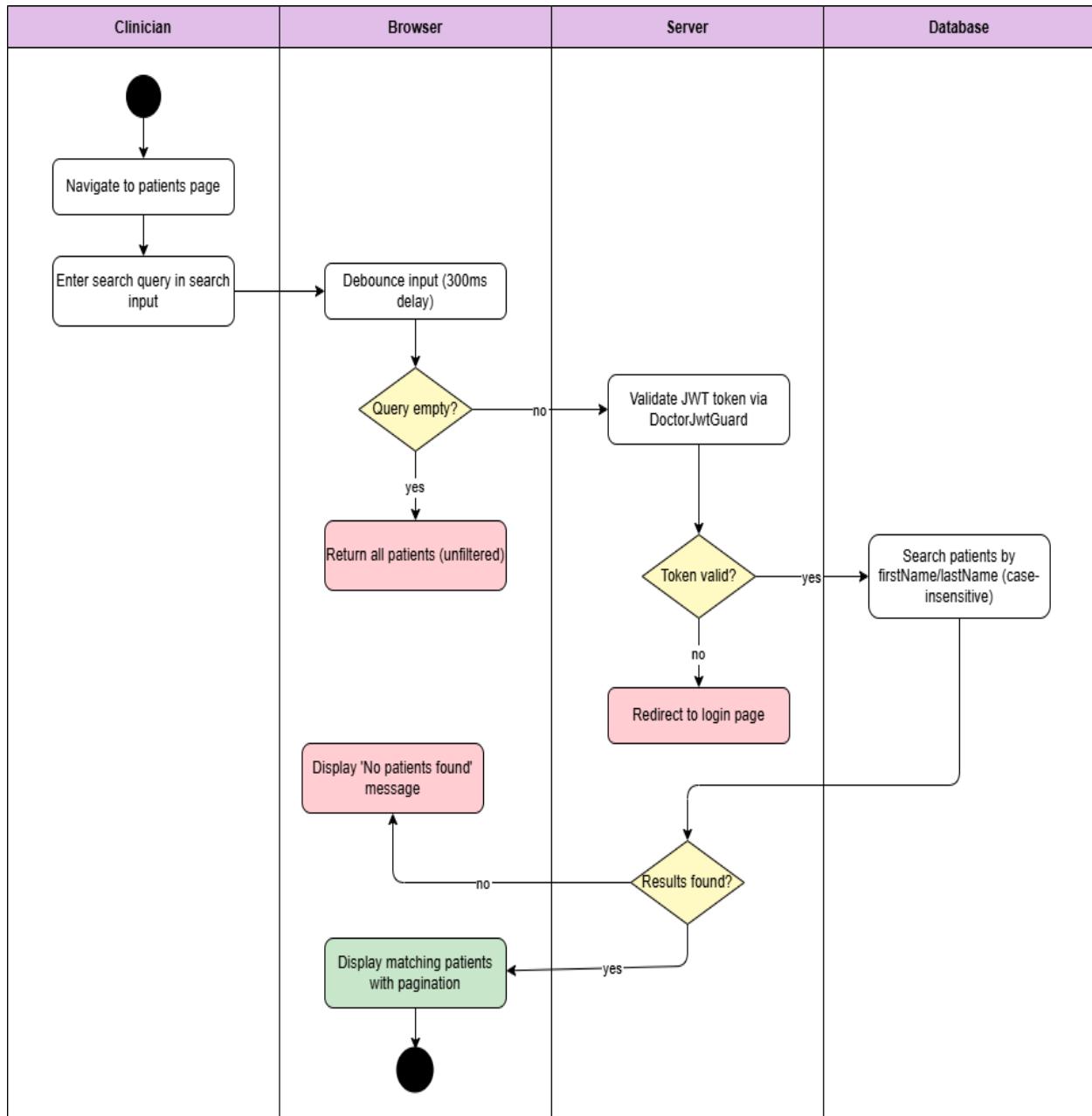


Diagram 39,Activity Diagram (Search Patient )

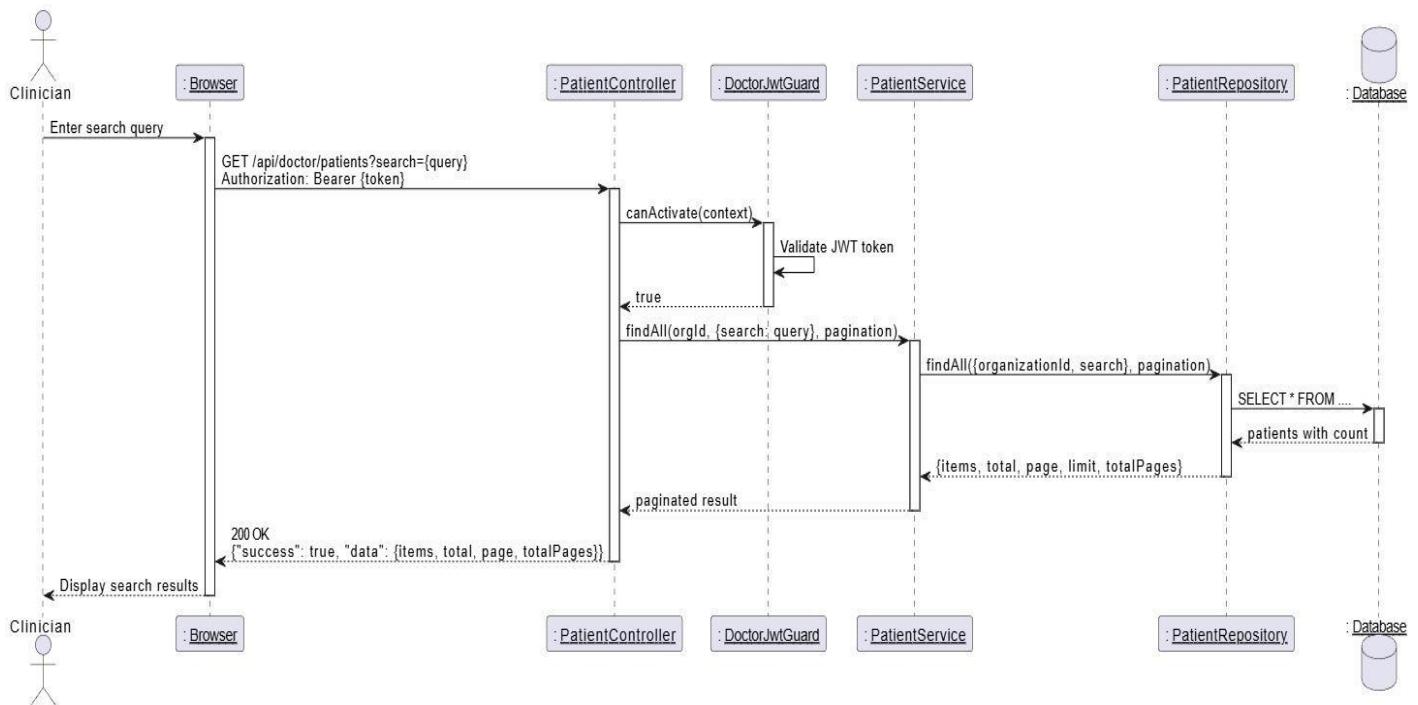


Diagram 40, Sequence Diagram (Search Patient )

- **View Patient Profile :**

<b>Use case ID</b>	<b>VEMR-FR-AM-19</b>
<b>Use case name</b>	View patient Profile
<b>Description</b>	The system allows clinicians to view a patient's profile with their visits and allergies.
<b>From</b>	Clinician
<b>Pre-conditions</b>	Clinician is authenticated
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. Clinician navigates to patients list page</li> <li>2. Clinician searches for patient by name</li> <li>3. Clinician clicks on patient name to view profile</li> <li>4. System validates JWT token via DoctorJwtGuard</li> <li>5. System retrieves patient data via PatientRepository</li> <li>6. System retrieves patient visits via EncounterRepository</li> <li>7. System retrieves patient allergies via AllergyRepository</li> <li>8. System displays patient profile with personal details, visits history, and allergies list</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Patient Not Found</b></p> <ul style="list-style-type: none"> <li>- At step 5, if patient ID does not exist</li> <li>- System displays error message</li> </ul> <p><b>A2: No Visits or Allergies</b></p> <ul style="list-style-type: none"> <li>- At step 6 or 7, if patient has no visits or allergies</li> <li>- System displays empty state message</li> </ul>
<b>Post condition</b>	Patient details are displayed

Table 26, Use Case Specification (View Patient Profile )

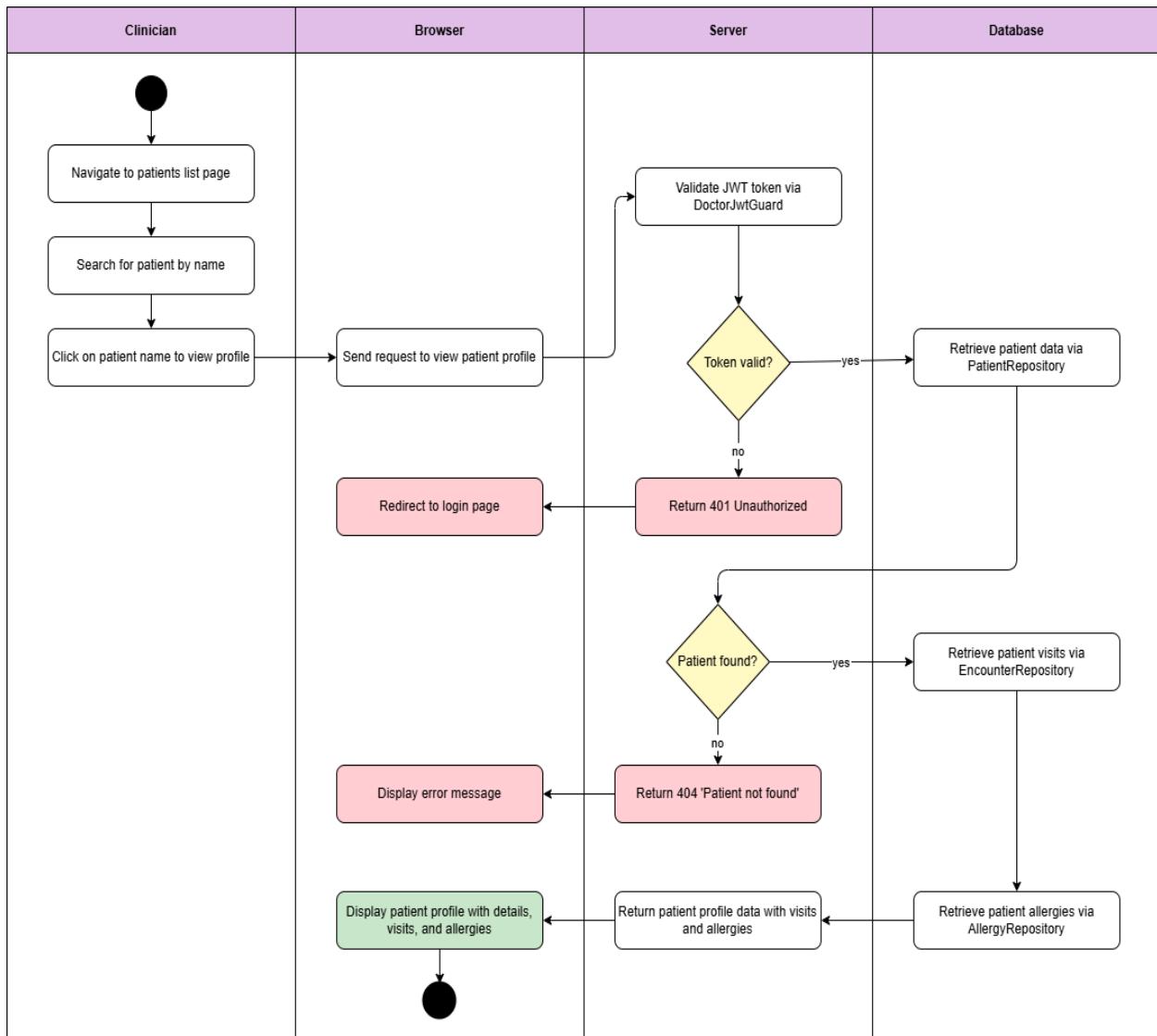


Diagram 41,Activity Diagram (View Patient Profile )

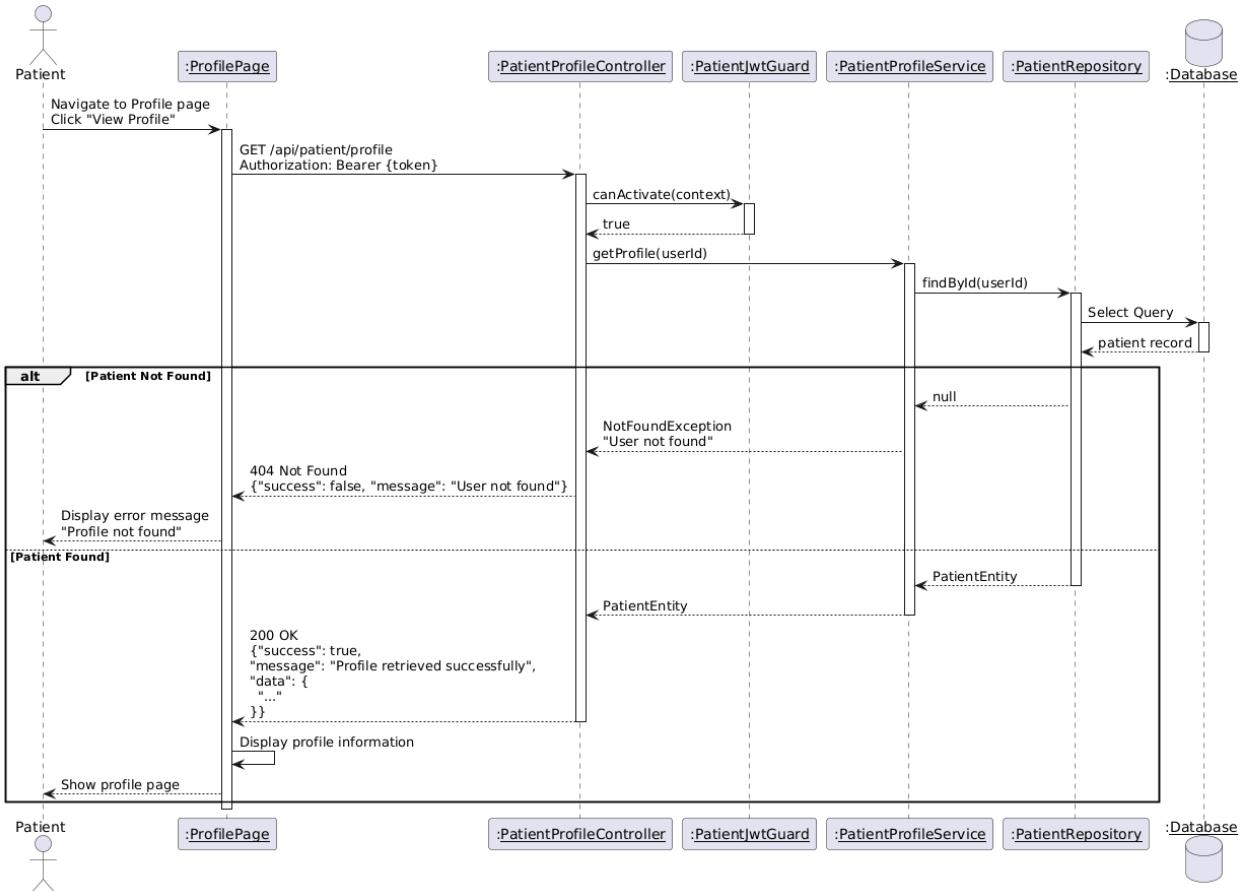


Diagram 42, Sequence Diagram (View Patient Profile )

- **View Patient Details :**

<b>Use case ID</b>	<b>VEMR-FR-AM-20</b>
<b>Use case name</b>	View patient details
<b>Description</b>	The system allows clinicians to view detailed patient information
<b>From</b>	Clinician

<b>Pre-conditions</b>	Clinician is authenticated
<b>Main scenario</b>	1.Clinician click on a patient from the patients list 2.System navigates to patient detail page 3.System authenticates user and retrieves patient details 4.System retrieves patient encounters and allergies 5.System displays patient profile with tabs (Visits, Allergies )
<b>Alternative scenario</b>	<b>A1: Patient Not Found</b> <ul style="list-style-type: none"> <li>- At step 3,if patient does not exist</li> <li>-System displays error page or redirects</li> </ul> <b>A2: Authentication Error</b> <ul style="list-style-type: none"> <li>- At step 3,if JWT token is invalid or expired</li> <li>-System redirects to login page</li> </ul>
<b>Post condition</b>	Patient details are displayed

Table 27, Use Case Specification (View Patient Details)

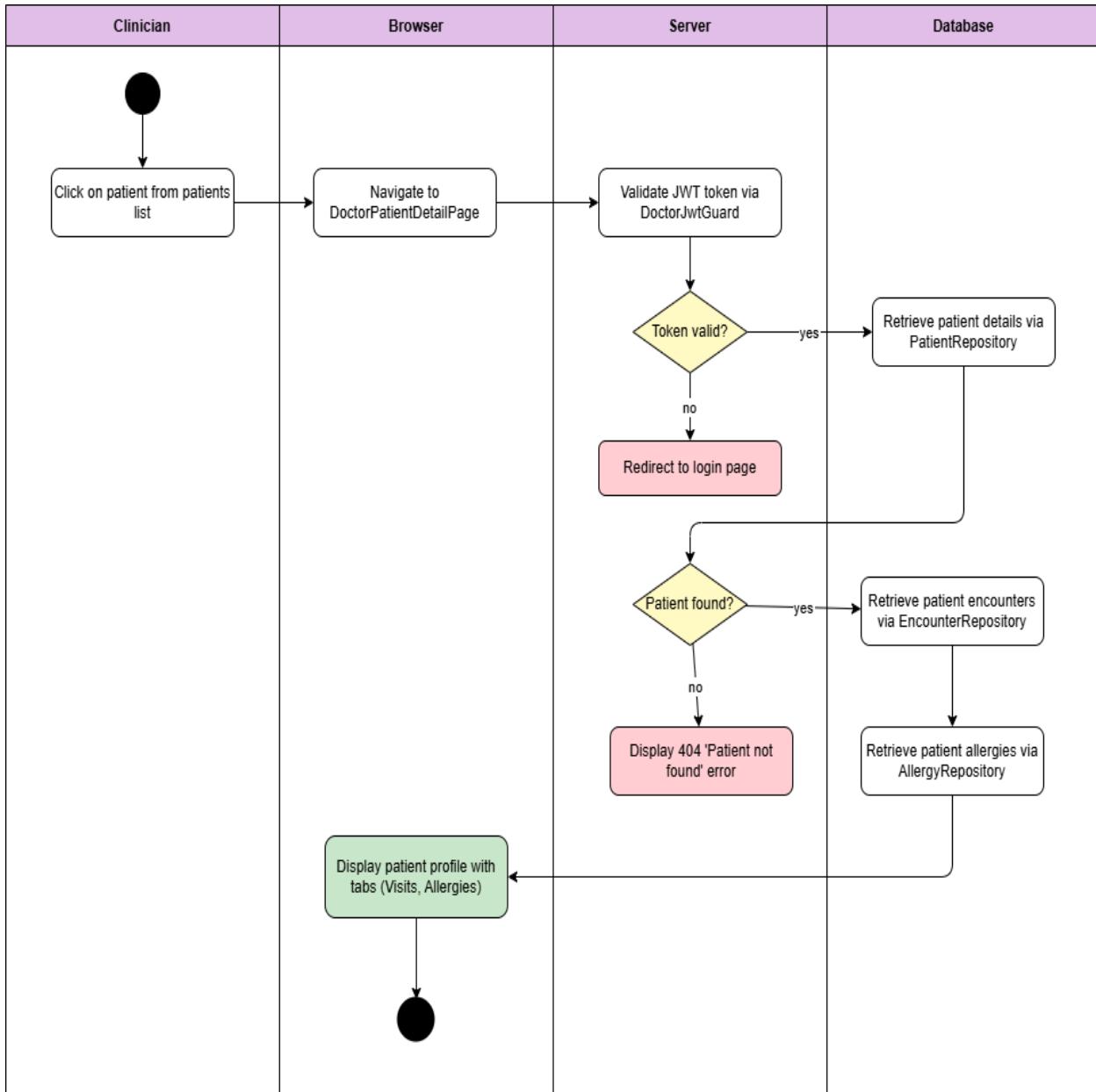


Diagram 37, Activity Diagram (View Patient Details)

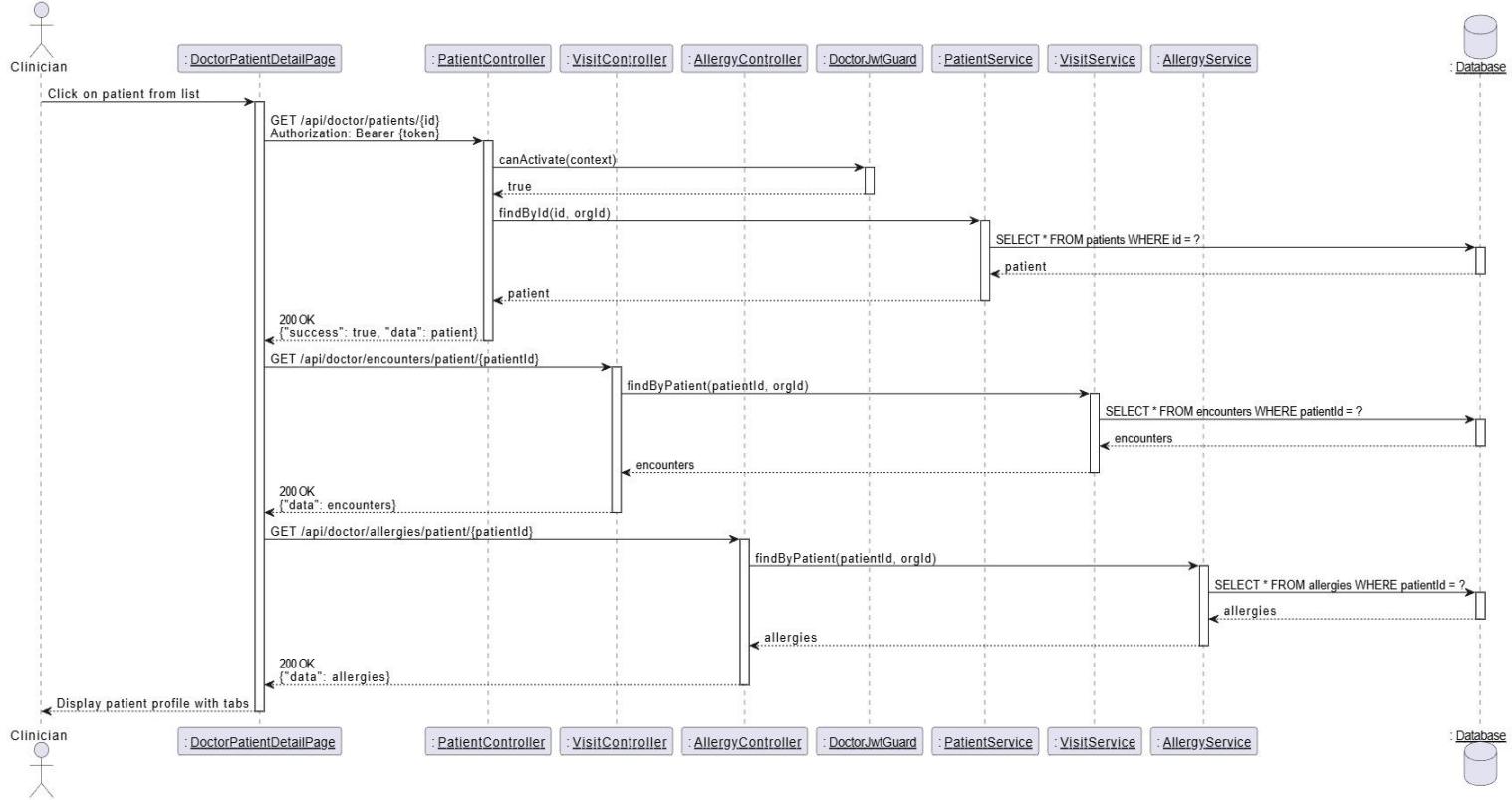


Diagram 43, Sequence Diagram (View Patient Details )

- Change Visit Status :

<b>Use case ID</b>	<b>VEMR-FR-AM-21</b>
<b>Use case name</b>	Change visit status
<b>Description</b>	The system allows clinicians to change visit status (Start - Complete - Cancel ).
<b>From</b>	Clinician
<b>Pre-conditions</b>	Clinician is authenticated
<b>Main scenario</b>	<p><b>Start Visit :</b></p> <ol style="list-style-type: none"> <li>1.Clinician clicks “Start visit” button on a planned visit .</li> <li>2.System authenticates user and validates visit status is a “planned”</li> <li>3.System updates status to “in_progress” and sets statTime</li> <li>4.System displays success message.</li> </ol> <p><b>Complete Visit :</b></p> <ol style="list-style-type: none"> <li>1.Clinician clicks “complete visit” button on an in_progress visit.</li> <li>2.System authenticates user and validates visit status is a “in_progress”</li> <li>3.System updates status to “Complete” and sets endTime</li> <li>4.System displays success message.</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Invalid Status Transition</b></p> <ul style="list-style-type: none"> <li>- At step 2,if visit status does not allow the transition</li> <li>-System displays error message</li> </ul> <p><b>A2: Authentication Error</b></p> <ul style="list-style-type: none"> <li>- At step 2,if JWT token is invalid or expired</li> <li>-System redirects to login page</li> </ul>
<b>Post condition</b>	

Table 28, Use Case Specification (Change Visit Status)

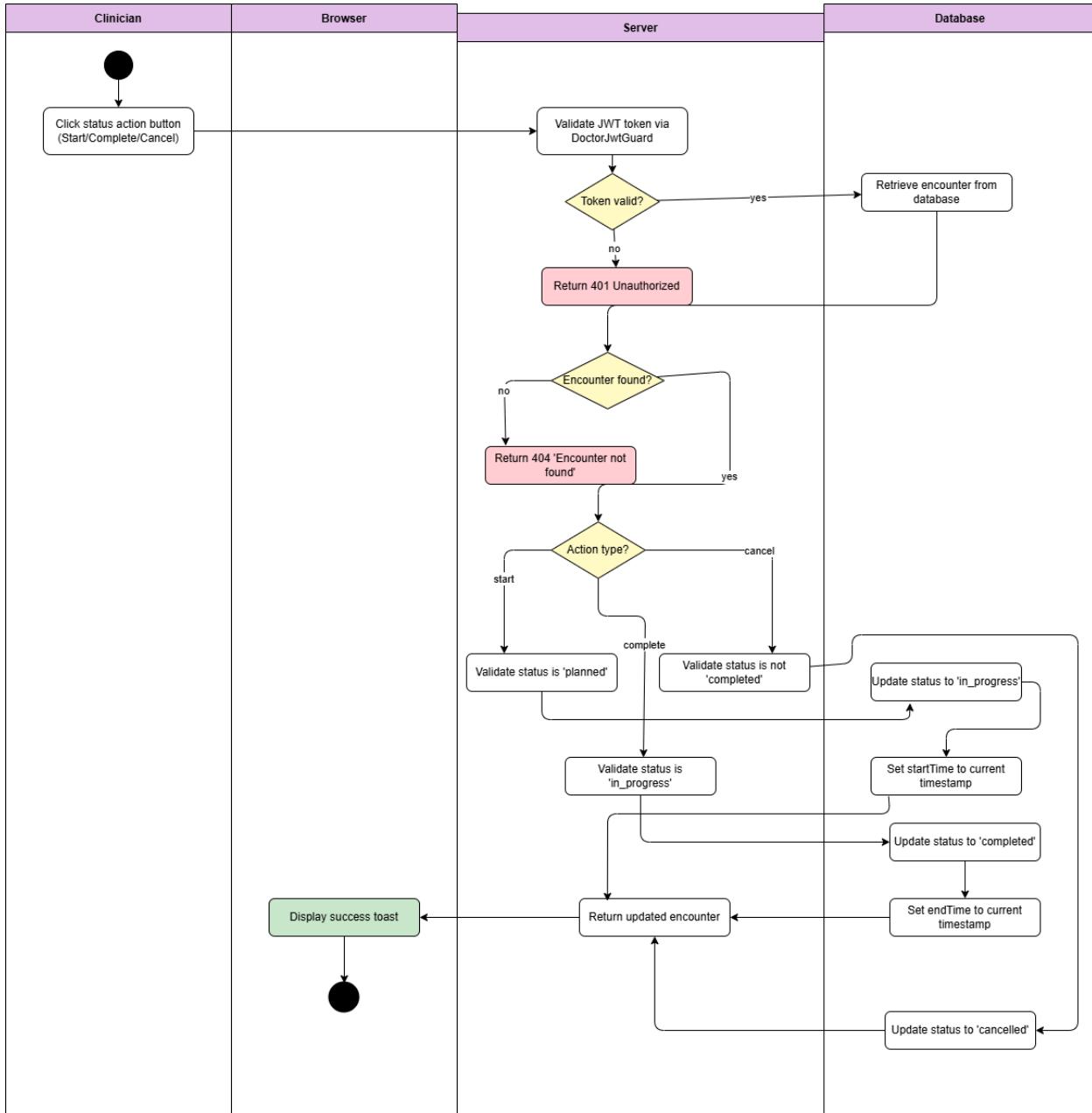


Diagram 44, Activity Diagram (Change Visit Status )

## Chapter 4 - System Analysis

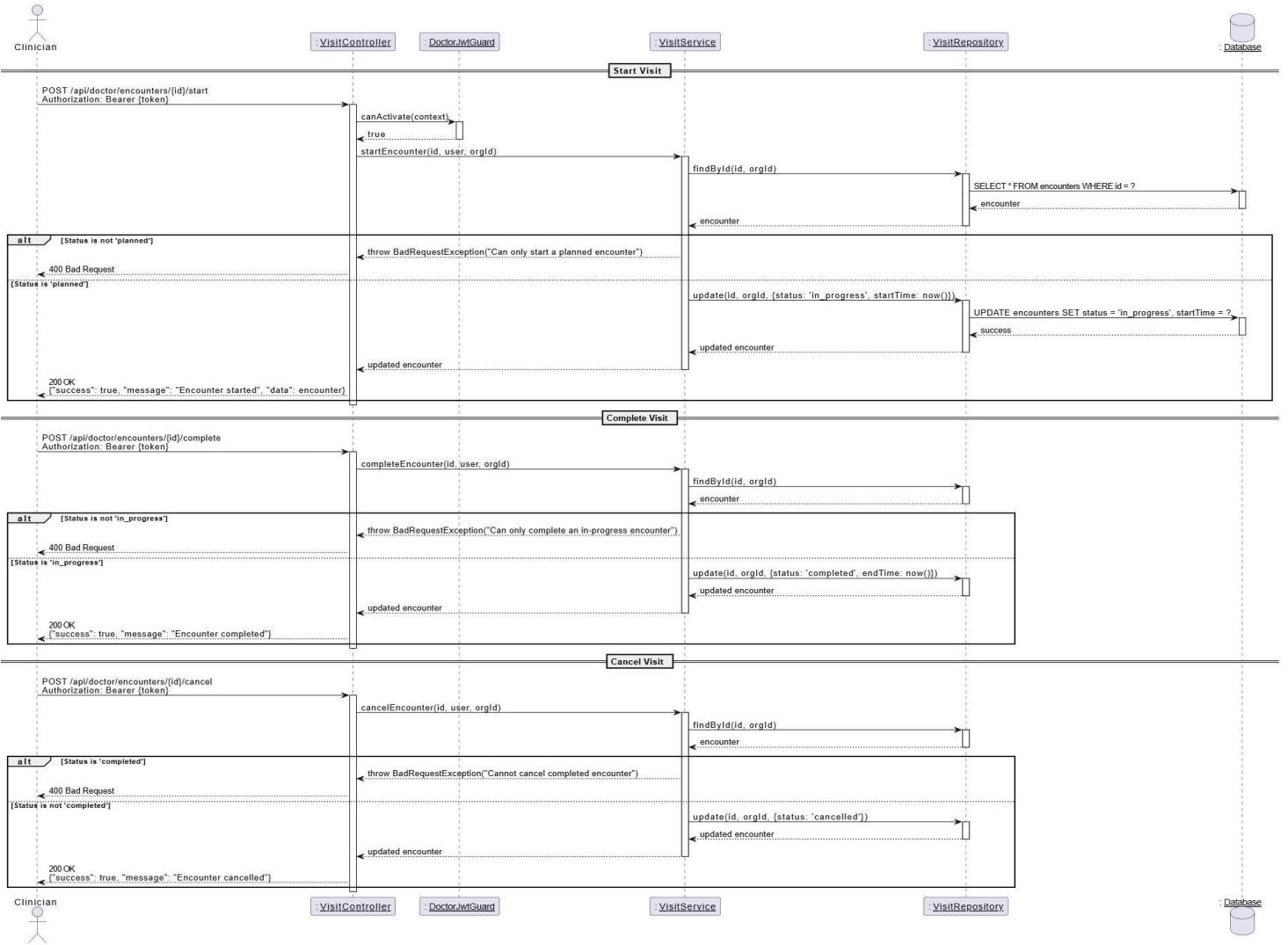


Diagram 45, Sequence Diagram (Change Visit Status )

- **View Visit List:**

<b>Use case ID</b>	<b>VEMR-FR-AM-22</b>
<b>Use case name</b>	View visit list
<b>Description</b>	The system allows clinicians to view a paginated list
<b>From</b>	Clinician
<b>Pre-conditions</b>	Clinician is authenticated
<b>Main scenario</b>	<p>1.Clinician navigated to visits page      2.System authenticates user and retrieves paginated encounter      3.System displays visits in a table with patient name,status,date,type.      4.Clinicians can filter by status , search , or paginate</p>
<b>Alternative scenario</b>	<p><b>A1: No visit</b></p> <ul style="list-style-type: none"> <li>- At step 2,if no encounters exist</li> <li>-System displays “No visits found” message</li> </ul> <p><b>A2: Authentication Error</b></p> <ul style="list-style-type: none"> <li>- At step 2,if JWT token is invalid or expired</li> <li>-System redirects to login page</li> </ul>
<b>Post condition</b>	Visits list in displayed with pagination

Table 29, Use Case Specification (View Visit List)

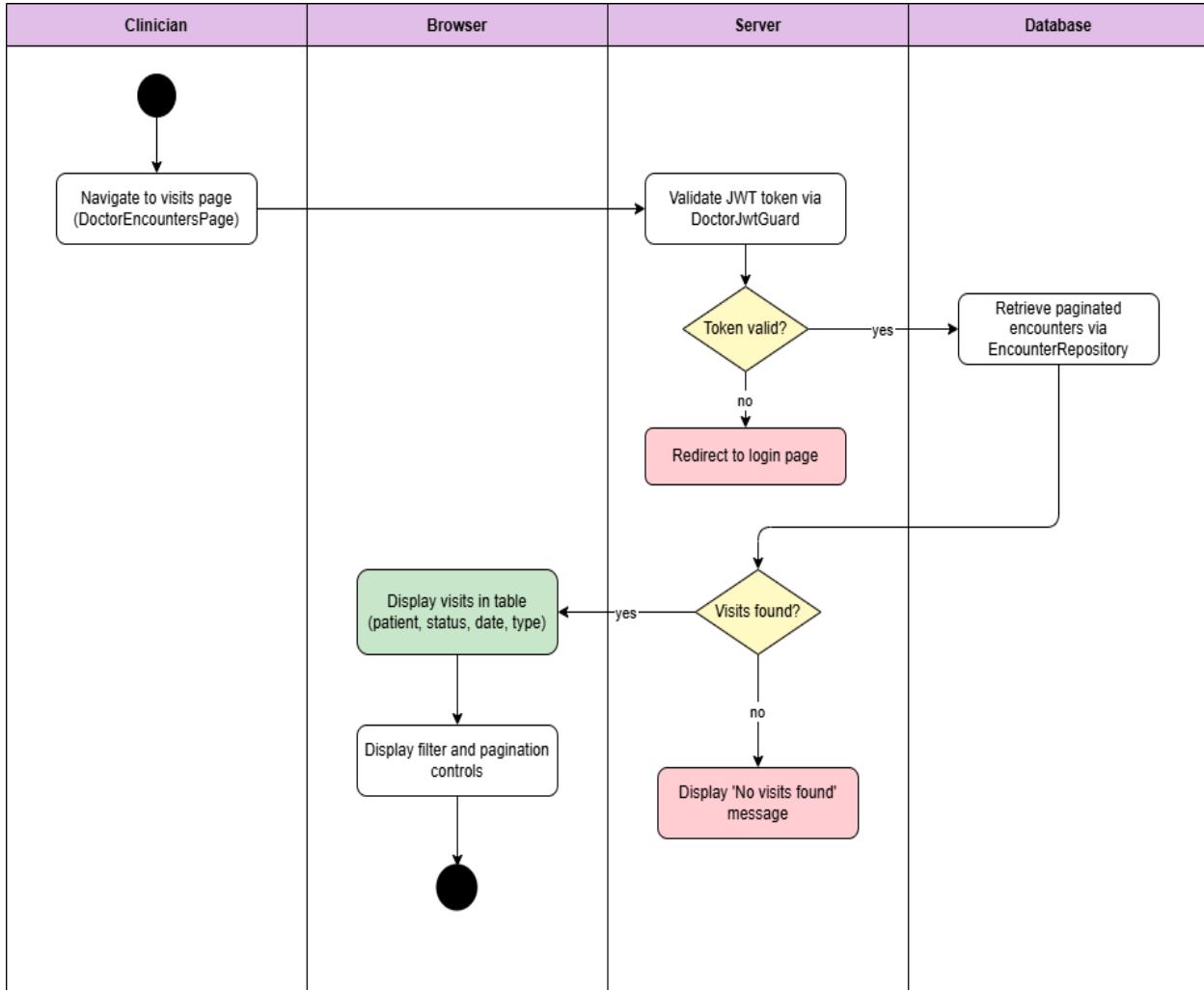


Diagram 46, Activity Diagram (View Visit List)

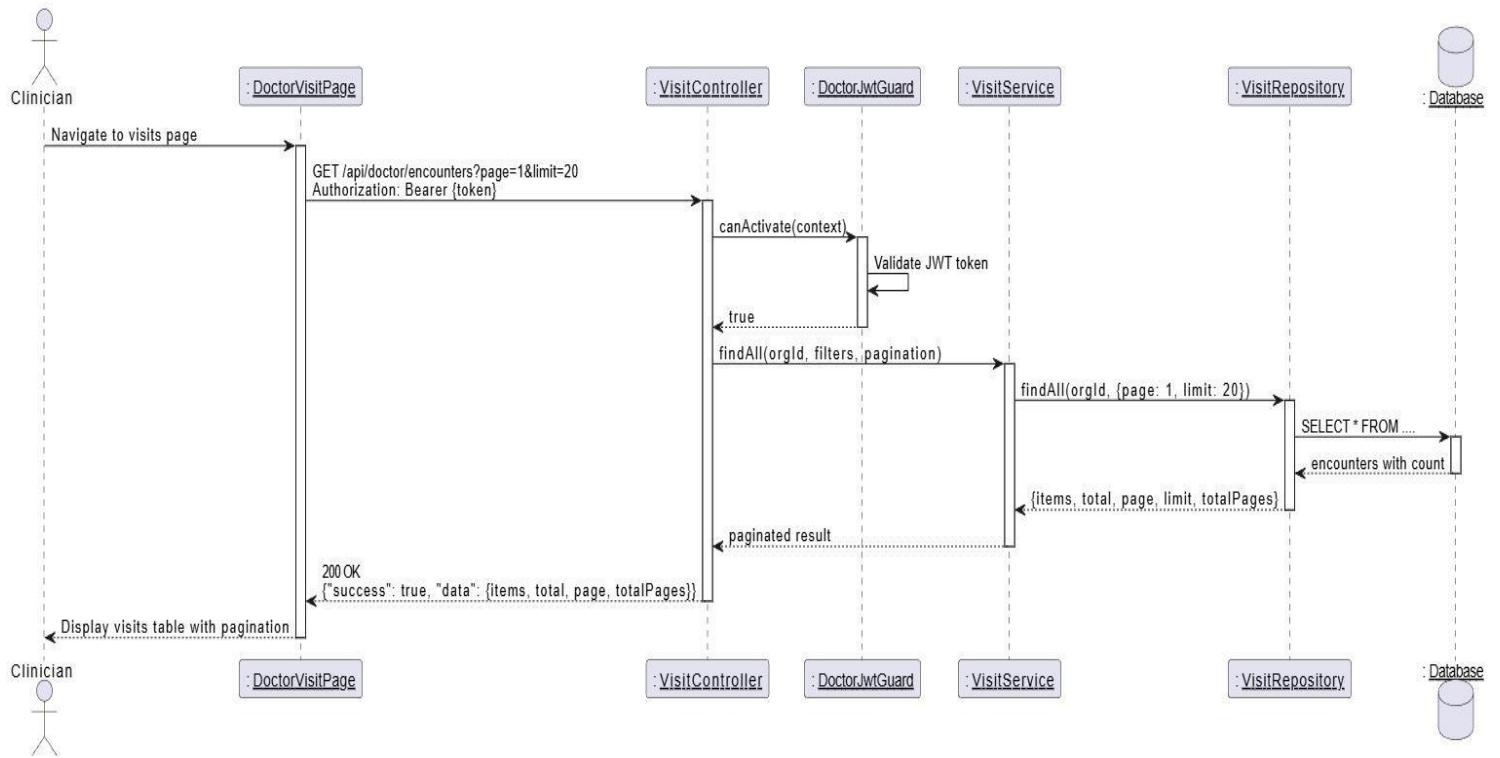


Diagram 47, Sequence Diagram (View Visit List)

- **View Patient Vital Signs :**

<b>Use case ID</b>	<b>VEMR-FR-AM-23</b>
<b>Use case name</b>	View Patient Vital Signs
<b>Description</b>	The system allows clinicians to view vital signs recorded
<b>From</b>	Clinician
<b>Pre-conditions</b>	Clinician is authenticated
<b>Main scenario</b>	<p>1.Clinician opens visit detail page and clicks “Vital signs” tab      2.System authenticates user and retrieves vital signs for this encounter      3.System displays vital signs grid ( BP - HR - Temp - RR - Spo2 - Weight -Height - BMI ).</p>
<b>Alternative scenario</b>	<p><b>A1: No Vital Record</b></p> <ul style="list-style-type: none"> <li>- At step 2,if no vital exist for this visit .        -System displays empty state with “- - “ values</li> </ul> <p><b>A2: Authentication Error</b></p> <ul style="list-style-type: none"> <li>- At step 2,if JWT token is invalid or expired        -System redirects to login page</li> </ul>
<b>Post condition</b>	Visits list

Table 30, Use Case Specification (View Patient Vital Signs)

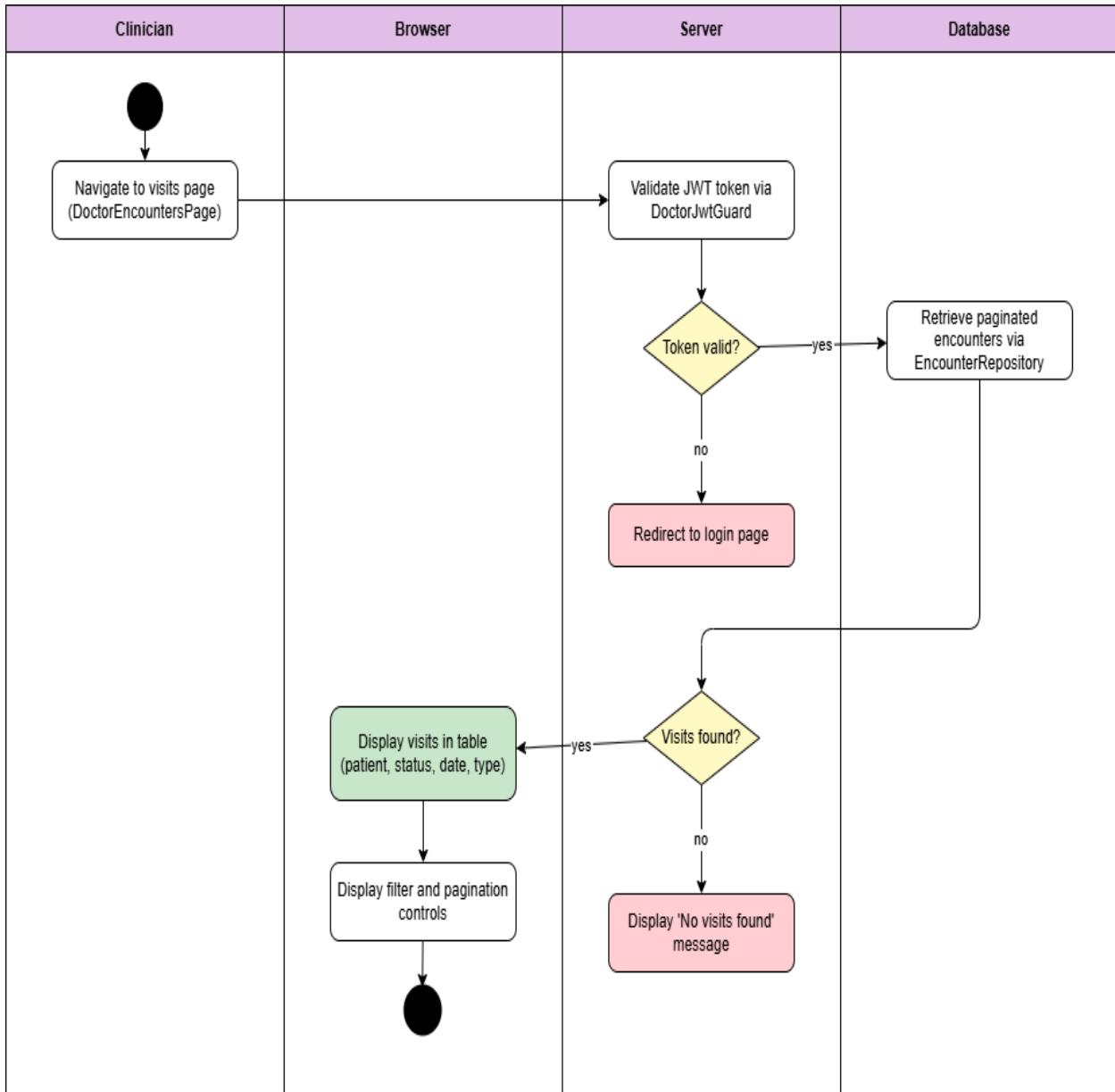


Diagram 48, Activity Diagram (View Patient Vital Signs)

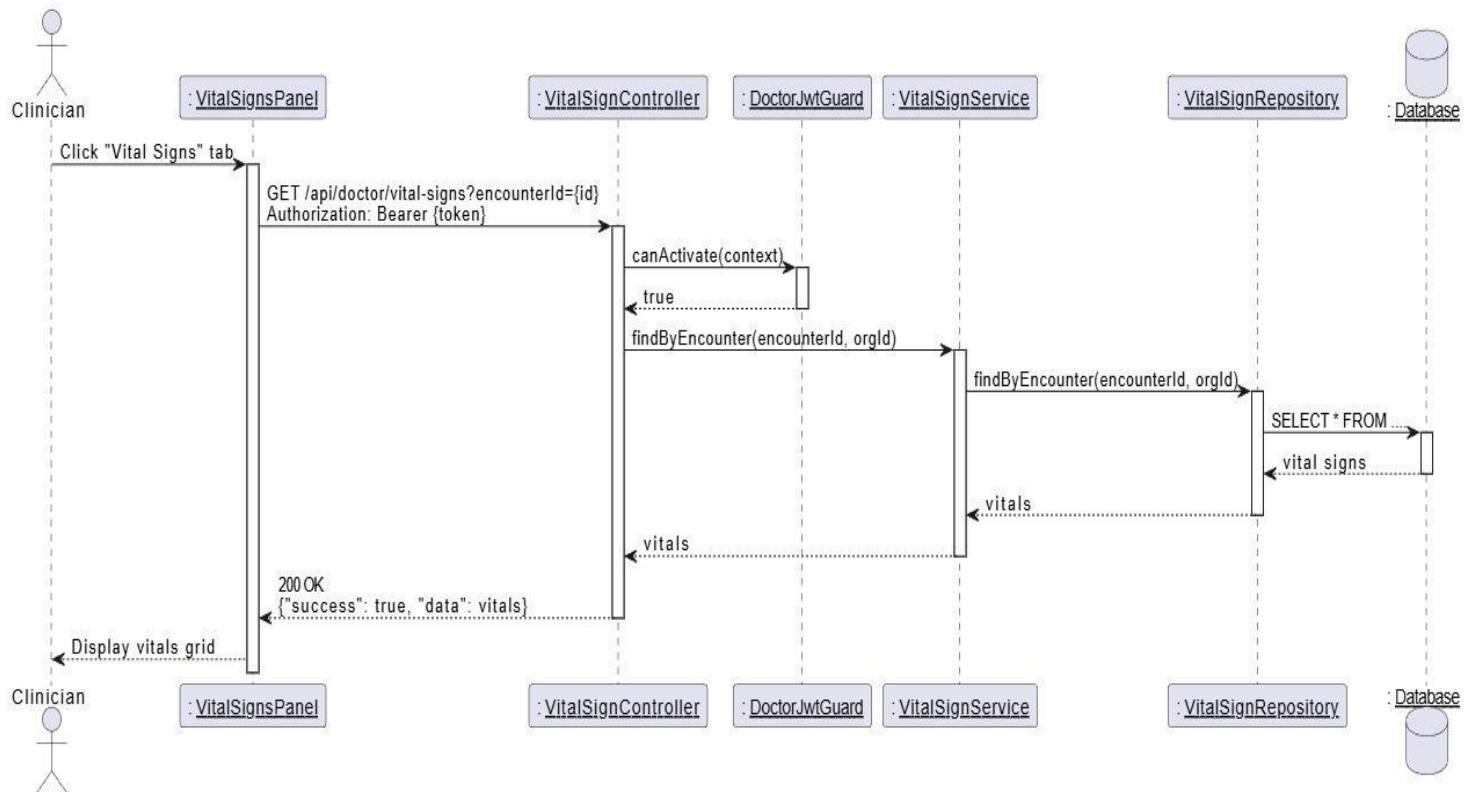


Diagram 49, SequenceDiagram (View Patient Vital Signs)

## ● Edit Patient Vital Signs :

<b>Use case ID</b>	<b>VEMR-FR-AM-24</b>
<b>Use case name</b>	Edit Patient Vital Signs
<b>Description</b>	The system allows clinicians to record and edit vital signs
<b>From</b>	Clinician
<b>Pre-conditions</b>	Clinician is authenticated
<b>Main scenario</b>	<p>1.Clinician clicks “Record Vitals ” button on vital signs panel      2.System displays vital sign form modal      3.Clinicians selects vital sign type and enters value and unit      4.Clinicians submits form      5.System validates input and authenticates user      6.System creates vital sign record linked to encounter      7.System displays updated vitals grid</p>
<b>Alternative scenario</b>	<p><b>A1: Visit Not In Progress</b></p> <ul style="list-style-type: none"> <li>- if visit is not in progress.</li> <li>-Record button is hidden or disabled</li> </ul> <p><b>A2: Invalid Value</b></p> <ul style="list-style-type: none"> <li>- At step 5,if value is invalid (out of range -wrong format )</li> <li>-System displays validation error</li> </ul> <p><b>A3: Authentication Error</b></p> <ul style="list-style-type: none"> <li>- At step 2,if JWT token is invalid or expired</li> <li>-System redirects to login page</li> </ul>
<b>Post condition</b>	Visits sign is recorded for this visit

Table 31, Use Case Specification (Edit Patient Vital Signs)

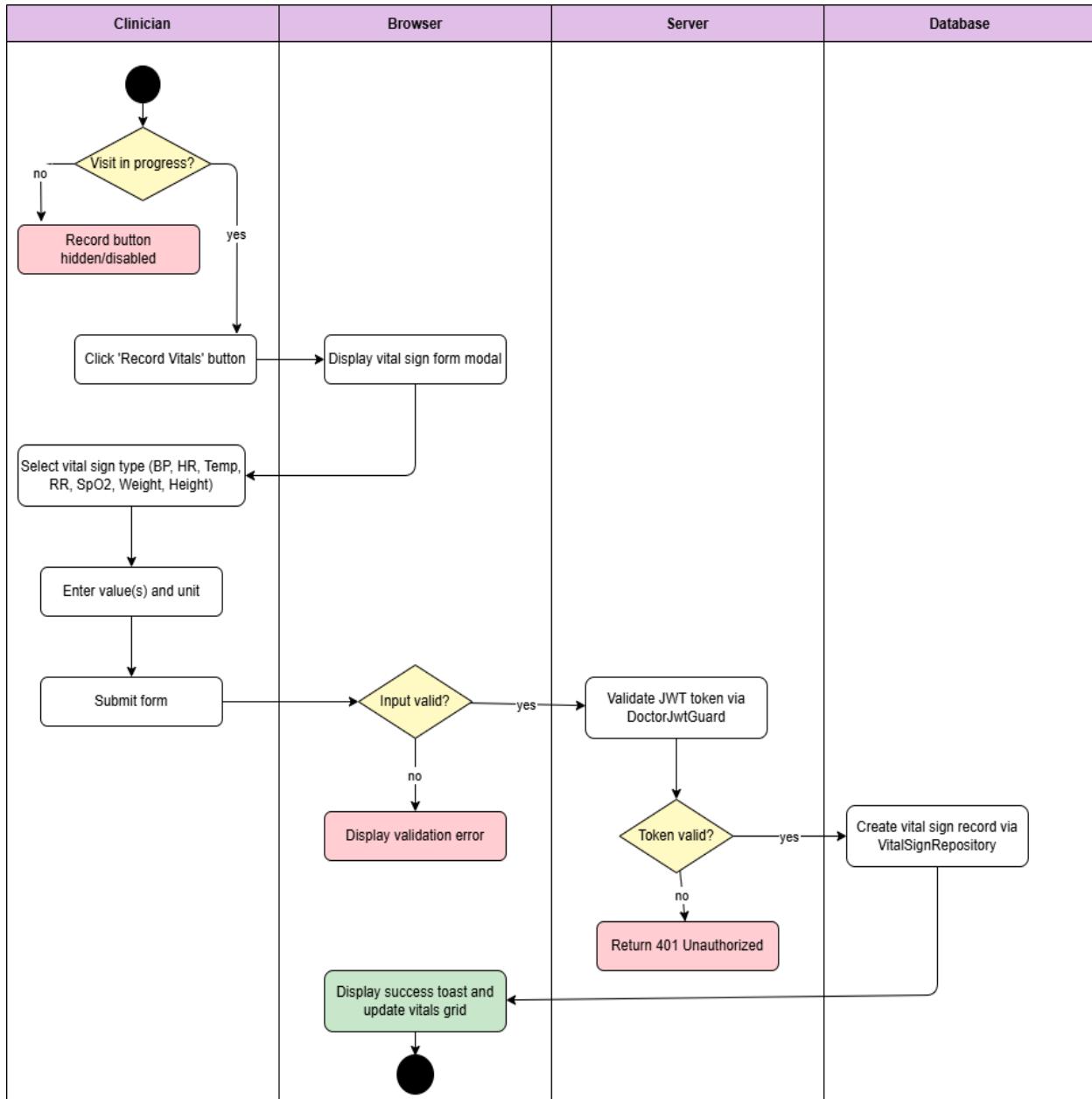


Diagram 50, Activity Diagram (Edit Patient Vital Signs)

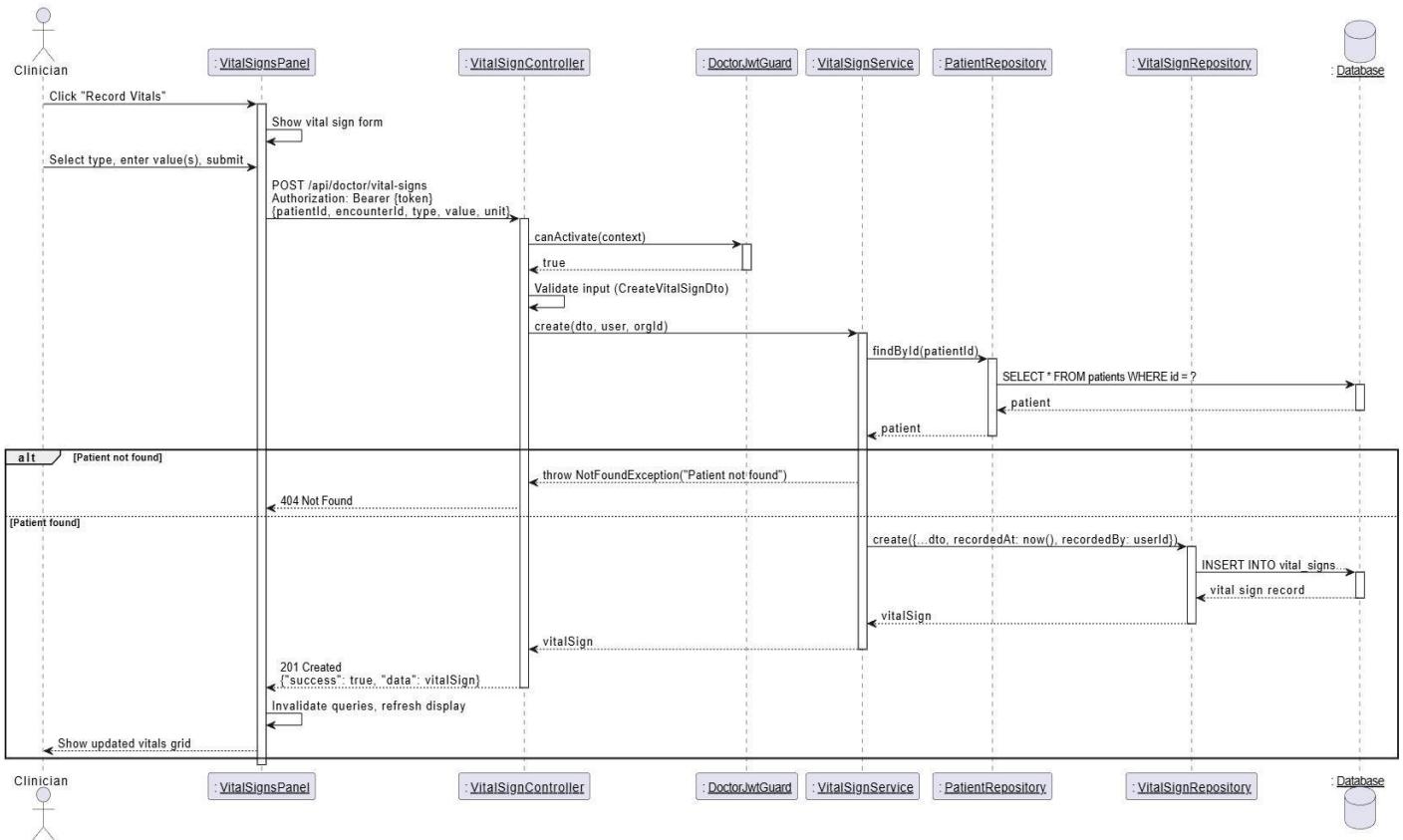


Diagram 51, Sequence Diagram (Edit Patient Vital Signs)

- **View Allergies:**

<b>Use case ID</b>	<b>VEMR-FR-AM-25</b>
<b>Use case name</b>	View Allergies
<b>Description</b>	The system allows clinicians to view a patient recorded allergies with severity
<b>From</b>	Clinician
<b>Pre-conditions</b>	Clinician is authenticated
<b>Main scenario</b>	<p>1.Clinician opens visit details or patient profile page and clicks “Allergies” tab.</p> <p>2.System authenticates user and retrieves patient allergies</p> <p>3.System displays allergies list with allergies,type,severity,reaction</p>
<b>Alternative scenario</b>	<p><b>A1: No Allergies</b></p> <ul style="list-style-type: none"> <li>- At step 2 ,if no allergies recorded for patient</li> <li>-System displays “No allergies recorded “ message</li> </ul> <p><b>A2: Authentication Error</b></p> <ul style="list-style-type: none"> <li>- At step 2,if JWT token is invalid or expired</li> <li>-System redirects to login page</li> </ul>
<b>Post condition</b>	Allergies are displayed in a list format

Table 32, Use Case Specification (View Allergies )

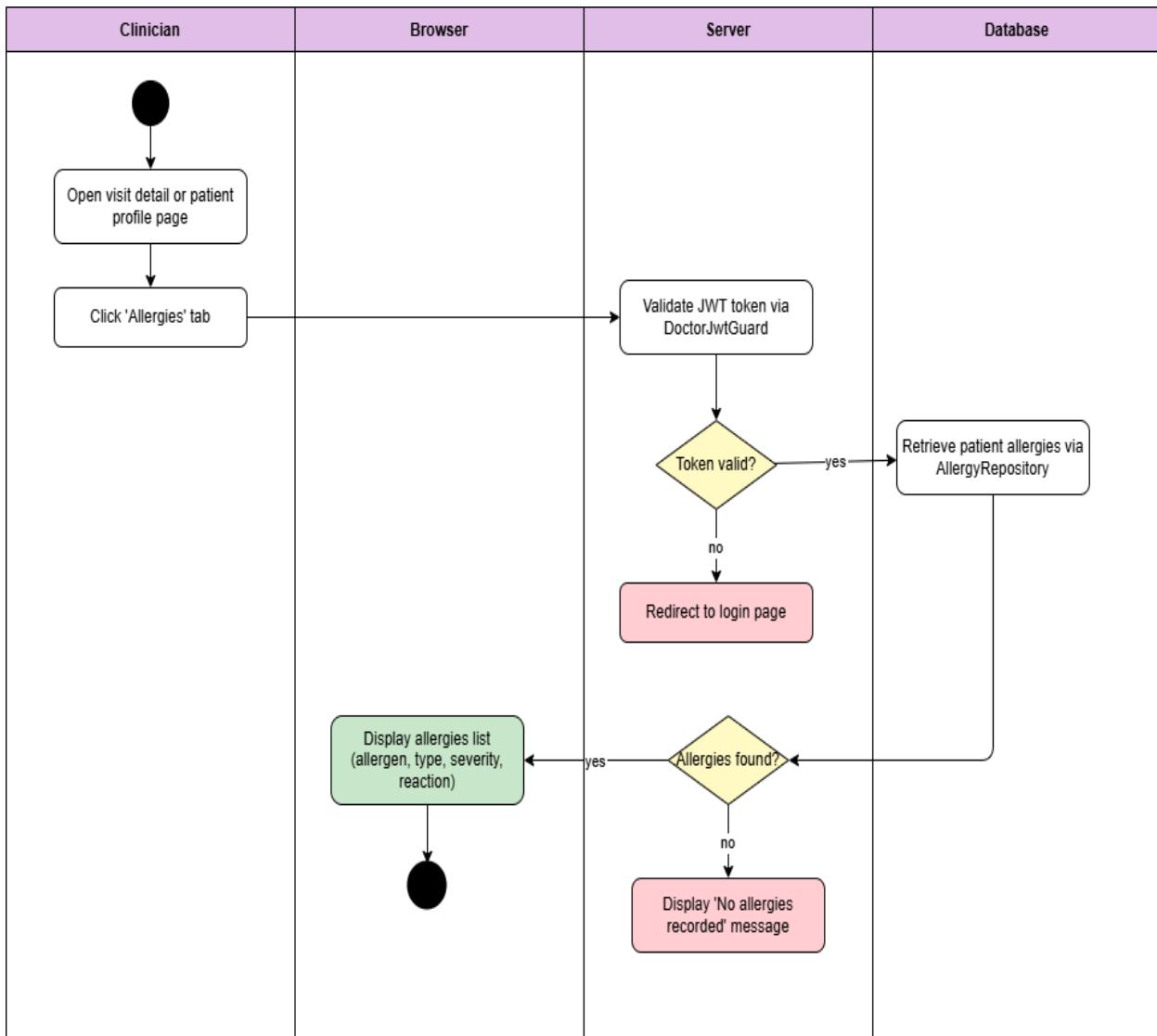


Diagram 52, Activity Diagram (View Allergies )

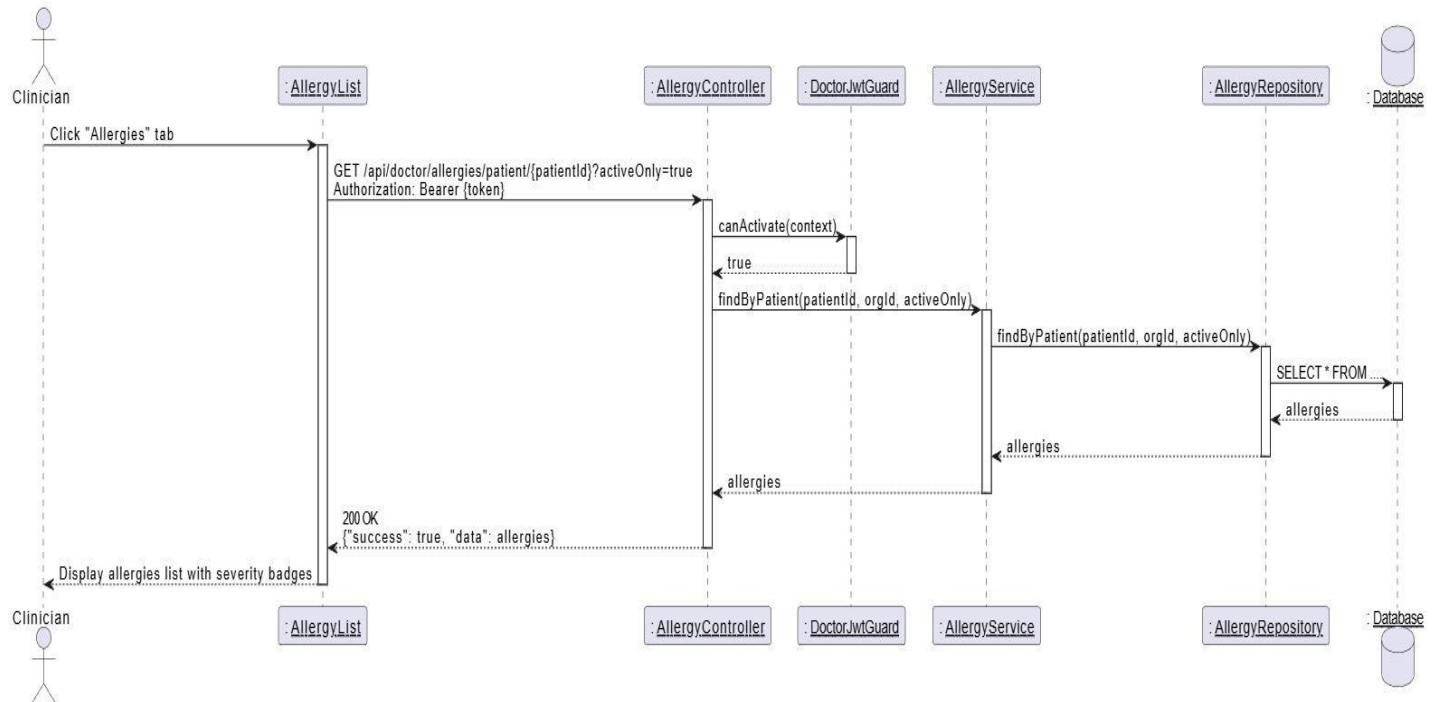


Diagram 53, Sequence Diagram (View Allergies )

## ● Create Allergies :

<b>Use case ID</b>	<b>VEMR-FR-AM-26</b>
<b>Use case name</b>	Create Allergies
<b>Description</b>	The system allows clinicians to add a new allergy record for a patient
<b>From</b>	Clinician
<b>Pre-conditions</b>	Clinician is authenticated
<b>Main scenario</b>	<p>1.Clinician clicks “Add Allergy” button on allergies panel      2.System displays allergy form modal      3.Clinicians enters allergy form details ( type , allergen , severity , reaction , onset date , notes ).      4.Clinicians submits form      5.System validates input, authenticates user , and verifies patient exist.      6.System checks for duplicate allergen and creates allergy record      7.System closes modal and refreshed allergy list</p>
<b>Alternative scenario</b>	<p><b>A1: Duplicate Or Not Found Error</b></p> <ul style="list-style-type: none"> <li>- At step 5 - 6 ,if allergen already exists or patient not found</li> <li>-System displays error message .</li> </ul> <p><b>A2: Visit Not In Progress</b></p> <ul style="list-style-type: none"> <li>- If visit is not in progress</li> <li>-Add button is hidden or disabled</li> </ul> <p><b>A3: Authentication Error</b></p> <ul style="list-style-type: none"> <li>- At step 2,if JWT token is invalid or expired</li> <li>-System redirects to login page</li> </ul>

<b>Post condition</b>	Allergies is created for the patient
-----------------------	--------------------------------------

Table 33, Use Case Specification (Create Allergies )

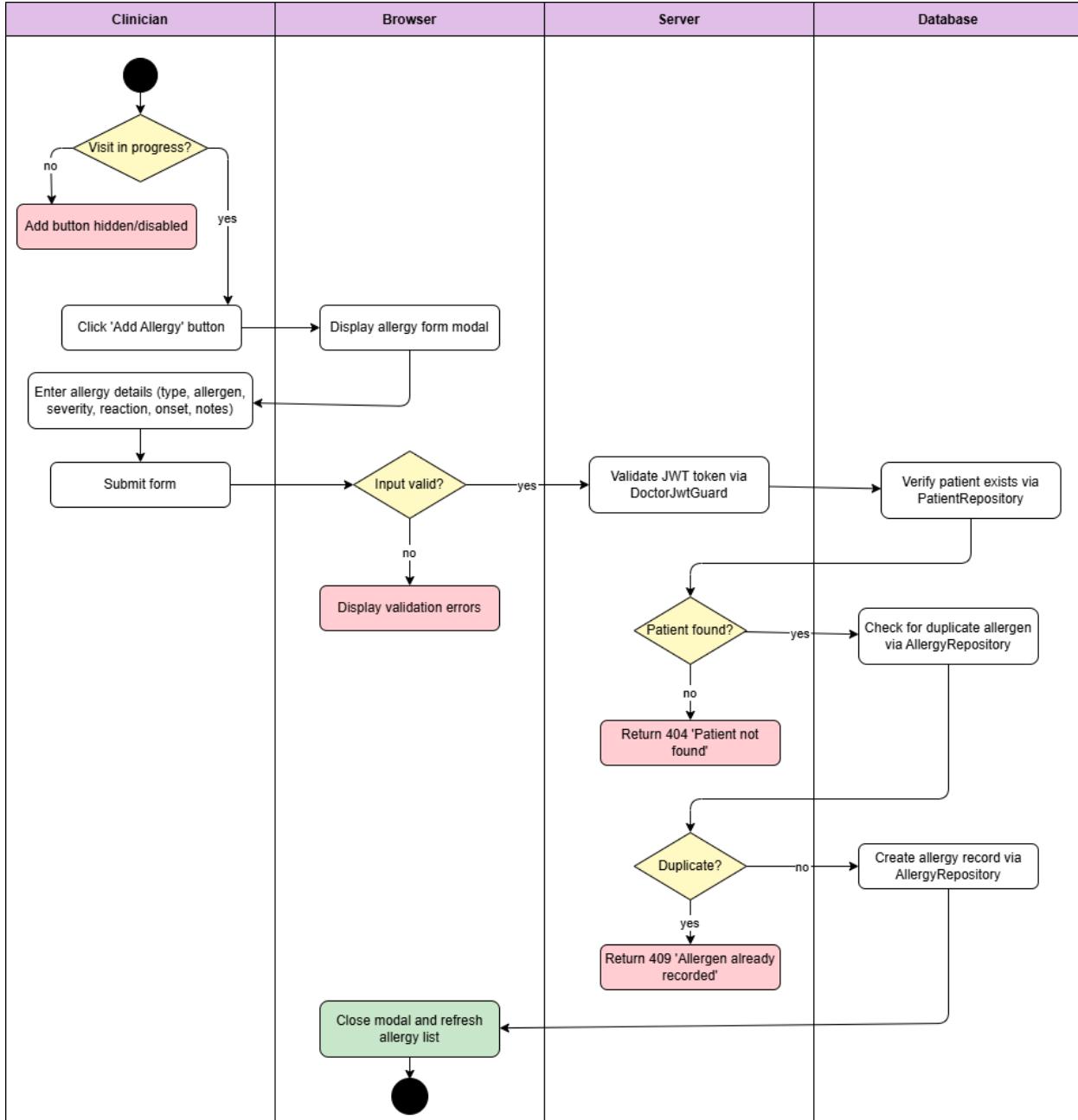


Diagram 54, Activity Diagram (Create Allergies )

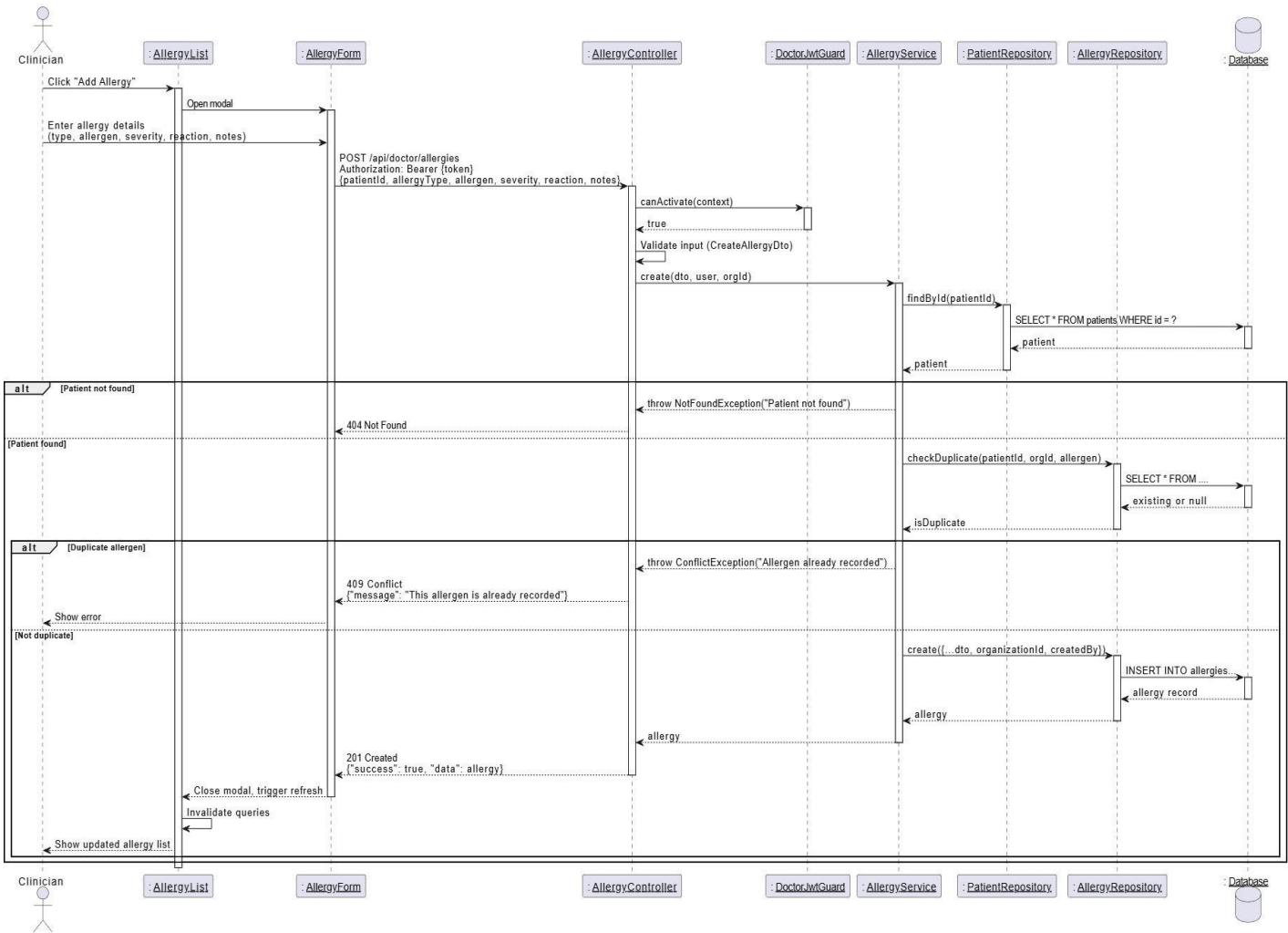


Diagram 55, Sequence Diagram (Create Allergies )

## ● Update Allergies :

<b>Use case ID</b>	<b>VEMR-FR-AM-27</b>
<b>Use case name</b>	Update Allergies
<b>Description</b>	The system allows clinicians to update existing allergy information for a patient
<b>From</b>	Clinician
<b>Pre-conditions</b>	Clinician is authenticated
<b>Main scenario</b>	<p>1.Clinician clicks “Edit” button on allergy in the list      2.System displays allergy from modal with existing data      3.Clinicians modifies allergy details and submit form      4.System validates input, authenticates user , and retrieves existing allergy      5.System checks for duplicate if allergen changed and updates allergy record      6.System closes modal and refreshed allergy list</p>
<b>Alternative scenario</b>	<p><b>A1: Duplicate Or Not Found Error</b></p> <ul style="list-style-type: none"> <li>- At step 4 - 5 ,if new allergen already exists or allergy not found</li> <li>-System displays error message .</li> </ul> <p><b>A2: Authentication Error</b></p> <ul style="list-style-type: none"> <li>- At step 4,if JWT token is invalid or expired</li> <li>-System redirects to login page</li> </ul>
<b>Post condition</b>	Allergies is updated

Table 34, Use Case Specification (Update Allergies )

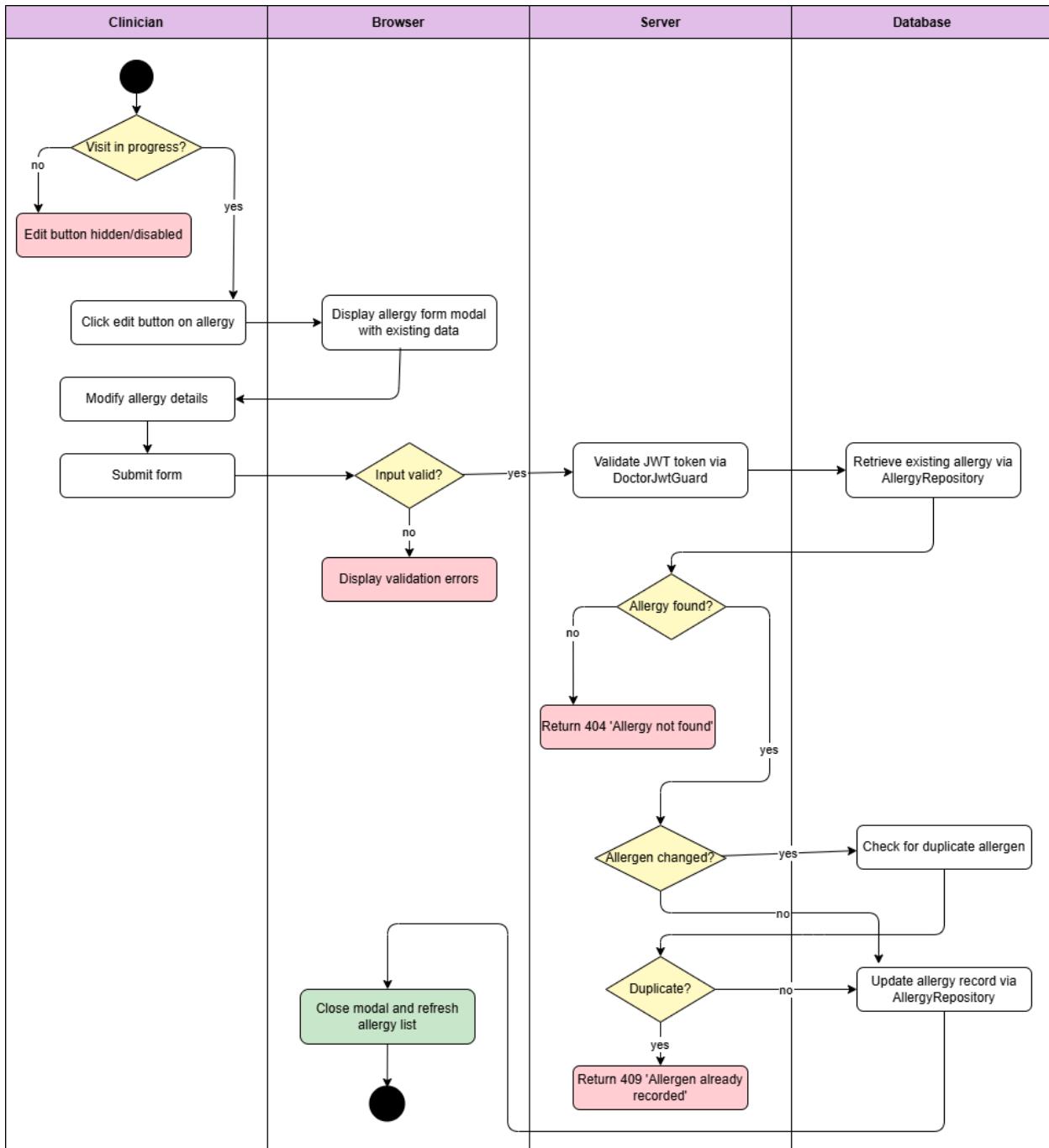


Diagram 56, ActivityDiagram (Update Allergies )

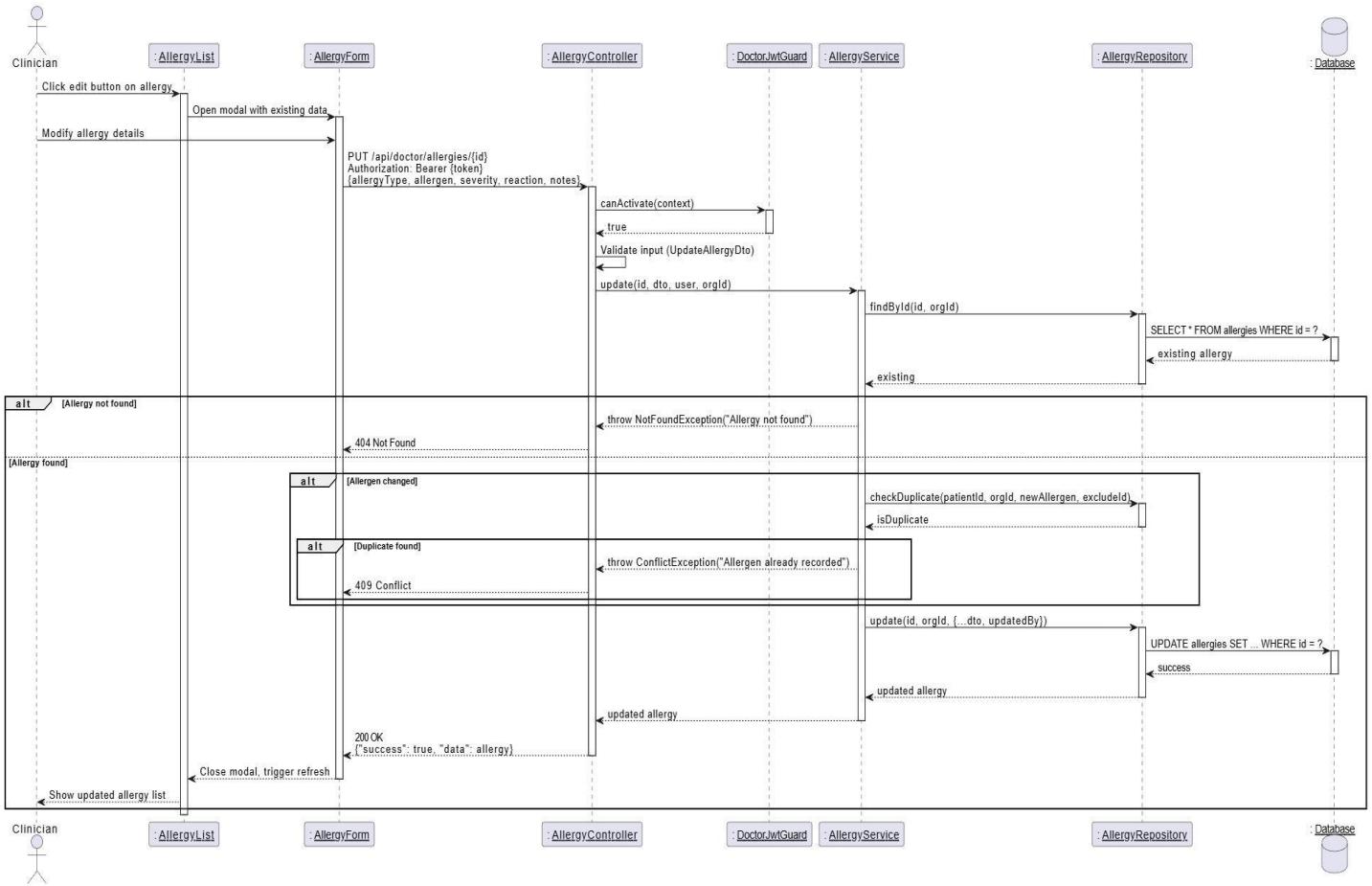


Diagram 57, Sequence Diagram (Update Allergies )

- **Delete Allergies :**

<b>Use case ID</b>	<b>VEMR-FR-AM-028</b>
<b>Use case name</b>	Delete Allergies
<b>Description</b>	The system allows clinicians to delete an allergy record from a patient
<b>From</b>	Clinician
<b>Pre-conditions</b>	Clinician is authenticated
<b>Main scenario</b>	<p>1.Clinician clicks “Delete ” button on allergy in the list      2.System prompts for configuration dialog      3.Clinicians confirms deletion      4.System authenticates user and retrieves allergy      5.System soft-deletes allergy record ( set deleteAt , deleteBy )      6.System displays success message and refreshes allergy list</p>
<b>Alternative scenario</b>	<p><b>A1: Allergy Not Found Error</b></p> <ul style="list-style-type: none"> <li>- At step 4 ,if allergy does not exist</li> <li>-System displays error message .</li> </ul> <p><b>A2: Authentication Error</b></p> <ul style="list-style-type: none"> <li>- At step 4,if JWT token is invalid or expired</li> <li>-System redirects to login page</li> </ul> <p><b>A3: User Cancel</b></p> <ul style="list-style-type: none"> <li>- At step 3, if clinician cancels confirmation</li> <li>-No action is taken</li> </ul>

<b>Post condition</b>	Allergies is soft-deleted(deletedAt is set )
-----------------------	--

Table 35, Use Case Specification (Delete Allergies )

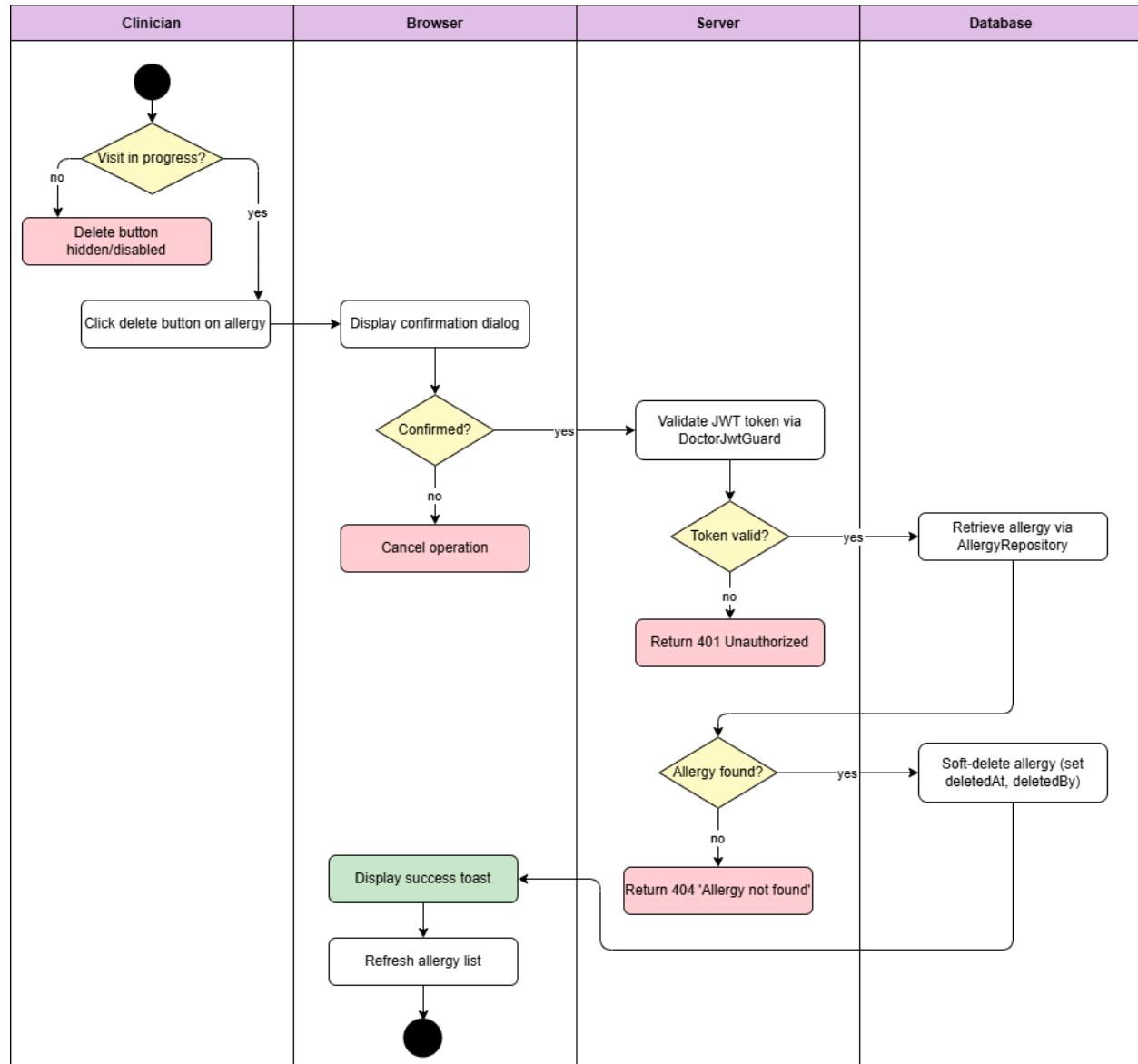


Diagram 58, ActivityDiagram (Delete Allergies )

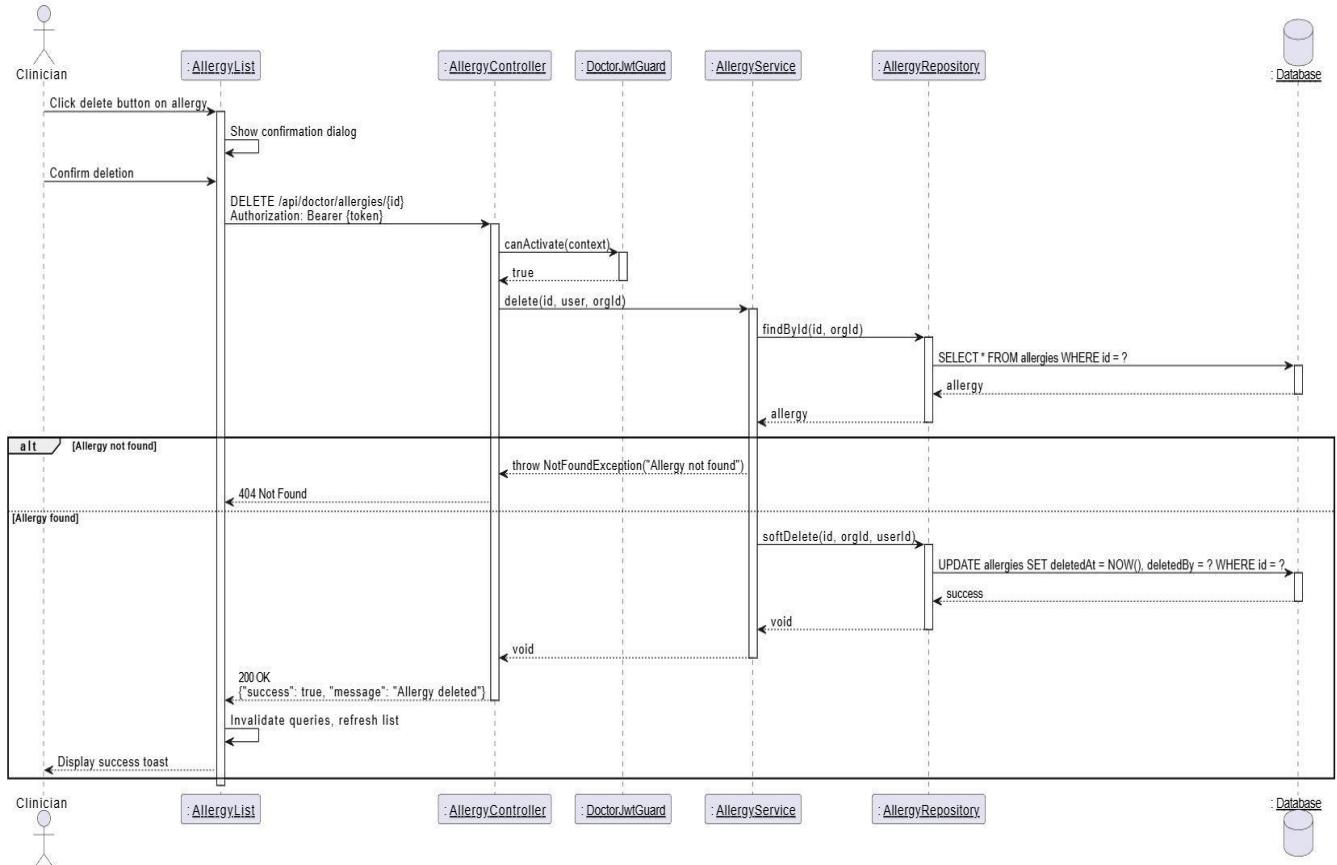


Diagram 59, Sequence Diagram (Delete Allergies )

- **Create Clinicians Account:**

<b>Use case ID</b>	<b>VEMR-FR-AM-29</b>
<b>Use case name</b>	Create clinicians account
<b>Description</b>	The system allows administrators to create new clinicians account
<b>From</b>	Admin
<b>Pre-conditions</b>	Admin is authenticated
<b>Main scenario</b>	<p>1.Admin navigates to doctor management page      2.Admin click “Add Doctor” button      3.System displays CreateDoctorForm modal      4.Admin enters doctor details      5.Admin submits the form      6.System validates input and check email uniqueness      7.System hashes password and creates doctor account with status “inactive”      8.System displays success message</p>
<b>Alternative scenario</b>	<p><b>A1: Validation Error</b></p> <ul style="list-style-type: none"> <li>- At step 6 ,if required fields are missing or email format invalid        -System displays error message on form</li> </ul> <p><b>A2: Authentication Error</b></p> <ul style="list-style-type: none"> <li>- At step 6 ,if JWT token is invalid or expired        -System redirects to login page</li> </ul> <p><b>A3: Email Already Exist</b></p> <ul style="list-style-type: none"> <li>- At step 6, if email is already registered</li> </ul>

	-System displays “Email already exist” error
<b>Post condition</b>	New doctor account is created with status

Table 36, Use Case Specification (Create Clinicians Account )

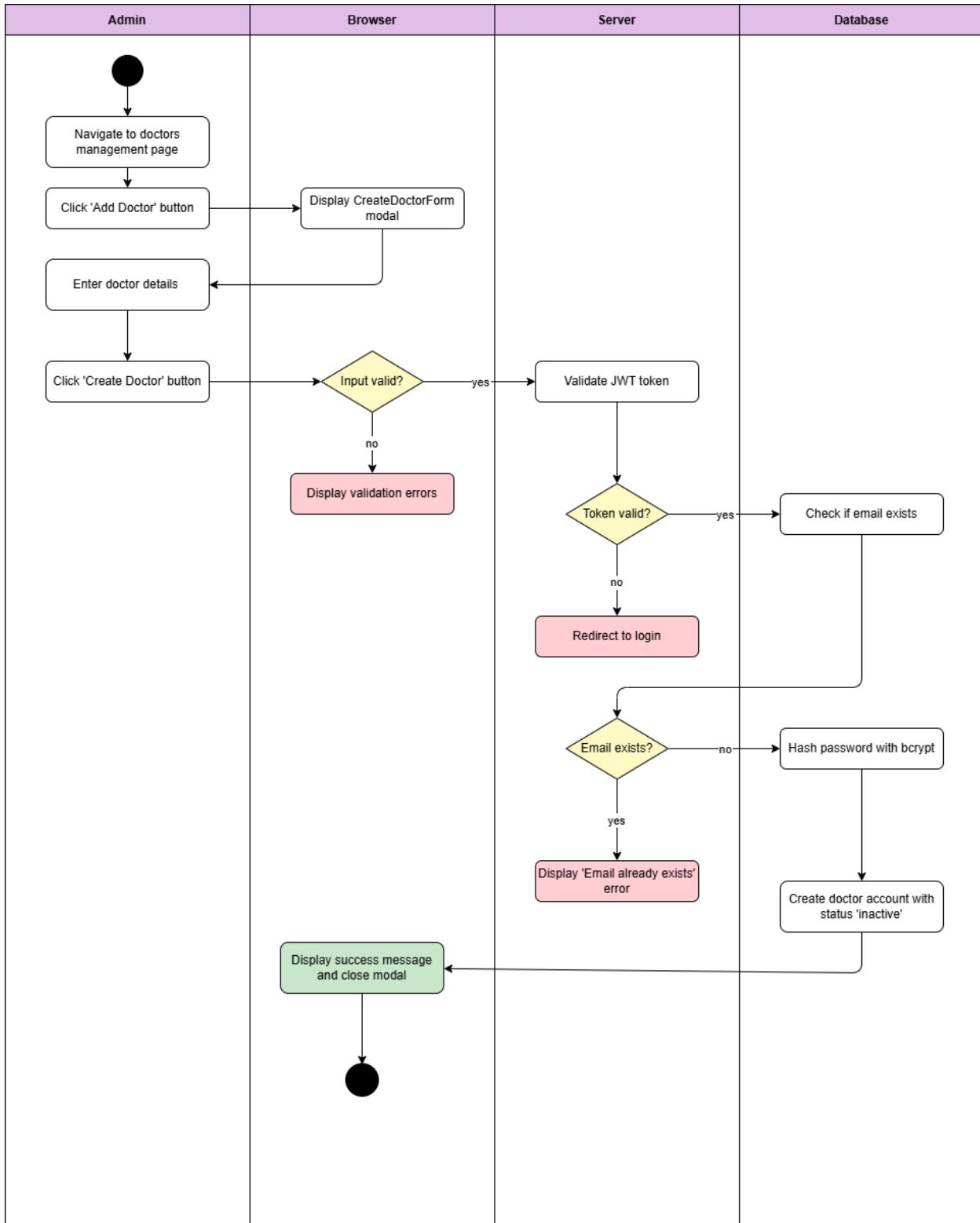


Diagram 60, Activity Diagram (Create Clinicians Account )

## Chapter 4 - System Analysis

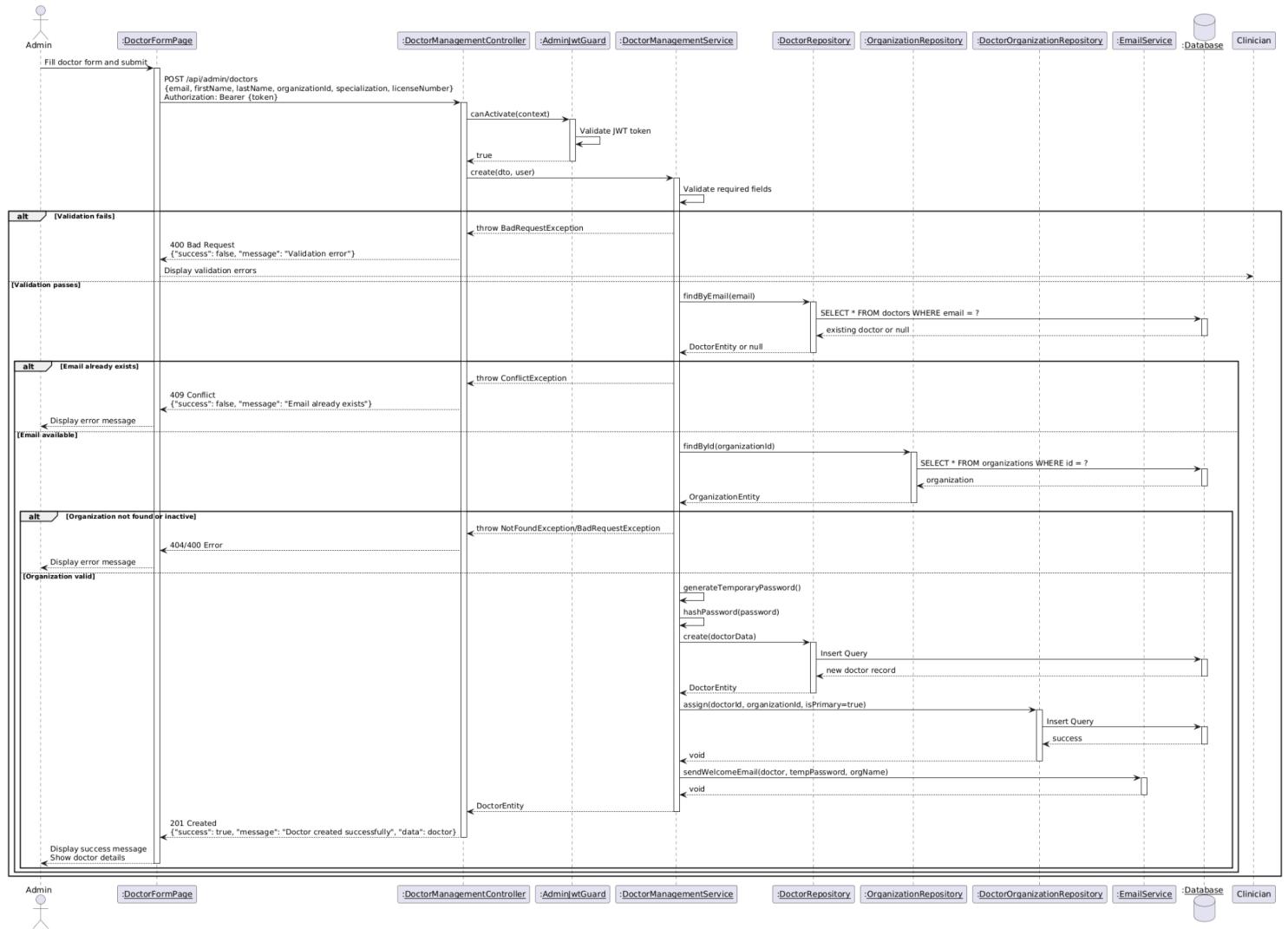


Diagram 61, Sequence Diagram (Create Clinicians Account )

## ● Update Clinicians Account:

<b>Use case ID</b>	<b>VEMR-FR-AM-30</b>
<b>Use case name</b>	Update clinicians account
<b>Description</b>	The system allows administrators to update existing clinicians account
<b>From</b>	Admin
<b>Pre-conditions</b>	Admin is authenticated
<b>Main scenario</b>	<p>1.Admin navigates to doctor management page      2.Admin selects a doctor from the list      3.Admin click "Edit" button      4.System displays EditDoctorForm modal with current doctor information      5.Admin modifies doctor details      6.Admin submits the form      7.System validates input      8.System updates doctor account information      9.System displays success message and refreshes doctor list</p>
<b>Alternative scenario</b>	<p><b>A1: Validation Error</b></p> <ul style="list-style-type: none"> <li>- At step 7 ,if required fields are missing or email format invalid</li> <li>-System displays error message on form</li> </ul> <p><b>A2: Authentication Error</b></p> <ul style="list-style-type: none"> <li>- At step 7 ,if JWT token is invalid or expired</li> <li>-System redirects to login page</li> </ul> <p><b>A3: Doctor Not Found</b></p> <ul style="list-style-type: none"> <li>- At step 4, if doctor ID is invalid</li> <li>-System displays “Doctor Not Found” error</li> </ul>
<b>Post condition</b>	Doctor account information is updated

Table 37, Use Case Specification (Update Clinicians Account )

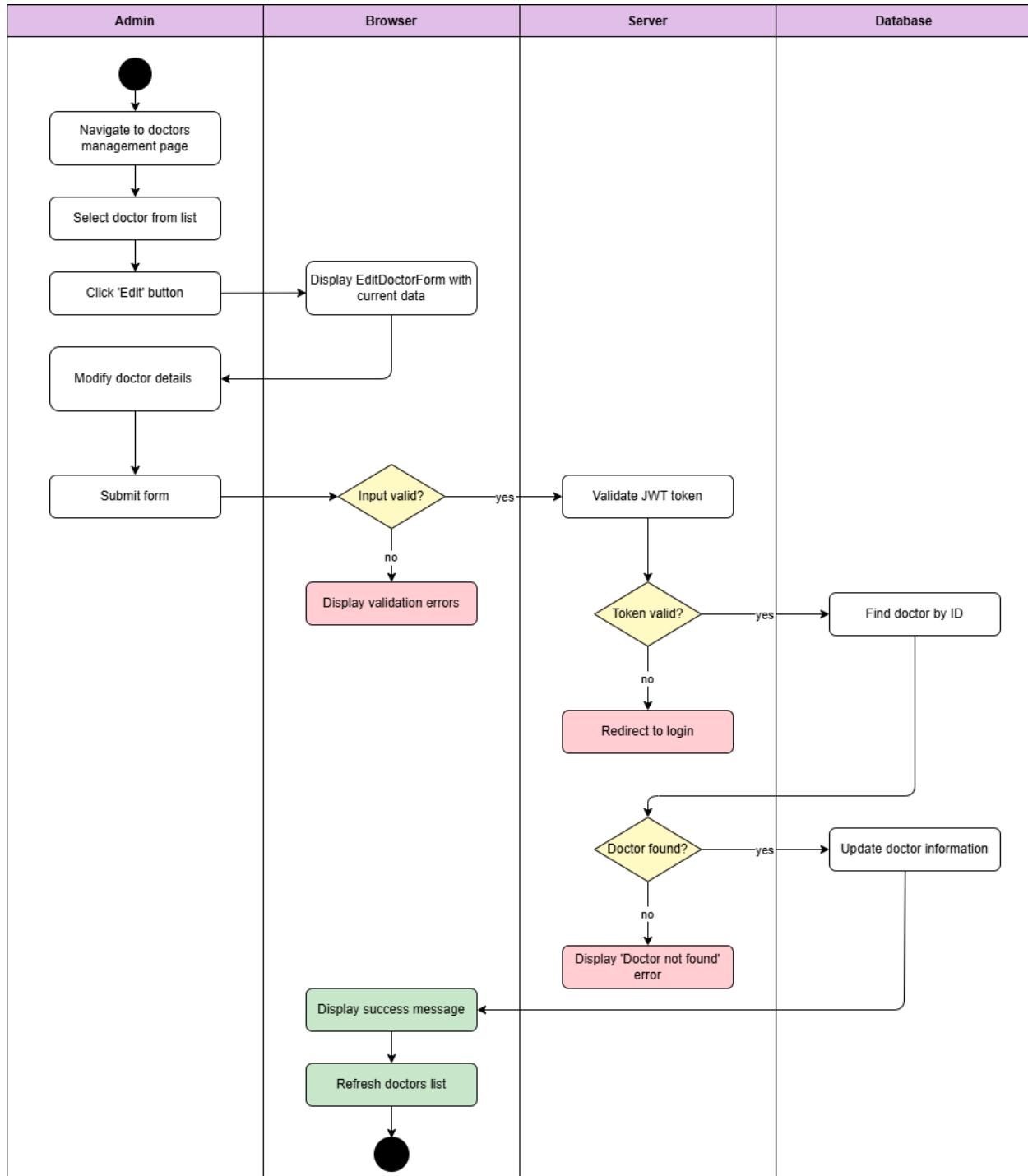


Diagram 62, Activity Diagram (Update Clinicians Account )

## Chapter 4 - System Analysis

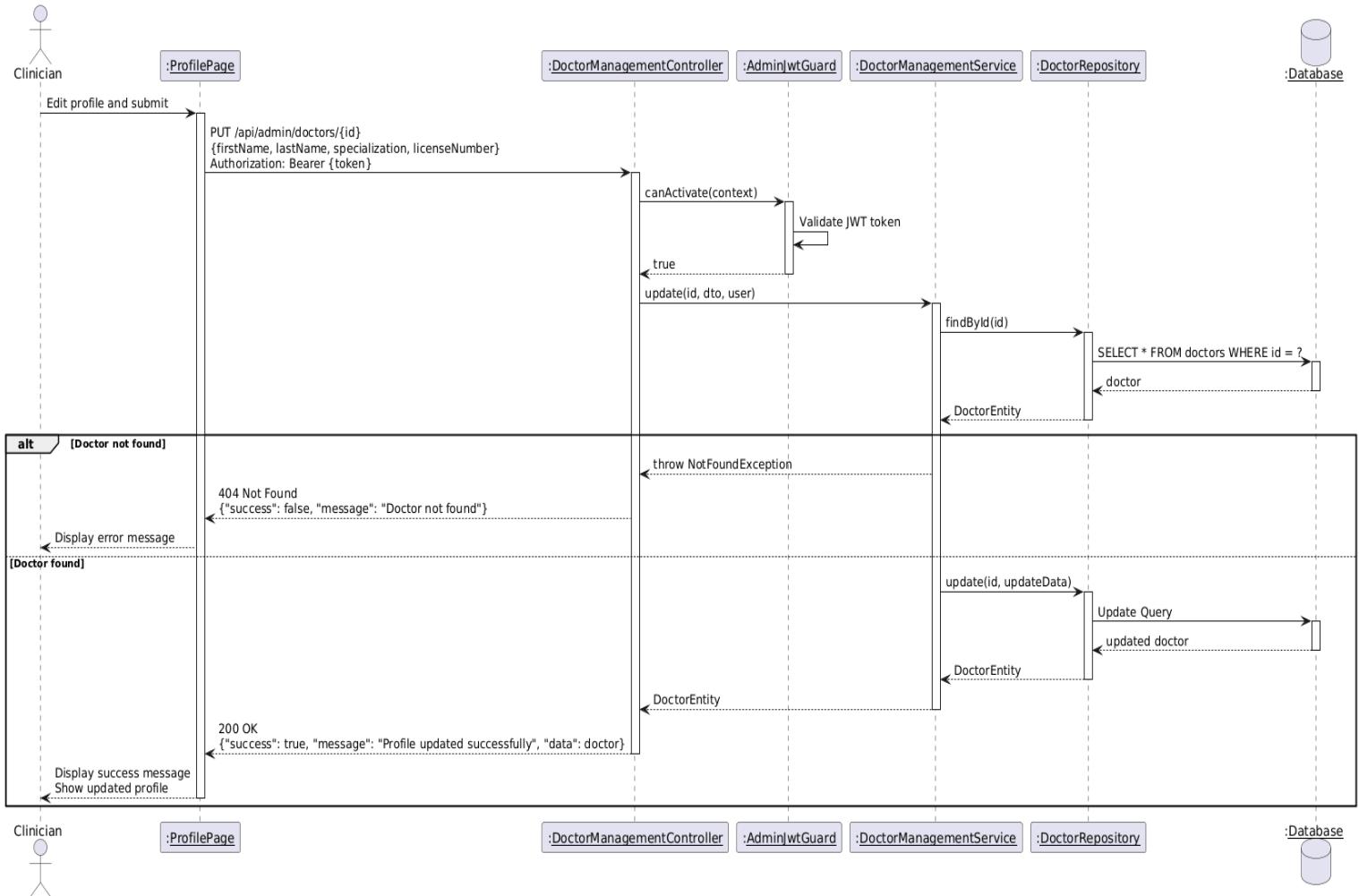


Diagram 63, Sequence Diagram (Update Clinicians Account )

## ● View Clinicians Account List :

<b>Use case ID</b>	<b>VEMR-FR-AM-31</b>
<b>Use case name</b>	View clinicians account List
<b>Description</b>	The system allows administrators to view a paginated list of all clinicians Account with search and filter capabilities
<b>From</b>	Admin
<b>Pre-conditions</b>	Admin is authenticated
<b>Main scenario</b>	<p>1.Admin navigates to doctor management page      2.System displays list of all doctor with :      ( Name - Email - Specialization - Account Status - Organization )      3.Admin can:</p> <ul style="list-style-type: none"> <li>- Search by name , email , or specialization .</li> <li>- Filter by account status ( active , inactive , suspended )</li> <li>- Filter by organization</li> <li>- Sort by any column</li> <li>- Navigate through pages</li> </ul>
<b>Alternative scenario</b>	<p><b>A1: No Doctors Found</b></p> <ul style="list-style-type: none"> <li>- At step 2 ,if no doctors match search/filter criteria</li> <li>-System displays “No doctor found” message</li> </ul> <p><b>A2: Authentication Error</b></p> <ul style="list-style-type: none"> <li>- At step 7 ,if JWT token is invalid or expired</li> <li>-System redirects to login page</li> </ul>
<b>Post condition</b>	Doctor list is displayed with current data

Table 38, Use Case Specification (View Clinicians Account List )

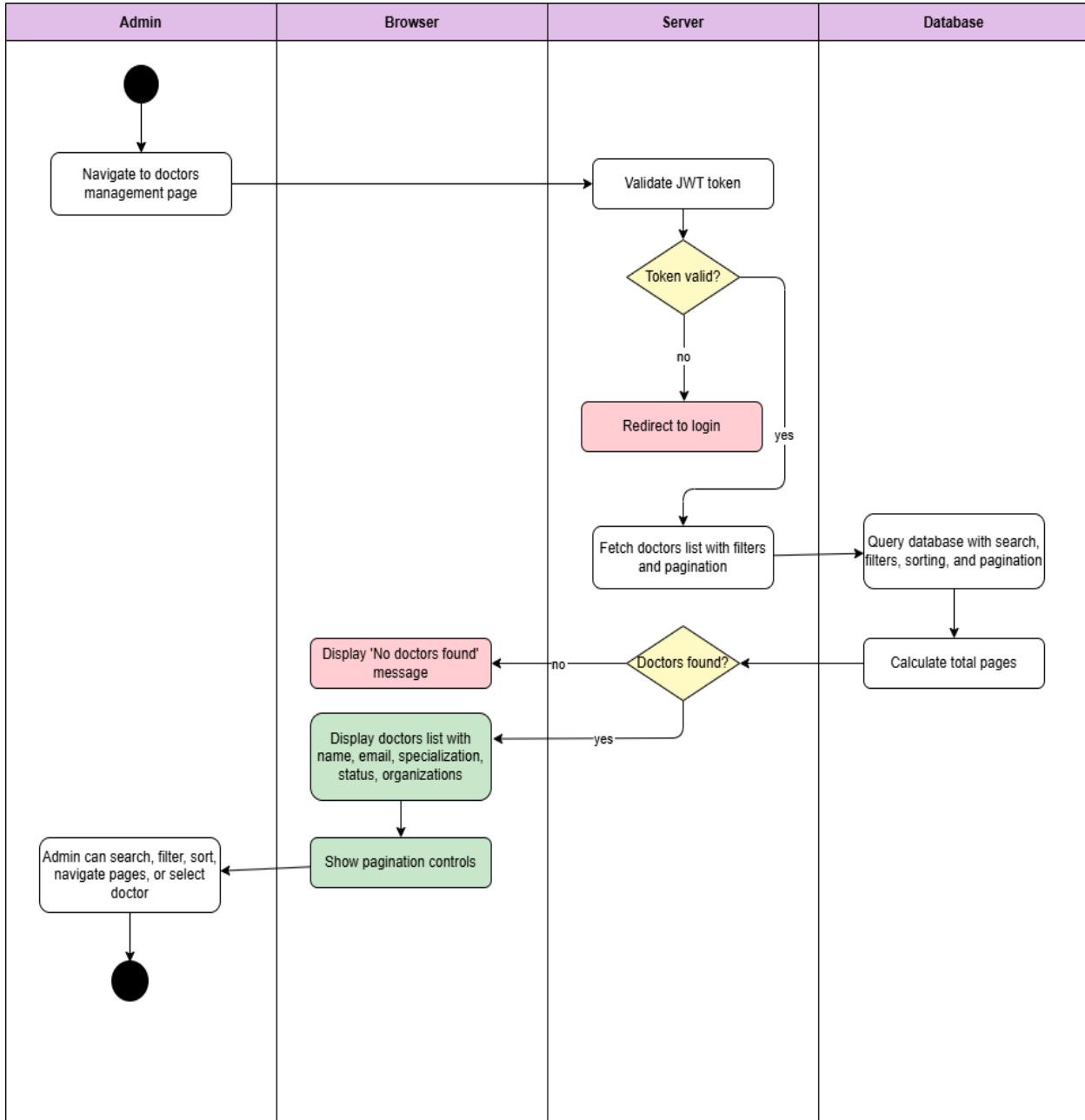


Diagram 64, Activity Diagram (View Clinicians Account List )

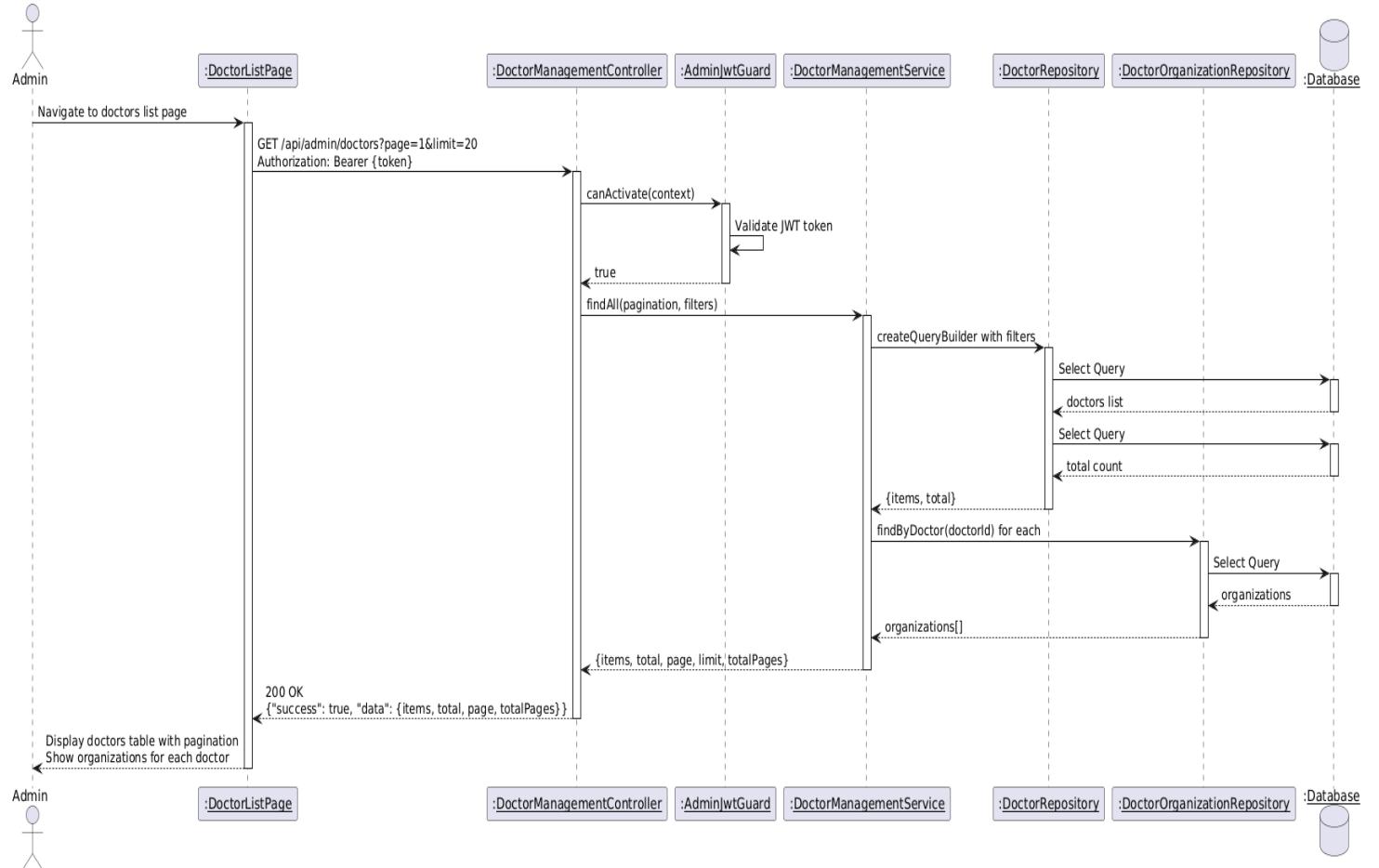


Diagram 65, Sequence Diagram (View Clinicians Account List )

- View Clinicians Account Details:

<b>Use case ID</b>	<b>VEMR-FR-AM-32</b>
<b>Use case name</b>	View clinicians account Details
<b>Description</b>	The system allows administrators to view details information about a specific clinician account
<b>From</b>	Admin
<b>Pre-conditions</b>	Admin is authenticated
<b>Main scenario</b>	<p>1.Admin navigates to doctor management page      2.Admin select a doctor from the list      3.Admin click “View Details” button or click on doctor name      4.System displays doctor details page with :</p> <ul style="list-style-type: none"> <li>- Personal information</li> <li>- Account status</li> <li>- Organization</li> <li>- Account activity</li> <li>- Statistics</li> </ul>
<b>Alternative scenario</b>	<p><b>A1: Doctor Not Found</b></p> <ul style="list-style-type: none"> <li>- At step 4 ,if doctor ID is invalid</li> <li>-System displays “Doctor Not Found” error and redirects to list</li> </ul> <p><b>A2: Authentication Error</b></p> <ul style="list-style-type: none"> <li>- At step 4 ,if JWT token is invalid or expired</li> <li>-System redirects to login page</li> </ul>
<b>Post condition</b>	Doctor details are displayed

Table 39, Use Case Specification (View Clinicians Account Details)

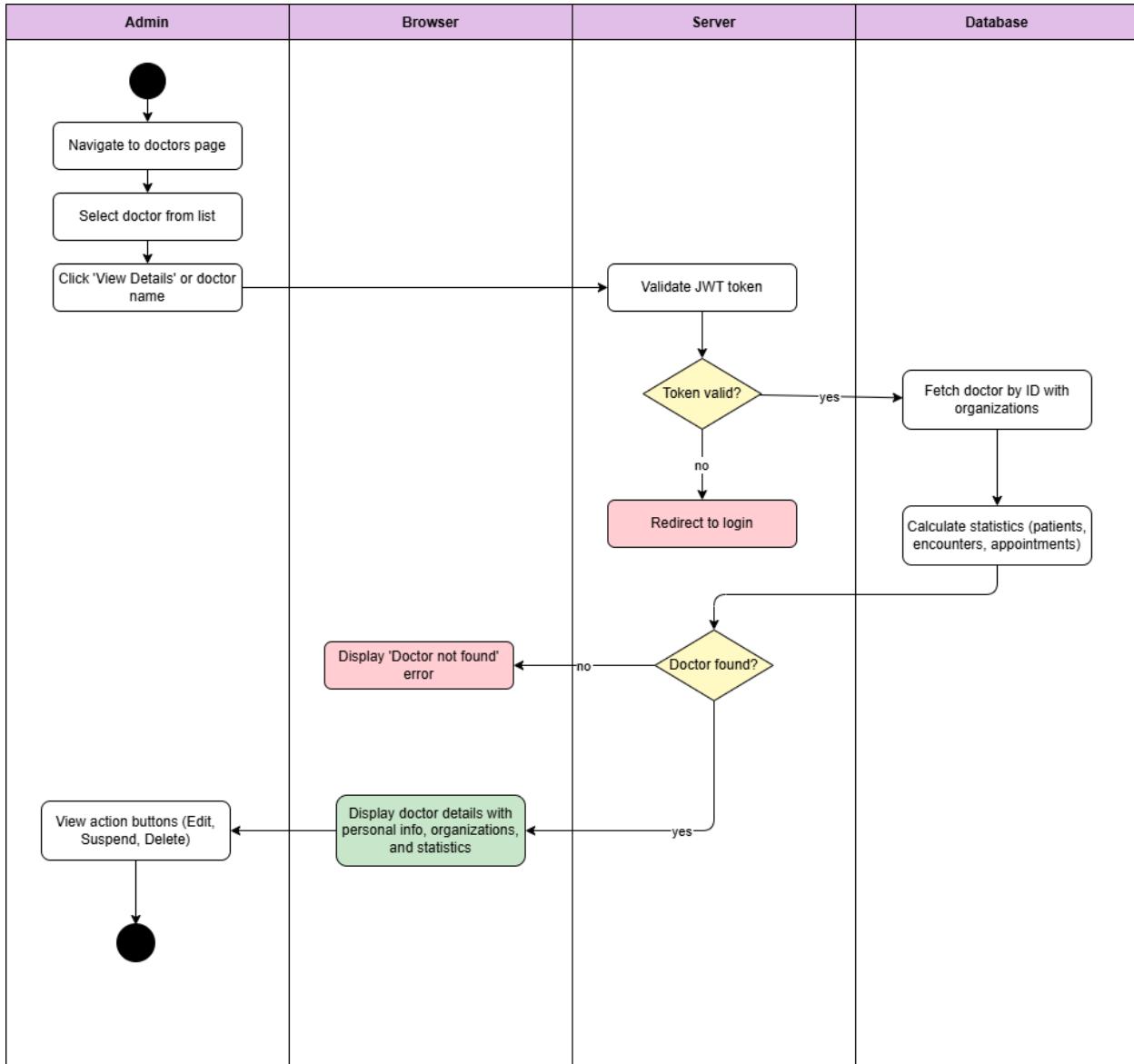


Diagram 66, Activity Diagram (View Clinicians Account Details )

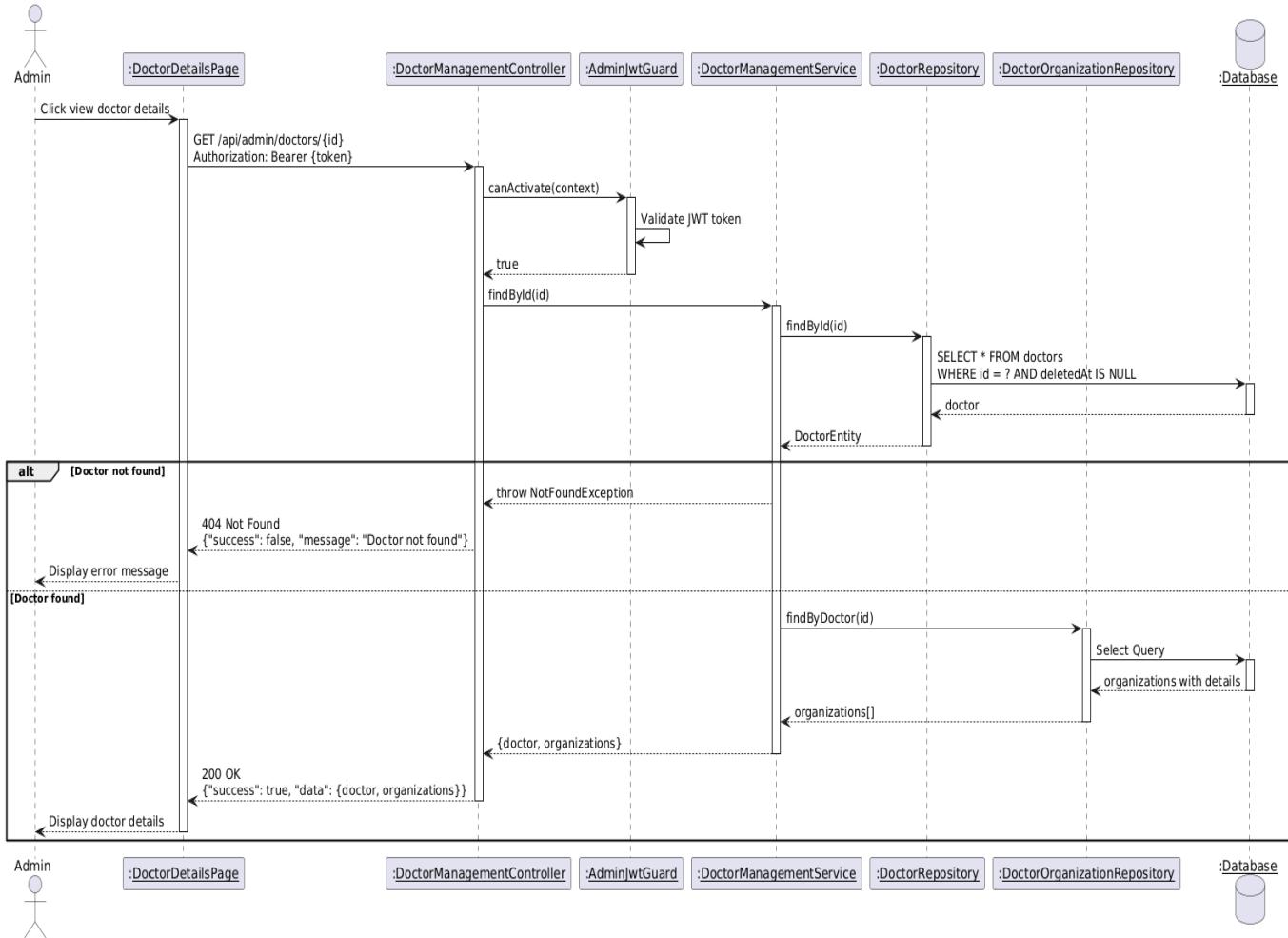


Diagram 67, Sequence Diagram (View Clinicians Account List )

## ● Delete Clinicians Account:

<b>Use case ID</b>	VEMR-FR-AM-33
<b>Use case name</b>	Delete clinicians account
<b>Description</b>	The system allows administrators to delete a clinician account
<b>From</b>	Admin
<b>Pre-conditions</b>	Admin is authenticated
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1.Admin navigates to doctor management page</li> <li>2.Admin selects a doctor from the list</li> <li>3.Admin click "Delete" button</li> <li>4.System displays confirmation dialog with warning message</li> <li>5.Admin confirm deletion</li> <li>6.System validates that doctor has no active appointments</li> <li>7.System performs soft delete ( Set deletedAt timestamp )</li> <li>8.System displays success message</li> <li>9.Doctor is removed from the list</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Doctor Has Active Appointments</b></p> <ul style="list-style-type: none"> <li>- At step 6 ,if doctor has upcoming or in-progress appointments</li> <li>-System displays error : “Cannot delete doctor with active appointments”</li> </ul> <p><b>A2: Doctor Not Found</b></p> <ul style="list-style-type: none"> <li>- At step 6, if Doctor ID is invalid</li> <li>-System displays “Doctor Not Found “ error</li> </ul> <p><b>A3: Authentication Error</b></p> <ul style="list-style-type: none"> <li>- At step 6 ,if JWT token is invalid or expired</li> <li>-System redirects to login page</li> </ul>
<b>Post condition</b>	Doctor account is soft deleted

Table 40, Use Case Specification (Delete Clinicians Account)

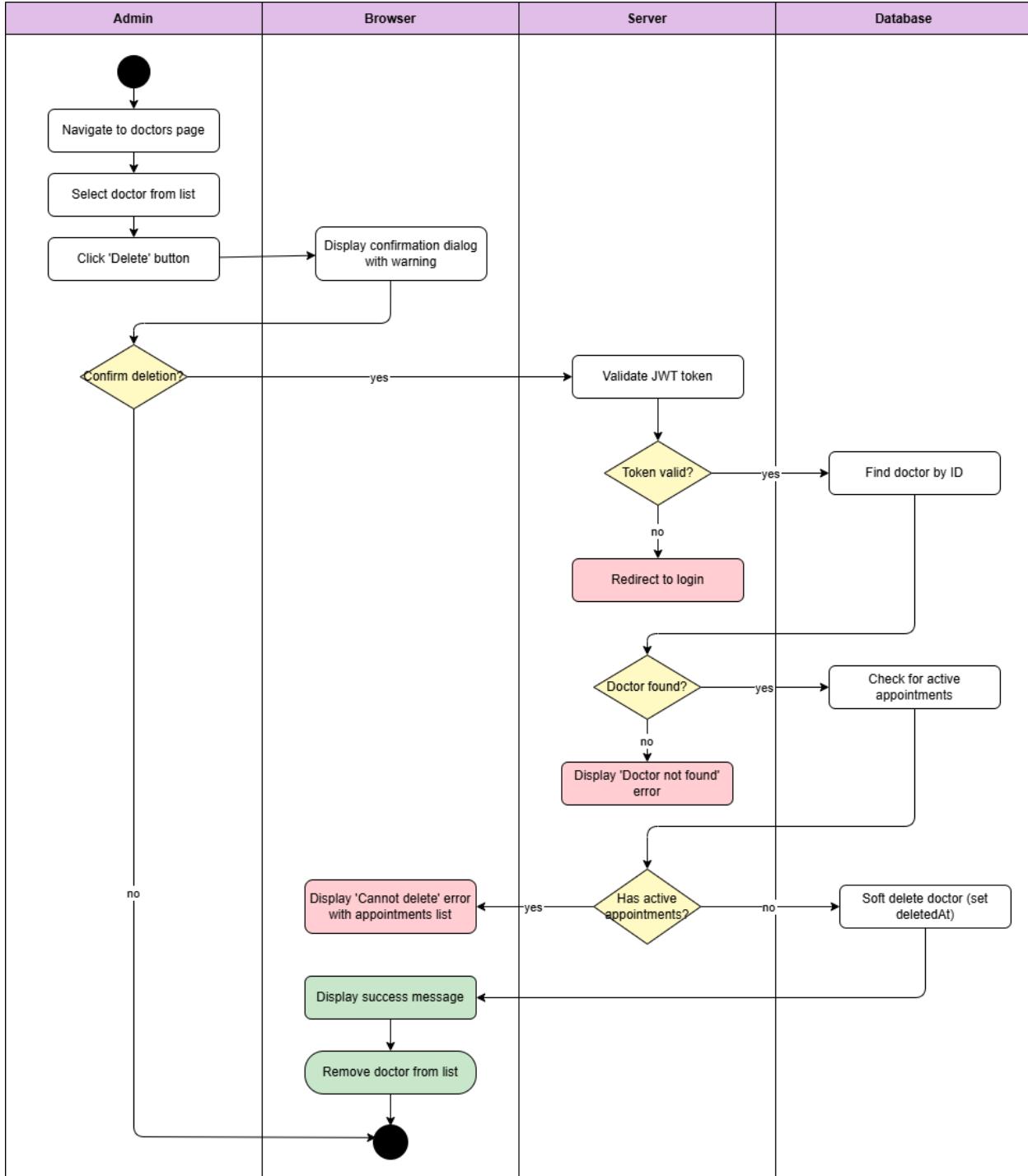


Diagram 68, Activity Diagram (Delete Clinicians Account )

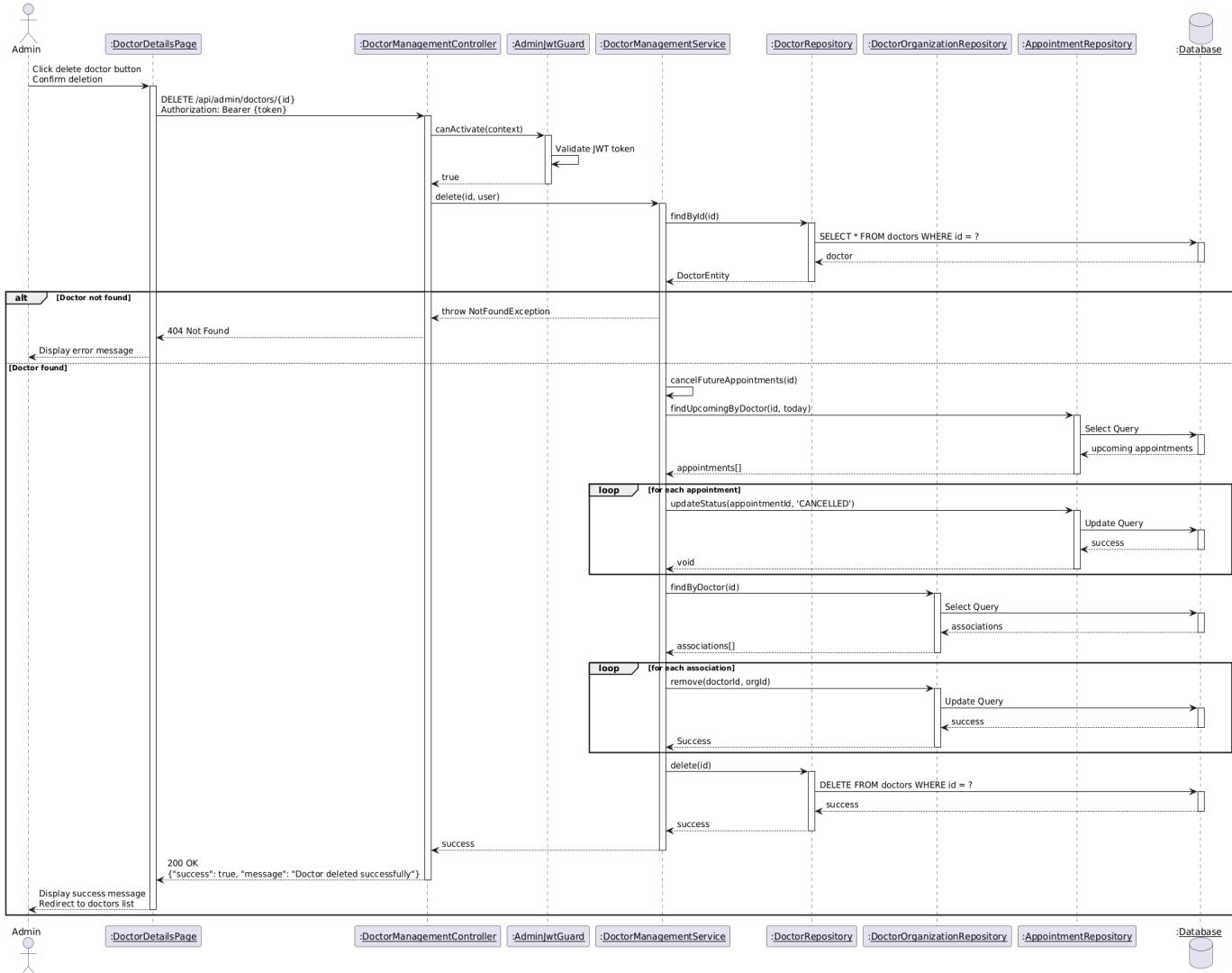


Diagram 69, Sequence Diagram (View Clinicians Account List )

- **Search Clinicians Account:**

<b>Use case ID</b>	<b>VEMR-FR-AM-34</b>
<b>Use case name</b>	Search clinicians account
<b>Description</b>	The system allows administrators to search for clinician accounts by name or email.
<b>From</b>	Admin
<b>Pre-conditions</b>	Admin is authenticated
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>Administrator navigates to clinicians management page</li> <li>Administrator enters search query in search box</li> <li>System debounces input (300ms delay)</li> <li>System validates JWT token via AdminJwtGuard</li> <li>System searches clinicians by first name, last name, or email via DoctorRepository</li> <li>System returns paginated search results</li> <li>System displays matching clinician accounts in table</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Unauthorized Access</b></p> <ul style="list-style-type: none"> <li>- At step 4, if JWT token is invalid or expired</li> <li>-System returns 401 Unauthorized</li> </ul> <p><b>A2: No Results Found</b></p> <ul style="list-style-type: none"> <li>- At step 5, if no clinicians match the search query</li> <li>-System displays "No clinicians found" message</li> </ul> <p><b>A3: Empty Search Query</b></p> <ul style="list-style-type: none"> <li>- At step 3, if search query is empty or cleared</li> <li>-System returns all clinicians with pagination</li> </ul>

<b>Post condition</b>	Administrator can view search results with pagination
-----------------------	---

Table 41, Use Case Specification (Search Clinicians Account)

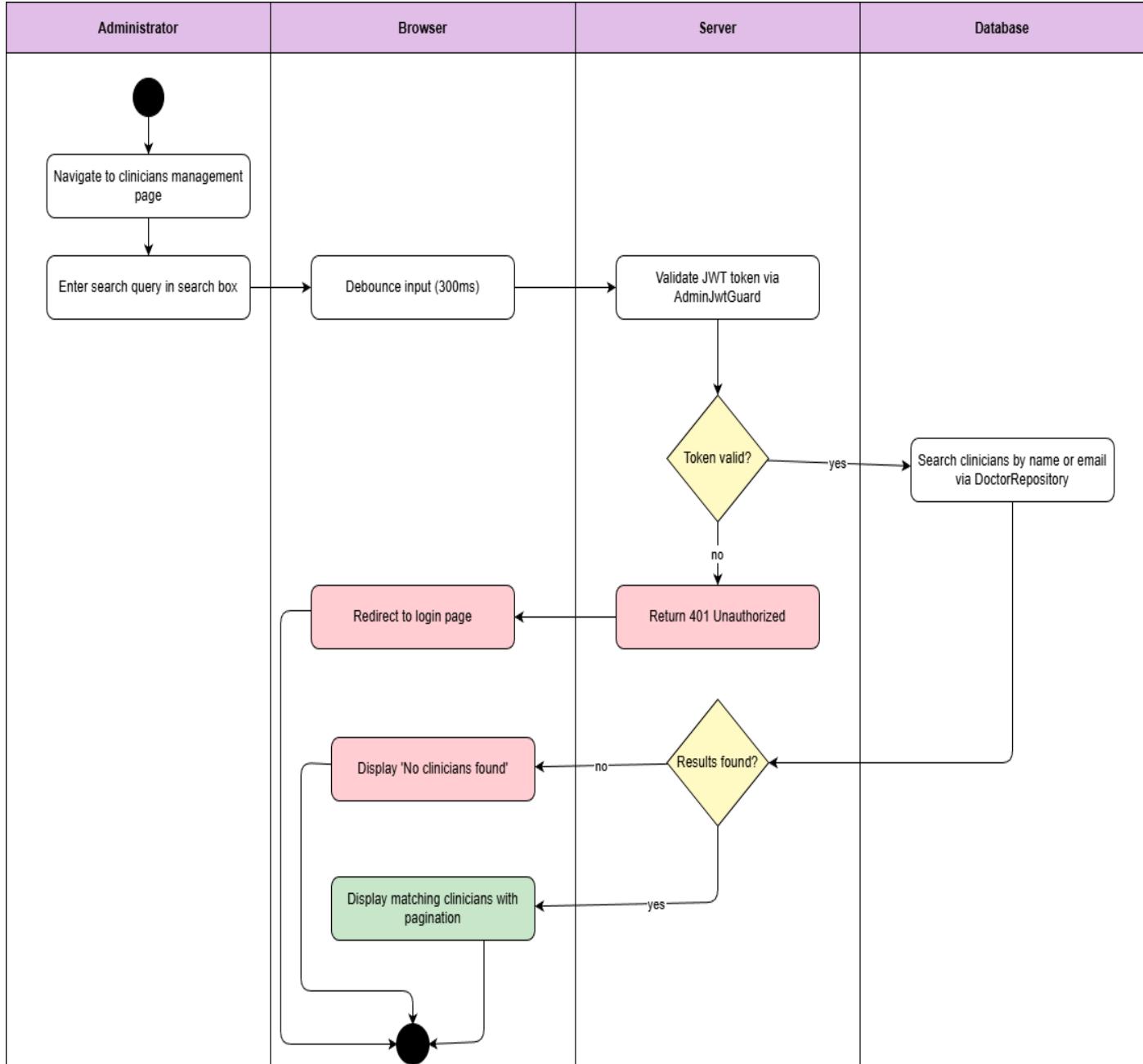


Diagram 70, Activity Diagram (Search Clinicians Account List )

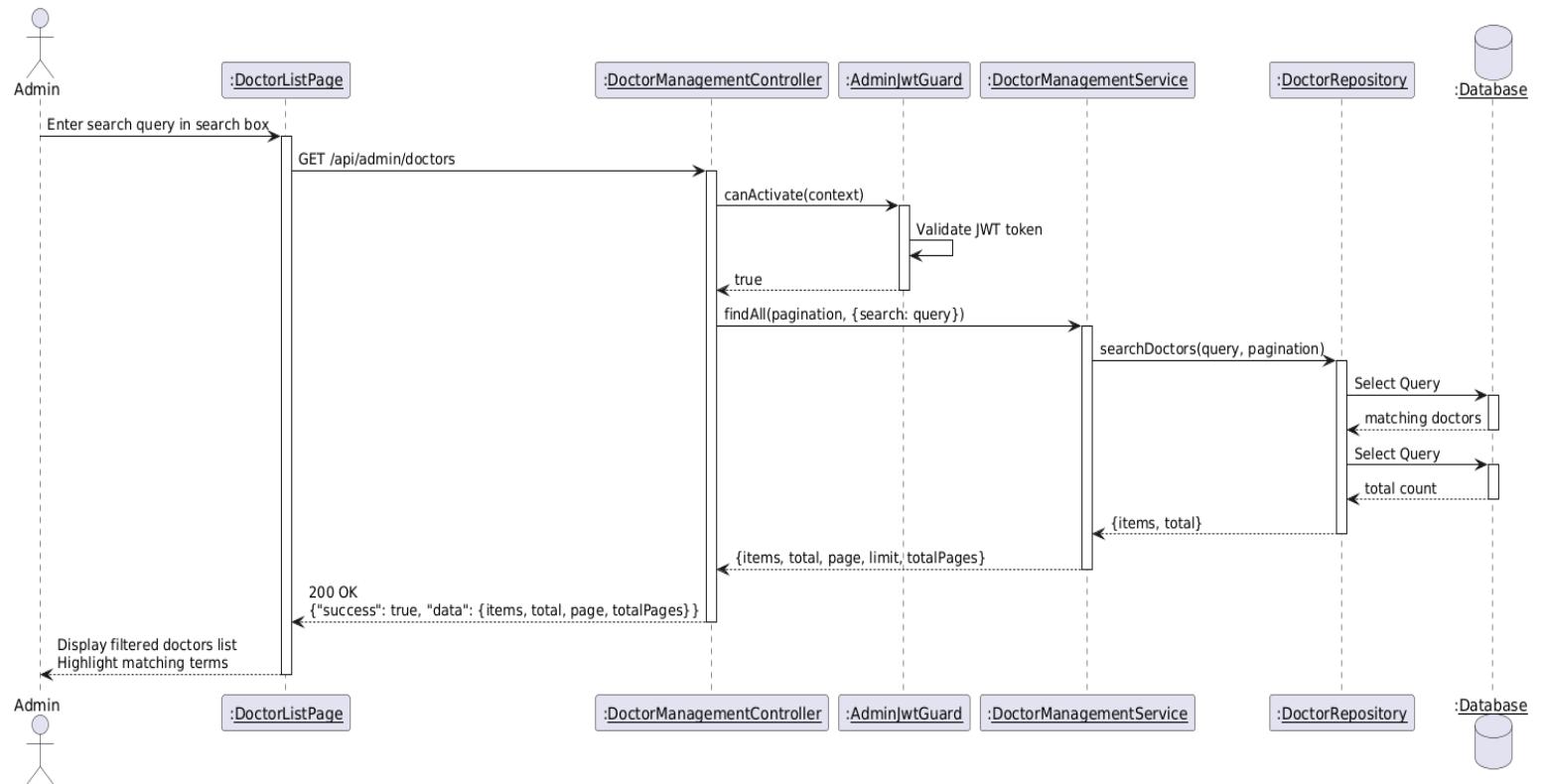


Diagram 71, Sequence Diagram (Search Clinicians Account List )

- **Create Organization :**

<b>Use case ID</b>	<b>VEMR-FR-AM-35</b>
<b>Use case name</b>	Create organization
<b>Description</b>	The system allows administrators to create a new healthcare organization.
<b>From</b>	Admin
<b>Pre-conditions</b>	Admin is authenticated
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. Administrator navigates to organizations management page</li> <li>2. Administrator clicks "Create Organization" button</li> <li>3. System displays organization creation form</li> <li>4. Administrator enters organization details (name, address, phone, email)</li> <li>5. Administrator submits form</li> <li>6. System validates input fields</li> <li>7. System validates JWT token via AdminJwtGuard</li> <li>8. System checks for duplicate organization name via OrganizationRepository</li> <li>9. System creates new organization record</li> <li>10. System displays success message and refreshes organization list</li> </ol>

<b>Alternative scenario</b>	<p><b>A1: Unauthorized Access</b></p> <ul style="list-style-type: none"> <li>- At step 7, if JWT token is invalid or expired</li> </ul> <p>System returns 401 Unauthorized</p> <p><b>A2: Validation Error</b></p> <ul style="list-style-type: none"> <li>- At step 6, if required fields are empty or invalid</li> </ul> <p>-System displays validation error messages</p> <p><b>A3: Duplicate Organization</b></p> <ul style="list-style-type: none"> <li>- At step 8, if organization name already exists</li> </ul> <p>-System displays error message</p>
<b>Post condition</b>	New organization is created in the system

Table 42, Use Case Specification (Create Organization )

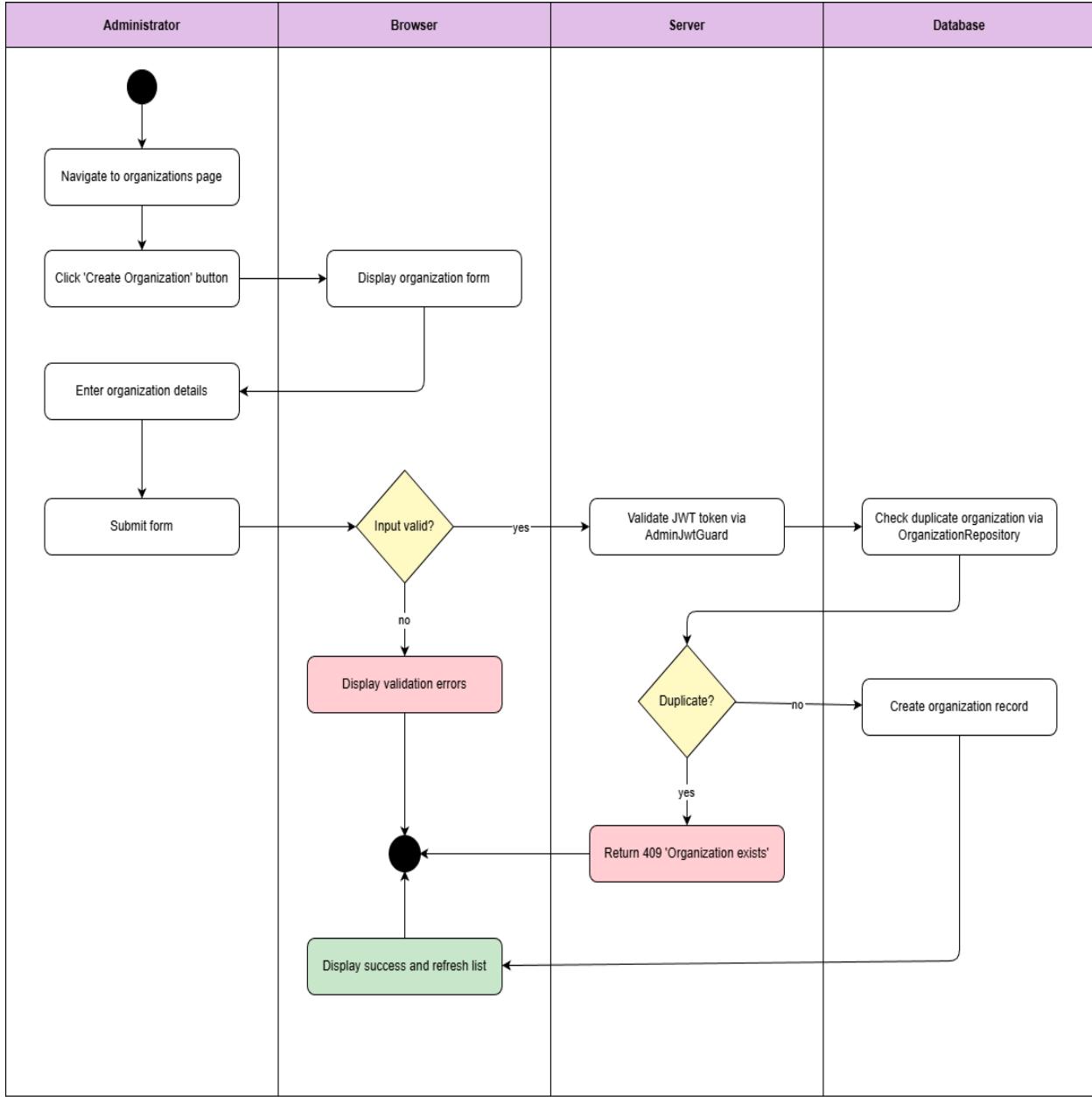


Diagram 72, Activity Diagram (Create Organization )

## Chapter 4 - System Analysis

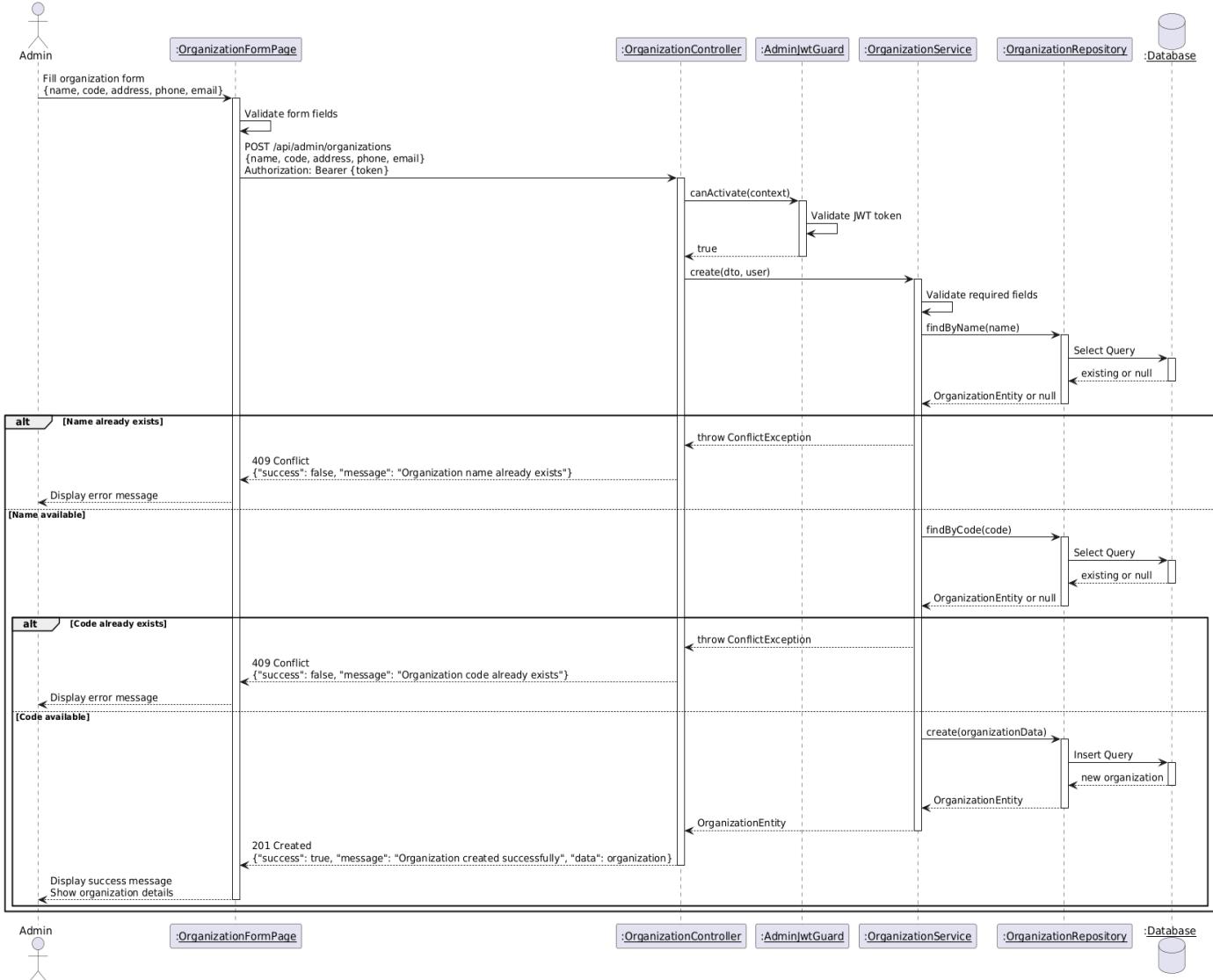


Diagram 73, Sequence Diagram (Create Organization )

## ● Update Organization :

<b>Use case ID</b>	VEMR-FR-AM-36
<b>Use case name</b>	Update organization
<b>Description</b>	The system allows administrators to update organization information.
<b>From</b>	Admin
<b>Pre-conditions</b>	Admin is authenticated
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. Administrator navigates to organizations list page</li> <li>2. Administrator clicks "Edit" button on organization row</li> <li>3. System displays organization edit form with current data</li> <li>4. Administrator modifies organization details</li> <li>5. Administrator submits form</li> <li>6. System validates input fields</li> <li>7. System validates JWT token via AdminJwtGuard</li> <li>8. System updates organization record via OrganizationRepository</li> <li>9. System displays success message and refreshes organization list</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Unauthorized Access</b></p> <ul style="list-style-type: none"> <li>- At step 7, if JWT token is invalid or expired</li> </ul> <p style="margin-left: 20px;">System returns 401 Unauthorized</p> <p><b>A2: Validation Error</b></p> <ul style="list-style-type: none"> <li>- At step 6, if required fields are empty or invalid</li> </ul> <p style="margin-left: 20px;">-System displays validation error messages</p> <p><b>A3: Organization Not Found</b></p> <ul style="list-style-type: none"> <li>- At step 8, if organization ID does not exist</li> </ul> <p style="margin-left: 20px;">-System displays error message</p>
<b>Post condition</b>	Organization information is updated in the system

Table 43, Use Case Specification (Update Organization )

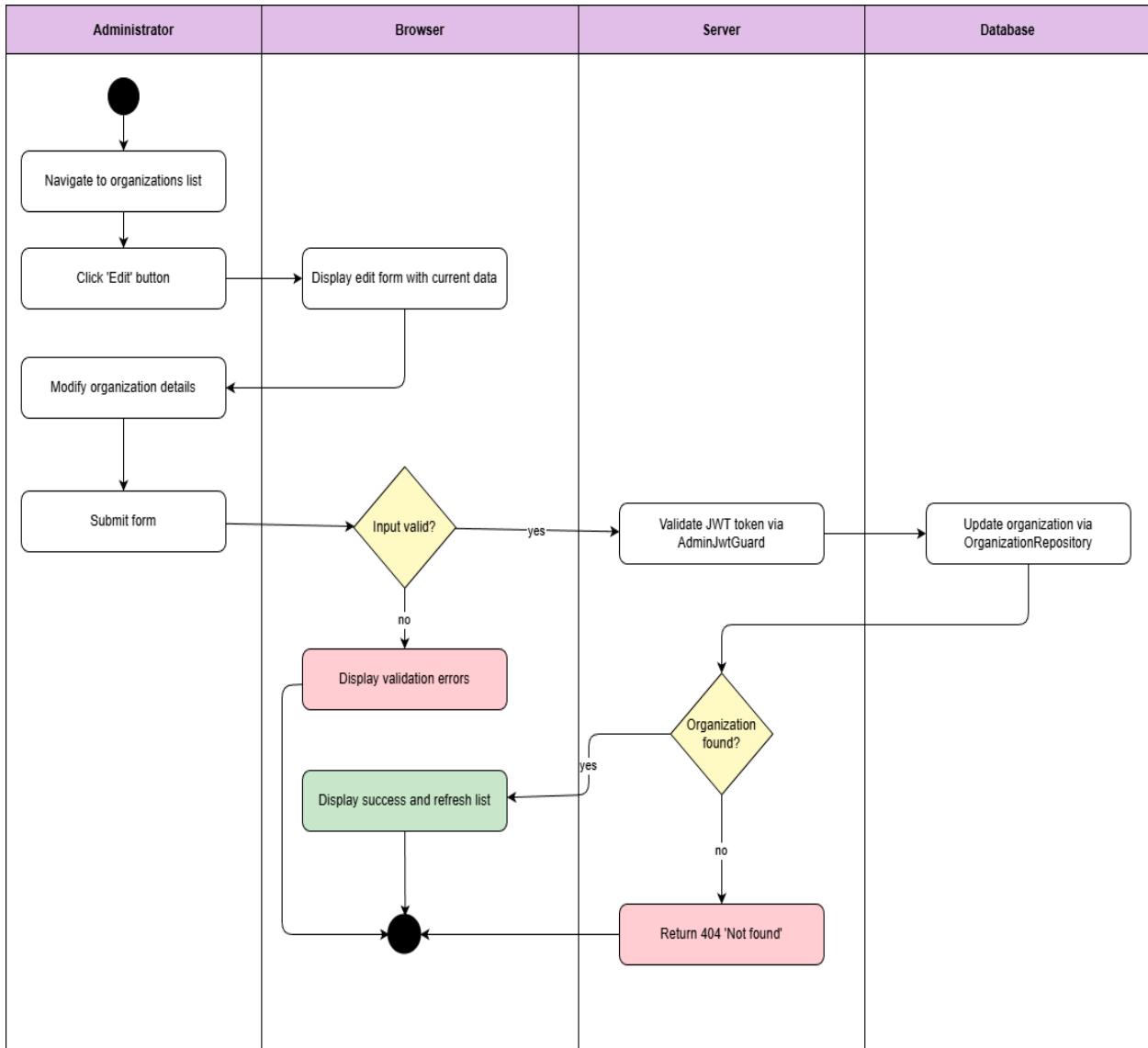


Diagram 74, Activity Diagram (Update Organization )

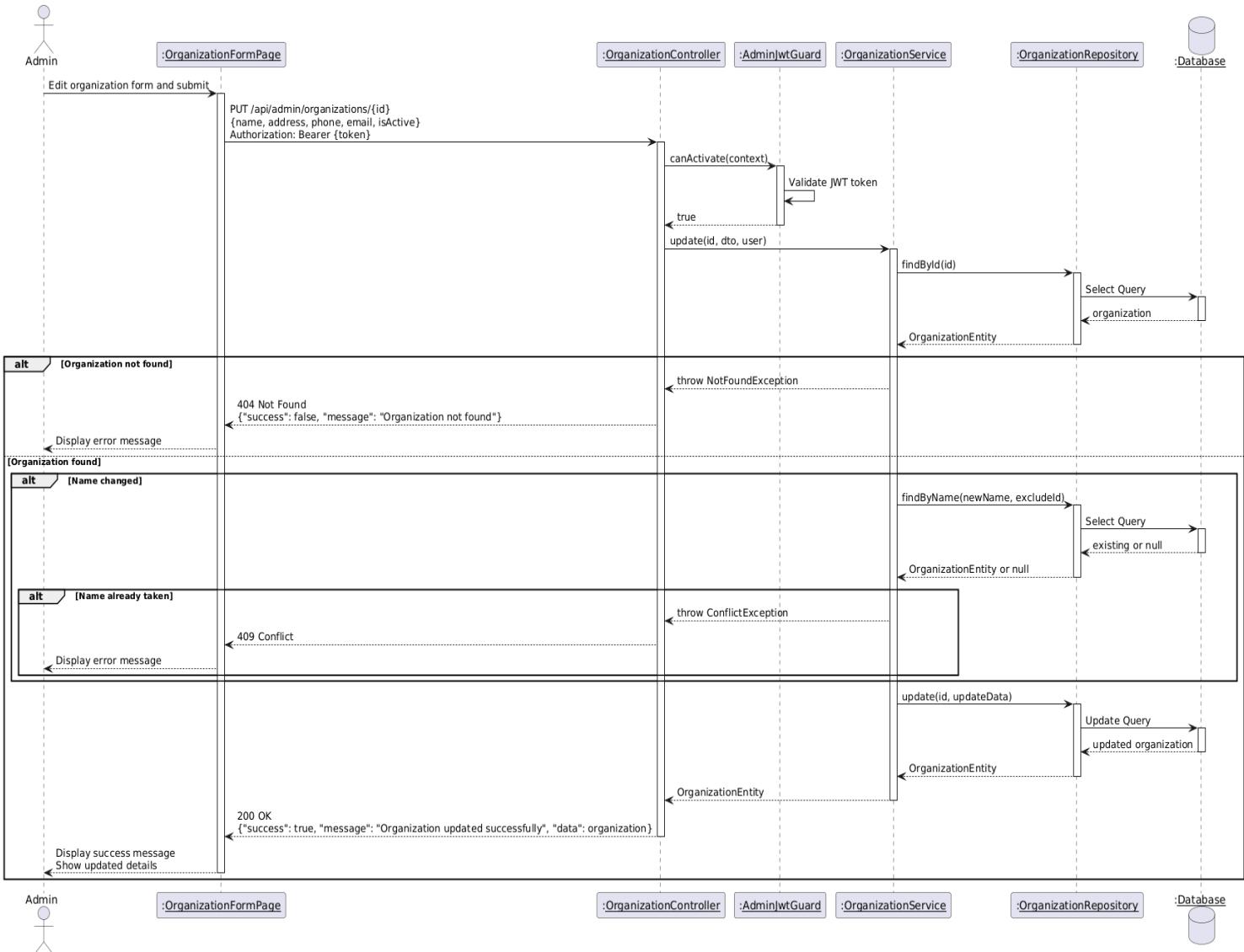


Diagram 75, Sequence Diagram (Update Organization )

## ● View Organization :

<b>Use case ID</b>	<b>VEMR-FR-AM-37</b>
<b>Use case name</b>	View organization
<b>Description</b>	The system allows administrators to view all healthcare organizations with pagination
<b>From</b>	Admin
<b>Pre-conditions</b>	Admin is authenticated
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. Administrator navigates to organizations management page</li> <li>2. System validates JWT token via AdminJwtGuard</li> <li>3. System retrieves organizations list via OrganizationRepository</li> <li>4. System returns paginated organizations with total count</li> <li>5. System displays organizations in table with pagination controls</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Unauthorized Access</b></p> <ul style="list-style-type: none"> <li>- At step 2, if JWT token is invalid or expired System returns 401 Unauthorized</li> </ul> <p><b>A2: No Organizations Found</b></p> <ul style="list-style-type: none"> <li>- At step 3, if no organizations exist in the system -System displays "No organizations found" message</li> </ul>
<b>Post condition</b>	Organizations list is displayed with pagination

Table 44, Use Case Specification (View Organization )

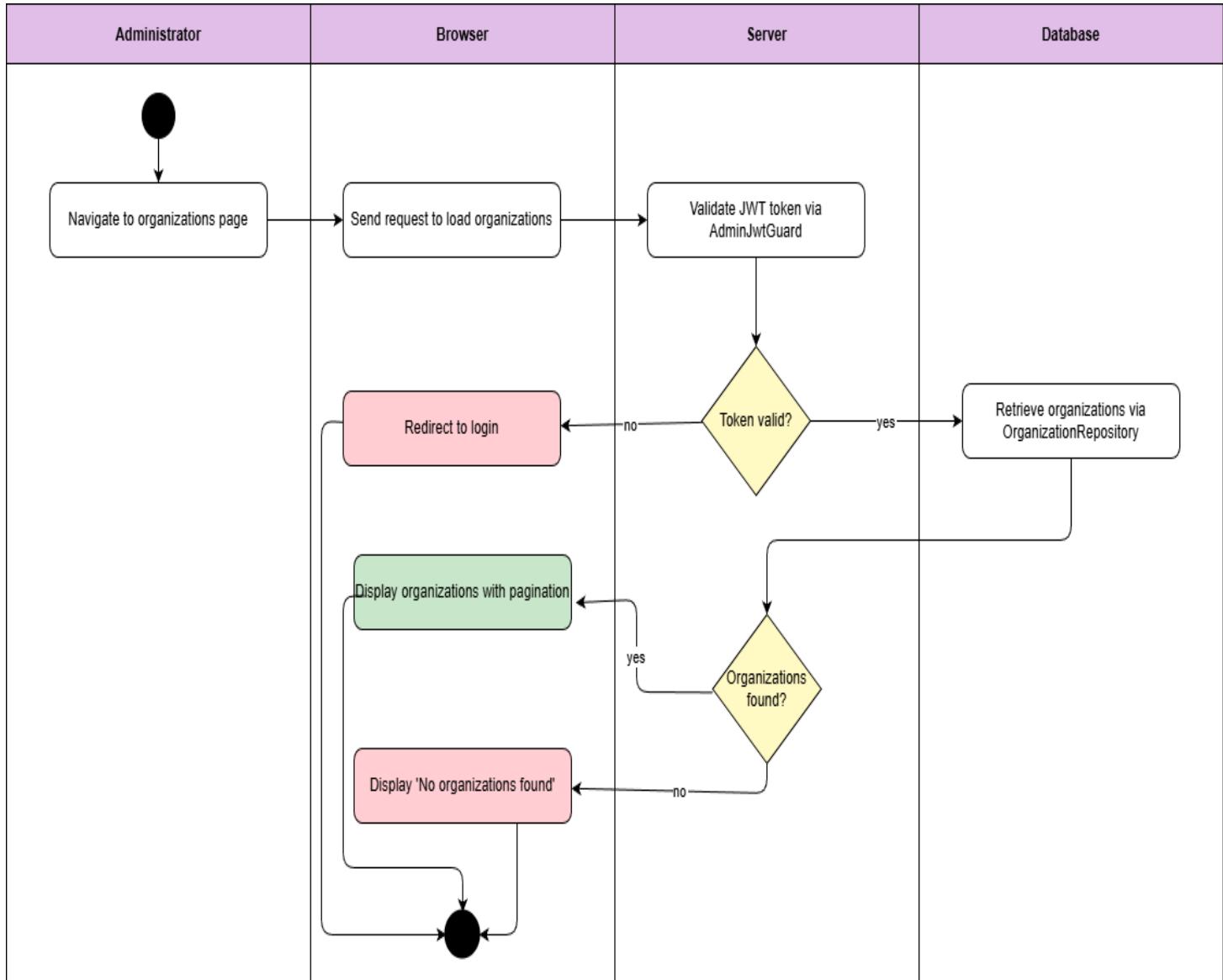


Diagram 76, Activity Diagram (View Organization )

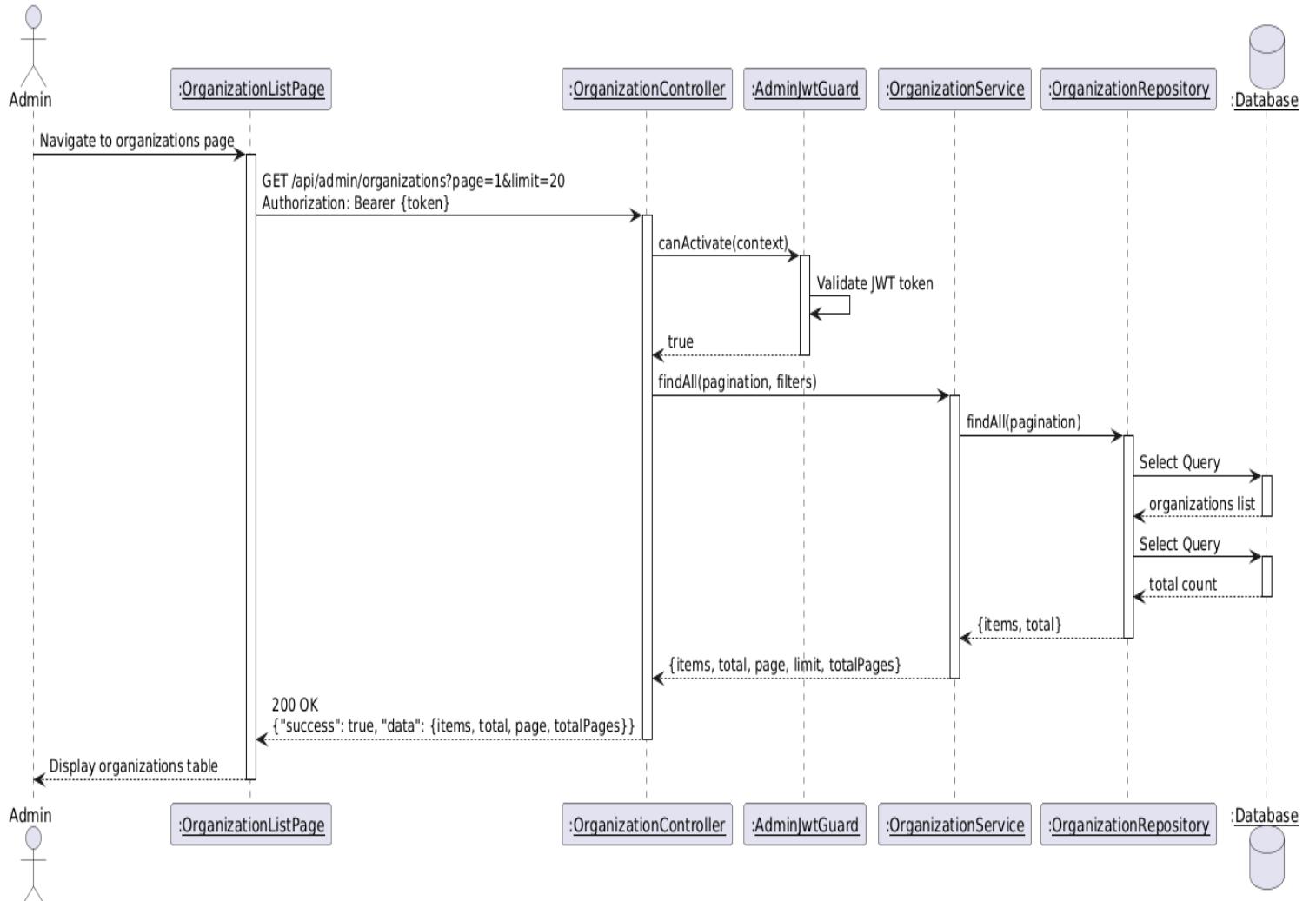


Diagram 77, Sequence Diagram (View Organization )

- **View Organization Details :**

<b>Use case ID</b>	<b>VEMR-FR-AM-38</b>
<b>Use case name</b>	View organization Details
<b>Description</b>	The system allows administrators to view detailed information about a specific organization.
<b>From</b>	Admin
<b>Pre-conditions</b>	Admin is authenticated
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>Administrator navigates to organizations list page</li> <li>Administrator clicks on organization name to view details</li> <li>System validates JWT token via AdminJwtGuard</li> <li>System retrieves organization data via OrganizationRepository</li> <li>System retrieves associated clinicians via DoctorRepository</li> <li>System displays organization details with clinicians list</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Unauthorized Access</b></p> <ul style="list-style-type: none"> <li>- At step 3, if JWT token is invalid or expired System returns 401 Unauthorized</li> </ul> <p><b>A2: No Organizations Found</b></p> <ul style="list-style-type: none"> <li>- At step 4, if organization ID does not exist -System displays error message</li> </ul> <p><b>A3: No Clinicians</b></p> <ul style="list-style-type: none"> <li>- System displays empty state message -System displays "No organizations found" message</li> </ul>

<b>Post condition</b>	Organization details are displayed with complete information
-----------------------	--

Table 45, Use Case Specification (View Organization Details )

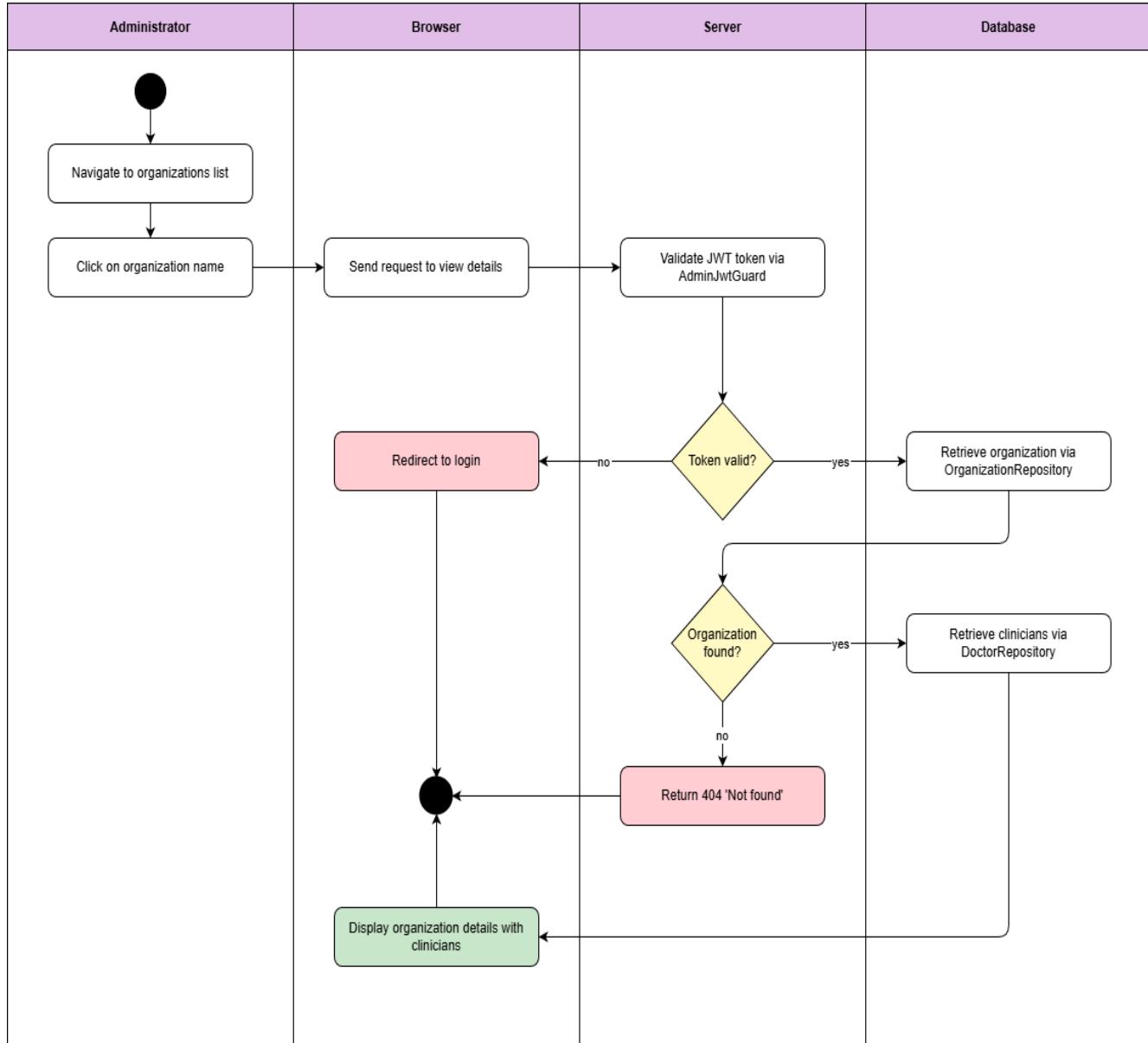


Diagram 78, Activity Diagram (View Organization Details )

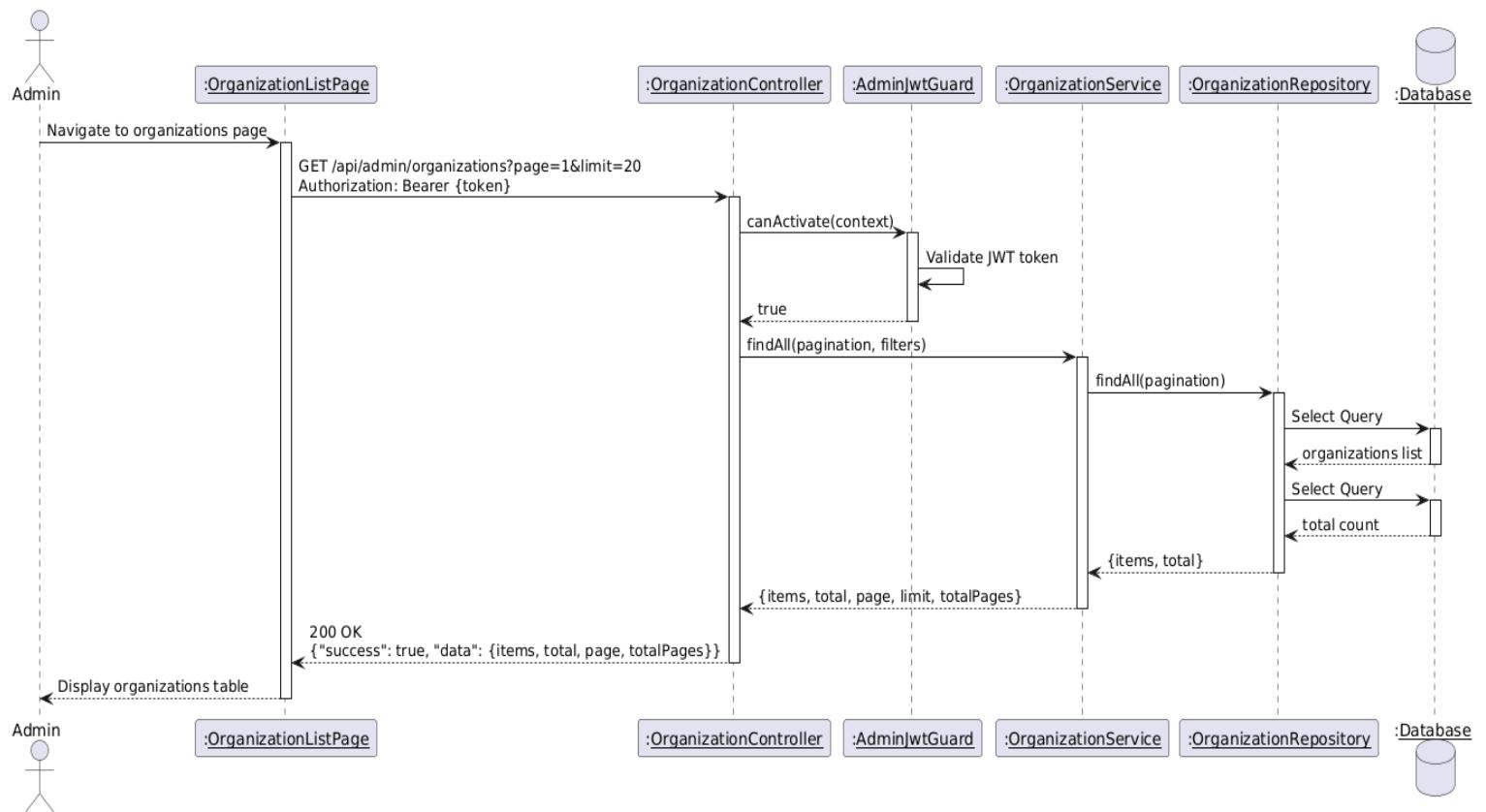


Diagram 79, Sequence Diagram (View Organization Details )

- **Delete Organization :**

<b>Use case ID</b>	<b>VEMR-FR-AM-39</b>
<b>Use case name</b>	Delete Organization
<b>Description</b>	The system allows administrators to delete a healthcare organization from the system
<b>From</b>	Admin
<b>Pre-conditions</b>	Admin is authenticated
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. Administrator navigates to organizations list page</li> <li>2. Administrator clicks "Delete" button on organization row</li> <li>3. System displays confirmation dialog</li> <li>4. Administrator confirms deletion</li> <li>5. System validates JWT token via AdminJwtGuard</li> <li>6. System checks for associated clinicians via DoctorRepository</li> <li>7. System deletes organization record via OrganizationRepository</li> <li>8. System displays success message and refreshes organization list</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Unauthorized Access</b></p> <ul style="list-style-type: none"> <li>- At step 5, if JWT token is invalid or expired System returns 401 Unauthorized</li> </ul> <p><b>A2: Organization Not Found</b></p> <ul style="list-style-type: none"> <li>- At step 7, if organization ID does not exist -System displays error message</li> </ul> <p><b>A3: Organization Has Clinicians</b></p>

	<ul style="list-style-type: none"><li>- At step 6, if organization has associated clinicians</li><li>-System displays error message</li></ul>
<b>Post condition</b>	Organization is removed from the system

Table 46, Use Case Specification (Delete Organization )

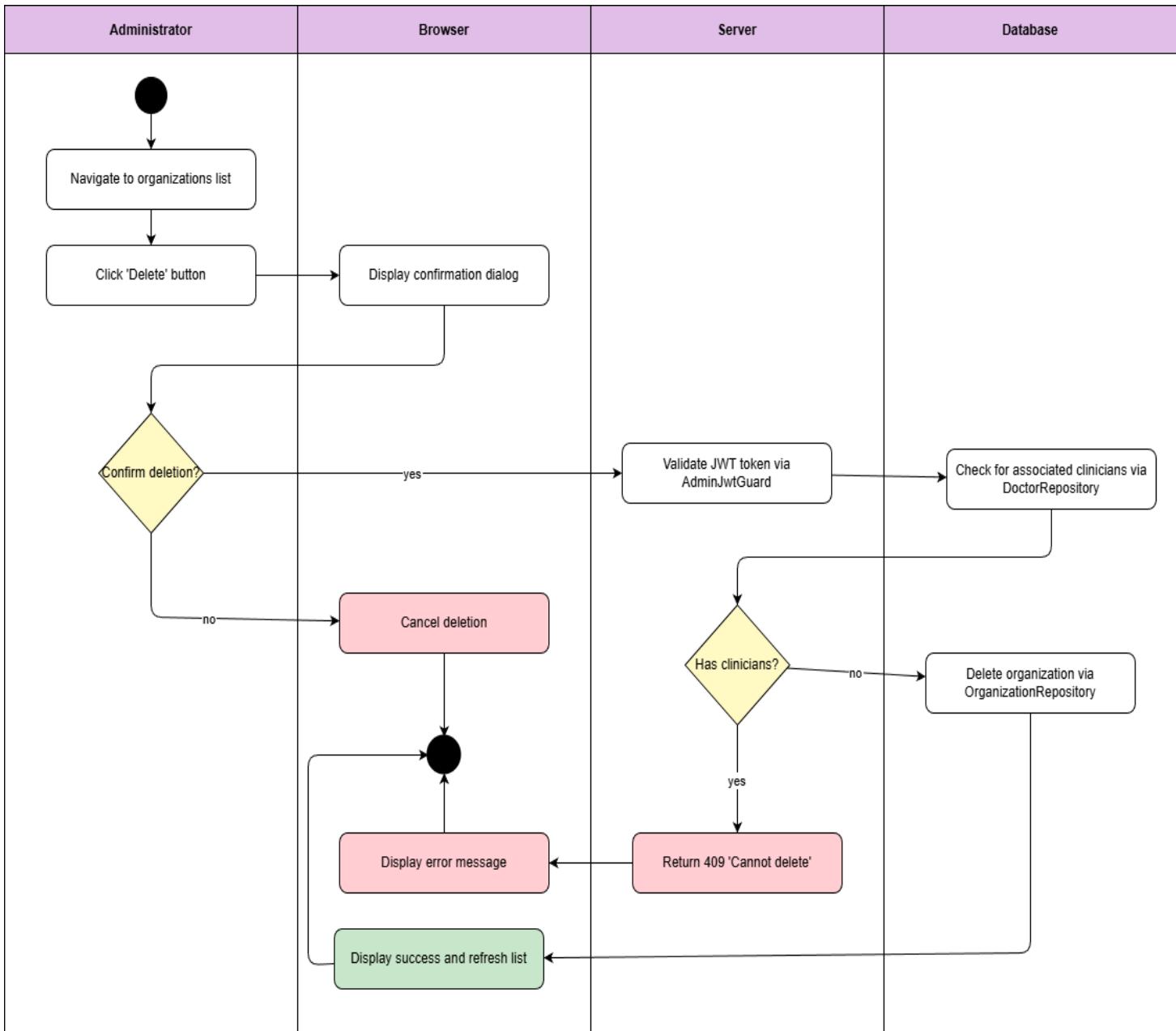


Diagram 80, Activity Diagram (Delete Organization )

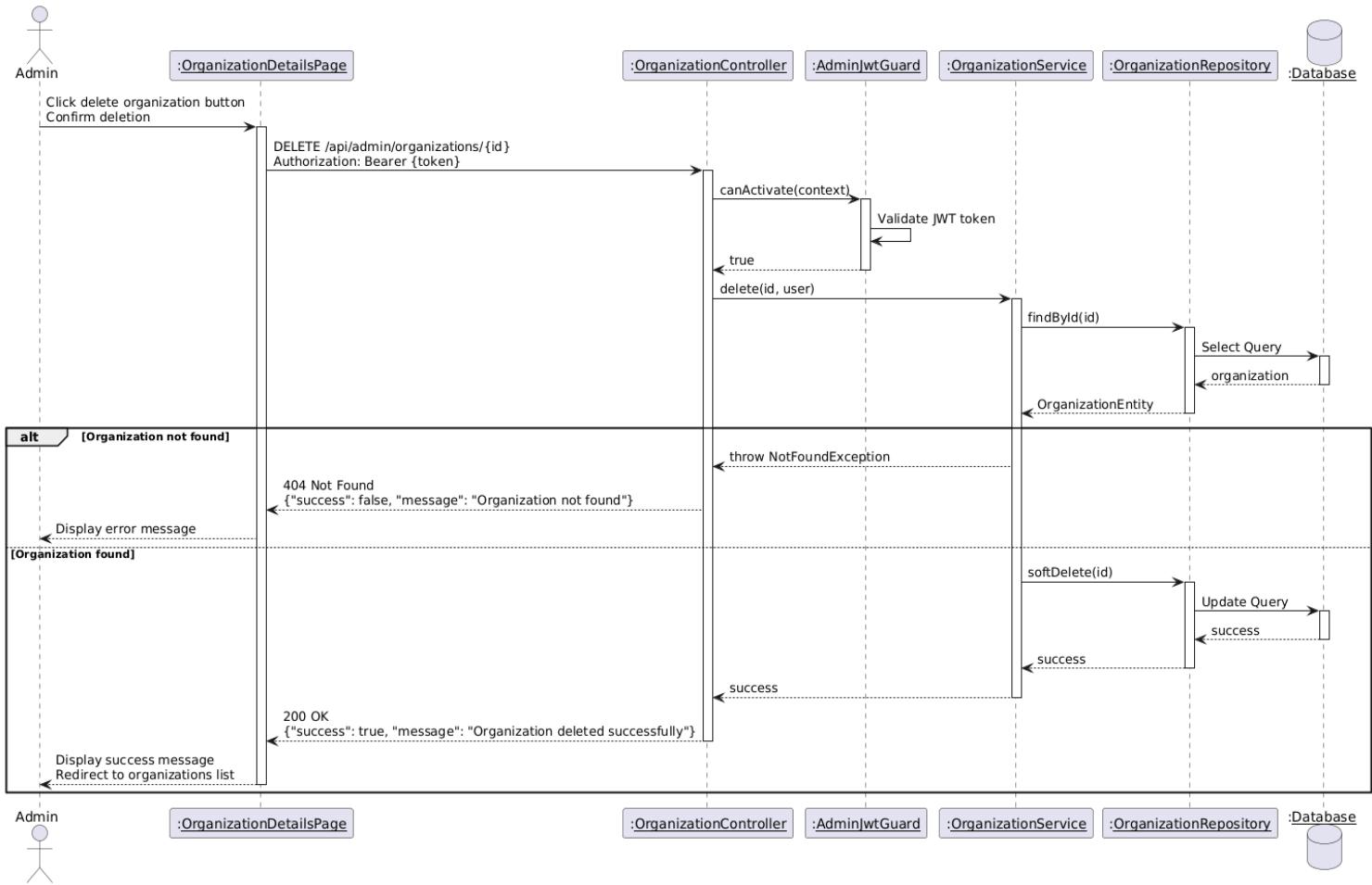


Diagram 81, Sequence Diagram (Delete Organization )

## ● Search Organization :

<b>Use case ID</b>	<b>VEMR-FR-AM-40</b>
<b>Use case name</b>	Search Organization
<b>Description</b>	The system allows administrators to search for organizations by name or location.
<b>From</b>	Admin
<b>Pre-conditions</b>	Admin is authenticated
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. Administrator navigates to organizations management page</li> <li>2. Administrator enters search query in search box</li> <li>3. System debounces input (300ms delay)</li> <li>4. System validates JWT token via AdminJwtGuard</li> <li>5. System searches organizations by name or address via OrganizationRepository</li> <li>6. System returns paginated search results</li> <li>7. System displays matching organizations in table</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Unauthorized Access</b></p> <ul style="list-style-type: none"> <li>- At step 4, if JWT token is invalid or expired</li> </ul> <p>System returns 401 Unauthorized</p> <p><b>A2: No Results Found</b></p> <ul style="list-style-type: none"> <li>- At step 5, if no organizations match the search query</li> </ul> <p>-System returns empty list</p> <p><b>A3: Empty Search Query</b></p> <ul style="list-style-type: none"> <li>- At step 3, if search query is empty or cleared</li> </ul> <p>- System returns all organizations with pagination</p>
<b>Post condition</b>	Administrator can view search results with pagination

Table 47, Use Case Specification (Search Organization )

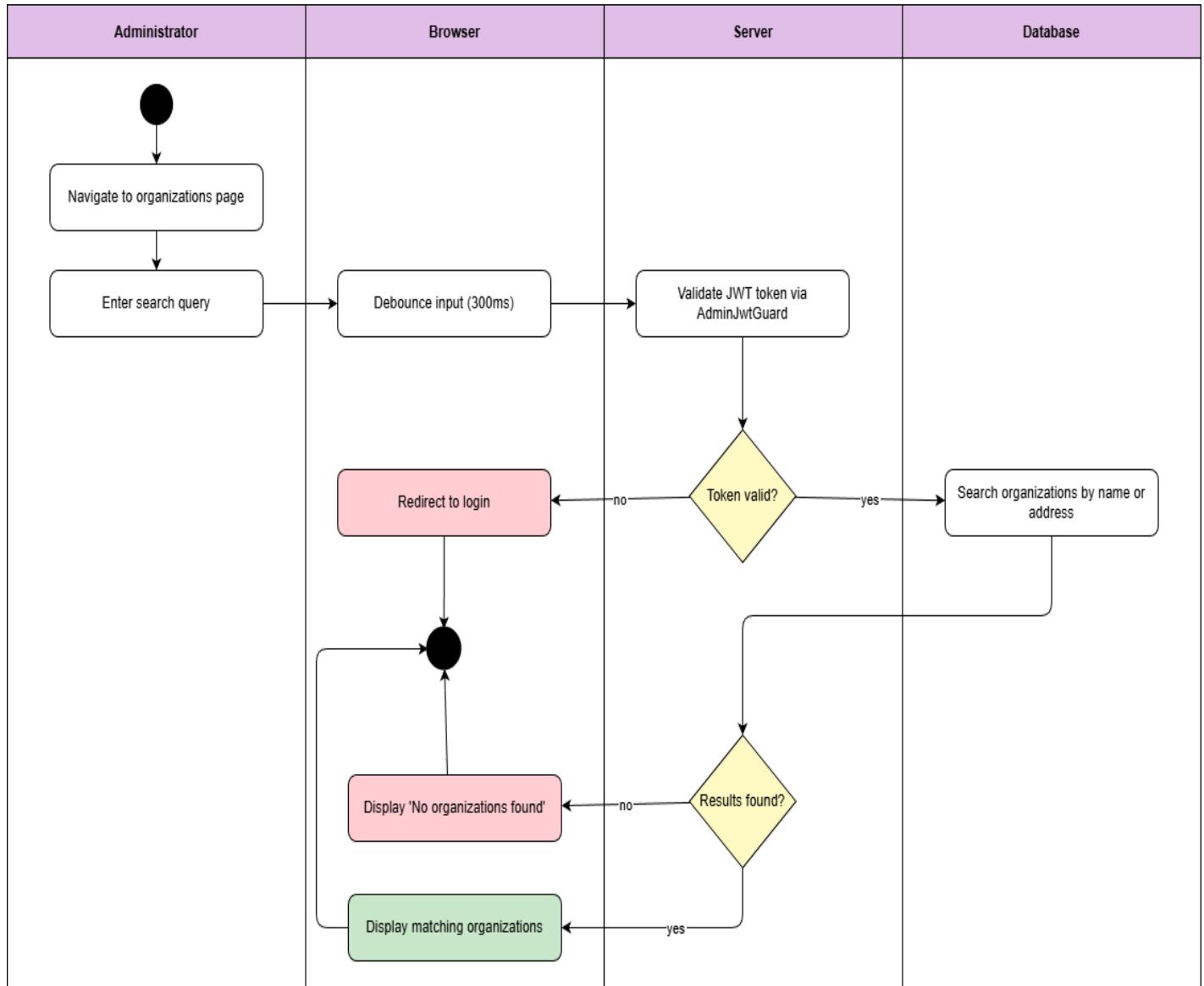


Diagram 82, Activity Diagram (Search Organization )

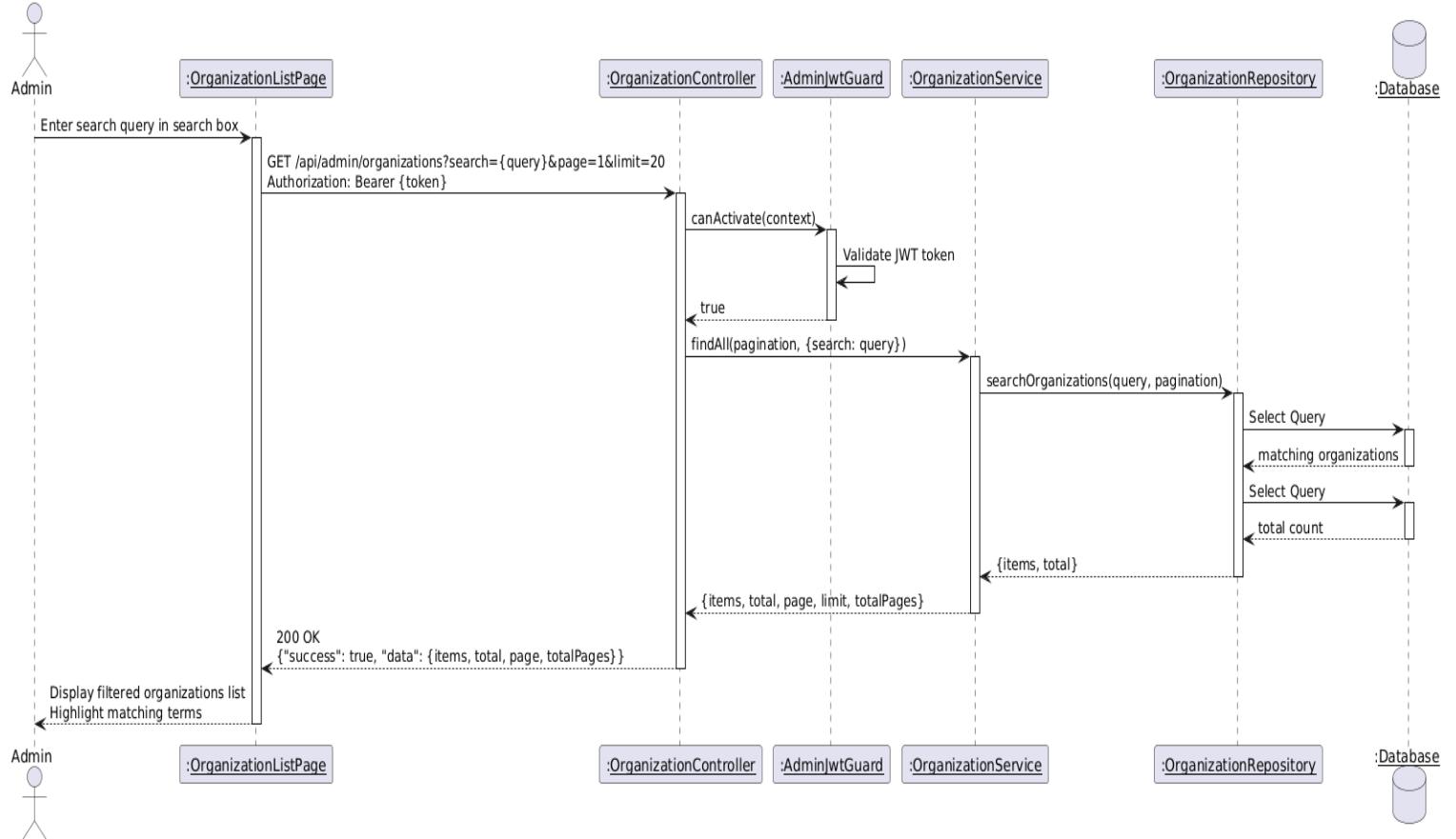


Diagram 83, Sequence Diagram (Search Organization )

- **View Admin Analytics :**

<b>Use case ID</b>	<b>VEMR-FR-AM-41</b>
<b>Use case name</b>	View admin analytics
<b>Description</b>	The system allows administrators to view system-wide analytics and statistics.
<b>From</b>	Admin
<b>Pre-conditions</b>	Admin is authenticated
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. Administrator navigates to admin dashboard</li> <li>2. System validates JWT token via AdminJwtGuard</li> <li>3. System retrieves total patients count via PatientRepository</li> <li>4. System retrieves total clinicians count via DoctorRepository</li> <li>5. System retrieves total organizations count via OrganizationRepository</li> <li>6. System retrieves total visits count via EncounterRepository</li> <li>7. System calculates visit statistics by status</li> <li>8. System displays analytics dashboard with charts and statistics</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Unauthorized Access</b></p> <ul style="list-style-type: none"> <li>- At step 2, if JWT token is invalid or expired System returns 401 Unauthorized</li> </ul> <p><b>A2: No Data Available</b></p> <ul style="list-style-type: none"> <li>- At steps 3-6, if no data exists in the system - System shows empty state message</li> </ul>
<b>Post condition</b>	Analytics dashboard is displayed with system statistics

Table 48, Use Case Specification (View Admin Analytics )

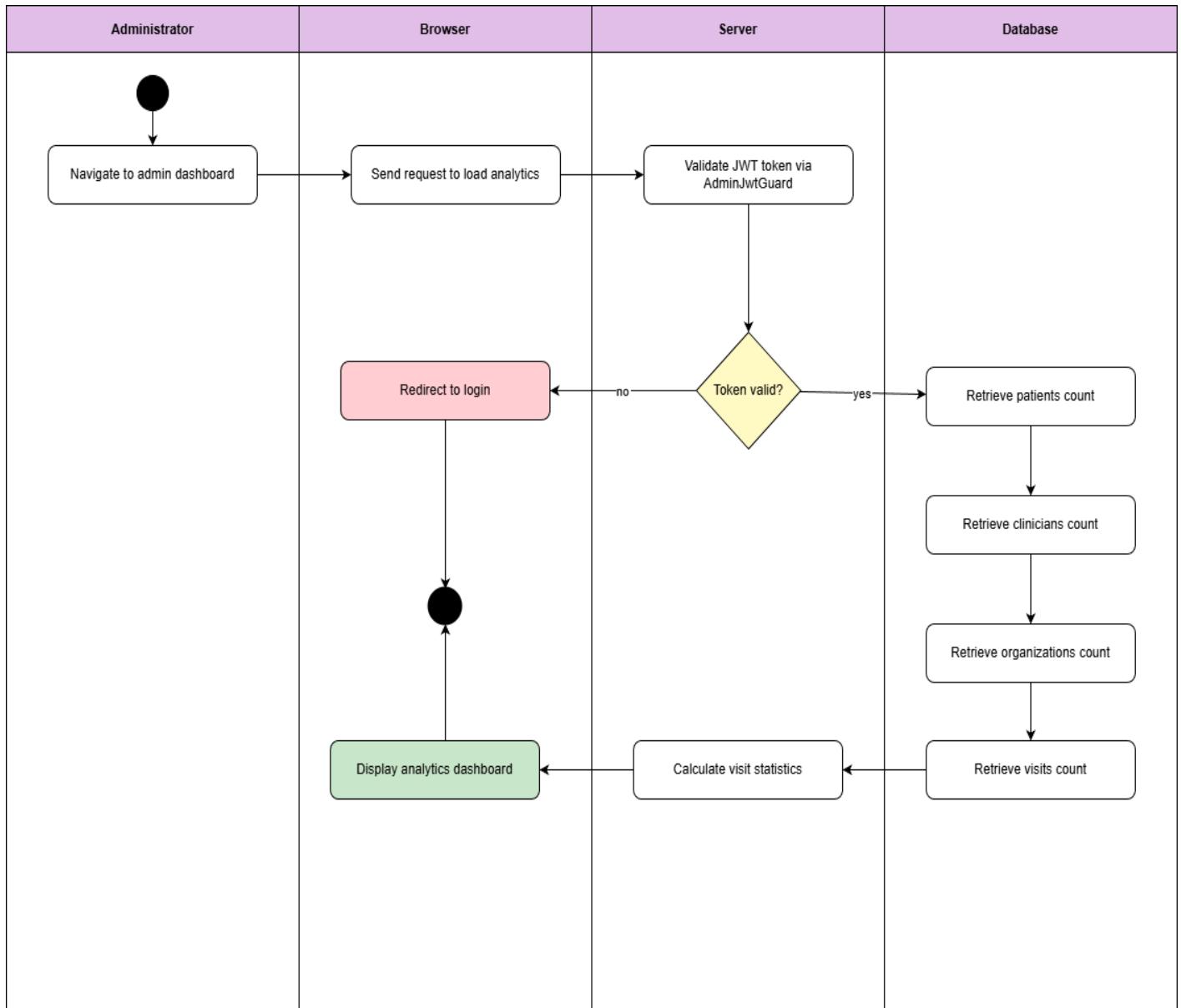


Diagram 84, Activity Diagram ( View Admin Analytics )

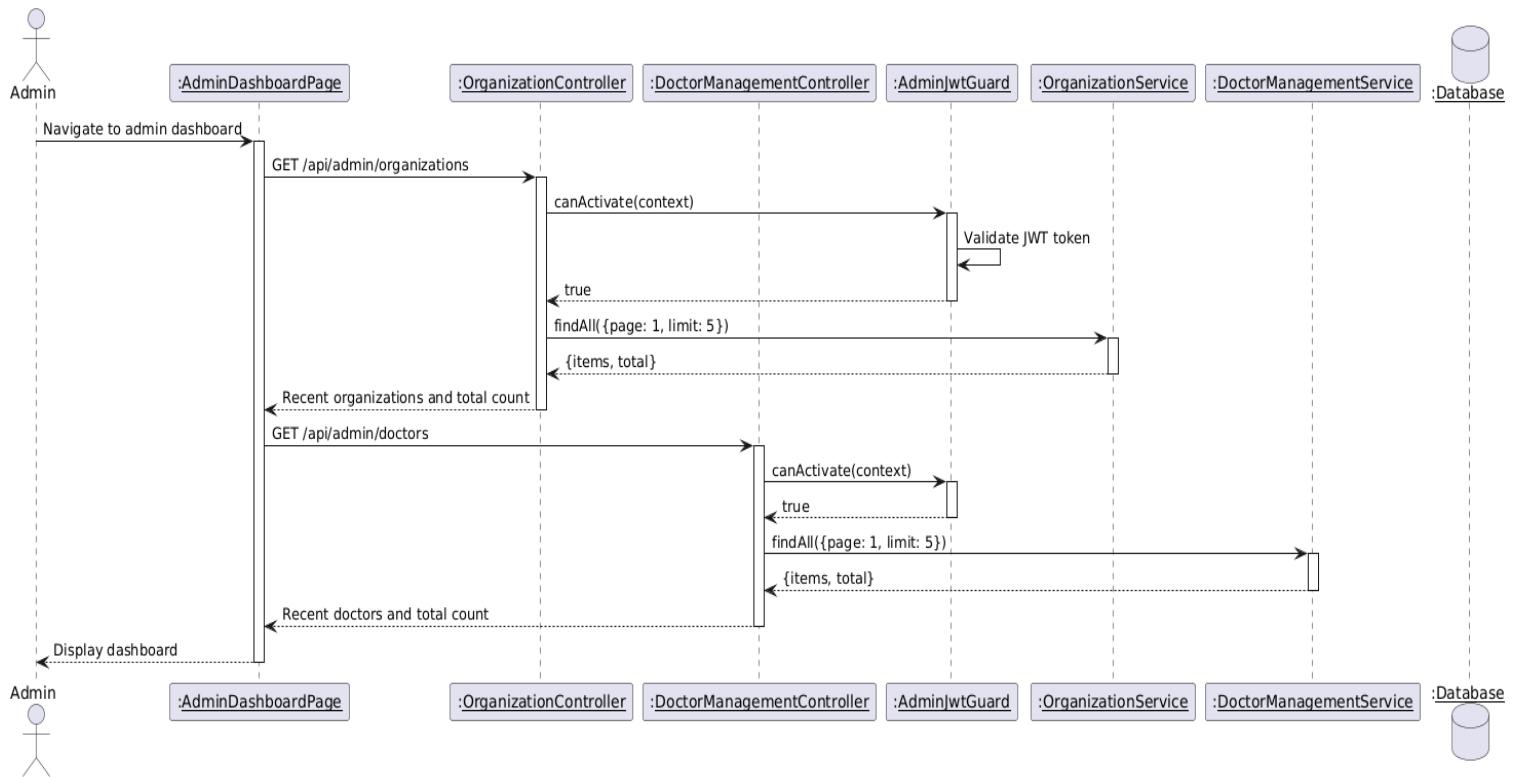


Diagram 85, Sequence Diagram ( View Admin Analytics )

- **View All Appointment:**

<b>Use case ID</b>	<b>VEMR-FR-AM-42</b>
<b>Use case name</b>	View All Appointment
<b>Description</b>	The system allows clinicians to view their complete appointment schedule
<b>From</b>	Clinician
<b>Pre-conditions</b>	Clinician is authenticated
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. Clinician navigates to appointments page</li> <li>2. System validates JWT token via DoctorJwtGuard</li> <li>3. System retrieves all appointments for clinician via AppointmentRepository</li> <li>4. System returns paginated appointments with patient details</li> <li>5. System displays appointments list with date, time, patient name, and status</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Unauthorized Access</b></p> <ul style="list-style-type: none"> <li>- At step 2, if JWT token is invalid or expired</li> <li>- System redirects to login page</li> </ul> <p><b>A2: No Appointments Found</b></p> <ul style="list-style-type: none"> <li>- At step 3, if clinician has no appointments</li> <li>- System returns empty list</li> </ul>
<b>Post condition</b>	Appointments list is displayed with pagination

Table 49, Use Case Specification ( View All Appointment )

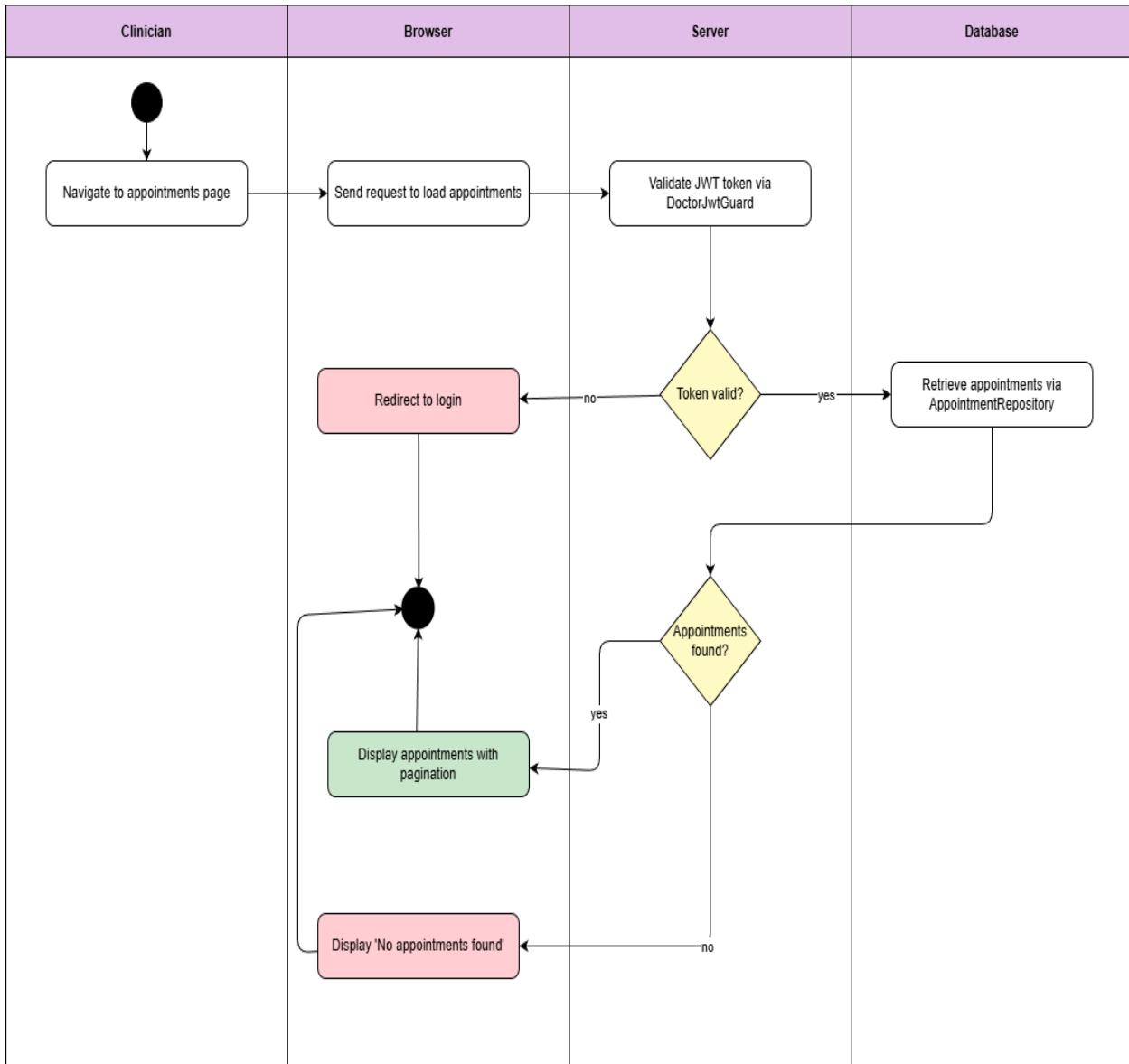


Diagram 86, Activity Diagram (View All Appointment )

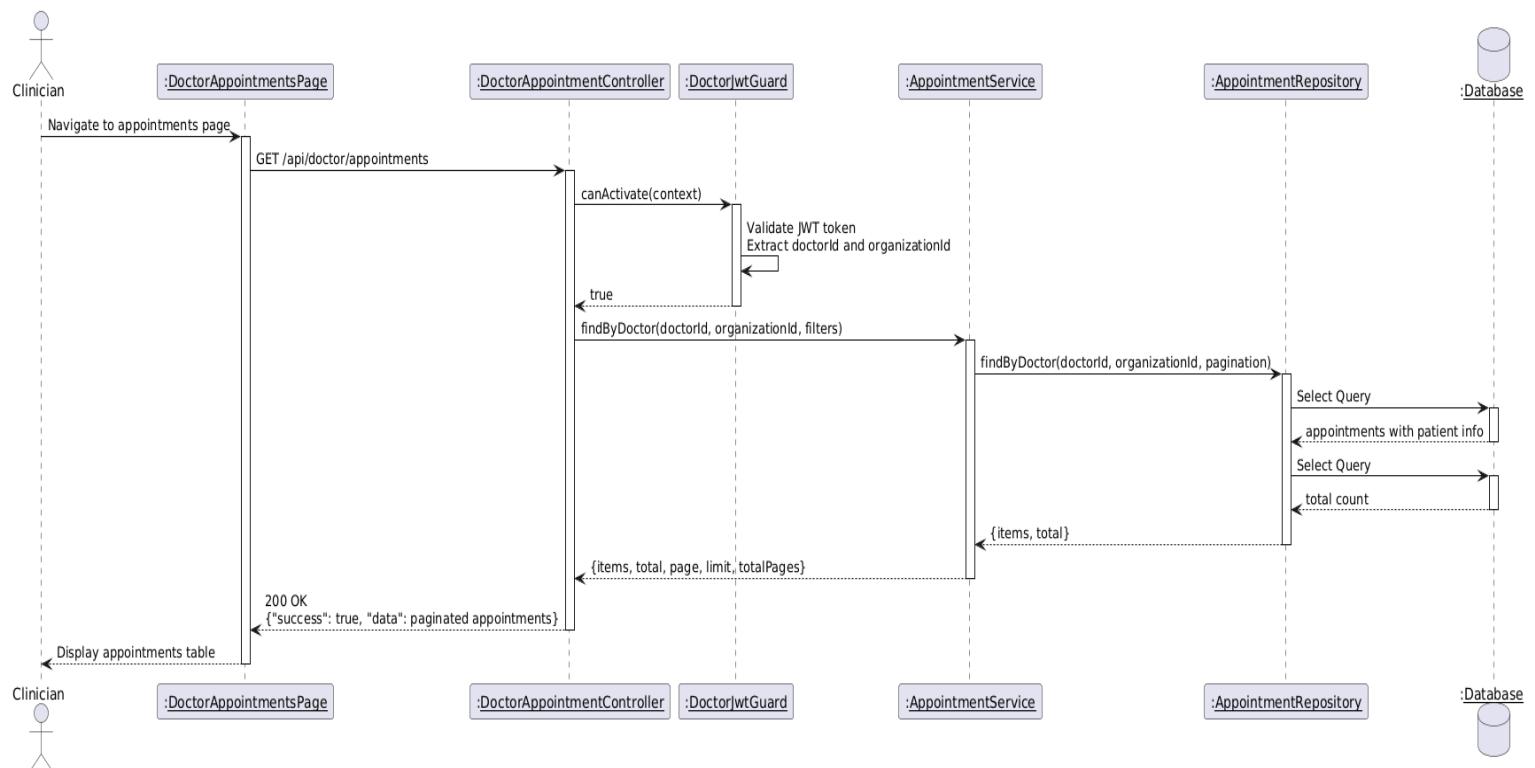


Diagram 87, Sequence Diagram (View All Appointment )

- **View Daily Appointments:**

<b>Use case ID</b>	<b>VEMR-FR-AM-43</b>
<b>Use case name</b>	View Daily Appointments
<b>Description</b>	The system allows clinicians to view their daily appointment schedule.
<b>From</b>	Clinician
<b>Pre-conditions</b>	Clinician is authenticated
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. Clinician navigates to daily appointments view</li> <li>2. System validates JWT token via DoctorJwtGuard</li> <li>3. System retrieves today's appointments for clinician via AppointmentRepository</li> <li>4. System filters appointments by current date</li> <li>5. System displays daily appointments with time slots and patient details</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Unauthorized Access</b></p> <ul style="list-style-type: none"> <li>- At step 2, if JWT token is invalid or expired</li> <li>- System redirects to login page</li> </ul> <p><b>A2: No Appointments Today</b></p> <ul style="list-style-type: none"> <li>- At step 3, if clinician has no appointments for today</li> <li>- System returns empty list</li> </ul>
<b>Post condition</b>	Daily appointments are displayed in chronological order

Table 50, Use Case Specification ( View Daily Appointments )

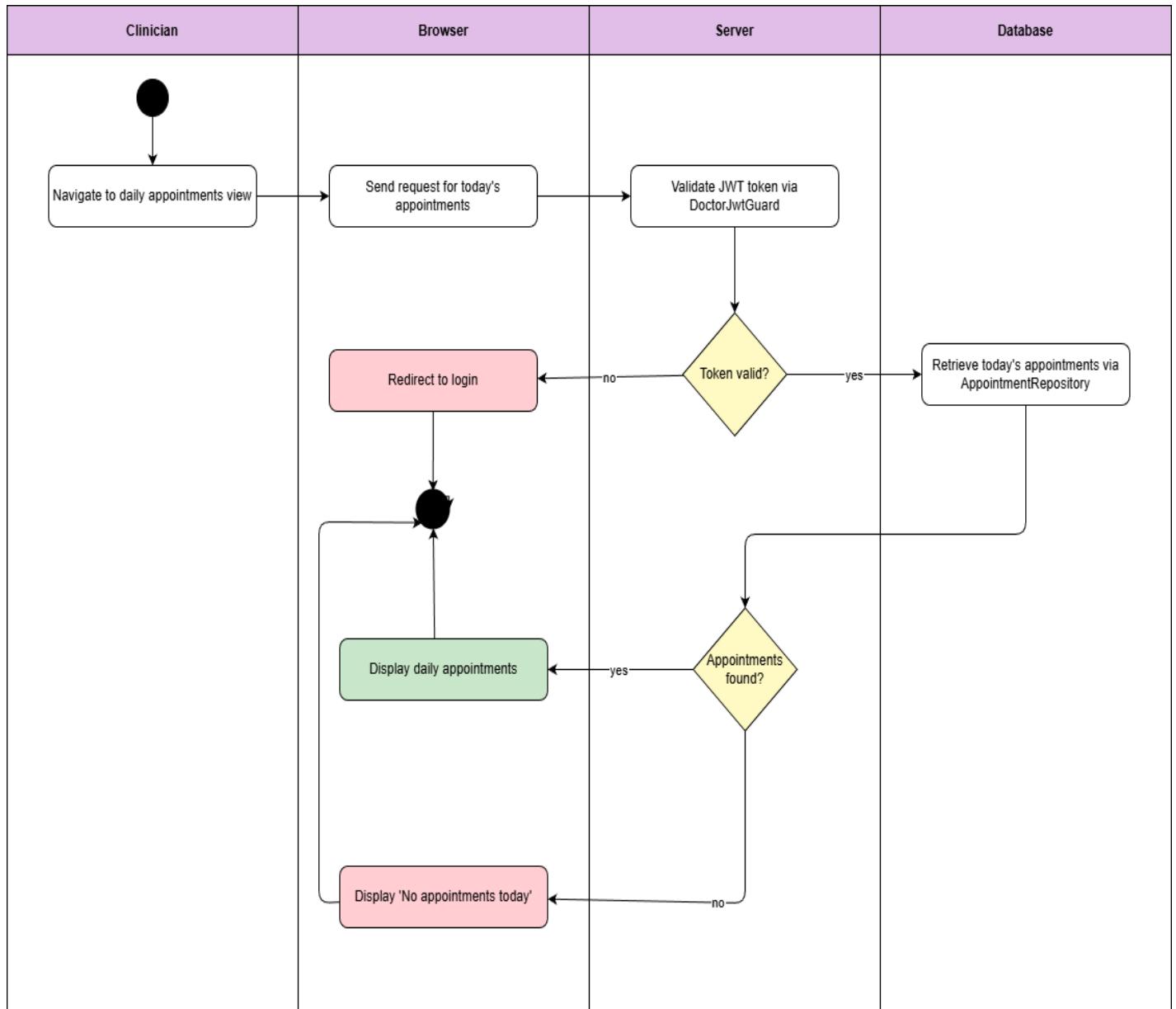


Diagram 88, Activity Diagram (View Daily Appointment )

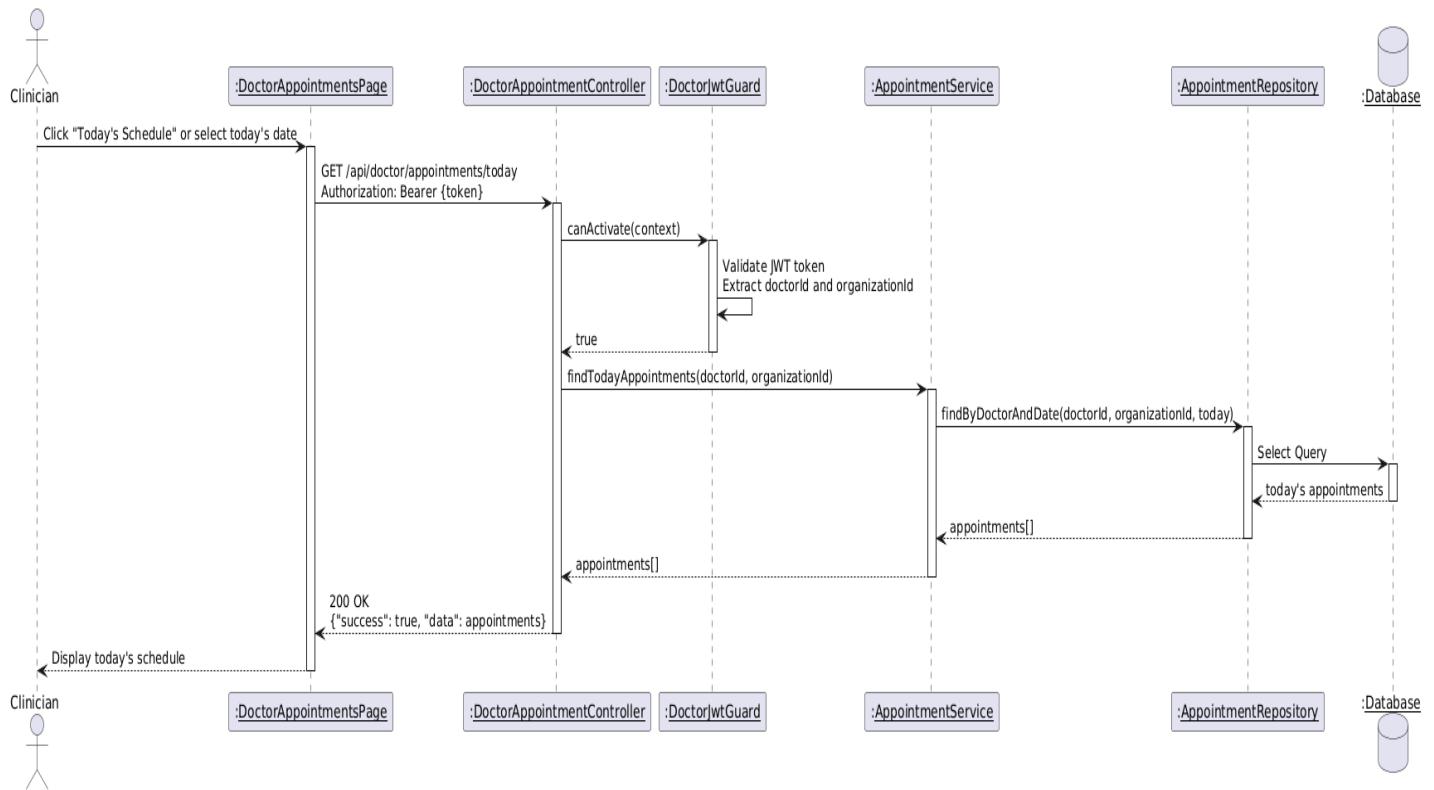


Diagram 89, Sequence Diagram (View Daily Appointment )

- **Filter Appointments Per Date :**

<b>Use case ID</b>	<b>VEMR-FR-AM-44</b>
<b>Use case name</b>	Filter Appointments Per Date
<b>Description</b>	The system allows clinicians to filter appointments by specific date
<b>From</b>	Clinician
<b>Pre-conditions</b>	Clinician is authenticated
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. Clinician navigates to appointments page</li> <li>2. Clinician selects date from date picker</li> <li>3. System validates JWT token via DoctorJwtGuard</li> <li>4. System retrieves appointments for selected date via AppointmentRepository</li> <li>5. System displays filtered appointments for the selected date</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Unauthorized Access</b></p> <ul style="list-style-type: none"> <li>- At step 3, if JWT token is invalid or expired</li> <li>- System redirects to login page</li> </ul> <p><b>A2: No Appointments on Selected Date</b></p> <ul style="list-style-type: none"> <li>- At step 4, if no appointments exist for selected date</li> <li>- System displays "No appointments found for this date" message</li> </ul>
<b>Post condition</b>	Clinician can view appointments for specific date

Table 51, Use Case Specification ( Filter Appointments Per Date)

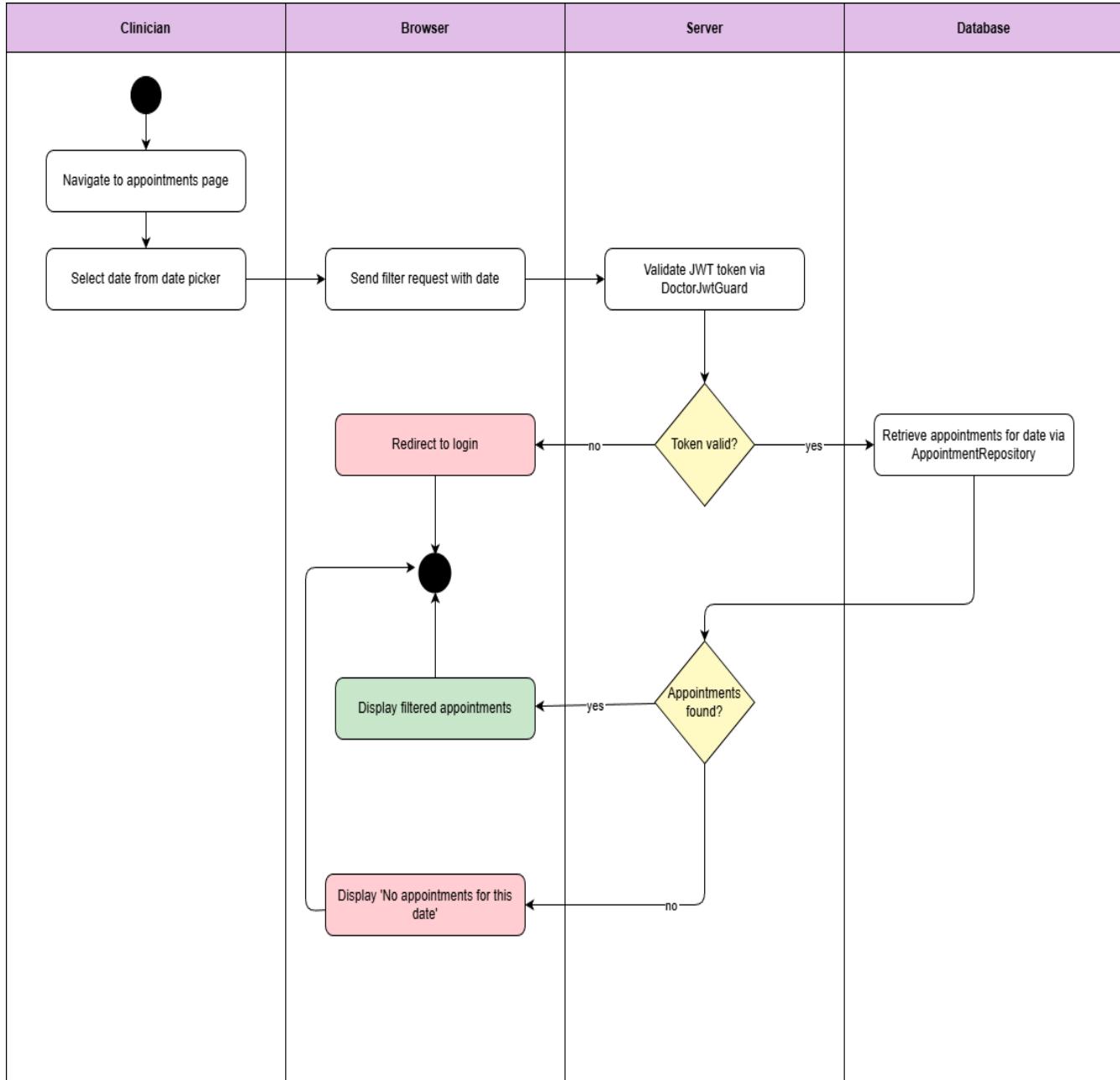


Diagram 90, Activity Diagram (Filter Appointments Per Date)

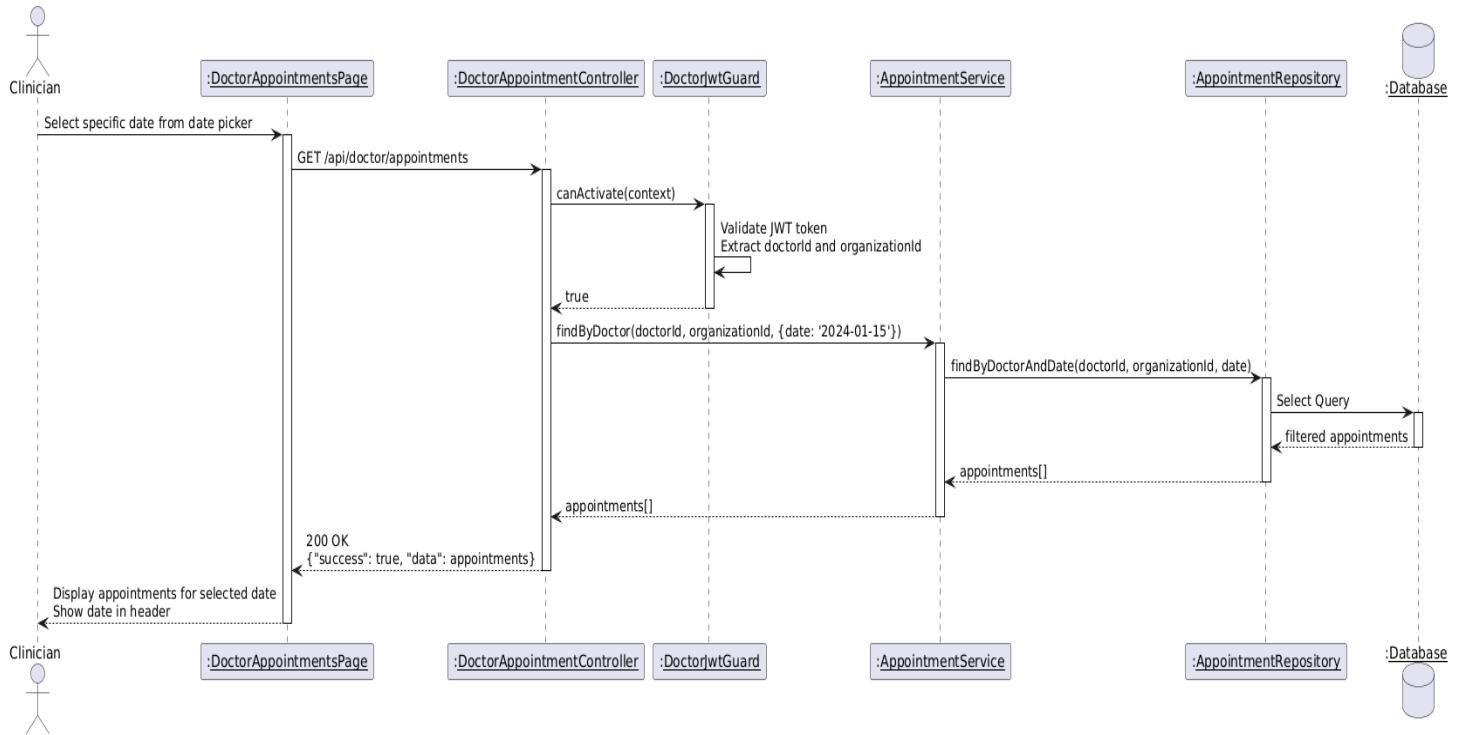


Diagram 91, Sequence Diagram (Filter Appointments Per Date)

- Filter Appointments Per Status:

<b>Use case ID</b>	<b>VEMR-FR-AM-45</b>
<b>Use case name</b>	Filter Appointments Per Status
<b>Description</b>	The system allows clinicians to filter appointments by status (scheduled, completed, cancelled, no-show).
<b>From</b>	Clinician
<b>Pre-conditions</b>	Clinician is authenticated
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. Clinician navigates to appointments page</li> <li>2. Clinician selects status from filter dropdown</li> <li>3. System validates JWT token via DoctorJwtGuard</li> <li>4. System retrieves appointments with selected status via AppointmentRepository</li> <li>5. System displays filtered appointments by status</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Unauthorized Access</b></p> <ul style="list-style-type: none"> <li>- At step 3, if JWT token is invalid or expired</li> <li>- System redirects to login page</li> </ul> <p><b>A2: No Appointments on Selected Status</b></p> <ul style="list-style-type: none"> <li>- At step 4, if no appointments exist for selected status</li> <li>- System displays "No appointments found for this status " message</li> </ul>
<b>Post condition</b>	Clinician can view appointments by status

Table 52, Use Case Specification ( Filter Appointments Per Status )

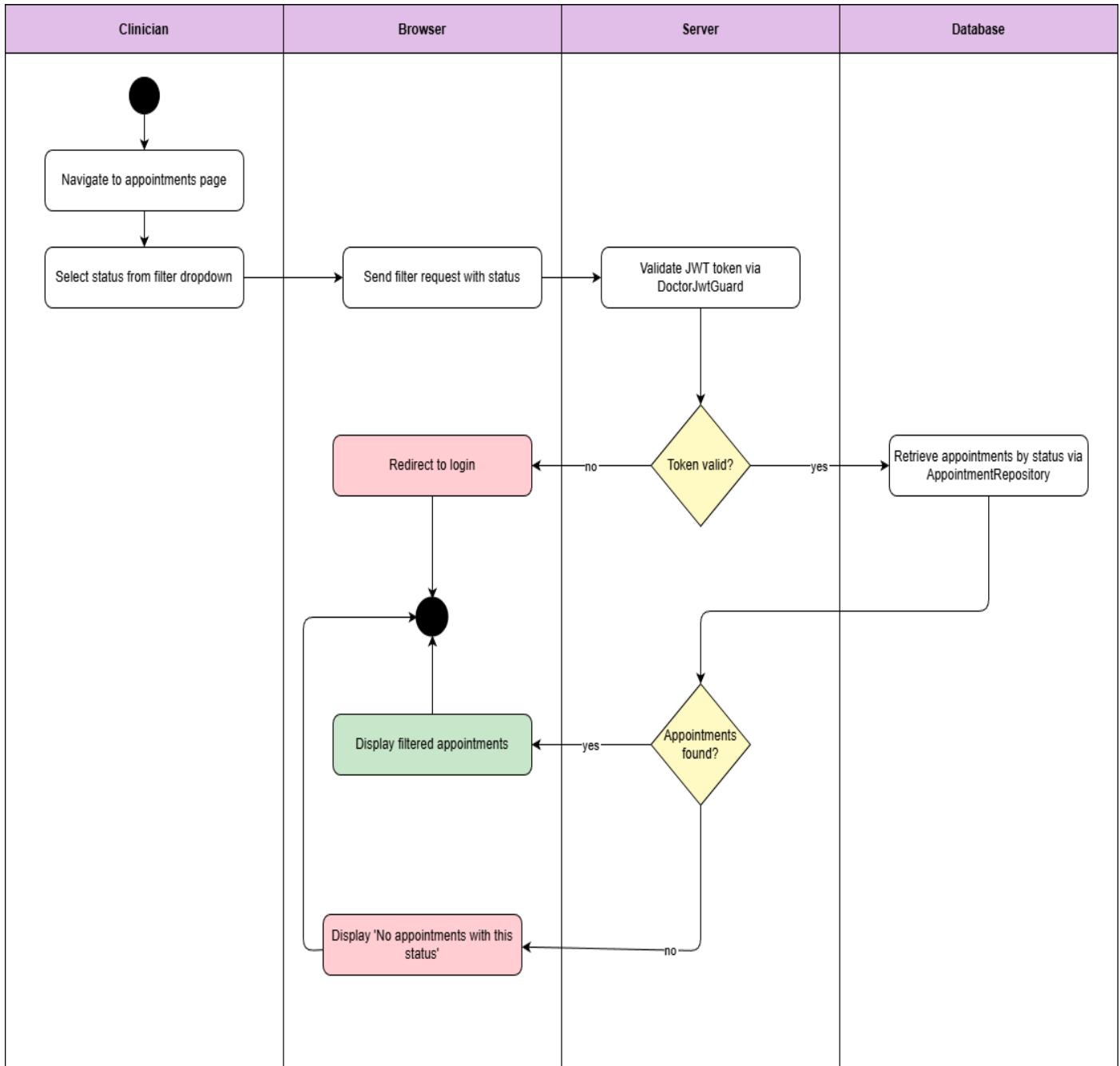


Diagram 92, Activity Diagram (Filter Appointments Per Status)

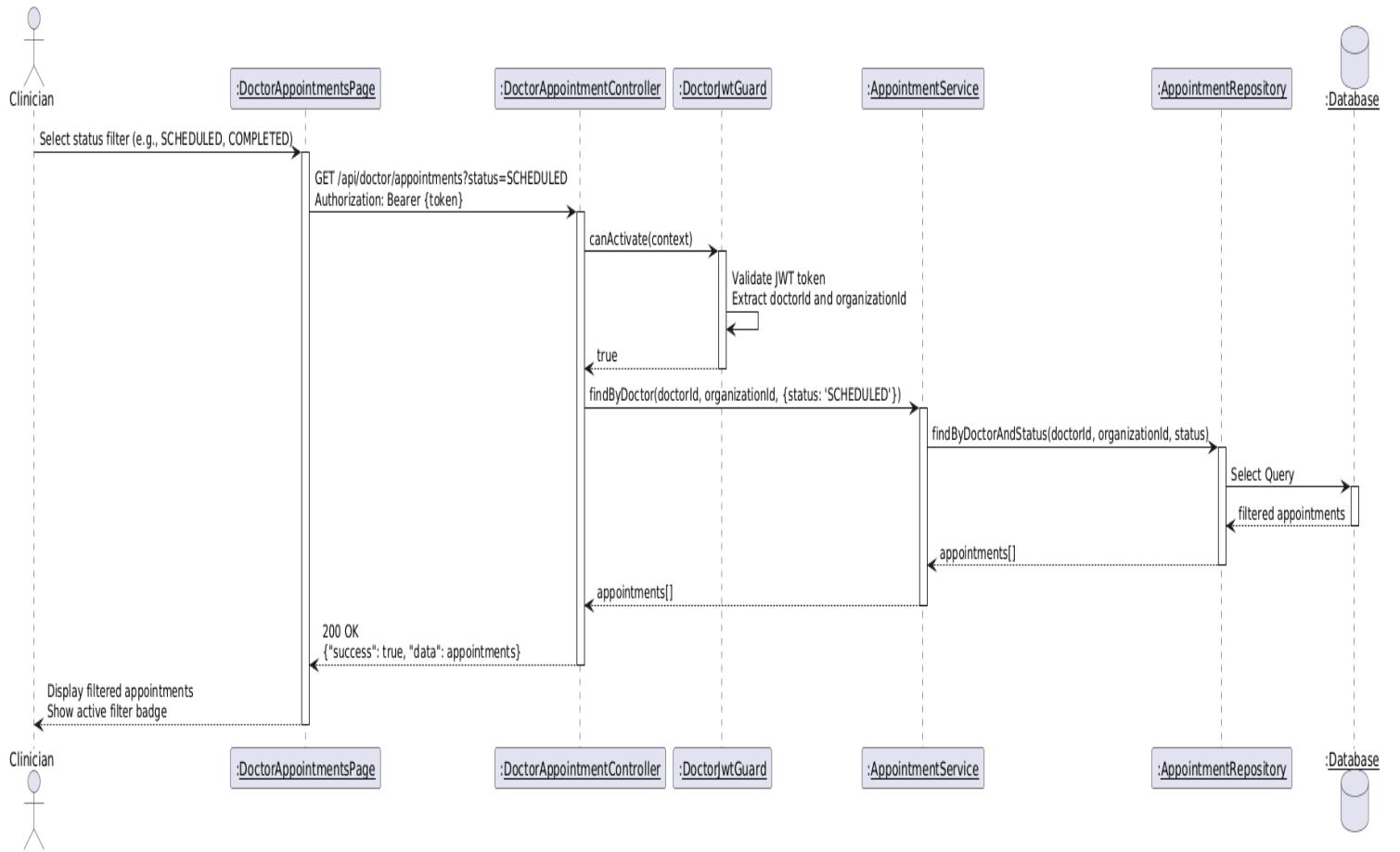


Diagram 93, Sequence Diagram (Filter Appointments Per Status)

- **View Appointment Details:**

<b>Use case ID</b>	<b>VEMR-FR-AM-46</b>
<b>Use case name</b>	View Appointment Details
<b>Description</b>	The system allows clinicians to view detailed information about a specific appointment
<b>From</b>	Clinician
<b>Pre-conditions</b>	Clinician is authenticated
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. Clinician navigates to appointments list</li> <li>2. Clinician clicks on appointment to view details</li> <li>3. System validates JWT token via DoctorJwtGuard</li> <li>4. System retrieves appointment details via AppointmentRepository</li> <li>5. System retrieves patient information via PatientRepository</li> <li>6. System displays appointment details with patient info, date, time, status, and notes</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Unauthorized Access</b></p> <ul style="list-style-type: none"> <li>- At step 3, if JWT token is invalid or expired</li> <li>- System redirects to login page</li> </ul> <p><b>A2: Appointment Not Found</b></p> <ul style="list-style-type: none"> <li>- At step 4, if appointment ID does not exist</li> <li>- System displays error message</li> </ul>
<b>Post condition</b>	Clinician can view patient and appointment information

Table 53, Use Case Specification ( View Appointment Details)

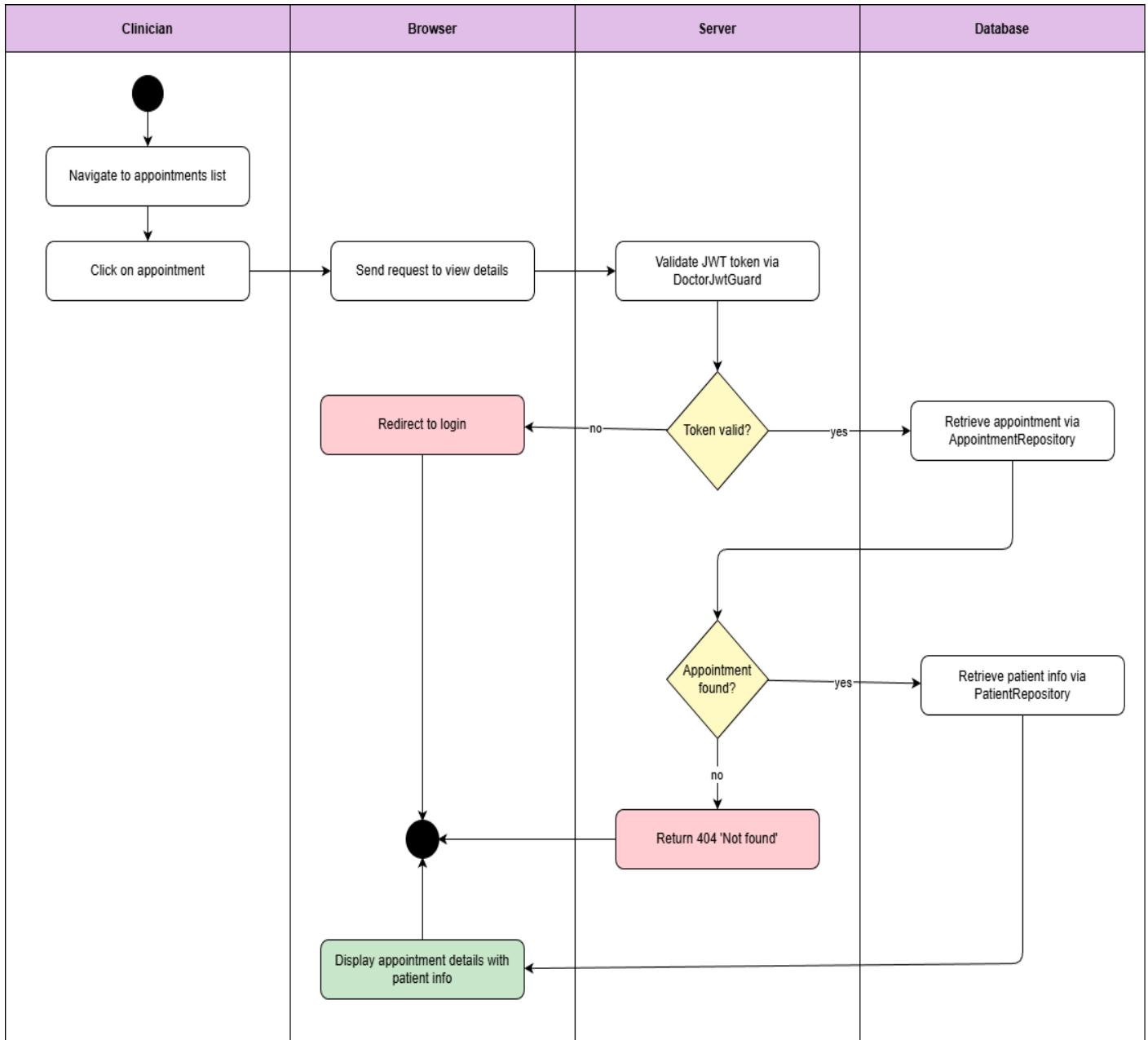


Diagram 94, Activity Diagram (View Appointment Details)

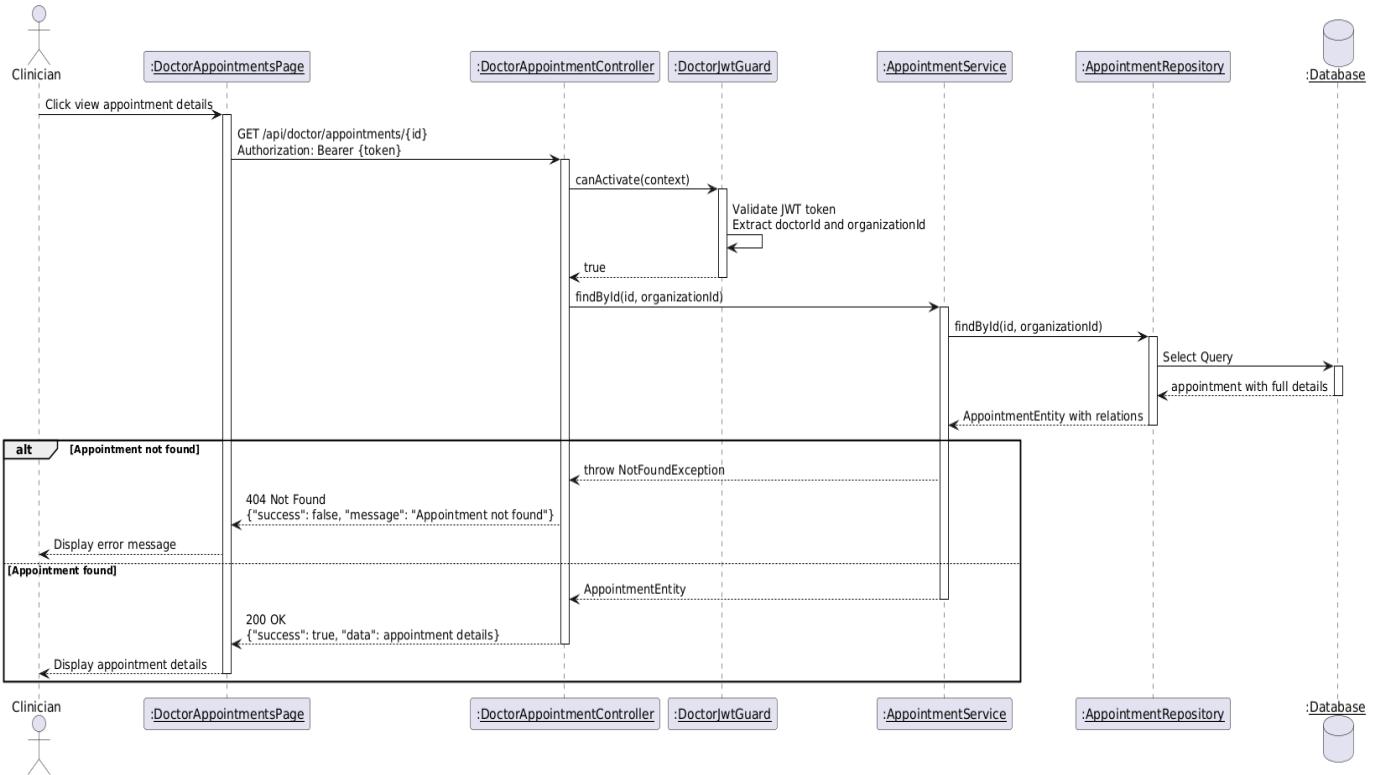


Diagram 95, Sequence Diagram (View Appointment Details)

- **Check In an Appointment:**

<b>Use case ID</b>	<b>VEMR-FR-AM-47</b>
<b>Use case name</b>	Check In an Appointment
<b>Description</b>	The system allows clinicians to check in a patient for their appointment.
<b>From</b>	Clinician
<b>Pre-conditions</b>	Clinician is authenticated
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. Clinician navigates to appointments list</li> <li>2. Clinician selects appointment</li> <li>3. Clinician clicks "Check In" button</li> <li>4. System validates JWT token via DoctorJwtGuard</li> <li>5. System verifies appointment status is scheduled</li> <li>6. System updates appointment status to "checked-in" via AppointmentRepository</li> <li>7. System displays success message and refreshes appointments list</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Unauthorized Access</b></p> <ul style="list-style-type: none"> <li>- At step 4, if JWT token is invalid or expired</li> <li>- System redirects to login page</li> </ul> <p><b>A2: Appointment Not Found</b></p> <ul style="list-style-type: none"> <li>- At step 5, if appointment ID does not exist</li> <li>- System displays error message</li> </ul> <p><b>A3: Invalid Status</b></p> <ul style="list-style-type: none"> <li>- At step 5, if appointment is not in scheduled status</li> <li>- System displays error message</li> </ul>
<b>Post condition</b>	Appointment status is updated to checked-in

Table 54, Use Case Specification ( Check In an Appointment)

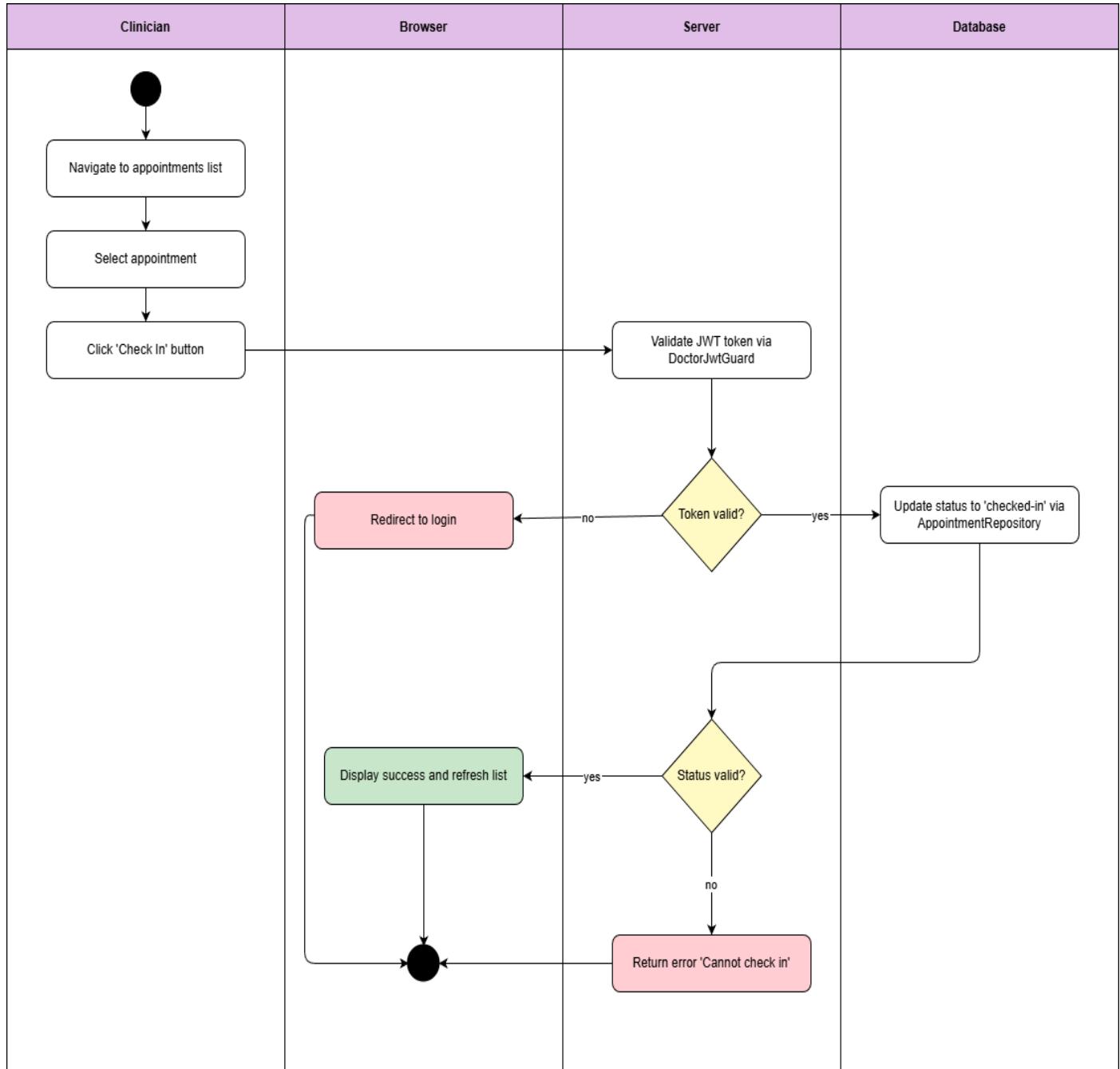


Diagram 96, Activity Diagram (Check In an Appointment)

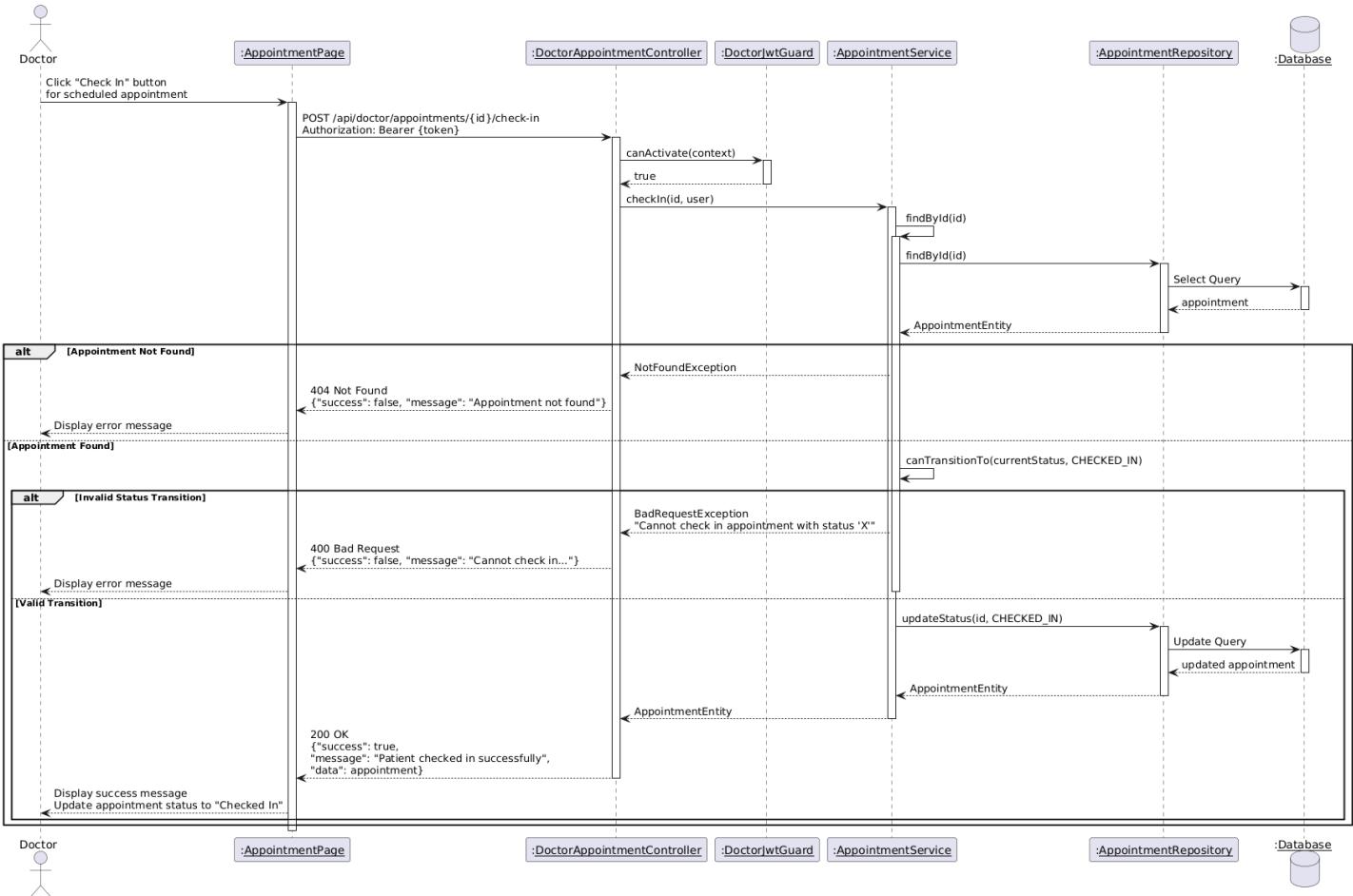


Diagram 97, Sequence Diagram (Check In an Appointment)

## ● Cancel an Appointment :

<b>Use case ID</b>	<b>VEMR-FR-AM-48</b>
<b>Use case name</b>	Cancel an Appointment
<b>Description</b>	The system allows clinicians or patients to cancel an appointment
<b>From</b>	Clinician
<b>Pre-conditions</b>	Clinician is authenticated
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. User navigates to appointments list</li> <li>2. User selects appointment</li> <li>3. User clicks "Cancel" button</li> <li>4. System displays cancellation confirmation dialog</li> <li>5. User confirms cancellation</li> <li>6. System validates JWT token via JwtGuard</li> <li>7. System verifies appointment can be cancelled</li> <li>8. System updates appointment status to "cancelled" via AppointmentRepository</li> <li>9. System releases appointment slot</li> <li>10. System displays success message and refreshes appointments list</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Unauthorized Access</b></p> <ul style="list-style-type: none"> <li>- At step 4, if JWT token is invalid or expired</li> <li>- System redirects to login page</li> </ul> <p><b>A2: Appointment Not Found</b></p> <ul style="list-style-type: none"> <li>- At step 7, if appointment ID does not exist</li> <li>- System displays error message</li> </ul> <p><b>A3: Cannot Cancel</b></p> <ul style="list-style-type: none"> <li>- At step 7, if appointment is already completed</li> <li>- System displays error message</li> </ul>
<b>Post condition</b>	Appointment status is updated to checked-Appointment status is updated to cancelled

Table 55, Use Case Specification ( Cancel an Appointment )

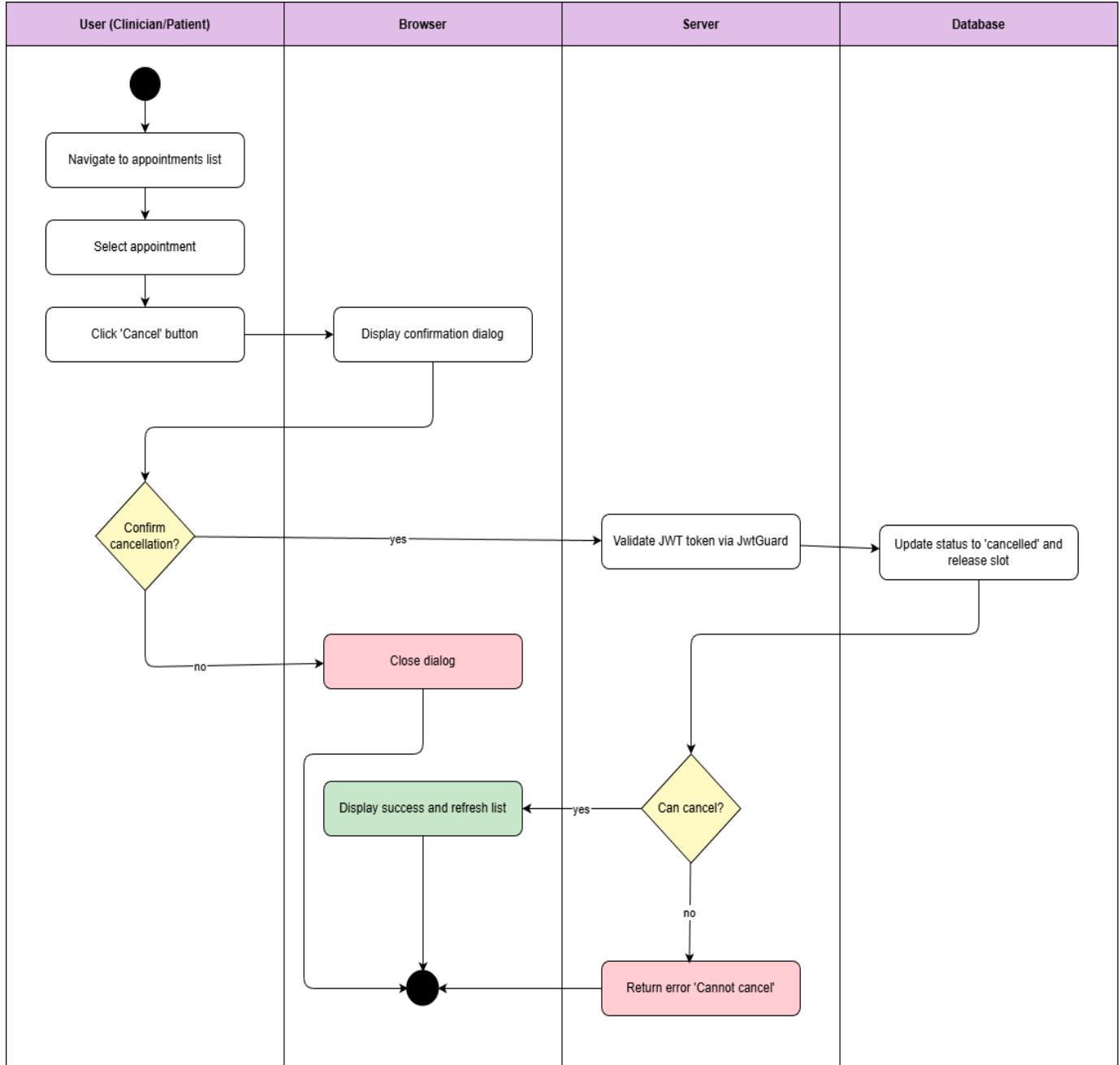


Diagram 98, Activity Diagram (Cancel an Appointment )

## Chapter 4 - System Analysis

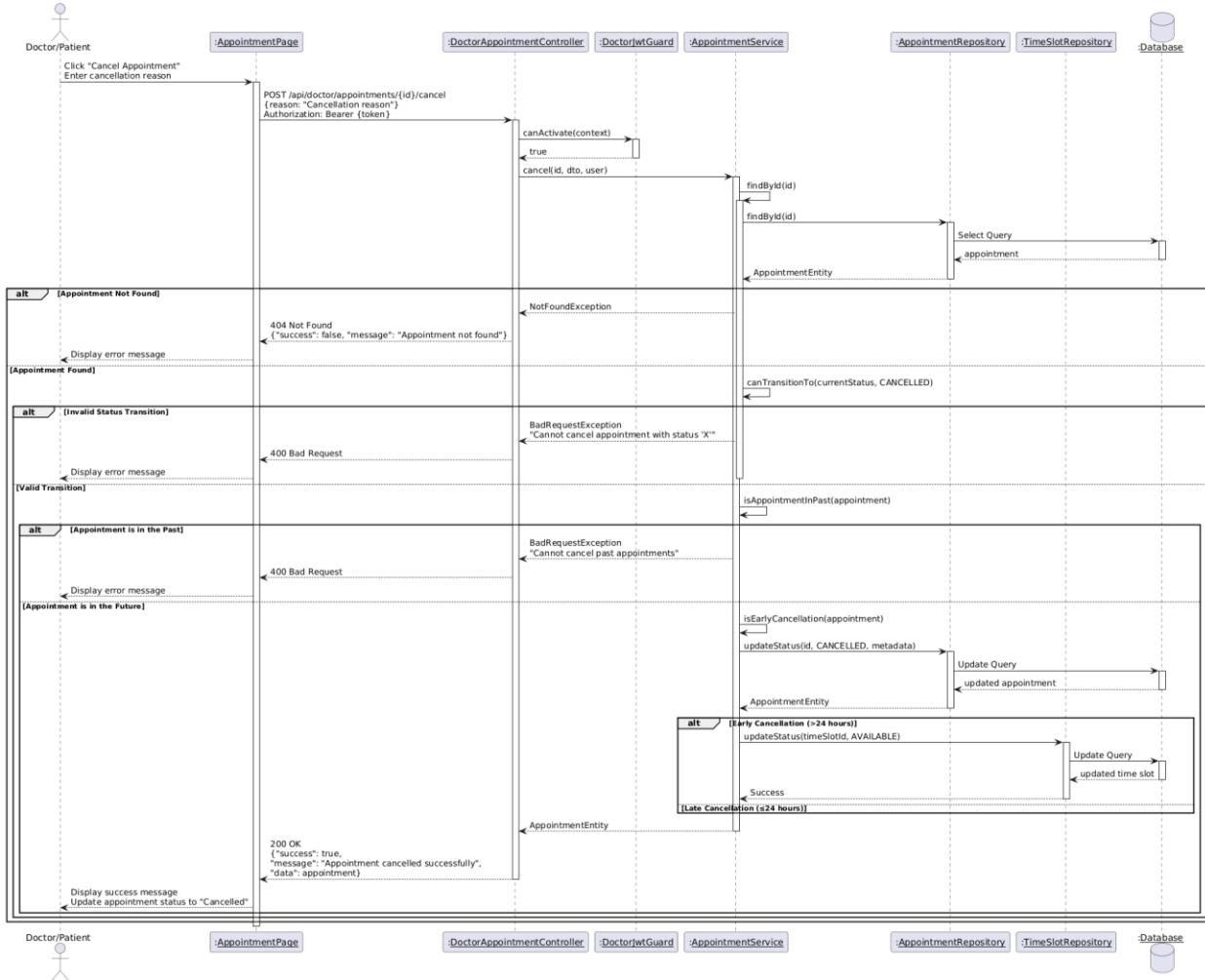


Diagram 99, Sequence Diagram (Cancel an Appointment )

- **Set an Appointment as No Show :**

<b>Use case ID</b>	<b>VEMR-FR-AM-49</b>
<b>Use case name</b>	Set an Appointment as No Show
<b>Description</b>	The system allows clinicians to mark an appointment as no-show when patient doesn't arrive.
<b>From</b>	Clinician
<b>Pre-conditions</b>	Clinician is authenticated
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>11. Clinician navigates to appointments list</li> <li>2. Clinician selects appointment</li> <li>3. Clinician clicks "Mark as No Show" button</li> <li>4. System validates JWT token via DoctorJwtGuard</li> <li>5. System verifies appointment status</li> <li>6. System updates appointment status to "no-show" via AppointmentRepository</li> <li>7. System releases appointment slot</li> <li>8. System displays success message and refreshes appointments list</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Unauthorized Access</b></p> <ul style="list-style-type: none"> <li>- At step 4, if JWT token is invalid or expired</li> <li>- System redirects to login page</li> </ul> <p><b>A2: Appointment Not Found</b></p> <ul style="list-style-type: none"> <li>- At step 5, if appointment ID does not exist</li> <li>- System displays error message</li> </ul> <p><b>A3: Invalid Status</b></p> <ul style="list-style-type: none"> <li>- At step 5, if appointment is already completed or cancelled</li> <li>- System displays error message</li> </ul>
<b>Post condition</b>	-Appointment status is updated to no-show

Table 56, Use Case Specification ( Set an Appointment as No Show )

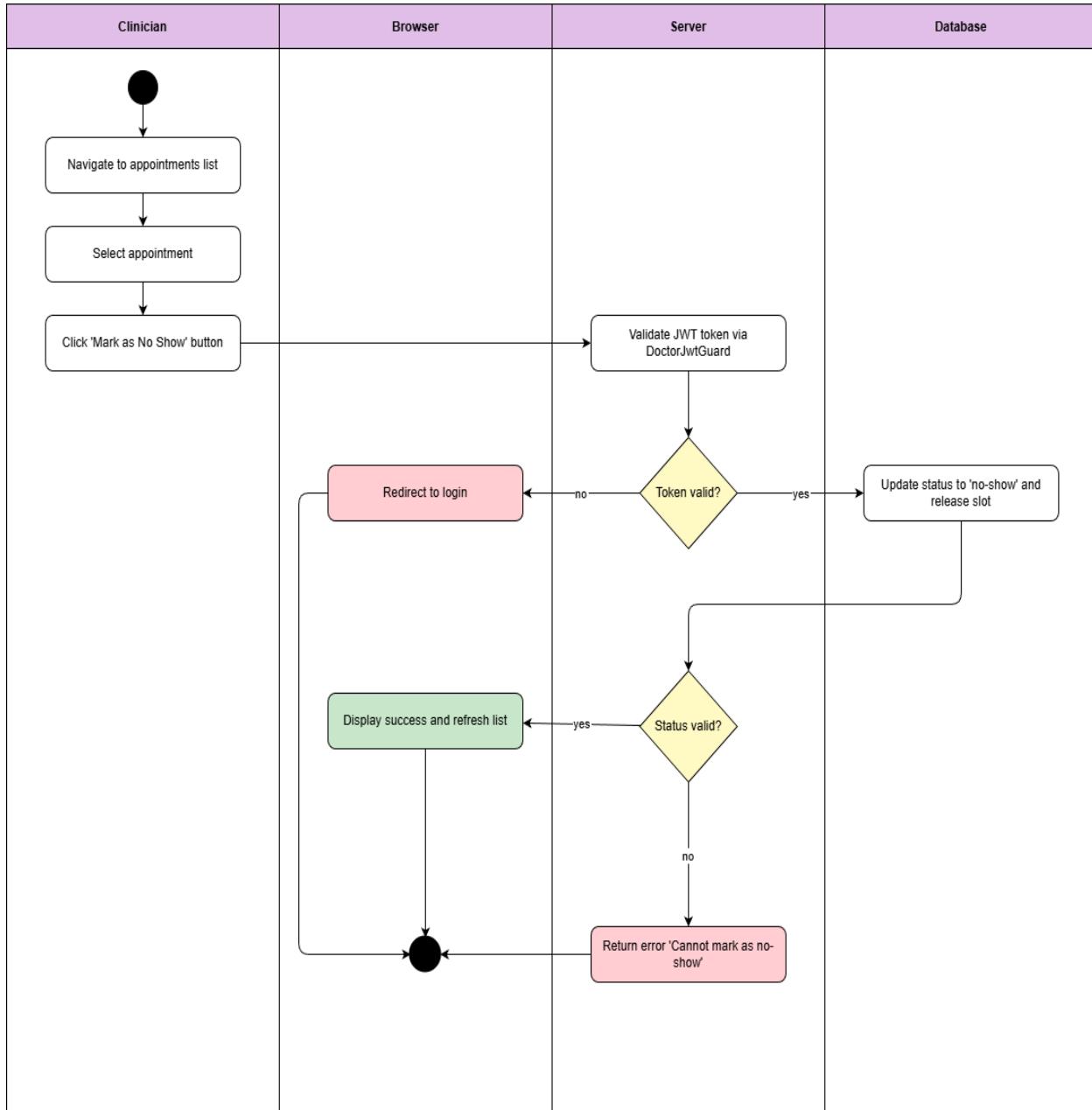


Diagram 100, Activity Diagram (Set an Appointment as No Show)

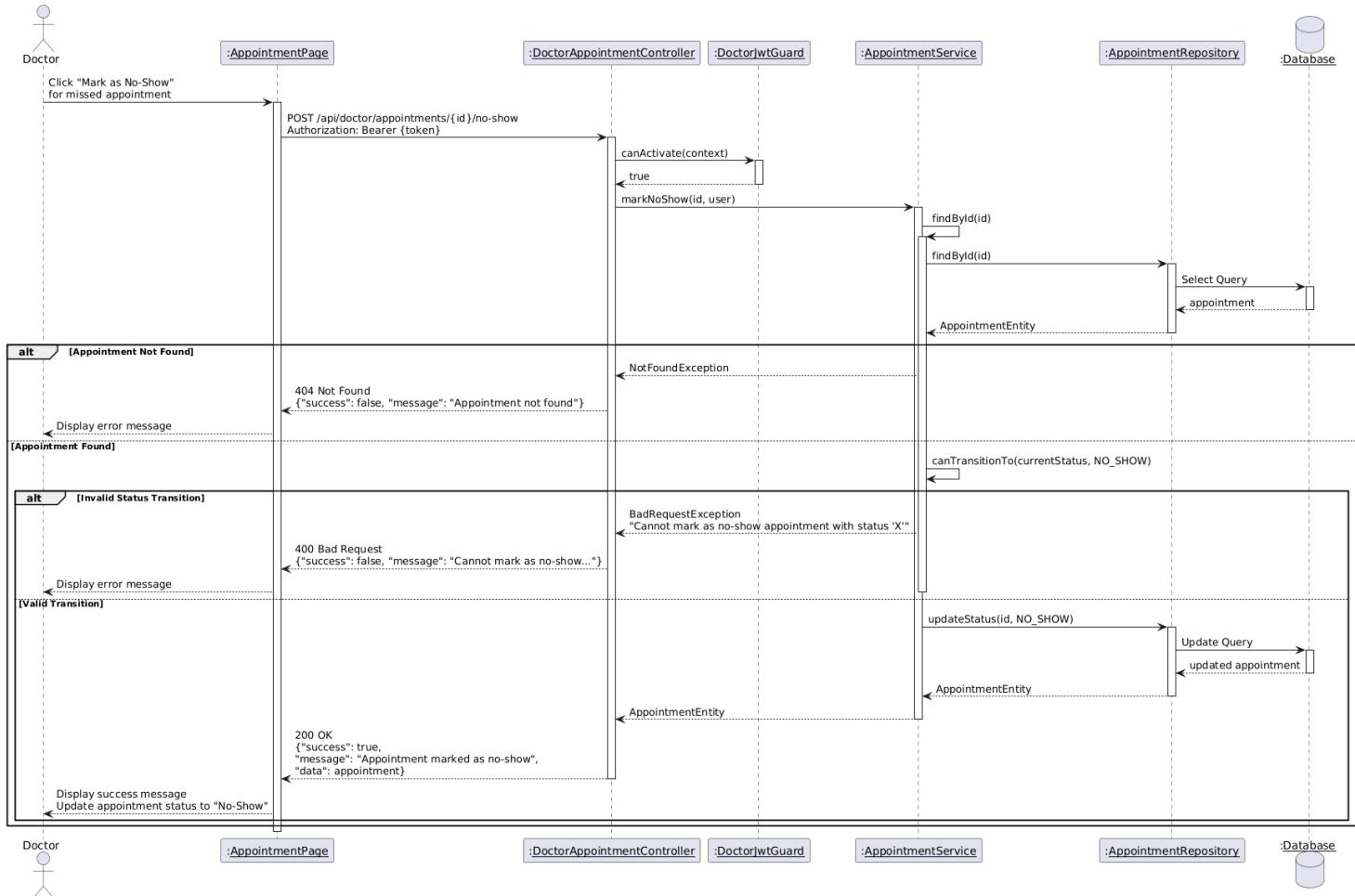


Diagram 101, Sequence Diagram (Set an Appointment as No Show)

- **View Doctor Analytics :**

<b>Use case ID</b>	<b>VEMR-FR-AM-50</b>
<b>Use case name</b>	View Doctor Analytics
<b>Description</b>	The system allows clinicians to view their personal analytics and performance metrics.
<b>From</b>	Clinician
<b>Pre-conditions</b>	Clinician is authenticated
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. Clinician navigates to analytics dashboard</li> <li>2. System validates JWT token via DoctorJwtGuard</li> <li>3. System retrieves total appointments count via AppointmentRepository</li> <li>4. System retrieves completed visits count via EncounterRepository</li> <li>5. System calculates appointment statistics by status</li> <li>6. System displays analytics dashboard with charts and metrics</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Unauthorized Access</b></p> <ul style="list-style-type: none"> <li>- At step 2, if JWT token is invalid or expired</li> <li>- System redirects to login page</li> </ul> <p><b>A2: No Data Available</b></p> <ul style="list-style-type: none"> <li>- At steps 3-4, if no data exists</li> <li>- System shows empty state message</li> </ul>
<b>Post condition</b>	Clinician can view personal statistics

Table 57, Use Case Specification ( View Doctor Analytics )

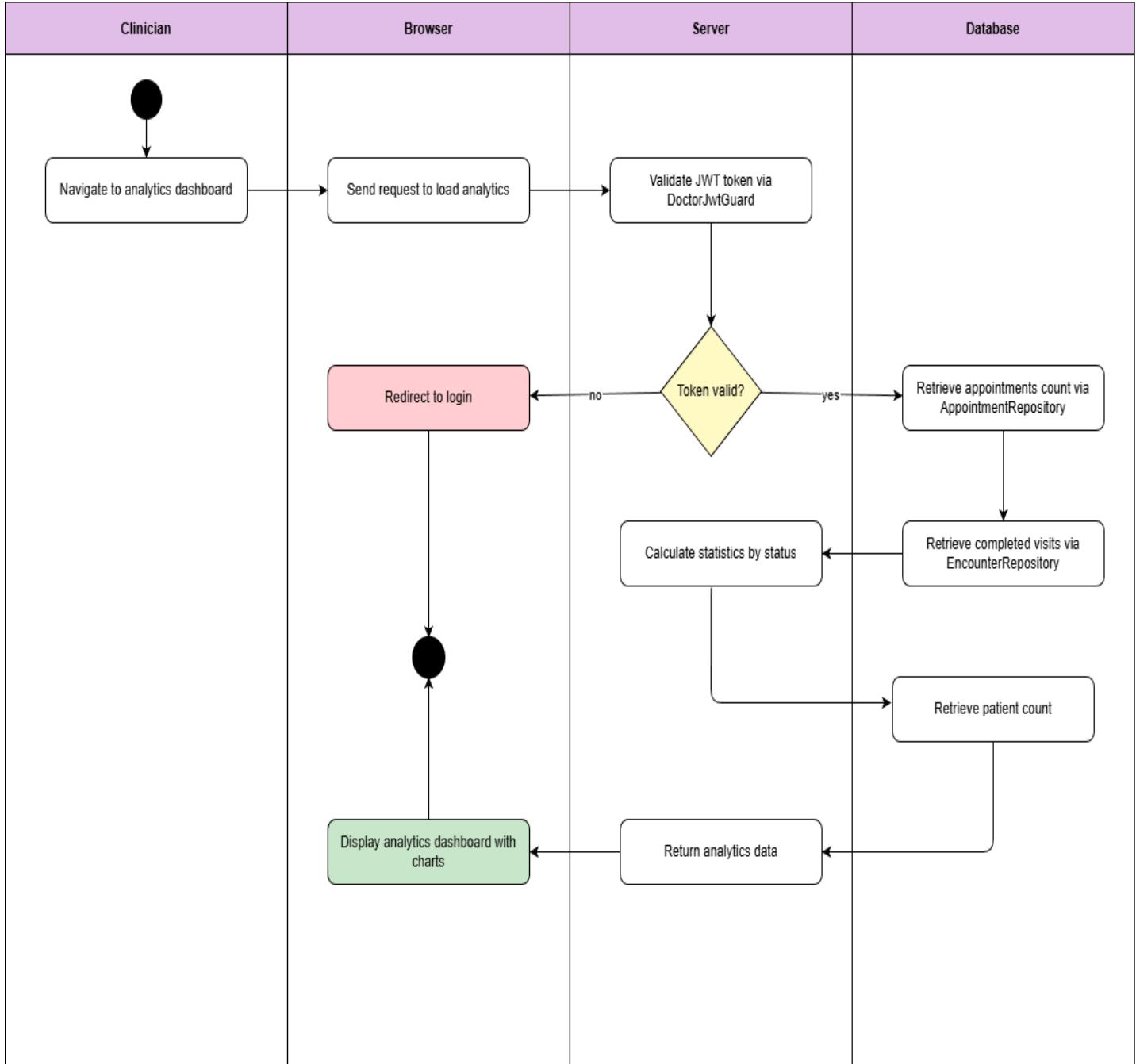


Diagram 102, Activity Diagram ( View Doctor Analytics )

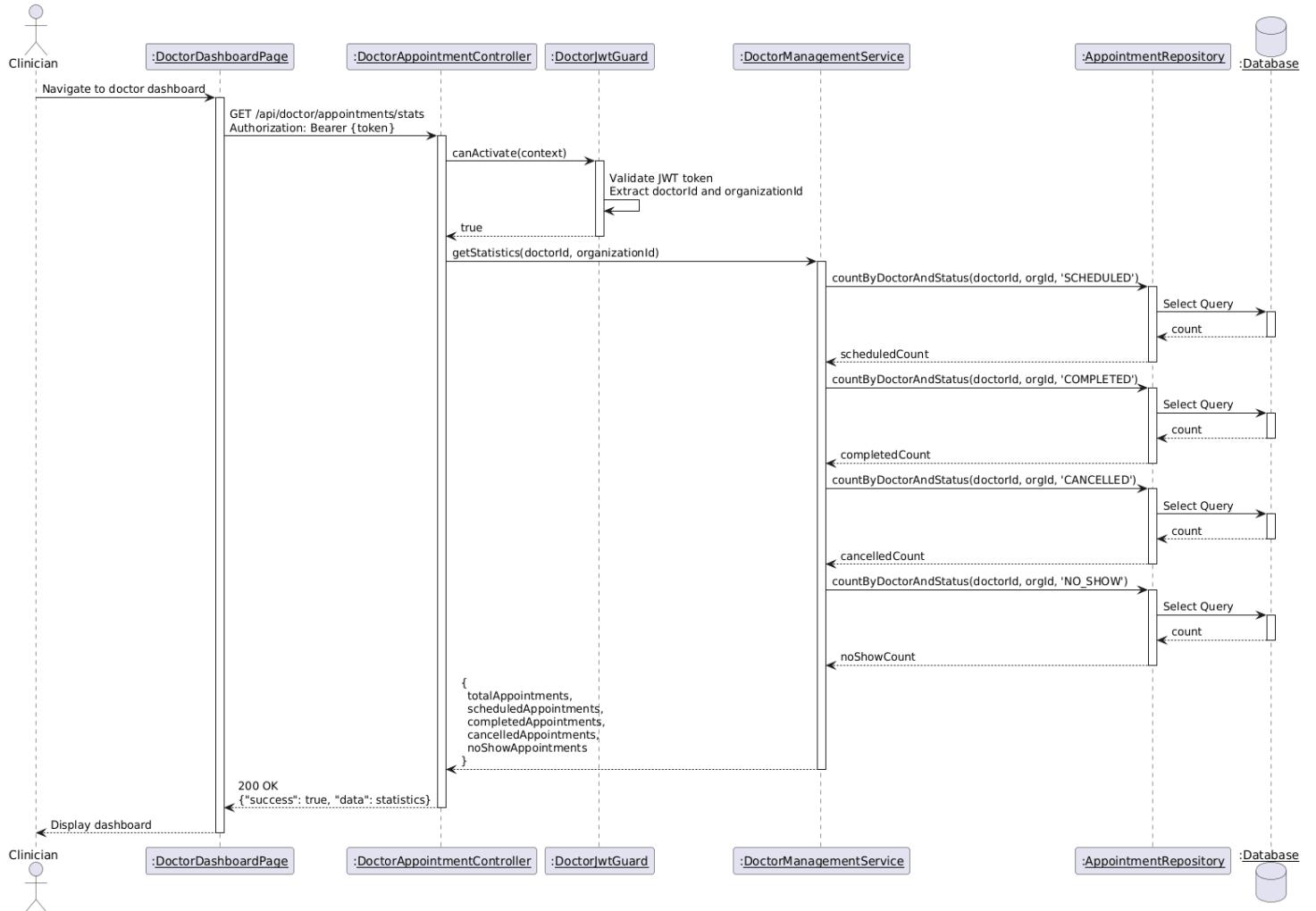


Diagram 103, Sequence Diagram ( View Doctor Analytics )

- **Create Schedule:**

<b>Use case ID</b>	<b>VEMR-FR-AM-51</b>
<b>Use case name</b>	Create Schedule
<b>Description</b>	The system allows clinicians to create their work schedule
<b>From</b>	Clinician
<b>Pre-conditions</b>	Clinician is authenticated
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. Clinician navigates to schedule management page</li> <li>2. Clinician clicks "Create Schedule" button</li> <li>3. System displays schedule creation form</li> <li>4. Clinician enters schedule details (day, start time, end time)</li> <li>5. Clinician submits form</li> <li>6. System validates input fields</li> <li>7. System validates JWT token via DoctorJwtGuard</li> <li>8. System creates schedule record via ScheduleRepository</li> <li>9. System displays success message and refreshes schedule list</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Unauthorized Access</b></p> <ul style="list-style-type: none"> <li>- At step 7, if JWT token is invalid or expired</li> <li>- System redirects to login page</li> </ul> <p><b>A2: Validation Error</b></p> <ul style="list-style-type: none"> <li>- At step 6, if required fields are empty or invalid</li> <li>- System displays validation error messages</li> </ul>
<b>Post condition</b>	New schedule is created in the system

Table 58, Use Case Specification ( Create Schedule )

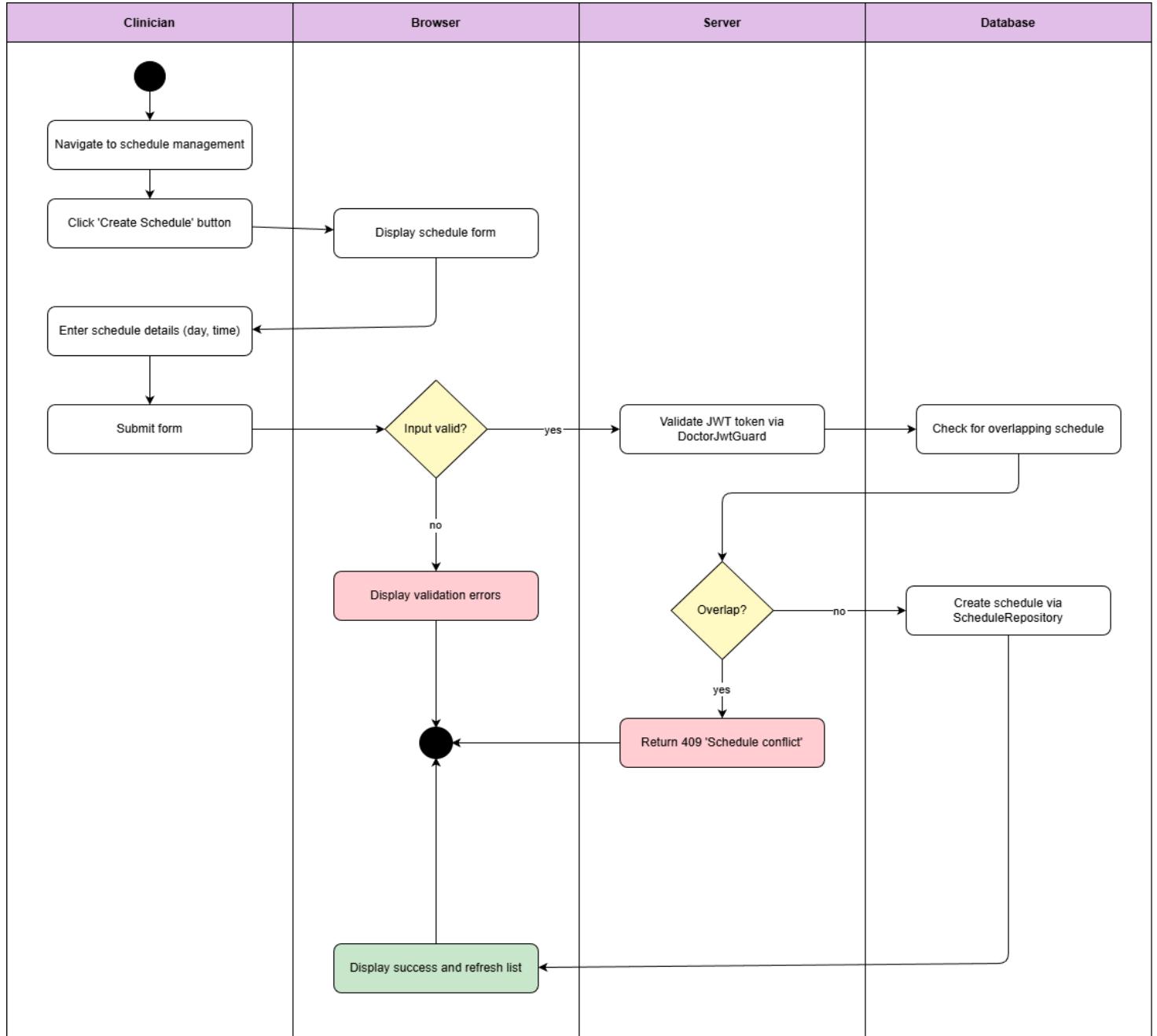


Diagram 104, Activity Diagram ( Create Schedule )

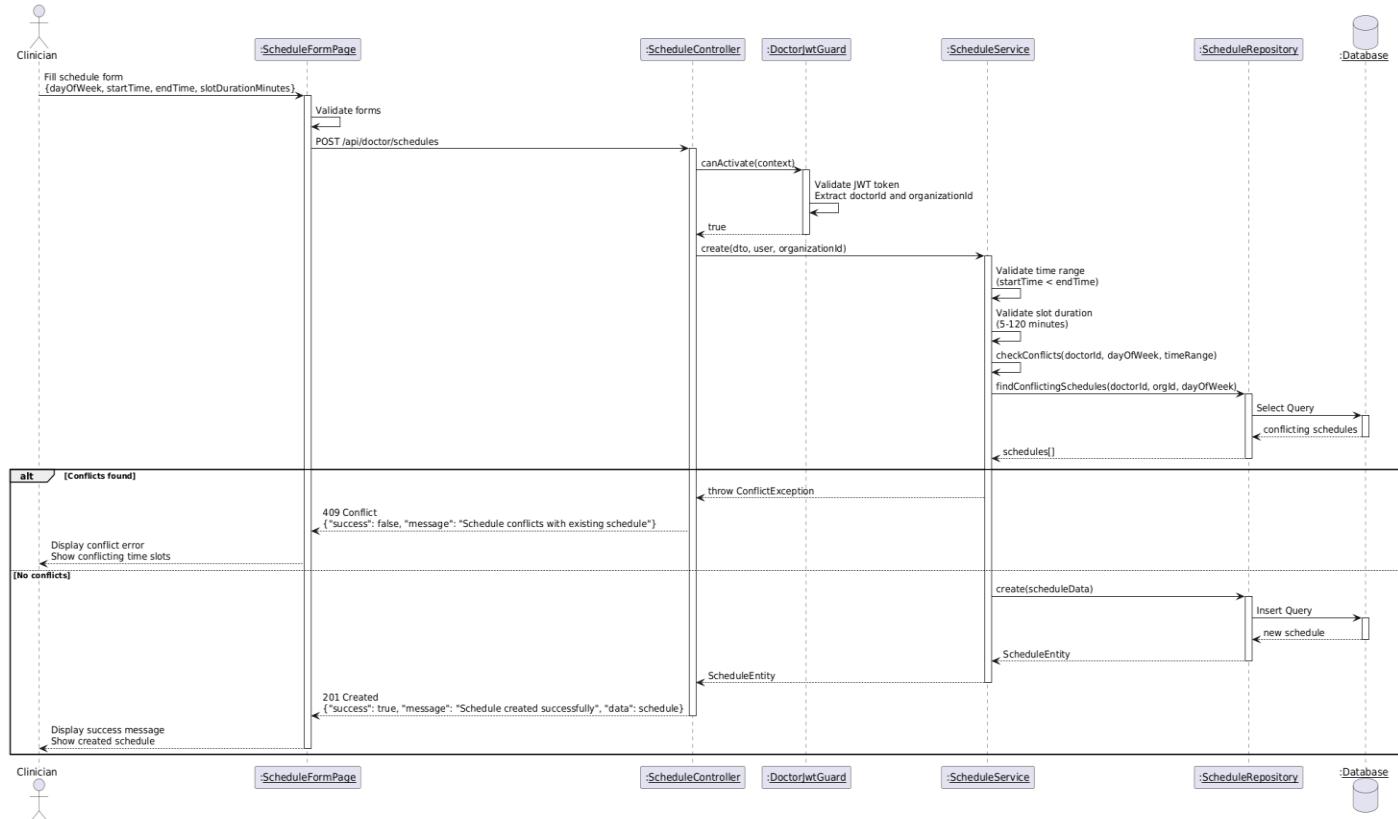


Diagram 105, Sequence Diagram ( Create Schedule )

● Edit Schedule :

<b>Use case ID</b>	VEMR-FR-AM-52
<b>Use case name</b>	Edit Schedule
<b>Description</b>	The system allows clinicians to edit their existing work schedule.
<b>From</b>	Clinician
<b>Pre-conditions</b>	Clinician is authenticated
<b>Main scenario</b>	<ol style="list-style-type: none"><li>1. Clinician navigates to schedule list</li><li>2. Clinician clicks "Edit" button on schedule</li><li>3. System displays edit form with current data</li><li>4. Clinician modifies schedule details</li><li>5. Clinician submits form</li><li>6. System validates input fields</li><li>7. System validates JWT token via DoctorJwtGuard</li><li>8. System updates schedule via ScheduleRepository</li><li>9. System displays success message and refreshes schedule list</li></ol>

<b>Alternative scenario</b>	<p><b>A1: Unauthorized Access</b></p> <ul style="list-style-type: none"> <li>- At step 7, if JWT token is invalid or expired</li> <li>- System redirects to login page</li> </ul> <p><b>A2: Validation Error</b></p> <ul style="list-style-type: none"> <li>- At step 6, if required fields are empty or invalid</li> <li>- System displays validation error messages</li> </ul> <p><b>A3: Schedule Not Found</b></p> <ul style="list-style-type: none"> <li>- At step 8, if schedule ID does not exist</li> <li>- System displays error message</li> </ul>
<b>Post condition</b>	Schedule information is updated in the system

Table 59, Use Case Specification ( Edit Schedule )

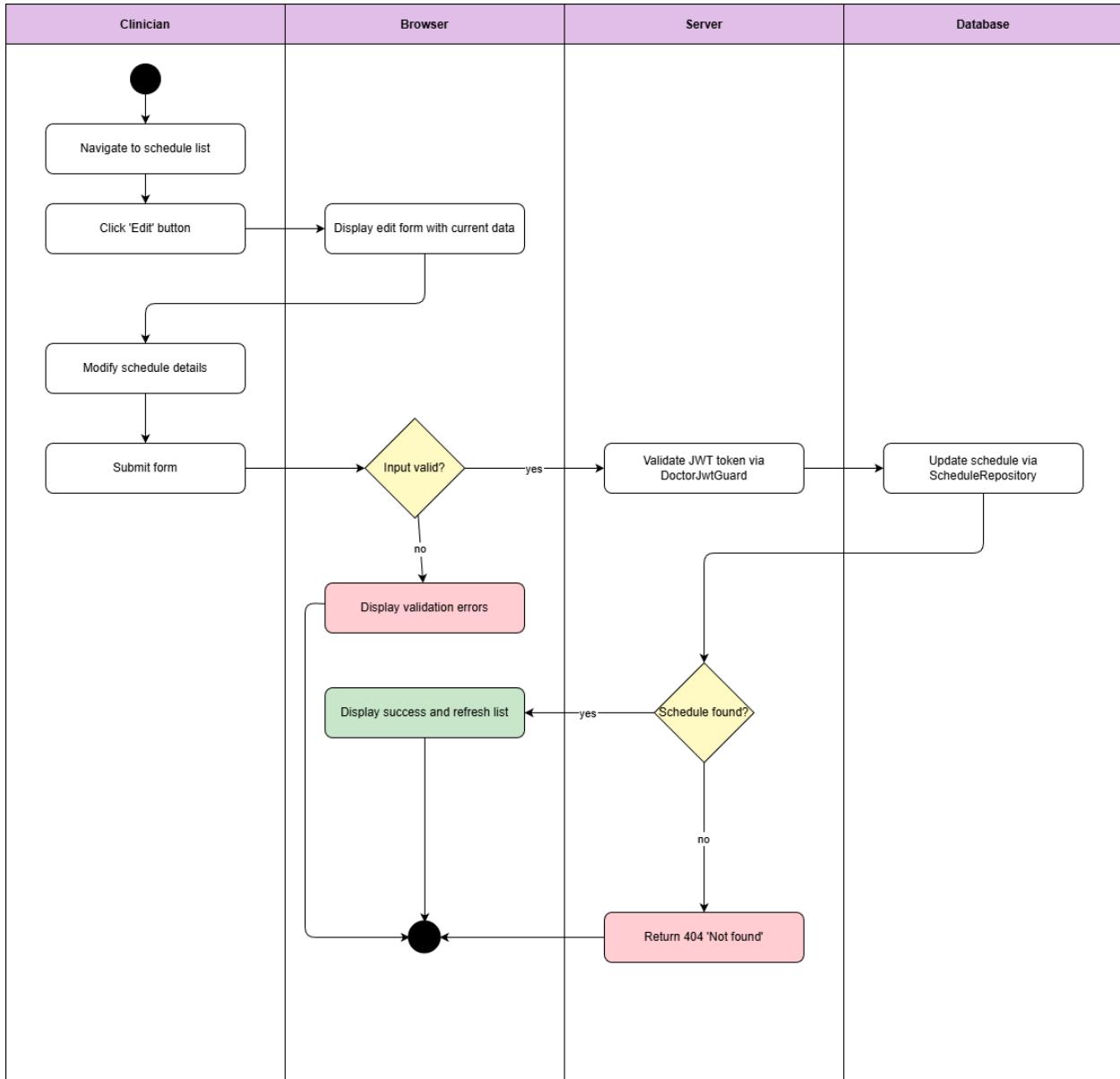


Diagram 106, Activity Diagram ( Edit Schedule )

## Chapter 4 - System Analysis

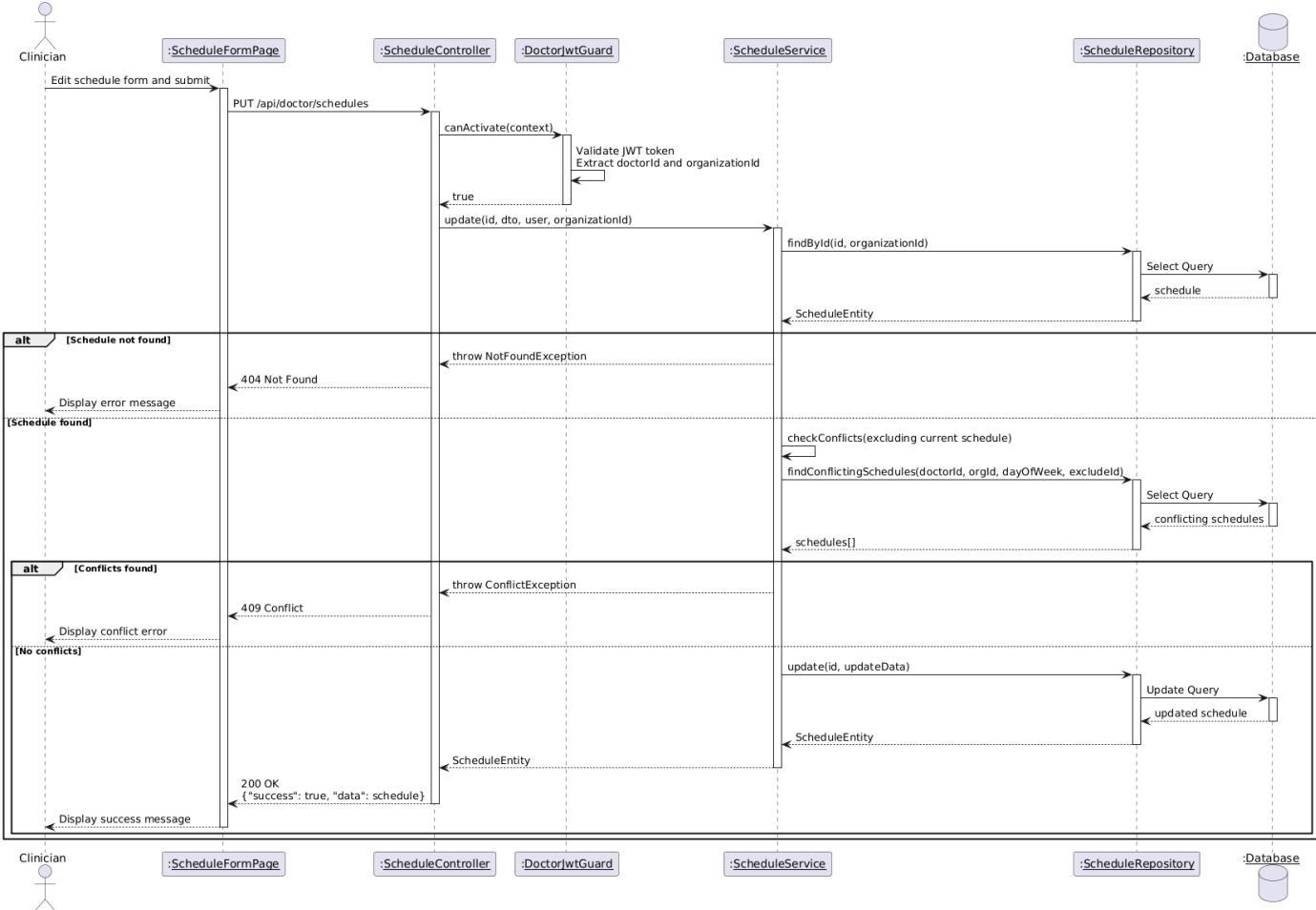


Diagram 107, Sequence Diagram ( Edit Schedule )

## ● Delete Schedule :

<b>Use case ID</b>	VEMR-FR-AM-53
<b>Use case name</b>	Delete Schedule
<b>Description</b>	The system allows clinicians to delete their work schedule.
<b>From</b>	Clinician
<b>Pre-conditions</b>	Clinician is authenticated
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. Clinician navigates to schedule list</li> <li>2. Clinician clicks "Delete" button on schedule</li> <li>3. System displays confirmation dialog</li> <li>4. Clinician confirms deletion</li> <li>5. System validates JWT token via DoctorJwtGuard</li> <li>6. System deletes schedule via ScheduleRepository</li> <li>7. System displays success message and refreshes schedule list</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Unauthorized Access</b></p> <ul style="list-style-type: none"> <li>- At step 5, if JWT token is invalid or expired</li> <li>- System redirects to login page</li> </ul> <p><b>A2: Schedule Not Found</b></p> <ul style="list-style-type: none"> <li>- At step 6, if schedule ID does not exist</li> <li>- System displays error message</li> </ul> <p><b>A3: Deletion Cancelled</b></p> <ul style="list-style-type: none"> <li>- At step 4, if clinician cancels confirmation</li> <li>- No deletion occurs</li> </ul>
<b>Post condition</b>	Schedule is removed from the system

Table 60, Use Case Specification ( Delete Schedule )

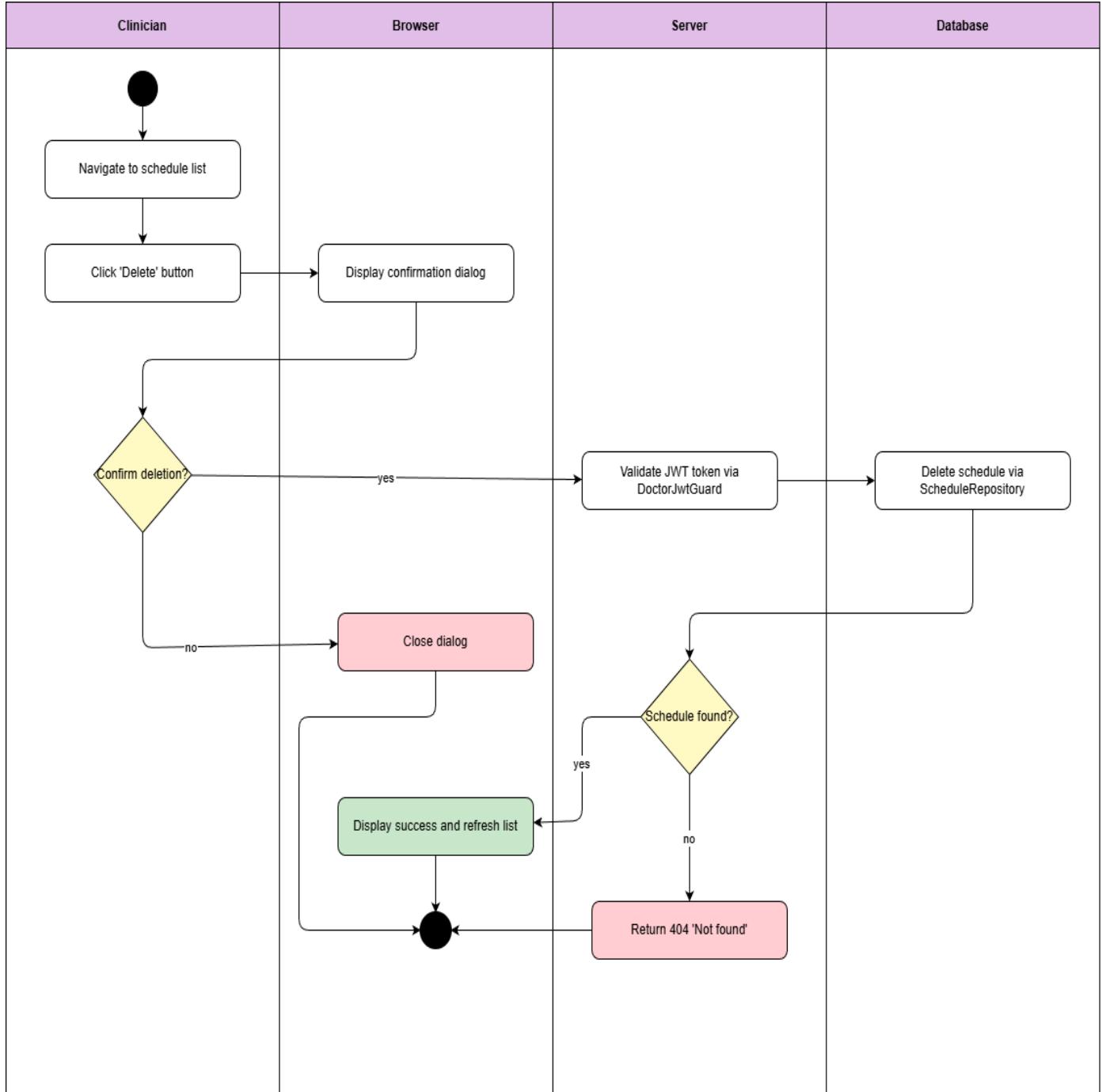


Diagram 108, Activity Diagram ( Delete Schedule )

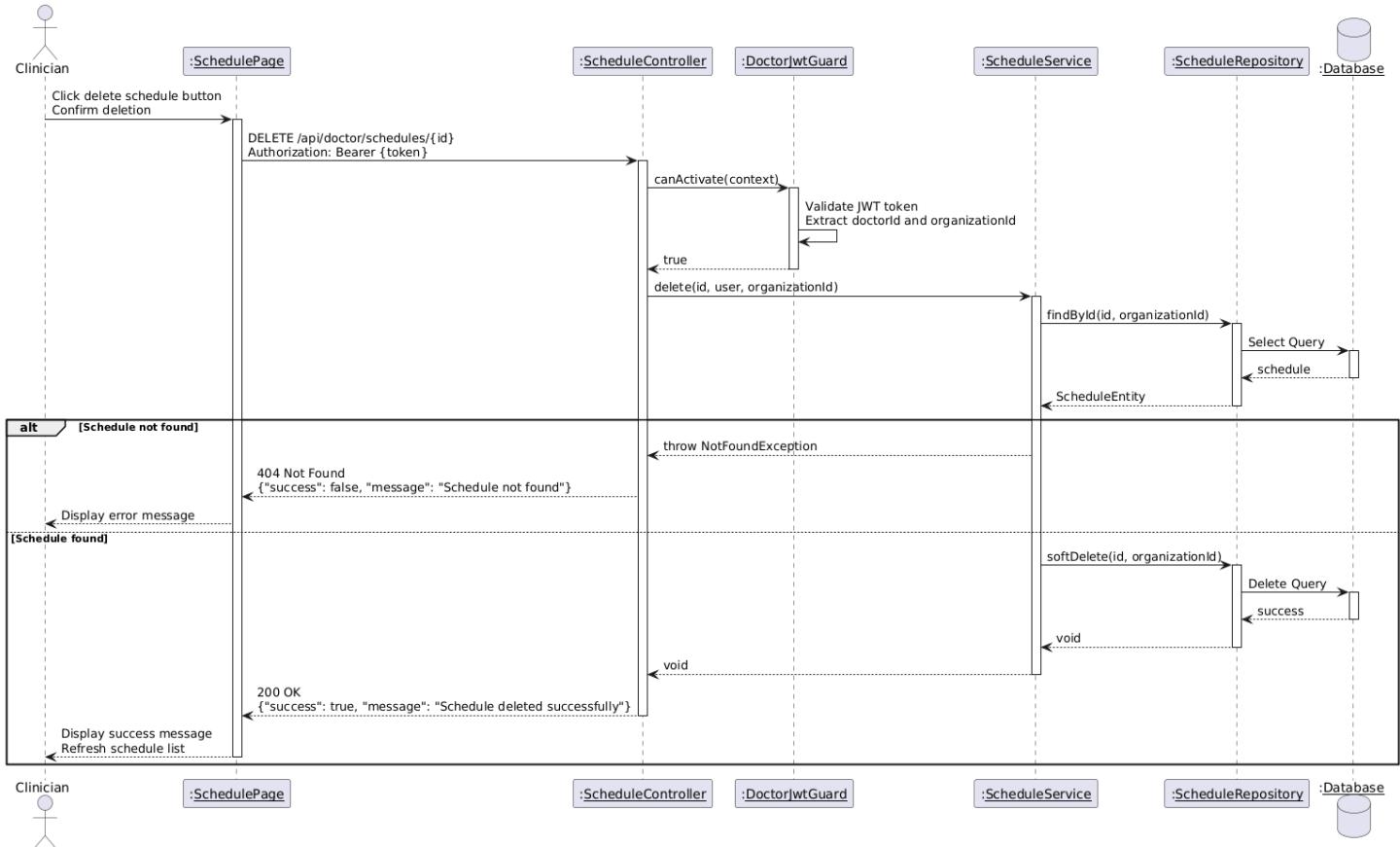


Diagram 109, Activity Diagram ( Delete Schedule )

## ● Generate Visit Slots :

<b>Use case ID</b>	<b>VEMR-FR-AM-54</b>
<b>Use case name</b>	Generate Visit Slots
<b>Description</b>	The system allows clinicians to automatically generate available visit slots based on their schedule.
<b>From</b>	Clinician
<b>Pre-conditions</b>	Clinician is authenticated
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. Clinician navigates to schedule management page</li> <li>2. Clinician clicks "Generate Slots" button</li> <li>3. System validates JWT token via DoctorJwtGuard</li> <li>4. System retrieves clinician schedules via ScheduleRepository</li> <li>5. System generates time slots based on schedule and slot duration</li> <li>6. System creates available slots via SlotRepository</li> <li>7. System displays success message with generated slots count</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Unauthorized Access</b></p> <ul style="list-style-type: none"> <li>- At step 3, if JWT token is invalid or expired</li> <li>- System redirects to login page</li> </ul> <p><b>A2: No Schedule Found</b></p> <ul style="list-style-type: none"> <li>- At step 4, if schedule ID does not exist</li> <li>- System displays "Please create schedule first" message</li> </ul>
<b>Post condition</b>	Available visit slots are generated

Table 61, Use Case Specification ( Generate Visit Slots)

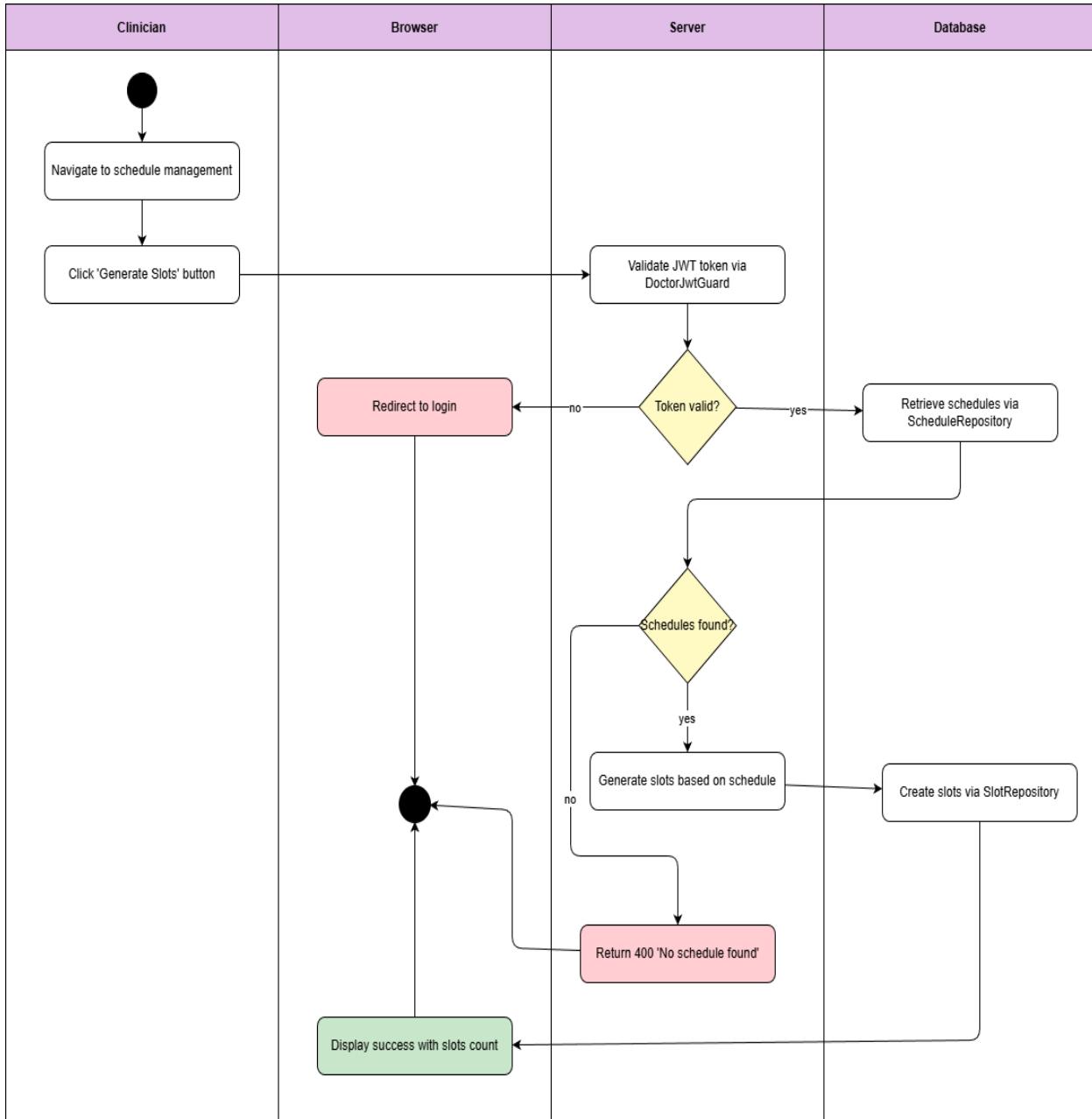


Diagram 110, Activity Diagram ( Generate Visit Slots )

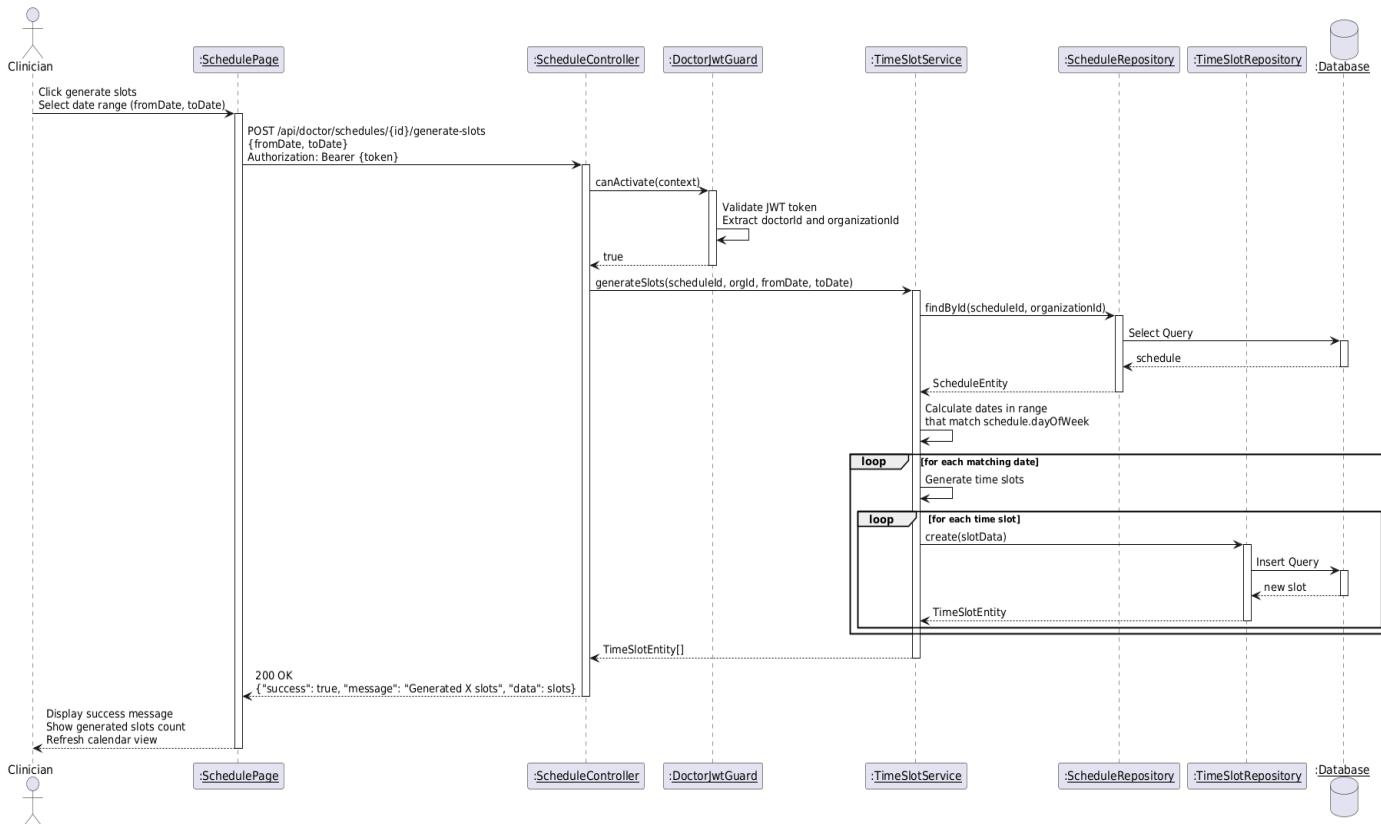


Diagram 111, Sequence Diagram ( Generate Visit Slots )

- **Set Appointment Status:**

<b>Use case ID</b>	<b>VEMR-FR-AM-55</b>
<b>Use case name</b>	Set Appointment Status
<b>Description</b>	The system allows clinicians to automatically generate available visit slots based on their schedule.
<b>From</b>	Clinician
<b>Pre-conditions</b>	Clinician is authenticated
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. Clinician navigates to schedule management page</li> <li>2. Clinician clicks "Generate Slots" button</li> <li>3. System validates JWT token via DoctorJwtGuard</li> <li>4. System retrieves clinician schedules via ScheduleRepository</li> <li>5. System generates time slots based on schedule and slot duration</li> <li>6. System creates available slots via SlotRepository</li> <li>7. System displays success message with generated slots count</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Unauthorized Access</b></p> <ul style="list-style-type: none"> <li>- At step 3, if JWT token is invalid or expired</li> <li>- System redirects to login page</li> </ul> <p><b>A2: No Schedule Found</b></p> <ul style="list-style-type: none"> <li>- At step 4, if schedule ID does not exist</li> <li>- System displays "Please create schedule first" message</li> </ul>
<b>Post condition</b>	Available visit slots are generated

Table 62, Use Case Specification ( Set Appointment Status )

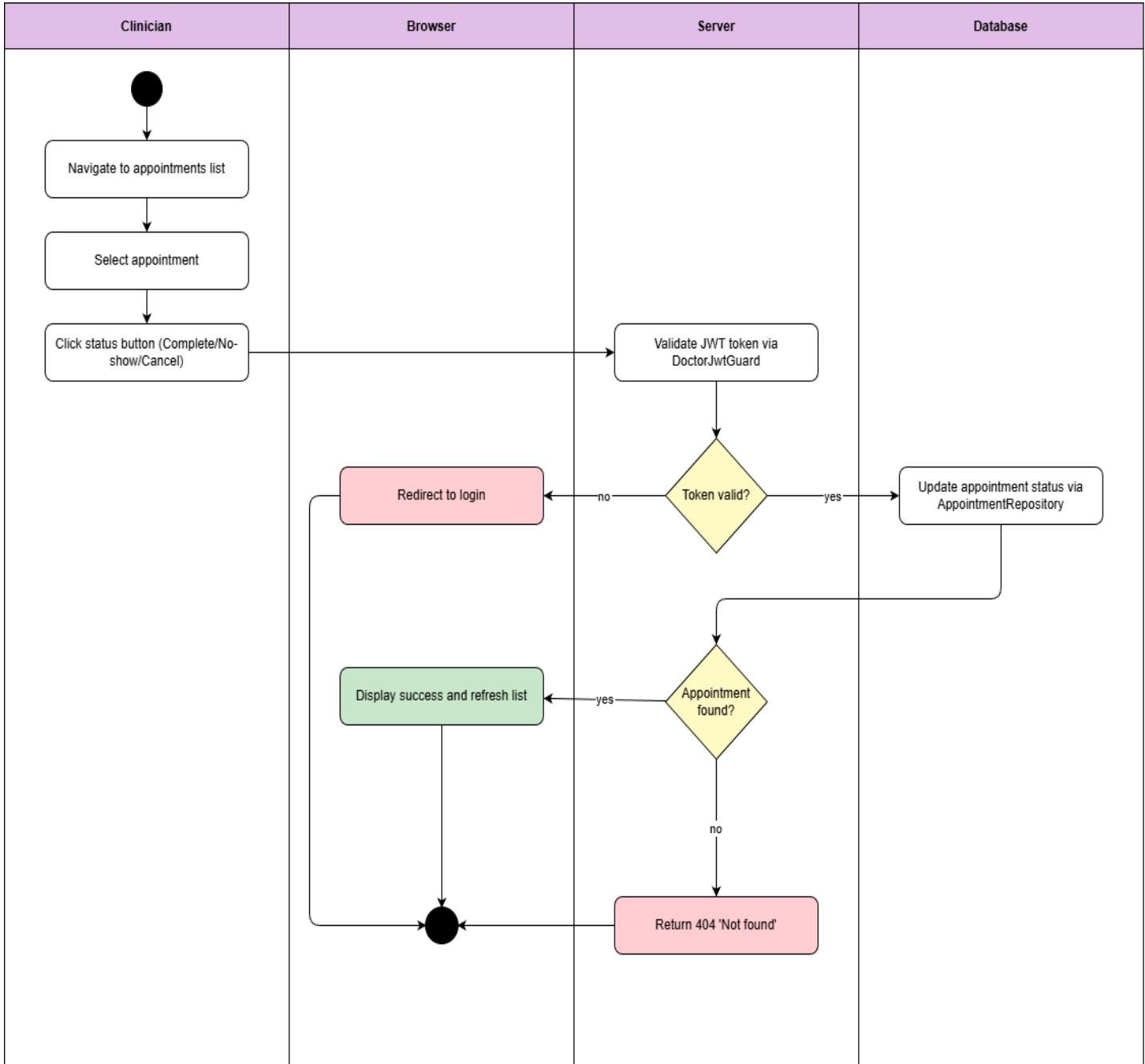


Diagram 112, Activity Diagram ( Set Appointment Status )

## Chapter 4 - System Analysis

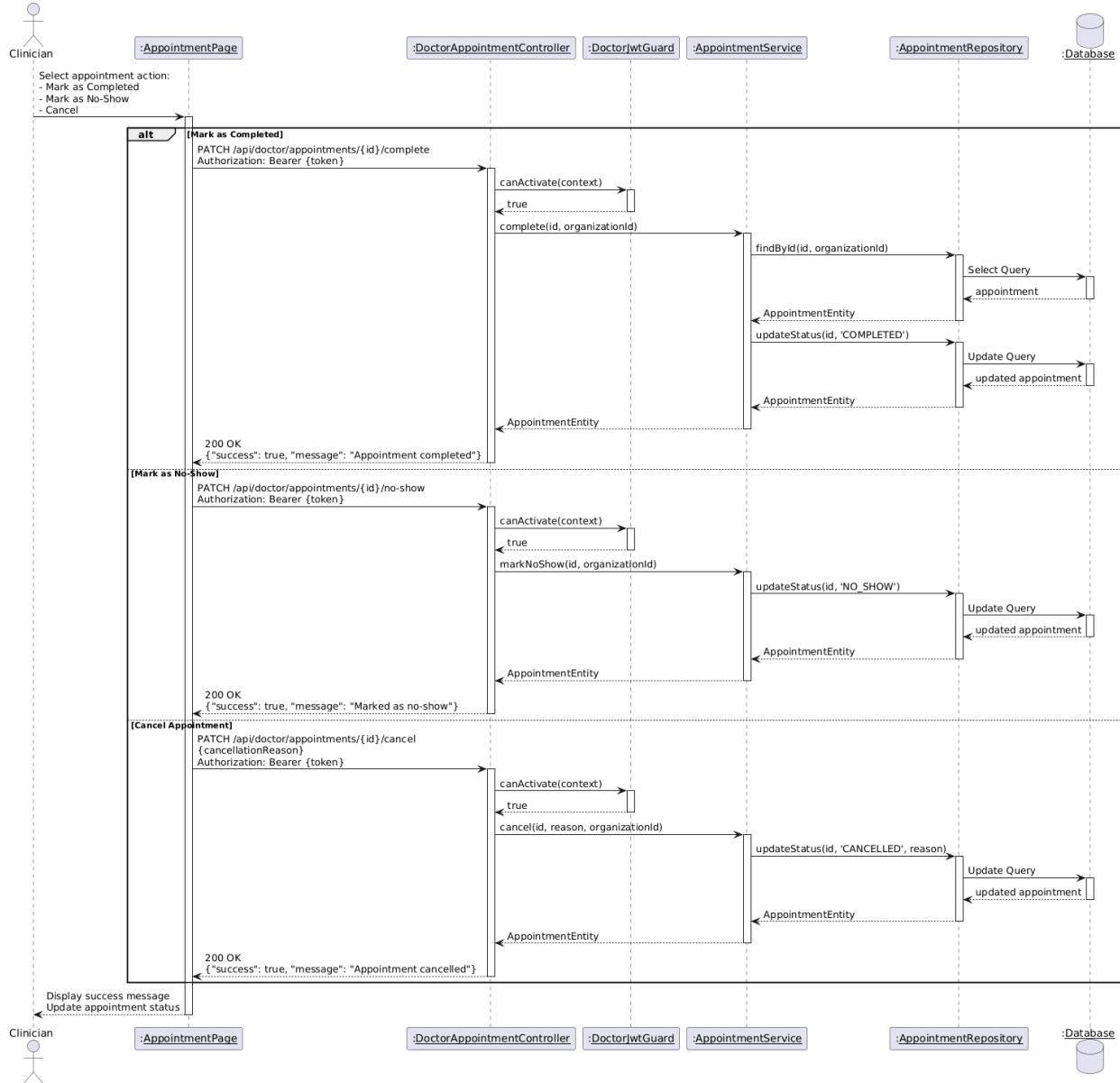


Diagram 113, Sequence Diagram ( Set Appointment Status )

- **Set Available Time Slots:**

<b>Use case ID</b>	<b>VEMR-FR-AM-56</b>
<b>Use case name</b>	Set Available Time Slots
<b>Description</b>	The system allows clinicians to define their available time slots for appointments.
<b>From</b>	Clinician
<b>Pre-conditions</b>	Clinician is authenticated
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. Clinician navigates to availability management page</li> <li>2. Clinician selects date and time range</li> <li>3. Clinician sets slot duration</li> <li>4. Clinician submits availability form</li> <li>5. System validates JWT token via DoctorJwtGuard</li> <li>6. System creates time slots via SlotRepository</li> <li>7. System displays success message with created slots</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Unauthorized Access</b></p> <ul style="list-style-type: none"> <li>- At step 5, if JWT token is invalid or expired</li> <li>- System redirects to login page</li> </ul> <p><b>A2: Validation Error</b></p> <ul style="list-style-type: none"> <li>- At step 4, if required fields are empty or invalid</li> <li>- System displays validation error messages</li> </ul> <p><b>A3: Overlapping Slots</b></p> <ul style="list-style-type: none"> <li>- At step 6, if slots overlap with existing slots</li> <li>- System displays error message</li> </ul>
<b>Post condition</b>	Available time slots are generated

Table 63, Use Case Specification ( Set Available Time Slots )

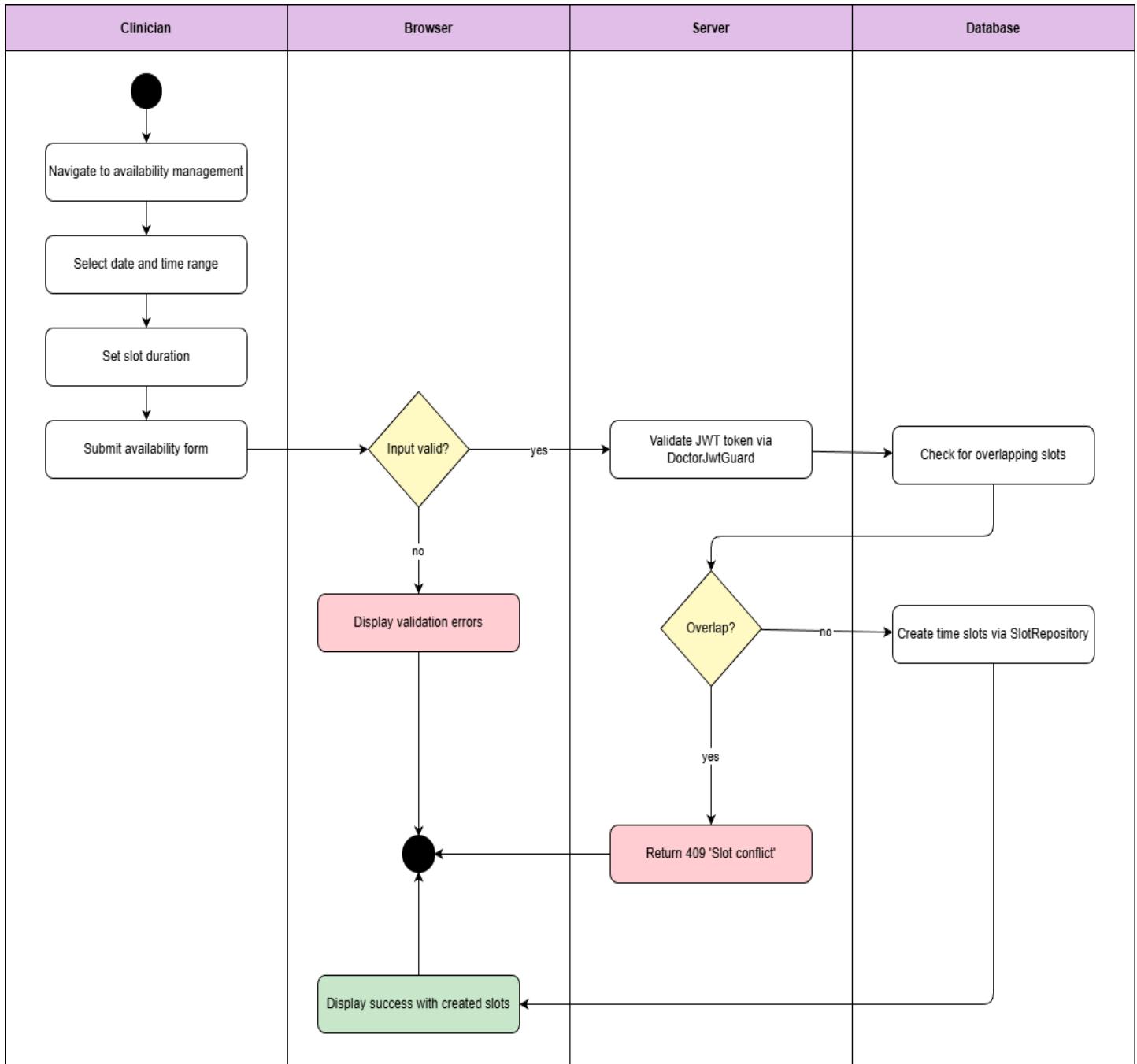


Diagram 114, Activity Diagram ( Set Available Time Slots )

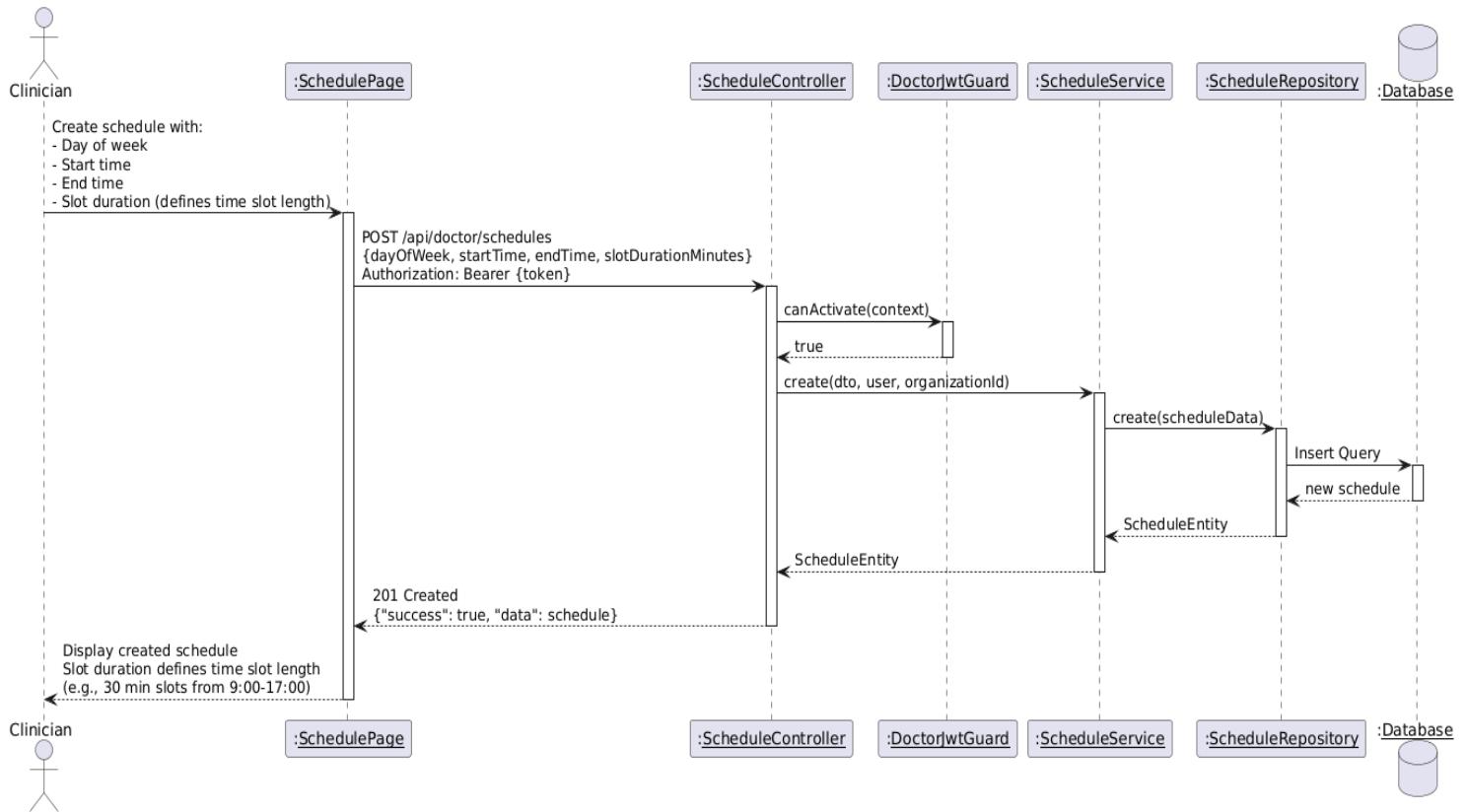


Diagram 115, Sequence Diagram ( Set Available Time Slots )

## ● View Organizations :

<b>Use case ID</b>	<b>VEMR-FR-AM-57</b>
<b>Use case name</b>	View Organizations
<b>Description</b>	The system allows patients to view available healthcare organizations
<b>From</b>	Patient
<b>Pre-conditions</b>	Patient is authenticated
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. Patient navigates to organizations page</li> <li>2. System validates JWT token via PatientJwtGuard</li> <li>3. System retrieves organizations list via OrganizationRepository</li> <li>4. System displays organizations with details</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Unauthorized Access</b></p> <ul style="list-style-type: none"> <li>- At step 2, if JWT token is invalid or expired</li> <li>- System redirects to login page</li> </ul> <p><b>A2: No Organizations Found</b></p> <ul style="list-style-type: none"> <li>- At step 3, if no organizations exist</li> <li>- System displays "No organizations available" message</li> </ul> <p><b>A3: Overlapping Slots</b></p> <ul style="list-style-type: none"> <li>- At step 6, if slots overlap with existing slots</li> <li>- System displays "No organizations available" message</li> </ul>
<b>Post condition</b>	Patient can view available healthcare organizations

Table 64, Use Case Specification ( View Organizations )

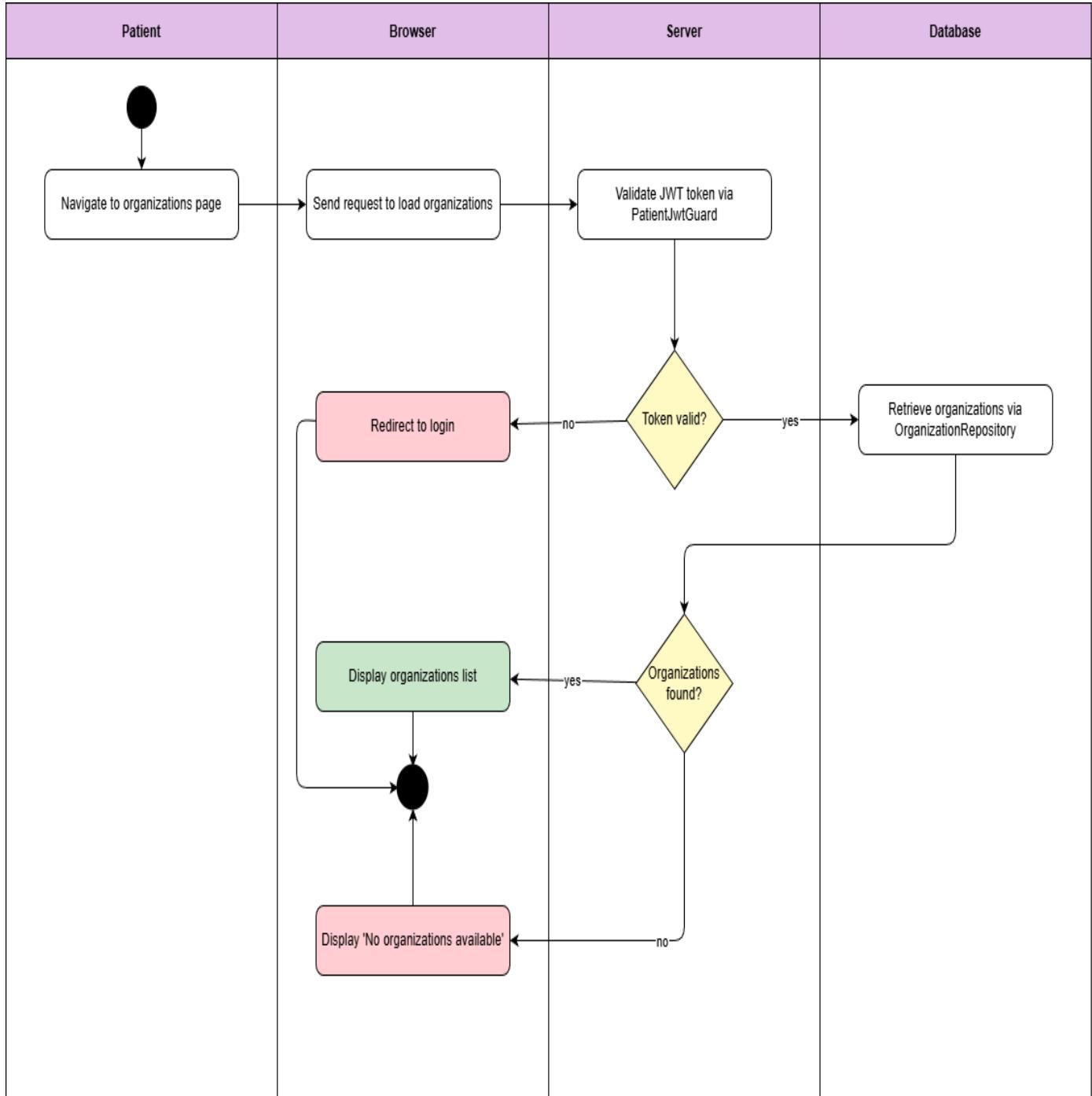


Diagram 116, Activity Diagram ( View Organizations )

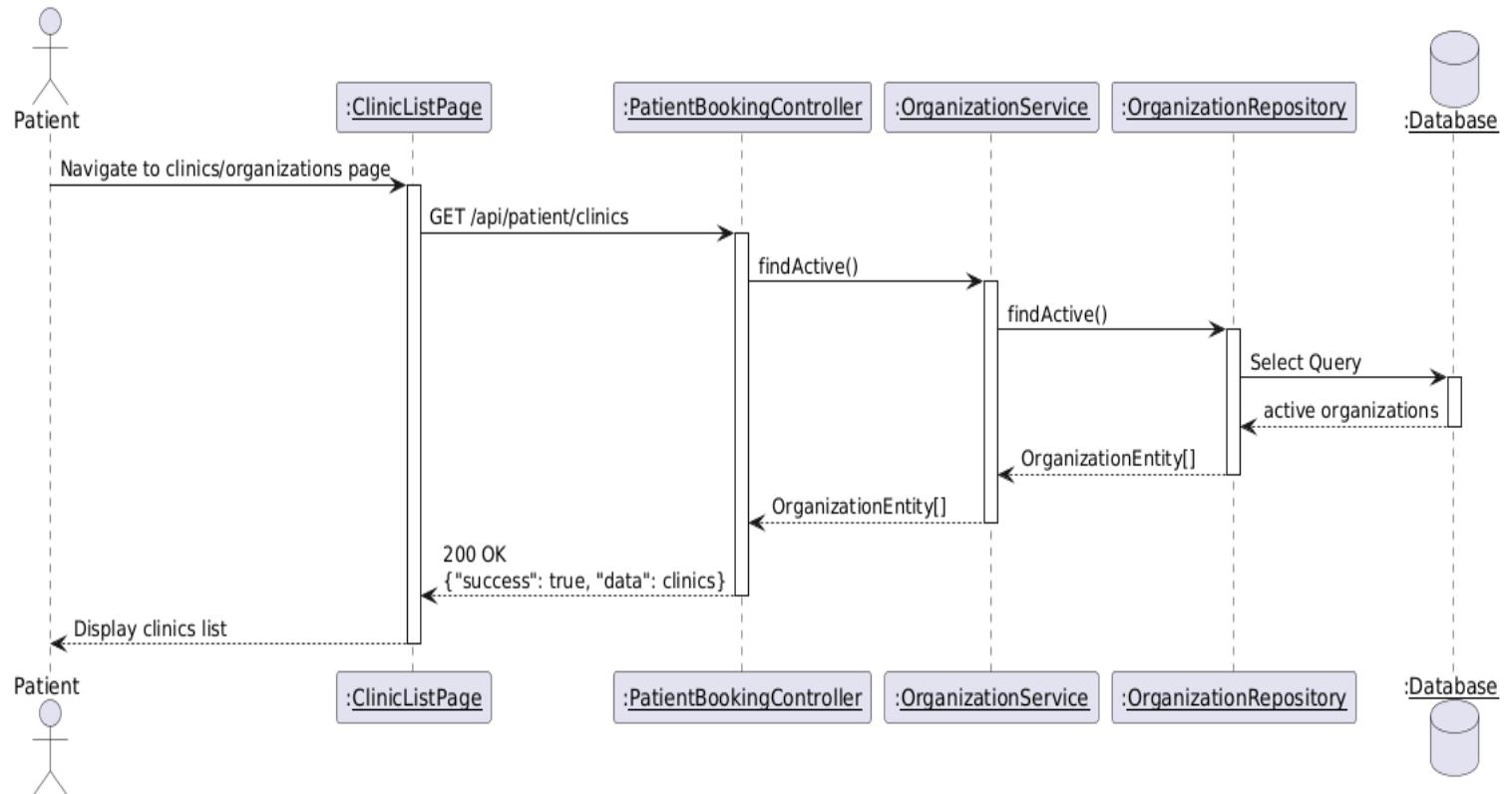


Diagram 117, Sequence Diagram ( View Organizations )

## ● View Doctor In Organizations :

<b>Use case ID</b>	<b>VEMR-FR-AM-58</b>
<b>Use case name</b>	View Doctors in Organizations
<b>Description</b>	The system allows patients to view doctors within specific organizations
<b>From</b>	Patient
<b>Pre-conditions</b>	Patient is authenticated
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. Patient navigates to organizations page</li> <li>2. System validates JWT token via PatientJwtGuard</li> <li>3. System retrieves organizations list via OrganizationRepository</li> <li>4. System displays organizations with details</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Unauthorized Access</b></p> <ul style="list-style-type: none"> <li>- At step 3 , if JWT token is invalid or expired</li> <li>-System redirects to login page</li> </ul> <p><b>A2: No Doctors Found</b></p> <ul style="list-style-type: none"> <li>- At step 4, if organization has no doctors</li> <li>- System displays "No doctors available in this organization" message</li> </ul>
<b>Post condition</b>	Patient can view available doctors

Table 65, Use Case Specification (View Doctors in Organizations)

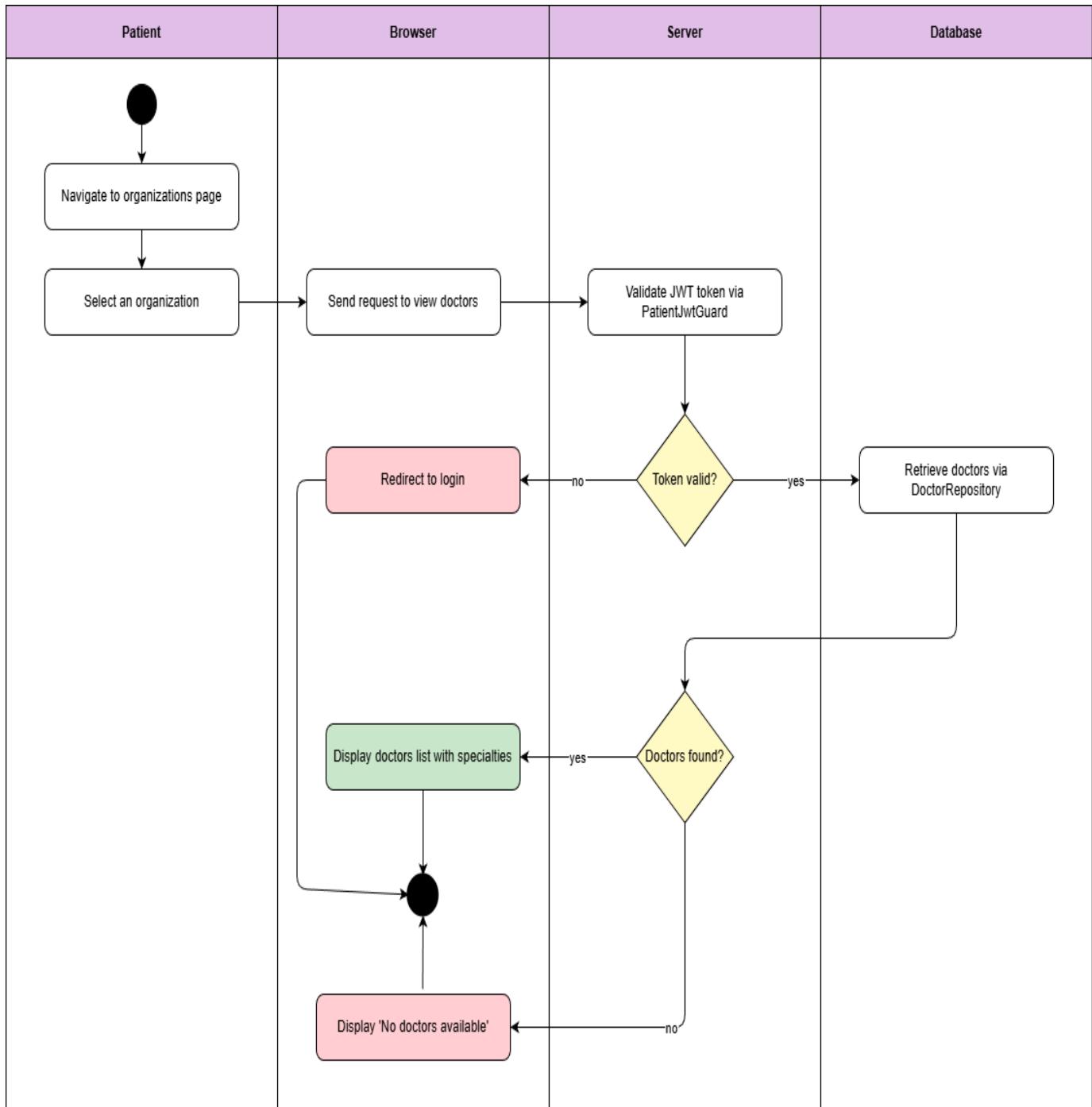


Diagram 117, Activity Diagram ( View Doctors in Organizations )

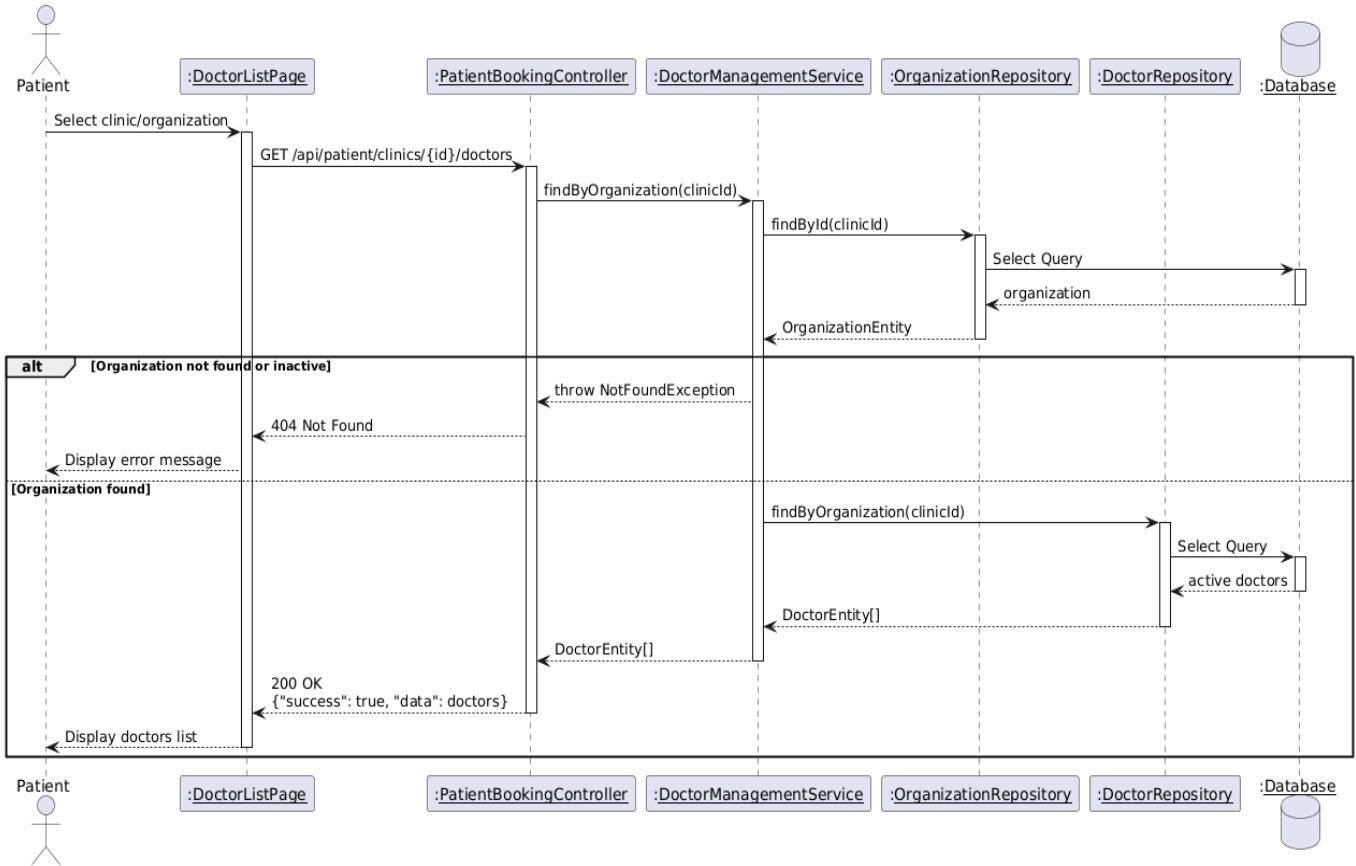


Diagram 118, Activity Diagram ( View Doctors in Organizations )

- **View Available Appointment:**

<b>Use case ID</b>	<b>VEMR-FR-AM-59</b>
<b>Use case name</b>	View Available Appointments
<b>Description</b>	The system allows patients to view available appointment slots
<b>From</b>	Patient
<b>Pre-conditions</b>	Patient is authenticated
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. Patient navigates to doctor profile</li> <li>2. Patient clicks "View Available Slots" button</li> <li>3. System validates JWT token via PatientJwtGuard</li> <li>4. System retrieves available slots via SlotRepository</li> <li>5. System displays available appointment slots by date and time</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Unauthorized Access</b></p> <ul style="list-style-type: none"> <li>- At step 3 , if JWT token is invalid or expired</li> <li>-System redirects to login page</li> </ul> <p><b>A2: No Slots Available</b></p> <ul style="list-style-type: none"> <li>- At step 4, if doctor has no available slots</li> <li>- System displays "No available slots" message</li> </ul>
<b>Post condition</b>	Available appointment slots are displayed

Table 66, Use Case Specification (View Available Appointment )

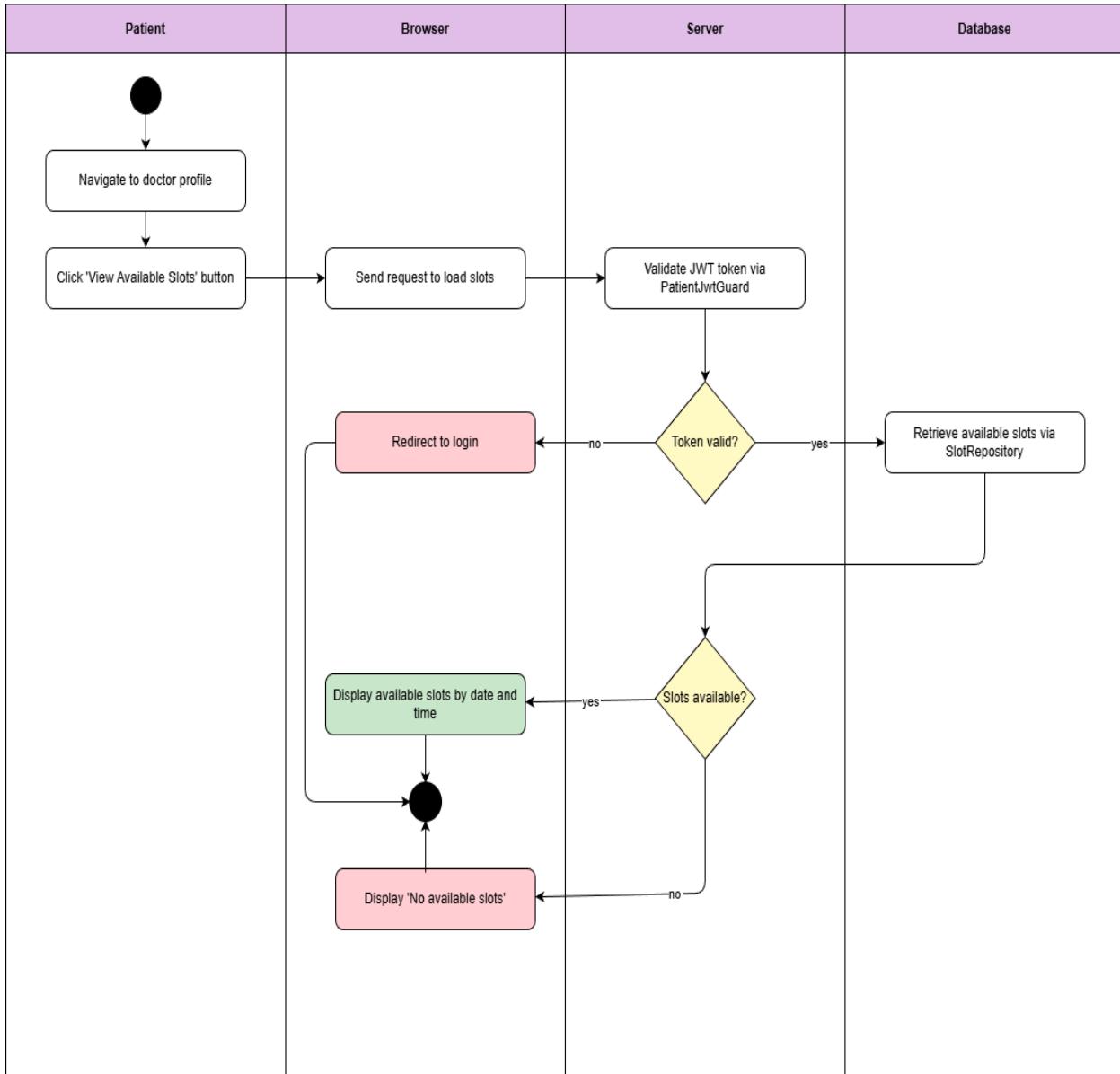


Diagram 119, Activity Diagram ( View Available Appointment )

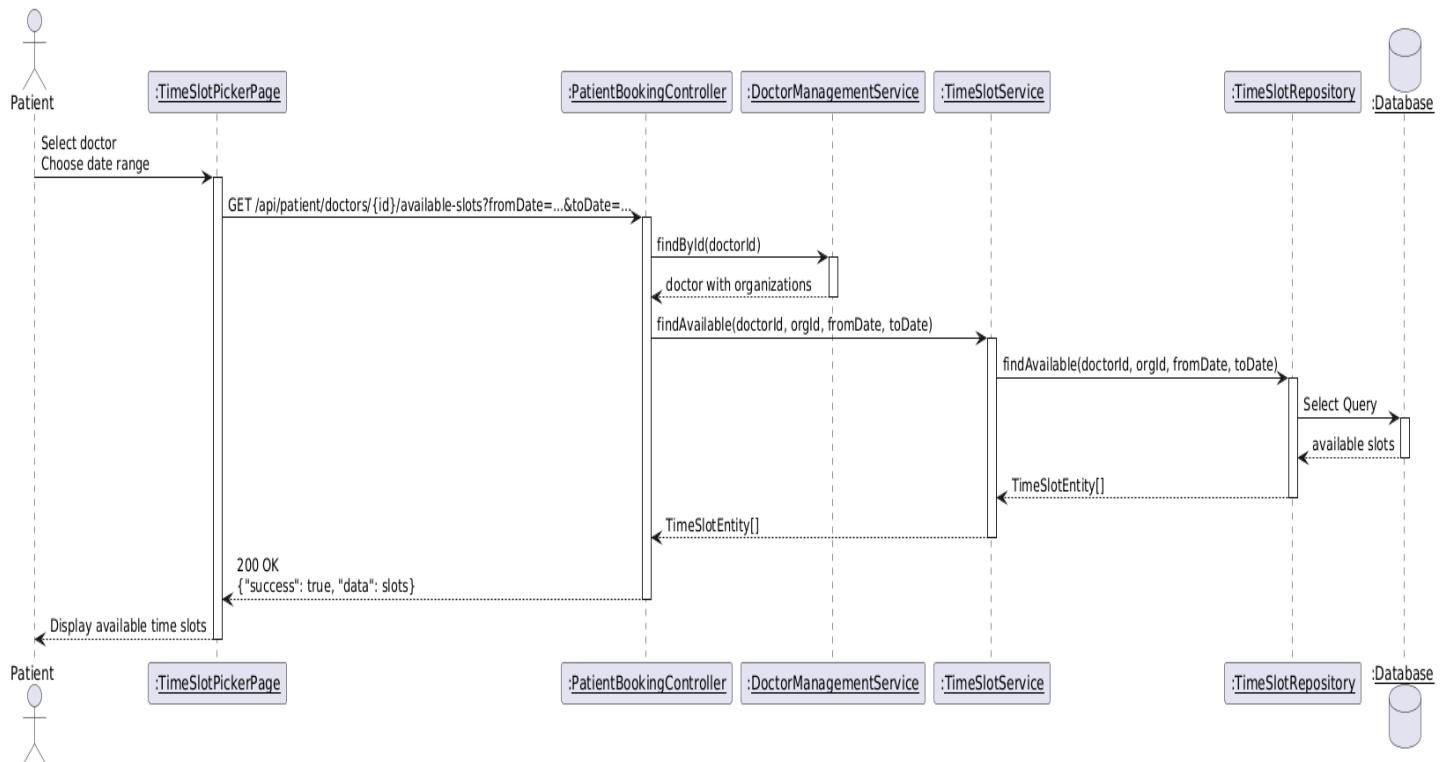


Diagram 120, Sequence Diagram ( View Available Appointment )

## ● Book an Appointment :

<b>Use case ID</b>	<b>VEMR-FR-AM-60</b>
<b>Use case name</b>	Book an Appointment
<b>Description</b>	The system allows patients to book an appointment with a clinician.
<b>From</b>	Patient
<b>Pre-conditions</b>	Patient is authenticated
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. Patient views available appointment slots</li> <li>2. Patient selects desired slot</li> <li>3. Patient enters appointment reason</li> <li>4. Patient confirms booking</li> <li>5. System validates JWT token via PatientJwtGuard</li> <li>6. System checks slot availability via SlotRepository</li> <li>7. System creates appointment via AppointmentRepository</li> <li>8. System marks slot as booked</li> <li>9. System displays success message with appointment details</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Unauthorized Access</b></p> <ul style="list-style-type: none"> <li>- At step 5 , if JWT token is invalid or expired</li> <li>-System redirects to login page</li> </ul> <p><b>A2: Slot No Longer Available</b></p> <ul style="list-style-type: none"> <li>- At step 6, if slot is already booked</li> <li>- System displays error message</li> </ul> <p><b>A3: Validation Error</b></p>

	<ul style="list-style-type: none"><li>- At step 3, if appointment reason is empty</li><li>- System displays validation error</li></ul>
<b>Post condition</b>	Slot is marked as booked

Table 67, Use Case Specification (Book an Appointment )

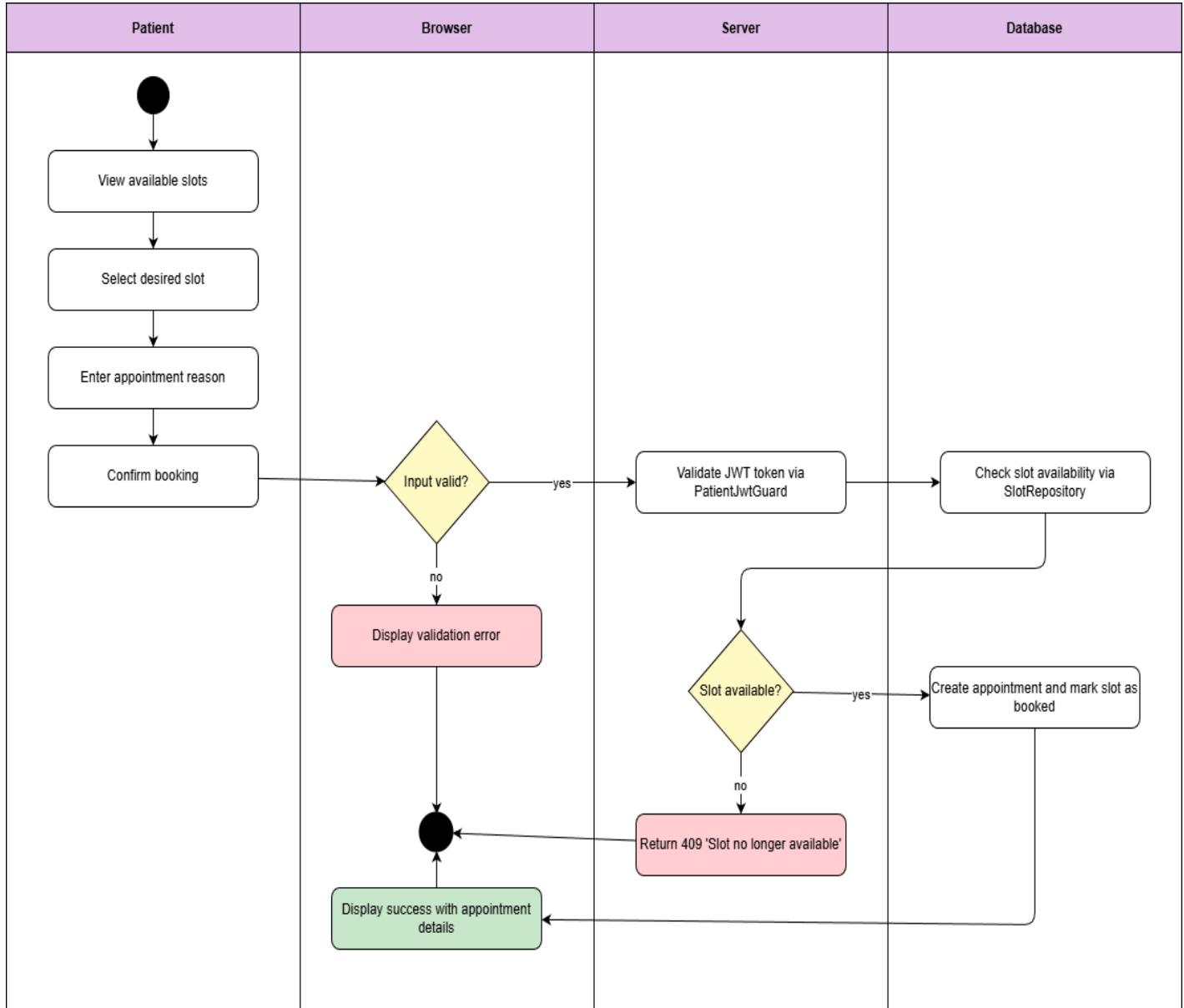


Diagram 121, Activity Diagram ( Book an Appointment )

## Chapter 4 - System Analysis

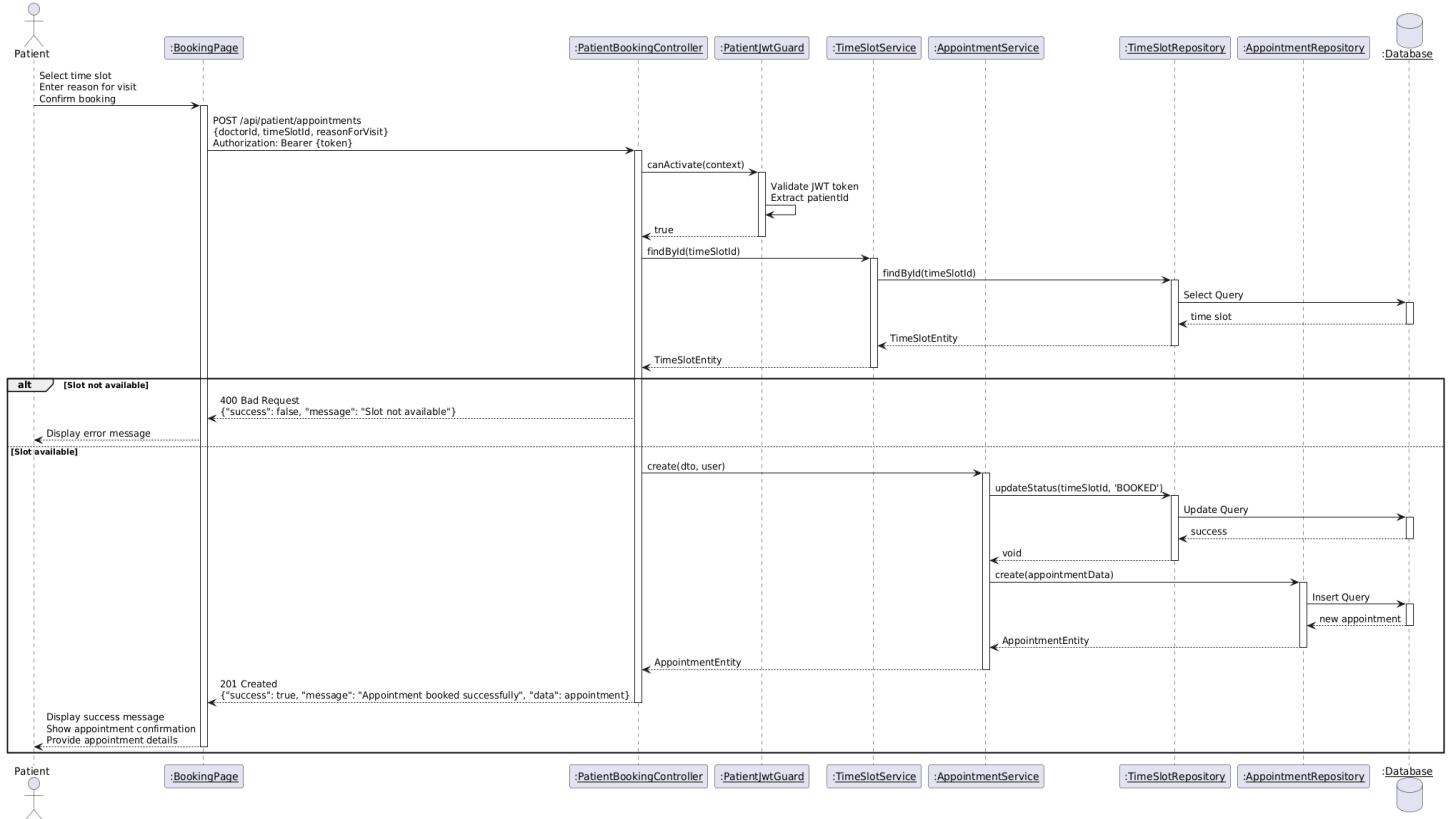


Diagram 122, Sequence Diagram ( Book an Appointment )

- **View Own Appointments :**

<b>Use case ID</b>	<b>VEMR-FR-AM-61</b>
<b>Use case name</b>	View Own Appointments
<b>Description</b>	The system allows patients to view their own appointment history and upcoming appointments.
<b>From</b>	Patient
<b>Pre-conditions</b>	Patient is authenticated
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. Patient navigates to my appointments page</li> <li>2. System validates JWT token via PatientJwtGuard</li> <li>3. System retrieves patient appointments via AppointmentRepository</li> <li>4. System displays appointments with doctor name, date, time, and status</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Unauthorized Access</b></p> <ul style="list-style-type: none"> <li>- At step 2 , if JWT token is invalid or expired</li> <li>-System redirects to login page</li> </ul> <p><b>A2: No Appointments Found</b></p> <ul style="list-style-type: none"> <li>- At step 3, if patient has no appointments</li> <li>- System returns empty list</li> </ul>
<b>Post condition</b>	Patient can view appointment history

Table 68, Use Case Specification (View Own Appointments )

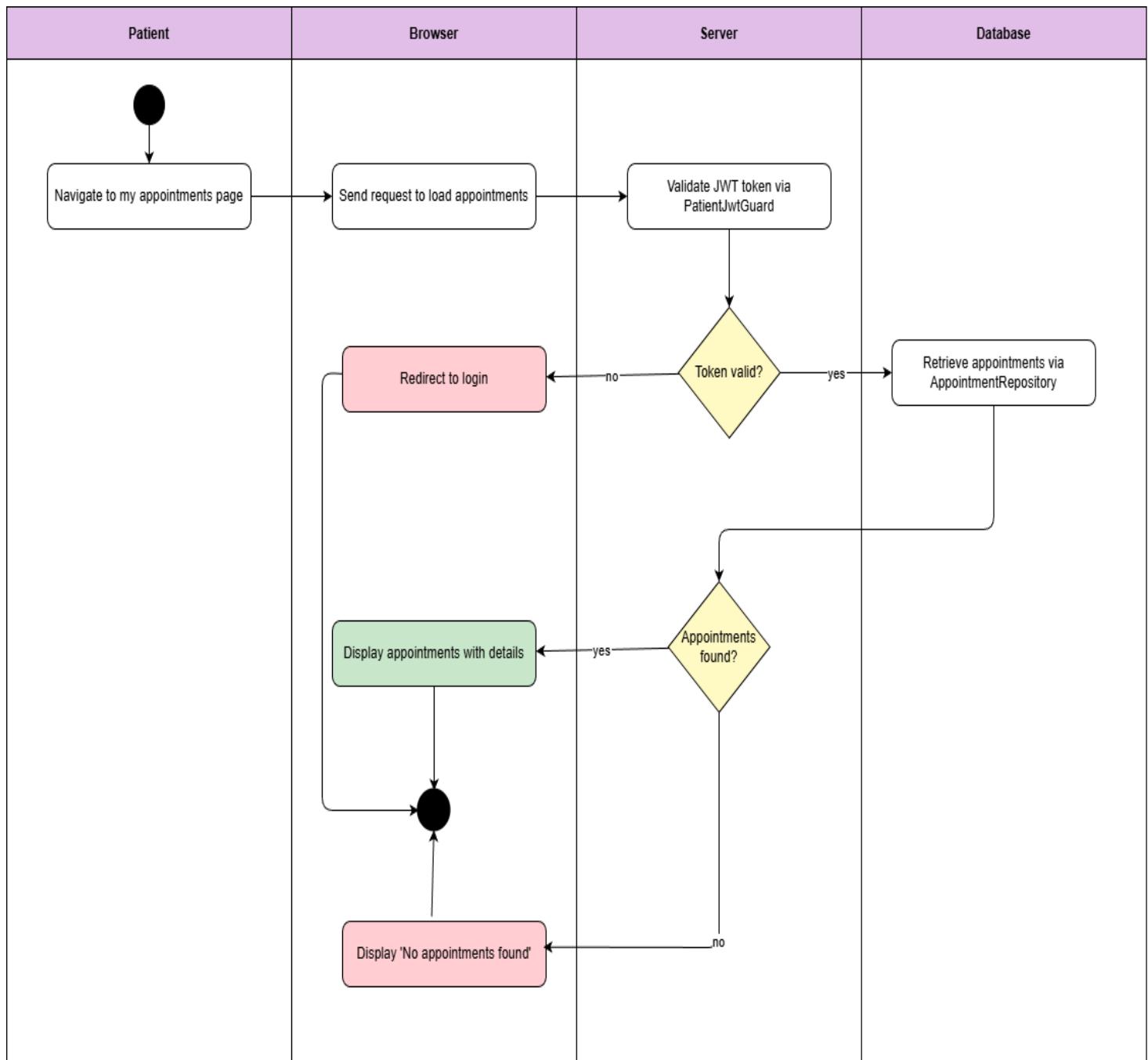


Diagram 123, Activity Diagram ( View Own Appointments )

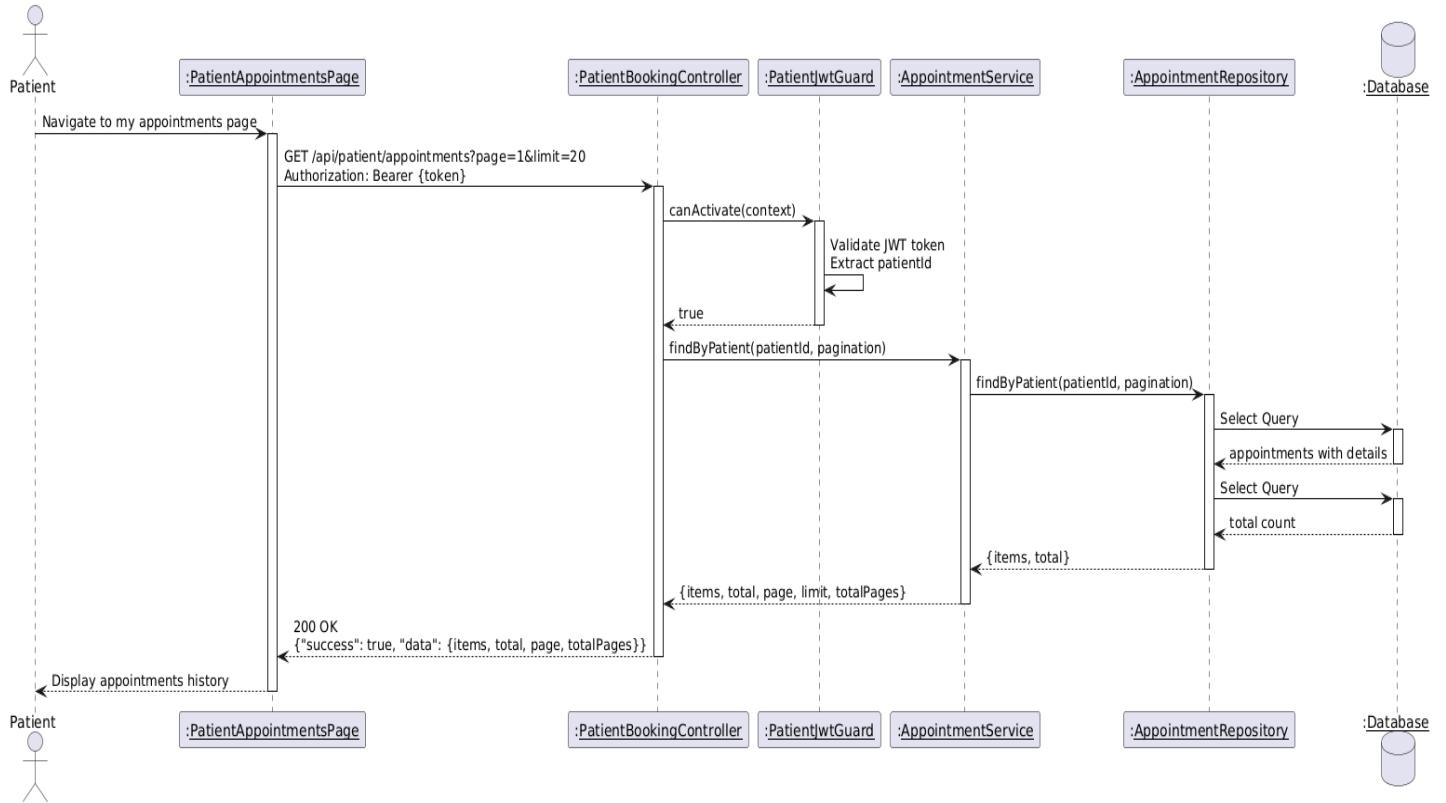


Diagram 124, Sequence Diagram ( View Own Appointments )

## ● Filter Appointment by Status :

<b>Use case ID</b>	<b>VEMR-FR-AM-62</b>
<b>Use case name</b>	Filter Appointment by Status
<b>Description</b>	The system allows patients to view their own appointment history and upcoming appointments.
<b>From</b>	Patient
<b>Pre-conditions</b>	Patient is authenticated
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. Patient navigates to my appointments page</li> <li>2. Patient selects status from filter dropdown</li> <li>3. System validates JWT token via PatientJwtGuard</li> <li>4. System retrieves appointments with selected status via AppointmentRepository</li> <li>5. System displays filtered appointments</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Unauthorized Access</b></p> <ul style="list-style-type: none"> <li>- At step 3 , if JWT token is invalid or expired</li> <li>-System redirects to login page</li> </ul> <p><b>A2: No Appointments with Selected Status</b></p> <ul style="list-style-type: none"> <li>- At step 4, if no appointments match the selected status</li> <li>- System displays "No appointments found with this status" message</li> </ul>
<b>Post condition</b>	Appointments are filtered by selected status

Table 69, Use Case Specification (Filter Appointment by Status )

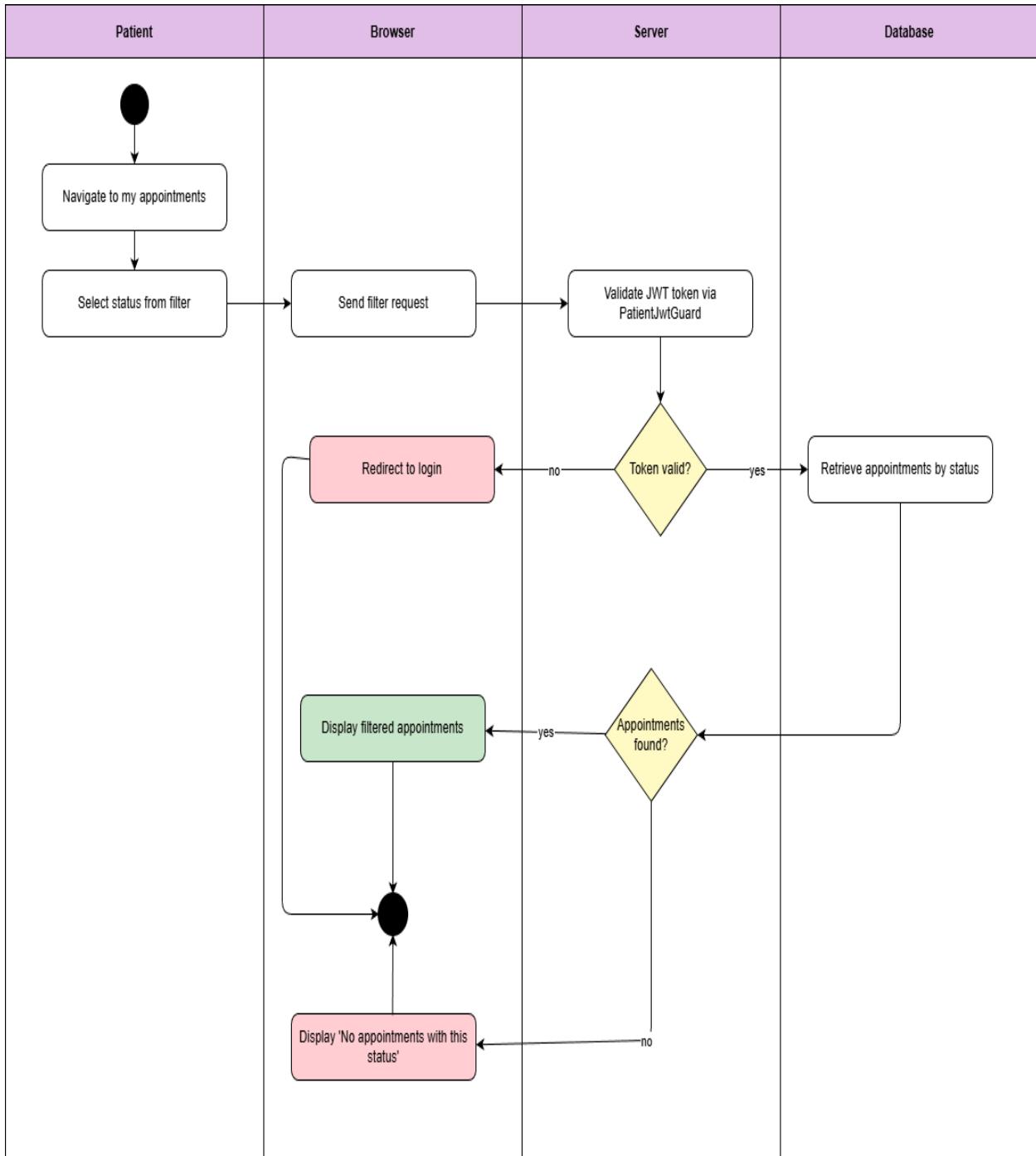


Diagram 125, Activity Diagram ( Filter Appointment by Status )

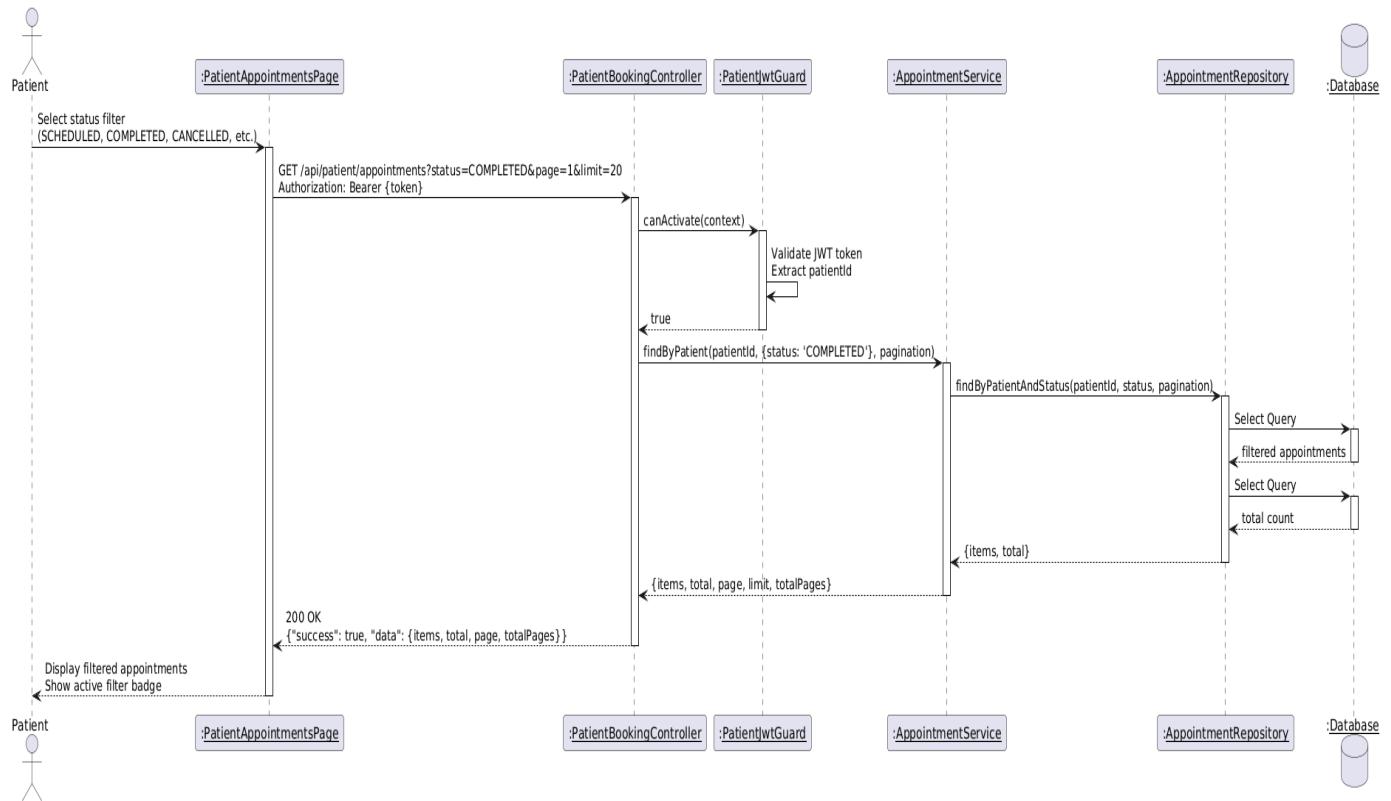


Diagram 126, Sequence Diagram ( Filter Appointment by Status )

- **View Own Medical Record :**

<b>Use case ID</b>	<b>VEMR-FR-AM-63</b>
<b>Use case name</b>	View Own Medical Record
<b>Description</b>	The system allows patients to view their own medical records.
<b>From</b>	Patient
<b>Pre-conditions</b>	Patient is authenticated
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. Patient navigates to medical records page</li> <li>2. System validates JWT token via PatientJwtGuard</li> <li>3. System retrieves patient medical records via EncounterRepository</li> <li>4. System displays medical records with diagnoses and treatments</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Unauthorized Access</b></p> <ul style="list-style-type: none"> <li>- At step 2 , if JWT token is invalid or expired</li> <li>-System redirects to login page</li> </ul> <p><b>A2: No Medical Records Found</b></p> <ul style="list-style-type: none"> <li>- At step 3, if patient has no medical records</li> <li>- System displays "No medical records found" message</li> </ul>
<b>Post condition</b>	Patient can view their health information

Table 70, Use Case Specification (View Own Medical Records )

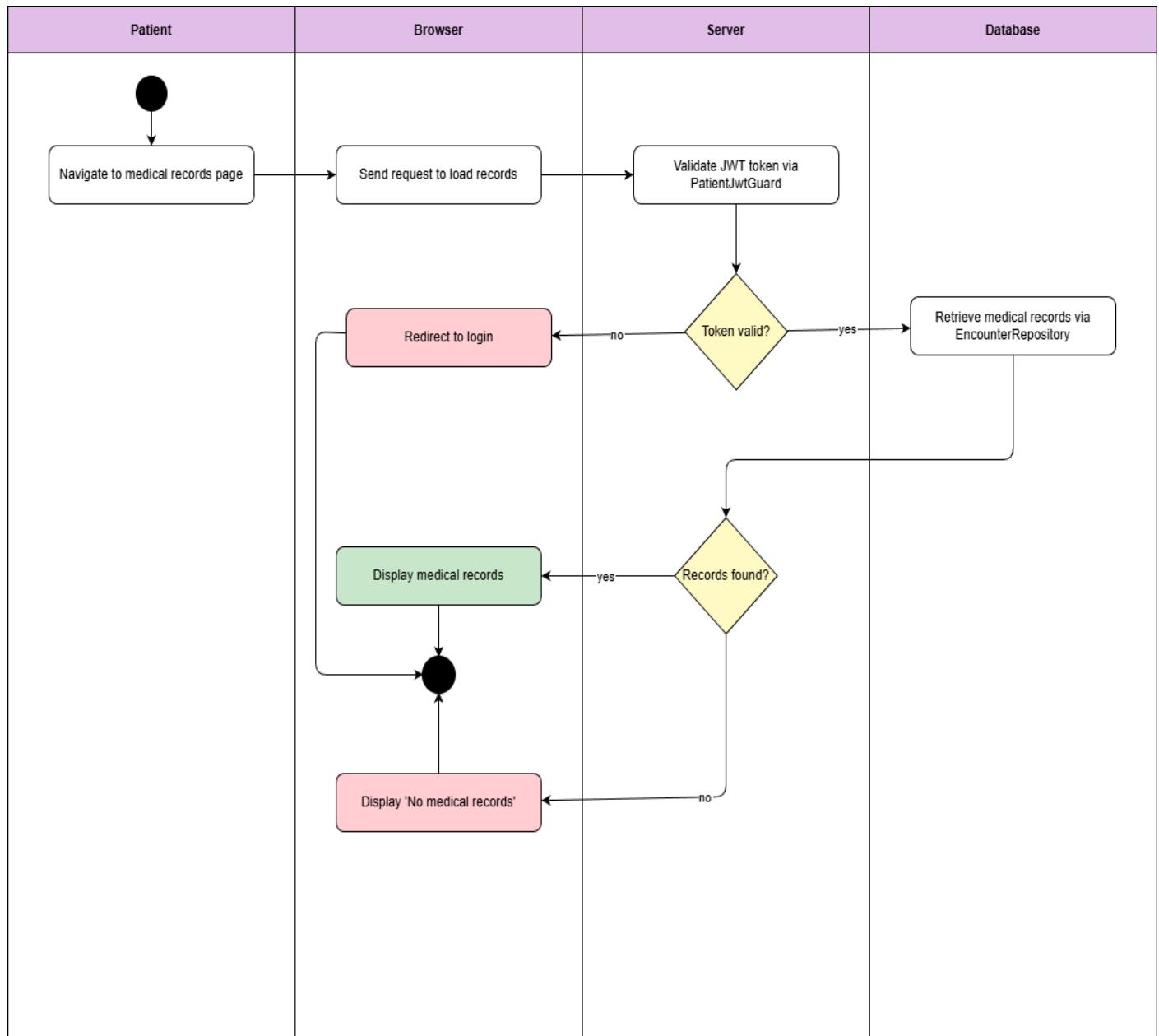


Diagram 127, Activity Diagram ( View Own Medical Records )

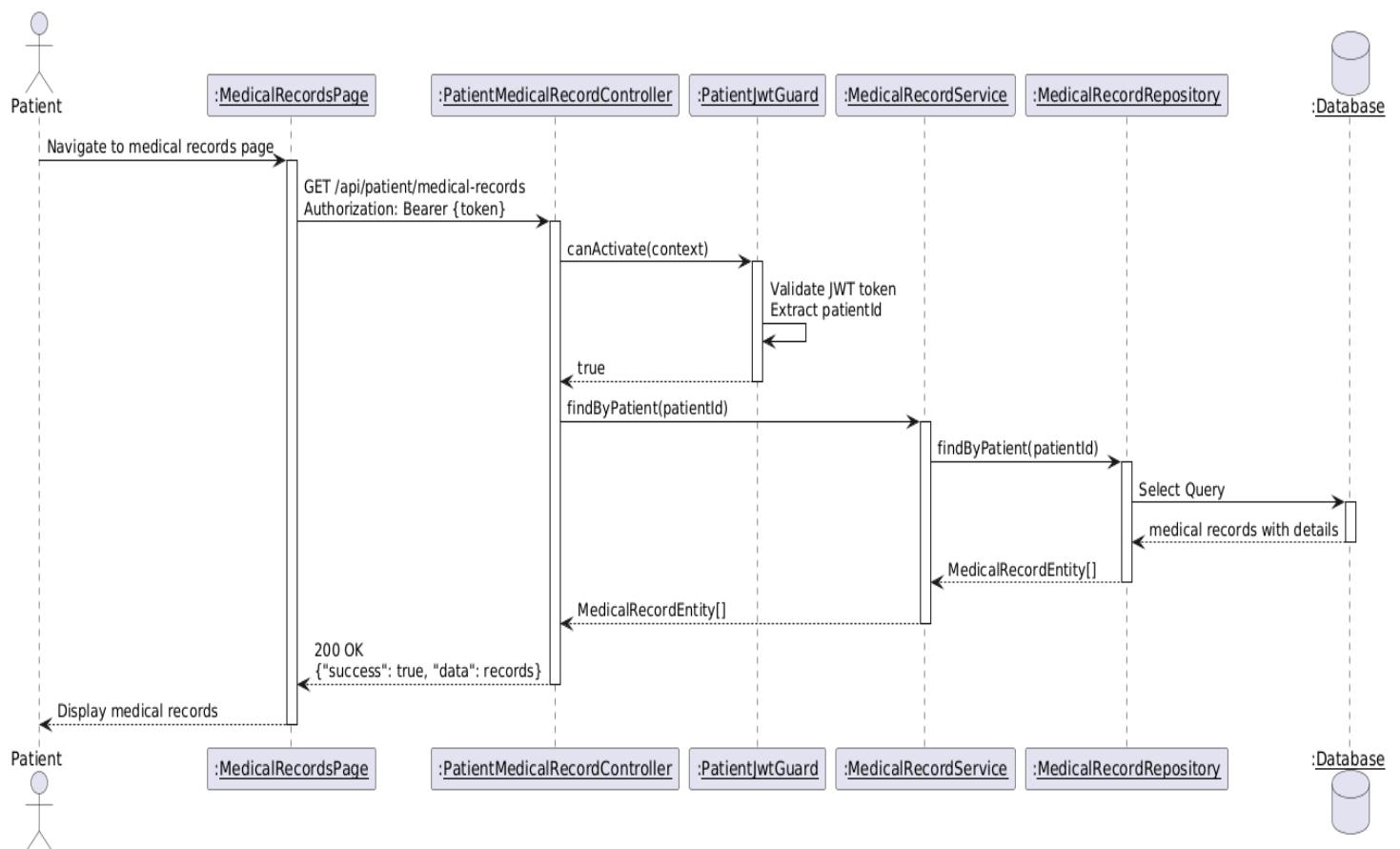


Diagram 128, Sequence Diagram ( View Own Medical Records )

- **View Own Allergies :**

<b>Use case ID</b>	<b>VEMR-FR-AM-64</b>
<b>Use case name</b>	View Own Allergies
<b>Description</b>	The system allows patients to view their own allergy information.
<b>From</b>	Patient
<b>Pre-conditions</b>	Patient is authenticated
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. Patient navigates to allergies page</li> <li>2. System validates JWT token via PatientJwtGuard</li> <li>3. System retrieves patient allergies via AllergyRepository</li> <li>4. System displays allergies with type, allergen, severity, and reaction</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Unauthorized Access</b></p> <ul style="list-style-type: none"> <li>- At step 2 , if JWT token is invalid or expired</li> <li>-System redirects to login page</li> </ul> <p><b>A2: No Allergies Found</b></p> <ul style="list-style-type: none"> <li>- At step 3, if patient has no recorded allergies</li> <li>- System displays "No allergies recorded" message</li> </ul>
<b>Post condition</b>	Patient can view their allergy information

Table 71, Use Case Specification ( View Own Allergies )

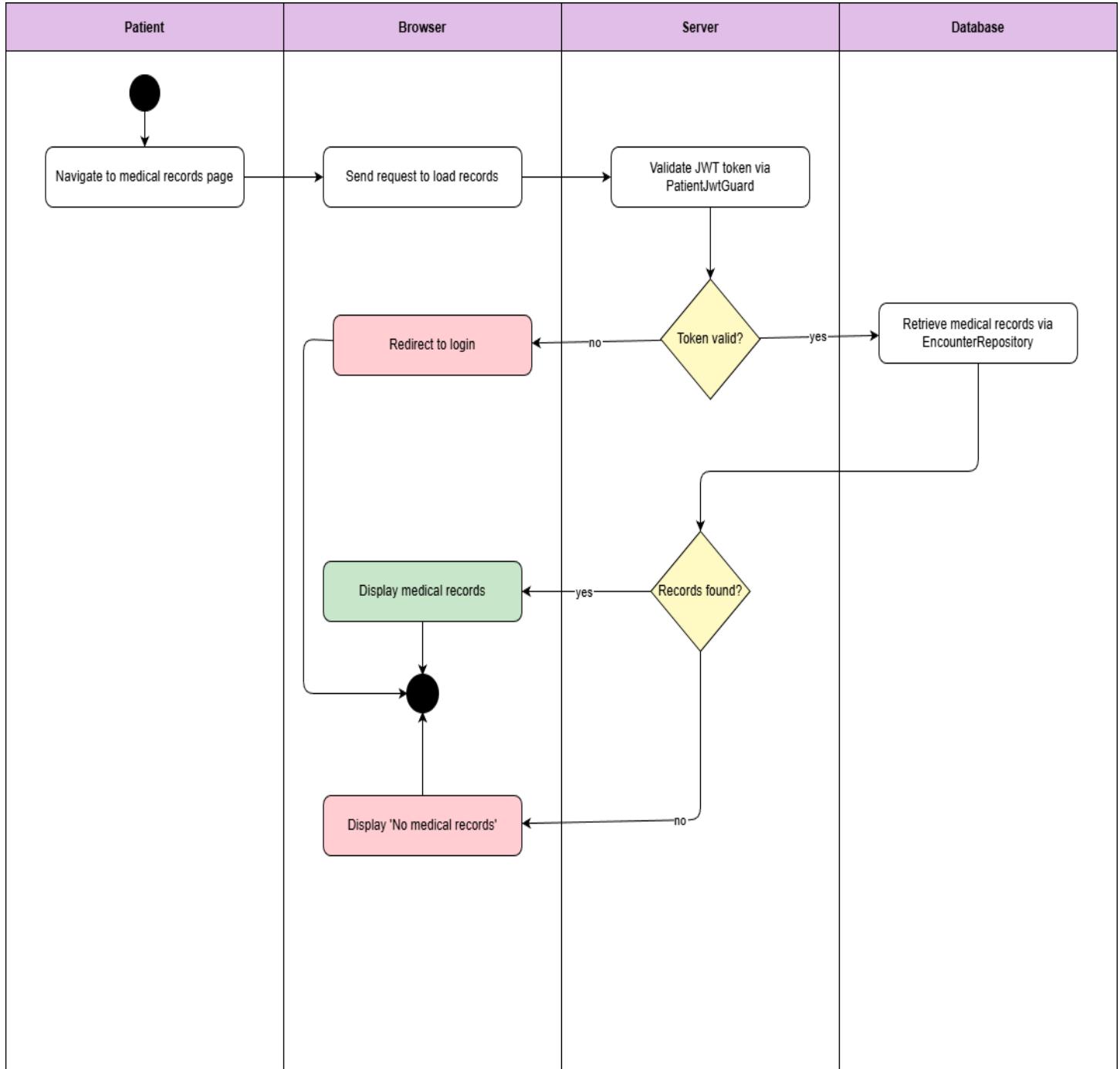


Diagram 129, Activity Diagram ( View Own Allergies )

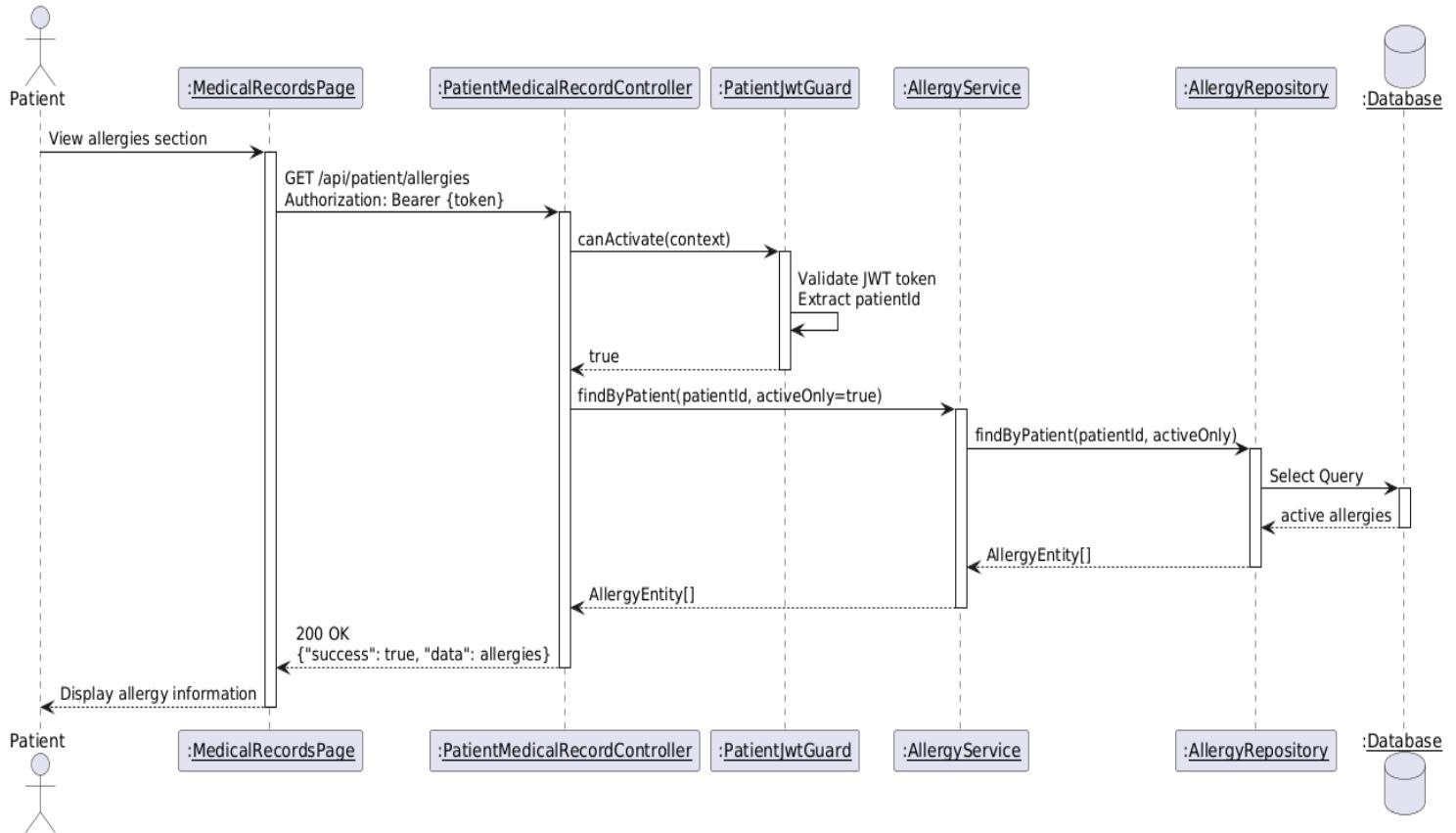


Diagram 130, Sequence Diagram ( View Own Allergies )

- **View Own Visits History :**

<b>Use case ID</b>	<b>VEMR-FR-AM-65</b>
<b>Use case name</b>	View Own Visits History
<b>Description</b>	The system allows patients to view their complete visit history.
<b>From</b>	Patient
<b>Pre-conditions</b>	Patient is authenticated
<b>Main scenario</b>	<ol style="list-style-type: none"> <li>1. Patient navigates to visits history page</li> <li>2. System validates JWT token via PatientJwtGuard</li> <li>3. System retrieves patient visits via EncounterRepository</li> <li>4. System displays visits with date, doctor, reason, and status</li> </ol>
<b>Alternative scenario</b>	<p><b>A1: Unauthorized Access</b></p> <ul style="list-style-type: none"> <li>- At step 2 , if JWT token is invalid or expired</li> <li>-System redirects to login page</li> </ul> <p><b>A2: No Visits Found</b></p> <ul style="list-style-type: none"> <li>- At step 3, if patient has no visit history</li> <li>- System displays "No visits found" message</li> </ul>
<b>Post condition</b>	Patient can view their complete medical visit history

Table 72, Use Case Specification ( View Own Visits History )

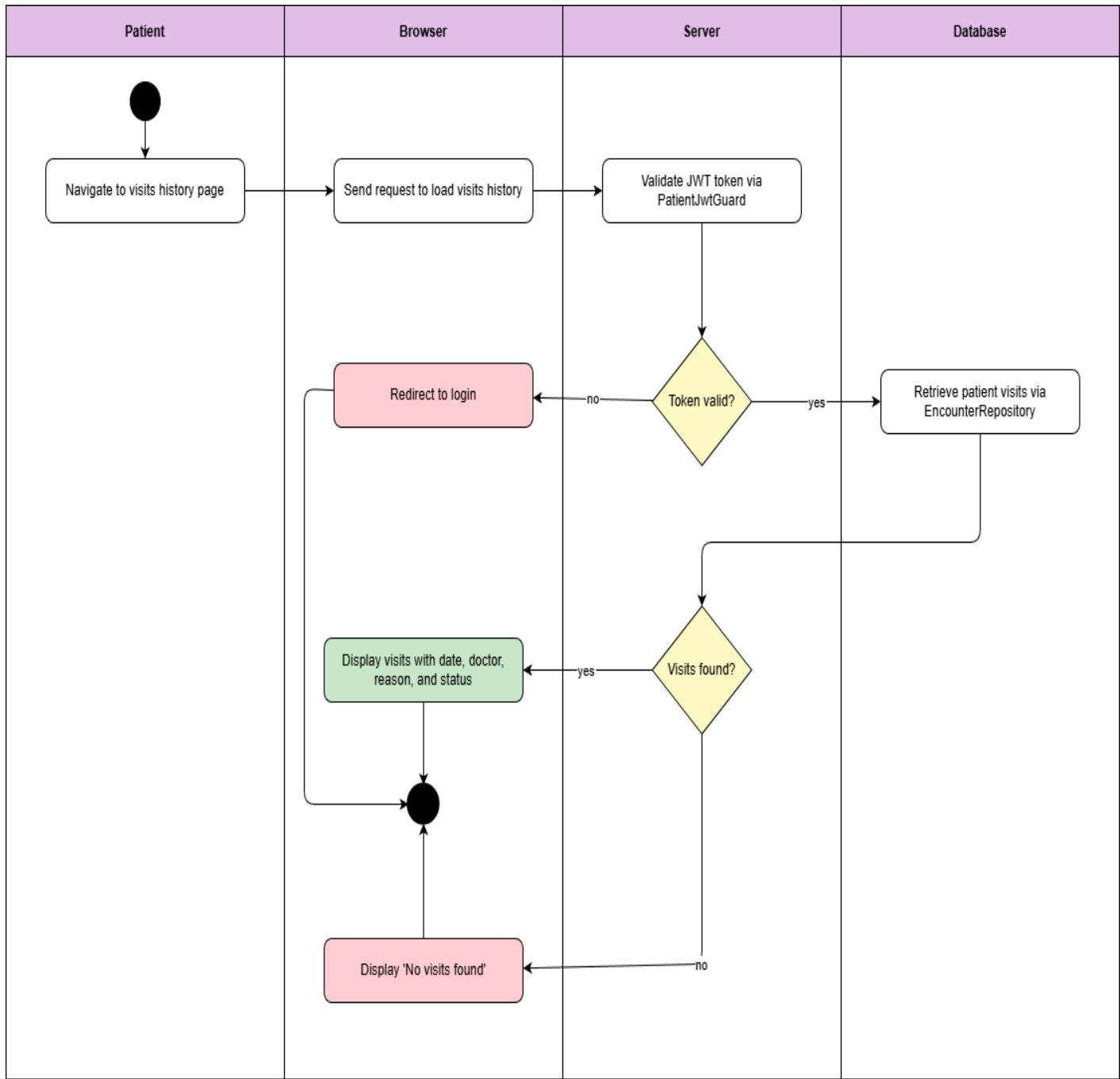


Diagram 131, Activity Diagram ( View Own Visits History )

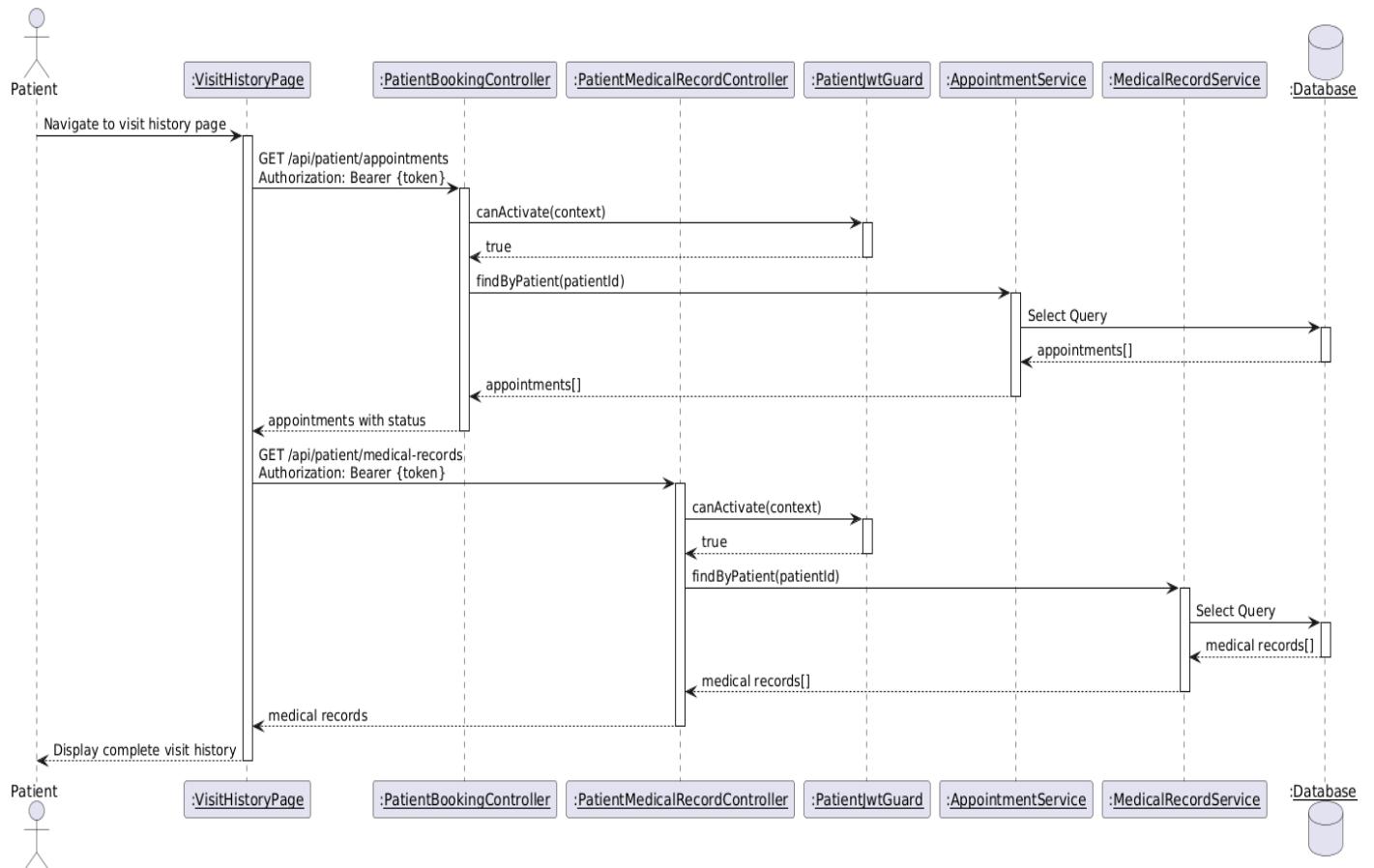


Diagram 132, Sequence Diagram ( View Own Visits History )

## 4.7 RTM V.2

ID	Title	Analysis Section	Design Section	Code	Integration Test	Unit Test
VEMR-FR-PM-01	The system should allow patients to create a new account by providing email, password, first name, and last name.	<a href="#"><u>Specification and Diagrams</u></a>				
VEMR-FR-PM-02	The system should allow patients to login to the system using their email and password credentials.	<a href="#"><u>Specification and Diagrams</u></a>				
VEMR-FR-PM-03	The system should allow patients to verify their email address before accessing the system.	<a href="#"><u>Specification and Diagrams</u></a>				
VEMR-FR-PM-04	The system should allow users to request a password reset link via email and set a new password.	<a href="#"><u>Specification and Diagrams</u></a>				
VEMR-FR-PM-05	The system should allow users to view their profile information including personal details.	<a href="#"><u>Specification and Diagrams</u></a>				

VEMR-FR-PM-06	The system should allow users to update their profile information such as name, phone number, and other details.	<a href="#"><u>Specification and Diagrams</u></a>				
VEMR-FR-PM-07	The system should allow users to change their current password to a new one.	<a href="#"><u>Specification and Diagrams</u></a>				
VEMR-FR-AU-08	The system should allow administrators to login to the system using their credentials.	<a href="#"><u>Specification and Diagrams</u></a>				
VEMR-FR-AU-09	The system should allow clinicians/doctors to login to the system using their email and password.	<a href="#"><u>Specification and Diagrams</u></a>				
VEMR-FR-VM-10	The system should allow clinicians to create a new patient visit with date, type, reason, and chief complaint.	<a href="#"><u>Specification and Diagrams</u></a>				

VEMR-FR-VM-11	The system should allow clinicians to search visits by patient name or reason for visit.	<a href="#">Specification and Diagrams</a>				
VEMR-FR-VM-12	The system should allow clinicians to edit visit details when the visit is in progress.	<a href="#">Specification and Diagrams</a>				
VEMR-FR-VM-13	The system should allow clinicians to save changes made to a visit including medical record data.	<a href="#">Specification and Diagrams</a>				
VEMR-FR-VM-14	The system should allow clinicians to delete a visit from the system.	<a href="#">Specification and Diagrams</a>				
VEMR-FR-VM-15	The system should allow clinicians to view complete details of a visit including medical record, vitals, and allergies.	<a href="#">Specification and Diagrams</a>				
VEMR-FR-VM-16	The system should allow voice transcription to appear in real-time as the	<a href="#">Specification and Diagrams</a>				

	clinician speaks using Whisper.					
VEMR-FR-VM-17	The system should allow clinicians to view a paginated list of all patients in the system.	<a href="#">Specification and Diagrams</a>				
VEMR-FR-VM-18	The system allows clinicians to search for patient.	<a href="#">Specification and Diagrams</a>				
VEMR-FR-VM-19	The system should allow clinicians to view a patient's profile with their visits and allergies.	<a href="#">Specification and Diagrams</a>				
VEMR-FR-VM-20	The system should allow clinicians to view detailed patient information including contact and demographic data.	<a href="#">Specification and Diagrams</a>				
VEMR-FR-VM-21	The system should allow clinicians to change visit status: start (planned to in-progress), complete, or cancel.	<a href="#">Specification and Diagrams</a>				

VEMR-FR-VM-22	The system should allow clinicians to view a paginated list of all visits with filtering options.	<a href="#"><u>Specification and Diagrams</u></a>				
VEMR-FR-VS-23	The system should allow clinicians to view vital signs recorded for a specific visit.	<a href="#"><u>Specification and Diagrams</u></a>				
VEMR-FR-VS-24	The system should allow clinicians to record and edit vital signs (BP, HR, temp, etc.) during an active visit.	<a href="#"><u>Specification and Diagrams</u></a>				
VEMR-FR-AM-25	The system should allow clinicians to view a patient's recorded allergies with severity and reaction details.	<a href="#"><u>Specification and Diagrams</u></a>				
VEMR-FR-AM-26	The system should allow clinicians to add a new allergy record for a patient with type, allergen, severity, and reaction.	<a href="#"><u>Specification and Diagrams</u></a>				

VEMR-FR-AM-27	The system should allow clinicians to update existing allergy information for a patient.	<a href="#"><u>Specification and Diagrams</u></a>				
VEMR-FR-AM-28	The system should allow clinicians to delete an allergy record from a patient's profile.	<a href="#"><u>Specification and Diagrams</u></a>				
VEMR-FR-CM-29	The system should allow the admin to create clinicians' accounts.	<a href="#"><u>Specification and Diagrams</u></a>				
VEMR-FR-CM-30	The system should allow the clinicians and admins to update their account.	<a href="#"><u>Specification and Diagrams</u></a>				
VEMR-FR-CM-31	The system should allow the admin to view the clinicians accounts list.	<a href="#"><u>Specification and Diagrams</u></a>				
VEMR-FR-CM-32	The system should allow the admin to view the clinicians account details.	<a href="#"><u>Specification and Diagrams</u></a>				

VEMR-FR-CM-33	The system should allow the admin to delete the clinician's accounts.	<a href="#"><u>Specification and Diagrams</u></a>				
VEMR-FR-CM-34	The system should allow the admin to search the clinician's accounts.	<a href="#"><u>Specification and Diagrams</u></a>				
VEMR-FR-OM-35	The system should allow the admin to create organizations.	<a href="#"><u>Specification and Diagrams</u></a>				
VEMR-FR-OM-36	The system should allow the admin to update organization information.	<a href="#"><u>Specification and Diagrams</u></a>				
VEMR-FR-OM-37	The system should allow the admin to view all organizations.	<a href="#"><u>Specification and Diagrams</u></a>				
VEMR-FR-OM-38	The system should allow the admin to view detailed information about a specific organization.	<a href="#"><u>Specification and Diagrams</u></a>				

VEMR-FR-OM-39	The system should allow the admin to delete an organization.	<a href="#">Specification and Diagrams</a>				
VEMR-FR-OM-40	The system should allow the admin to search for organizations.	<a href="#">Specification and Diagrams</a>				
VEMR-FR-AN-41	The system should allow administrators to view system-wide analytics and reports.	<a href="#">Specification and Diagrams</a>				
VEMR-FR-AP-42	The system should allow clinicians to view their appointment schedule.	<a href="#">Specification and Diagrams</a>				
VEMR-FR-AP-43	The system should allow clinicians to view their daily appointment schedule.	<a href="#">Specification and Diagrams</a>				
VEMR-FR-AP-44	The system should allow clinicians to filter appointments by specific date.	<a href="#">Specification and Diagrams</a>				
VEMR-FR-AP-45	The system should allow clinicians to	<a href="#">Specification and Diagrams</a>				

	filter appointments by status.					
VEMR-FR-AP-46	The system should allow the doctor to check in a patient's appointment.	<a href="#">Specification and Diagrams</a>				
VEMR-FR-AP-47	The system should allow the doctor to cancel a patient's appointment.	<a href="#">Specification and Diagrams</a>				
VEMR-FR-AP-48	The system should allow the doctor to set an appointment as no show.	<a href="#">Specification and Diagrams</a>				
VEMR-FR-DA-49	The system should allow clinicians to view their personal analytics and performance metrics.	<a href="#">Specification and Diagrams</a>				
VEMR-FR-SM-50	The system should allow clinicians to create their work schedule.	<a href="#">Specification and Diagrams</a>				
VEMR-FR-SM-51	The system should allow clinicians to edit	<a href="#">Specification and Diagrams</a>				

	their existing work schedule.					
VEMR-FR-SM-52	The system should allow clinicians to delete their work schedule.	<a href="#">Specification and Diagrams</a>				
VEMR-FR-SM-53	The system should allow clinicians to automatically generate available visit slots based on their schedule.	<a href="#">Specification and Diagrams</a>				
VEMR-FR-SM-54	The system should allow clinicians to mark appointments as completed, no-show, or cancelled.	<a href="#">Specification and Diagrams</a>				
VEMR-FR-SM-55	The system should allow clinicians to view detailed information about a specific appointment.	<a href="#">Specification and Diagrams</a>				
VEMR-FR-SM-56	The system should allow clinicians to define their available time slots for appointments.	<a href="#">Specification and Diagrams</a>				

VEMR-FR-PP-57	The system should allow patients to view available healthcare organizations.	<a href="#"><u>Specification and Diagrams</u></a>				
VEMR-FR-PP-58	The system should allow patients to view doctors within specific organizations.	<a href="#"><u>Specification and Diagrams</u></a>				
VEMR-FR-PP-59	The system should allow patients to view available appointment slots.	<a href="#"><u>Specification and Diagrams</u></a>				
VEMR-FR-PP-60	The system should allow patients to book an appointment with a clinician.	<a href="#"><u>Specification and Diagrams</u></a>				
VEMR-FR-PP-61	The system should allow patients to view their own appointment history and upcoming appointments.	<a href="#"><u>Specification and Diagrams</u></a>				
VEMR-FR-PP-62	The system should allow patients to filter their appointments by status.	<a href="#"><u>Specification and Diagrams</u></a>				

VEMR-FR-PP-63	The system should allow patients to view their own medical records.	<a href="#"><u>Specification and Diagrams</u></a>				
VEMR-FR-PP-64	The system should allow patients to view their own allergy information.	<a href="#"><u>Specification and Diagrams</u></a>				
VEMR-FR-PP-65	The system should allow patients to view their complete visit history.	<a href="#"><u>Specification and Diagrams</u></a>				

Table 73 - RTM V.2

## **Chapter 5 - System Design**

## 5.1 Introduction:

The system employs a layered architecture combined with a client-server architecture pattern. This approach provides clear separation of concerns, enhances maintainability, and enables scalable development. The architecture is designed to ensure efficient data flow, interaction, and integration between components.

## 5.2 Architecture Layers

### 5.2.1 Client Side

This is the place where users interact with the system through a web-based user interface.

### 5.2.2 Server-Side

**Presentation Layer:** The API Gateway represents the entry point for User requests. This layer handles communication between the User and the server.

**Business Logic Layer:** Contains the core functionality of the system, including Account management module, Patient visit management module, Speech to Text Module , Medical record Module, Administration Module, Appointment Management Module .

**Data Access Layer:** Provides an abstraction to interact with the database using the Repository pattern, responsible for data access operations.

### 5.2.3 Database

The database stores persistent data and is accessed via repositories in the Data Access Layer. The system uses PostgreSQL as his database.

## 5.3 Sprint 1 Design

### 5.3.1 System Architecture & Component Diagram

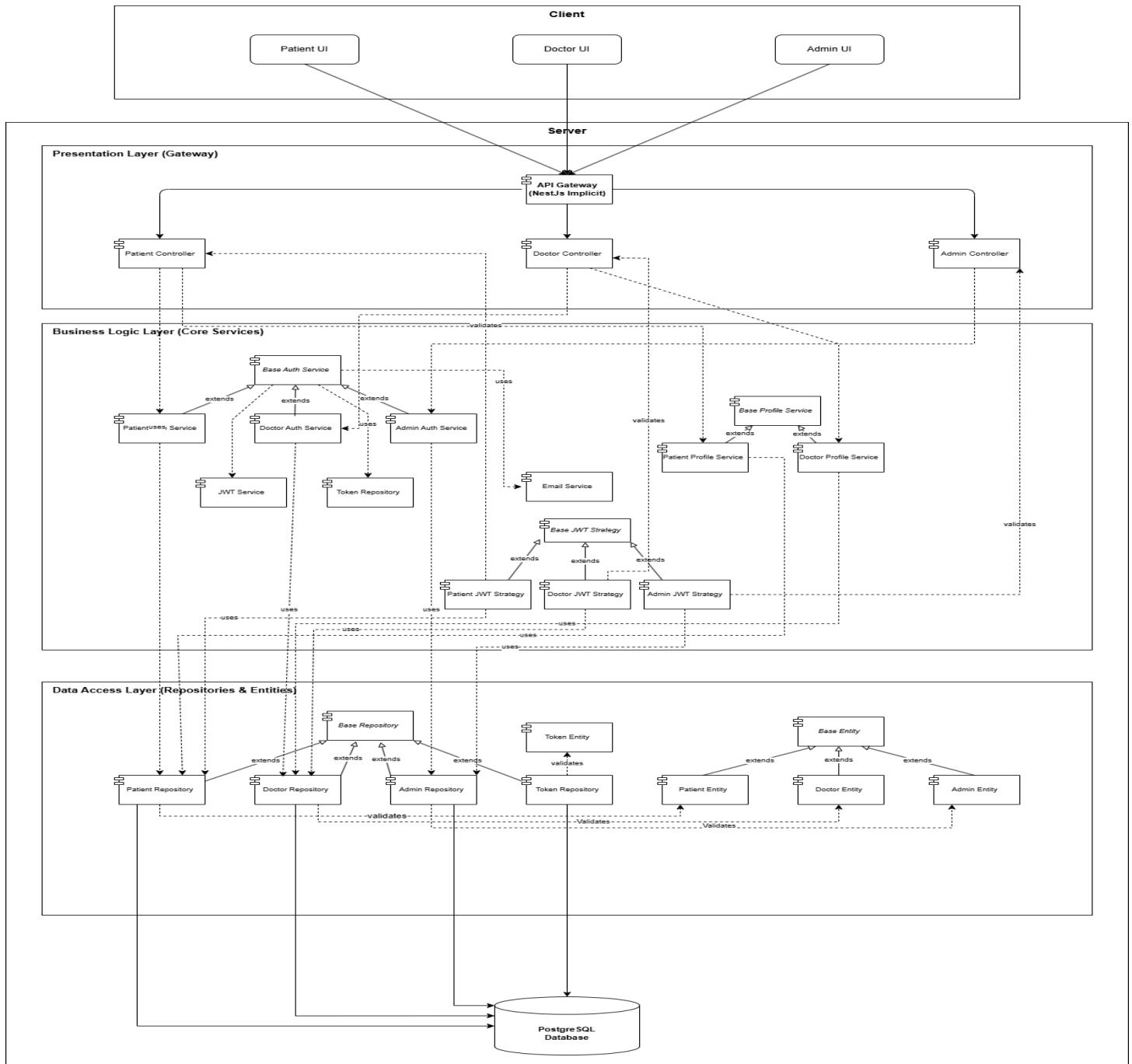


Figure 133, Sprint 1 Component Diagram

## Chapter 5 - System Design

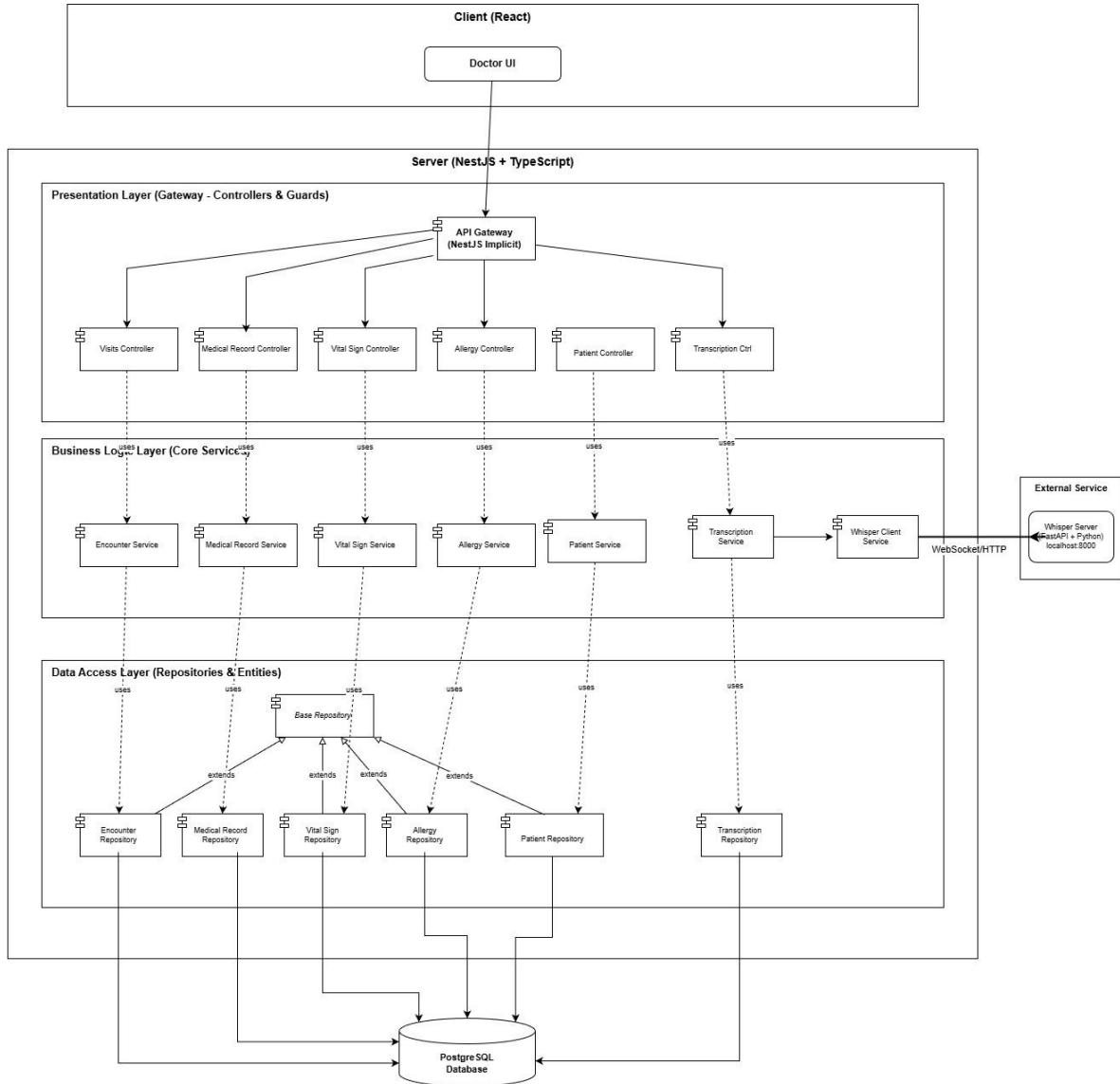


Figure 134, Sprint 2 + 3 Component Diagram

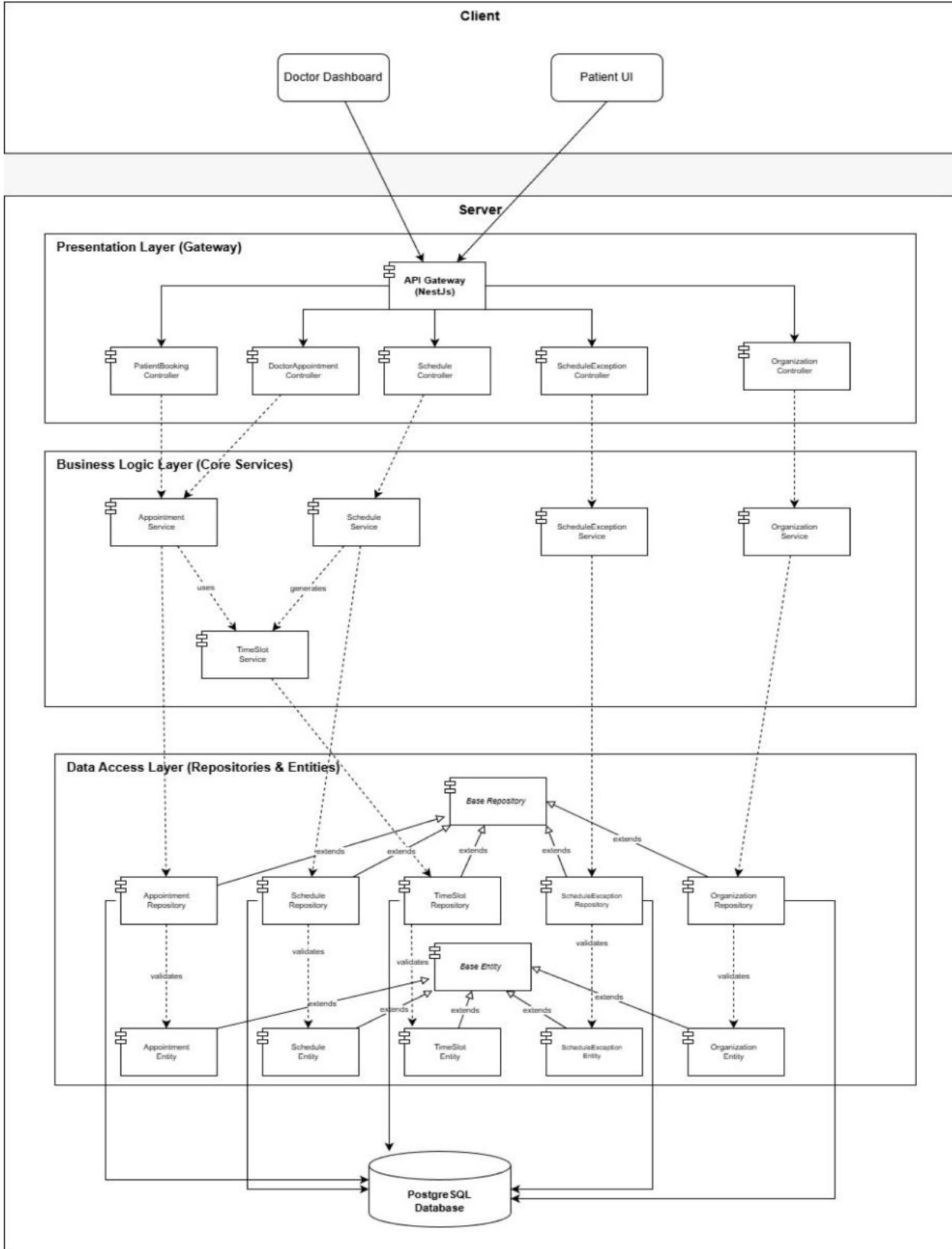


Figure 135, Sprint 4 Component diagram

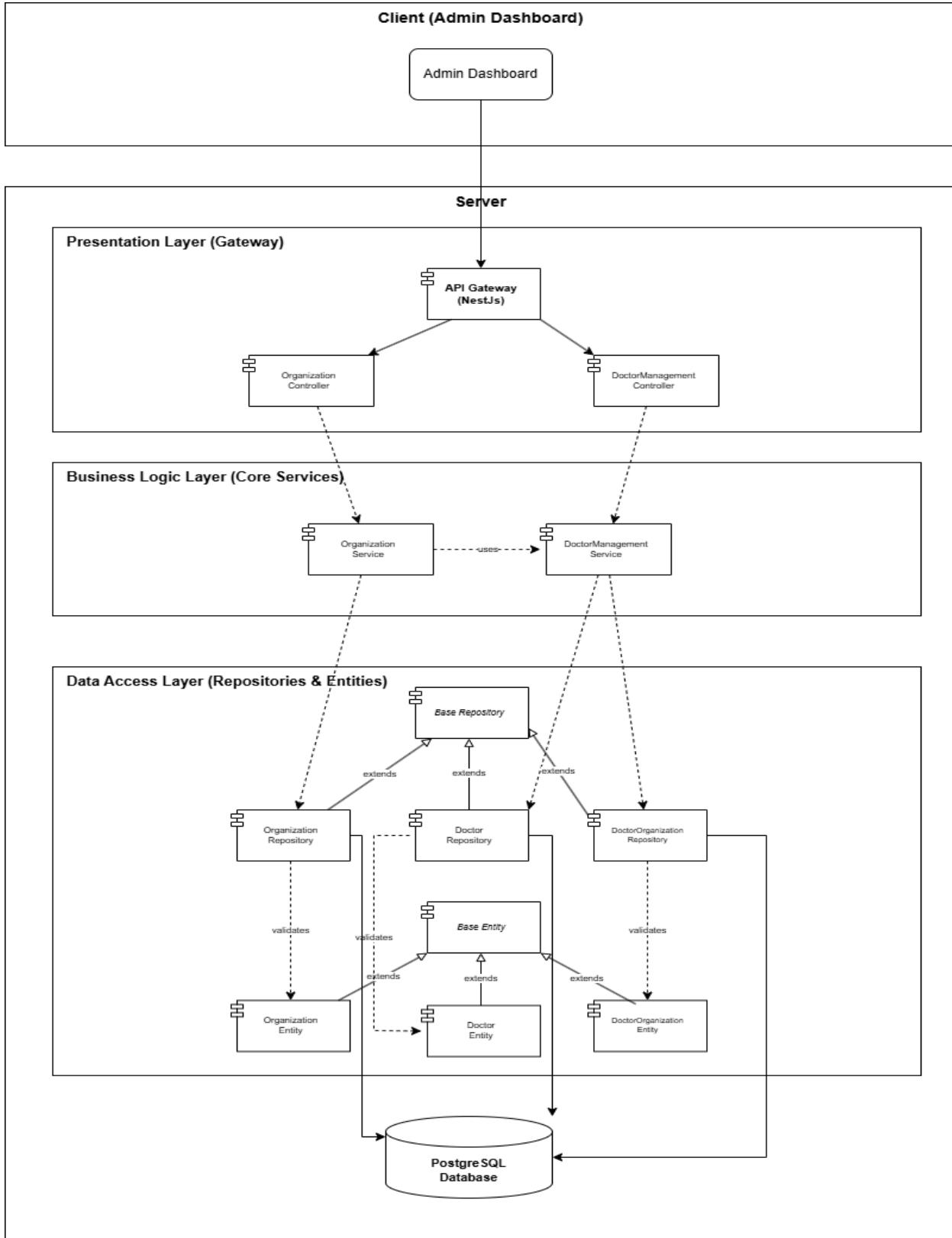


Figure 136, Sprint 5 Component Diagram

### 5.3.2 Class Diagram

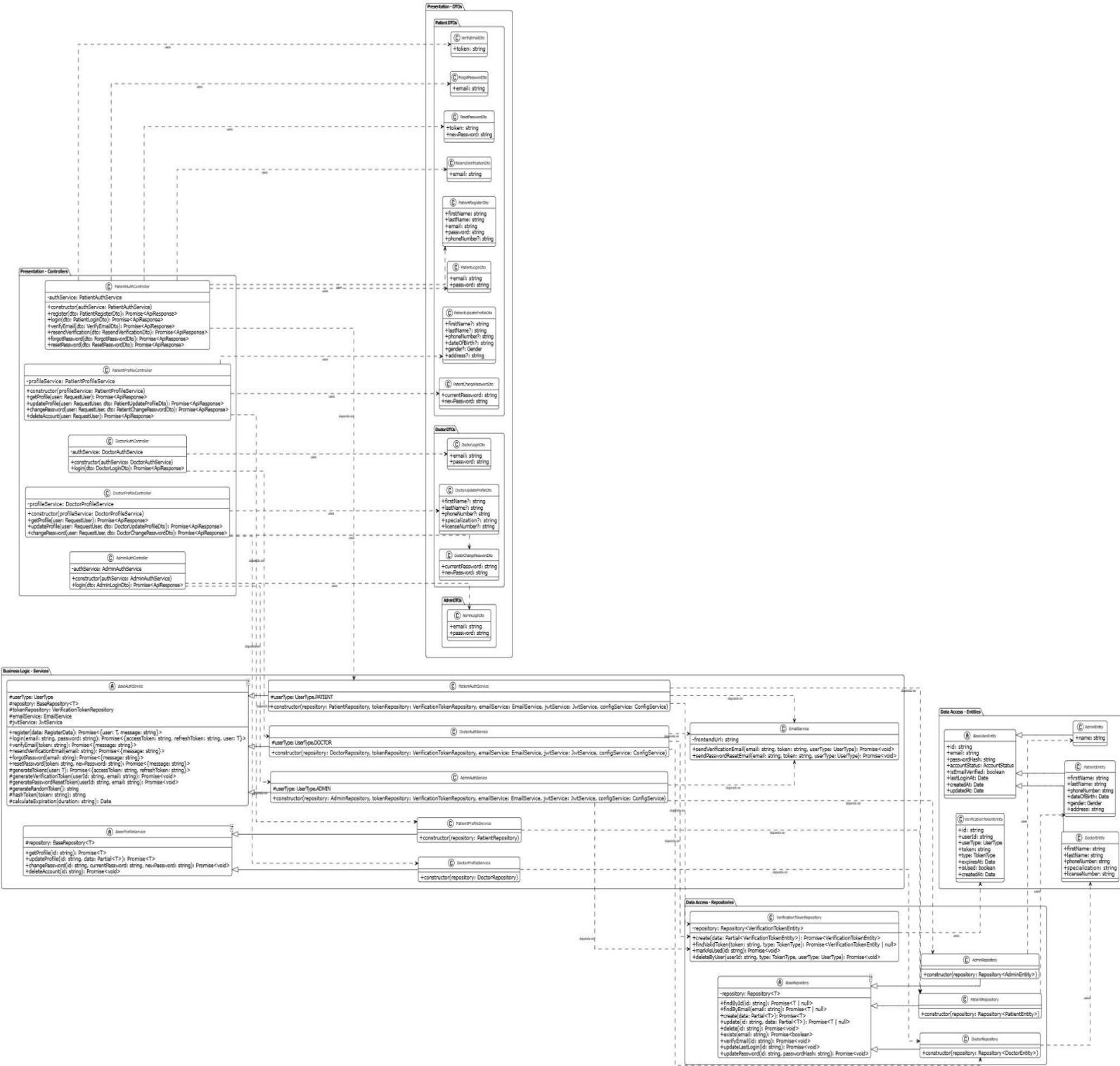


Figure 137, Sprint 1 Class Diagram

## Chapter 5 - System Design

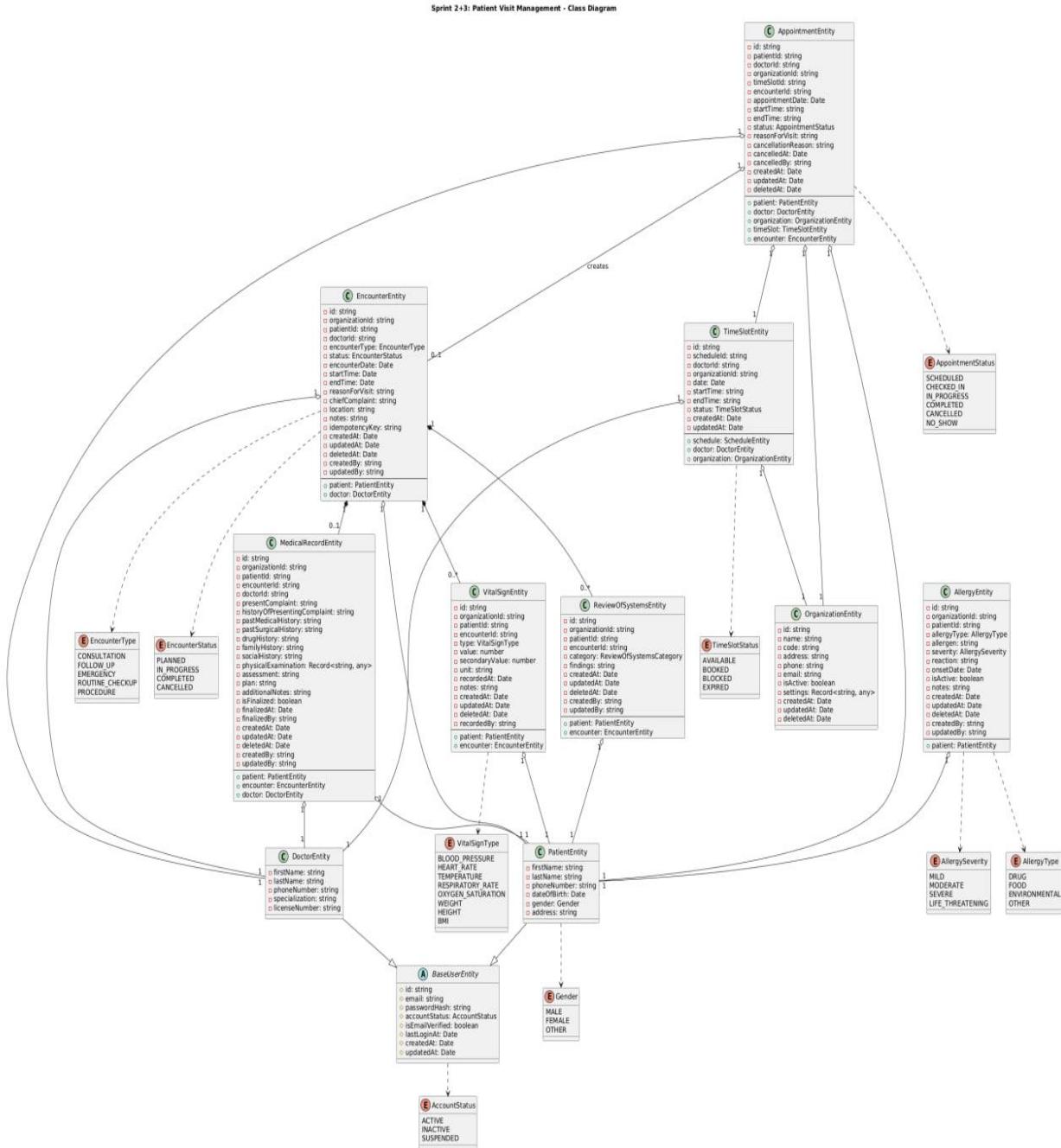


Figure 138, Sprint 2+3 Component Diagram

## Chapter 5 - System Design

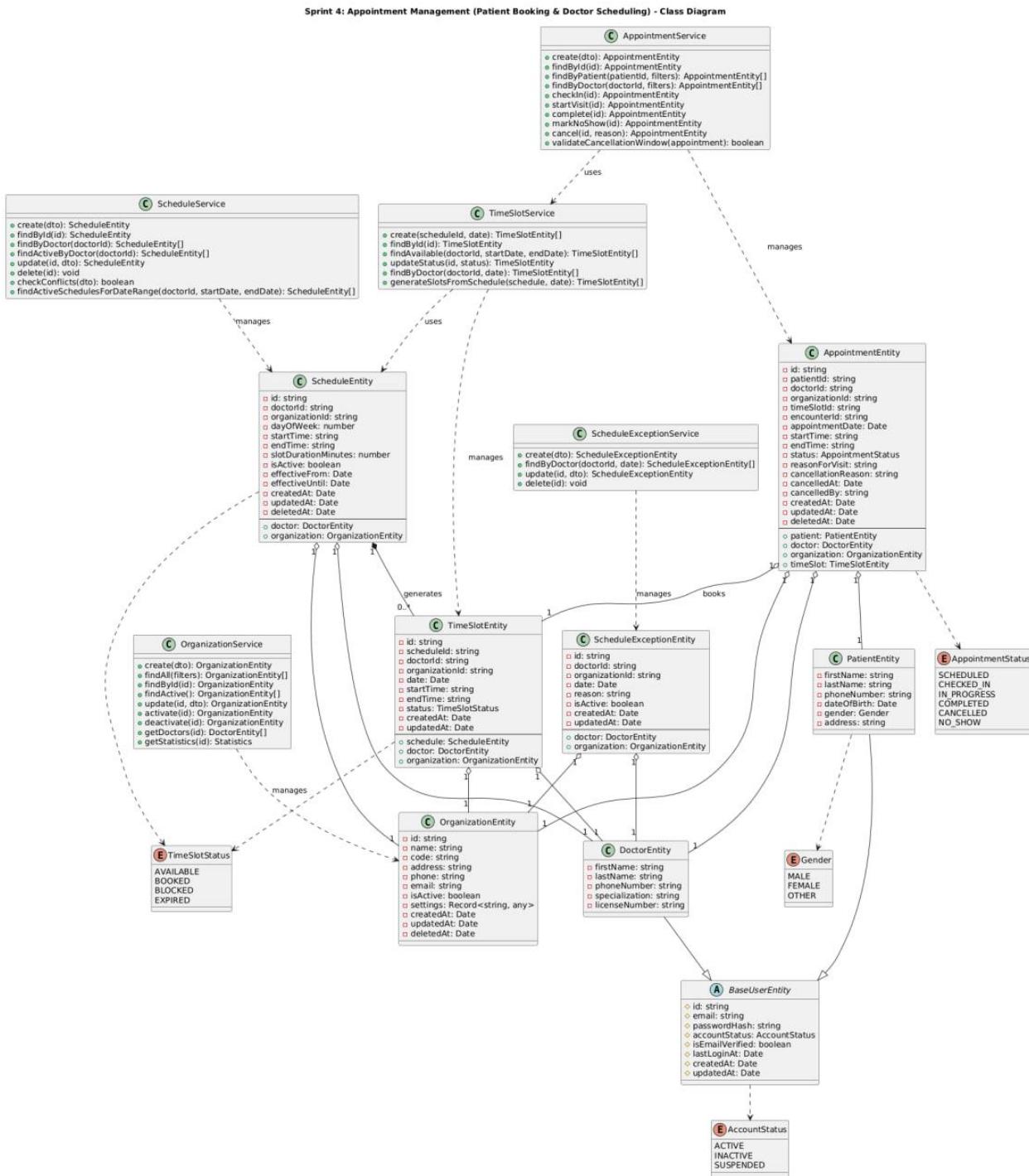


Figure 139, Sprint 4 Component Diagram

## Chapter 5 - System Design

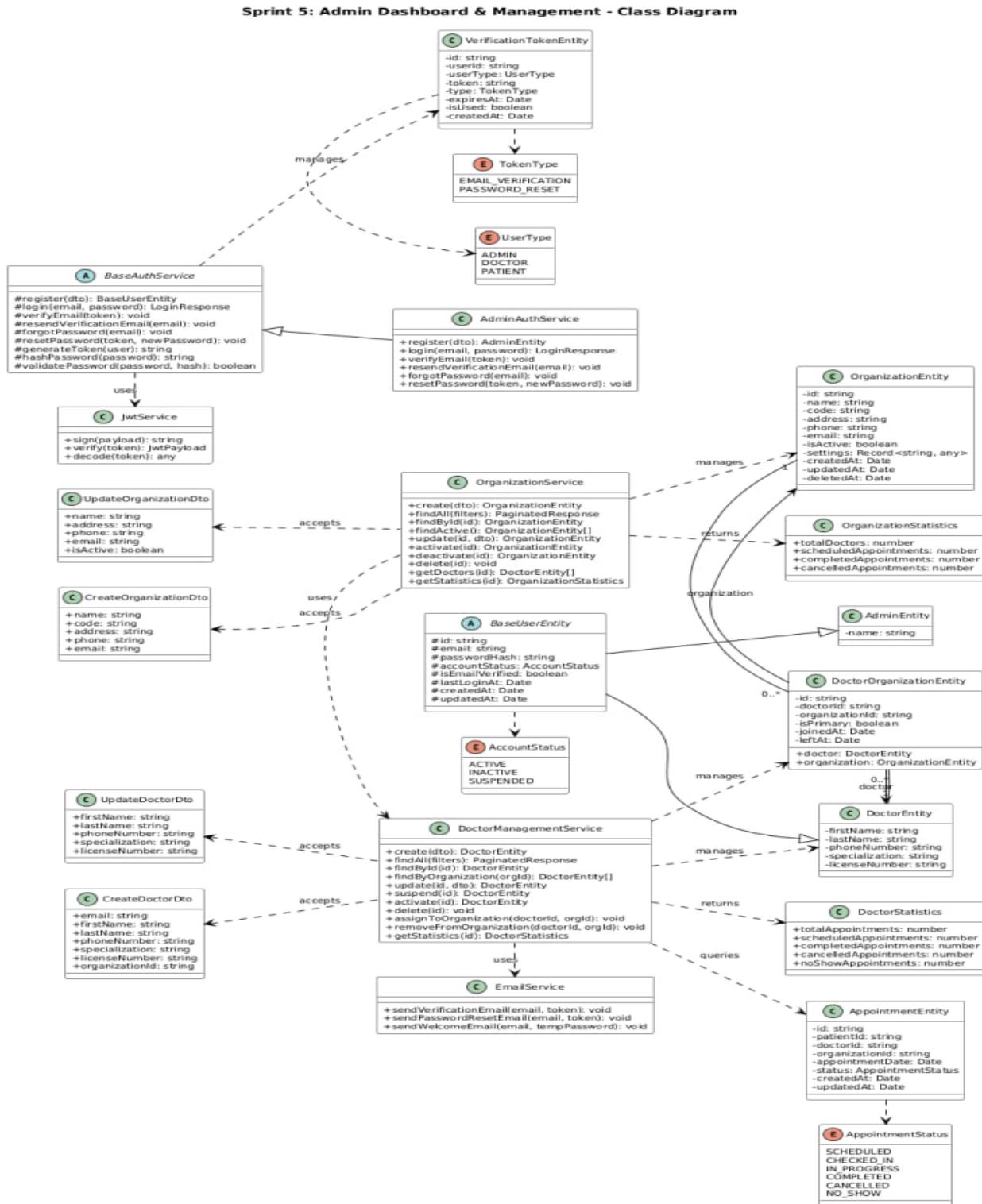


Figure 140, Sprint 5 Class Diagram

5.5 RTM V.3

ID	Title	Analysis Section	Design Section	Code	Integration Test	Unit Test
VEMR-FR-PM-01	The system should allow patients to create a new account by providing email, password, first name, and last name.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-PM-02	The system should allow patients to login to the system using their email and password credentials.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-PM-03	The system should allow patients to verify their email address before accessing the system.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-PM-04	The system should allow users to request a password reset link via email and set a new password.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-PM-05	The system should allow users to view their profile information including personal details.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			

VEMR-FR-PM-06	The system should allow users to update their profile information such as name, phone number, and other details.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-PM-07	The system should allow users to change their current password to a new one.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-AU-08	The system should allow administrators to login to the system using their credentials.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-AU-09	The system should allow clinicians/doctors to login to the system using their email and password.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-VM-10	The system should allow clinicians to create a new patient visit with date, type, reason, and chief complaint.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			

VEMR-FR-VM-11	The system should allow clinicians to search visits by patient name or reason for visit.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-VM-12	The system should allow clinicians to edit visit details when the visit is in progress.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-VM-13	The system should allow clinicians to save changes made to a visit including medical record data.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-VM-14	The system should allow clinicians to delete a visit from the system.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-VM-15	The system should allow clinicians to view complete details of a visit including medical record, vitals, and allergies.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-VM-16	The system should allow voice transcription to appear in real-time as the	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			

	clinician speaks using Whisper.					
VEMR-FR-VM-17	The system should allow clinicians to view a paginated list of all patients in the system.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-VM-18	The system allows clinicians to search for patient.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-VM-19	The system should allow clinicians to view a patient's profile with their visits and allergies.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-VM-20	The system should allow clinicians to view detailed patient information including contact and demographic data.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-VM-21	The system should allow clinicians to change visit status: start (planned to in-progress), complete, or cancel.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			

VEMR-FR-VM-22	The system should allow clinicians to view a paginated list of all visits with filtering options.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-VS-23	The system should allow clinicians to view vital signs recorded for a specific visit.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-VS-24	The system should allow clinicians to record and edit vital signs (BP, HR, temp, etc.) during an active visit.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-AM-25	The system should allow clinicians to view a patient's recorded allergies with severity and reaction details.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-AM-26	The system should allow clinicians to add a new allergy record for a patient with type, allergen, severity, and reaction.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			

VEMR-FR-AM-27	The system should allow clinicians to update existing allergy information for a patient.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-AM-28	The system should allow clinicians to delete an allergy record from a patient's profile.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-CM-29	The system should allow the admin to create clinicians' accounts.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-CM-30	The system should allow the clinicians and admins to update their account.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-CM-31	The system should allow the admin to view the clinicians accounts list.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-CM-32	The system should allow the admin to view the clinicians account details.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			

VEMR-FR-CM-33	The system should allow the admin to delete the clinician's accounts.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-CM-34	The system should allow the admin to search the clinician's accounts.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-OM-35	The system should allow the admin to create organizations.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-OM-36	The system should allow the admin to update organization information.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-OM-37	The system should allow the admin to view all organizations.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-OM-38	The system should allow the admin to view detailed information about a specific organization.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			

VEMR-FR-OM-39	The system should allow the admin to delete an organization.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-OM-40	The system should allow the admin to search for organizations.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-AN-41	The system should allow administrators to view system-wide analytics and reports.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-AP-42	The system should allow clinicians to view their appointment schedule.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-AP-43	The system should allow clinicians to view their daily appointment schedule.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-AP-44	The system should allow clinicians to filter appointments by specific date.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-AP-45	The system should allow clinicians to	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			

	filter appointments by status.					
VEMR-FR-AP-46	The system should allow the doctor to check in a patient's appointment.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-AP-47	The system should allow the doctor to cancel a patient's appointment.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-AP-48	The system should allow the doctor to set an appointment as no show.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-DA-49	The system should allow clinicians to view their personal analytics and performance metrics.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-SM-50	The system should allow clinicians to create their work schedule.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-SM-51	The system should allow clinicians to edit	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			

	their existing work schedule.					
VEMR-FR-SM-52	The system should allow clinicians to delete their work schedule.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-SM-53	The system should allow clinicians to automatically generate available visit slots based on their schedule.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-SM-54	The system should allow clinicians to mark appointments as completed, no-show, or cancelled.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-SM-55	The system should allow clinicians to view detailed information about a specific appointment.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-SM-56	The system should allow clinicians to define their available time slots for appointments.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			

VEMR-FR-PP-57	The system should allow patients to view available healthcare organizations.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-PP-58	The system should allow patients to view doctors within specific organizations.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-PP-59	The system should allow patients to view available appointment slots.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-PP-60	The system should allow patients to book an appointment with a clinician.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-PP-61	The system should allow patients to view their own appointment history and upcoming appointments.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-PP-62	The system should allow patients to filter their appointments by status.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			

VEMR-FR-PP-63	The system should allow patients to view their own medical records.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-PP-64	The system should allow patients to view their own allergy information.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			
VEMR-FR-PP-65	The system should allow patients to view their complete visit history.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>			

Table 74- RTM V.3

## **Chapter 6 - Implementation and Testing**

## 6.1 Introduction

This chapter details the implementation and testing phase of the Voice-Based Electronic Medical Records (EMR) System, focusing on how each system component and interface was developed, integrated, and validated. The goal is to document how the requirements defined in earlier chapters were realized into a working application and how each module was verified through testing to ensure correctness, usability, performance, security, and compliance with healthcare standards. The implementation follows industry best practices for medical software development.

## 6.2 Technologies Used

### 1. Node.js

Node.js is a powerful JavaScript runtime built on Chrome's V8 JavaScript engine that allows developers to execute JavaScript code on the server-side. It enables building scalable and high-performance applications using an event-driven, non-blocking I/O model. Node.js is particularly well-suited for building microservices and API-driven applications, making it ideal for our EMR system's backend architecture.

#### **It's Role in Our System:**

- Powers the backend API server handling all clinical, administrative, and scheduling operations
- Manages concurrent requests from multiple users (doctors, patients, administrators) efficiently
- Provides real-time capabilities for appointments and system updates
- Enables seamless integration with external services such as email notifications and transcription services
- Supports asynchronous processing of medical records and clinical documentation

## 2. Nest.Js

NestJS is a progressive Node.js framework for building efficient, reliable, and scalable server-side applications. It uses TypeScript by default and combines elements of Object-Oriented Programming (OOP), Functional Programming (FP), and Functional Reactive Programming (FRP). NestJS provides an out-of-the-box application architecture that allows developers to create highly testable, scalable, and maintainable applications.

### **It's Role In Our System:**

- Serves as the primary backend framework organizing the entire application architecture
- Implements dependency injection for better code organization and testability
- Provides modular structure separating concerns (authentication, clinical services, scheduling)
- Offers built-in support for validation, guards, and interceptors for security
- Facilitates the creation of RESTful APIs and WebSocket gateways for real-time features

## 3. TypeScript

TypeScript is a strongly typed programming language that builds on JavaScript by adding static type definitions. It enhances code quality, readability, and maintainability by catching errors during development rather than at runtime.

TypeScript's interfaces, classes, and generics have been instrumental in creating a robust architecture for our EMR system, ensuring type safety across the entire codebase.

### **It's Role In Our System:**

- Defines strict types for all medical entities (patients, doctors, appointments, medical records)
- Ensures data integrity through compile-time type checking
- Provides IntelliSense and autocomplete for improved developer productivity
- Creates clear contracts between different system modules through interfaces
- Reduces runtime errors in critical healthcare operations

## 4. React

React is a popular JavaScript library developed by Meta for building user interfaces, primarily for web applications. It is based on a component-driven architecture, allowing developers to create reusable UI elements and manage complex application states efficiently. React employs a virtual DOM for optimized rendering, ensuring high performance and responsiveness. It's particularly suited for building dynamic, single-page applications (SPAs) and offers flexibility to integrate with other libraries or frameworks for managing functionality like routing or state management.

### It's Role In Our System:

- Powers the frontend user interfaces for patients, doctors, and administrators
- Manages complex UI states for appointment scheduling and medical record viewing
- Provides responsive and intuitive interfaces for clinical documentation
- Enables real-time updates for appointment status changes
- Implements role-based UI components for different user types

## 5. PostgreSQL

PostgreSQL is a powerful, open-source object-relational database system with a strong reputation for reliability, feature robustness, and performance. It supports advanced data types, complex queries, and ACID (Atomicity, Consistency, Isolation, Durability) compliance, making it ideal for applications requiring data integrity and consistency.

### **It's Role In Our System:**

- Stores all critical medical data including patient records, appointments, and clinical documentation
- Maintains referential integrity between related entities (patients, doctors, encounters, allergies)
- Supports complex queries for medical record retrieval and reporting
- Provides transaction support for ensuring data consistency in multi-step operations
- Enables efficient indexing for fast retrieval of patient information

## **6. Jest**

Jest is a delightful JavaScript testing framework with a focus on simplicity and support for TypeScript. It provides a complete testing solution with built-in test runners, assertion libraries, and mocking capabilities. Our project leverages Jest for comprehensive unit and integration testing of all system components.

### **It's Role In Our System:**

- Tests all authentication flows (registration, login, password reset)
- Validates clinical service operations (allergies, encounters, medical records)
- Verifies scheduling logic (appointments, time slots, conflicts)
- Tests administrative functions (doctor management, organization setup)
- Ensures business rules are correctly implemented (cancellation policies, access control)

## 6.3 Testing Report

### 6.3.1 Introduction

This report provides a comprehensive analysis of the test coverage and quality assurance measures implemented for the Electronic Medical Records (EMR) System backend. The testing suite demonstrates exceptional coverage across all critical system components, with 324 tests across 21 test suites for unit testing.

And all API Endpoints tested for integration testing using postman

### 6.3.2 Unit Testing Metrics

```
===== Coverage summary =====
Statements : 95.96% ( 381/397 )
Branches   : 93.04% ( 107/115 )
Functions   : 92.68% ( 76/82 )
Lines       : 95.68% ( 355/371 )
=====
Test Suites: 21 passed, 21 total
Tests:      324 passed, 324 total
Snapshots:  0 total
Time:       14.001 s
Ran all test suites.
```

Figure 141, Testing metrics picture

### 6.3.3 Integration Testing

We did integration testing using postman and we put all the test results on the Jira Tickets

## Chapter 6 - Implementation and Testing

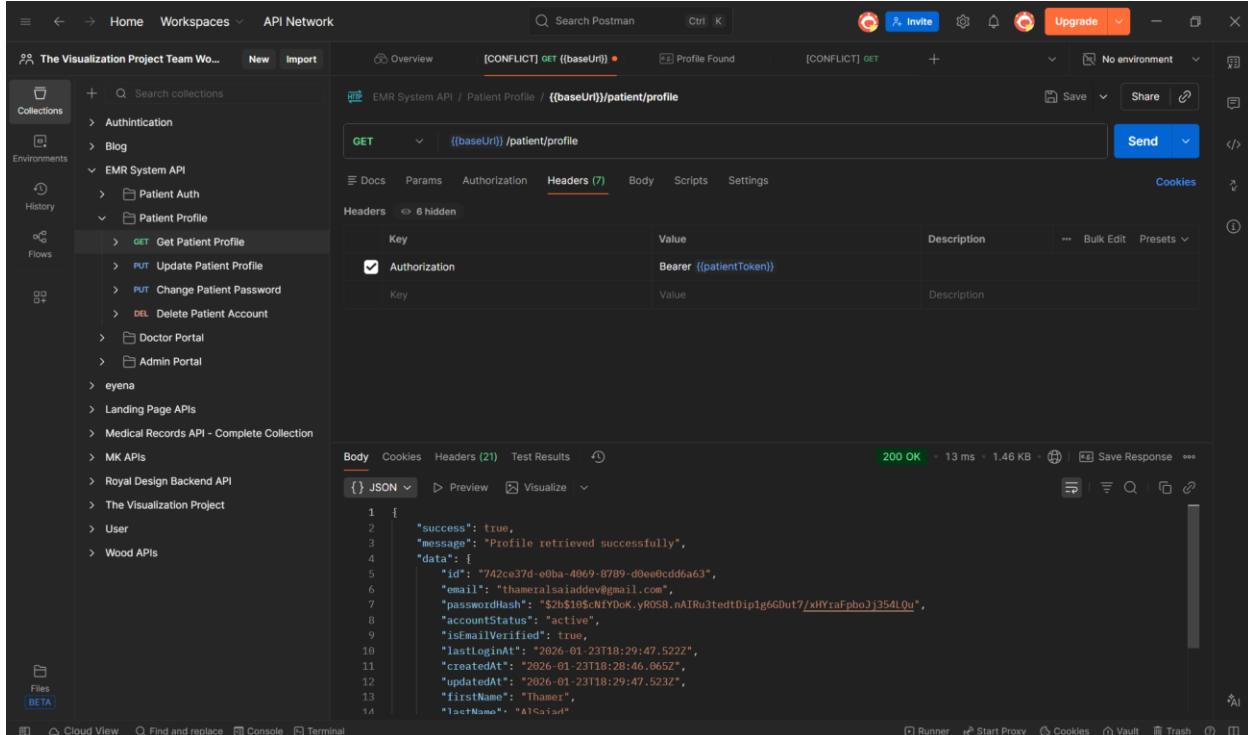


Figure 142, Postman Integration Testing

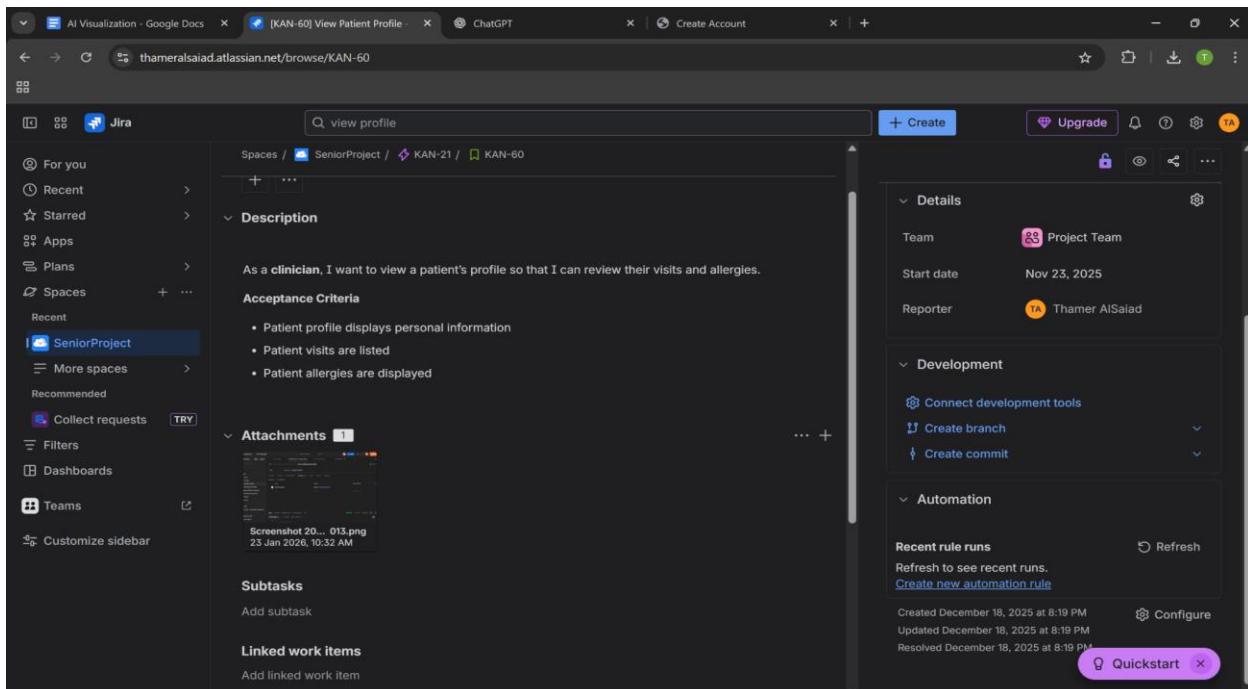
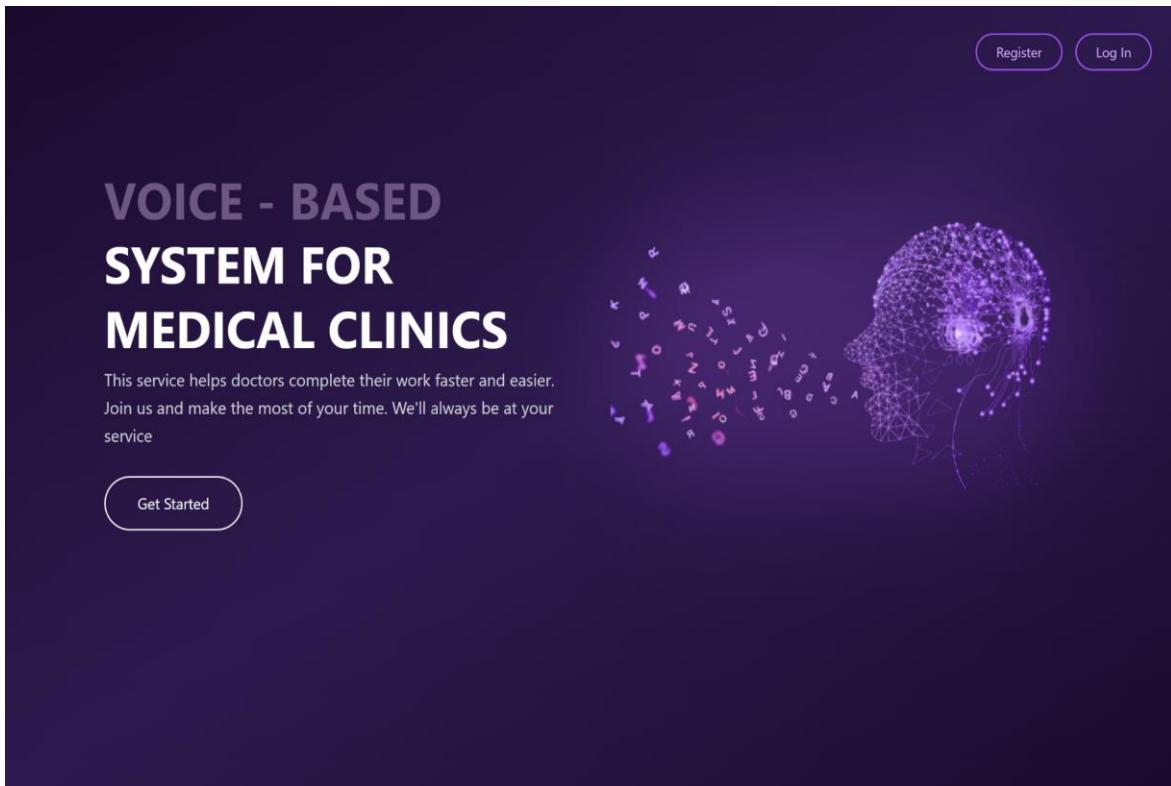


Figure 143, Jira Ticket with postman test attached

## 6.3 System Interfaces



**Figure 144, Main user interface**

The image shows the 'Create An Account' form. It includes fields for First name and Last name, both with placeholder text 'John Doe'. There is also an Email Address field and two Password fields for entering and confirming a password. A note below the password fields says 'Use 8 or more characters with letters, numbers & symbols'. A 'Show password' checkbox is available. A large 'Create an account' button is at the bottom. To the right of the form is a circular illustration featuring three medical professionals (a male doctor, a female doctor, and a nurse) surrounded by medical icons like a heart rate monitor, test tubes, and pills.

**Create An Account**

First name  Last name

Email Address

Password  Confirm your password

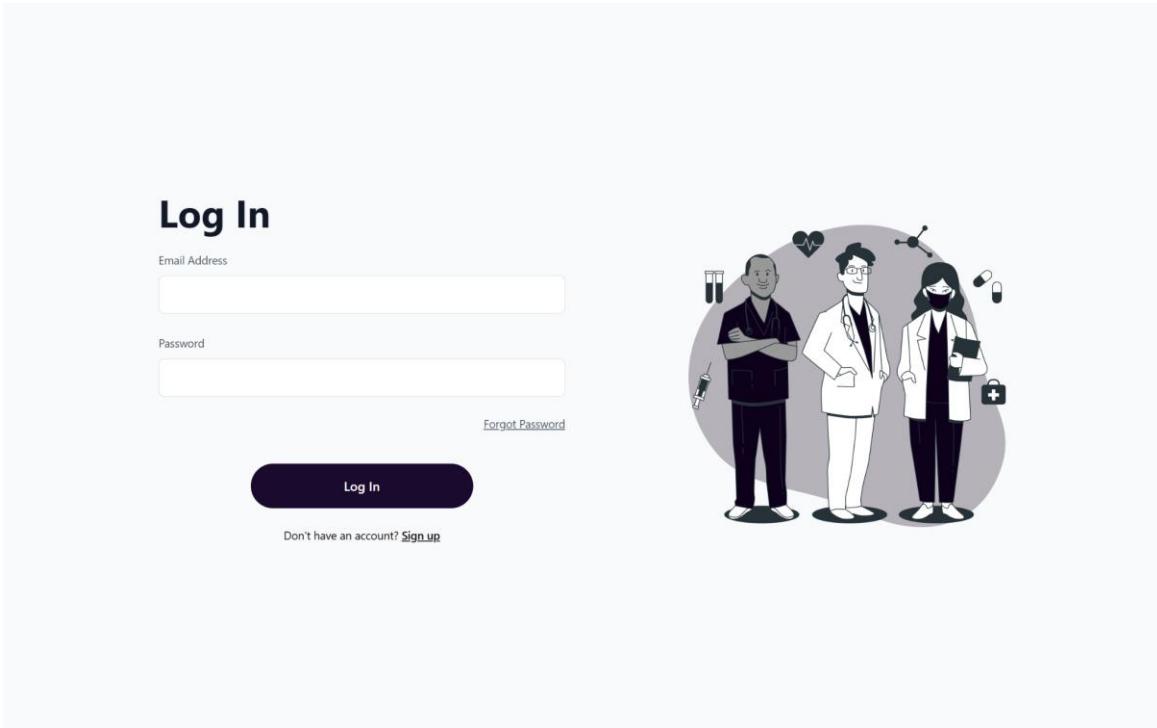
Use 8 or more characters with letters, numbers & symbols

Show password

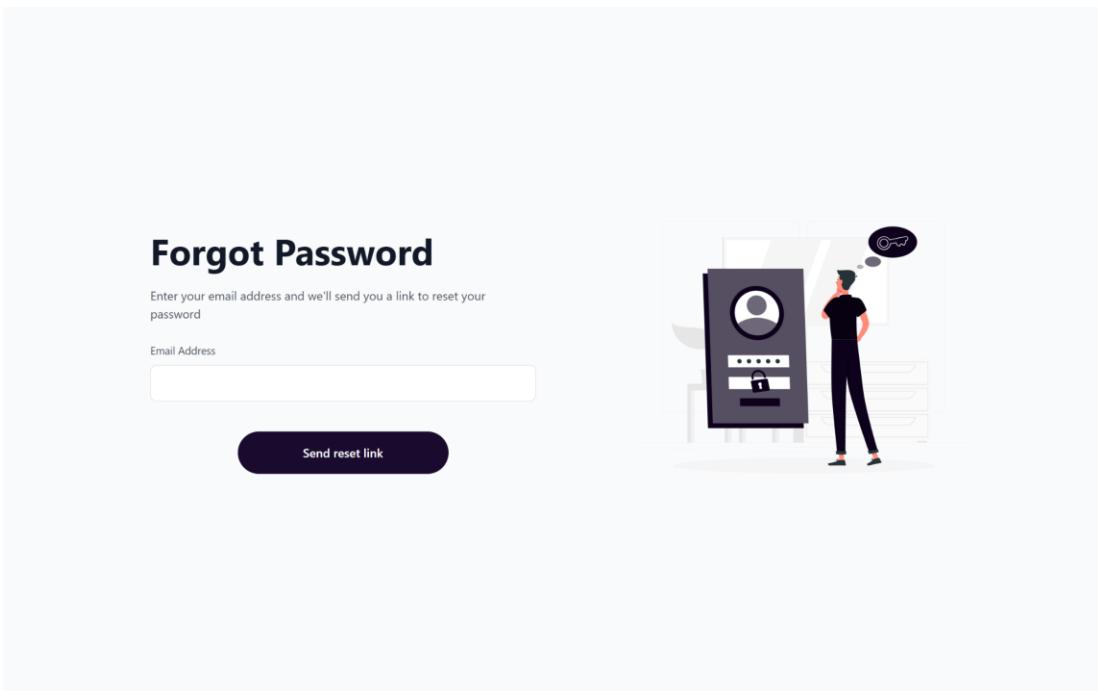
**Create an account**

Already have an account? [Login](#)

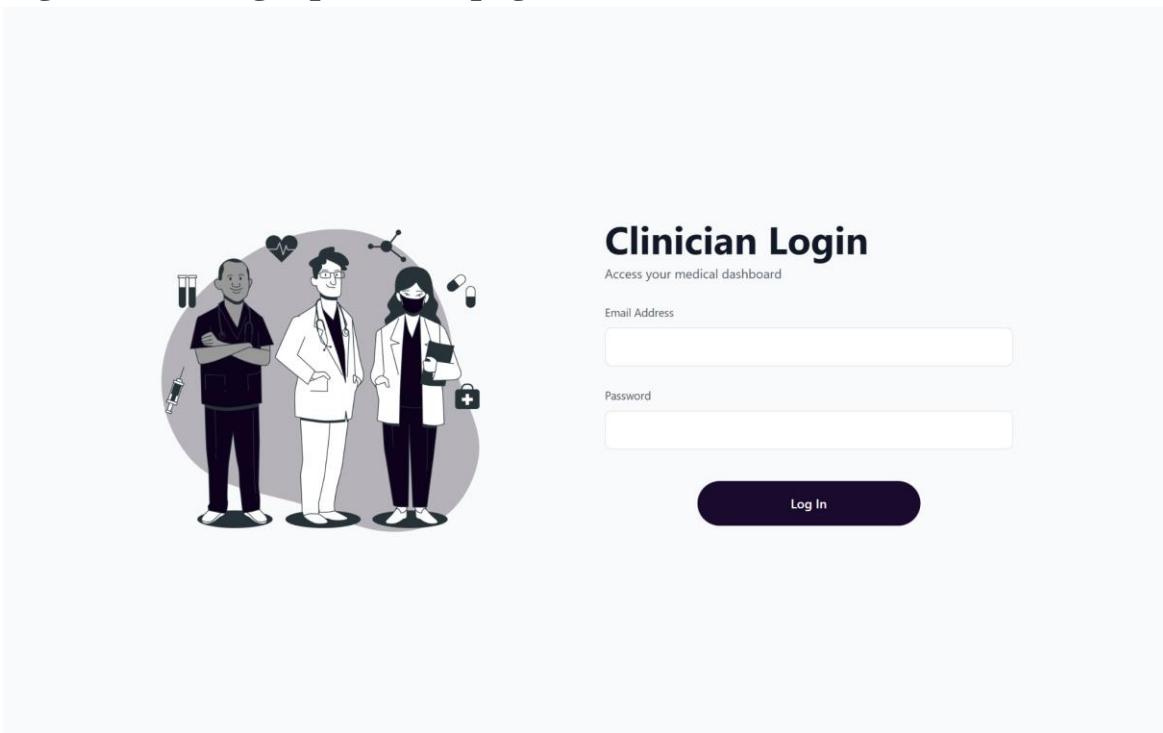
**Figure 145, User registration page**



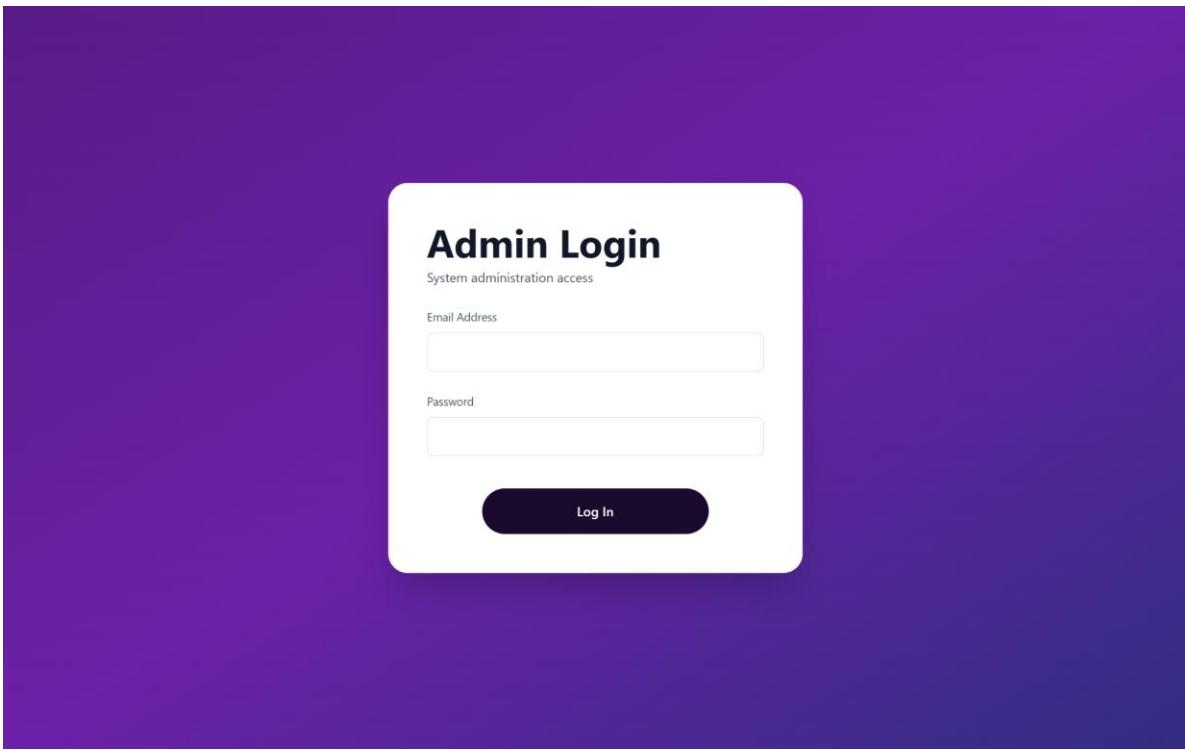
**Figure 146, Login page**



**Figure 147, Forgot password page**

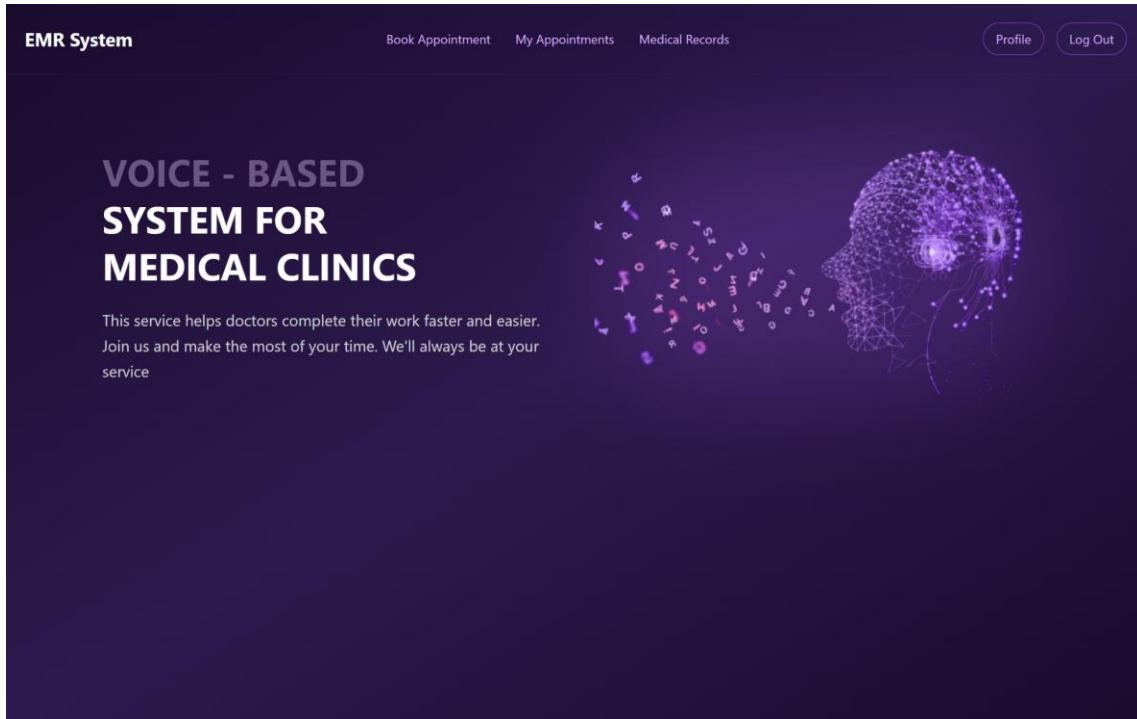


**Figure 148, Clinician Login page**

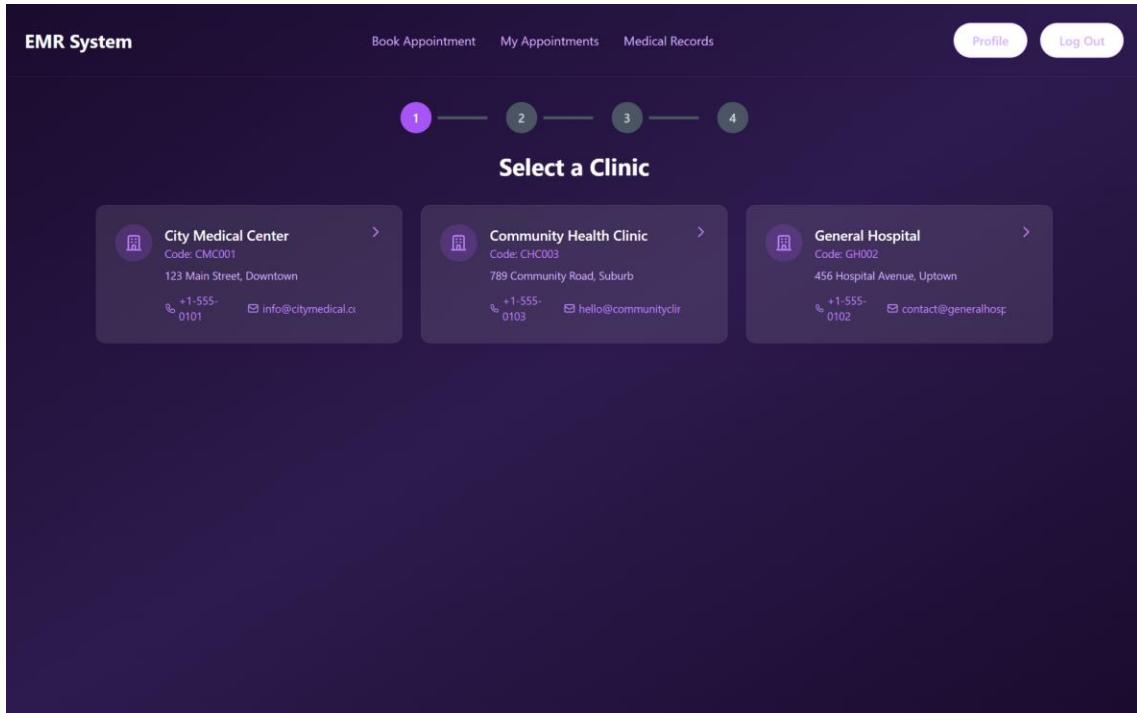


**Figure 149, Admin Login page**

## Chapter 6 - Implementation and Testing

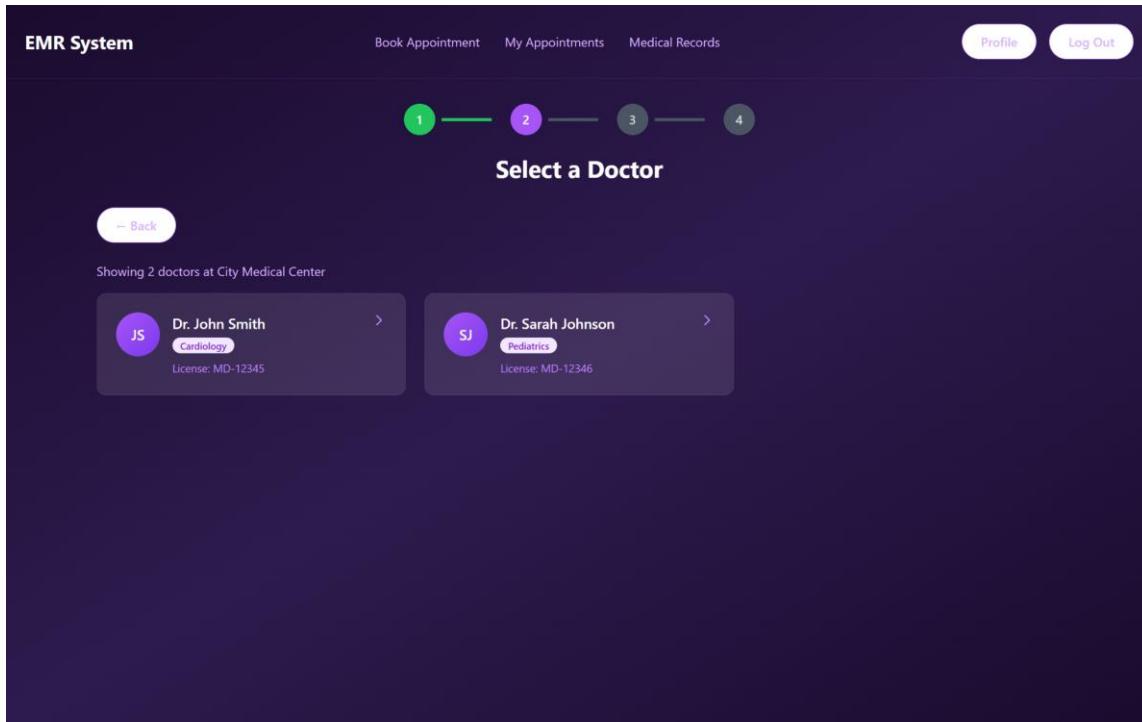


**Figure 150, User home page**

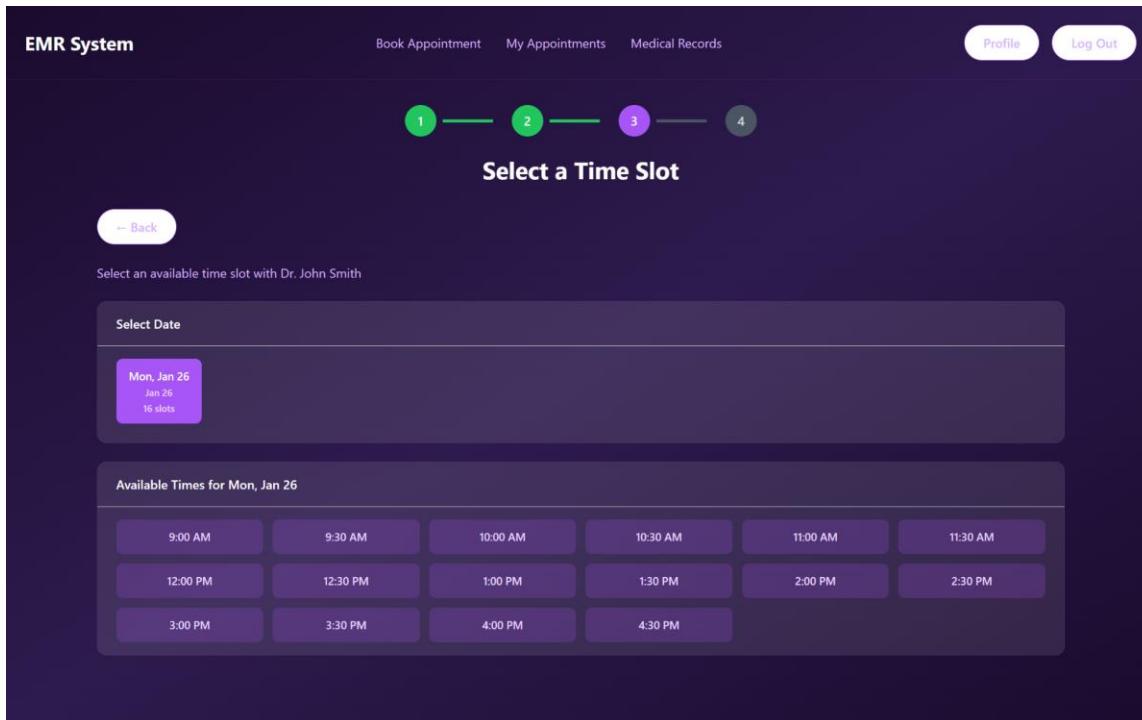


**Figure 151, Page for selecting the organization to book an appointment**

## Chapter 6 - Implementation and Testing



**Figure 152, Page for selecting the clinician to book an appointment**



**Figure 153, Page for selecting a time slot to book an appointment**

## Chapter 6 - Implementation and Testing

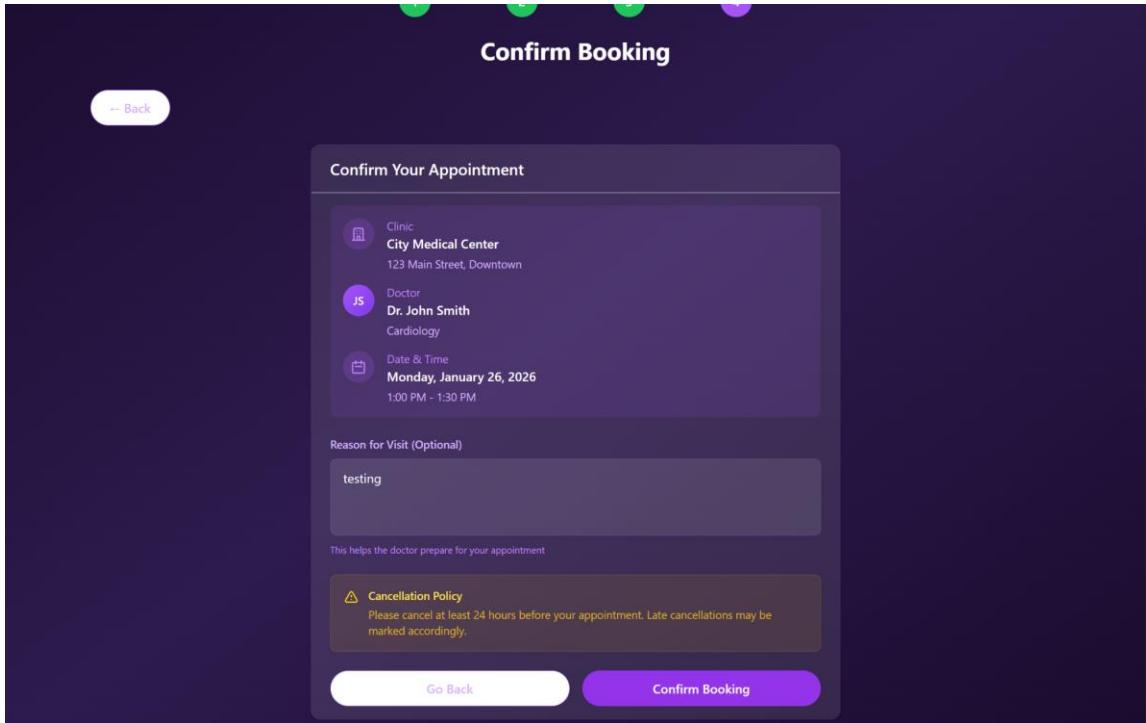


Figure 154, Booking confirmation page

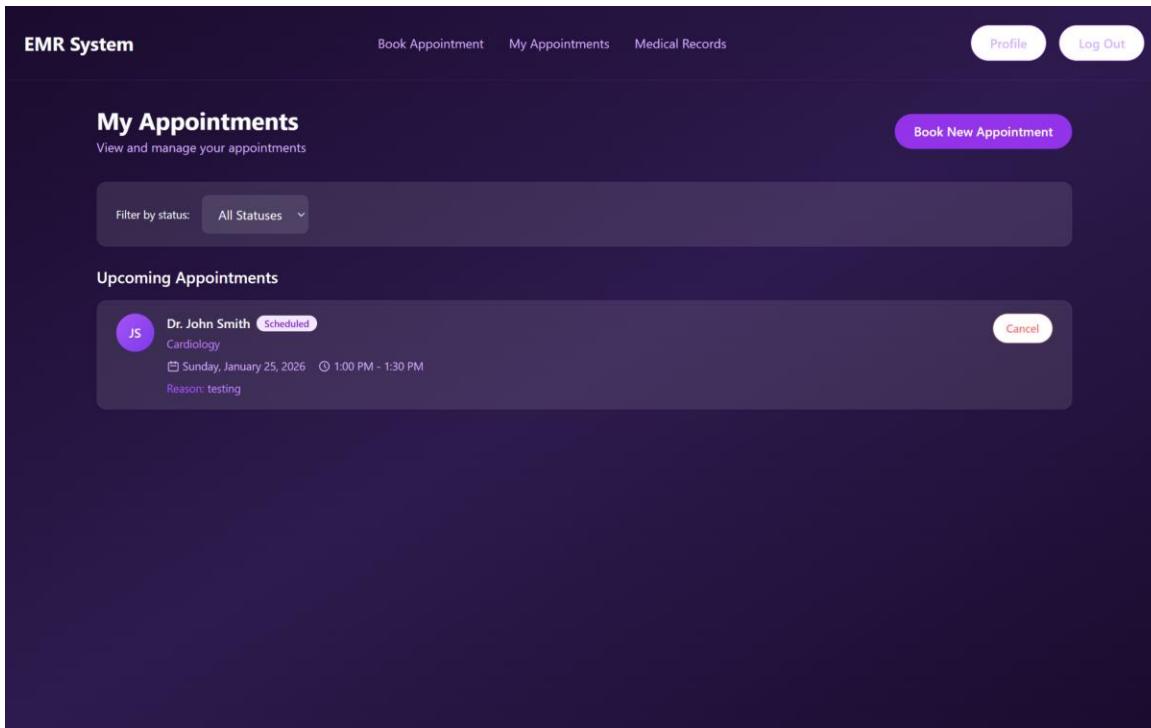


Figure 155, My Bookings View Page

## Chapter 6 - Implementation and Testing

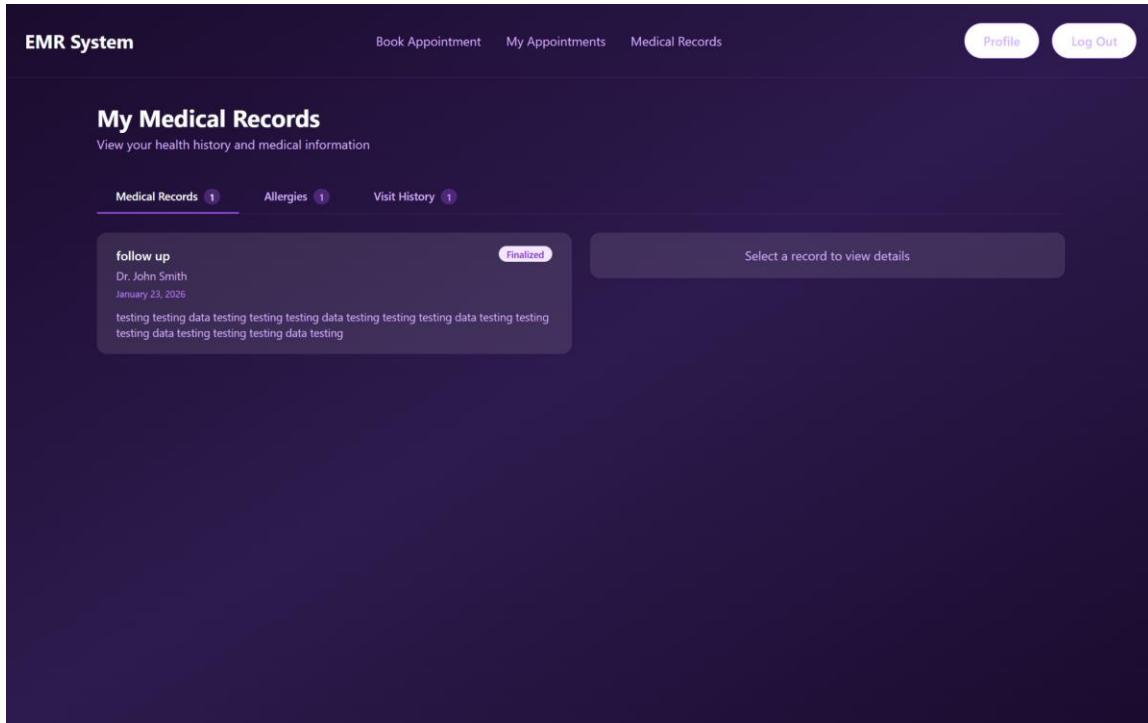


Figure 156, View my medical record page

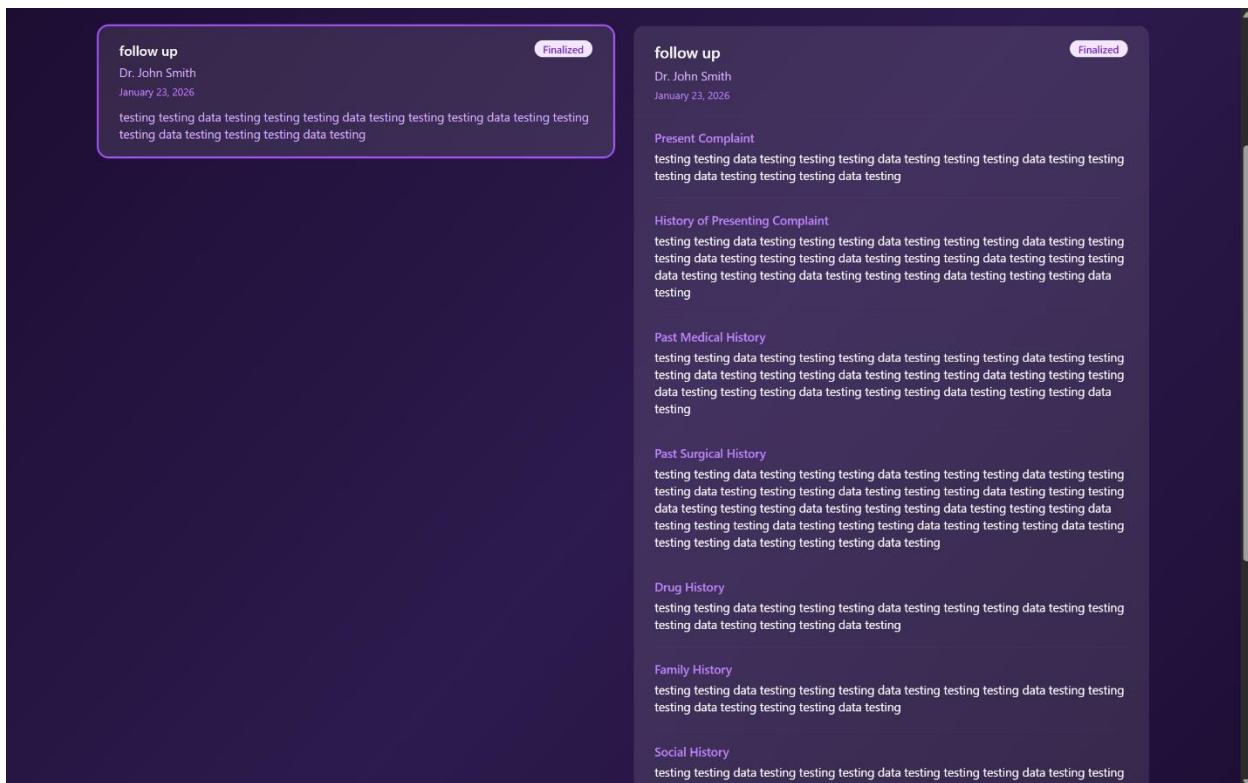


Figure 157, View own Medical Record Details

## Chapter 6 - Implementation and Testing

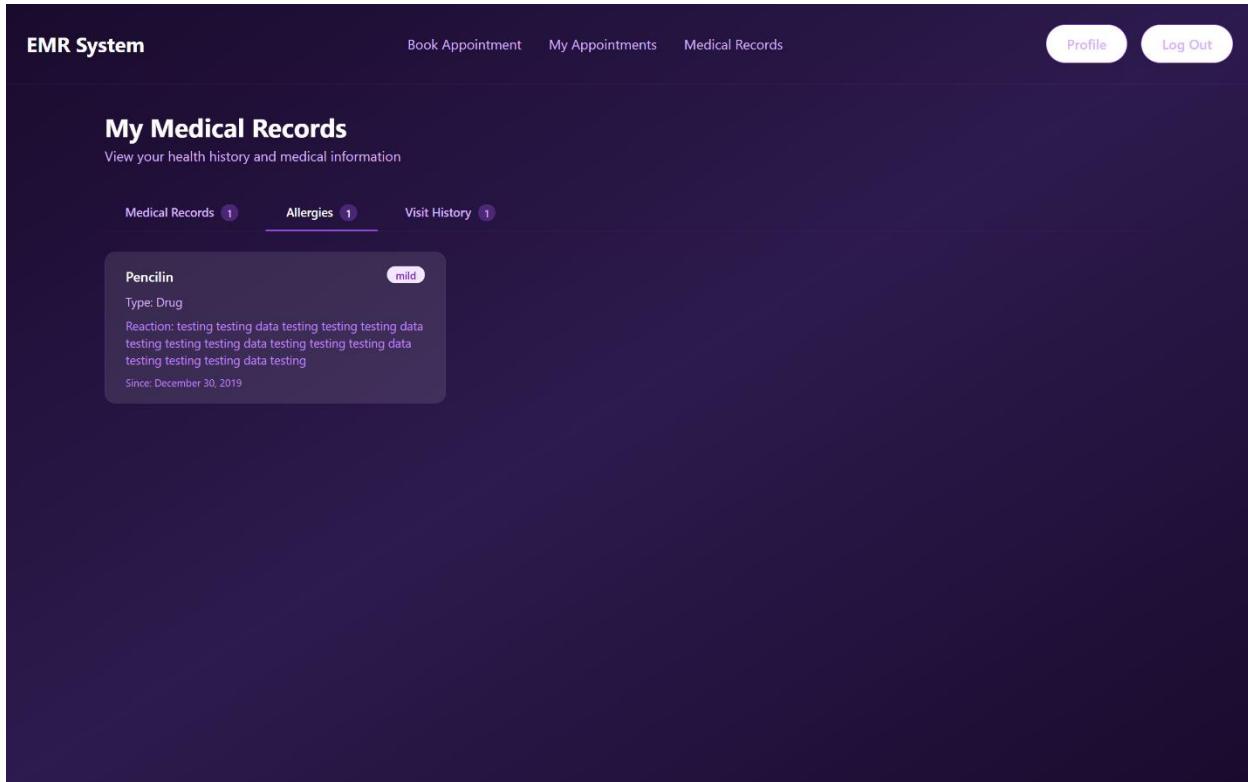


Figure 158, View own allergies

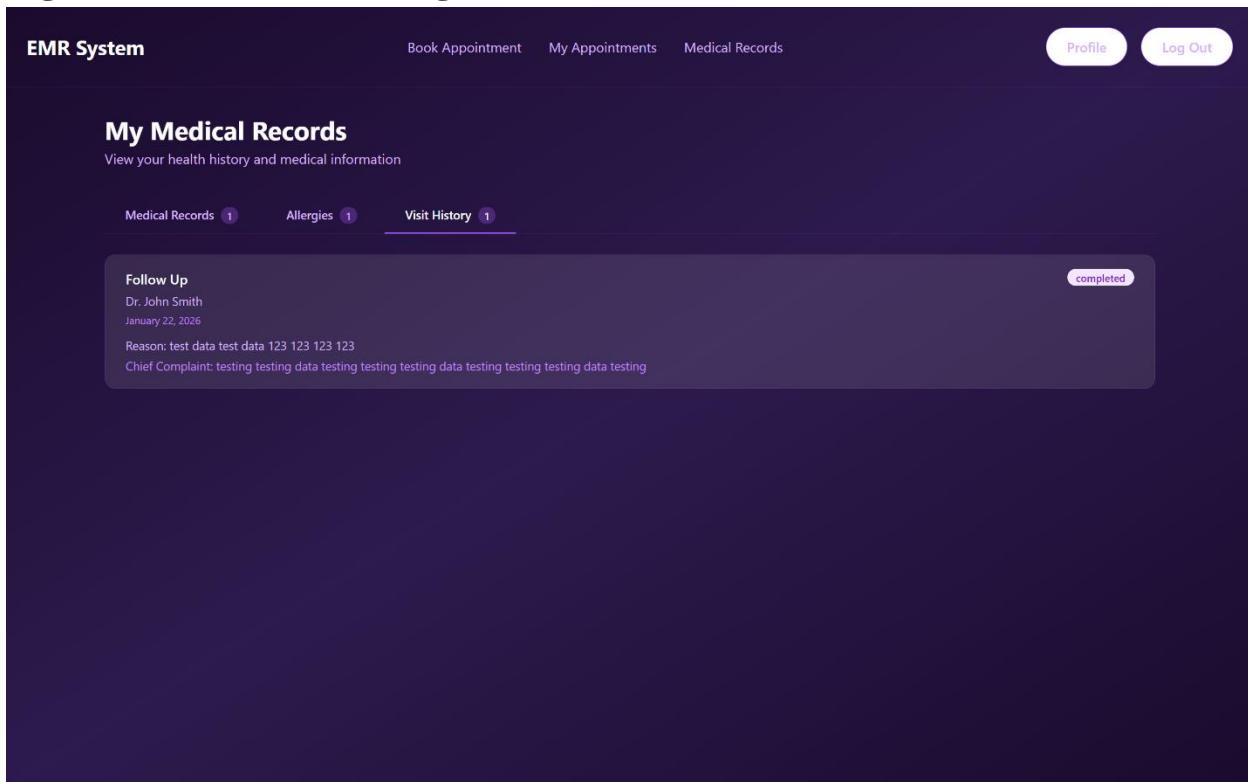
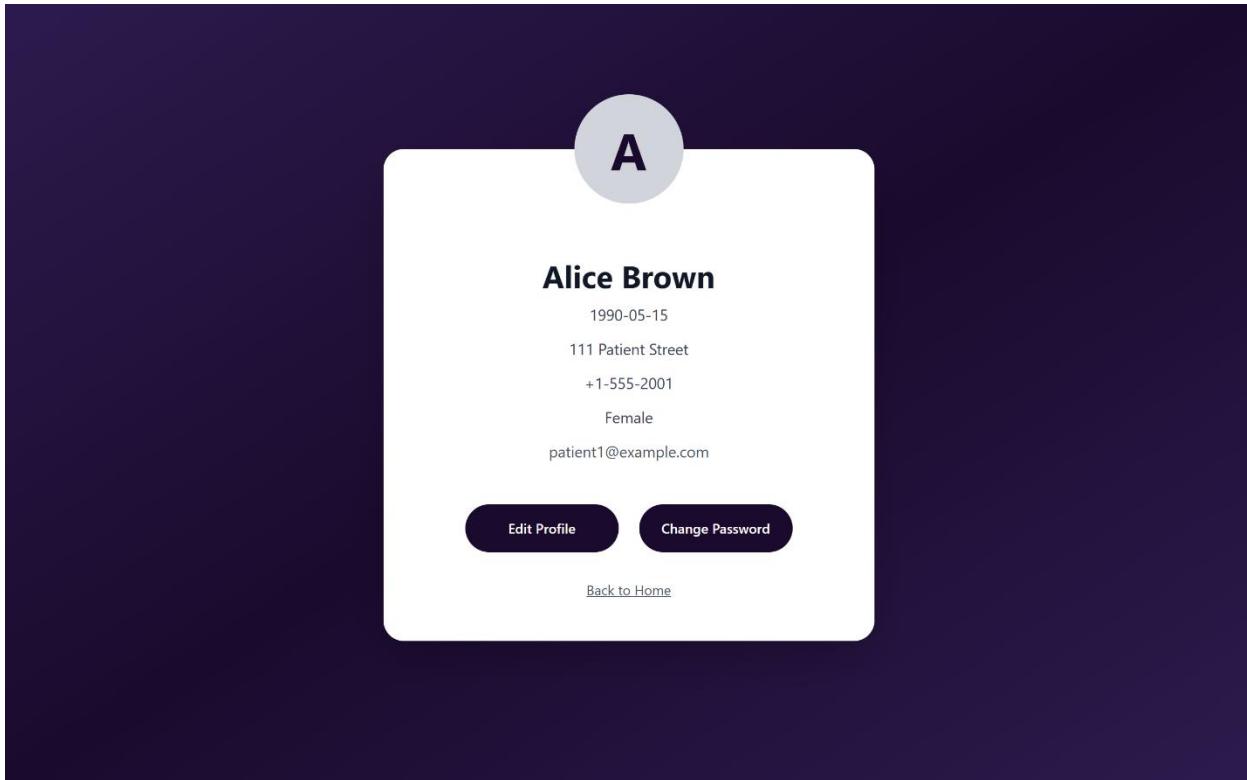
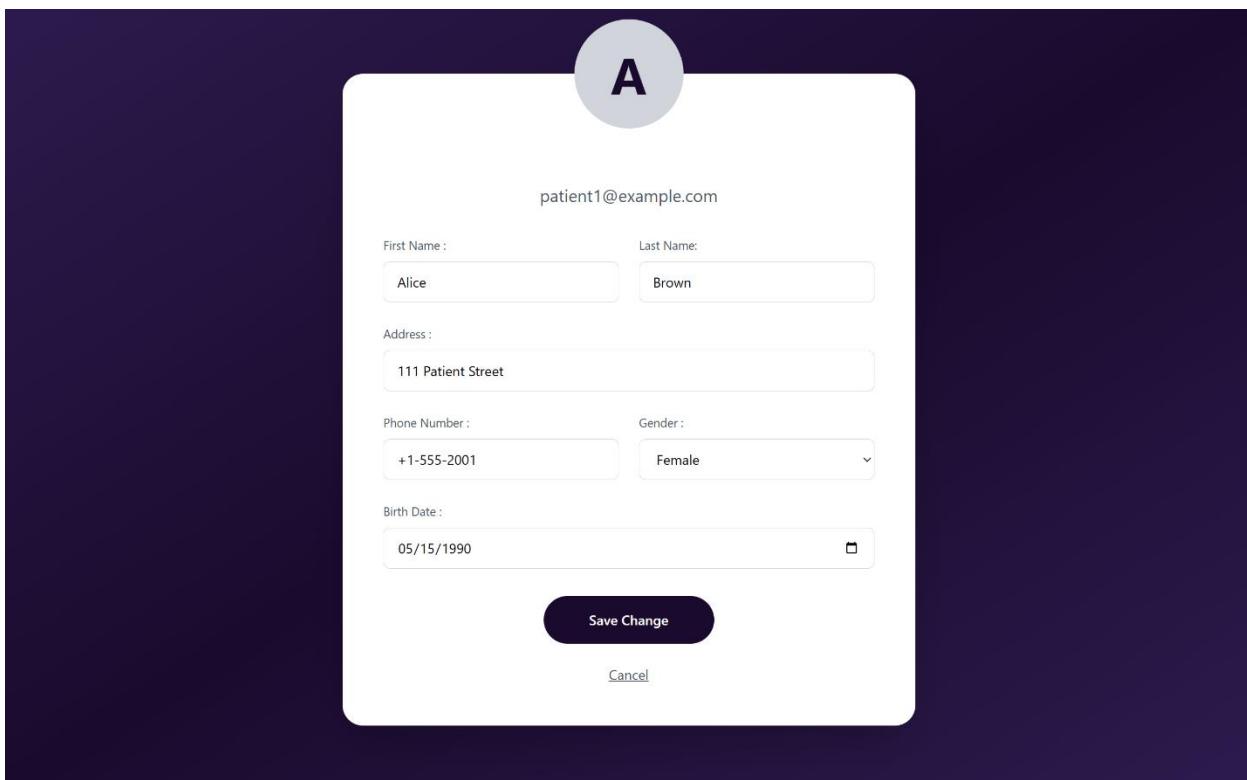


Figure 159, View Own Visit History Details

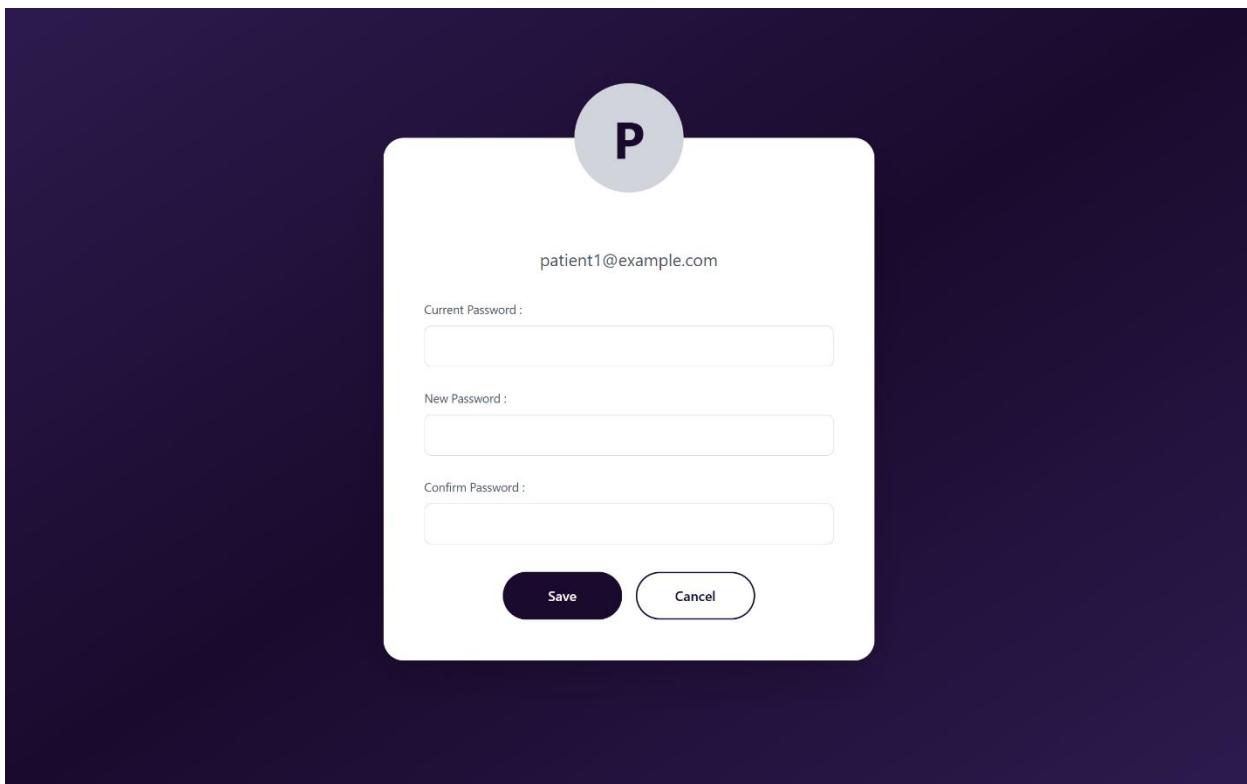


**Figure 160, View Own Profile**



**Figure 161, Edit Own Profile**

## Chapter 6 - Implementation and Testing



**Figure 162, Change Password**

A screenshot of the Doctor Dashboard. On the left is a dark sidebar with the title 'EMR Portal' at the top. Below it are navigation links: 'Dashboard', 'Schedule', 'Appointments', 'Visits', and 'Patients'. At the bottom of the sidebar are 'Dr. Alice Brown' and a 'Logout' link. The main area is titled 'Dashboard' with the sub-instruction 'Welcome back, Doctor'. It features three purple summary cards: 'Active Visits' (0), 'Total Visits' (1), and a 'Quick Action' button labeled 'New Visit' with a plus sign. Below these are two tables: 'Active Visits' (No active visits) and 'Recent Visits' (listing 'Alice Brown' with a 'follow up' note and a 'completed' status).

Active Visits		Recent Visits	
0	View All	Alice Brown	View All
No active visits		follow up • 1/22/2026	
		completed	

**Figure 163, View Doctor Dashboard and Analytics**

## Chapter 6 - Implementation and Testing

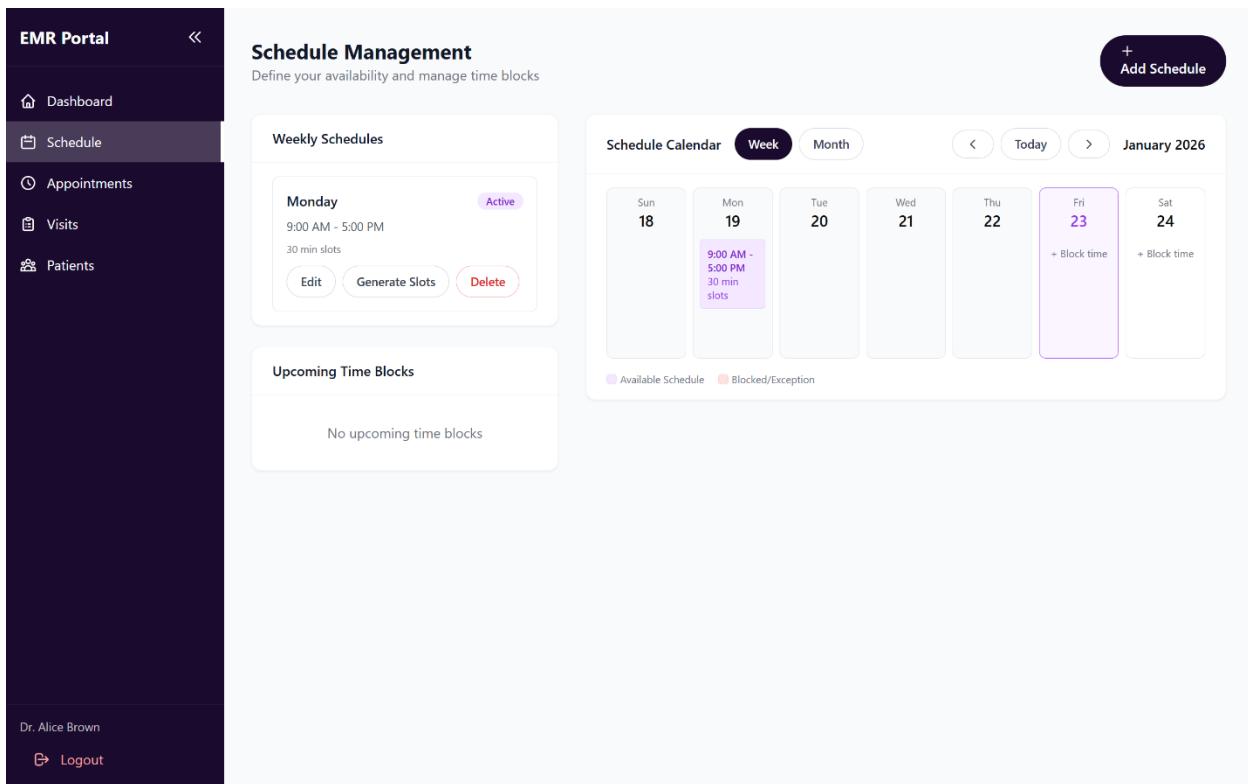


Figure 164, View Own Schedules

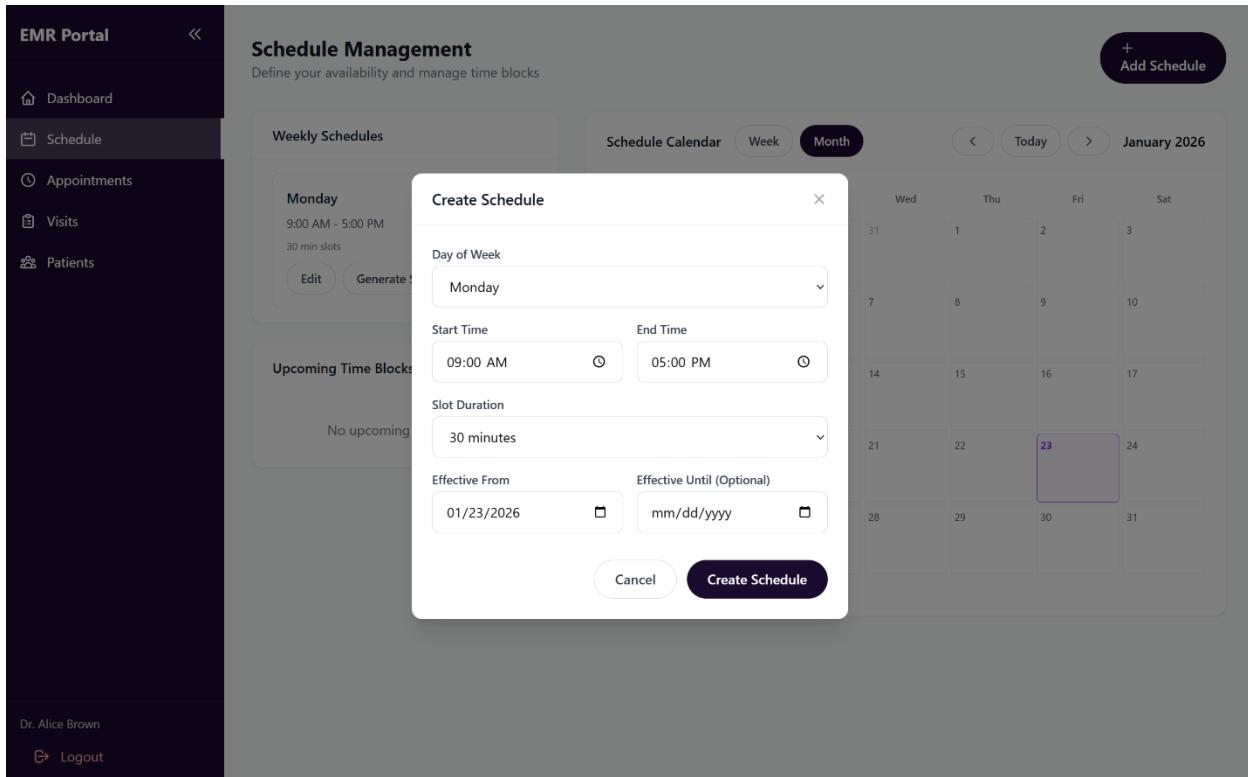
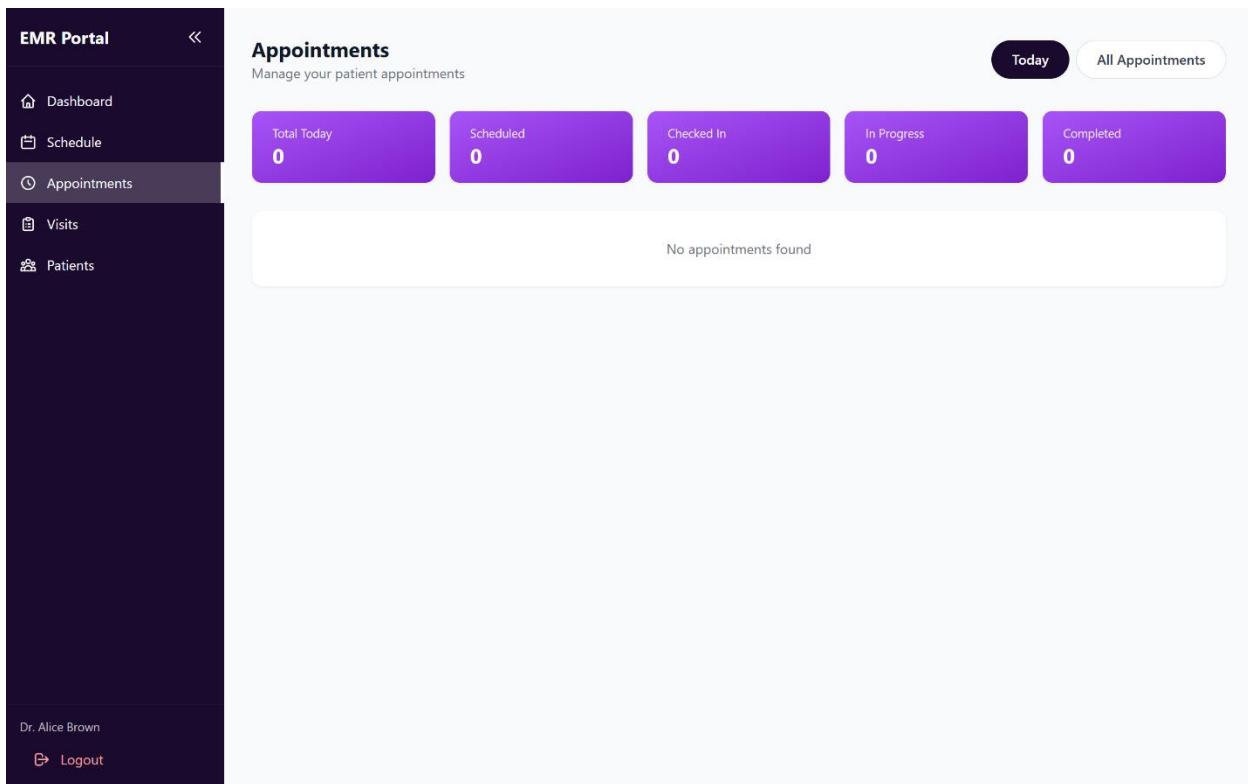
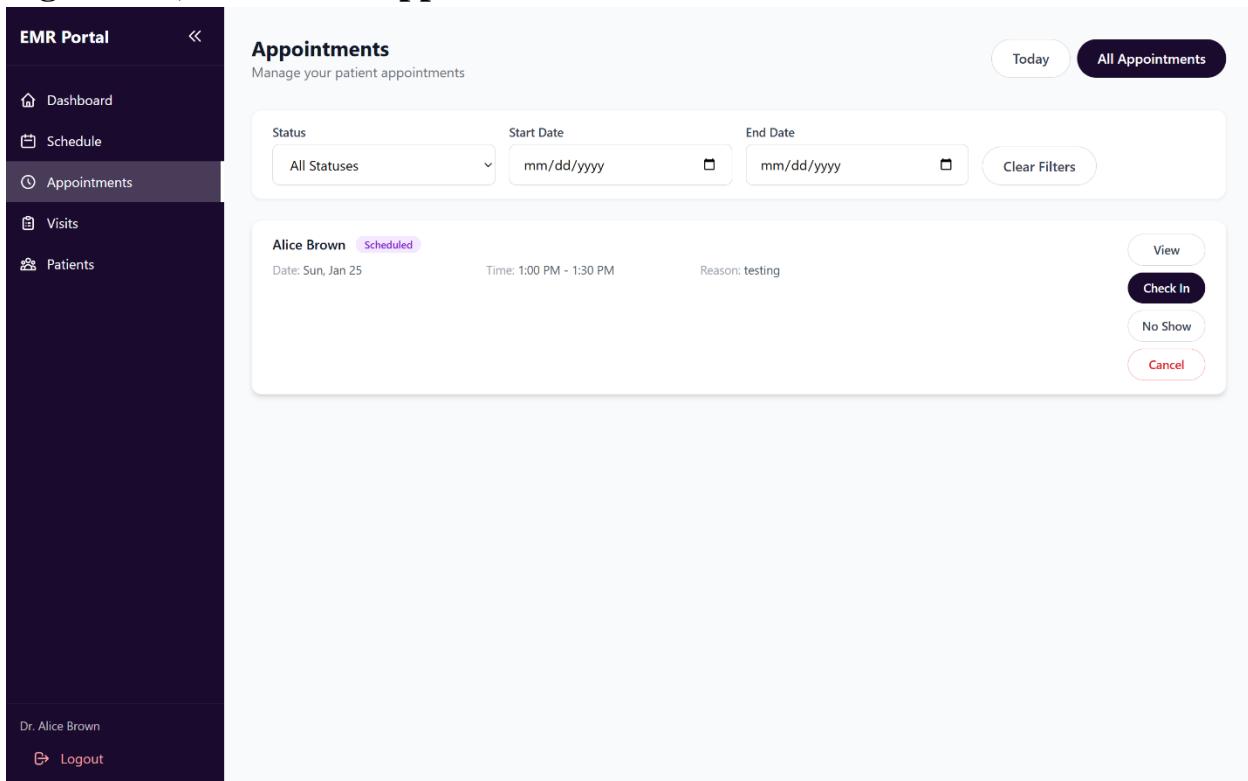


Figure 165, Create Schedule

## Chapter 6 - Implementation and Testing

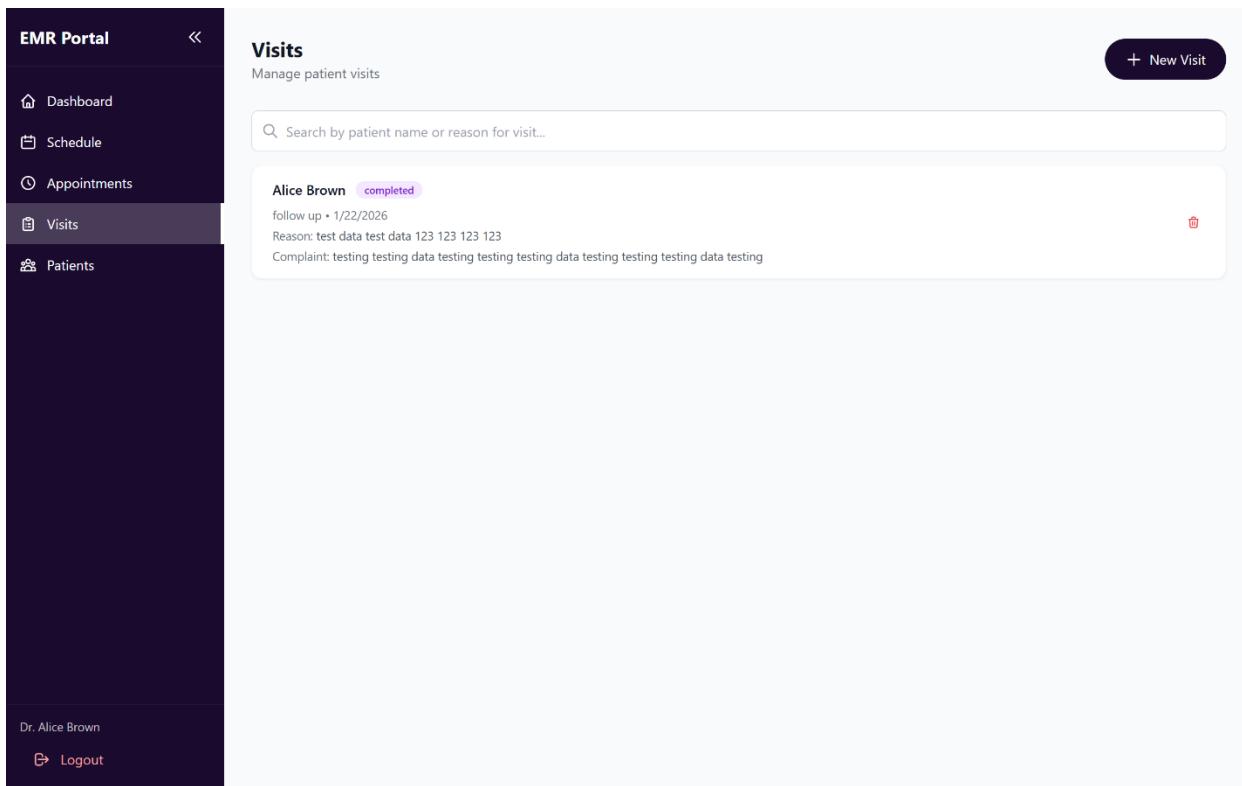


**Figure 166, View Own Appointments**

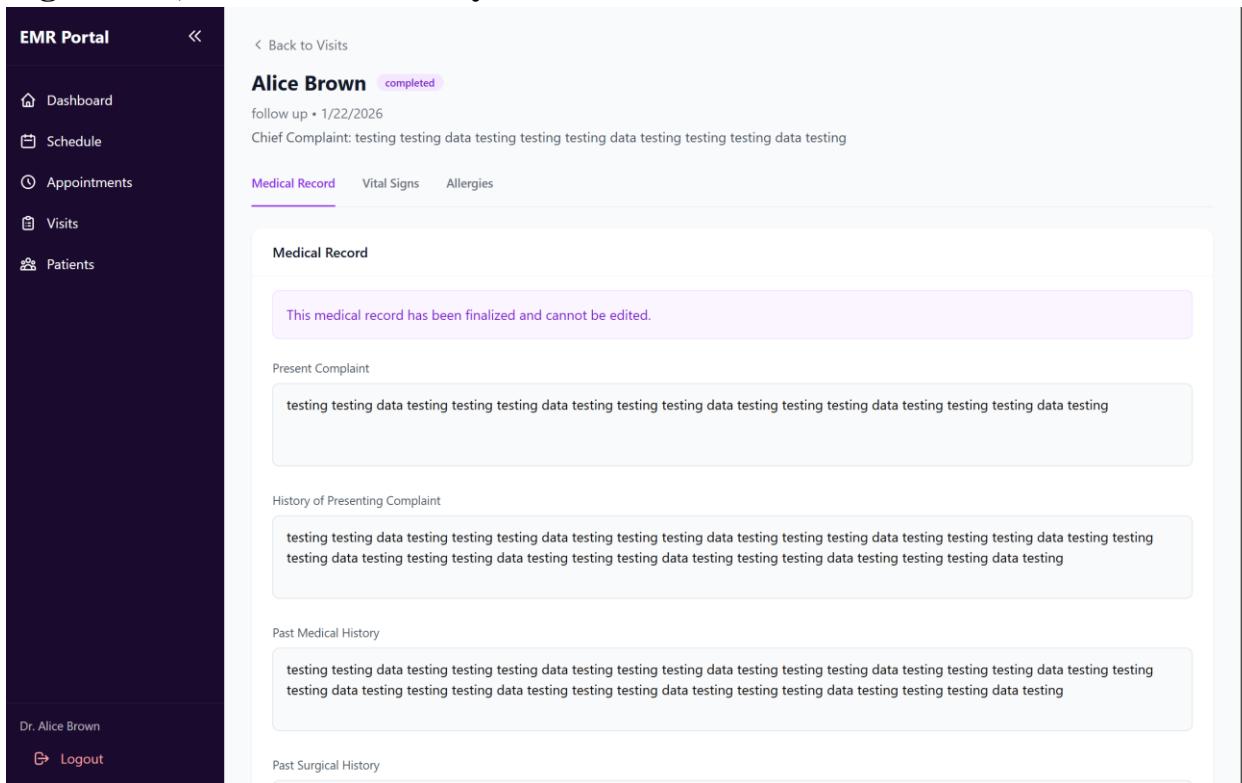


**Figure 167, View All Appointments**

## Chapter 6 - Implementation and Testing



**Figure 168, View Visit History**



**Figure 169, View Medical Record**

## Chapter 6 - Implementation and Testing

The screenshot shows the EMR Portal interface for a patient named Alice Brown. The left sidebar includes links for Dashboard, Schedule, Appointments, Visits, and Patients. The main content area shows a completed visit for Alice Brown on 1/22/2026. The Chief Complaint is listed as "testing testing data testing testing data testing testing data testing". The Vital Signs tab is selected, displaying the following vital signs:

Blood Pressure	Heart Rate	Temperature	Respiratory Rate
50.00 mmHg	90.00 bpm	38.00 °C	120.00 /min

SpO2	Weight	Height	BMI
30.00 %	70.00 kg	170.00 cm	100.00 kg/m <sup>2</sup>

The Vitals History section lists the following measurements with their respective times:

Vital Sign	Value	Time
BMI	100.00 kg/m <sup>2</sup>	12:18:00 PM
Height	170.00 cm	12:17:55 PM
Weight	70.00 kg	12:17:51 PM
SpO2	30.00 %	12:17:45 PM
Respiratory Rate	120.00 /min	12:17:40 PM
Temperature	38.00 °C	12:17:35 PM
Blood Pressure	50.00 mmHg	12:17:30 PM

Figure 170, View Visit's Vital Signs.

The screenshot shows the EMR Portal interface for a patient named Alice Brown. The left sidebar includes links for Dashboard, Schedule, Appointments, Visits, and Patients. The main content area shows a completed visit for Alice Brown on 1/22/2026. The Chief Complaint is listed as "testing testing data testing testing data testing testing data testing". The Allergies tab is selected, displaying the following information:

Pencillin mild drug  
Reaction: testing testing data testing testing data testing testing data testing testing data testing

Figure 171, View Visit's Allergies.

## Chapter 6 - Implementation and Testing

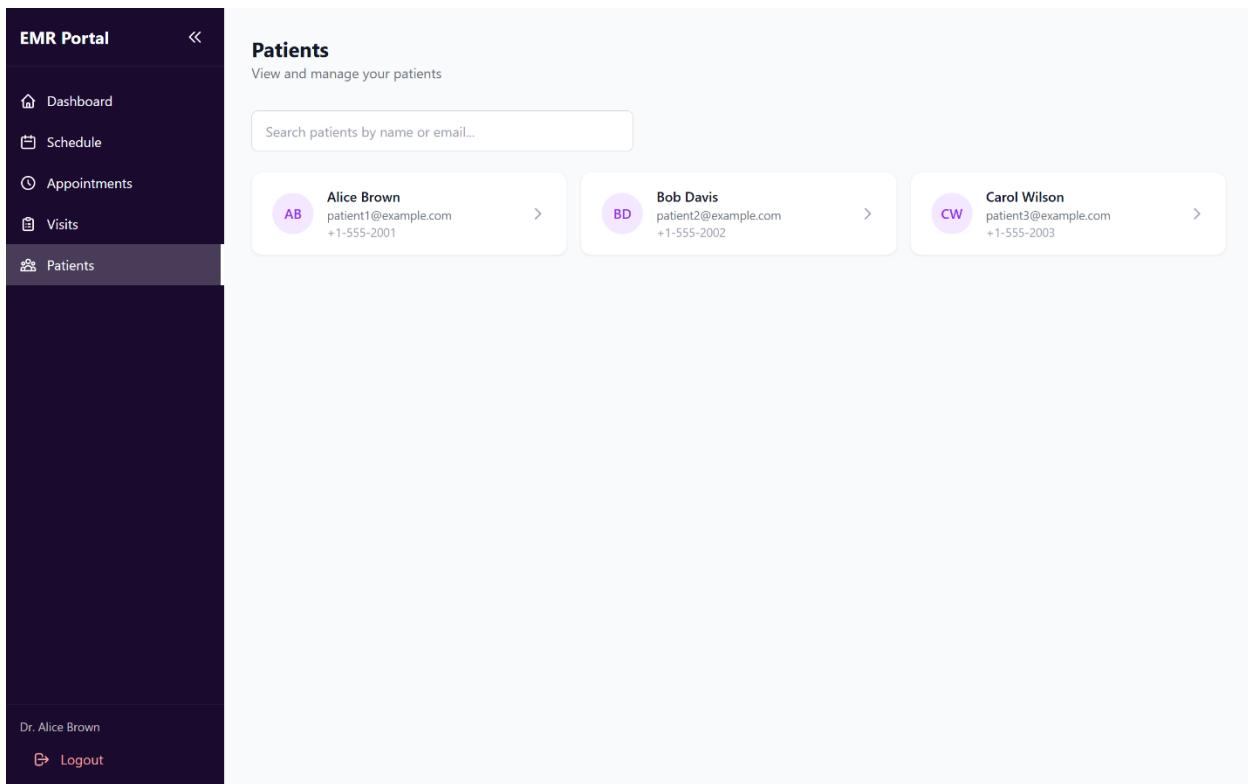


Figure 172, View Patients Lists.

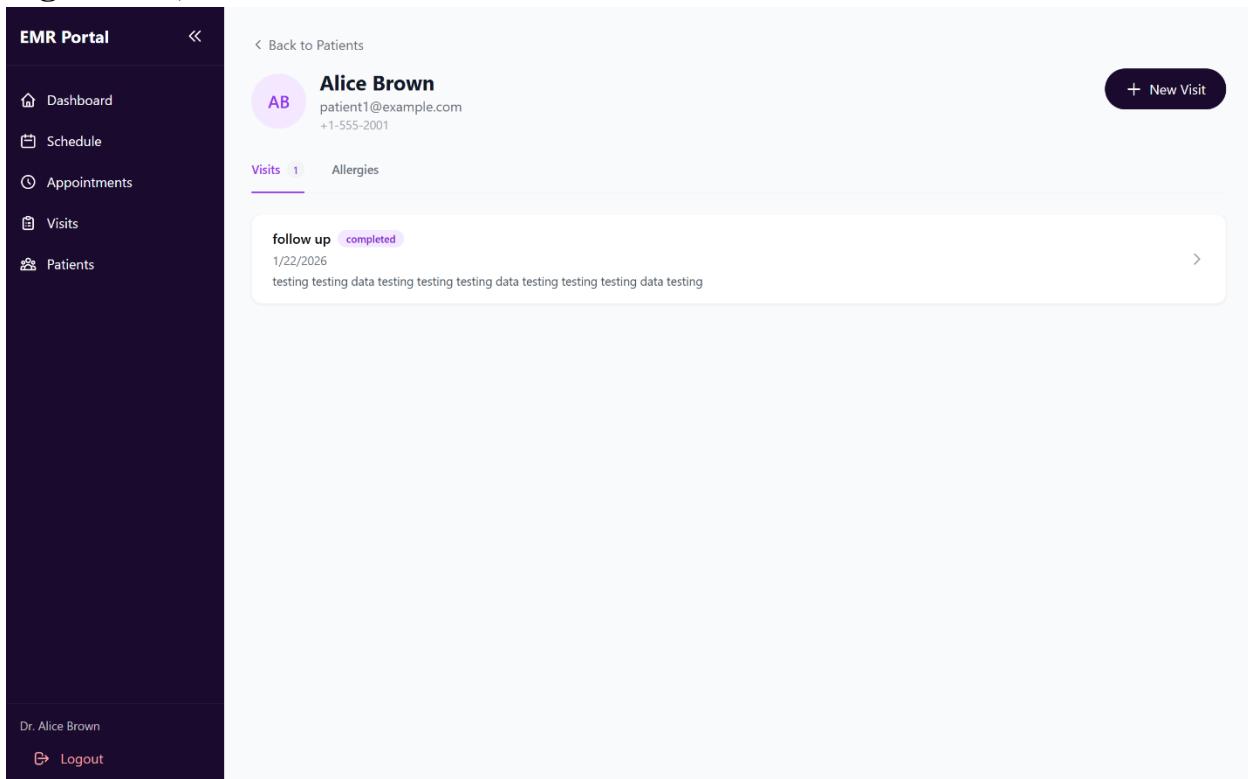


Figure 173, View Patient's Record.

## Chapter 6 - Implementation and Testing

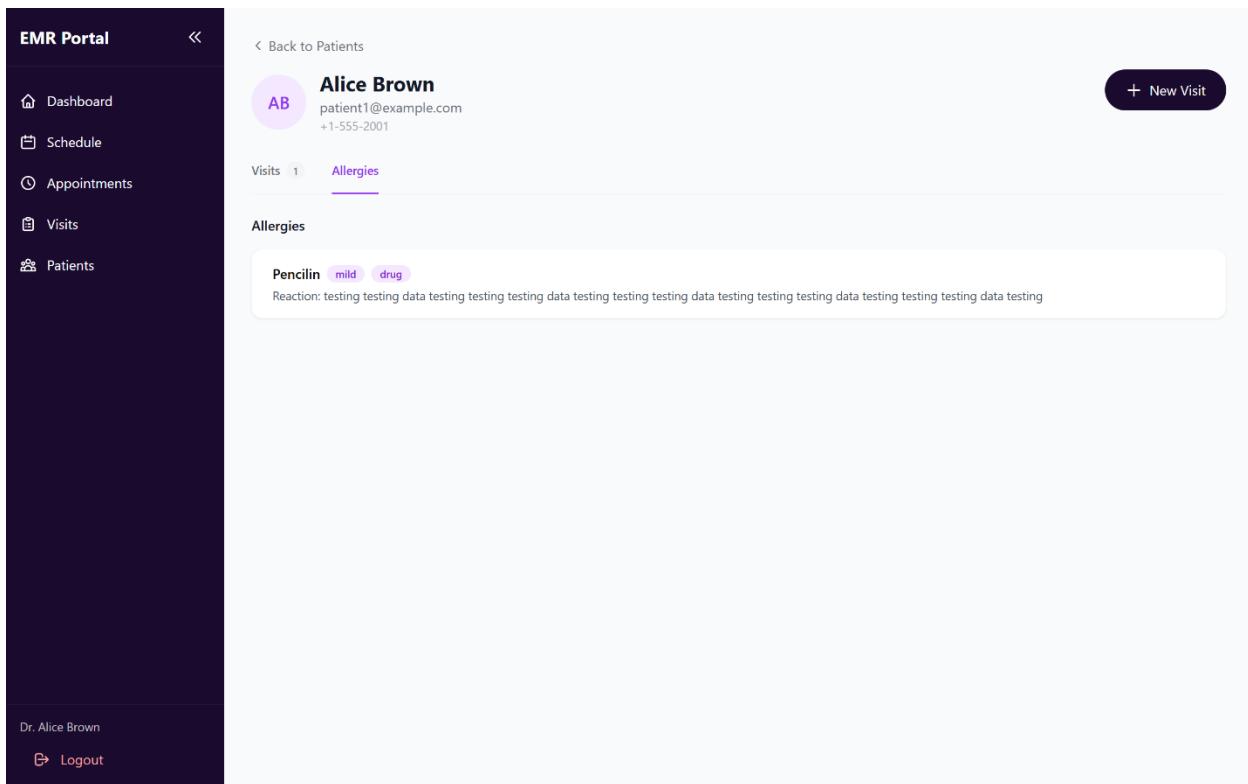


Figure 174, View Patient's Allergies.

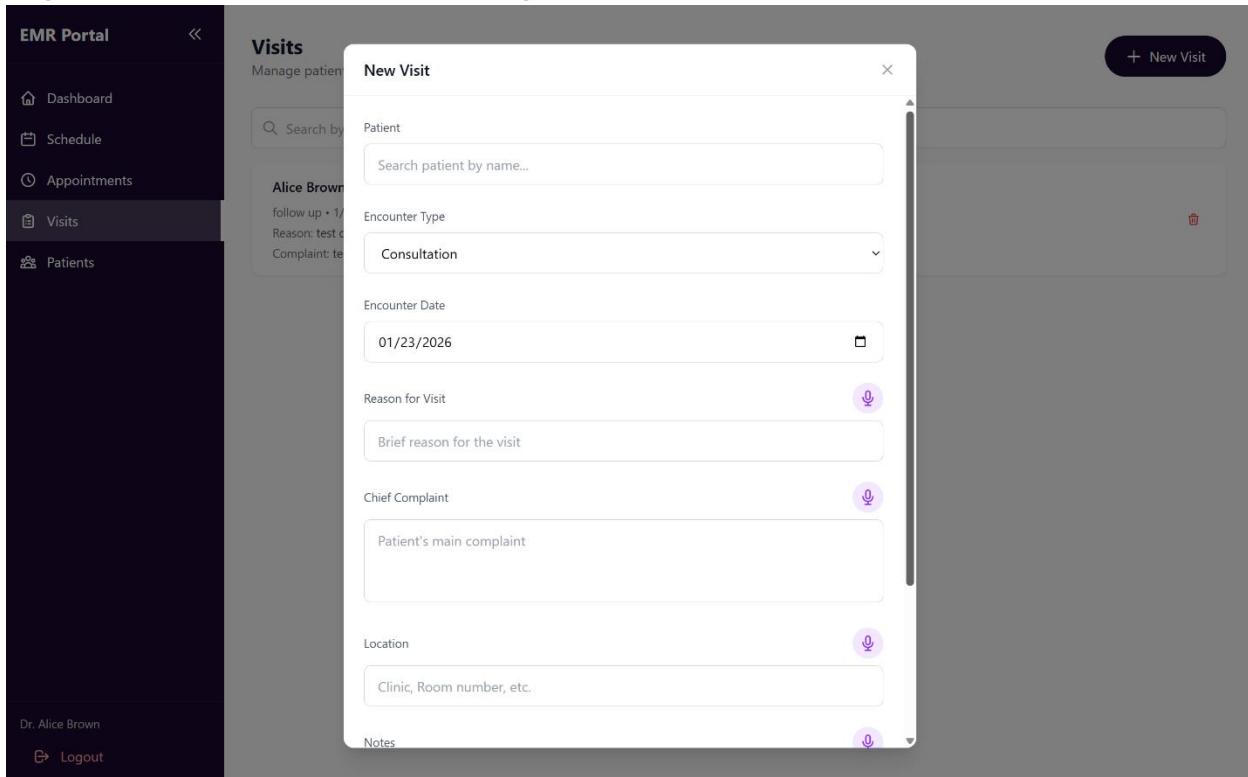


Figure 175, Create New Visit.

## Chapter 6 - Implementation and Testing

The screenshot shows the Admin Dashboard of a medical clinic system. The left sidebar includes links for Admin Portal, Dashboard, Organizations, Doctors, and Logout. The main dashboard displays the following information:

- Total Organizations:** 3 (3 active)
- Total Doctors:** 3 (3 active)
- Quick Actions:** Add Organization (Create a new clinic), Add Doctor (Register a new doctor), Manage All (View all records).
- Recent Organizations:**
  - City Medical Center (CMC001) - Active
  - Community Health Clinic (CHC003) - Active
  - General Hospital (GH002) - Active
- Recent Doctors:**
  - Dr. Sarah Johnson (Pediatrics • dr.johnson@emr.com) - Active
  - Dr. John Smith (Cardiology • dr.smith@emr.com) - Active
  - Dr. Michael Williams (General Practice • dr.williams@emr.com) - Active

Figure 176, View Admin Dashboard And Analytics.

The screenshot shows the Organizations page of the Admin Portal. The left sidebar includes links for Admin Portal, Dashboard, Organizations, Doctors, and Logout. The main content area displays the following:

**Organizations**  
Manage clinics and medical facilities

A search bar is present: Search by name or code...

Name	Code	Email	Phone	Actions
City Medical Center	CMC001	info@citymedical.com	+1-555-0101	
Community Health Clinic	CHC003	hello@communityclinic.com	+1-555-0103	
General Hospital	GH002	contact@generalhospital.com	+1-555-0102	

Figure 177, View Organization List.

## Chapter 6 - Implementation and Testing

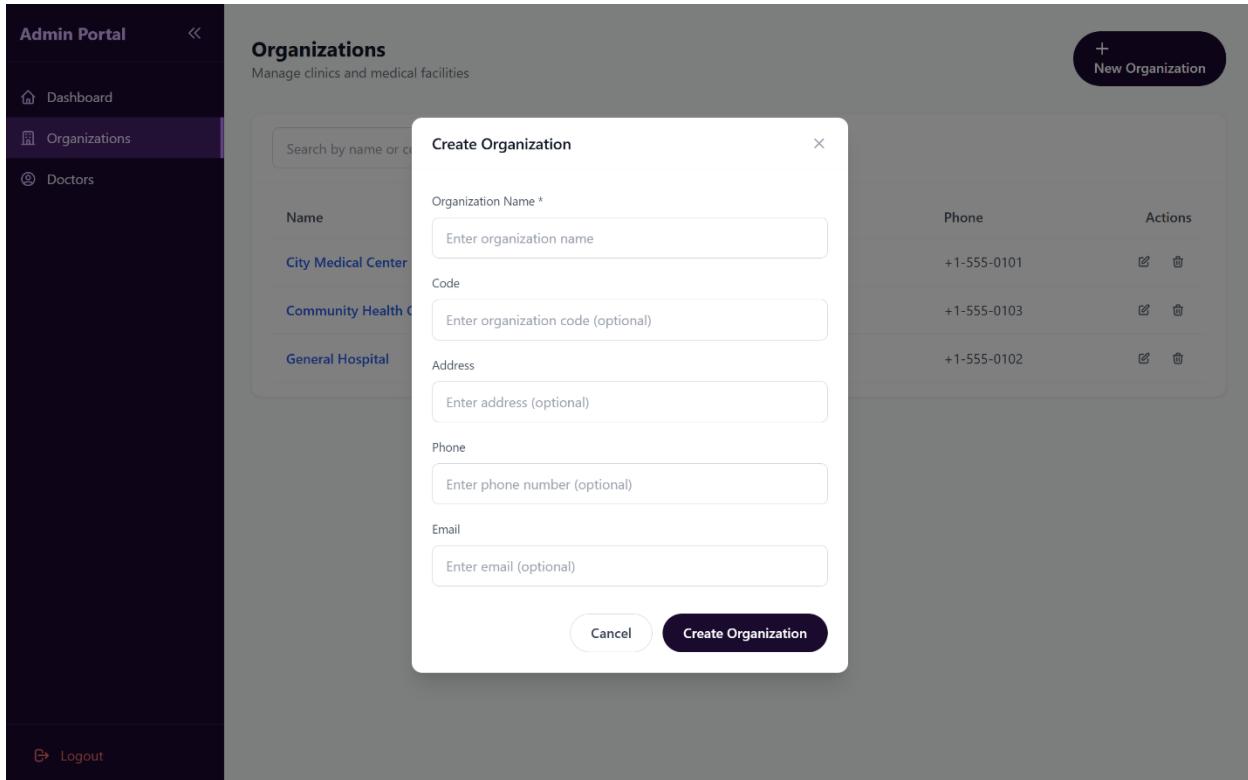


Figure 178, Create New Organization.

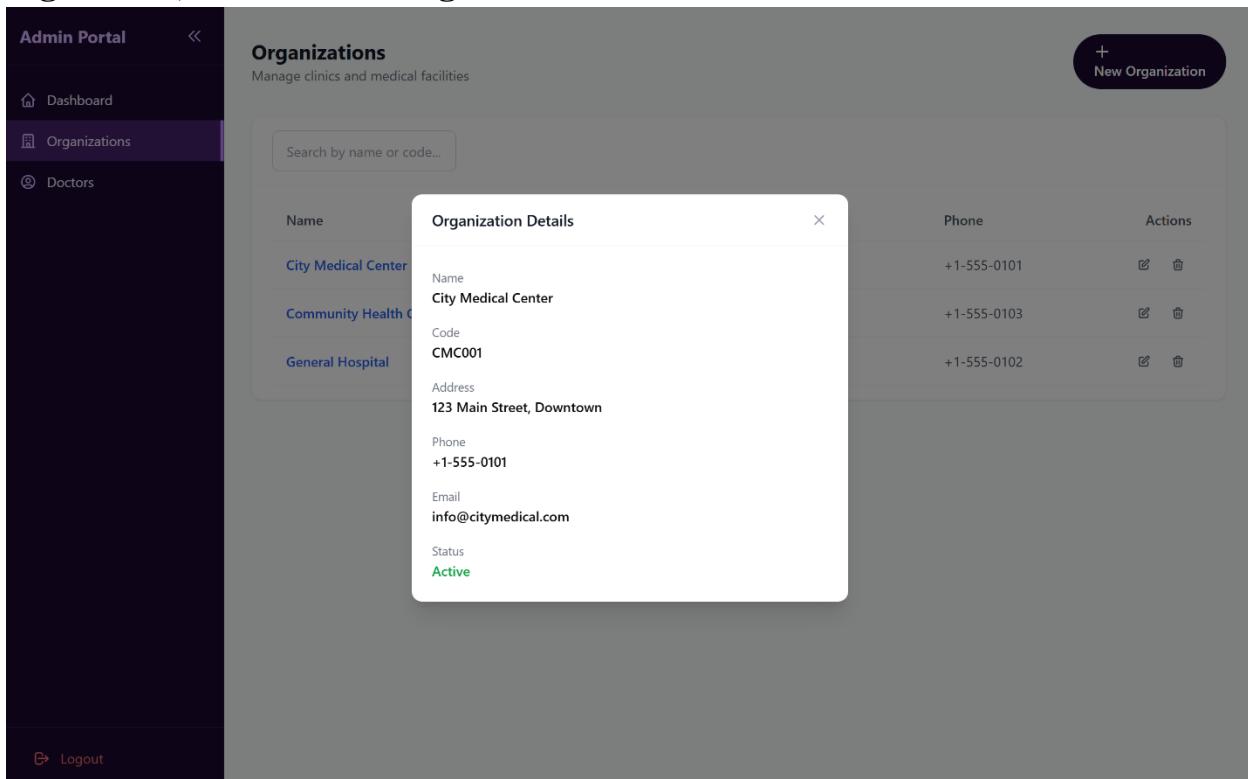


Figure 179, View Organization Details.

## Chapter 6 - Implementation and Testing

The screenshot shows the Admin Portal interface. On the left is a dark sidebar with a purple header containing the text "Admin Portal". Below the header are three menu items: "Dashboard", "Organizations", and "Doctors", with "Doctors" being the active tab. At the bottom of the sidebar is a "Logout" button. The main content area has a white header with the title "Doctors" and a subtitle "Manage doctor accounts and assignments". To the right of the header is a dark blue button with a white plus sign and the text "New Doctor". Below the header is a search bar with the placeholder "Search by name, email, or...". The main content is a table with the following columns: Name, Email, Specialization, Organization, and Actions. The table contains three rows of data:

Name	Email	Specialization	Organization	Actions
Dr. Sarah Johnson	dr.johnson@emr.com	Pediatrics	City Medical Center	
Dr. John Smith	dr.smith@emr.com	Cardiology	City Medical Center	
Dr. Michael Williams	dr.williams@emr.com	General Practice	Community Health Clinic	

Figure 180, View Doctors List.

This screenshot shows the "Create Doctor" dialog box overlaid on the Admin Portal's Doctors list. The dialog has a light gray background and a white header with the title "Create Doctor" and a close button. The form fields are as follows:

- Email \*: A text input field with the placeholder "Enter email address".
- Password \*: A text input field with the placeholder "Enter password (min 6 characters)".
- Name:
  - First Name \*: A text input field with the placeholder "Enter first name".
  - Last Name \*: A text input field with the placeholder "Enter last name".
- Phone Number: A text input field with the placeholder "Enter phone number (optional)".
- Specialization: A text input field with the placeholder "e.g., Cardiology, Pediatrics (optional)".
- License Number: A text input field with the placeholder "Enter license number (optional)".
- Organization \*:
  - A dropdown menu with the placeholder "Select an organization".
  - A list of organizations on the right side of the dialog: "City Medical Center", "City Medical Center", and "Community Health Clinic", each with edit and delete icons.

At the bottom of the dialog are two buttons: "Cancel" and "Create Doctor".

Figure 181, Create New Doctor.

## Chapter 6 - Implementation and Testing

The screenshot shows the Admin Portal interface. On the left, a sidebar has 'Admin Portal' at the top, followed by 'Dashboard', 'Organizations', and 'Doctors' (which is highlighted in purple). At the bottom of the sidebar are 'Logout' and a user icon. The main content area is titled 'Doctors' with the subtitle 'Manage doctor accounts and assignments'. A search bar says 'Search by name, email, or phone'. Below it is a table with columns: Name, Email, Specialization, Organization, and Actions. The table lists three doctors: Dr. Sarah Johnson, Dr. John Smith, and Dr. Michael Williams. Dr. John Smith's row is expanded, showing a modal window titled 'Doctor Details' with fields: Name (Dr. John Smith), Email (dr.smith@emr.com), Phone (+1-555-1001), Specialization (Cardiology), and License Number (MD-12345). The 'Actions' column for each doctor includes edit and delete icons.

**Figure 182, View Doctors Details.**

### 6.4 Test Cases

## Chapter 6 - Implementation and Testing

Test Case ID	Test Name	Purpose	Test Data	Steps	Expected Result	Actual Result
TC-001	Register new user successfully	Verify user registration with email, password hashing, and verification token generation	Email: "john@example.com" Password: "Password123!" First Name: "John" Last Name: "Doe"	1. Check if email exists2. Hash password3. Create user record4. Generate verification token5. Send verification email	User account created successfully with hashed password, verification token generated, and email sent	PASSED
TC-002	Throw ConflictException if email exists	Verify duplicate email prevention	Email: "existing@example.com"	1. Check if email exists2. Throw ConflictException	ConflictException thrown with message "Email already exists"	PASSED
TC-003	Hash password before storing	Verify passwords are securely hashed using bcrypt	Email: "test@example.com" Password: "Password123!"	1. Check email availability2. Hash password using bcrypt3. Verify hash differs from plain password4. Create user with hashed password	Password is hashed and stored securely, hash differs from plain text	PASSED

## Chapter 6 - Implementation and Testing

TC-004	Login successfully with valid credentials	Verify login flow with JWT token generation	Email: "test@example.com" Password: "password123"	1. Find user by email 2. Compare password hash 3. Generate JWT tokens 4. Update last login 5. Return tokens and user data	JWT access and refresh tokens returned with user profile (password excluded)	PASSED
TC-005	Throw UnauthorizedException for invalid email	Verify login fails with non-existent email	Email: "wrong@example.com" Password: "password"	1. Find user by email 2. User not found 3. Throw UnauthorizedException	UnauthorizedException thrown with message "Invalid credentials"	PASSED
TC-006	Throw UnauthorizedException for invalid password	Verify login fails with incorrect password	Email: "test@example.com" Password: "wrongpassword"	1. Find user by email 2. Compare password hash 3. Hash comparison fails 4. Throw UnauthorizedException	UnauthorizedException thrown with message "Invalid credentials"	PASSED

## Chapter 6 - Implementation and Testing

TC-007	Throw UnauthorizedException if email not verified	Verify unverified users cannot login	Email: "test@example.com" isEmailVerified: false	1. Find user by email2. Check email verification status3. Throw UnauthorizedException	UnauthorizedException thrown with message "Please verify your email"	PASSED
TC-008	Throw UnauthorizedException if account not active	Verify suspended accounts cannot login	Email: "test@example.com" accountStatus: "SUSPENDED"	1. Find user by email2. Check account status3. Throw UnauthorizedException	UnauthorizedException thrown with message "Account is not active"	PASSED
TC-009	Verify email successfully	Verify email verification with valid token	Token: "valid-token-123"	1. Find verification token2. Check token expiration3. Update user email verification status4. Mark token as used	User email marked as verified, token marked as used	PASSED

## Chapter 6 - Implementation and Testing

TC-010	Throw BadRequestException for invalid token	Verify invalid tokens are rejected	Token: "invalid-token"	1. Find verification token 2. Token not found 3. Throw BadRequestException	BadRequestException thrown with message "Invalid or expired verification token"	PASSED
TC-011	Throw BadRequestException for expired token	Verify expired verification tokens are rejected	Token: "expired-token" expiresAt: 25 hours ago	1. Find verification token 2. Check token expiration 3. Token is expired 4. Throw BadRequestException	BadRequestException thrown with message "Verification token has expired"	PASSED
TC-012	Throw BadRequestException for already used token	Verify used tokens cannot be reused	Token: "used-token" isUsed: true	1. Find verification token 2. Check if token is used 3. Throw BadRequestException	BadRequestException thrown with message "Token has already been used"	PASSED

## Chapter 6 - Implementation and Testing

TC-013	Resend verification email successfully	Verify verification email can be resent	Email: "test@example.com" isEmailVerified: false	1. Find user by email2. Delete old tokens3. Generate new token4. Send verification email	New verification token generated and email sent successfully	PASSED
TC-014	Throw BadRequestException if user not found	Verify error handling for non-existent user	Email: "notfound@example.com"	1. Find user by email2. User not found3. Throw BadRequestException	BadRequestException thrown with message "User not found"	PASSED
TC-015	Throw BadRequestException if email already verified	Verify verified users cannot request resend	Email: "test@example.com" isEmailVerified: true	1. Find user by email2. Check verification status3. Throw BadRequestException	BadRequestException thrown with message "Email is already verified"	PASSED

## Chapter 6 - Implementation and Testing

TC-016	Send password reset email	Verify password reset email is sent for valid user	Email: "test@example.com"	1. Find user by email2. Delete old reset tokens3. Generate reset token4. Send reset email	Reset token generated and email sent with reset link	PASSED
TC-017	Return generic message if user not found	Verify security by not revealing user existence	Email: "notfound@example.com"	1. Find user by email2. User not found3. Return generic success message	Generic message returned without revealing user doesn't exist	PASSED
TC-018	Reset password successfully	Verify password can be reset using valid reset token	Token: "reset-token-123"New Password: "NewPassword123!"	1. Find reset token2. Check token expiration3. Hash new password4. Update user password5. Mark token as used	Password updated successfully, token marked as used	PASSED

## Chapter 6 - Implementation and Testing

TC-019	Throw BadRequestException for invalid reset token	Verify invalid reset tokens are rejected	Token: "invalid-token"	1. Find reset token2. Token not found3. Throw BadRequestException	BadRequestException thrown with message "Invalid or expired reset token"	PASSED
TC-020	Throw BadRequestException for expired reset token	Verify expired reset tokens are rejected	Token: "expired-token" expiresAt: 2 hours ago	1. Find reset token2. Check token expiration3. Token is expired4. Throw BadRequestException	BadRequestException thrown with message "Reset token has expired"	PASSED
TC-021	Calculate expiration for seconds	Verify expiration calculation for seconds format	Duration: "30s"	1. Parse duration string2. Calculate expiration date3. Return date 30 seconds in future	Expiration date correctly calculated for 30 seconds	PASSED

## Chapter 6 - Implementation and Testing

TC-022	Calculate expiration for minutes	Verify expiration calculation for minutes format	Duration: "15m"	1. Parse duration string 2. Calculate expiration date 3. Return date 15 minutes in future	Expiration date correctly calculated for 15 minutes	PASSED
TC-023	Calculate expiration for hours	Verify expiration calculation for hours format	Duration: "24h"	1. Parse duration string 2. Calculate expiration date 3. Return date 24 hours in future	Expiration date correctly calculated for 24 hours	PASSED
TC-024	Calculate expiration for days	Verify expiration calculation for days format	Duration: "7d"	1. Parse duration string 2. Calculate expiration date 3. Return date 7 days in future	Expiration date correctly calculated for 7 days	PASSED

## Chapter 6 - Implementation and Testing

TC-025	AdminAuthService should be defined	Verify AdminAuthService is properly instantiated	N/A	1. Create service instance2. Check if defined	Service is defined and ready to use	PASSED
TC-026	Have userType set to ADMIN	Verify correct user type configuration for admin	N/A	1. Access userType property2. Verify value is ADMIN	userType property equals UserType.ADMIN	PASSED
TC-027	Extend BaseAuthService	Verify inheritance and method availability for admin	N/A	1. Check for register method2. Check for login method3. Check for verifyEmail method	All BaseAuthService methods are available	PASSED

## Chapter 6 - Implementation and Testing

TC-028	DoctorAuthService should be defined	Verify DoctorAuthService is properly instantiated	N/A	1. Create service instance2. Check if defined	Service is defined and ready to use	PASSED
TC-029	Have userType set to DOCTOR	Verify correct user type configuration for doctor	N/A	1. Access userType property2. Verify value is DOCTOR	userType property equals UserType.DOCTOR	PASSED
TC-030	Extend BaseAuthService	Verify inheritance and method availability for doctor	N/A	1. Check for register method2. Check for login method3. Check for verifyEmail method	All BaseAuthService methods are available	PASSED

## Chapter 6 - Implementation and Testing

TC-031	Register a new patient successfully	Verify patient registration with all required fields	Email: "john@test.com" Password: "Password123!"First Name: "John"Last Name: "Doe"	1. Check email availability2. Hash password3. Create patient record4. Generate verification token5. Send verification email	Patient account created with verification email sent	PASSED
TC-032	Throw ConflictException if email already exists	Verify duplicate email prevention for patients	Email: "existing@test.com"	1. Check if email exists2. Throw ConflictException	ConflictException thrown preventing duplicate registration	PASSED
TC-033	Login successfully with valid credentials	Verify patient can login with correct credentials	Email: "patient@test.com" Password: "password"	1. Find patient by email2. Verify password3. Generate JWT tokens4. Update last login	Access token returned with patient profile	PASSED

## Chapter 6 - Implementation and Testing

TC-034	Throw UnauthorizedException for invalid credentials	Verify login fails with wrong email	Email: "invalid@test.com"	1. Find patient by email 2. Patient not found 3. Throw UnauthorizedException	UnauthorizedException thrown	PASSED
TC-035	Throw UnauthorizedException for wrong password	Verify login fails with incorrect password	Email: "patient@test.com" Password: "wrongpassword"	1. Find patient by email 2. Compare password 3. Comparison fails 4. Throw UnauthorizedException	UnauthorizedException thrown	PASSED
TC-036	Throw UnauthorizedException if email not verified	Verify unverified patients cannot login	Email: "patient@test.com" isEmailVerified: false	1. Find patient 2. Check verification status 3. Throw UnauthorizedException	UnauthorizedException with verification message	PASSED

## Chapter 6 - Implementation and Testing

TC-037	Throw UnauthorizedException if account not active	Verify suspended patients cannot login	Email: "patient@test.com" accountStatus: "SUSPENDED"	1. Find patient2. Check account status3. Throw UnauthorizedException	UnauthorizedException with account status message	PASSED
TC-038	Verify email successfully	Verify patient email verification process	Token: "valid-token"	1. Find token2. Verify token validity3. Update patient email status4. Mark token as used	Email verified successfully	PASSED
TC-039	Throw BadRequestException for invalid token	Verify invalid verification tokens are rejected	Token: "invalid-token"	1. Find token2. Token not found3. Throw BadRequestException	BadRequestException thrown	PASSED

## Chapter 6 - Implementation and Testing

TC-040	Throw BadRequestException for expired token	Verify expired tokens are rejected	Token: "expired-token" expiresAt: past date	1. Find token2. Check expiration3. Throw BadRequestException	BadRequestException with expiration message	PASSED
TC-041	Send password reset email for existing user	Verify password reset flow for patients	Email: "patient@test.com"	1. Find patient2. Delete old tokens3. Generate reset token4. Send email	Reset email sent successfully	PASSED
TC-042	Return success message even for non-existing user	Verify security by not revealing user existence	Email: "nonexistent@test.com"	1. Find patient2. Patient not found3. Return generic message	Generic success message returned	PASSED

## Chapter 6 - Implementation and Testing

TC-043	Reset password successfully	Verify password reset with valid token	Token: "valid-token"New Password: "NewPassword123!"	1. Find token2. Verify validity3. Hash new password4. Update password5. Mark token used	Password reset successfully	PASSED
TC-044	Throw BadRequestException for invalid token	Verify invalid reset tokens are rejected	Token: "invalid-token"	1. Find token2. Token not found3. Throw BadRequestException	BadRequestException thrown	PASSED
TC-045	Service should be defined	Verify PatientAuthService is properly configured	N/A	1. Check service definition2. Verify instantiation	Service properly defined and configured	PASSED

## Chapter 6 - Implementation and Testing

TC-046	Use PatientRepository	Verify repository injection	N/A	1. Check repository injection 2. Verify availability	PatientRepository properly injected	PASSED
TC-047	Use VerificationTokenRepository	Verify token repository injection	N/A	1. Check repository injection 2. Verify availability	VerificationTokenRepository properly injected	PASSED
TC-048	Use EmailService	Verify email service injection	N/A	1. Check service injection 2. Verify availability	EmailService properly injected	PASSED

## Chapter 6 - Implementation and Testing

TC-049	Use JwtService	Verify JWT service injection	N/A	1. Check service injection 2. Verify availability	JwtService properly injected	PASSED
TC-050	Use ConfigService	Verify config service injection	N/A	1. Check service injection 2. Verify availability	ConfigService properly injected	PASSED
TC-051	Create an allergy successfully	Verify allergy record creation with patient validation	Patient ID: "patient-123" Allergen: "Penicillin" Type: "DRUG" Severity : "SEVERE"	1. Verify patient exists 2. Check for duplicate 3. Create allergy record	Allergy record created with all details	PASSED

## Chapter 6 - Implementation and Testing

TC-052	Throw NotFoundException if patient not found	Verify patient existence validation	Patient ID: "non-existent"	1. Find patient2. Patient not found3. Throw NotFoundException	NotFoundException thrown	PASSED
TC-053	Throw ConflictException if duplicate allergy exists	Verify duplicate allergen prevention	Patient ID: "patient-123" Allergen: "Penicillin" (exists)	1. Check for duplicate allergen2. Duplicate found3. Throw ConflictException	ConflictException with duplicate message	PASSED
TC-054	Return allergy by id	Verify allergy retrieval by ID	Allergy ID: "allergy-123"	1. Find allergy by ID2. Return allergy record	Allergy record returned with all fields	PASSED

## Chapter 6 - Implementation and Testing

TC-055	Throw NotFoundException if allergy not found	Verify error handling for missing allergy	Allergy ID: "non-existent"	1. Find allergy2. Not found3. Throw NotFoundException	NotFoundException thrown	PASSED
TC-056	Return allergies for a patient	Verify patient allergy list retrieval	Patient ID: "patient-123"	1. Query allergies by patient2. Return list	List of patient allergies returned	PASSED
TC-057	Return active allergies by default	Verify default filtering to active allergies only	Patient ID: "patient-123" activeOnly: true (default)	1. Query allergies2. Filter by isActive = true3. Return active only	Only active allergies returned	PASSED

## Chapter 6 - Implementation and Testing

TC-058	Return all allergies when activeOnly is false	Verify ability to retrieve all allergies including inactive	Patient ID: "patient-123"activeOnly: false	1. Query allergies2. No active filter3. Return all	All allergies (active and inactive) returned	PASSED
TC-059	Return paginated allergies	Verify pagination support for allergies	Page: 1Limit: 10	1. Query with pagination2. Apply limit and offset3. Return page	Paginated allergy list returned	PASSED
TC-060	Filter by patient id when provided	Verify patient-specific filtering	Patient ID: "patient-123"	1. Apply patient filter2. Query allergies3. Return filtered results	Only specified patient's allergies returned	PASSED

## Chapter 6 - Implementation and Testing

TC-061	Update allergy successfully	Verify allergy update functionality	Allergy ID: "allergy-123"New Severity: "MODERATE"	1. Find allergy 2. Update fields 3. Save changes	Allergy updated with new severity	PASSED
TC-062	Throw NotFoundException if allergy not found	Verify update error handling	Allergy ID: "non-existent"	1. Find allergy 2. Not found 3. Throw NotFoundException	NotFoundException thrown	PASSED
TC-063	Check for duplicates when allergen is changed	Verify duplicate validation on update	Allergy ID: "allergy-123"New Allergen: "Aspirin"	1. Check if new allergen exists 2. Proceed if unique	Duplicate check performed	PASSED

## Chapter 6 - Implementation and Testing

TC-064	Throw ConflictException if new allergen is duplicate	Verify duplicate prevention on update	New Allergen: "Penicillin" (exists)	1. Check for duplicate 2. Duplicate found 3. Throw ConflictException	ConflictException thrown	PASSED
TC-065	Not check duplicates if allergen unchanged	Verify optimization when allergen not changed	Allergy ID: "allergy-123" Allergen: unchanged	1. Compare allergen values 2. Skip duplicate check if same	Duplicate check skipped for efficiency	PASSED
TC-066	Not check duplicates if allergen is same value	Verify duplicate check optimization	Allergen: "Penicillin" → "Penicillin"	1. Compare values 2. Skip check if identical	Duplicate check skipped	PASSED

## Chapter 6 - Implementation and Testing

TC-067	Soft delete allergy	Verify soft delete functionality with timestamp	Allergy ID: "allergy-123"	1. Find allergy 2. Set deletedAt timestamp 3. Save	Allergy soft deleted (deletedAt set)	PASSED
TC-068	Throw NotFoundException if allergy not found	Verify delete error handling	Allergy ID: "non-existent"	1. Find allergy 2. Not found 3. Throw NotFoundException	NotFoundException thrown	PASSED
TC-069	Create an encounter successfully	Verify medical encounter creation with PLANNED status	Patient ID: "patient-123" Doctor ID: "doctor-456" Type: "CONSULTATION"	1. Verify patient exists 2. Create encounter with PLANNED status 3. Return created encounter	Encounter created with PLANNED status	PASSED

## Chapter 6 - Implementation and Testing

TC-070	Throw NotFoundException if patient not found	Verify patient validation for encounter	Patient ID: "non-existent"	1. Find patient2. Patient not found3. Throw NotFoundException	NotFoundException thrown	PASSED
TC-071	Return existing encounter if idempotency key matches	Verify idempotency key prevents duplicate encounters	Idempotency Key: "unique-key-123" (already used)	1. Check for existing encounter with key2. Return existing encounter	Existing encounter returned without creating duplicate	PASSED
TC-072	Return encounter by id	Verify encounter retrieval by ID	Encounter ID: "encounter-123"	1. Find encounter by ID2. Return encounter	Encounter returned with all details	PASSED

## Chapter 6 - Implementation and Testing

TC-073	Throw NotFoundException if encounter not found	Verify error handling for missing encounter	Encounter ID: "non-existent"	1. Find encounter2. Not found3. Throw NotFoundException	NotFoundException thrown	PASSED
TC-074	Start a planned encounter	Verify status transition from PLANNED to IN_PROGRESS	Encounter ID: "encounter- 123>Status: "PLANNED"	1. Find encounter2. Verify status is PLANNED3. Update status to IN_PROGRESS 4. Set start time	Encounter status changed to IN_PROGRESS with start time recorded	PASSED
TC-075	Throw BadRequestExce ption if encounter not planned	Verify status validation before starting	Encounter ID: "encounter- 123>Status: "COMPLETED"	1. Find encounter2. Check status3. Throw BadRequestExcep tion	BadRequestExcep tion thrown	PASSED

## Chapter 6 - Implementation and Testing

TC-076	Complete an in-progress encounter	Verify status transition from IN_PROGRESS to COMPLETED	Encounter ID: "encounter-123>Status: "IN_PROGRESS "	1. Find encounter2. Verify status is IN_PROGRESS 3. Update status to COMPLETED4. Set end time	Encounter status changed to COMPLETED with end time recorded	PASSED
TC-077	Throw BadRequestException if encounter not in progress	Verify status validation before completing	Encounter ID: "encounter-123>Status: "PLANNED"	1. Find encounter2. Check status3. Throw BadRequestException	BadRequestException thrown	PASSED
TC-078	Cancel a planned encounter	Verify cancellation of PLANNED encounter	Encounter ID: "encounter-123>Status: "PLANNED"	1. Find encounter2. Verify status is not COMPLETED3. Update status to CANCELLED	Encounter status changed to CANCELLED	PASSED

## Chapter 6 - Implementation and Testing

TC-079	Throw BadRequestException if encounter already completed	Verify completed encounters cannot be cancelled	Encounter ID: "encounter-123>Status: "COMPLETED"	1. Find encounter2. Check status3. Throw BadRequestException	BadRequestException thrown	PASSED
TC-080	Cancel an in-progress encounter	Verify cancellation of IN_PROGRESS encounter	Encounter ID: "encounter-123>Status: "IN_PROGRESS "	1. Find encounter2. Update status to CANCELLED	Encounter status changed to CANCELLED	PASSED
TC-081	Update encounter successfully	Verify encounter update functionality	Encounter ID: "encounter-123"New Reason: "Follow-up"	1. Find encounter2. Verify access3. Update fields	Encounter updated successfully	PASSED

## Chapter 6 - Implementation and Testing

TC-082	Throw ForbiddenException if not the assigned doctor	Verify access control for encounter updates	Encounter ID: "encounter-123"Assigned Doctor: "doctor-456"Requesting User: "doctor-789"	1. Find encounter2. Check if user is assigned doctor3. Throw ForbiddenException	ForbiddenException thrown with message "You can only update your own encounters"	PASSED
TC-083	Throw BadRequestException if encounter completed	Verify completed encounters cannot be updated	Encounter ID: "encounter-123>Status: "COMPLETED"	1. Find encounter2. Check status3. Throw BadRequestException	BadRequestException thrown	PASSED
TC-084	Throw BadRequestException if encounter cancelled	Verify cancelled encounters cannot be updated	Encounter ID: "encounter-123>Status: "CANCELLED"	1. Find encounter2. Check status3. Throw BadRequestException	BadRequestException thrown	PASSED

## Chapter 6 - Implementation and Testing

TC-085	Allow admin to update any encounter	Verify admin override for access control	Encounter ID: "encounter-123"User Role: "ADMIN"	1. Find encounter2. Check if user is admin3. Allow update	Admin can update any encounter	PASSED
TC-086	Soft delete encounter	Verify soft delete functionality	Encounter ID: "encounter-123"	1. Find encounter2. Set deletedAt timestamp	Encounter soft deleted	PASSED
TC-087	Throw ForbiddenException if not the assigned doctor	Verify access control for encounter deletion	Encounter ID: "encounter-123"Requesting User: "doctor-789"	1. Find encounter2. Check access3. Throw ForbiddenException	ForbiddenException thrown	PASSED

## Chapter 6 - Implementation and Testing

TC-088	Allow admin to delete any encounter	Verify admin override for deletion	Encounter ID: "encounter-123"User Role: "ADMIN"	1. Find encounter2. Check if user is admin3. Allow deletion	Admin can delete any encounter	PASSED
TC-089	Return paginated encounters	Verify pagination support for encounters	Page: 1Limit: 10	1. Query encounters2. Apply pagination3. Return page	Paginated encounter list returned	PASSED
TC-090	Apply filters when provided	Verify filtering by various criteria	Filters: status, patient, doctor	1. Apply filters2. Query encounters3. Return filtered results	Filtered encounters returned	PASSED

## Chapter 6 - Implementation and Testing

TC-091	Return encounters for patient	Verify patient-specific encounter list	Patient ID: "patient-123"	1. Query encounters by patient2. Return list	Patient's encounters returned	PASSED
TC-092	Return active encounters for doctor	Verify doctor's active encounter list	Doctor ID: "doctor-456"	1. Query active encounters by doctor2. Return list	Doctor's active encounters returned	PASSED
TC-093	Not check idempotency if key not provided	Verify optional idempotency key	No idempotency key	1. Check for idempotency key2. Skip check if not provided3. Create encounter	Encounter created without idempotency check	PASSED

## Chapter 6 - Implementation and Testing

TC-094	Handle encounter status transitions correctly	Verify state machine transitions	Various status combinations	1. Test valid transitions 2. Reject invalid transitions	State machine works correctly	PASSED
TC-095	Validate encounter data integrity	Verify data validation rules	Various encounter data	1. Validate required fields 2. Validate data types 3. Validate relationships	Data integrity maintained	PASSED
TC-096	Create a medical record successfully	Verify medical record creation linked to encounter	Patient ID: "patient-123" Encounter ID: "encounter-456" Record Data: { ... }	1. Verify patient exists 2. Verify encounter exists 3. Check for existing record 4. Create medical record	Medical record created successfully	PASSED

## Chapter 6 - Implementation and Testing

TC-097	Throw NotFoundException if patient not found	Verify patient validation for medical record	Patient ID: "non-existent"	1. Find patient2. Patient not found3. Throw NotFoundException	NotFoundException thrown	PASSED
TC-098	Throw NotFoundException if encounter not found	Verify encounter validation	Encounter ID: "non-existent"	1. Find encounter2. Encounter not found3. Throw NotFoundException	NotFoundException thrown	PASSED
TC-099	Throw BadRequestException if record already exists	Verify duplicate prevention for encounter records	Encounter ID: "encounter-456" (has record)	1. Check for existing record2. Record found3. Throw BadRequestException	BadRequestException thrown	PASSED

## Chapter 6 - Implementation and Testing

TC-100	Return medical record by id	Verify medical record retrieval	Record ID: "record-123"	1. Find record by ID2. Return record	Medical record returned	PASSED
TC-101	Throw NotFoundException if record not found	Verify error handling for missing records	Record ID: "non-existent"	1. Find record2. Not found3. Throw NotFoundException	NotFoundException thrown	PASSED
TC-102	Update medical record successfully	Verify medical record update	Record ID: "record-123"New Data: {...}	1. Find record2. Check access3. Check if finalized4. Update record	Medical record updated	PASSED

## Chapter 6 - Implementation and Testing

TC-103	Throw ForbiddenException if not the assigned doctor	Verify access control for updates	Record ID: "record-123"Requesting User: "doctor-789"	1. Find record2. Check access3. Throw ForbiddenException	ForbiddenException thrown	PASSED
TC-104	Throw BadRequestException if record is finalized	Verify finalized records cannot be updated	Record ID: "record-123"isFinalized: true	1. Find record2. Check if finalized3. Throw BadRequestException	BadRequestException thrown	PASSED
TC-105	Finalize medical record successfully	Verify record finalization	Record ID: "record-123"	1. Find record2. Check if already finalized3. Set isFinalized to true	Medical record finalized	PASSED

## Chapter 6 - Implementation and Testing

TC-106	Throw BadRequestException if already finalized	Verify records cannot be re-finalized	Record ID: "record-123" isFinalized: true	1. Find record2. Check if finalized3. Throw BadRequestException	BadRequestException thrown	PASSED
TC-107	Soft delete medical record	Verify soft delete functionality	Record ID: "record-123"	1. Find record2. Check if finalized3. Set deletedAt timestamp	Medical record soft deleted	PASSED
TC-108	Throw BadRequestException if record is finalized	Verify finalized records cannot be deleted	Record ID: "record-123" isFinalized: true	1. Find record2. Check if finalized3. Throw BadRequestException	BadRequestException thrown	PASSED

## Chapter 6 - Implementation and Testing

TC-109	Create a vital sign successfully	Verify vital signs recording for patient	Patient ID: "patient-123"Type: "BLOOD_PRESSURE"Value: "120/80"	1. Verify patient exists2. Create vital sign record	Vital sign recorded successfully	PASSED
TC-110	Throw NotFoundException if patient not found	Verify patient validation for vital signs	Patient ID: "non-existent"	1. Find patient2. Patient not found3. Throw NotFoundException	NotFoundException thrown	PASSED
TC-111	Create multiple vital signs successfully	Verify bulk creation support	Patient ID: "patient-123"Vital Signs: [...], [...]	1. Verify patient exists2. Create multiple vital signs	Multiple vital signs created	PASSED

## Chapter 6 - Implementation and Testing

TC-112	Throw NotFoundException if patient not found	Verify bulk creation validation	Patient ID: "non-existent"	1. Find patient2. Patient not found3. Throw NotFoundException	NotFoundException thrown	PASSED
TC-113	Return vital sign by id	Verify vital sign retrieval	Vital Sign ID: "vs-123"	1. Find vital sign by ID2. Return vital sign	Vital sign returned	PASSED
TC-114	Throw NotFoundException if vital sign not found	Verify error handling for missing vital signs	Vital Sign ID: "non-existent"	1. Find vital sign2. Not found3. Throw NotFoundException	NotFoundException thrown	PASSED

## Chapter 6 - Implementation and Testing

TC-115	Return vital signs for a patient	Verify patient's vital sign history	Patient ID: "patient-123"	1. Query vital signs by patient2. Return list	Patient's vital signs returned	PASSED
TC-116	Return vital signs for an encounter	Verify encounter-specific vital signs	Encounter ID: "encounter-456"	1. Query vital signs by encounter2. Return list	Encounter's vital signs returned	PASSED
TC-117	Return latest vital sign of a type	Verify retrieval of most recent value by type	Patient ID: "patient-123"Type: "BLOOD_PRESSURE"	1. Query vital signs by patient and type2. Order by date descending3. Return first	Latest vital sign of type returned	PASSED

## Chapter 6 - Implementation and Testing

TC-118	Return null if no vital sign found	Verify handling of no data scenario	Patient ID: "patient-123" Type: "TEMPERATURE"	1. Query vital signs 2. No results found 3. Return null	Null returned	PASSED
TC-119	Return latest vital signs of all types	Verify retrieval of latest of each vital sign type	Patient ID: "patient-123"	1. Query all vital sign types 2. Get latest of each type 3. Return map	Latest vital signs of all types returned	PASSED
TC-120	Soft delete vital sign	Verify soft delete functionality	Vital Sign ID: "vs-123"	1. Find vital sign 2. Set deletedAt timestamp	Vital sign soft deleted	PASSED

## Chapter 6 - Implementation and Testing

TC-121	Throw NotFoundException if vital sign not found	Verify delete error handling	Vital Sign ID: "non-existent"	1. Find vital sign 2. Not found 3. Throw NotFoundException	NotFoundException thrown	PASSED
TC-122	Validate vital sign data ranges	Verify data integrity validation	Various vital sign values	1. Validate value ranges 2. Validate data types 3. Reject invalid values	Data validation works correctly	PASSED
TC-123	Create a past medical procedure successfully	Verify past procedure documentation	Patient ID: "patient-123" Procedure: "Appendectomy" Date: "2020-01-15"	1. Verify patient exists 2. Create procedure record	Past medical procedure documented	PASSED

## Chapter 6 - Implementation and Testing

TC-124	Throw NotFoundException if patient not found	Verify patient validation for procedures	Patient ID: "non-existent"	1. Find patient2. Patient not found3. Throw NotFoundException	NotFoundException thrown	PASSED
TC-125	Create procedure with minimal data	Verify support for optional fields	Patient ID: "patient-123"Procedure: "Surgery"	1. Verify patient exists2. Create with minimal data	Procedure created with optional fields null	PASSED
TC-126	Return procedure by id	Verify procedure retrieval	Procedure ID: "proc-123"	1. Find procedure by ID2. Return procedure	Procedure returned	PASSED

## Chapter 6 - Implementation and Testing

TC-127	Throw NotFoundException if procedure not found	Verify error handling for missing procedures	Procedure ID: "non-existent"	1. Find procedure 2. Not found 3. Throw NotFoundException	NotFoundException thrown	PASSED
TC-128	Return all procedures for a patient	Verify patient's procedure history	Patient ID: "patient-123"	1. Query procedures by patient 2. Return list	Patient's procedures returned	PASSED
TC-129	Return empty array if no procedures found	Verify handling of no data	Patient ID: "patient-456"	1. Query procedures 2. No results 3. Return empty array	Empty array returned	PASSED

## Chapter 6 - Implementation and Testing

TC-130	Return paginated procedures	Verify pagination support	Page: 1Limit: 10	1. Query with pagination 2. Apply limit and offset 3. Return page	Paginated procedure list returned	PASSED
TC-131	Filter by patient id when provided	Verify patient-specific filtering	Patient ID: "patient-123"	1. Apply patient filter 2. Query procedures 3. Return filtered results	Only specified patient's procedures returned	PASSED
TC-132	Update procedure successfully	Verify procedure update	Procedure ID: "proc-123"New Data: {...}	1. Find procedure 2. Update fields 3. Save	Procedure updated	PASSED

## Chapter 6 - Implementation and Testing

TC-133	Throw NotFoundException if procedure not found	Verify update error handling	Procedure ID: "non-existent"	1. Find procedure 2. Not found 3. Throw NotFoundException	NotFoundException thrown	PASSED
TC-134	Update with partial data	Verify partial update support	Procedure ID: "proc-123"Partial Data: { notes: "..."}	1. Find procedure 2. Update only provided fields 3. Save	Procedure partially updated	PASSED
TC-135	Soft delete procedure successfully	Verify soft delete functionality	Procedure ID: "proc-123"	1. Find procedure 2. Set deletedAt timestamp	Procedure soft deleted	PASSED

## Chapter 6 - Implementation and Testing

TC-136	Throw NotFoundException if procedure not found	Verify delete error handling	Procedure ID: "non-existent"	1. Find procedure 2. Not found 3. Throw NotFoundException	NotFoundException thrown	PASSED
TC-137	Create a review of systems successfully	Verify system review documentation	Patient ID: "patient-123"System: "CARDIOVASCULAR"Finding: "Normal"	1. Verify patient exists 2. Create review record	Review of systems documented	PASSED
TC-138	Throw NotFoundException if patient not found	Verify patient validation for reviews	Patient ID: "non-existent"	1. Find patient 2. Patient not found 3. Throw NotFoundException	NotFoundException thrown	PASSED

## Chapter 6 - Implementation and Testing

TC-139	Create review without encounter id	Verify support for standalone reviews	Patient ID: "patient-123"System: "RESPIRATOR Y"No Encounter ID	1. Verify patient exists2. Create review without encounter	Standalone review created	PASSED
TC-140	Create review with minimal data	Verify support for optional fields	Patient ID: "patient-123"System: "NEUROLOGICAL"	1. Verify patient exists2. Create with minimal data	Review created with optional fields null	PASSED
TC-141	Create multiple reviews successfully	Verify bulk creation support	Patient ID: "patient-123"Reviews: [...], [...]	1. Verify patient exists2. Create multiple reviews	Multiple reviews created	PASSED

## Chapter 6 - Implementation and Testing

TC-142	Throw NotFoundException if patient not found	Verify bulk creation validation	Patient ID: "non-existent"	1. Find patient2. Patient not found3. Throw NotFoundException	NotFoundException thrown	PASSED
TC-143	Create bulk reviews without encounter id	Verify bulk standalone reviews	Patient ID: "patient-123"Reviews: [{...}, {...}]No Encounter IDs	1. Verify patient exists2. Create bulk without encounters	Bulk standalone reviews created	PASSED
TC-144	Return review by id	Verify review retrieval	Review ID: "review-123"	1. Find review by ID2. Return review	Review returned	PASSED

## Chapter 6 - Implementation and Testing

TC-145	Throw NotFoundException if review not found	Verify error handling for missing reviews	Review ID: "non-existent"	1. Find review2. Not found3. Throw NotFoundException	NotFoundException thrown	PASSED
TC-146	Return all reviews for a patient	Verify patient's review history	Patient ID: "patient-123"	1. Query reviews by patient2. Return list	Patient's reviews returned	PASSED
TC-147	Return empty array if no reviews found	Verify handling of no data	Patient ID: "patient-456"	1. Query reviews2. No results3. Return empty array	Empty array returned	PASSED

## Chapter 6 - Implementation and Testing

TC-148	Return all reviews for an encounter	Verify encounter-specific reviews	Encounter ID: "encounter-456"	1. Query reviews by encounter 2. Return list	Encounter's reviews returned	PASSED
TC-149	Return empty array if no reviews found	Verify handling of no encounter data	Encounter ID: "encounter-789"	1. Query reviews 2. No results 3. Return empty array	Empty array returned	PASSED
TC-150	Return reviews filtered by category	Verify category-based filtering	Patient ID: "patient-123"Category: "CARDIOVASCULAR"	1. Apply category filter 2. Query reviews 3. Return filtered results	Reviews of specified category returned	PASSED

## Chapter 6 - Implementation and Testing

TC-151	Return empty array if no reviews in category	Verify handling of empty category	Patient ID: "patient-123"Category: "MUSCULOSKELETAL"	1. Query reviews by category 2. No results 3. Return empty array	Empty array returned	PASSED
TC-152	Update review successfully	Verify review update	Review ID: "review-123"New Finding: "Abnormal"	1. Find review 2. Update fields 3. Save	Review updated	PASSED
TC-153	Throw NotFoundException if review not found	Verify update error handling	Review ID: "non-existent"	1. Find review 2. Not found 3. Throw NotFoundException	NotFoundException thrown	PASSED

## Chapter 6 - Implementation and Testing

TC-154	Update with partial data	Verify partial update support	Review ID: "review-123"Partial Data: {notes: "..."}	1. Find review2. Update only provided fields3. Save	Review partially updated	PASSED
TC-155	Soft delete review successfully	Verify soft delete functionality	Review ID: "review-123"	1. Find review2. Set deletedAt timestamp	Review soft deleted	PASSED
TC-156	Throw NotFoundException if review not found	Verify delete error handling	Review ID: "non-existent"	1. Find review2. Not found3. Throw NotFoundException	NotFoundException thrown	PASSED

## Chapter 6 - Implementation and Testing

TC-157	Create a schedule successfully	Verify doctor availability schedule creation	Doctor ID: "doctor-123"Day: Monday (1)Start: "09:00"End: "17:00"Slot Duration: 30 min	1. Validate day of week (0-6). 2. Validate time range. 3. Check for conflicts. 4. Create schedule.	Schedule created successfully	PASSED
TC-158	Throw ConflictException if schedule overlaps	Verify prevention of overlapping schedules	Doctor ID: "doctor-123"Day: Monday (1)Start: "10:00"End: "14:00"(Conflicts with existing)	1. Validate inputs. 2. Check for time overlap. 3. Throw ConflictException.	ConflictException thrown	PASSED
TC-159	Throw BadRequestException if end time before start time	Verify time validation	Start Time: "17:00"End Time: "09:00"	1. Compare start and end times. 2. Throw BadRequestException.	BadRequestException thrown	PASSED

## Chapter 6 - Implementation and Testing

TC-160	Throw BadRequestException if day of week is invalid	Verify day validation (0-6)	Day of Week: 7 (invalid)	1. Validate day range 2. Throw BadRequestException	BadRequestException thrown	PASSED
TC-161	Throw BadRequestException if day of week is negative	Verify negative day validation	Day of Week: -1 (invalid)	1. Validate day range 2. Throw BadRequestException	BadRequestException thrown	PASSED
TC-162	Throw BadRequestException if slot duration is too small	Verify minimum 5 minutes duration	Slot Duration: 4 minutes	1. Validate duration range 2. Throw BadRequestException	BadRequestException thrown	PASSED

## Chapter 6 - Implementation and Testing

TC-163	Throw BadRequestException if slot duration is too large	Verify maximum 120 minutes duration	Slot Duration: 150 minutes	1. Validate duration range 2. Throw BadRequestException	BadRequestException thrown	PASSED
TC-164	Use default slot duration if not provided	Verify default 30 minutes duration	No slot duration provided	1. Check if duration provided 2. Use default 30 minutes 3. Create schedule	Schedule created with 30-minute slots	PASSED
TC-165	Set effectiveFrom to current date if not provided	Verify default effective date	No effectiveFrom provided	1. Check if date provided 2. Use current date 3. Create schedule	Schedule created with current date	PASSED

## Chapter 6 - Implementation and Testing

TC-166	Use provided effectiveFrom and effectiveUntil dates	Verify custom date ranges	effectiveFrom: "2024-01-01"effectiveUntil : "2024-12-31"	1. Use provided dates2. Create schedule	Schedule created with custom date range	PASSED
TC-167	Return schedule by id	Verify schedule retrieval	Schedule ID: "schedule-123"	1. Find schedule by ID2. Return schedule	Schedule returned	PASSED
TC-168	Throw NotFoundException if schedule not found	Verify error handling for missing schedules	Schedule ID: "non-existent"	1. Find schedule2. Not found3. Throw NotFoundException	NotFoundException thrown	PASSED

## Chapter 6 - Implementation and Testing

TC-169	Return schedules for a doctor	Verify doctor's schedule list	Doctor ID: "doctor-123"	1. Query schedules by doctor2. Return list	Doctor's schedules returned	PASSED
TC-170	Update schedule successfully	Verify schedule update	Schedule ID: "schedule-123"New End Time: "18:00"	1. Find schedule2. Check for conflicts3. Update schedule	Schedule updated	PASSED
TC-171	Throw ConflictException if update causes overlap	Verify conflict prevention on update	Schedule ID: "schedule-123"New Times cause conflict	1. Check for conflicts2. Conflict found3. Throw ConflictException	ConflictException thrown	PASSED

## Chapter 6 - Implementation and Testing

TC-172	Throw BadRequestException if day of week is invalid	Verify update day validation	Day of Week: 8 (invalid)	1. Validate day range 2. Throw BadRequestException	BadRequestException thrown	PASSED
TC-173	Throw BadRequestException if day of week is negative	Verify update negative day validation	Day of Week: -2 (invalid)	1. Validate day range 2. Throw BadRequestException	BadRequestException thrown	PASSED
TC-174	Throw BadRequestException if new times are invalid	Verify update time validation	Start: "18:00" End: "10:00"	1. Validate time range 2. Throw BadRequestException	BadRequestException thrown	PASSED

## Chapter 6 - Implementation and Testing

TC-175	Throw BadRequestException if slot duration is too small	Verify update duration validation	Slot Duration: 3 minutes	1. Validate duration 2. Throw BadRequestException	BadRequestException thrown	PASSED
TC-176	Throw BadRequestException if slot duration is too large	Verify update max duration validation	Slot Duration: 180 minutes	1. Validate duration 2. Throw BadRequestException	BadRequestException thrown	PASSED
TC-177	Not check conflicts if day and times are not changing	Verify optimization for unchanged values	Schedule ID: "schedule-123"Only notes changed	1. Compare values 2. Skip conflict check 3. Update	Conflict check skipped	PASSED

## Chapter 6 - Implementation and Testing

TC-178	Check conflicts when day is changed	Verify conflict check on day change	Schedule ID: "schedule-123"Day changed to Tuesday	1. Detect day change 2. Check for conflicts 3. Update if no conflicts	Conflict check performed	PASSED
TC-179	Check conflicts when start time is changed	Verify conflict check on start change	Schedule ID: "schedule-123"Start time changed	1. Detect time change 2. Check for conflicts 3. Update if no conflicts	Conflict check performed	PASSED
TC-180	Check conflicts when end time is changed	Verify conflict check on end change	Schedule ID: "schedule-123"End time changed	1. Detect time change 2. Check for conflicts 3. Update if no conflicts	Conflict check performed	PASSED

## Chapter 6 - Implementation and Testing

TC-181	Return active schedules for doctor	Verify active schedule filtering	Doctor ID: "doctor-123"	1. Query schedules by doctor2. Filter by isActive = true3. Return active schedules	Active schedules returned	PASSED
TC-182	Check for conflicts without exclusion	Verify conflict detection	Doctor ID: "doctor-123" Day: Monday Time: 09:00-17:00	1. Query existing schedules2. Check for overlaps3. Return conflict status	Conflict detection works	PASSED
TC-183	Check for conflicts with exclusion	Verify conflict detection excluding current	Schedule ID: "schedule-123" Exclude from check	1. Query existing schedules2. Exclude current schedule3. Check for overlaps	Conflict detection with exclusion works	PASSED

## Chapter 6 - Implementation and Testing

TC-184	Return schedules for date range	Verify date range filtering	Doctor ID: "doctor-123"Start Date: "2024-01-01"End Date: "2024-12-31"	1. Query schedules by doctor2. Filter by date range3. Return schedules	Schedules in date range returned	PASSED
TC-185	Create an appointment successfully	Verify appointment booking with time slot	Patient ID: "patient-123"Doctor ID: "doctor-456"Time Slot ID: "slot-789"Reason: "Checkup"	1. Find time slot2. Verify slot is available3. Mark slot as booked4. Create appointment	Appointment created with SCHEDULED status	PASSED
TC-186	Throw NotFoundException if time slot not found	Verify time slot validation	Time Slot ID: "non-existent"	1. Find time slot2. Not found3. Throw NotFoundException	NotFoundException thrown	PASSED

## Chapter 6 - Implementation and Testing

TC-187	Throw ConflictException if time slot not available	Verify prevention of double booking	Time Slot ID: "slot-789" Status: "BOOKED"	1. Find time slot2. Check slot status3. Throw ConflictException	ConflictException thrown	PASSED
TC-188	Return appointment by id	Verify appointment retrieval	Appointment ID: "apt-123"	1. Find appointment by ID2. Return appointment	Appointment returned	PASSED
TC-189	Throw NotFoundException if appointment not found	Verify error handling for missing appointments	Appointment ID: "non-existent"	1. Find appointment2. Not found3. Throw NotFoundException	NotFoundException thrown	PASSED

## Chapter 6 - Implementation and Testing

TC-190	Cancel a scheduled appointment	Verify cancellation of SCHEDULED appointment	Appointment ID: "apt-123"Status: "SCHEDULED"	1. Find appointment2. Check status3. Update to CANCELLED	Appointment cancelled	PASSED
TC-191	Throw BadRequestException if appointment already completed	Verify completed appointments cannot be cancelled	Appointment ID: "apt-123"Status: "COMPLETED"	1. Find appointment2. Check status3. Throw BadRequestException	BadRequestException thrown	PASSED
TC-192	Not release time slot for late cancellations	Verify <24 hours cancellation policy	Appointment ID: "apt-123"Appointment Date: 12 hours from now	1. Find appointment2. Check if <24 hours before3. Cancel appointment4. Keep slot as BOOKED	Appointment cancelled, slot remains BOOKED	PASSED

## Chapter 6 - Implementation and Testing

TC-193	Release time slot for early cancellations	Verify >24 hours cancellation policy	Appointment ID: "apt-123" Appointment Date: 3 days from now	1. Find appointment2. Check if >24 hours before3. Cancel appointment4. Release time slot	Appointment cancelled, slot changed to AVAILABLE	PASSED
TC-194	Throw BadRequestException for past appointments	Verify past appointments cannot be cancelled	Appointment ID: "apt-123" Appointment Date: Yesterday	1. Find appointment2. Check if in past3. Throw BadRequestException	BadRequestException thrown	PASSED
TC-195	Check in a scheduled appointment	Verify status transition to CHECKED_IN	Appointment ID: "apt-123" Status: "SCHEDULED"	1. Find appointment2. Verify status is SCHEDULED3. Update to CHECKED_IN	Appointment status changed to CHECKED_IN	PASSED

## Chapter 6 - Implementation and Testing

TC-196	Throw BadRequestException if appointment not scheduled	Verify check-in validation	Appointment ID: "apt-123"Status: "COMPLETED"	1. Find appointment2. Check status3. Throw BadRequestException	BadRequestException thrown	PASSED
TC-197	Start a visit for checked-in appointment	Verify status transition to IN_PROGRESS	Appointment ID: "apt-123"Status: "CHECKED_IN"	1. Find appointment2. Verify status is CHECKED_IN3. Create encounter4. Update to IN_PROGRESS	Appointment status changed to IN_PROGRESS	PASSED
TC-198	Complete an in-progress appointment	Verify status transition to COMPLETED	Appointment ID: "apt-123"Status: "IN_PROGRESS"	1. Find appointment2. Verify status is IN_PROGRESS3. Update to COMPLETED	Appointment status changed to COMPLETED	PASSED

## Chapter 6 - Implementation and Testing

TC-199	Mark appointment as no-show	Verify no-show handling	Appointment ID: "apt-123" Status: "SCHEDULED"	1. Find appointment 2. Verify valid transition 3. Update to NO_SHOW	Appointment status changed to NO_SHOW	PASSED
TC-200	Allow valid transitions	Verify state machine transitions	Various status combinations	1. Test valid transitions 2. Verify all allowed	Valid transitions work correctly	PASSED
TC-201	Reject invalid transitions	Verify prevention of invalid state changes	Invalid status combinations	1. Test invalid transitions 2. Verify all rejected	Invalid transitions rejected	PASSED

## Chapter 6 - Implementation and Testing

TC-202	Allow cancellation from SCHEDULED	Verify SCHEDULED to CANCELLED transition	Status: "SCHEDULED" → "CANCELLED"	1. Check transition validity 2. Allow transition	Transition allowed	PASSED
TC-203	Allow cancellation from CHECKED_IN	Verify CHECKED_IN to CANCELLED transition	Status: "CHECKED_IN" → "CANCELLED"	1. Check transition validity 2. Allow transition	Transition allowed	PASSED
TC-204	Allow NO_SHOW from SCHEDULED	Verify SCHEDULED to NO_SHOW transition	Status: "SCHEDULED" → "NO_SHOW"	1. Check transition validity 2. Allow transition	Transition allowed	PASSED

## Chapter 6 - Implementation and Testing

TC-205	Allow NO_SHOW from CHECKED_IN	Verify CHECKED_IN to NO_SHOW transition	Status: "CHECKED_IN" → "NO_SHOW"	1. Check transition validity 2. Allow transition	Transition allowed	PASSED
TC-206	Not allow transitions from COMPLETED	Verify terminal state validation	Status: "COMPLETED" → any	1. Check transition validity 2. Reject transition	Transition rejected	PASSED
TC-207	Not allow transitions from CANCELLED	Verify terminal state validation	Status: "CANCELLED" → any	1. Check transition validity 2. Reject transition	Transition rejected	PASSED

## Chapter 6 - Implementation and Testing

TC-208	Not allow transitions from NO_SHOW	Verify terminal state validation	Status: "NO_SHOW" → any	1. Check transition validity 2. Reject transition	Transition rejected	PASSED
TC-209	Return true for past appointments	Verify past date detection	Appointment Date: Yesterday	1. Compare appointment date with current date 2. Return true if past	Returns true	PASSED
TC-210	Return false for future appointments	Verify future date detection	Appointment Date: Tomorrow	1. Compare appointment date with current date 2. Return false if future	Returns false	PASSED

## Chapter 6 - Implementation and Testing

TC-211	Return true for cancellations more than 24 hours before	Verify early cancellation detection	Appointment Date: 3 days from nowCancellation Time: Now	1. Calculate time difference 2. Return true if >24 hours	Returns true	PASSED
TC-212	Return false for cancellations less than 24 hours before	Verify late cancellation detection	Appointment Date: 12 hours from nowCancellation Time: Now	1. Calculate time difference 2. Return false if <24 hours	Returns false	PASSED
TC-213	Return paginated appointments for patient	Verify patient appointment list	Patient ID: "patient-123"Page: 1Limit: 10	1. Query appointments by patient 2. Apply pagination 3. Return page	Paginated appointment list returned	PASSED

## Chapter 6 - Implementation and Testing

TC-214	Return paginated appointments for doctor	Verify doctor appointment list	Doctor ID: "doctor-456"Page: 1Limit: 10	1. Query appointments by doctor2. Apply pagination3. Return page	Paginated appointment list returned	PASSED
TC-215	Return valid transitions map	Verify state machine configuration	N/A	1. Get transitions map2. Verify all valid transitions defined	Valid transitions map returned	PASSED
TC-216	Generate slots for matching days	Verify time slot generation from schedule	Schedule ID: "schedule-123"Date Range: Next 7 days	1. Find schedule2. Generate slots for matching days3. Return slots	Time slots generated for matching days	PASSED

## Chapter 6 - Implementation and Testing

TC-217	Throw NotFoundException if schedule not found	Verify schedule validation	Schedule ID: "non-existent"	1. Find schedule 2. Not found 3. Throw NotFoundException	NotFoundException thrown	PASSED
TC-218	Throw BadRequestException if schedule is inactive	Verify active schedule validation	Schedule ID: "schedule-123" isActive: false	1. Find schedule 2. Check if active 3. Throw BadRequestException	BadRequestException thrown	PASSED
TC-219	Not generate slots for non-matching days	Verify day matching logic	Schedule: Monday onlyDate Range includes Tuesday	1. Check day of week 2. Skip non-matching days 3. Generate only for Monday	Slots generated only for matching days	PASSED

## Chapter 6 - Implementation and Testing

TC-220	Return time slot by id	Verify time slot retrieval	Time Slot ID: "slot-123"	1. Find time slot by ID2. Return time slot	Time slot returned	PASSED
TC-221	Throw NotFoundException if slot not found	Verify error handling for missing slots	Time Slot ID: "non-existent"	1. Find time slot2. Not found3. Throw NotFoundException	NotFoundException thrown	PASSED
TC-222	Mark available slot as booked	Verify status change from AVAILABLE to BOOKED	Time Slot ID: "slot-123" Status: "AVAILABLE"	1. Find time slot2. Verify status is AVAILABLE3. Update to BOOKED	Time slot status changed to BOOKED	PASSED

## Chapter 6 - Implementation and Testing

TC-223	Throw BadRequestException if slot is not available	Verify booking validation	Time Slot ID: "slot-123"Status: "BOOKED"	1. Find time slot2. Check status3. Throw BadRequestException	BadRequestException thrown	PASSED
TC-224	Mark booked slot as available	Verify status change from BOOKED to AVAILABLE	Time Slot ID: "slot-123"Status: "BOOKED"	1. Find time slot2. Verify status is BOOKED3. Update to AVAILABLE	Time slot status changed to AVAILABLE	PASSED
TC-225	Throw BadRequestException if slot is not booked	Verify release validation	Time Slot ID: "slot-123"Status: "AVAILABLE"	1. Find time slot2. Check status3. Throw BadRequestException	BadRequestException thrown	PASSED

## Chapter 6 - Implementation and Testing

TC-226	Return available slots	Verify filtering by AVAILABLE status	Doctor ID: "doctor-123" Date: Tomorrow	1. Query time slots2. Filter by status = AVAILABLE3. Return slots	Available time slots returned	PASSED
TC-227	Block slots in range	Verify slot blocking for exceptions	Doctor ID: "doctor-123" Start: "2024-01-15 09:00" End: "2024-01-15 12:00"	1. Find slots in range2. Update status to BLOCKED3. Return count	Slots in range blocked	PASSED
TC-228	Expire old slots	Verify marking past slots as EXPIRED	Cutoff Date: Yesterday	1. Find slots before cutoff2. Update status to EXPIRED3. Return count	Past slots marked as EXPIRED	PASSED

## Chapter 6 - Implementation and Testing

TC-229	Calculate correct slot count	Verify slot count calculation	Start: "09:00"End: "17:00"Duration: 30 min	1. Calculate time difference2. Divide by slot duration3. Return count	Correct slot count calculated (16 slots)	PASSED
TC-230	Handle different durations	Verify duration-based calculation	Start: "09:00"End: "17:00"Duration: 60 min	1. Calculate with different duration2. Return count	Correct slot count for 60-min duration (8 slots)	PASSED
TC-231	Create exception and block slots	Verify exception creation and slot blocking	Doctor ID: "doctor-123"Date: "2024-01-15"Start: "09:00"End: "12:00"Reason: "Conference"	1. Validate date and time2. Create exception3. Block affected slots4. Return exception	Exception created and slots blocked	PASSED

## Chapter 6 - Implementation and Testing

TC-232	Throw BadRequestException if end time before start time	Verify time validation for exceptions	Start: "12:00"End: "09:00"	1. Compare times 2. Throw BadRequestException	BadRequestException thrown	PASSED
TC-233	Throw BadRequestException for past dates	Verify past date validation	Date: Yesterday	1. Compare with current date 2. Throw BadRequestException	BadRequestException thrown	PASSED
TC-234	Identify affected appointments	Verify finding appointments in exception range	Exception Date/Time Range	1. Query appointments in range 2. Return list	Affected appointments identified	PASSED

## Chapter 6 - Implementation and Testing

TC-235	Return exception by id	Verify exception retrieval	Exception ID: "exception-123"	1. Find exception by ID2. Return exception	Exception returned	PASSED
TC-236	Throw NotFoundException if not found	Verify error handling for missing exceptions	Exception ID: "non-existent"	1. Find exception2. Not found3. Throw NotFoundException	NotFoundException thrown	PASSED
TC-237	Delete exception and unblock slots	Verify exception removal and slot unblocking	Exception ID: "exception-123"	1. Find exception2. Unblock affected slots3. Delete exception	Exception deleted and slots unblocked	PASSED

## Chapter 6 - Implementation and Testing

TC-238	Return affected appointments	Verify listing affected appointments	Exception ID: "exception-123"	1. Get exception date/time range 2. Query appointments 3. Return list	Affected appointments returned	PASSED
TC-239	Check if time is blocked	Verify time blocking validation	Doctor ID: "doctor-123" Date/Time: "2024-01-15 10:00"	1. Query exceptions for doctor 2. Check if time falls in exception 3. Return boolean	Returns true if blocked, false otherwise	PASSED
TC-240	Return upcoming exceptions	Verify future exception list	Doctor ID: "doctor-123"	1. Query exceptions for doctor 2. Filter by date $\geq$ today 3. Return list	Upcoming exceptions returned	PASSED

## Chapter 6 - Implementation and Testing

TC-241	Create an organization successfully	Verify organization creation with all details	Name: "City Medical Center"Code: "CMC001"Address: "123 Main St"Phone: "555-1234"	1. Validate name is not empty2. Check code uniqueness3. Create organization4. Set isActive to true	Organization created with all details	PASSED
TC-242	Throw ConflictException if code already exists	Verify unique code validation	Code: "CMC001" (already exists)	1. Check if code exists2. Throw ConflictException	ConflictException thrown	PASSED
TC-243	Throw BadRequestException if name is empty	Verify name required validation	Name: "" (empty string)	1. Validate name2. Throw BadRequestException	BadRequestException thrown	PASSED

## Chapter 6 - Implementation and Testing

TC-244	Throw BadRequestException if name is only whitespace	Verify whitespace validation	Name: " " (only spaces)	1. Trim and validate name2. Throw BadRequestException	BadRequestException thrown	PASSED
TC-245	Create organization without code	Verify optional code field	Name: "General Hospital" No code provided	1. Validate name2. Create without code	Organization created without code	PASSED
TC-246	Trim whitespace from all fields	Verify input sanitization	Name: " Hospital " Address: " 123 St "	1. Trim all string fields 2. Create organization	Organization created with trimmed values	PASSED

## Chapter 6 - Implementation and Testing

TC-247	Return organization by id	Verify organization retrieval	Organization ID: "org-123"	1. Find organization by ID 2. Return organization	Organization returned	PASSED
TC-248	Throw NotFoundException if organization not found	Verify error handling for missing organizations	Organization ID: "non-existent"	1. Find organization 2. Not found 3. Throw NotFoundException	NotFoundException thrown	PASSED
TC-249	Return paginated organizations	Verify pagination support	Page: 1 Limit: 10	1. Query with pagination 2. Apply limit and offset 3. Return page	Paginated organization list returned	PASSED

## Chapter 6 - Implementation and Testing

TC-250	Filter by search term when provided	Verify search functionality	Search: "Medical"	1. Apply search filter 2. Query organizations 3. Return filtered results	Organizations matching search returned	PASSED
TC-251	Not include search filter when search is undefined	Verify optional search	No search term	1. Check if search provided 2. Query without filter	All organizations returned	PASSED
TC-252	Return active organizations	Verify active organization filtering	N/A	1. Query organizations 2. Filter by isActive = true 3. Return active only	Active organizations returned	PASSED

## Chapter 6 - Implementation and Testing

TC-253	Update organization successfully	Verify organization update	Organization ID: "org-123" New Name: "Updated Medical Center"	1. Find organization 2. Update fields 3. Return updated organization	Organization updated successfully	PASSED
TC-254	Throw NotFoundException if organization not found	Verify update error handling	Organization ID: "non-existent"	1. Find organization 2. Not found 3. Throw NotFoundException	NotFoundException thrown	PASSED
TC-255	Throw ConflictException if new code already exists	Verify code uniqueness on update	New Code: "GH002" (exists)	1. Check if new code exists 2. Throw ConflictException	ConflictException thrown	PASSED

## Chapter 6 - Implementation and Testing

TC-256	Not check code conflict if code unchanged	Verify optimization for unchanged code	Organization ID: "org-123" Code: unchanged	1. Compare code values 2. Skip check if same	Code conflict check skipped	PASSED
TC-257	Update all fields with trimmed values	Verify update sanitization	Name: " New Name " Address: " New Address "	1. Trim all fields 2. Update organization	Organization updated with trimmed values	PASSED
TC-258	Handle undefined optional fields	Verify optional field handling	Only name updated Other fields undefined	1. Update only provided fields 2. Keep others unchanged	Only specified fields updated	PASSED

## Chapter 6 - Implementation and Testing

TC-259	Activate an organization	Verify setting isActive to true	Organization ID: "org-123" isActive: false	1. Find organization 2. Set isActive to true	Organization activated	PASSED
TC-260	Deactivate an organization	Verify setting isActive to false	Organization ID: "org-123" isActive: true	1. Find organization 2. Set isActive to false	Organization deactivated	PASSED
TC-261	Soft delete organization	Verify soft delete functionality	Organization ID: "org-123"	1. Find organization 2. Set deletedAt timestamp	Organization soft deleted	PASSED

## Chapter 6 - Implementation and Testing

TC-262	Throw NotFoundException if organization not found	Verify delete error handling	Organization ID: "non-existent"	1. Find organization 2. Not found 3. Throw NotFoundException	NotFoundException thrown	PASSED
TC-263	Return organization statistics	Verify calculation of doctor and appointment counts	Organization ID: "org-123"	1. Count doctors in organization 2. Count appointments by status 3. Calculate totals 4. Return statistics	Statistics returned with correct counts	PASSED
TC-264	Throw NotFoundException if organization not found	Verify statistics error handling	Organization ID: "non-existent"	1. Find organization 2. Not found 3. Throw NotFoundException	NotFoundException thrown	PASSED

## Chapter 6 - Implementation and Testing

TC-265	Calculate total appointments correctly	Verify appointment count aggregation	Organization ID: "org-123"	1. Count all appointments 2. Group by status 3. Calculate total	Total appointments calculated correctly	PASSED
TC-266	Return doctors for organization	Verify organization's doctor list	Organization ID: "org-123"	1. Query doctors by organization 2. Return list	Organization's doctors returned	PASSED
TC-267	Throw NotFoundException if organization not found	Verify doctor list error handling	Organization ID: "non-existent"	1. Find organization 2. Not found 3. Throw NotFoundException	NotFoundException thrown	PASSED

## Chapter 6 - Implementation and Testing

TC-268	Additional organization validation tests	Verify comprehensive validation rules	Various invalid inputs	1. Test various validation rules 2. Verify all work correctly	All validation rules work correctly	PASSED
TC-269	Create a new doctor successfully	Verify doctor account creation with organization	Email: "doctor@test.com" Password: "Doctor123!" First Name: "John" Last Name: "Smith" Organization: "org-123"	1. Validate required fields 2. Check email uniqueness 3. Verify organization exists 4. Hash password 5. Create doctor 6. Assign to organization	Doctor account created and assigned to organization	PASSED
TC-270	Throw ConflictException if email already exists	Verify duplicate email prevention	Email: "existing@test.com"	1. Check if email exists 2. Throw ConflictException	ConflictException thrown	PASSED

## Chapter 6 - Implementation and Testing

TC-271	Throw BadRequestException if email is empty	Verify email required validation	Email: "" (empty)	1. Validate email 2. Throw BadRequestException	BadRequestException thrown	PASSED
TC-272	Throw NotFoundException if organization not found	Verify organization validation	Organization ID: "non-existent"	1. Find organization 2. Not found 3. Throw NotFoundException	NotFoundException thrown	PASSED
TC-273	Throw BadRequestException if organization is inactive	Verify active organization validation	Organization ID: "org-123" isActive: false	1. Find organization 2. Check if active 3. Throw BadRequestException	BadRequestException thrown	PASSED

## Chapter 6 - Implementation and Testing

TC-274	Throw BadRequestException if firstName is empty	Verify first name required	First Name: "" (empty)	1. Validate firstName 2. Throw BadRequestException	BadRequestException thrown	PASSED
TC-275	Throw BadRequestException if lastName is empty	Verify last name required	Last Name: "" (empty)	1. Validate lastName 2. Throw BadRequestException	BadRequestException thrown	PASSED
TC-276	Throw BadRequestException if organizationId is missing	Verify organization required	No organizationId provided	1. Validate organizationId 2. Throw BadRequestException	BadRequestException thrown	PASSED

## Chapter 6 - Implementation and Testing

TC-277	Generate temporary password if not provided	Verify auto-generation of secure password	No password provided	1. Generate random 12-character password 2. Hash password 3. Create doctor 4. Send welcome email	Temporary password generated and doctor created	PASSED
TC-278	Trim whitespace from fields	Verify input sanitization	Email: "doctor@test.com" " Name: " John "	1. Trim all string fields 2. Create doctor	Doctor created with trimmed values	PASSED
TC-279	Handle email service error gracefully	Verify email failure handling	Email service throws error	1. Attempt to send email 2. Catch error 3. Log error 4. Continue operation	Error logged, operation continues	PASSED

## Chapter 6 - Implementation and Testing

TC-280	Return doctor with organizations	Verify doctor retrieval with org relationships	Doctor ID: "doctor-123"	1. Find doctor by ID 2. Include organizations 3. Return doctor	Doctor returned with organizations list	PASSED
TC-281	Throw NotFoundException if doctor not found	Verify error handling for missing doctors	Doctor ID: "non-existent"	1. Find doctor 2. Not found 3. Throw NotFoundException	NotFoundException thrown	PASSED
TC-282	Update doctor successfully	Verify doctor update	Doctor ID: "doctor-123" New Specialization: "Cardiology"	1. Find doctor 2. Update fields 3. Return updated doctor	Doctor updated successfully	PASSED

## Chapter 6 - Implementation and Testing

TC-283	Throw NotFoundException if doctor not found	Verify update error handling	Doctor ID: "non-existent"	1. Find doctor 2. Not found 3. Throw NotFoundException	NotFoundException thrown	PASSED
TC-284	Suspend doctor and cancel future appointments	Verify account suspension and appointment cancellation	Doctor ID: "doctor-123" Status: "ACTIVE"	1. Find doctor 2. Verify not already suspended 3. Cancel future appointments 4. Update status to SUSPENDED	Doctor suspended, future appointments cancelled	PASSED
TC-285	Throw BadRequestException if already suspended	Verify prevention of re-suspension	Doctor ID: "doctor-123" Status: "SUSPENDED"	1. Find doctor 2. Check status 3. Throw BadRequestException	BadRequestException thrown	PASSED

## Chapter 6 - Implementation and Testing

TC-286	Activate doctor successfully	Verify activation of suspended account	Doctor ID: "doctor-123" Status: "SUSPENDED"	1. Find doctor 2. Verify has organizations 3. Update status to ACTIVE	Doctor activated successfully	PASSED
TC-287	Throw BadRequestException if no organizations assigned	Verify organization requirement	Doctor ID: "doctor-123" Organizations: [] (empty)	1. Find doctor 2. Check organizations 3. Throw BadRequestException	BadRequestException thrown	PASSED
TC-288	Throw BadRequestException if already active	Verify prevention of re-activation	Doctor ID: "doctor-123" Status: "ACTIVE"	1. Find doctor 2. Check status 3. Throw BadRequestException	BadRequestException thrown	PASSED

## Chapter 6 - Implementation and Testing

TC-289	Throw NotFoundException if doctor not found	Verify activation error handling	Doctor ID: "non-existent"	1. Find doctor 2. Not found 3. Throw NotFoundException	NotFoundException thrown	PASSED
TC-290	Assign doctor to organization successfully	Verify organization assignment creation	Doctor ID: "doctor-123" Organization ID: "org-456" Is Primary: true	1. Verify doctor exists 2. Verify organization exists and active 3. Create assignment	Doctor assigned to organization successfully	PASSED
TC-291	Throw NotFoundException if doctor not found	Verify assignment error handling	Doctor ID: "non-existent"	1. Find doctor 2. Not found 3. Throw NotFoundException	NotFoundException thrown	PASSED

## Chapter 6 - Implementation and Testing

TC-292	Throw NotFoundException if organization not found	Verify organization validation	Organization ID: "non-existent"	1. Find organization 2. Not found 3. Throw NotFoundException	NotFoundException thrown	PASSED
TC-293	Throw BadRequestException if organization is inactive	Verify active org validation	Organization ID: "org-456" isActive: false	1. Find organization 2. Check if active 3. Throw BadRequestException	BadRequestException thrown	PASSED
TC-294	Default isPrimary to false	Verify default primary flag	Doctor ID: "doctor-123" Organization ID: "org-456" No isPrimary specified	1. Create assignment 2. Set isPrimary to false by default	Assignment created with isPrimary = false	PASSED

## Chapter 6 - Implementation and Testing

TC-295	Remove doctor from organization successfully	Verify organization assignment removal	Doctor ID: "doctor-123" Organization ID: "org-456" (Doctor has 2 organizations)	1. Verify doctor exists 2. Check doctor has >1 organization 3. Remove assignment	Doctor removed from organization successfully	PASSED
TC-296	Throw BadRequestException if only one organization	Verify requirement of at least one org	Doctor ID: "doctor-123" Organizations: 1	1. Count doctor's organizations 2. Throw BadRequestException	BadRequestException thrown	PASSED
TC-297	Throw NotFoundException if assignment not found	Verify assignment validation	Doctor ID: "doctor-123" Organization ID: "org-999"	1. Find assignment 2. Not found 3. Throw NotFoundException	NotFoundException thrown	PASSED

## Chapter 6 - Implementation and Testing

TC-298	Throw NotFoundException if doctor not found	Verify removal error handling	Doctor ID: "non-existent"	1. Find doctor 2. Not found 3. Throw NotFoundException	NotFoundException thrown	PASSED
TC-299	Return doctors for organization	Verify organization's doctor list	Organization ID: "org-123"	1. Query doctors by organization 2. Return list	Organization's doctors returned	PASSED
TC-300	Throw NotFoundException if organization not found	Verify list error handling	Organization ID: "non-existent"	1. Find organization 2. Not found 3. Throw NotFoundException	NotFoundException thrown	PASSED

## Chapter 6 - Implementation and Testing

TC-301	Delete doctor and cancel appointments	Verify soft delete and appointment cancellation	Doctor ID: "doctor-123"	1. Find doctor 2. Cancel future appointments 3. Set deletedAt timestamp	Doctor soft deleted and appointments cancelled	PASSED
TC-302	Return doctor statistics	Verify calculation of appointment and patient counts	Doctor ID: "doctor-123"	1. Count appointments 2. Count unique patients 3. Calculate statistics 4. Return data	Statistics returned with correct counts	PASSED
TC-303	Return user profile	Verify user profile retrieval without password	User ID: "user-123"	1. Find user by ID 2. Remove password field 3. Return profile	User profile returned with all non-sensitive fields, password removed	PASSED

## Chapter 6 - Implementation and Testing

TC-304	Throw NotFoundException if user not found	Verify error handling for missing users	User ID: "non-existent"	1. Find user 2. Not found 3. Throw NotFoundException	NotFoundException thrown	PASSED
TC-305	Update user profile successfully	Verify profile update	User ID: "user-123" New Phone: "555-9999"	1. Find user 2. Update fields 3. Return updated profile	User profile updated successfully	PASSED
TC-306	Throw NotFoundException if user not found	Verify update error handling	User ID: "non-existent"	1. Find user 2. Not found 3. Throw NotFoundException	NotFoundException thrown	PASSED

## Chapter 6 - Implementation and Testing

TC-307	Change password successfully	Verify password update with validation	User ID: "user-123" Current Password: "OldPass123!" New Password: "NewPass123!"	1. Find user 2. Verify current password 3. Hash new password 4. Update password	Password changed successfully	PASSED
TC-308	Throw BadRequestException if current password is incorrect	Verify password verification	User ID: "user-123" Current Password: "WrongPass"	1. Find user 2. Verify current password 3. Verification fails 4. Throw BadRequestException	BadRequestException thrown	PASSED
TC-309	Throw NotFoundException if user not found	Verify password change error handling	User ID: "non-existent"	1. Find user 2. Not found 3. Throw NotFoundException	NotFoundException thrown	PASSED

## Chapter 6 - Implementation and Testing

TC-310	Delete account successfully	Verify soft delete of user account	User ID: "user-123"	1. Find user 2. Set deletedAt timestamp 3. Return success	User account soft deleted	PASSED
TC-311	DoctorProfileService should be defined	Verify service instantiation	N/A	1. Create service instance 2. Check if defined	Service is defined and ready to use	PASSED
TC-312	Extend BaseProfileService	Verify inheritance	N/A	1. Check inheritance chain 2. Verify methods available	Service extends BaseProfileService correctly	PASSED

## Chapter 6 - Implementation and Testing

TC-313	Use DoctorRepository	Verify repository injection	N/A	1. Check repository injection 2. Verify availability	DoctorRepository properly injected	PASSED
TC-314	Get doctor profile with organizations	Verify inclusion of organization relationships	Doctor ID: "doctor-123"	1. Find doctor 2. Include organizations 3. Return profile	Doctor profile returned with organizations list	PASSED
TC-315	Update doctor-specific fields	Verify doctor-specific data updates	Doctor ID: "doctor-123" Specialization: "Cardiology"	1. Find doctor 2. Update doctor-specific fields 3. Return updated profile	Doctor-specific fields updated successfully	PASSED

## Chapter 6 - Implementation and Testing

TC-316	PatientProfileService should be defined	Verify service instantiation	N/A	1. Create service instance 2. Check if defined	Service is defined and ready to use	PASSED
TC-317	Extend BaseProfileService	Verify inheritance	N/A	1. Check inheritance chain 2. Verify methods available	Service extends BaseProfileService correctly	PASSED
TC-318	Use PatientRepository	Verify repository injection	N/A	1. Check repository injection 2. Verify availability	PatientRepository properly injected	PASSED

## Chapter 6 - Implementation and Testing

TC-319	Get patient profile	Verify patient-specific profile retrieval	Patient ID: "patient-123"	1. Find patient 2. Remove password 3. Return profile	Patient profile returned with all fields	PASSED
TC-320	Update patient-specific fields	Verify patient-specific data updates	Patient ID: "patient-123" Date of Birth: "1990-01-01"	1. Find patient 2. Update patient-specific fields 3. Return updated profile	Patient-specific fields updated successfully	PASSED
TC-321	Create transporter with SMTP config	Verify email transporter initialization	SMTP config from environment	1. Read SMTP configuration 2. Create nodemailer transporter 3. Verify transporter created	Email transporter initialized successfully	PASSED

## Chapter 6 - Implementation and Testing

TC-322	Throw error if SMTP config is missing	Verify configuration validation	Missing SMTP_HOST	1. Check SMTP configuration 2. Throw error if missing	Error thrown with message about missing config	PASSED
TC-323	Throw error if FRONTEND_URL is missing	Verify frontend URL validation	Missing FRONTEND_URL	1. Check FRONTEND_URL 2. Throw error if missing	Error thrown with message about missing URL	PASSED
TC-324	Send verification email successfully	Verify email sending with verification link	To: "patient@test.com" Token: "verify-token-123"	1. Compose email with verification link 2. Send via email service 3. Return success	Email sent successfully with verification link	PASSED

Table 75, Test Cases Table

## **6.5 RTM V.4**

## Chapter 6 - Implementation and Testing

ID	Title	Analysis Section	Design Section	Code	Integration Test	Unit Test
VEMR-FR-PM-01	The system should allow patients to create a new account by providing email, password, first name, and last name.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-PM-02	The system should allow patients to login to the system using their email and password credentials.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-PM-03	The system should allow patients to verify their email address before accessing the system.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-PM-04	The system should allow users to request a password reset link via email and set a new password.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-PM-05	The system should allow users to view their profile information including personal details.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>

## Chapter 6 - Implementation and Testing

VEMR-FR-PM-06	The system should allow users to update their profile information such as name, phone number, and other details.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-PM-07	The system should allow users to change their current password to a new one.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-AU-08	The system should allow administrators to login to the system using their credentials.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-AU-09	The system should allow clinicians/doctors to login to the system using their email and password.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-VM-10	The system should allow clinicians to create a new patient visit with date, type, reason, and chief complaint.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>

## Chapter 6 - Implementation and Testing

VEMR-FR-VM-11	The system should allow clinicians to search visits by patient name or reason for visit.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-VM-12	The system should allow clinicians to edit visit details when the visit is in progress.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-VM-13	The system should allow clinicians to save changes made to a visit including medical record data.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-VM-14	The system should allow clinicians to delete a visit from the system.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-VM-15	The system should allow clinicians to view complete details of a visit including medical record, vitals, and allergies.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-VM-16	The system should allow voice transcription to appear in real-time as the	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>

## Chapter 6 - Implementation and Testing

	clinician speaks using Whisper.					
VEMR-FR-VM-17	The system should allow clinicians to view a paginated list of all patients in the system.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-VM-18	The system allows clinicians to search for patient.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-VM-19	The system should allow clinicians to view a patient's profile with their visits and allergies.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-VM-20	The system should allow clinicians to view detailed patient information including contact and demographic data.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-VM-21	The system should allow clinicians to change visit status: start (planned to in-progress), complete, or cancel.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>

## Chapter 6 - Implementation and Testing

VEMR-FR-VM-22	The system should allow clinicians to view a paginated list of all visits with filtering options.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-VS-23	The system should allow clinicians to view vital signs recorded for a specific visit.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-VS-24	The system should allow clinicians to record and edit vital signs (BP, HR, temp, etc.) during an active visit.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-AM-25	The system should allow clinicians to view a patient's recorded allergies with severity and reaction details.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-AM-26	The system should allow clinicians to add a new allergy record for a patient with type, allergen, severity, and reaction.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>

## Chapter 6 - Implementation and Testing

VEMR-FR-AM-27	The system should allow clinicians to update existing allergy information for a patient.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-AM-28	The system should allow clinicians to delete an allergy record from a patient's profile.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-CM-29	The system should allow the admin to create clinicians' accounts.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-CM-30	The system should allow the clinicians and admins to update their account.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-CM-31	The system should allow the admin to view the clinicians accounts list.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-CM-32	The system should allow the admin to view the clinicians account details.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>

## Chapter 6 - Implementation and Testing

VEMR-FR-CM-33	The system should allow the admin to delete the clinician's accounts.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-CM-34	The system should allow the admin to search the clinician's accounts.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-OM-35	The system should allow the admin to create organizations.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-OM-36	The system should allow the admin to update organization information.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-OM-37	The system should allow the admin to view all organizations.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-OM-38	The system should allow the admin to view detailed information about a specific organization.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>

## Chapter 6 - Implementation and Testing

VEMR-FR-OM-39	The system should allow the admin to delete an organization.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-OM-40	The system should allow the admin to search for organizations.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-AN-41	The system should allow administrators to view system-wide analytics and reports.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-AP-42	The system should allow clinicians to view their appointment schedule.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-AP-43	The system should allow clinicians to view their daily appointment schedule.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-AP-44	The system should allow clinicians to filter appointments by specific date.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-AP-45	The system should allow clinicians to	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>

## Chapter 6 - Implementation and Testing

	filter appointments by status.					
VEMR-FR-AP-46	The system should allow the doctor to check in a patient's appointment.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-AP-47	The system should allow the doctor to cancel a patient's appointment.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-AP-48	The system should allow the doctor to set an appointment as no show.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-DA-49	The system should allow clinicians to view their personal analytics and performance metrics.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-SM-50	The system should allow clinicians to create their work schedule.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-SM-51	The system should allow clinicians to edit	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>

## Chapter 6 - Implementation and Testing

	their existing work schedule.					
VEMR-FR-SM-52	The system should allow clinicians to delete their work schedule.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-SM-53	The system should allow clinicians to automatically generate available visit slots based on their schedule.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-SM-54	The system should allow clinicians to mark appointments as completed, no-show, or cancelled.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-SM-55	The system should allow clinicians to view detailed information about a specific appointment.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-SM-56	The system should allow clinicians to define their available time slots for appointments.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>

## Chapter 6 - Implementation and Testing

VEMR-FR-PP-57	The system should allow patients to view available healthcare organizations.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-PP-58	The system should allow patients to view doctors within specific organizations.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-PP-59	The system should allow patients to view available appointment slots.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-PP-60	The system should allow patients to book an appointment with a clinician.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-PP-61	The system should allow patients to view their own appointment history and upcoming appointments.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-PP-62	The system should allow patients to filter their appointments by status.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>

## Chapter 6 - Implementation and Testing

VEMR-FR-PP-63	The system should allow patients to view their own medical records.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-PP-64	The system should allow patients to view their own allergy information.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>
VEMR-FR-PP-65	The system should allow patients to view their complete visit history.	<a href="#">Specification and Diagrams</a>	<a href="#">Component Diagram</a>	<a href="#">GitHub</a>	<a href="#">Jira</a>	<a href="#">GitHub</a>

Table 76- RTM V.4

## **Chapter 7 - Conclusion, future directions and references**

## 7.1 Introduction

The Electronic Medical Records (EMR) System has successfully established a comprehensive healthcare management platform aimed at revolutionizing the way medical practices manage patient care, clinical documentation, and appointment scheduling. This chapter provides a comprehensive summary of the project's achievements, reiterates how the system addresses identified healthcare management challenges, and outlines a future vision for continuous improvement and expansion.

## 7.2 Summary of Achievements

The development of the EMR System has yielded a robust healthcare management platform with several key achievements:

- **Integrating structured medical record management:** with support for allergies, vital signs, past medical procedures, and review of systems, the platform enables healthcare providers to maintain complete and accurate patient records, significantly improving the quality of clinical documentation and patient care continuity.
- **The system delivers a sophisticated scheduling system with features including:**
  - Real-time availability checking
  - Automated time slot management
  - Smart cancellation policies (early vs. late cancellation)
  - Multiple appointment status tracking (scheduled, checked-in, in-progress, completed, cancelled, no-show)
  - Conflict detection and prevention
- **A comprehensive security framework implementing:**
  - JWT-based authentication with access and refresh tokens
  - Bcrypt password hashing with 10 salt rounds
  - Email verification system
  - Account status management (active, inactive, suspended)
  - Role-specific authorization guards
- **Comprehensive User Management:** A secure system for user registration, authentication, and progress tracking ensures a personalized and continuous learning journey.
- **Intuitive User Interface:** The platform features a user-friendly and responsive UI, designed for seamless navigation across various devices.

## 7.3 Addressing Key Challenges

The EMR System directly addresses the limitations prevalent in traditional healthcare management:

- **Manual, paper-based medical records are prone to loss, damage, and are difficult to access.**
- **Phone-based scheduling leads to double bookings, missed appointments, and administrative overhead.**
- **Patients have minimal access to their own health information and limited control over appointments.**
- **Incomplete or inconsistent medical records across different visits and providers.**

## 7.4 System Reliability and Performance

The technical foundation of the EMR System ensures a reliable and performant healthcare environment:

- **Scalability: Architecture:** Modular NestJS backend with TypeORM for efficient database operations
  - **Database:** PostgreSQL with optimized indexes for fast queries
  - **Design:** Designed to handle multiple organizations, hundreds of doctors, and thousands of patients simultaneously
- **Extensibility:**
  - **Layers Structure:** Clean separation of concerns (Gateway → Service → Repository layers)
  - **Easy Integration:** New features can be added without disrupting existing functionality
  - **API-First Design:** RESTful APIs enable future mobile app development
- **Security:**
  - **Authentication:** JWT tokens with 7-day access and 30-day refresh token expiration
  - **Authorization:** Role-specific guards (AdminJwtGuard, DoctorJwtGuard, PatientJwtGuard)
- **Availability:** Aims for high uptime to ensure continuous access to the platform.

## 7.5 Future Vision: Proposed Enhancements

To further solidify the EMR System's position as a leading healthcare management platform, the following future enhancements are envisioned:

- **7.5.1 Advanced Clinical Features**
  - **E-Prescribing: Electronic prescription management with drug interaction checking**

- **Lab Integration:** Direct integration with laboratory systems for test ordering and results
  - **Imaging Integration:** PACS integration for viewing radiology images
  - **Clinical Decision Support:** AI-powered alerts for drug interactions, allergies, and clinical guidelines
  - **Telemedicine:** Video consultation capabilities for remote patient care
- **7.5.2 Enhanced Scheduling Features**
  - **Recurring Appointments:** Support for follow-up appointment series
  - **Waitlist Management:** Automatic notification when earlier slots become available
  - **Multi-Provider Scheduling:** Book appointments requiring multiple providers
  - **Resource Scheduling:** Schedule rooms, equipment, and other clinic resources
  - **SMS/Email Reminders:** Automated appointment reminders to reduce no-shows
- **7.5.3 Patient Portal Enhancements**
  - **Secure Messaging:** Direct communication between patients and providers
  - **Document Upload:** Patients can upload insurance cards, referrals, and other documents
  - **Family Account Management:** Parents managing children's appointments and records
  - **Health Tracking:** Patient-entered vital signs, symptoms, and medication adherence
  - **Bill Payment:** Online payment for medical services
- **7.5.4 Reporting and Analytics**
  - **Clinical Analytics Dashboard:** Track patient outcomes, quality metrics, and clinical indicators
  - **Financial Reports:** Revenue tracking, billing reports, and insurance claim analytics
  - **Operational Metrics:** Appointment utilization, wait times, and provider productivity
  - **Compliance Reports:** HIPAA audit logs, access reports, and security monitoring
  - **Custom Report Builder:** Allow administrators to create custom reports
- **7.5.5 Mobile Applications**
  - **Native Mobile Apps:** iOS and Android apps for patients and providers
  - **Offline Capability:** Allow doctors to document visits without internet connectivity
  - **Push Notifications:** Real-time alerts for appointments, messages, and lab results
  - **Biometric Authentication:** Fingerprint and face recognition for secure mobile access
- **7.5.6 Interoperability**
  - **HL7/FHIR Integration:\*\*** Standard healthcare data exchange protocols
  - **Health Information Exchange (HIE):\*\*** Share patient data with other health care systems
  - **Insurance Verification:\*\*** Real-time insurance eligibility checking
  - **Pharmacy Integration:\*\*** Electronic prescription transmission to pharmacies
- **7.5.7 AI and Machine Learning**
  - **Predictive Analytics:\*\*** Predict no-show probability and optimize scheduling
  - **Natural Language Processing:\*\*** Auto-populate clinical notes from voice dictation
  - **Risk Stratification:\*\*** Identify high-risk patients requiring proactive care

- **Appointment Optimization:** AI-driven scheduling recommendations
- **7.5.8 Administrative Enhancements**
  - **Billing and Claims Management:** Integrated medical billing system
  - **Inventory Management:** Track medical supplies and equipment
  - **Staff Scheduling:** Manage provider schedules and time-off requests
  - **Compliance Management:** Track certifications, licenses, and training requirements
  - **Multi-Location Support:** Enhanced features for healthcare systems with multiple clinics

## 7.6 Conclusion

*The EMR System has successfully created an innovative and effective platform for healthcare management, addressing critical needs in clinical documentation, appointment scheduling, and patient engagement. By implementing modern technologies, robust security measures, and user-centric design, the system empowers healthcare providers to deliver better patient care while improving operational efficiency.*

## 7.7 References

### Backend Technologies

- **Node.js**  
Node.js Documentation: <https://nodejs.org/docs/>
- **NestJS**  
NestJS Documentation: <https://docs.nestjs.com/>
- **TypeScript**  
TypeScript Documentation: <https://www.typescriptlang.org/docs/>
- **TypeORM**  
TypeORM Documentation: <https://typeorm.io/>
- **PostgreSQL**  
PostgreSQL Documentation: <https://www.postgresql.org/docs/>
- **JWT (JSON Web Tokens)**  
JWT Introduction: <https://jwt.io/introduction>
- **Bcrypt**  
Bcrypt npm Package: <https://www.npmjs.com/package/bcrypt>
- **Jest**  
Jest Documentation: <https://jestjs.io/docs/getting-started>

## Frontend Technologies

- **React**  
React Documentation: <https://react.dev/>
- **TypeScript**  
TypeScript Documentation: <https://www.typescriptlang.org/docs/>
- **Tailwind CSS**  
Tailwind CSS Documentation: <https://tailwindcss.com/docs>
- **Vite**  
Vite Documentation: <https://vitejs.dev/guide/>
- **React Router**  
React Router Documentation: <https://reactrouter.com/>

## Healthcare Standards

- **HL7 FHIR**  
FHIR Documentation: <https://www.hl7.org/fhir/>
- **HIPAA Compliance**  
HIPAA Guidelines: <https://www.hhs.gov/hipaa/>

## Development Resources

- **MDN Web Docs**  
MDN Web Docs: <https://developer.mozilla.org/>
- **Stack Overflow**  
Stack Overflow: <https://stackoverflow.com/>
- **GitHub**  
GitHub: <https://github.com/>

## Testing and Quality

- **Jest Testing Framework**  
Jest Documentation: <https://jestjs.io/>
- **ESLint**  
ESLint Documentation: <https://eslint.org/docs/>