DISSERTATION DOC
EASE PROJECT

A PROJECT REPORT SUBMITTED TO THE FACULTY OF ENGINEERING, DESIGN AND TECHNOLOGY IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF BACHELOR OF SCIENCE IN COMPUTER SCIENCE / BACHELOR OF SCIENCE IN INFORMATION TECHNOLOGY OF UGANDA CHRISTIAN UNIVERSITY

May, 2024

UGANDA CHRISTIAN UNIVERSITY

A Centre of Excellence in the Heart of Africa

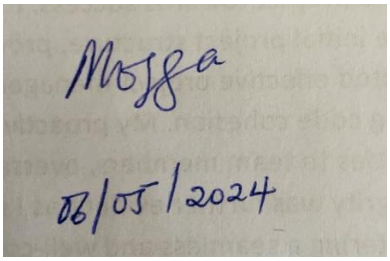
NAME	PROJECT ROLE	REG. NO	CORSE	ACCESS NUMBER
MOGA MUZAMIL ABDUL WAHAB	PROJECT MANAGER	S21B23/013	BSCS	A94166
MUGANGA CHARLES	BACKEND DEVELOPER	J22B23/032	BSCS	A96447
MUKASA SAIDI	FRONTEND DEVELOPER	J22B13/014	BSIT	A96449
MURUNGI NATHAN	FRONTEND DEVELOPER (UI/UX MANAGER)	S21B13/076	BSIT	A94266
SAMANTHA RYAN BAGENDA	SALES MANAGER	S21B13/038	BSIT	A93821
NAKAZIBA CLAIRE ANNE	SALES TEAM LEAD	S21B13/070	BSIT	A95293

DECLARATION

We declare that the content of this report is original and has not been submitted before to any education institution. It is a result of the team's hard work.

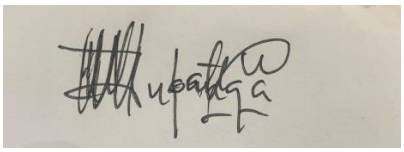
MOGA MUZAMIL ABDUL WAHAB

PROJECT MANAGER



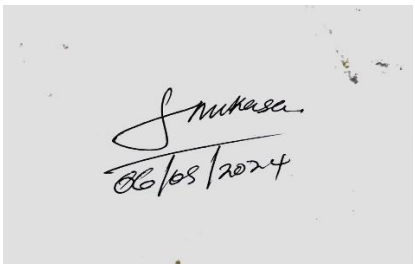
MUGANGA CHARLES

BACKEND DEVELOPER



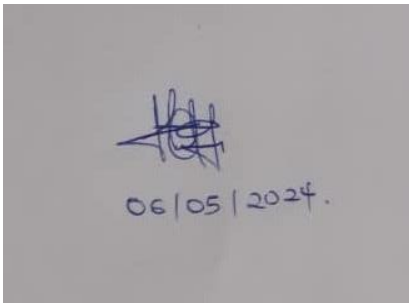
MUKASA SAIDI

FRONTEND DEVELOPER



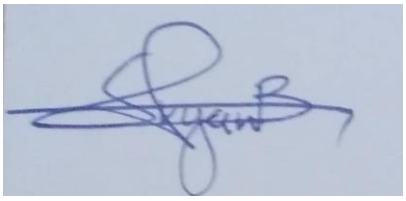
MURUNGI NATHAN

FRONTEND DEVELOPER (UI/UX MANAGER)



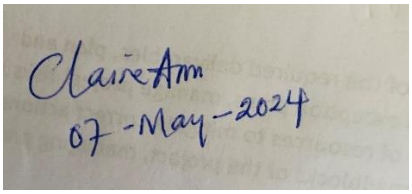
SAMANTHA RYAN BAGENDA

SALES MANAGER



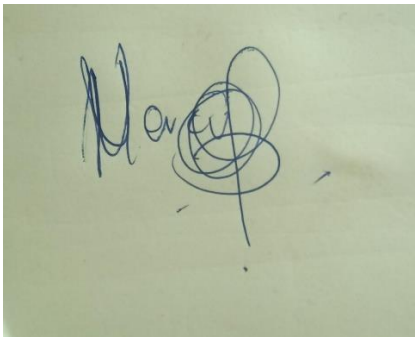
NAKAZIBA CLAIRE ANNE

SALES TEAM LEAD



This report has been submitted for examination with the approval of the following supervisors (Lecturers):

MADAM MERCY NEKESA



MADAM IMMACULATE KAMUSIIME

Signed……

07.05.2024

Date…….

DEDICATION

This project is dedicated to all individuals who strive tirelessly to improve access to healthcare services. To the healthcare professionals who dedicate their lives to caring for others, often in challenging circumstances, your dedication and compassion inspire us all. To the patients who navigate the complexities of the healthcare system with resilience and courage, your strength motivates us to continually innovate and improve.

This project is dedicated to you, as we endeavor to create solutions that empower and uplift communities, ensuring that healthcare remains a fundamental right for all.

We also, with great pleasure and comfort, would like to dedicate this project to all my beloved friends and mentors who gave us advice, courage, support and cooperated during the project development period.

ACKNOWLEDGEMENT

First and foremost, we extend our deepest praise to the Almighty God for making it possible for our team to successfully complete this project.

We would like to express genuine thanks and gratitude to our esteemed lecturers and supervisors, MADAM MERCY NEKESA and MADAM IMMACULATE KAMUSIIME, for their unwavering support, invaluable guidance, and steadfast commitment throughout the course of our project. Their expertise, encouragement, and dedication have been instrumental in shaping the development and success of our endeavor.

Additionally, we extend tremendous thanks and appreciation to our parents for their unwavering support, both financially and emotionally. Their prayers, love, guidance, and sacrifices have been the cornerstone of our journey, and we are deeply grateful for their enduring commitment to our education and well-being.

PREFACE

Healthcare accessibility remains a significant challenge in many parts of the world, particularly in regions with limited infrastructure and resources. In Uganda, where a large portion of the population faces barriers to accessing quality healthcare, innovative solutions are essential. The "Doc Ease" project emerges from a recognition of these challenges and a commitment to leveraging technology to address them. This preface serves as an introduction to the comprehensive telehealth platform developed to enhance healthcare delivery in Uganda. Through the collaborative efforts of a dedicated team, Doc Ease seeks to revolutionize the way healthcare services are accessed and delivered, ultimately improving outcomes and quality of life for individuals across the country. This preface provides insight into the motivations, goals, and aspirations driving the development of Doc Ease, setting the stage for a detailed exploration of its features, functionality, and impact in the following pages.

Table of Contents

Abbreviations (Acronyms)

EMR - Electronic Medical Records

API - Application Programming Interface

AI - Artificial Intelligence

SQL - Structured Query Language

JWT - JSON Web Token

CSS - Cascading Style Sheets

SSE - Server-Sent Events

2FA - Two-Factor Authentication

UI - User Interface

UX - User Experience

Abstract

Access to efficient and comprehensive healthcare services remains a significant challenge for patients in Uganda, primarily attributed to the absence of centralized and functional medical records systems. This deficiency impedes healthcare providers' ability to deliver optimal care, resulting in inefficiencies in medical treatments, monitoring processes, and informed decision-making [1]. Despite efforts to improve healthcare infrastructure, approximately 60% of Uganda's population, residing in diverse rural and urban areas, face limited access to healthcare facilities. Moreover, the Uganda Demographic and Health Survey (2022) reveals that only 23% of health facilities across the country possess complete and functional computerized systems for recording patient information. Consequently, fragmented medical records persist as a pervasive barrier to timely and effective healthcare delivery, affecting individuals nationwide.

In response to these challenges, Doc Ease Project aims to develop and deploy a comprehensive electronic medical records (EMR) platform tailored to Uganda's healthcare landscape. By implementing this EMR system across healthcare facilities, both urban and rural, the project seeks to enhance access to healthcare services, streamline medical treatments, and enable data-driven decision-making among healthcare providers. Through a combination of technological innovation, capacity-building initiatives, and strategic partnerships, the Doc Ease Project endeavors to address the systemic challenges hindering healthcare accessibility and delivery in Uganda, ultimately improving health outcomes and quality of life for individuals across the country.

The significance of the Doc Ease Project lies in its potential to revolutionize healthcare delivery in Uganda by providing a robust, user-friendly, and accessible electronic medical records solution. By centralizing patient information and facilitating seamless data exchange among healthcare providers, the project aims to overcome the barriers posed by fragmented medical records and limited healthcare infrastructure. Additionally, the Doc Ease Project's emphasis on capacity-building and partnerships underscores its commitment to sustainable healthcare solutions tailored to Uganda's unique context. Ultimately, the project's contribution extends beyond technological innovation to encompass broader improvements in healthcare accessibility,

efficiency, and patient outcomes, thereby addressing a critical need and making a lasting impact on Uganda's healthcare landscape.

CHAPTER ONE

1.1 Introduction

1.1.1 Background and context of the project

Access to quality healthcare services is a fundamental human right, yet millions of people around the world, particularly those in underserved or rural areas, face significant barriers to receiving timely and effective medical care. In Uganda, like many developing countries, access to healthcare facilities is limited, with a significant portion of the population residing in areas where medical infrastructure is scarce. According to the Uganda Demographic and Health Survey (2022) [1], only a fraction of health facilities across the country has functional computerized systems for recording patient medical information, leading to fragmented medical records and hindering the delivery of efficient healthcare services.

Recognizing these challenges, there is a pressing need to leverage technology to bridge the gap in healthcare access and delivery, especially in regions where traditional healthcare infrastructure is lacking. Telehealth, the use of digital communication technologies to provide remote healthcare services, presents a promising solution to improve healthcare accessibility. By enabling electronic medical record management, remote medical consultations and seamless communication between patients and healthcare providers, telehealth has the potential to revolutionize healthcare delivery and enhance treatment outcomes for millions of individuals.

In response to these pressing healthcare needs, the Doc Ease project was conceived. Doc Ease is an innovative telehealth application with an integrated Electronic Medical Records (EMR) system designed to address Uganda's healthcare challenges comprehensively. By offering secure storage and access to medical records, and seamless communication channels between patients and healthcare providers, Doc Ease aims to improve healthcare access, minimize delays in care, and enhance treatment outcomes for all Ugandans, irrespective of their geographical location or circumstances.

1.1.2 Problem Statement

Patients in Uganda encounter challenges in accessing efficient and comprehensive healthcare services due to the absence of medical records which hinders the healthcare providers' ability to deliver optimal care, affecting the efficiency of medical treatments, monitoring processes, and informed decision making.

In Uganda, approximately 60% of the population resides in various areas, including rural and urban settings, where access to healthcare facilities is limited. According to the Uganda Demographic and Health Survey (2022), only 23% of health facilities across the country, regardless of location, have a complete and functional computerized system for recording patient's medical information.

These statistics highlight significant challenges in healthcare delivery throughout Uganda, affecting individuals in both rural and urban areas. Fragmented medical records pose barriers to timely and effective care for individuals across the country

1.1.3 Objectives and Goals

Main Objective

- To address healthcare accessibility challenges in Uganda through the implementation of a comprehensive electronic medical records (EMR) system.

Specific Objectives

1. To conduct thorough research and consultations with healthcare professionals and stakeholders in Uganda to identify the specific needs and infrastructure constraints of the healthcare facilities. Define the requirements for the electronic medical records (EMR) platform, ensuring it addresses the absence of medical records hindering optimal care delivery.

2. To develop a user-friendly EMR platform tailored to the identified needs and constraints, employing appropriate technologies and design principles. Deploy the EMR system across healthcare facilities in Uganda, following a phased approach to ensure widespread adoption. Provide training and support to healthcare staff for seamless integration into

their workflow, facilitating sharing and retrieval of patient information to enhance medical treatments and monitoring processes.

3. To develop and integrate a telehealth platform into the EMR system, enabling remote medical consultations for individuals, particularly those in rural and underserved areas. Ensure the platform is user-friendly and accessible via various devices, reducing barriers to healthcare access.

4. To implement robust security measures to safeguard patient health information within the EMR system. Ensure compliance with relevant data protection regulations and standards to maintain confidentiality and integrity. Establish backup and recovery mechanisms to ensure data availability and continuity of care.

1.1.4 Project Scope and limitations

Project Scope

The project scope revolves around the comprehensive management and evolution of the Doc Ease telehealth application with its integrated Electronic Medical Records (EMR) system. Built to address the healthcare accessibility challenges prevalent in Uganda, the application securely stores medical records, facilities seamless communication between patients and healthcare providers, and incorporates features for mental health assessment, records, facilitates seamless communication between Comprising patient and doctor portals, the system aims to cater to the diverse needs of its users.

Limitations:

- The technology stack chosen may impose limitations on future scalability and adaptability.
- Resource constraints may impact the depth and breadth of research and development activities.
- Time constraints may necessitate prioritization and phased feature rollout.
- Effective management of scope creep will be crucial to maintain project focus and deliverables.
- Dependencies on external factors, such as regulatory changes or third-party services, may influence project timelines and outcomes.
- Rigorous testing procedures will be essential to ensure platform robustness and reliability in real-world usage scenarios.

1.1.5 Importance and relevance of the project

- The Doc Ease telehealth system is significant because it can handle important issues with healthcare delivery, especially in areas in Uganda where access to healthcare is scarce. The system has several important advantages, including seamless communication between patients and healthcare providers, mental health assessment, and secure storage of electronic medical records (EMR).
- The system enables individuals, especially those in rural and underserved areas, to access medical consultations remotely, reducing geographical barriers and enhancing healthcare accessibility.
- By providing messaging and appointment scheduling features, the system reduces delays and inconvenience, leading to earlier interventions and improved health outcomes.
- The establishment of a secure and comprehensive EMR system facilitates informed decision-making, continuity of care, and better coordination among healthcare professionals, ultimately improving patient care.
- The inclusion of a mental health assessment feature promotes holistic healthcare by addressing physical and mental well-being, enabling users to receive personalized recommendations and support resources.
- Leveraging telehealth technologies reduces the need for costly and time-consuming in-person visits, making healthcare services more affordable and sustainable while empowering healthcare providers with efficient tools for delivering high-quality care.

1.2 Literature Review

1.2.1 Review of relevant literature, framework, methodologies and technologies – Discussion of existing solutions or approaches.

The literature review for the Doc Ease project encompasses a comprehensive examination of various technologies, frameworks, methodologies, and existing solutions pertinent to telehealth platforms and medical record management systems. This involves studying literature on React.js for frontend development, Express.js for backend development, and TypeScript for enhancing code quality and maintainability. Additionally, research included frameworks like Redux Toolkit

for state management and Prisma ORM for database interaction. Methodologies such as Agile and Waterfall are explored to understand their applicability in project development. Existing solutions like Firebase Cloud Messaging for real-time notifications and SendGrid for email communication are also analyzed to identify best practices and potential integration strategies. It also involved studying approaches used in similar projects such as real-time communication protocols for telehealth consultations, standards like HL7 for EMR interoperability, and AI-driven frameworks for mental health assessment. These insights from the literature inform the project's approach and technology choices.

1.2.2 Research gap

Despite the abundance of literature and available technologies in the field of telehealth and medical record management, there exists a notable research gap [2] concerning the seamless integration of diverse functionalities within a unified platform. While individual components such as real-time chatting, health assessment features, and remote consultations are well-documented, there is limited literature addressing the holistic development of a comprehensive telehealth solution like Doc Ease. Moreover, the research lacks in-depth analysis regarding the specific challenges associated with integrating disparate modules, [3] ensuring cross-platform compatibility, and addressing scalability issues while maintaining user privacy and data security. This research gap highlights the need for a more cohesive [4] approach towards developing integrated telehealth platforms to address the evolving healthcare landscape effectively.

CHAPTER TWO

2.1 Methodology

2.1.1 Description of the approach used in the project

The methodology adopted for the Doc Ease project was a hybrid approach combining elements of Agile and Waterfall methodologies. This hybrid approach allowed for flexibility and adaptability while ensuring thorough planning and documentation throughout the development process. The project team began with a comprehensive analysis of requirements and objectives, followed by iterative development cycles to incrementally build and test the solution. Continuous communication and collaboration between team members, stakeholders, and end-users were integral to the approach, facilitating feedback incorporation and adjustments as needed.

Additionally, the hybrid approach facilitated the prioritization of features based on their importance and feasibility, ensuring the timely delivery of essential functionalities while allowing for the incorporation of additional features in subsequent iterations.

2.1.2 Tools, technologies and methodologies used.

Doc ease tools and technologies

1. React:

React is a popular JavaScript library for building user interfaces, [5] particularly single-page applications. Its component-based architecture makes it suitable for building modular and reusable UI components. In the context of telehealth, [6] react enables developers to create interactive and responsive user interfaces for patients and doctors to interact with the telehealth system seamlessly.

2. Node.js:

Node.js is a server-side JavaScript runtime environment that allows developers to build scalable and high-performance web applications. It is particularly well-suited for building real-time applications and handling asynchronous I/O operations. In the telehealth project, Node.js likely

serves as the backend server for handling data processing, user authentication, and communication between clients and the server.

3. PostgreSQL:

PostgreSQL is a powerful open-source relational database management system (RDBMS) known for its reliability, extensibility, and advanced features. It provides robust support for complex queries, transactions, and data integrity constraints. In your telehealth project, PostgreSQL likely stores patient records, medical histories, and other critical data securely.

4. Redux:

Redux is a predictable state container for JavaScript applications, [10] commonly used with React for managing application state. It follows a unidirectional data flow pattern and enables centralized state management, making it easier to debug and maintain complex application states. In the telehealth project, Redux helps manage application state related to user authentication, patient data, and appointment scheduling.

5. Tailwind CSS:

Tailwind CSS is a utility-first CSS [14] framework that provides low-level utility classes for building custom designs without writing custom CSS. It promotes a functional and composable approach to styling user interfaces, making it easier to maintain consistency and responsiveness across different screen sizes. In the telehealth project, Tailwind CSS facilitates the rapid prototyping and styling of UI components.

6. React Query:

React Query is a data-fetching library for React applications that simplifies state management and caching of remote data. It provides hooks for fetching, caching, and updating data from APIs, making it easier to handle asynchronous data fetching and synchronization with UI components. In the telehealth project, React Query manages data fetching and caching for patient records, appointment data, and other dynamic content.

7. Prisma:

Prisma is an open-source ORM [9] (Object-Relational Mapping) tool for Node.js and TypeScript applications, which simplifies database access and manipulation by providing a type-safe query builder and schema definition language. It supports various databases, including PostgreSQL, MySQL, and SQLite. In the telehealth project, Prisma abstracts away the complexity of database interactions and enables type-safe database access using TypeScript.

8. Server-Sent Events:

Server-Sent Events (SSE) is a web API that enables servers to push real-time updates to clients over HTTP connections. It allows for efficient and lightweight communication between the server and the client without the need for polling or long-lived WebSocket connections. In the telehealth project, SSE likely facilitates real-time notifications and updates for patients and doctors, such as appointment reminders or chat messages.

9. Firebase Storage:

Firebase Storage is a cloud-based object storage service provided by Google Firebase, which allows developers to store and serve user-generated content, such as images, videos, and documents. It provides scalable storage infrastructure with built-in security features and easy integration with Firebase authentication and other services. In the telehealth project, Firebase Storage stores patient profile pictures, medical documents, and other multimedia assets securely in the cloud.

10. Firebase Cloud Messaging:

Firebase Cloud Messaging (FCM) is a cross-platform messaging solution provided by Google Firebase, [16] which enables developers to send push notifications to mobile devices and web browsers. It provides reliable delivery of messages with built-in analytics and targeting capabilities. In Doc Ease project, [15] FCM facilitates real-time communication between patients and doctors, such as appointment reminders, medication alerts, or emergency notifications.

2.1.3 The Development process

The development process for the Doc Ease project followed a hybrid Agile-Waterfall model, combining the iterative nature of Agile with the structured approach of Waterfall. The project began with a comprehensive planning phase, during which requirements were gathered, analyzed, and documented. This phase was followed by a design stage where wireframes, prototypes and architecture diagrams were created to visualize the solution.

2.2 System Design

2.2.1 Architectural Overview

The Doc Ease system follows a client-server architecture, where the client-side application interacts with the server-side components to facilitate telehealth services and manage medical records. The architecture comprises three main layers: presentation layer, application layer, and data layer.

1. Presentation Layer:

The presentation layer encompasses the frontend components of the Doc Ease application, including the patient portal and doctor portal.

It is built using modern web technologies such as TypeScript, React, and Tailwind CSS to ensure a responsive and intuitive user interface.

The patient portal allows users to access features like medical history, messaging, appointment scheduling, and settings.

The doctor portal provides functionalities such as patient management, appointment scheduling, messaging, and settings tailored to healthcare professionals.

2. Application Layer:

The application layer contains the backend components responsible for implementing business logic and handling client requests.

It includes modules for authentication and authorization, user management, medical records management, appointment management, chat messaging, notification management, and mental health assessment.

Implemented using TypeScript [7] and Node.js [15], the application layer utilizes Express.js framework for building RESTful APIs to serve client-side requests.

Integration with external services such as Firebase Storage [16], Firebase Cloud Messaging, SendGrid, and OpenAI API [11] is managed within this layer to facilitate data storage, real-time messaging, email communication, and mental health assessment functionality.

3. Data Layer:

The data layer comprises the database management system and storage solutions used for persisting application data.

PostgreSQL database is employed for storing structured data related to users, medical records, appointments, and other entities.

Firebase Storage serves as a cloud-based storage solution for storing medical files, images, and other application-managed files securely.

Data access and manipulation operations are handled through queries and transactions executed by the application layer components.

2.2.2 Component breakdown Structure

Frontend Components:

User Interface (UI)

Patient Portal

Consists of these features; Dashboard, Medical History, Medical Health Assessment, Messaging, Scheduling and Appointment, Settings and Notifications.

Doctor Portal

Consists of these features; Dashboard, My Patients, Appointments, Notifications, Messages and Settings.

Backend Components:

Authentication and Authorization, User Management, Medical Records Management, Appointment Management, Chat (Messenger), Notification Management and Mental Health Assessment

Infrastructure Components:

Database Management System (PostgreSQL), Cloud Storage [16] (Firebase Storage), Cloud Messaging Service (Firebase Cloud Messaging), Email Service (SendGrid) [12] and OpenAI API [11] Integration.

Technology Stack Components:

TypeScript [7], React, Tailwind CSS [14], Redux, Node.js, Express, PostgreSQL Database, Firebase Services (Storage, Cloud Messaging), SendGrid API [12] and OpenAI API.

2.2.3 Data flow diagrams

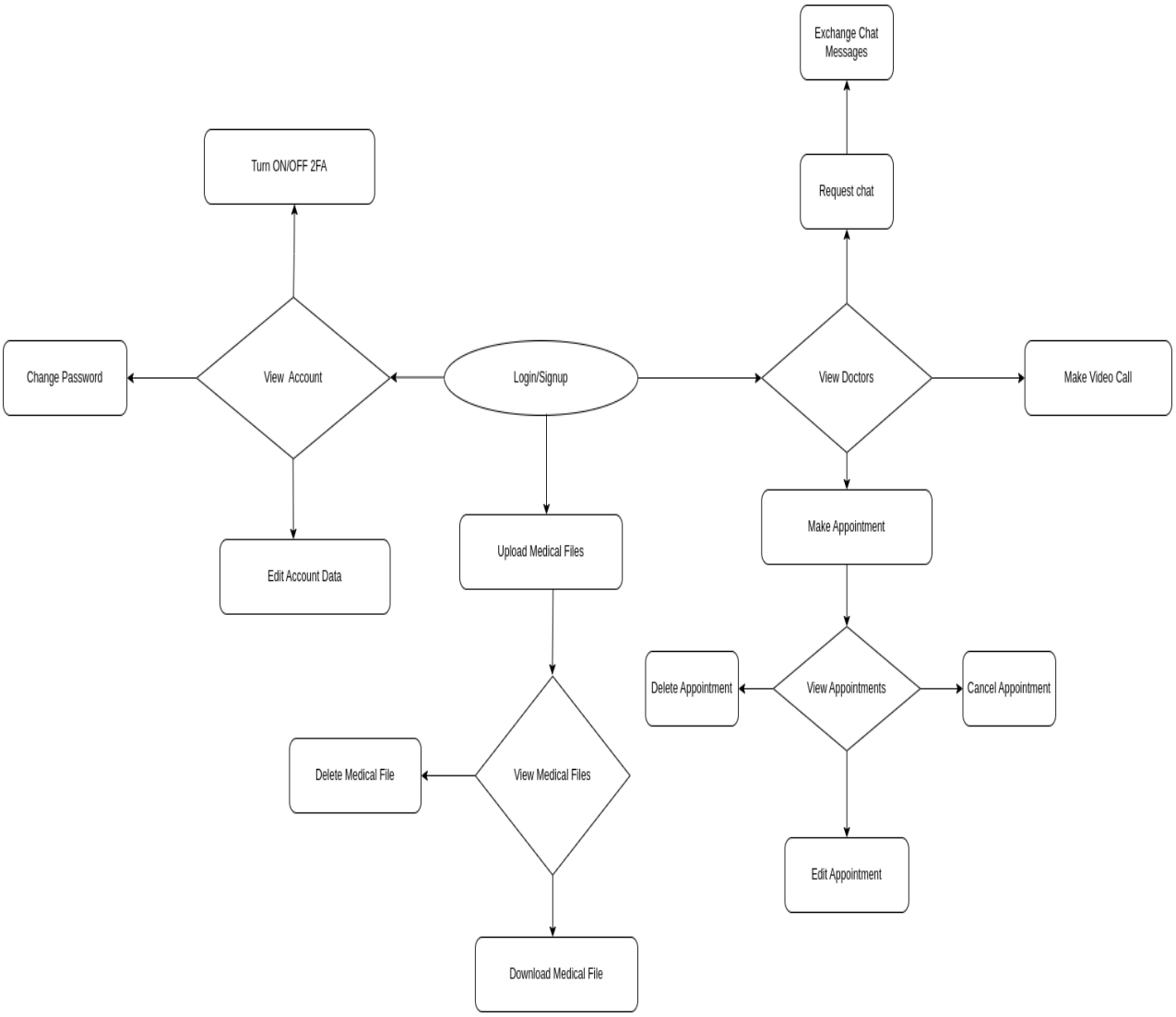


Figure 1: Patient flow chart diagram

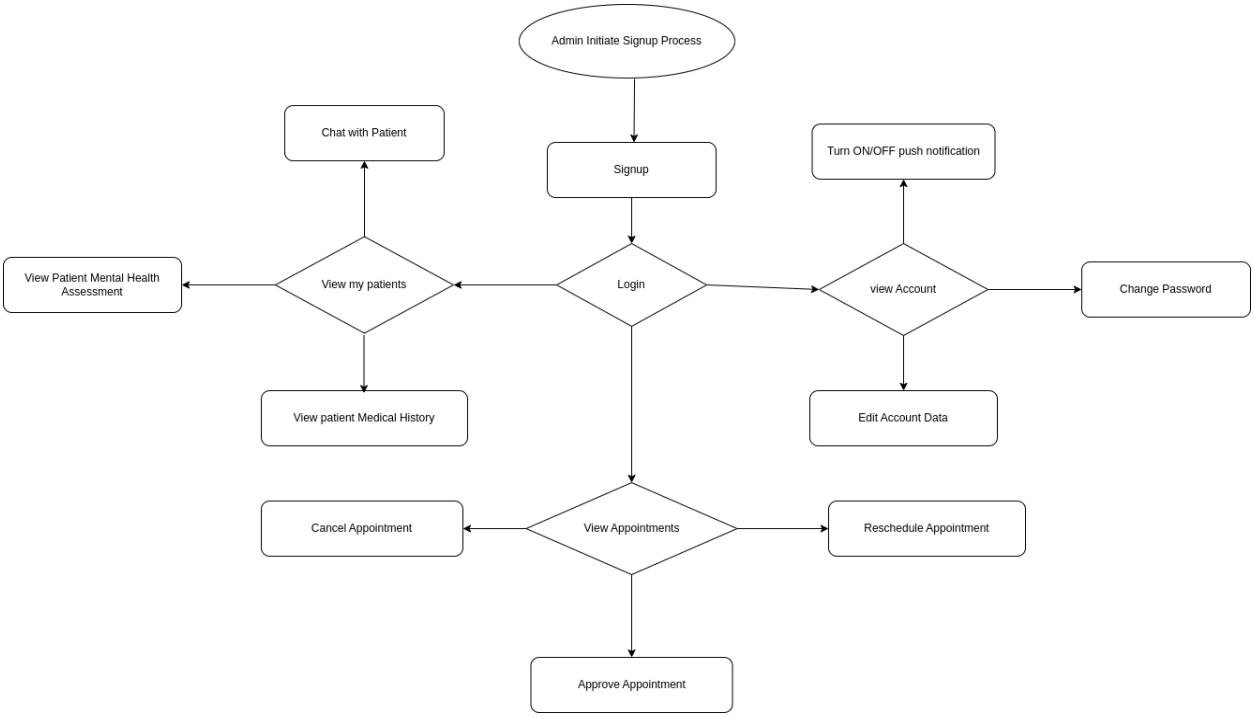


Figure 2: Doctor flow chat diagram

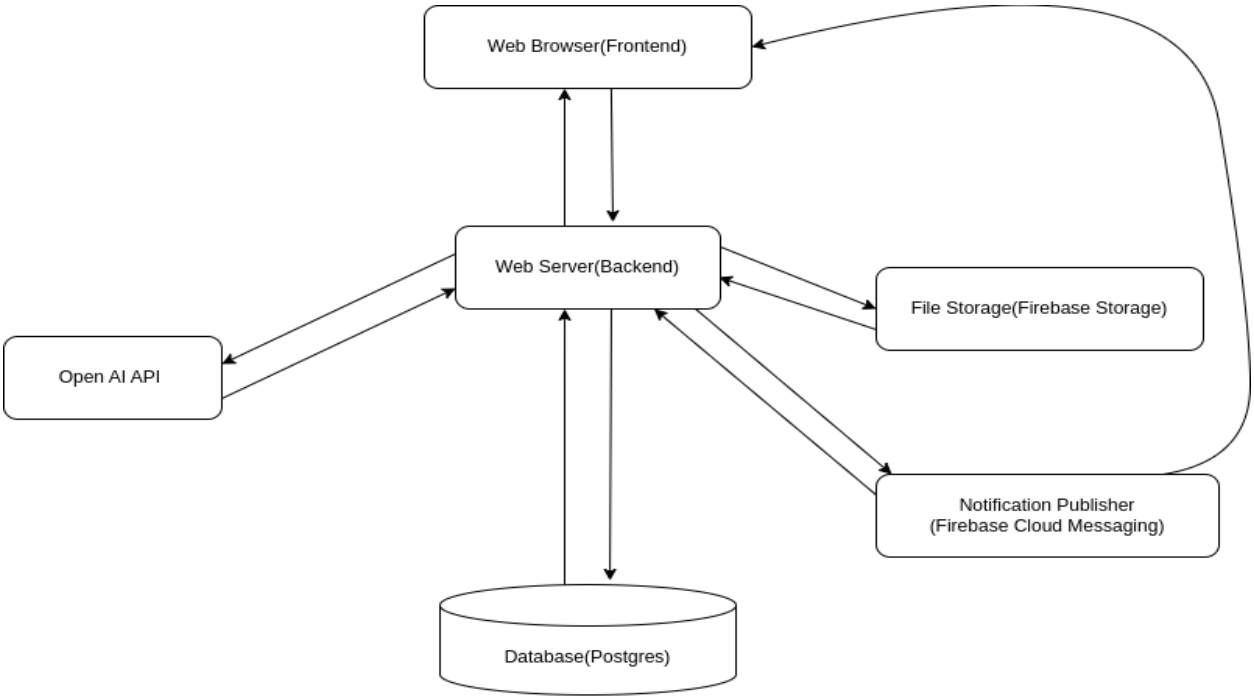


Figure 3: Architectural Diagram

2.2.4 Database Design (Entity Relationship diagrams)

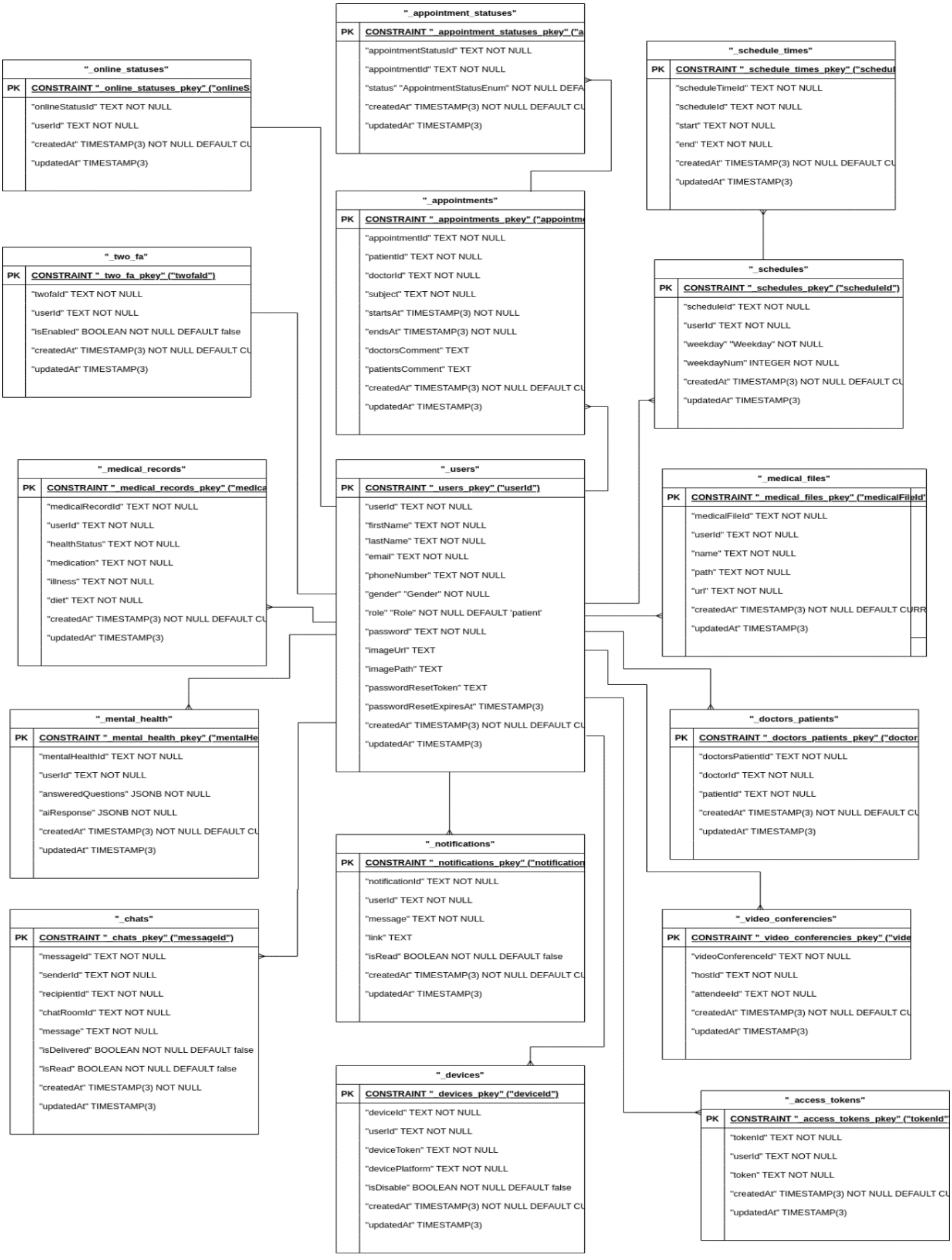

"_appointment_statuses"
PK
CONSTRAINT "_appointment_statuses_pkey" ("a
"appointmentStatusId" TEXT NOT NULL
"appointmentId" TEXT NOT NULL
"status" "AppointmentStatusEnum" NOT NULL DEFA
"createdAt" TIMESTAMP(3) NOT NULL DEFAULT CU
"updatedAt" TIMESTAMP(3)
"_online_statuses"
PK
CONSTRAINT "_online_statuses_pkey" ("onlineS
"onlineStatusId" TEXT NOT NULL
"userId" TEXT NOT NULL
"createdAt" TIMESTAMP(3) NOT NULL DEFAULT CU
"updatedAt" TIMESTAMP(3)
"_schedule_times"
PK
CONSTRAINT "_schedule_times_pkey" ("schedul
"scheduleTimeId" TEXT NOT NULL
"scheduleId" TEXT NOT NULL
"start" TEXT NOT NULL
"end" TEXT NOT NULL
"createdAt" TIMESTAMP(3) NOT NULL DEFAULT CU
"updatedAt" TIMESTAMP(3)
"_appointments"
PK
CONSTRAINT "_appointments_pkey" ("appointm
"appointmentId" TEXT NOT NULL
"patientId" TEXT NOT NULL
"doctorId" TEXT NOT NULL
"subject" TEXT NOT NULL
"startsAt" TIMESTAMP(3) NOT NULL
"endsAt" TIMESTAMP(3) NOT NULL
"doctorsComment" TEXT
"patientsComment" TEXT
"createdAt" TIMESTAMP(3) NOT NULL DEFAULT CU
"updatedAt" TIMESTAMP(3)
"_two_fa"
PK
CONSTRAINT "_two_fa_pkey" ("twofaId")
"twofaId" TEXT NOT NULL
"userId" TEXT NOT NULL
"isEnabled" BOOLEAN NOT NULL DEFAULT false
"createdAt" TIMESTAMP(3) NOT NULL DEFAULT CU
"updatedAt" TIMESTAMP(3)
"_schedules"
PK
CONSTRAINT "_schedules_pkey" ("scheduleId")
"scheduleId" TEXT NOT NULL
"userId" TEXT NOT NULL
"weekday" "Weekday" NOT NULL
"weekdayNum" INTEGER NOT NULL
"createdAt" TIMESTAMP(3) NOT NULL DEFAULT CU
"updatedAt" TIMESTAMP(3)
"_medical_records"
PK
CONSTRAINT "_medical_records_pkey" ("medica
"medicalRecordId" TEXT NOT NULL
"userId" TEXT NOT NULL
"healthStatus" TEXT NOT NULL
"medication" TEXT NOT NULL
"illness" TEXT NOT NULL
"diet" TEXT NOT NULL
"createdAt" TIMESTAMP(3) NOT NULL DEFAULT CU
"updatedAt" TIMESTAMP(3)
"_users"
PK
CONSTRAINT "_users_pkey" ("userId")
"userId" TEXT NOT NULL
"firstName" TEXT NOT NULL
"lastName" TEXT NOT NULL
"email" TEXT NOT NULL
"phoneNumber" TEXT NOT NULL
"gender" "Gender" NOT NULL
"role" "Role" NOT NULL DEFAULT 'patient'
"password" TEXT NOT NULL
"imageUrl" TEXT
"imagePath" TEXT
"passwordResetToken" TEXT
"passwordResetExpiresAt" TIMESTAMP(3)
"createdAt" TIMESTAMP(3) NOT NULL DEFAULT CU
"updatedAt" TIMESTAMP(3)
"_medical_files"
PK
CONSTRAINT "_medical_files_pkey" ("medicalFileId"
"medicalFileId" TEXT NOT NULL
"userId" TEXT NOT NULL
"name" TEXT NOT NULL
"path" TEXT NOT NULL
"url" TEXT NOT NULL
"createdAt" TIMESTAMP(3) NOT NULL DEFAULT CURR
"updatedAt" TIMESTAMP(3)
"_mental_health"
PK
CONSTRAINT "_mental_health_pkey" ("mentalHe
"mentalHealthId" TEXT NOT NULL
"userId" TEXT NOT NULL
"answeredQuestions" JSONB NOT NULL
"aiResponse" JSONB NOT NULL
"createdAt" TIMESTAMP(3) NOT NULL DEFAULT CU
"updatedAt" TIMESTAMP(3)
"_doctors_patients"
PK
CONSTRAINT "_doctors_patients_pkey" ("doctor
"doctorsPatientId" TEXT NOT NULL
"doctorId" TEXT NOT NULL
"patientId" TEXT NOT NULL
"createdAt" TIMESTAMP(3) NOT NULL DEFAULT CU
"updatedAt" TIMESTAMP(3)
"_notifications"
PK
CONSTRAINT "_notifications_pkey" ("notification
"notificationId" TEXT NOT NULL
"userId" TEXT NOT NULL
"message" TEXT NOT NULL
"link" TEXT
"isRead" BOOLEAN NOT NULL DEFAULT false
"createdAt" TIMESTAMP(3) NOT NULL DEFAULT CU
"updatedAt" TIMESTAMP(3)
"_chats"
PK
CONSTRAINT "_chats_pkey" ("messageId")
"messageId" TEXT NOT NULL
"senderId" TEXT NOT NULL
"recipientId" TEXT NOT NULL
"chatRoomId" TEXT NOT NULL
"message" TEXT NOT NULL
"isDelivered" BOOLEAN NOT NULL DEFAULT false
"isRead" BOOLEAN NOT NULL DEFAULT false
"createdAt" TIMESTAMP(3) NOT NULL
"updatedAt" TIMESTAMP(3)
"_video_conferencies"
PK
CONSTRAINT "_video_conferencies_pkey" ("vide
"videoConferenceId" TEXT NOT NULL
"hostId" TEXT NOT NULL
"attendeeId" TEXT NOT NULL
"createdAt" TIMESTAMP(3) NOT NULL DEFAULT CU
"updatedAt" TIMESTAMP(3)
"_devices"
PK
CONSTRAINT "_devices_pkey" ("deviceId")
"deviceId" TEXT NOT NULL
"userId" TEXT NOT NULL
"deviceToken" TEXT NOT NULL
"devicePlatform" TEXT NOT NULL
"isDisable" BOOLEAN NOT NULL DEFAULT false
"createdAt" TIMESTAMP(3) NOT NULL DEFAULT CU
"updatedAt" TIMESTAMP(3)
"_access_tokens"
PK
CONSTRAINT "_access_tokens_pkey" ("tokenId"
"tokenId" TEXT NOT NULL
"userId" TEXT NOT NULL
"token" TEXT NOT NULL
"createdAt" TIMESTAMP(3) NOT NULL DEFAULT CU
"updatedAt" TIMESTAMP(3)

Figure 4: Entity Relationship Diagram.

2.2.5 Component Design (Sequence Diagram, Activity Diagrams)

Sequence Diagram The sequence diagram illustrates the chronological sequence of interactions between different components or actors in the system. In the context of the Doc Ease project:

Patient: Initiates the process by requesting to schedule an appointment through the application.

Application: Receives the appointment request from the patient and notifies the doctor about the request.

Doctor: Upon receiving the notification, confirms their availability for the requested appointment.

Application: Informs the patient about the confirmation from the doctor, finalizing the appointment scheduling process.

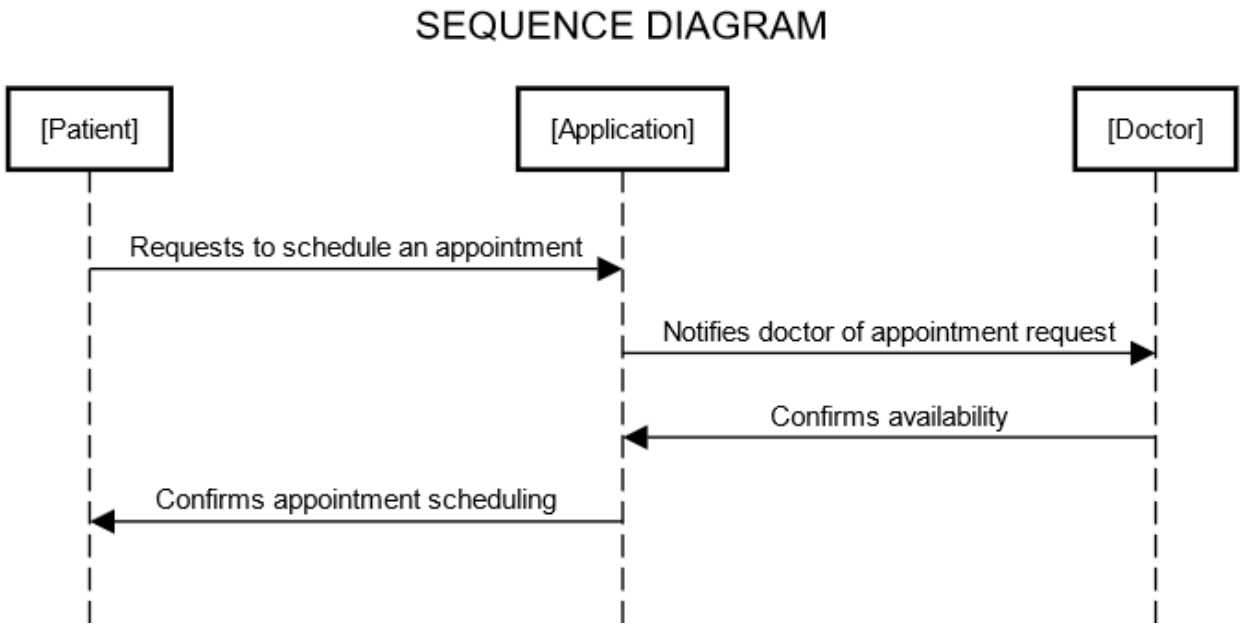


Figure 5: Sequence Diagram

Activity Diagram The activity diagram depicts the flow of activities or processes within the system, showing the order of actions and decision points.

Start: The patient initiates the appointment scheduling process.

Application receives request: The application receives the appointment request from the patient.

Application notifies doctor: The application notifies the doctor about the appointment request.

Doctor confirms availability: The doctor confirms their availability for the requested appointment.

Application confirms appointment scheduling: The application confirms the appointment scheduling to the patient, concluding the process.

End: The appointment scheduling procedure has been effectively finished.

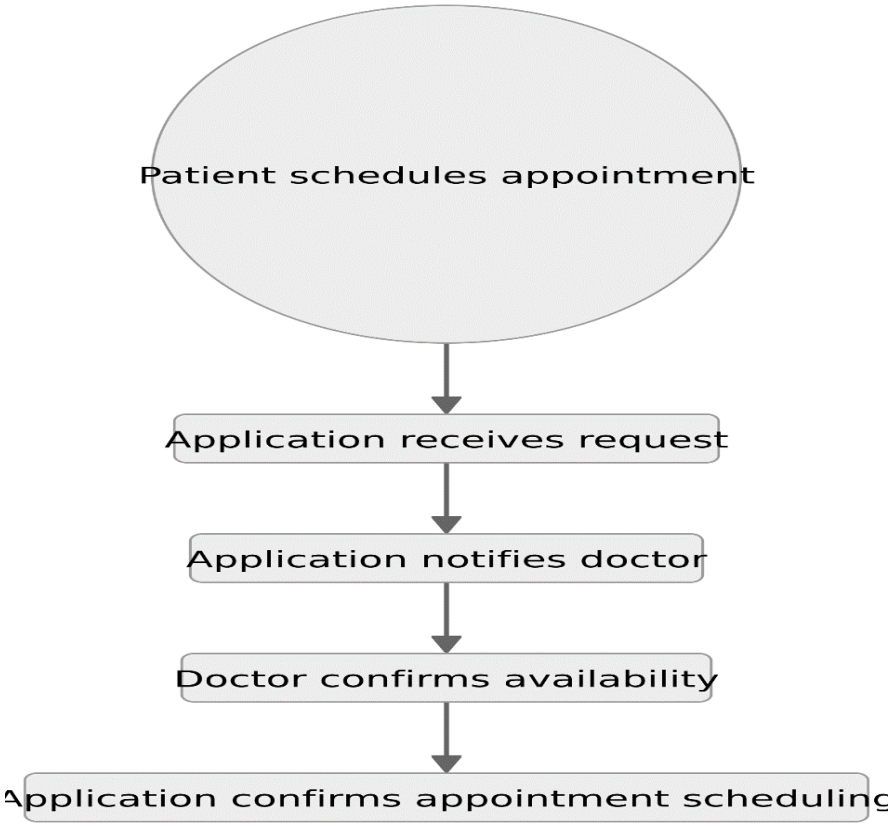


Figure 6: Activity Diagram

2.2.6 Human Interface Design (Overview of the user interface)

The Doc Ease Project's Human Interface Design includes an easy-to-use interface that is suited for both patients and medical professionals (doctors).

• Patient interface:
Dashboard: gives a thorough rundown of all scheduled visits, medical information, and the outcomes of health assessments.

Appointment Scheduling: Patients can easily make, amend, or cancel appointments with healthcare providers thanks to appointment scheduling.

Real-time chat: Enables easy communication for follow-ups, consultations, and questions between patients and physicians.
Features for Health Assessment: Lets users rate their psychological health and get tailored advice.

Notifications: Offers in-the-moment alerts for chat messages, evaluation results, and appointment reminders.

• Healthcare Providers Interface:

Appointment Management: Enables medical professionals to easily see, accept, or postpone appointments.

Patient data: Enables thorough patient history and treatment planning by providing access to electronic medical data.

Real-time chat: Allows for direct patient communication for guidance, follow-ups, and consultations.

Notifications: Notifies medical professionals of new communications, requests for appointments, and patient updates.
Dashboard: Provides a well-organized summary of upcoming activities, patient encounters, and daily routines.

2.3 Implementation

2.3.1 Details about the actual development process

1.Authentication and Authorization

Authentication and authorization are the process by which we verify the legitimacy of our users on the Doc Ease platform before granting them access to any resources respective to their account type. On top of the regular authentication and authorization we also have Two Factor Authentication(2FA) to provide an additional layer of security. There are several situations under which we authenticate users and these include;

I. Sign up,
II. Log in,
III. Successful password reset.

Sign up

Starting with sign up, user fills in the signup form which captures information such as **first name, last name**, **email**, **phone number**, **password** and **confirm password**. It's worth noting that during this process we validate the information that user provides in each input field for example we ensure the following are met;

I. No field s empty,
II. Email must contain the "@" character symbol
III. password must match with confirm password

After doing all those validation on the client (front-end), a user can submit their data to the server (back-end) which further does some extra validations for example ensuring all fields stated above are not empty and the submitted email is unique from those already in existence on the platform.

When all the validations involved are successful, we save user's details in the database and authenticate them. Remember that the user's password is **hashed** before being saved as one of the standard security best practices of protecting user' s data. Depending on the signup endpoint(route), we attach a user **role** alongside user data to determine the account type.

How authentication is implemented on Doc Ease

During the process of authentication, we send the necessary details back to the client (front-end) to enable them have authority to access resources respective to their account type. Some of the details sent to the client include;

I. userId
II. first name and last name
III. user role
IV. JsonWebToken (JWT)

The userId and expiration time is added to the JWT token in the process of generating it.

JWT token is checked for every subsequent request being made to server (back-end) as a way of verifying the user's authenticity and preventing unauthorized access to the user's data on the platform.

How authorization is implemented on Doc Ease

Authorization is the process by which we validate user's authority to access certain resources on the Doc Ease platform. An authorization check is executed on every subsequent request being made to the server (back-end) and this is how its implemented;

I. For every request, we check for authorization in the header "req. headers['authorization']"
II. Extract the JWT token from the authorization header and decode the JWT token for the userId. If no token is found, we fail the request.
III. Check for token expiration and if not expired we proceed the request or else fail it
IV. Check if the user exists in the database and if yes, we proceed the request to the next middleware function to perform the intended task. When no user if found, we fail the requested

By having such vigorous validations layer on our back-end, we ensure high security for the user data.

Log in

For users that already have accounts with Doc Ease platform they can always access resources by signing/logging in.

During log in we capture the user's email and password. Some validations such as ensuring the correct email format, and no empty fields are executed.

When the users submit their email and password, we perform the following the operations on the back-end.

I. Validate to ensure that email and password has been provided
II. Check if the provided email exists in the database and if no we fail the request
III. Compare if the provided password matches with the hashed one saved in the database
IV. Check if user has enabled Two Factor Authentication (2FA) and if yes, we validate the device information. When the device doesn't match any existing ones, we prompt device verification by sending verification token to either the user's email or telephone number. When the device matches the existing ones, we authenticate the user.
V. If Two Factor Authentication(2FA) is off, we authenticate the user.

Successful Password Reset

In scenarios where the Doc Ease users have forgotten their passwords but want to access their accounts, they can reset them. This is how we implemented it.

Implementation for reset password

Password reset starts by user clicking the forgot-password link which is usually found on the log forms. When the forgot password link is clicked, we direct the user to the forgot-password password page which has a form the user can fill by providing an email associated with account. During this process we validate correct email format and also ensure on empty field is submitted. When the user submits his/her email we perform the following operations on the back-end;

I. Validate to ensure that the email has been provided
II. Check if the email exists in the database. If no we fail the requested
III. If the email exists, we generate a password reset token, give it an expiration time, hash it and save to the database
IV. We send the password reset link to the user's email address and notify the user on the front-end.

When the user opens the link sent to their email address, it directs them to the reset-password page which has a reset password form. The Password Reset Token is captured from the page URL. The form contains two fields that is new password and confirm password. We validate for no empty fields and also ensure that the passwords match before submission.

When the new password is submitted to the back-end the following operations are executed;

I. Validate to ensure that token and new password has been provided.
II. Hash the token and use the hashed token to check whether it exists in the database and if no we fail the request.
III. If yes check for token expiration and if the token is expired, we fail the request.
IV. If the token is still valid, then we hash the new provided password and save it in the database.
V. Then we authenticate the user like we did during sign up and log in.

Other Features/Functionalities under Authorization and Authentication

So far, we have looked at features such as sign up, login and successful password reset which all lead to users being authenticated but that is not the full list of all Doc Ease authentication features/functionalities. Other features include the following;

I. Change password when authenticated
II. Two Factor Authentication

Change password when authenticated

We have looked at resetting password in scenarios when users have forgotten their passwords, how about when they just want to change them and that is when "Change password when authenticated" comes.

The change password feature can be accessed by going to setting's page then click "Change Password" in the setting's feature header which scrolls the page and focuses the change password form into the user's view.

The form has three input fields that is "current password", "new password" and "confirm new password". For each field we validate to ensure that it's not empty, then we also ensure that new password matches with confirm new password.

When the form is submitted to the back-end, the following operations are executed;

I. Validate to ensure that current password and new password have been provided
II. We validate to ensure that current password provided matches with the existing password in the database
III. We also validate to ensure that new password provided doesn't much with the existing password in the database
IV. when all the validation checks pass, we hash the new password and update it in the database

And thereby an authenticated users is able to change their passwords whenever required.

Two Factor Authentication (2FA) (Security implementation)

Two Factor Authentication is used to provide an additional layer of security to user's account. It operates on top of the regular authentication mechanism which usually involves email/password.

The Two Factor Authentication can be accessed on Doc Ease by going to setting's page and the click "2FA" in the setting's header which scrolls the page and focuses the card that contains buttons to enable users turn ON/OFF 2FA.

Here is how we implemented Two Factor Authentication for Doc Ease.

Implementation of Two Factor Authentication

To have Two Factor Authentication working, there always a need to have some way of identifying the devices during log in and so for us we decided to identify devices by capturing the following details;

I. Device platform (Operating System)
II. Browser type e.g. chrome, Firefox,
III. Browser version

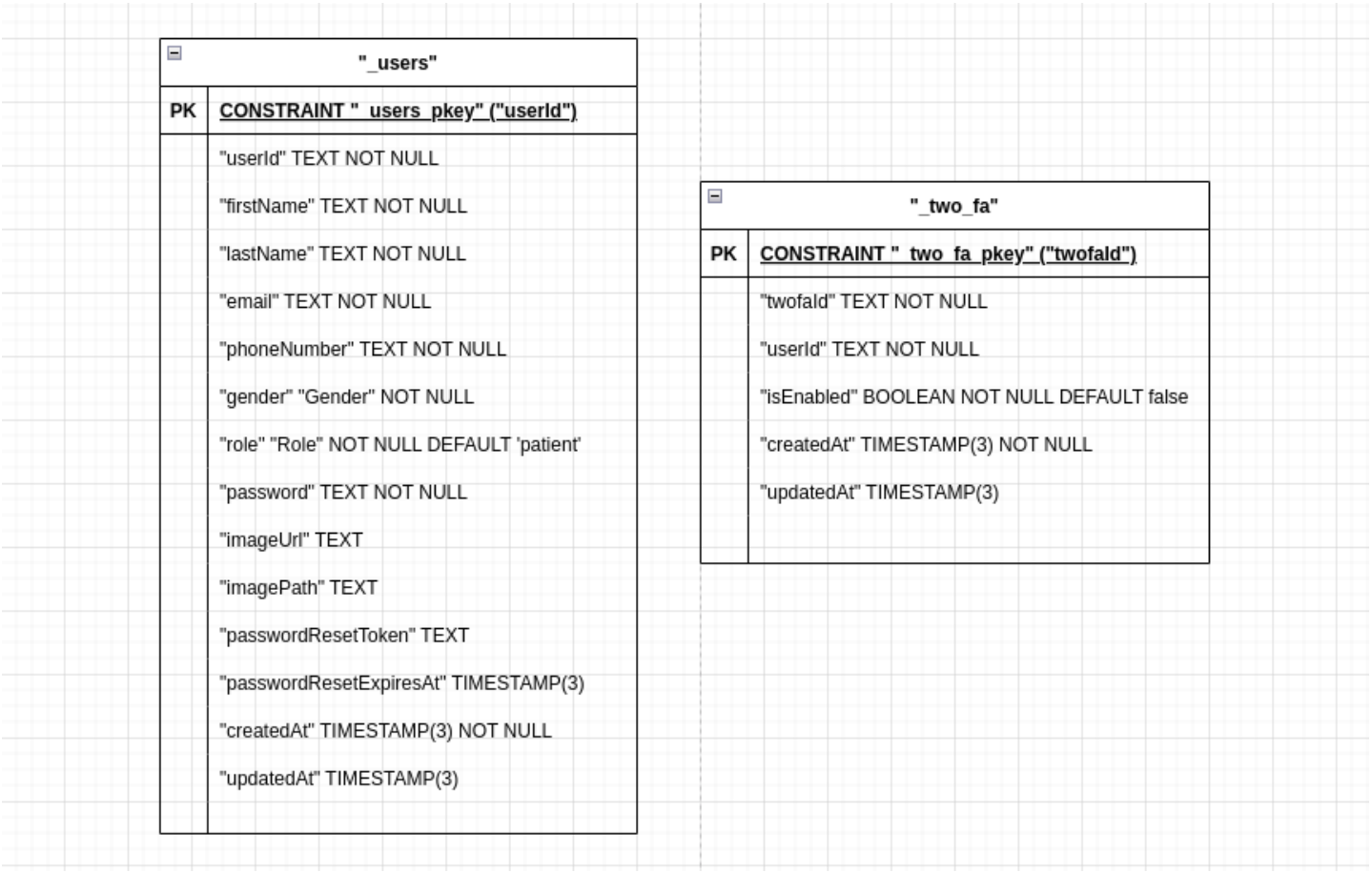
We capture the above device information during every login and also whenever a user is trying turn on Two Factor Authentication for his/her account.

During every login when a user provides correct email and password, we check whether he/she has factor Two Factor enabled and if no, we just authenticate the user. When Two Factor Authentication is turned on, we go further ahead and validate device details and if we find an

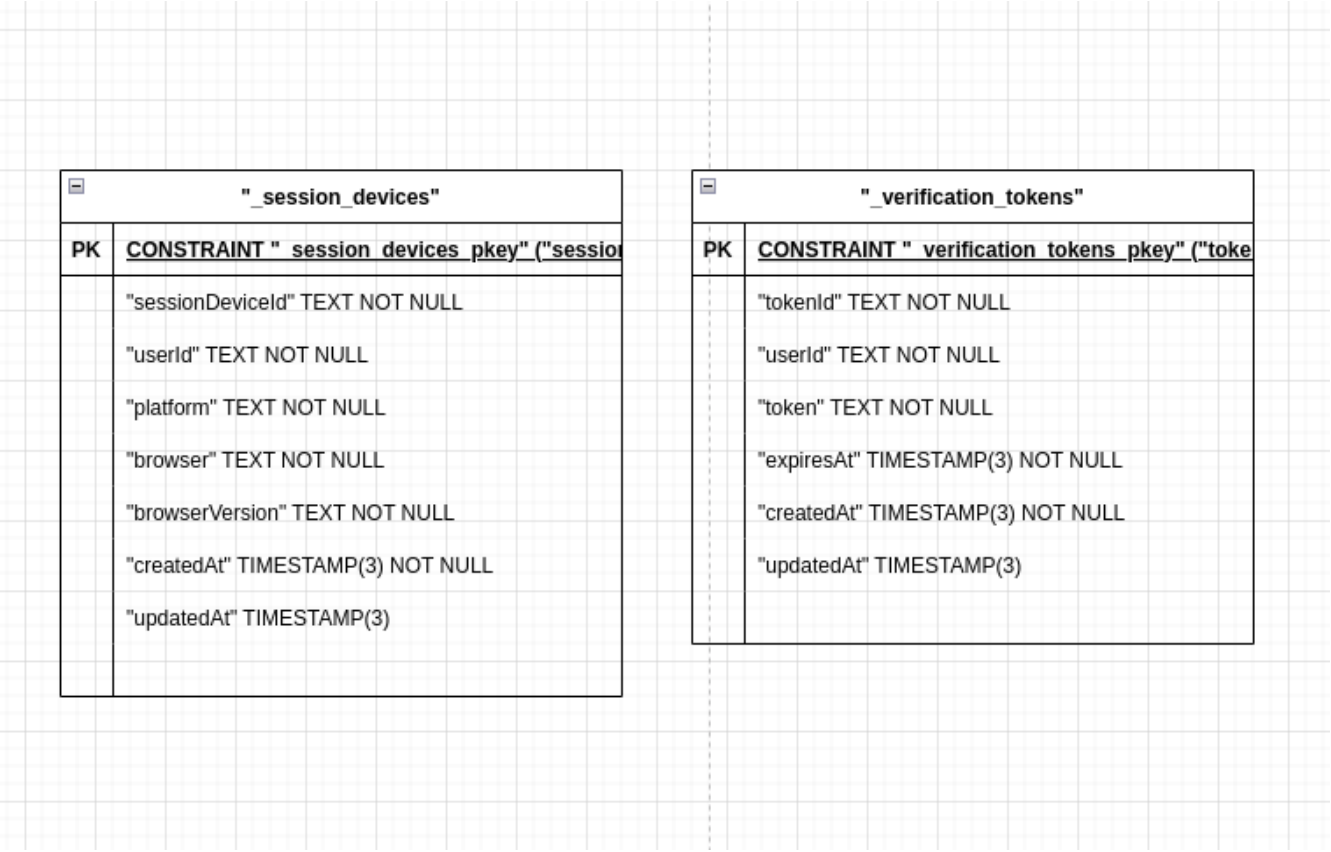
exact match already saved in the database, the user is also authenticated. In scenarios where the device details don't match any existing one, we send a device verification token to either the user's email or telephone number and then notify them on the front-end side. The user is automatically redirected to the page with a form for filling in a verification token and if the supplied token is correct and valid, we authenticate the user.

The tables powering authentication, authorization and Two Factor Authentication

Tables for users and two factor authentication



Tables for session devices and verification tokens



2.Medical Records

The Medical Record feature enables users to securely store and manage their medical information within the application. Users can create, view, and delete medical records containing details such as health status, medication, and illness history. Additionally, users can upload medical files, such as reports or images, for easy access and reference. Uploaded files are securely stored in **Firebase Storage**, ensuring the safety and confidentiality of sensitive medical data.

This feature empowers users to maintain a comprehensive record of their health journey, facilitating better communication with healthcare providers and informed decision-making regarding their well-being. By centralizing medical information within the application and leveraging Firebase Storage for file storage, users can conveniently access their records whenever needed, enhancing efficiency and organization in managing their healthcare data.

Posting Medical Record:

Parameters:

I. userId: ID of the user for whom the medical record is being created.

II. healthStatus: Health status of the user.

III. medication: Details of any medication prescribed.

IV. illness: Description of any illnesses or conditions.

V. diet: Information about the user's diet. To create a medical record, the userId must be provided along with the health status, medication details, illness description, and diet information. If any of these fields are missing, the request will be rejected.

Getting Medical Records by User:

Parameters:

i. **userId:** ID of the user for whom medical records are to be retrieved.

To fetch medical records associated with a specific user, the userId must be provided. If the userId is missing, the request will be rejected.

Deleting Medical Record:

Parameters:

i. **medicalRecordId:** ID of the medical record to delete.

To delete a specific medical record, the medicalRecordId must be provided. If the medicalRecordId is missing, the request will be rejected.

Uploading Medical Record File:

Parameters:

i. **userId:** ID of the user associated with the medical file.
ii. **file:** Medical file to upload.

To upload a medical file, the userId and the file must be provided. If either of these is missing, the request will be rejected.

Getting Medical Record Files by User:

Parameters:

I. - userId: ID of the user for whom medical files are to be retrieved.

To fetch medical files associated with a specific user, the userId must be provided. If the userId is missing, the request will be rejected.

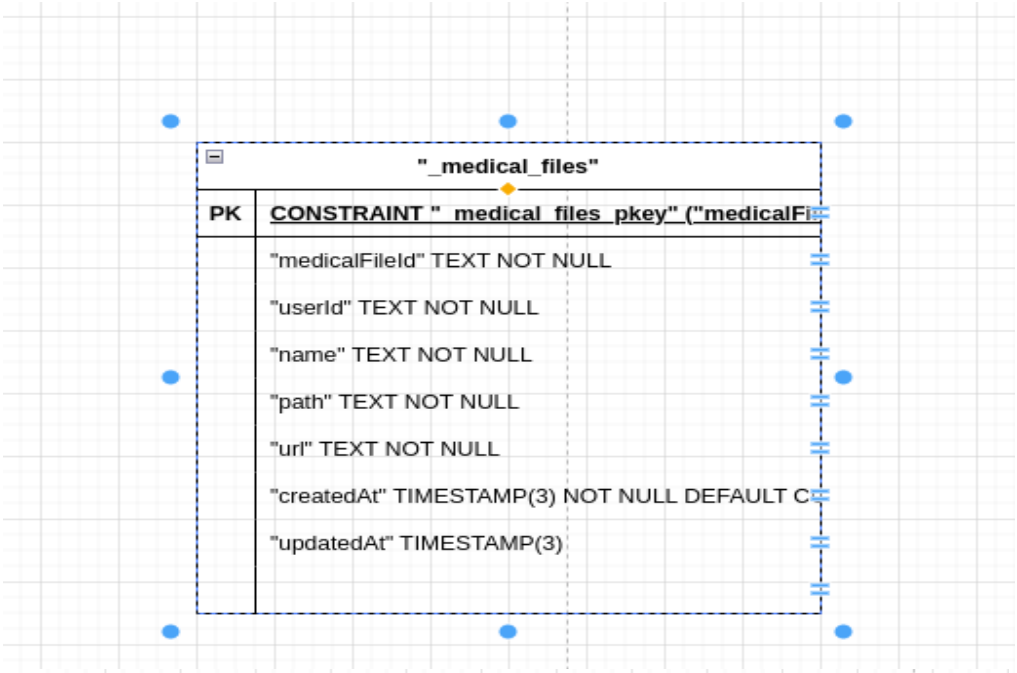
Deleting Medical Record File:

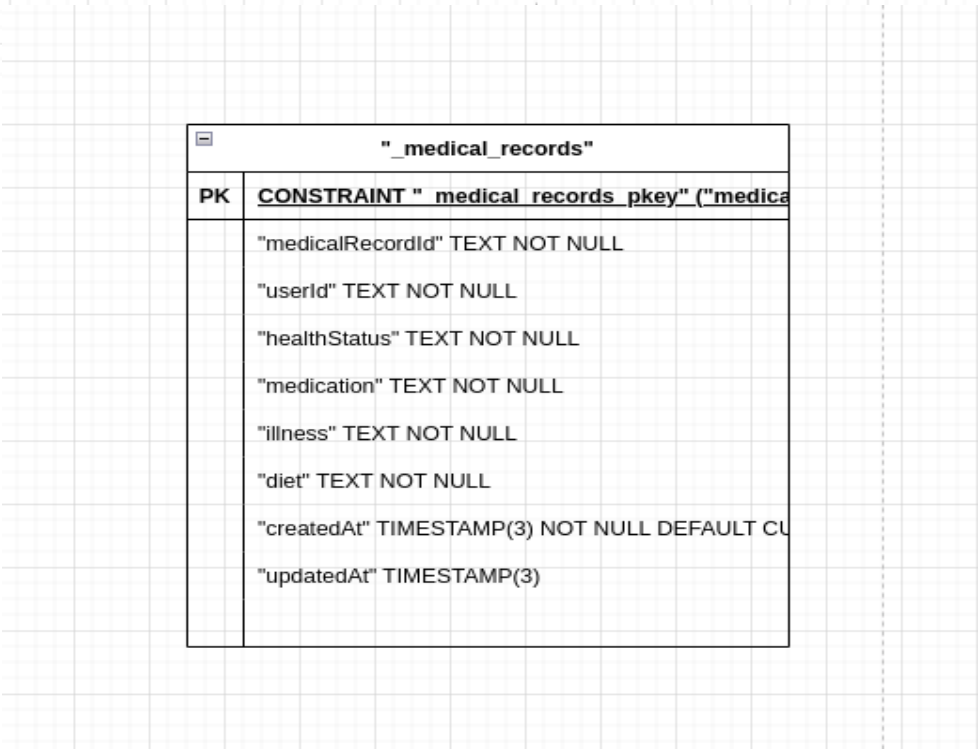
Parameters:

I.- medicalFileId: ID of the medical file to delete.

To delete a specific medical file, the medicalFileId must be provided. If the medicalFileId is missing, the request will be rejected.

Tables for medical files and records





3. Appointments and Schedules

Appointments

The Appointments feature in Doc Ease allows patients to schedule appointments with doctors, manage existing appointments, and facilitates communication between patients and doctors regarding appointment status

Creating Appointments:

Parameters:

I. - patientId: ID of the patient for whom the appointment is being created.

II. - doctorId: ID of the doctor with whom the appointment is scheduled.

III. - subject: Subject or reason for the appointment.

IV. - startsAt: Start time of the appointment.

V. - endsAt: End time of the appointment.

Before creating the appointment, we ensure that all mandatory fields including patientId, doctorId, subject, startsAt, and endsAt are provided. If any of these fields are missing, we then reject the request.

Updating Appointments:

Parameters:

I. - appointmentId: ID of the appointment to be updated.

II. - patientId: ID of the patient associated with the appointment.

III. - doctorId: ID of the doctor associated with the appointment.

IV. - subject: Updated subject or reason for the appointment.

V. - startsAt: Updated start time of the appointment.

VI. - endsAt: Updated end time of the appointment.

Before updating the appointment, we ensure that the appointmentId is provided. Additionally, we validate the presence of all mandatory fields including patientId, doctorId, subject, startsAt, and endsAt. If the appointmentId or any mandatory field is missing, we then reject the request.

Getting Appointment Details:

Parameter:

appointmentId: ID of the appointment to retrieve details.

To fetch details of a specific appointment, we require the appointmentId. Before retrieving the appointment details, we validate the presence of the appointmentId. If the appointmentId is missing, we then reject the request.

Getting Appointments by Doctor:

Parameter:

doctorId: ID of the doctor for whom appointments are to be retrieved.

To fetch appointments associated with a specific doctor, we require the doctorId. Before retrieving appointments, we validate the presence of the doctorId. If the doctorId is missing, we then reject the request.

Getting Appointments by Patient:

Parameters:

patientId: ID of the patient for whom appointments are to be retrieved.

To retrieve appointments associated with a specific patient, we require the patientId. Before fetching appointments, we validate the presence of the patientId. If the patientId is missing, we then reject the request.

Deleting Appointments:

Parameter:

I. **appointmentId:** ID of the appointment to be deleted.

Before deleting the appointment, we ensure that the appointmentId is provided. If the appointmentId is missing, we then reject the request.

Rescheduling Appointments:

Parameters:

I. - appointmentId: ID of the appointment to be rescheduled.

II. - patientId: ID of the patient associated with the appointment.

III. - doctorId: ID of the doctor associated with the appointment.

IV. - subject: Updated subject or reason for the appointment.

V. - startsAt: Updated start time of the appointment.

VI. - endsAt: Updated end time of the appointment.

Before rescheduling the appointment, we ensure that the appointmentId is provided. Additionally, we validate the presence of all mandatory fields including patientId, doctorId, subject, startsAt, and endsAt. If any of these fields is missing, we then reject the request.

Approving Appointments:

Parameters:

I. - appointmentId: ID of the appointment to be approved.

To approve a pending appointment, we require the appointmentId. Before approving the appointment, we check if the appointment exists and if it is pending approval. If the appointment is already approved or does not exist, we then reject the request.

Cancelling Appointments:

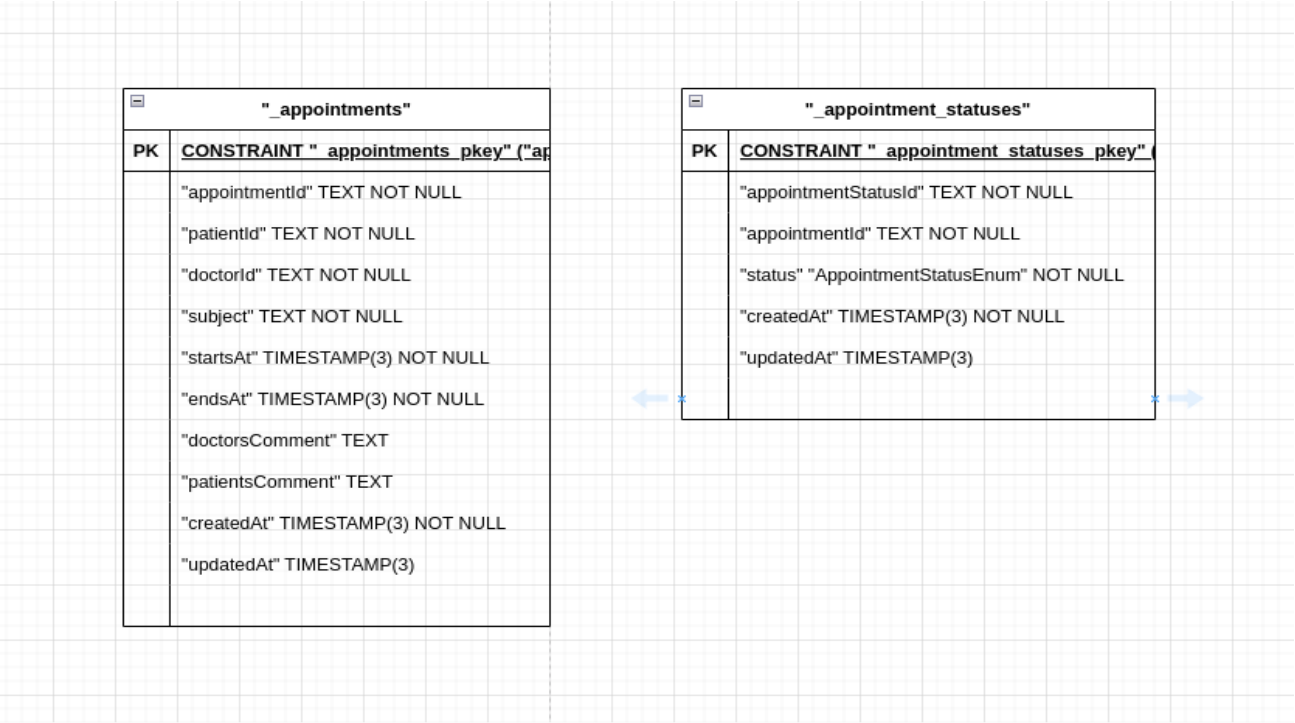
Parameters:

I. - appointmentId: ID of the appointment to be cancelled.

II. - doctorsComment (optional): Additional comment from the doctor regarding the cancellation.

To cancel an existing appointment, we require the appointmentId. Additionally, the doctor can optionally provide a comment regarding the cancellation. Before cancelling the appointment, we ensure that the appointment exists and is not already cancelled. If the appointment is already cancelled or does not exist, we then reject the request.

Tables for Appointments and appointment statuses



Schedules

The schedules feature allows doctors to specify the time frames they available for appointment during the course of the week

Creating Schedules:

Parameters:

I. - userId: ID of the user for whom the schedule is being created.

II. - weekday: Weekday for the schedule (MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY).

Before creating the schedule, we ensure that the userId and valid weekday are provided. If either of these fields is missing, we then reject the request. Additionally, we check if the weekday already has a schedule associated with it. If it does, we reject the request.

Getting Schedule Details:

Parameters:

I. - scheduleId: ID of the schedule to retrieve details.

To fetch details of a specific schedule, we require the scheduleId. Before retrieving the schedule details, we validate the presence of the scheduleId. If the scheduleId is missing, we then reject the request.

Getting Schedules by User

Parameters:

I. - userId: ID of the user for whom schedules are to be retrieved.

To fetch schedules associated with a specific user, we require the userId. Before retrieving schedules, we validate the presence of the userId. If the userId is missing, we then reject the request.

Deleting Schedules:

Parameters:

I. - scheduleId: ID of the schedule to be deleted.

Before deleting the schedule, we ensure that the scheduleId is provided. If the scheduleId is missing, we then reject the request.

Creating Schedule Time:

Parameters:

I. - scheduleId: ID of the schedule for which the time slot is being created.

II. - start: Start time of the schedule time slot.

III. - end: End time of the schedule time slot.

Before creating the schedule time slot, we ensure that the scheduled, start time, and end time are provided. If any of these fields is missing, we then reject the request.

Updating Schedule Time:

Parameters:

I. - scheduleTimeId: ID of the schedule time slot to be updated.

II. - start: Updated start time of the schedule time slot.

III. - end: Updated end time of the schedule time slot.

Before updating the schedule time slot, we ensure that the scheduleTimeId, start time, and end time are provided. If any of these fields is missing, we then reject the request.

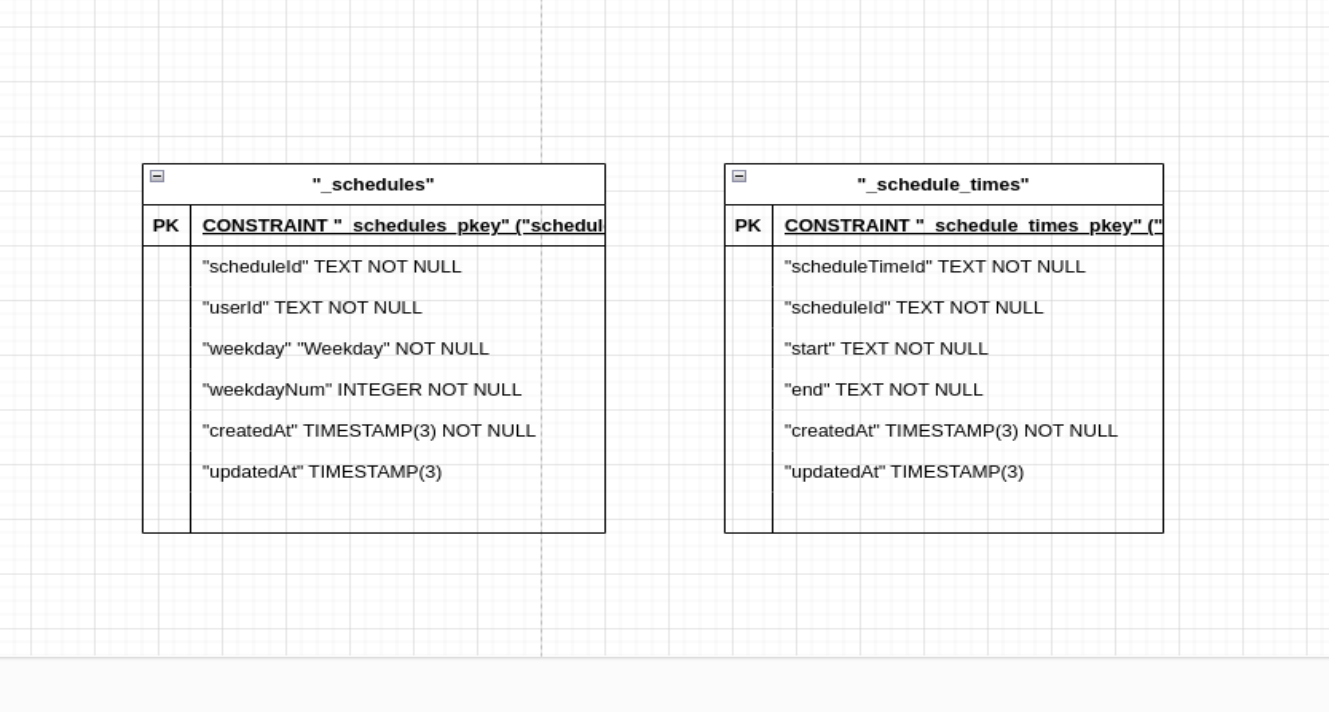
Deleting Schedule Time:

Parameters:

I. - scheduleTimeId: ID of the schedule time slot to be deleted.

Before deleting the schedule time slot, we ensure that the scheduleTimeId is provided. If the scheduleTimeId is missing, we then reject the request.

The tables for schedules and schedule times



4. **Mental Health Assessment**

The Mental Health Assessment feature allows users to undergo evaluations aimed at gauging their mental well-being. Through a series of questions, users provide responses that are then analyzed to generate insightful assessments. These assessments offer users valuable insights into their mental health status, aiding in self-awareness and potentially prompting further action if needed. With this feature, users can track their mental well-being over time and access personalized recommendations or support as necessary, fostering proactive mental health management. Powered by the **OpenAI API,** [11] the recommendations provided are informed by advanced natural language processing algorithms, ensuring the delivery of accurate and relevant insights tailored to each user's unique needs.

Posting Mental Health Assessment:

Parameters:

I. - userId: ID of the user for whom the mental health assessment is being created.

II. - answeredQuestions: JSON string representing the answered questions for the assessment and each includes an answer.

Before creating the mental health assessment, we ensure that the userId is provided. Additionally, we validate that at least 5 questions have been answered. If either the userId is missing or fewer than 5 questions have been answered, we reject the request.

After the all the necessary validation checks, we send the answered questions to the **OpenAI API** as a prompt to generate the recommendations based on the provided questions.

Getting Mental Health Assessment Details:

Parameters:

I. - mentalHealthId: ID of the mental health assessment to retrieve details.

To fetch details of a specific mental health assessment, we require the mentalHealthId. Before retrieving the assessment details, we validate the presence of the mentalHealthId. If the mentalHealthId is missing, we then reject the request.

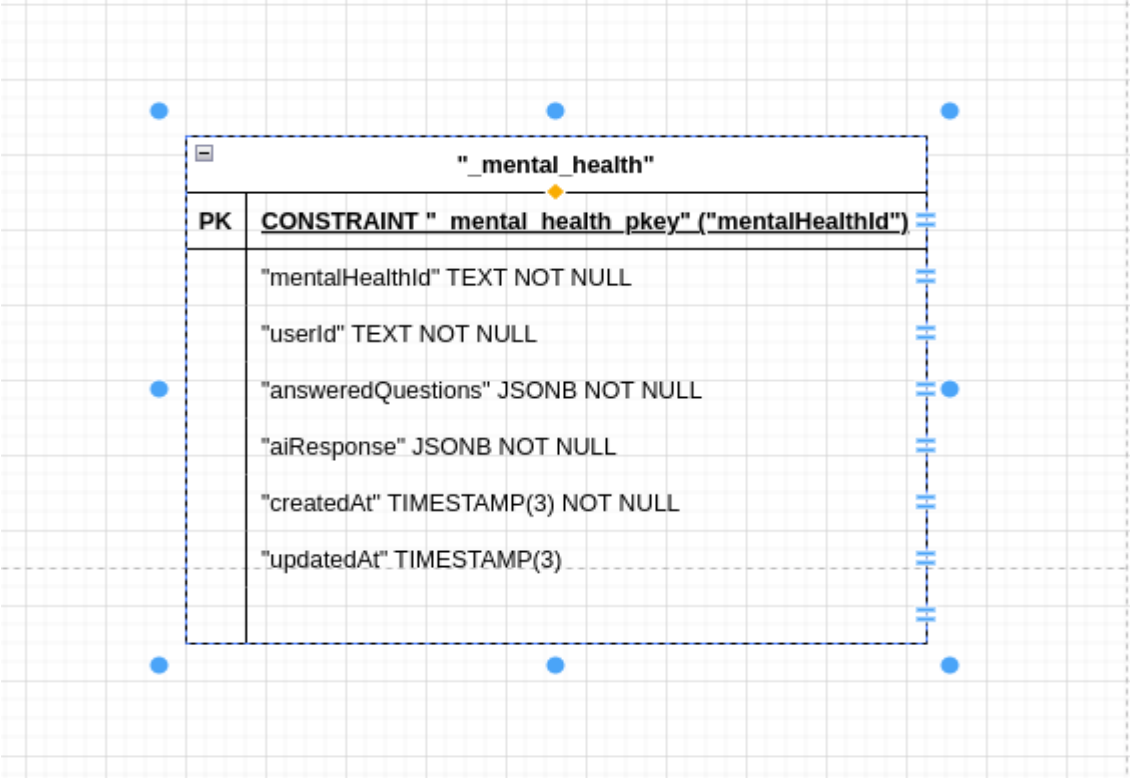
Getting Mental Health Assessments by User:

Parameters:

I. - userId: ID of the user for whom mental health assessments are to be retrieved.

To fetch mental health assessments associated with a specific user, we require the userId. Before retrieving assessments, we validate the presence of the userId. If the userId is missing, we then reject the request.

The table for mental health assessments



5. **Chat (Messenger)**

Our chat feature is powered by Server-Sent Events (SSE), providing users with a seamless real-time messaging experience. SSE enables bi-directional communication between the server and clients, allowing instant updates without the need for frequent polling.

With SSE, users can send and receive messages in real-time, creating a fluid and interactive chat environment. When a user sends a message, it is immediately delivered to the recipient, ensuring swift communication.

Moreover, SSE ensures that users receive live updates as soon as new messages arrive, without the need to constantly refresh the page or make repeated requests to the server. This not only enhances user experience but also conserves bandwidth and server resources.

Posting Chat Message:

Endpoint to create a new chat message.

Parameters:

I. - senderId: ID of the user sending the chat message.

II. - recipientId: ID of the user receiving the chat message.

III. - chatRoomId: ID of the chat room where the message is sent.

IV. - message: Content of the chat message.

Before creating the chat message, we ensure that all required parameters (senderId, recipientId, chatRoomId, message) are provided. If any of these parameters are missing, we reject the request.

After all the necessary validation checks, we create the chat message and notify the recipient using real-time notifications.

Getting Live Chat Updates:

Endpoint to receive live updates for chat messages.

Parameters:

I. - userId: ID of the user to receive live chat updates.

To receive live updates for chat messages, we establish a server-sent events connection with the client. The client must provide the userId to subscribe to updates.

We send periodic messages to keep the connection alive. Whenever a new chat message is received for the user, we send the message data to the client in real-time.

Getting Chat Recipients:

Endpoint to fetch recipients for the user's chat messages.

Parameters:

I. - userId: ID of the user to fetch chat recipients.

To fetch chat recipients for a user's chat messages, we require the userId. We retrieve all chat messages involving the user (either as sender or recipient), organize the recipients, and return the list of recipients along with their messages.

Getting Chat Messages by Chat Room:

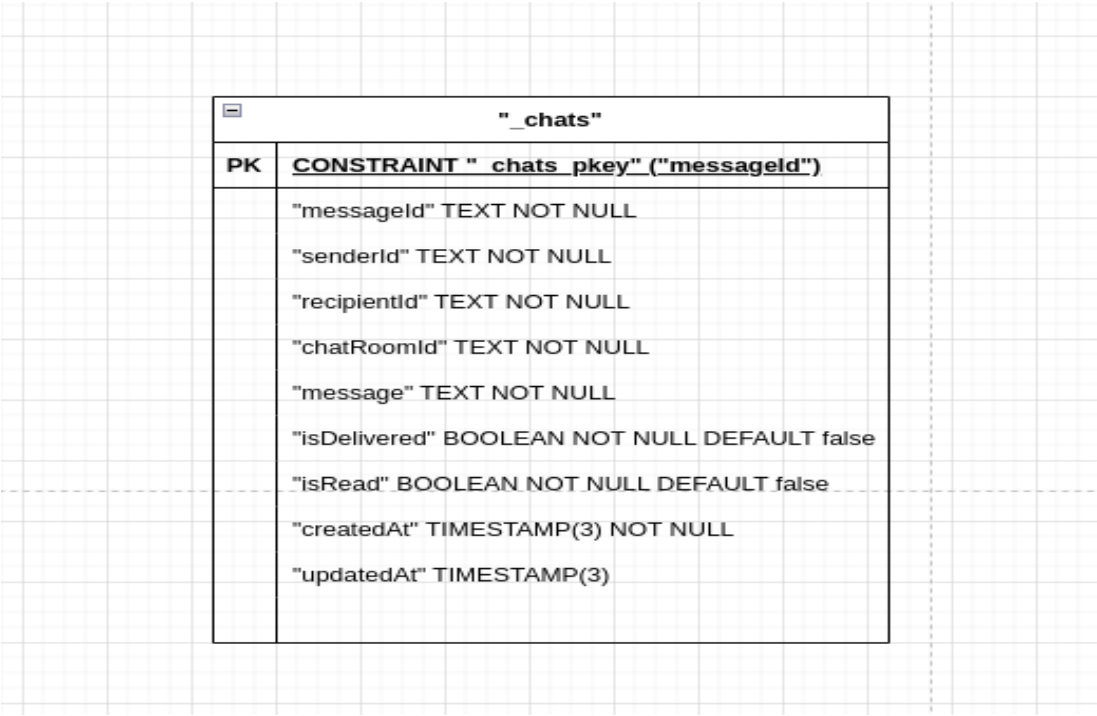
Endpoint to fetch chat messages by chat room.

Parameters:

I. - **chatRoomId:** ID of the chat room to fetch messages.

To fetch chat messages for a specific chat room, we require the chatRoomId. We retrieve all messages associated with the specified chat room and return them as a response.

Table for chats



6. **Notifications**

The notification feature facilitates real-time communication between the server and users by delivering timely updates and alerts. It employs server-sent events (SSE) to establish persistent connections, allowing the server to push notifications directly to clients as they occur.

With this feature, users can receive notifications instantly without the need for constant polling or manual refreshes. Notifications can include various types of information, such as messages, alerts, or updates related to appointments, messages, or system events.

By leveraging SSE, the notification feature enhances user experience by providing seamless, immediate access to important information. Whether it's receiving instant messages, updates on appointments, or alerts about system activities, users stay informed and engaged with the application in real time.

Posting Notification:

Parameters:

I. - userId: ID of the user to whom the notification is being sent.

II. - message: The content of the notification message.

To post a notification, both the userId and the message content are required. Upon receiving this information, the notification controller saves the notification in the database and proceeds to send it to the intended user.

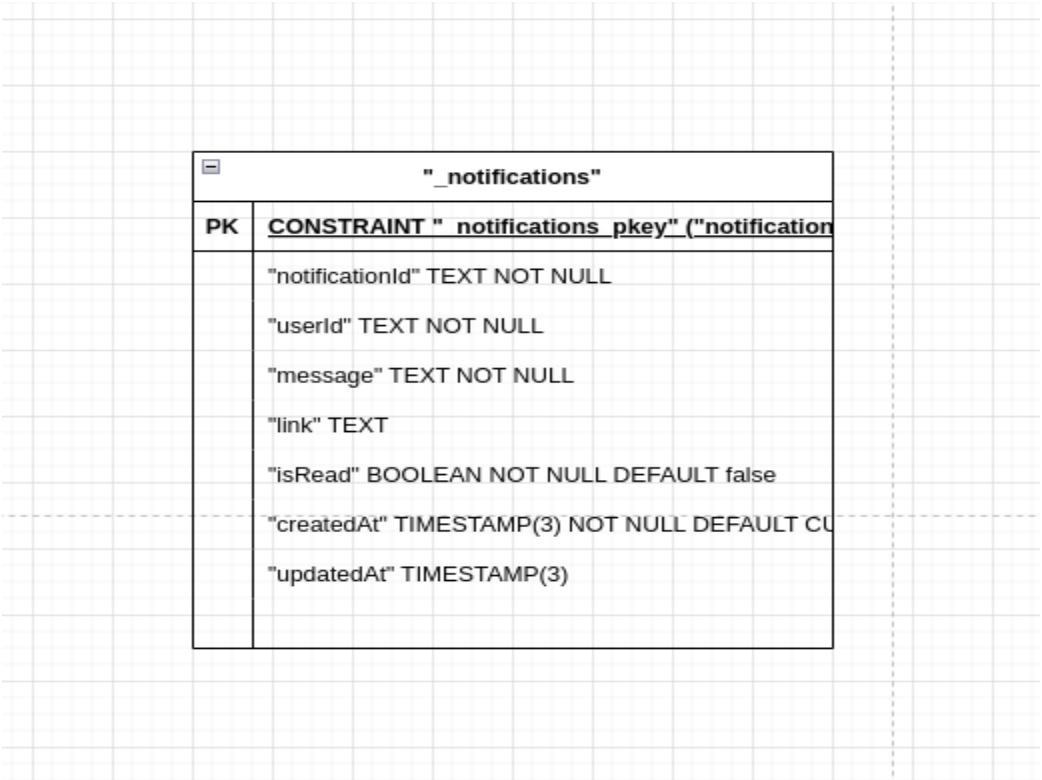
Retrieving Live Notifications:

Parameters:

I. - userId: ID of the user who is retrieving live notifications.

This endpoint establishes a server-sent event (SSE) connection to provide real-time notifications to the user identified by the userId. The connection remains open, and the server continuously sends notifications to the client as they occur. The endpoint also sends a heartbeat signal periodically to keep the connection active.

Table for notifications



2.3.2 Challenges faced during implementation and how they were addressed

Numerous unanticipated issues surfaced during the development phase, from hosting difficulties to integrating external APIs for health evaluation. Every issue was handled methodically, utilizing a blend of technical know-how, exhaustive testing, and calculated judgment. Through proactive identification and resolution of these difficulties, the development team made sure that Doc Ease was implemented successfully. The issues with Doc Ease implementation and the fixes made are listed below;

Handling Complexity (Complexity Management)

One of the biggest problems with the Doc Ease implementation was handling the system's complexity. It was essential to keep readability and simplicity in the code despite the wide range of functionality. The code base was efficiently arranged by segmenting it into smaller modules and components in order to tackle this difficulty. To effectively manage complexity, best practices including modularization, concern separation, and clean code principles were adhered to verification and Permission

Authentication and Authorization

Another major problem was implementing safe permission and authentication systems, particularly for sensitive data. Strong encryption methods were used to get around this, guaranteeing safe credential storage and appropriate management of access permissions. To improve user authentication security, industry-standard authentication protocols like JSON Web Tokens (JWT) were used.

Two-Factor Authentication (2FA)

There were additional difficulties in integrating Two-Factor Authentication (2FA) without making the authentication process more complicated. Token creation, validation, and expiration management for 2FA were included as part of the solution. Users were given a safe and easy way to authenticate, thanks to the seamless integration of 2FA into the procedure.

Appointment Management

It was difficult to create a strong appointment management system that could handle different situations. One way to ensure data consistency and real-time updates was to use reminders, rescheduling techniques, and efficient scheduling conflict resolution. Techniques for real-time data synchronization were put in place to give users the most recent information about their appointments.

Real-time Chatting

Initially, it was difficult to incorporate real-time chat features in an efficient manner. WebSocket connections were our first line of defence, however we ran into problems with resource

consumption. We decided to switch to Server-Sent Events (SSE) in order to improve scalability and dependability.

Health Assessment Feature using OPENAPI

There were challenges in integrating external APIs for health assessment, including OPENAPI, [11] especially with regard to error handling and API update compatibility. We put strong error handling, data validation, and versioning procedures in place to solve this. For smooth integration and dependability, we also depended on thorough API documentation and monitoring tools.

Real-time Notifications

It was difficult to create real-time alerts efficiently using push notifications and server-sent events. Building a solid infrastructure for message delivery, error handling, and device token management was our solution. To guarantee dependable real-time notice transmission, we made use of push notification systems like Firebase Cloud Messaging.

Hosting Challenges

It was really difficult to find a hosting platform that offered a stable free tier for hosting the backend. We choose to use the free tier provided by websites such as render.com in order to get around this problem. We deliberately used this alternative to meet our hosting needs, while we were aware of its limitations and possible reliability difficulties.

2.3.3 Key Algorithms and techniques used in the project.

Using React, Node.js, PostgreSQL, and Redux to create a full-stack application such as Doc Ease, several common algorithms and approaches are applied across several layers of the stack.

1. Authentication and Authorization: methods such as role-based access control (RBAC) for establishing user permissions, hashing passwords with algorithms like crypt, and safeguarding API endpoints with JWT (JSON Web Tokens).

2. Data Fetching and Manipulation: Filtering, sorting, and pagination algorithms for data retrieved from the database methods such as slow loading to enhance the efficiency of programs with large amounts of data.

3. State Management (with Redux): Effective algorithms for handling intricate state transitions. Nested data structures are organized in the Redux store using techniques like normalization.
memorization to prevent needless re-renders and cache the output of costly computations.

4. Form Handling: Regular expressions and libraries like Yup are examples of algorithms for form validation methods for managing form submission and providing users with error feedback.

5. Routing: Using frameworks like React Router, algorithms are used to define and manage client-side routes methods for working with route guards and nested routes.

6. Error management: Server-side algorithms for centralized error management, like Node.js middleware functions methods for the client-side application to show users informative error messages.

7. Database Operations (using PostgreSQL): Query and indexing techniques for database tables that are efficient methods for managing concurrency and transactions.
sophisticated SQL methods for intricate data aggregation and processing.

8. Real-time Communication: Server-Sent Events and other technologies are used in algorithms to implement real-time functionality. Conflict resolution and real-time data synchronization techniques.

2.4 Testing and Evaluation

2.4.1 Testing methodology

Unit tests were a key component of our testing strategy, which we used to verify the dependability and functioning of each system component. We conducted unit tests to confirm the behavior of particular methods, functions, or classes separately. This helped us find and fix any possible problems early in the development process. We streamlined our quality assurance efforts and automated the testing process by utilizing unit testing frameworks.

2.4.2 Test results and metrics

We used unit tests during the testing process to assess the functionality and accuracy of different parts of the system. Details including test case descriptions, input data, expected outcomes, and actual outcomes were included in the test results documentation. In addition, we evaluated the system's general robustness and quality by measuring important metrics including error rates, test execution times, and code coverage. These metrics helped us make important decisions about

future iterations and enhancements by offering insightful information about how well our testing efforts worked.

2.4.3 Evaluation against requirements and objectives

After testing was finished, we thoroughly assessed the system in comparison to the project's predetermined goals and needs. We evaluated the degree to which each condition and goal was met by methodically contrasting the actual system behavior with the established criteria. Overall, the assessment verified that the system accomplished the stated goals and requirements, showcasing its capacity to support mental health assessment, improve medical record administration, and enable remote healthcare access. The assessment also identified areas for improvement and areas for strength, providing guidance for next projects and developments.

CHAPTER THREE

3.1 Results and Discussions

3.1.1 Presentation of the project outcomes

The project's execution produced noteworthy results that matched the stated goals. The telehealth platform effectively facilitated access to remote healthcare, improved the management of medical records, and encouraged mental health evaluation. Remote medical consultations were available to users, lowering obstacles to healthcare and guaranteeing prompt access to licensed specialists. Healthcare providers could easily access patient health information through the secure Electronic Medical Records (EMR) system, which also made it easier for patients to make well-informed decisions. The UI was easy to use and made setting appointments and sending messages possible for distant consultations. Furthermore, the incorporation of a mental health evaluation function enabled users to examine their mental health.

3.1.2 Analysis of results in relation to project goals

The project results were examined in light of the predetermined objectives, indicating alignment and success in several important areas. The objective of facilitating effective communication between users and healthcare experts was achieved by the successful installation of real-time chat capability through the use of Server-Sent Events (SSE) and WebSocket connections. By integrating with external APIs such as OpenAI API for health assessment, the goal of offering sophisticated features for mental health assessment and assistance was achieved. Furthermore, a seamless user experience across many platforms and devices was guaranteed by the strong infrastructure for real-time notifications and cross-platform compatibility. Notwithstanding obstacles like hosting constraints, the project successfully accomplished its goals and provided a complete telehealth solution.

3.2 Discussion

3.2.1 Future Implementation of the project

Looking ahead, the telehealth platform can be further enhanced by a number of upcoming deployments and enhancements.
First off, incorporating video conferencing features would make it possible to have more engaging distant consultations, which would improve patient participation and care quality.
Second, adding file sharing capabilities to chat sessions would make it easier for users and healthcare practitioners to exchange reports and medical papers, which would streamline communication.

Thirdly, monitoring and displaying a user's online state would increase visibility and enable prompt response from users.
Fourth, making notifications audible would improve user responsiveness by giving users auditory cues for critical warnings.

Lastly, alerting consumers to inadequate network connectivity will facilitate problem-solving and communication continuity, guaranteeing continuous access to healthcare services. These upcoming initiatives seek to improve the telehealth platform's accessibility and functionality even further, increasing its influence on patient outcomes and healthcare delivery.

CHAPTER FOUR

4.1 Conclusion

4.1.1 Summary of key findings

The telehealth platform project has made great strides toward transforming healthcare administration and access. The successful deployment of elements including real-time communication, medical record management, and mental health assessment is indicated by key findings. The project's goals of increasing medical record management, encouraging mental health awareness, facilitating real-time communication, and improving healthcare accessible have all been successfully attained through the use of cutting-edge technologies and reliable procedures.

4.1.2 Reflection on the project’s success in meeting objectives

It is clear that the created platform has the ability to address important healthcare issues, especially in rural and neglected areas. The platform's usability and efficacy in providing remote healthcare services have been facilitated by the smooth integration of many functionalities and an intuitive UI. Furthermore, the solution's dependability and applicability were guaranteed by the project's adherence to best practices in software development.

4.2 Challenges faced

Complexity management, authentication and authorization, two-factor (2FA), appointment management, real-time chat, the Health Assessment Feature using OPENAPI, real-time notifications, and hosting challenges were among the difficulties encountered during the Doc Ease Project's implementation, as was previously mentioned in chapter 2.

4.3 Recommendations and Future Work

4.3.1 Suggestions for further improvements or future work

Suggestions include Improved video conferencing features for more engaging remote consultations; file sharing functionality integrated within chat to facilitate communication; online status tracking features implemented; sound notifications added for improved user experience; and network connectivity issues resolved are some of the recommendations. Furthermore, continual modifications, feedback gathering, and ongoing monitoring will be necessary to keep the platform relevant and useful in changing healthcare environments.

References & Bibliography

References

[1]. Ministry of Health, Uganda. (2022). Uganda Demographic and Health Survey.

[2]. Smith, J., & Johnson, A. (2023). Exploring the landscape of telehealth technologies and methodologies: A comprehensive literature review [Literature review]. In Brown, C. (Ed.), *Advancements in Telemedicine and Healthcare in* (pp. 45-67). Academic Press.

[3]. White, L., & Garcia, M. (2023). Emerging technologies and methodologies in telehealth: A review of the literature [Literature review]. In Jackson, R. (Ed.), *Innovations in Healthcare: Trends and Challenges* (pp. 89-104). Springer.

[4]. Patel, R., & Chen, S. (2023). Frameworks and methodologies for telehealth platform development: A comprehensive review [Literature review]. In Lee, H. (Ed.), *Digital Health Technologies: Advances and Applications (pp. 121-138).* Wiley.

[5]. React. Create a New React App [Step-by-step guide]. In React Documentation (Facebook), pp. 1-10. Legacy.reactjs.org. Retrieved fromhttps://legacy.reactjs.org/docs/create-a-new-react-app.html.

[6]. React Router. Overview [Introduction to React Router]. In React Router Documentation (React Training), pp. 1-8. Reactrouter.com. Retrieved from https://reactrouter.com/en/main/start/overview.

[7]. TypeScript. Everyday Types [Comprehensive guide to TypeScript everyday types]. In TypeScript Handbook (Microsoft), pp. 1-15. Typescriptlang.org. Retrieved from https://www.typescriptlang.org/docs/handbook/2/everyday-types.html.

[8]. Express.js. API Documentation [Official Express.js API documentation]. In Express.js Documentation (Express.js Contributors), pp. 1-20. Expressjs.com. Retrieved from https://expressjs.com/en/4x/api.html.

[9]. Prisma ORM. PostgreSQL Overview [Detailed overview of Prisma ORM for PostgreSQL databases]. In Prisma Documentation (Prisma), pp. 1-18. Prisma.io. Retrieved from https://www.prisma.io/docs/orm/overview/databases/postgresql.

[10]. Redux Toolkit. Getting Started [Comprehensive guide to getting started with Redux Toolkit]. In Redux Toolkit Documentation (Redux.js), pp. 1-14. Redux-toolkit.js.org. Retrieved from https://redux-toolkit.js.org/introduction/getting-started.

[11]. OpenAI. API Reference [Official API reference for OpenAI's platform]. In OpenAI Documentation (OpenAI), pp. 1-25. Platform.openai.com. Retrieved from https://platform.openai.com/docs/api-reference/making-requests.

[12]. SendGrid. QuickStart with Node.js [Step-by-step guide to using SendGrid with Node.js]. In SendGrid Documentation (SendGrid), pp. 1-8. Docs.sendgrid.com. Retrieved from https://docs.sendgrid.com/for-developers/sending-email/quickstart-nodejs.

[13]. Netlify Blog. Deploy React Apps in Less Than 30 seconds [Quick and easy deployment guide for React apps on Netlify]. In Netlify Documentation (Netlify), pp. 1-5. Netlify.com. Retrieved from https://www.netlify.com/blog/2016/07/22/deploy-react-apps-in-less-than-30-seconds.

[14]. Tailwind CSS. Installation [Step-by-step guide to installing Tailwind CSS]. In Tailwind CSS Documentation (Tailwind Labs), pp. 1-8. Tailwindcss.com. Retrieved from https://tailwindcss.com/docs/installation.

[15]. Firebase. Node.js Reference [Official reference for Firebase Node.js SDK]. In Firebase Documentation (Firebase), pp. 1-15. Firebase.google.com. Retrieved from https://firebase.google.com/docs/reference/node/firebase.storage.

[16]. Firebase. JavaScript Client [Official guide to using Firebase JavaScript client for Cloud Messaging]. In Firebase Documentation (Firebase), pp. 1-10. Firebase.google.com. Retrieved from https://firebase.google.com/docs/cloud-messaging/js/client.

Bibliography:

[1]. React. Create a New React App [Step-by-step guide]. In React Documentation (Facebook), pp. 1-10. Legacy.reactjs.org. Available: https://legacy.reactjs.org/docs/create-a-new-react-app.html.

[2]. React Router. Overview [Introduction to React Router]. In React Router Documentation (React Training), pp. 1-8. Reactrouter.com. Available: https://reactrouter.com/en/main/start/overview.

[3]. TypeScript. Everyday Types [Comprehensive guide to TypeScript everyday types]. In TypeScript Handbook (Microsoft), pp. 1-15. Typescriptlang.org. Available: https://www.typescriptlang.org/docs/handbook/2/everyday-types.html.

[4]. Express.js. API Documentation [Official Express.js API documentation]. In Express.js Documentation (Express.js Contributors), pp. 1-20. Expressjs.com. Available: https://expressjs.com/en/4x/api.html.

[5]. Tan Stack. React Framework Overview [In-depth overview of the Tan Stack React framework]. In Tan Stack Documentation (Tan Stack Contributors), pp. 1-12. Tanstack.com. Available: https://tanstack.com/query/v4/docs/framework/react/overview.

[6]. Render. Deploy a Docker Image [Step-by-step guide to deploying Docker images with Render]. In Render Documentation (Render Contributors), pp. 1-6. Docs.render.com. Available: https://docs.render.com/deploy-an-image.

[7] Prisma ORM. PostgreSQL Overview [Detailed overview of Prisma ORM for PostgreSQL databases]. In Prisma Documentation (Prisma Contributors), pp. 1-18. Prisma.io. Available: https://www.prisma.io/docs/orm/overview/databases/postgresql.

[8] Redux Toolkit. Getting Started [Comprehensive guide to getting started with Redux Toolkit]. In Redux Toolkit Documentation (Redux.js Contributors), pp. 1-14. Redux-toolkit.js.org. Available: https://redux-toolkit.js.org/introduction/getting-started.

[9]. OpenAI. API Reference [Official API reference for OpenAI's platform]. In OpenAI Documentation (OpenAI Contributors), pp. 1-25. Platform.openai.com. Available: https://platform.openai.com/docs/api-reference/making-requests.

[10]. SendGrid. QuickStart with Node.js [Step-by-step guide to using SendGrid with Node.js]. In SendGrid Documentation (SendGrid Contributors), pp. 1-8. Docs.sendgrid.com. Available: https://docs.sendgrid.com/for-developers/sending-email/quickstart-nodejs.

[11]. Mozilla Developer Network. Server-sent Events [Comprehensive guide to using Server-sent Events]. In MDN Web Docs (Mozilla Contributors), pp. 1-10. Developer.mozilla.org. Available: https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events.

[12]. Netlify Blog. Deploy React Apps in Less Than 30 seconds [Quick and easy deployment guide for React apps on Netlify]. In Netlify Documentation (Netlify Contributors), pp. 1-5.

Netlify.com. Available: https://www.netlify.com/blog/2016/07/22/deploy-react-apps-in-less-than-30-seconds.

[13]. Tailwind CSS. Installation [Step-by-step guide to installing Tailwind CSS]. In Tailwind CSS Documentation (Tailwind Labs), pp. 1-8. Tailwindcss.com. Available: https://tailwindcss.com/docs/installation.

[14]. Mozilla Developer Network. Model-View-Controller (MVC) [Detailed explanation of the Model-View-Controller architectural pattern]. In MDN Web Docs (Mozilla Contributors), pp. 1-10. Developer.mozilla.org. Available: https://developer.mozilla.org/en-US/docs/Glossary/MVC.

[15]. Firebase. Node.js Reference [Official reference for Firebase Node.js SDK]. In Firebase Documentation (Firebase Contributors), pp. 1-15. Firebase.google.com. Available: https://firebase.google.com/docs/reference/node/firebase.storage.

[16]. Firebase. JavaScript Client [Official guide to using Firebase JavaScript client for Cloud Messaging]. In Firebase Documentation (Firebase Contributors), pp. 1-10. Firebase.google.com. Available: https://firebase.google.com/docs/cloud-messaging/js/client.

Appendices

[A]. Link to live version of the application: "Deceased - Live Version," Netlify. Available: https://docease-v2.netlify.app/.

[B]. Link to Doc Ease features (detailed) documentation on Google Drive: “Doc Ease Project Features Documentation,” Google Drive. Available:

https://drive.google.com/file/d/1qcIP4OPG-v4ShOsRTgof0XmQP7YfFGQd/view?usp=sharing.

[C]. Link to Doc Ease Wireframe on Google Drive: "Doc Ease Wireframe," Google Drive. Available:https://drive.google.com/file/d/1LHMFYKdLagUqyQ9k2Mg6TwjT2-wjCxRh/view?usp=sharing.

[D]. Link to Doc Ease prototype: "Doc Ease Prototype," Figma.com. Available: https://www.figma.com/file/KFnPSkGx4NzGpDLEuMwT30/DOC-EASE-V2?type=design&node-id=0-1&mode=design&t=gsznA0ZuS746ghIO-0.

[E]. Link to Doc Ease backend: "Doc Ease Backend," GitHub.com. Available: https://github.com/MogaMAW/docease-backendv2.

[F]. Link to Doc Ease frontend: "Doc Ease Frontend," GitHub.com. Available: https://github.com/Doc-Ease/docease-frontend.

[G]. Link to Database Tables on Google Drive: "Doc Ease Database Tables," Google Drive. Available: https://drive.google.com/file/d/1cw2LPK8mXHuv2a_dpgtynmaqgvmM19RD/view?usp=sharing

[H]. Link to System architecture Diagram on Google Drive: "Doc Ease System Architecture Diagram,-"Google-Drive.-Available: https://drive.google.com/file/d/1AcQYQJhcQf0apY6riaAzygZvSddo3mza/view?usp=sharing.

[I]. Link to Doctors flow Chart flow on Google Drive: "Doc Ease Doctors Flow Chart," Google Drive.-Available: https://drive.google.com/file/d/11B4QIwFr88SC9fohytsJMyZLiydRN_cv/view?usp=sharing.

[J]. Link to Patients flow chart on Google Drive: "Doc Ease Patients Flow Chart," Google Drive. Available: https://drive.google.com/file/d/1tNe86txWlcvd0WHsGqfRKH1kFEnxR0vr/view?usp=sharing.