POZA SPECIALIST CLINIC APP

Contents

[Table of Figures 3](#_Toc136386903)

[Introduction 4](#_Toc136386904)

[Abstract 4](#_Toc136386905)

[Background 4](#_Toc136386906)

[Literature Review 4](#_Toc136386907)

[Research Objectives 5](#_Toc136386908)

[System Development Objectives 6](#_Toc136386909)

[Problem Statement 6](#_Toc136386910)

[Goals 7](#_Toc136386911)

[Deliverables 8](#_Toc136386912)

[Research Methodology: 8](#_Toc136386913)

[User Requirements 11](#_Toc136386914)

[Functional Requirements 12](#_Toc136386915)

[Non-Functional Requirements 13](#_Toc136386916)

[Project Schedule 13](#_Toc136386917)

[Dependencies 14](#_Toc136386918)

[Gantt Chart and Work Breakdown Structure 14](#_Toc136386919)

[Constraints 17](#_Toc136386920)

[System Analysis and Design 19](#_Toc136386921)

[Overview 19](#_Toc136386922)

[Design Approach 19](#_Toc136386923)

[Data Flow Diagrams 21](#_Toc136386924)

[a. Sign-Up Process 21](#_Toc136386925)

[b. Booking Process 22](#_Toc136386926)

[c. Doctor Registration 24](#_Toc136386927)

[d. Medical Chatbot Interaction 25](#_Toc136386928)

[e. Analytics Dashboard 26](#_Toc136386929)

[Class Diagrams 26](#_Toc136386930)

[The Chat Class 27](#_Toc136386931)

[The Persona Class 27](#_Toc136386932)

[The Doctor Class 28](#_Toc136386933)

[Appointment Class 28](#_Toc136386934)

[Notification Class 28](#_Toc136386935)

[Overview Class Diagram 29](#_Toc136386936)

[Database Schema and Entity-Relationship Diagrams 30](#_Toc136386937)

[Entity-Relationship Diagram 36](file:///C:\Users\Tevin\Documents\poza\docs\POZA%20SPECIALIST%20CLINIC%20APP.docx#_Toc136386938)

[Use Case Scenario 37](#_Toc136386939)

[Booking App 37](#_Toc136386940)

[Chat App 37](#_Toc136386941)

[Members App 37](#_Toc136386942)

[Coding 38](#_Toc136386943)

[User Profile Class Models and Interface Managers 38](#_Toc136386944)

[References 39](#_Toc136386945)

# Table of Figures

[Figure 1:Gantt Chart Task 1 to 2.2 7](file:///C:\Users\Tevin\Documents\poza\docs\POZA%20SPECIALIST%20CLINIC%20APP.docx#_Toc135939288)

[Figure 2:Task 2 to 3 8](file:///C:\Users\Tevin\Documents\poza\docs\POZA%20SPECIALIST%20CLINIC%20APP.docx#_Toc135939289)

[Figure 3:Task 3 to 4 8](file:///C:\Users\Tevin\Documents\poza\docs\POZA%20SPECIALIST%20CLINIC%20APP.docx#_Toc135939289)

[Figure 4:Task 4 to 6 8](file:///C:\Users\Tevin\Documents\poza\docs\POZA%20SPECIALIST%20CLINIC%20APP.docx#_Toc135939289)

[Figure 5:Task 6 to 7 9](file:///C:\Users\Tevin\Documents\poza\docs\POZA%20SPECIALIST%20CLINIC%20APP.docx#_Toc135939289)

# Introduction

## Abstract

This Django-powered web application enhances the medical sector by integrating traditional and emerging technologies. Users can book appointments for specialized health services and engage with a medical chatbot driven by OpenAI's GPT-3.5. The application provides a comprehensive platform for doctors and administrators to manage appointments and communicate with patients. Additionally, an analytics dashboard presents graphical insights based on appointment data, including prices, services, and scheduling information. By combining these features, the project optimizes the clinic booking process and improves healthcare accessibility and efficiency.

## Background

Telemedicine, utilizing electronic information and telecommunication technology, has emerged as a promising solution to address challenges in the medical sector. It offers improved access to care, reduced wait times, and enhanced communication between patients and healthcare providers. This programming project focuses on developing an online clinic that leverages telemedicine and digital interfaces to provide convenient healthcare services, particularly targeting urban areas. The project introduces a medical chatbot, named MediBot, powered by the GPT 3.5 Turbo API, to assist patients with health-related queries. Patients can book appointments for various services, and a fair scheduling algorithm assigns doctors based on service-specific availability. A payment module integrated with Stripe's API enables secure and seamless transactions. Doctors and administrative staff have dedicated panels to manage appointments, update patient notes, and monitor upcoming and past appointments. An analytics dashboard offers insights into patient sign-ups, service appointments, and cumulative pricing trends. The project aims to enhance the online booking experience and improve access to quality healthcare through the adoption of telemedicine and digital interfaces while preserving the essential role of human expertise in the medical field.

## Literature Review

Technology has emerged as a powerful tool to address the challenges faced by health clinics, as evidenced by various studies and real-world examples. In the context of appointment booking systems, traditional processes can be tedious and time-consuming, leading to patient dissatisfaction and inefficiencies. For instance, the booking site of Kenyatta National Hospital in Kenya is known for its complex and convoluted booking process, requiring multiple steps and forms to be filled out, leading to frustration among patients.

On the other hand, there are inspiring examples of seamless booking systems that have significantly improved the patient experience. Aga Khan University Hospital has implemented a user-friendly and intuitive booking form, streamlining the entire process. Their system allows patients to easily select desired services, preferred dates, and available time slots, minimizing the effort required to secure an appointment. This seamless integration simplifies the booking process, reduces wait times, and enhances patient satisfaction.

Mater Hospital has adopted a hybrid approach, combining static appointment times advertised on their website with the option for in-person or phone call bookings. This method provides flexibility for patients who prefer the convenience of online booking or those who may have specific requirements that necessitate direct communication. By offering multiple channels for appointment bookings, Mater Hospital caters to different patient preferences, improving accessibility and accommodating a wider range of individuals.

These real-world examples highlight the importance of implementing user-friendly and efficient booking systems in healthcare institutions. The use of technology to streamline the appointment booking process is crucial in reducing patient frustration, optimizing resource allocation, and improving overall operational efficiency. By adopting intuitive interfaces, simplifying steps, and providing multiple booking channels, healthcare institutions can enhance the patient experience and increase appointment accessibility.

Moreover, it is essential to consider the integration of advanced features within these systems. For instance, incorporating automated appointment reminders and notifications can help reduce no-show rates and improve attendance. [Sending reminders via email, SMS, or in-app notifications can ensure that patients are well-informed about their scheduled appointments, reducing the likelihood of missed appointments and optimizing clinic utilization.](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9720268/)

In conclusion, the literature and real-world examples demonstrate the significance of implementing efficient and user-friendly booking systems in healthcare institutions. While some institutions, like Kenyatta National Hospital, struggle with complex and time-consuming processes, others, such as Aga Khan University Hospital and Mater Hospital, have successfully streamlined their appointment booking systems. [By leveraging technology and incorporating intuitive interfaces, multiple booking channels, and advanced features, healthcare institutions can improve patient satisfaction, optimize resource allocation, and enhance overall healthcare delivery.](https://travel.mypocketdoctor.com/2021/09/06/telemedicine-use-in-kenya/)

## Research Objectives

1. To understand the specific requirements and preferences of patients, doctors, and administrators when utilizing a specialist booking website.
2. To assess the usability and user satisfaction of different user interface designs employed in the booking website.
3. To evaluate the performance and scalability of the booking website under varying levels of user traffic.
4. To identify and analyze the potential security risks involved in storing and transmitting sensitive patient and doctor information within the booking website.
5. To explore the potential advantages and opportunities of integrating the booking website with other systems, such as electronic health records or payment systems, to enhance functionality and efficiency.

## System Development Objectives

1. Designing a user-friendly interface that effectively caters to the requirements of patients, doctors, and administrators.
2. Implementing comprehensive features and functionality that enable patients to easily book appointments, doctors to efficiently manage appointments and take notes, and administrators to access analytics and effectively manage doctor accounts.
3. Ensuring the security and privacy of patient and doctor information through the implementation of robust security measures and protocols.
4. Conducting thorough testing of the booking website to identify and address any defects or issues, ensuring it meets all specified requirements and operates smoothly.
5. Deploying the booking website on a reliable and scalable platform capable of handling a substantial number of users, ensuring optimal performance and availability

## Problem Statement

In the medical sector, there is a need for a comprehensive solution that combines traditional and emerging technologies to enhance patient care. This project aims to address this need by developing a web application that enables patients to conveniently book appointments for specific health services. The application incorporates an efficient scheduling algorithm that automatically assigns the least-busy doctor to optimize the appointment process. Additionally, patients can interact with a medical chatbot powered by OpenAI's GPT-3.5 to obtain answers to medical-related queries.

A crucial functional requirement of the system is effective appointment management and communication with patients. Doctors and administrators should be able to access and update appointment information, while patients should receive notifications about their appointments and have access to their past and upcoming appointments. Furthermore, the application includes an analytics dashboard page that presents graphical representations of appointment data.

The project's acceptance criteria include:

* The application must allow patients to book appointments for specific health services.
* The application must employ an efficient scheduling algorithm to automatically assign the least-busy doctor.
* Patients should be able to interact with a medical chatbot powered by OpenAI's GPT-3.5.
* Doctors and administrators must be able to manage appointments and communicate with patients effectively.
* The application should include an analytics dashboard page displaying graphs based on appointment data.

This project can be deployed in various healthcare settings such as health clinics, hospitals, and other healthcare facilities, providing a valuable resource for improving the overall patient experience.

## Goals

1. Develop a user-friendly platform that allows patients to easily book appointments with specialist doctors, providing a seamless and convenient booking experience.
2. Implement a comprehensive appointment management system for doctors, enabling them to efficiently manage their appointments, update patient notes, and maintain a streamlined workflow.
3. Create an analytics dashboard for administrators, providing them with valuable insights and data on appointment trends, resource allocation, and overall clinic performance.
4. Ensure the highest standards of security and privacy for patient and doctor information by implementing robust security measures, including data encryption, access control, and adherence to relevant regulatory requirements.
5. Build a reliable and scalable system architecture that can effectively handle a large influx of users, ensuring optimal performance and availability during peak periods.

## Deliverables

1. **Fully Functional Specialist Booking Website**: A complete website that allows patients to book appointments with specialists. The website should have a user-friendly interface and should handle all the necessary functionalities, such as displaying available doctors and their schedules, allowing patients to book appointments, and sending notifications to both patients and doctors.
2. **User Manuals**: Detailed user manuals for patients, doctors, and admins. These manuals should provide step-by-step instructions on how to use the website, including registration, appointment booking, profile management, and any other relevant features. The manuals should be clear, concise, and easy to understand.
3. **Technical Documentation**: Comprehensive technical documentation for developers and system administrators. This documentation should cover the overall architecture of the system, database schema, API documentation (if applicable), deployment instructions, system requirements, and any other relevant technical information. The documentation should be well-organized and provide sufficient details for developers and administrators to understand and work with the system.
4. **Admin Dashboard**: A dedicated dashboard for admins to view analytics and manage doctor accounts. The dashboard should provide an overview of key metrics and statistics related to appointment bookings, patient-doctor interactions, and any other relevant data. Additionally, the dashboard should allow admins to manage doctor accounts, including adding new doctors, updating their information, and managing their availability.
5. **Security and Privacy Compliance**: Ensure that the system meets all security and privacy requirements. Implement appropriate measures to protect user data, including secure authentication and authorization mechanisms, data encryption, and adherence to relevant regulations and best practices. Perform thorough security testing to identify and address any vulnerabilities in the system.

## Research Methodology:

To gather information for the development of the specialist booking website and telemedicine application, a comprehensive research methodology was adopted. The methodology involved a combination of observational study, interviews, and a questionnaire survey. The key steps and methods employed are outlined below:

1. **Observational Study**

* Visiting clinics: Several clinics in Nairobi, including Medanta in Westlands, Getrudes in Kileleshwa, and Mater Hospital in South B, were visited to observe their operations and the processes involved in patient appointment booking and management.
* Help desk observation: The help desk at the reception area of the clinics was observed to understand how appointments are scheduled, managed, and coordinated.
* Nursing council interview: A leading figure in the nursing council was interviewed to gain valuable insights into the current telemedicine applications in Nairobi. This interview helped understand the challenges, opportunities, and best practices in telemedicine implementation.

1. **Questionnaire Survey**

* Designing the questionnaire: A questionnaire was developed to gather data from various stakeholders, including patients, doctors, administrators, and other relevant personnel involved in the healthcare system.
* Open-ended questions: Open-ended questions were included to allow respondents to provide detailed and contextual information about their experiences, challenges, and expectations regarding the appointment booking process and telemedicine.
* Close-ended questions: Close-ended questions, including multiple-choice questions (MCQs), were included to collect specific data on preferences, satisfaction levels, and demographic information.

|  |
| --- |
|  |
| Question | **Type** | **Options/Scale** |
| 1. Are you a patient or a healthcare provider? | Multiple Choice | Patient / Healthcare Provider |
| 2. How frequently do you visit healthcare facilities for medical consultations? | Multiple Choice | Once a month or less / 2-3 times a month / Once a week or more |
| 3. Have you ever used a telemedicine application for medical consultations? | Multiple Choice | Yes / No |
| 4. If yes, which telemedicine application(s) have you used? | Multiple Choice | [List of telemedicine applications] |
| 5. What factors do you consider when choosing a healthcare provider for medical consultations? | Open-ended |  |
| 6. How comfortable are you with booking appointments online? | Scale | Not at all comfortable / Somewhat comfortable / Very comfortable |
| 7. Have you encountered any difficulties or challenges while using telemedicine applications? If yes, please specify. | Open-ended |  |
| 8. On a scale of 1 to 5, how important is it for you to have access to a medical chatbot for assistance and guidance? | Scale | 1 (Not important) / 2 / 3 / 4 / 5 (Very important) |
| 9. How satisfied are you with the appointment booking process through telemedicine applications? | Scale | 1 (Very dissatisfied) / 2 / 3 / 4 / 5 (Very satisfied) |
| 10. In your opinion, what features or improvements would enhance the user experience of telemedicine applications? | Open-ended |  |

The questionnaire is designed to gather insights on users' experience with telemedicine applications. It includes multiple-choice questions to assess user demographics, usage patterns, and satisfaction levels, as well as open-ended questions to capture qualitative feedback and suggestions for improvement.

## User Requirements

The users of this application are **patients**, **doctors**, and **administrators**.

**Patients:**

* Sign up for an account: Patients should be able to create a new account by providing necessary information such as their name, contact details, and any other required information.
* Book appointments: Patients should have the ability to search for available doctors and health services and book appointments based on their preferences and availability.
* Interact with a medical chatbot: Patients should be able to communicate with a medical chatbot to get basic information or assistance regarding their health concerns.
* Receive notifications: Patients should receive notifications about their upcoming appointments, reminders, or any important updates related to their appointments.
* View appointments: Patients should have access to their past and upcoming appointments, including details such as appointment time, doctor information, and any notes or instructions related to the appointment.

**Doctors:**

* View appointments: Doctors should be able to view their scheduled appointments, including details such as patient information, appointment time, and any notes or instructions provided by the patient.
* Mark appointments as complete: Doctors should have the ability to mark appointments as complete once they have attended to the patient.
* Update appointment notes: Doctors should be able to add or update notes related to the appointments, such as diagnosis, treatment plans, or any follow-up instructions.
* Undo changes: Doctors should have the option to undo any changes made to appointment details or notes if required.
* Access analytics dashboard: Doctors should be able to access an analytics dashboard that provides insights and metrics related to their appointments, patient interactions, and other relevant data.

**Administrators:**

* Create doctor accounts: Administrators should have the authority to create new doctor accounts and provide them with the necessary credentials to access the system.
* Manage appointments: Administrators should be able to manage appointments, including adding, modifying, or canceling appointments as required.
* Access analytics dashboard: Administrators should have access to an analytics dashboard that provides insights and metrics related to appointments, patient-doctor interactions, and other relevant data.

**Access Requirements:**

To gain access to the platform, users typically require the following:

* A device with internet access: Users should have a computer, smartphone, or tablet with an internet connection to access the application.
* Modern web browser: Users should have a modern web browser that supports HTML5, CSS3, and JavaScript, such as Google Chrome, Mozilla Firefox, Microsoft Edge, or Apple Safari.
* Sufficient bandwidth: Users should have sufficient internet bandwidth to load the website and interact with its features smoothly.

## 

## Functional Requirements

The specialist booking website and telemedicine application should include the following functionality:

1. **Patient Functionality**:
   1. User Registration: Patients should be able to sign up for an account with their personal information and create login credentials.
   2. Appointment Booking: Patients should be able to book appointments for specific health services, selecting preferred dates and times.
   3. Medical Chatbot Interaction: Patients should be able to interact with a medical chatbot powered by OpenAI's GPT-3.5 to receive assistance and guidance related to their health concerns.
   4. Appointment Notifications: Patients should receive notifications about their appointments, including reminders and updates.
   5. Appointment History: Patients should be able to view their past and upcoming appointments, along with relevant details.
2. **Doctor Functionality**:
   1. Appointment Viewing: Doctors should have access to a dashboard where they can view their scheduled appointments, including patient information and appointment details.
   2. Appointment Completion: Doctors should be able to mark appointments as complete once the consultation is finished.
   3. Appointment Notes: Doctors should be able to update and maintain appointment notes, documenting relevant information and treatment details.
   4. Undo Changes: Doctors should have the ability to undo any accidental changes made to appointment records.
3. **Administrator Functionality**:
   1. Analytics Dashboard: Doctors and administrators should be able to access an analytics dashboard page that provides insights and metrics related to appointments, patient flow, and other relevant data.
   2. Doctor Account Creation: Administrators should have the capability to create and manage doctor accounts, including assigning appropriate permissions and access levels.
   3. Appointment Management: Administrators should be able to manage appointments, including rescheduling, canceling, and reassigning appointments as necessary.

## Non-Functional Requirements

The non-functional requirements for the application are as follows:

* Usability: The application should have a user-friendly interface and intuitive navigation to ensure ease of use for all users.
* Performance: The application should be highly responsive, with fast loading times and minimal latency, even when handling a high volume of users and appointments.
* Security: The application should have robust security measures in place to safeguard user data and prevent unauthorized access or breaches.
* Scalability: The application should be designed to handle increasing user and appointment loads without compromising performance or user experience, allowing for future growth and expansion.

## Project Schedule

This section outlines the key milestones and tasks involved in the project, including the implementation of core features such as the booking system and chatbot functionality, as well as any planned enhancements such as the integration of Google Maps API. The objective is to ensure that the project stays on track and is completed within the specified timeframe.

Images of the Gantt chart and Work Breakdown Structure are included along with a list of dependencies.

### Gantt Chart and Work Breakdown Structure

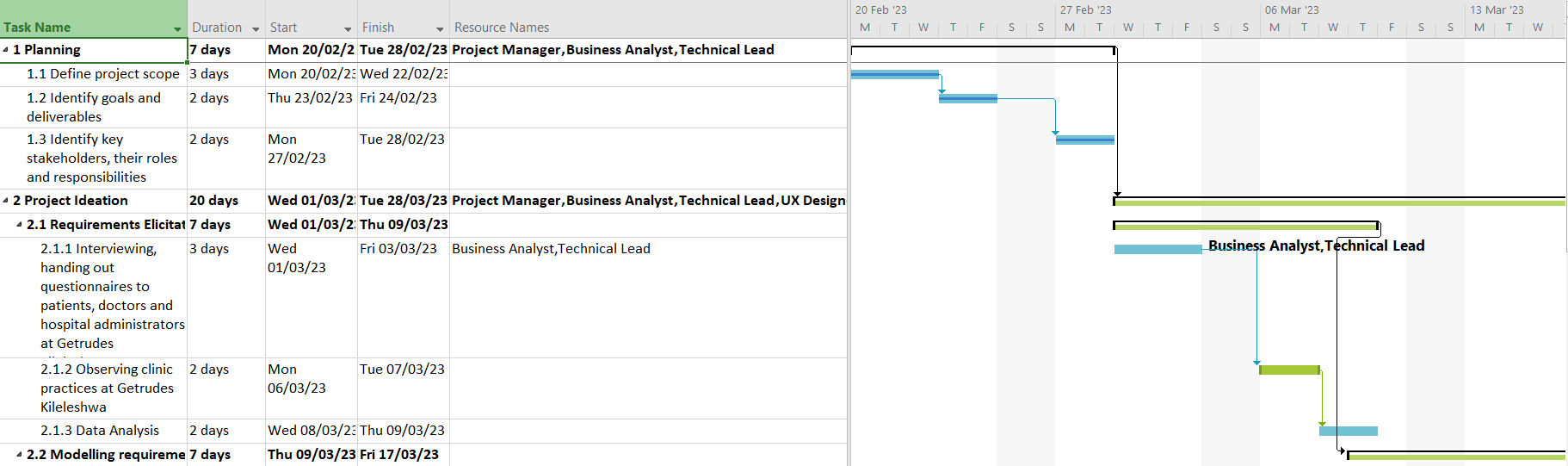


Figure 1:Gantt Chart Task 1 to 2.2

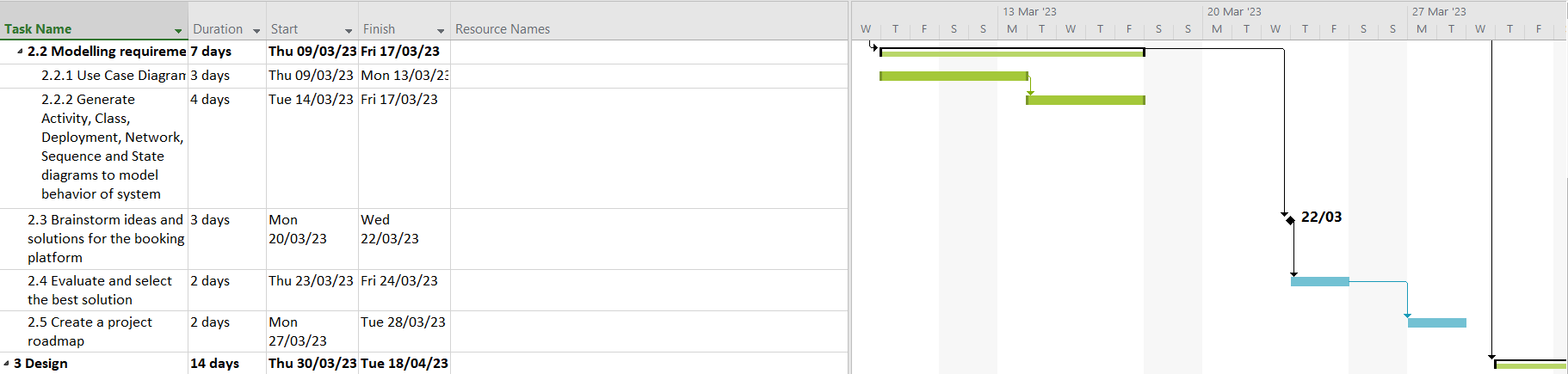
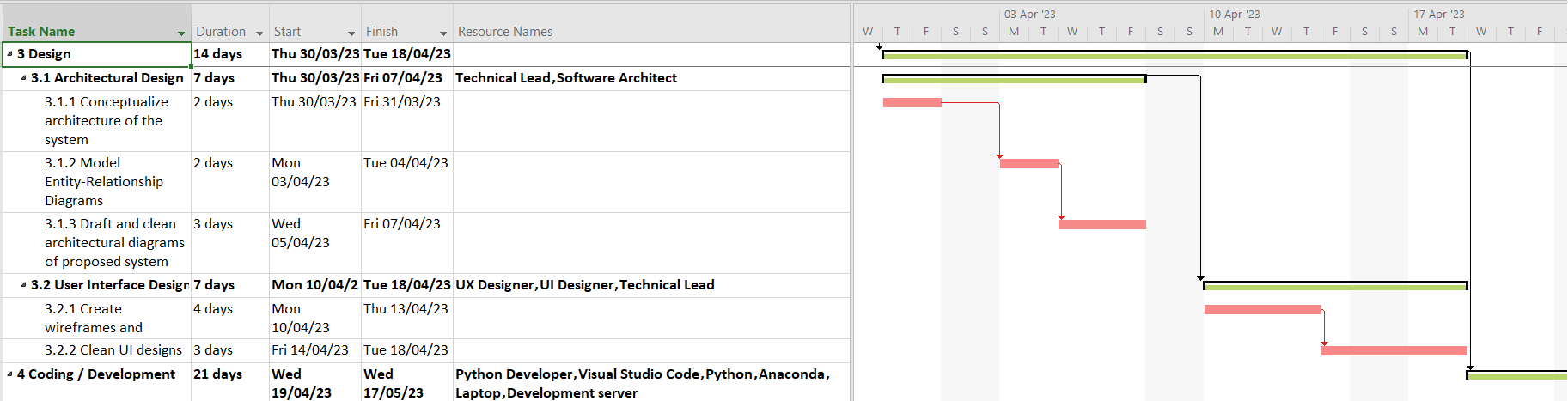
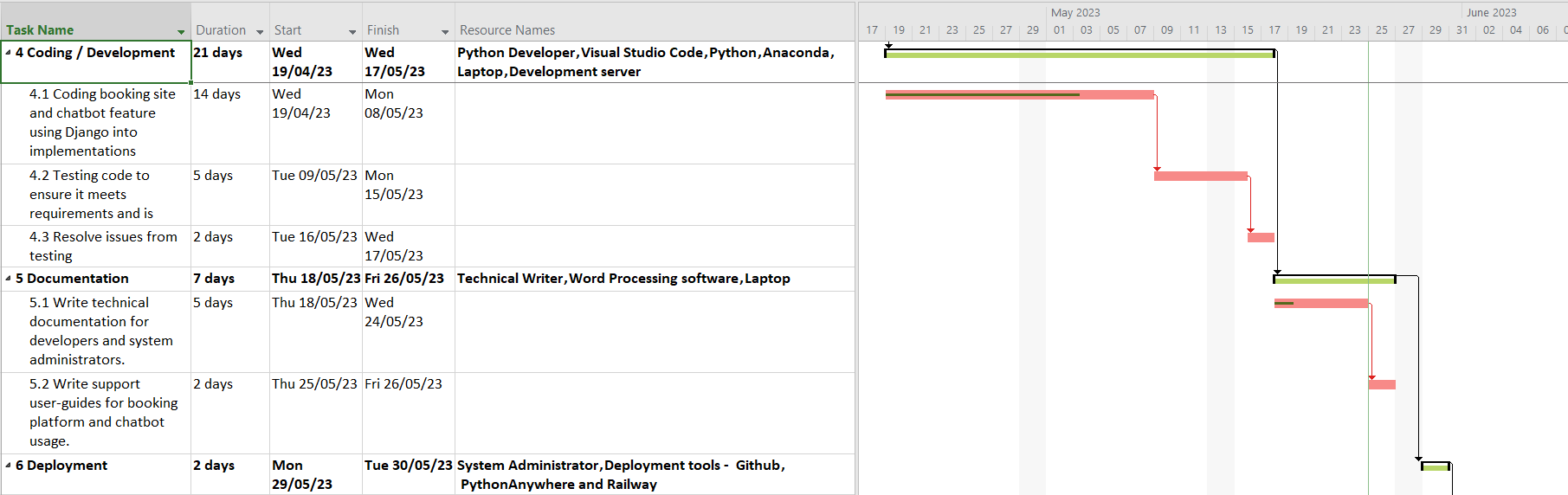


Figure 4: Task 4 to 6

Figure 3: Task 3 to 4

Figure 2:Task 2 to 3

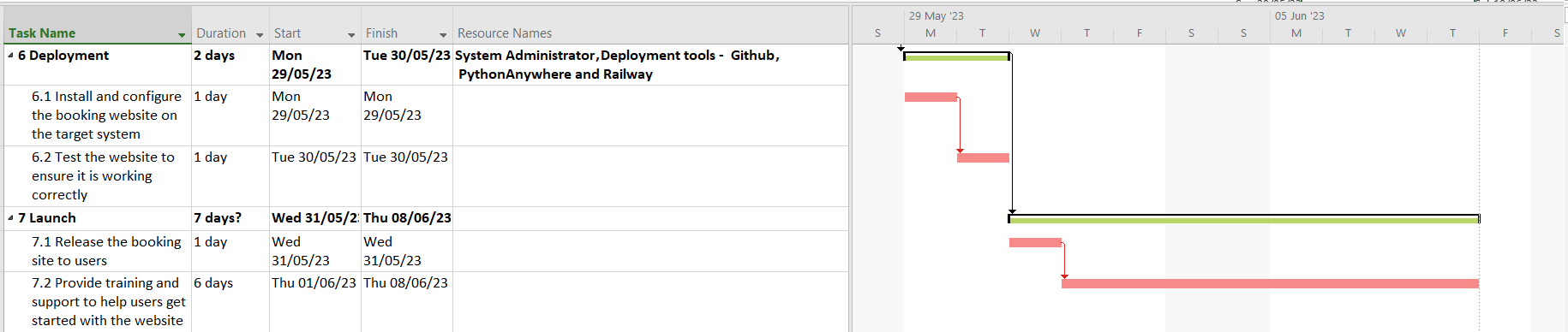


Figure 5: Task 6 to 7

**Planning**:

* Duration: 7 days (Mon 20/02/23 - Tue 28/02/23)
* Resources: Project Manager, Business Analyst, Technical Lead

**Project Ideation**:

* Duration: 20 days (Wed 01/03/23 - Tue 28/03/23)
* Resources: Project Manager, Business Analyst, Technical Lead, UX Designer

**Design**:

* Duration: 14 days (Thu 30/03/23 - Tue 18/04/23)
* Resources: Technical Lead, Software Architect, UX Designer, UI Designer

**Coding / Development**:

* Duration: 21 days (Wed 19/04/23 - Wed 17/05/23)
* Resources: Python Developer, Visual Studio Code, Python, Anaconda, Laptop, Development server

**Documentation**:

* Duration: 7 days (Thu 18/05/23 - Fri 26/05/23)
* Resources: Technical Writer, Word Processing software, Laptop

**Deployment**:

* Duration: 2 days (Mon 29/05/23 - Tue 30/05/23)
* Resources: System Administrator, Deployment tools - Github, PythonAnywhere, and Railway

**Launch**:

* Duration: 7 days? (Wed 31/05/23 - Thu 08/06/23)
* Release the booking site to users: 1 day (Wed 31/05/23)
* Provide training and support: 6 days (Thu 01/06/23 - Thu 08/06/23)

### Resources

* Project Manager
* Business Analyst
* Technical Lead
* UX Designer
* Software Architect
* UI Designer
* Python Developer
* Visual Studio Code
* Python
* Anaconda
* Laptop
* Development server
* Technical Writer
* Word Processing software
* System Administrator
* Deployment tools (Github, PythonAnywhere, Railway)

## Constraints

The development of the specialist health clinic booking site and medical chatbot is governed by the following constraints:

* + 1. **Time Constraint**: The project must be completed within a strict timeframe of three months. This constraint imposes a deadline for the development team to deliver a functional and tested application within the given time frame. It requires efficient project management, task prioritization, and effective utilization of resources to meet the project's timeline.
    2. **Budget Constraint**: The project operates under a limited budget for development costs. This constraint necessitates careful consideration of resource allocation and cost-effective strategies. Extra features, such as payment integration, will need to be either implied or implemented as demos due to budget limitations. The development team must prioritize essential functionalities and make efficient use of available resources to stay within the budget constraints.
    3. **Technology Constraint**: The application will be developed using the Django web framework. This constraint implies that the project will leverage the features and capabilities provided by Django to build the specialist health clinic booking site and medical chatbot. The development team should have expertise in Django and utilize its built-in functionalities, libraries, and best practices to ensure efficient development, maintainability, and scalability of the application.
    4. **Developing Intuitive User Interfaces:** Focusing on user experience (UX) design principles to ensure that the interface is easy to navigate, visually appealing, and provides a seamless user journey. It is essential to prioritize user feedback and conduct usability testing to refine and enhance the user interfaces.
    5. **Test-Driven Development**: Another constraint is to adopt a test-driven development (TDD) approach. This means writing tests before writing the actual code, ensuring that the code meets the specified requirements and passes the tests. TDD helps in identifying and fixing issues early in the development process, improving code quality, and providing confidence in the reliability of the application.
    6. **Building Own Payment Integration Platform**: The constraint of building your own payment integration platform implies that you will develop a custom solution for handling payments within the application, rather than relying on pre-built payment gateways or third-party solutions. This requires understanding the security and compliance requirements for handling financial transactions and implementing a robust and secure payment processing system within the application.

# System Analysis and Design

## Overview

The design phase of the project is a crucial step in developing the specialist health clinic booking site and medical chatbot. It involves creating a comprehensive plan for the system architecture, user interface design, data flow, and deployment infrastructure. The goal is to translate the project requirements into tangible design artifacts that serve as a blueprint for development.

During this phase, we will adopt a user-driven development approach, focusing on delivering a user-friendly and intuitive application. Rapid application development techniques will be employed to ensure efficient and iterative design iterations, allowing for quick feedback and adjustments.

The system architecture will follow the Model-View-Controller (MVC) pattern, dividing the application into distinct components for efficient development and maintenance. Entity relationship diagrams will be used to model the database schema, while data flow diagrams and class diagrams will illustrate the flow of information and system structure.

The design phase will also address constraints such as time and budget limitations. Extra features, such as payment integration, will be implied or implemented as demos to align with the budget constraints. Test-driven development principles will be embraced to ensure the quality and reliability of the system.

Overall, the design phase aims to provide a clear direction for the development team, establish a solid foundation for implementation, and ensure that the final product meets the specified requirements and objectives of the project.

## Design Approach

To ensure a systematic and effective design process, the following approach will be adopted:

1. **Data Flow Design (DFD)**:
   * Create Data Flow Diagrams (DFDs) to represent the flow of data within the system.
   * Identify the inputs, outputs, processes, and data stores.
   * Analyze the data flow and information exchange between components.
2. **System Architecture Design**:
   * Follow the Model-View-Controller (MVC) pattern.
   * Identify the core components and their interactions.
   * Create architectural diagrams to visualize the system structure.
3. **Database Design**:
   * Model the database schema using entity relationship diagrams.
   * Define tables, relationships, and attributes.
   * Ensure data integrity and efficient data retrieval.
4. **Class Design**:
   * Identify the main classes and their relationships.
   * Create class diagrams to represent the system's structure.
   * Define class attributes and methods.
5. **Activity and Network Design**:
   * Create activity diagrams to illustrate system processes and workflows.
   * Develop network diagrams to depict system interactions and dependencies.
6. **User Interface Design**:
   * Conduct user-centered design activities.
   * Create wireframes and mockups for UI elements.
   * Focus on intuitive and visually appealing interfaces.
7. **Deployment Design**:
   * Create a deployment diagram to outline the infrastructure setup.
     1. Identify servers, databases, and external services.
     2. Ensure scalability, reliability, and security.

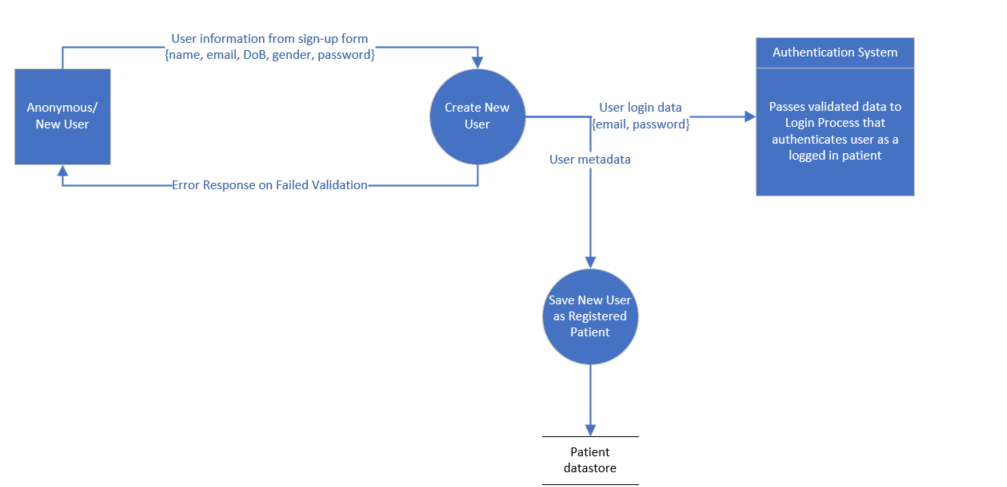
By following this design approach, I aim to:

* Clearly understand the flow of data within the system through DFDs.
* Establish a well-defined system architecture based on the MVC (Model-View-Controller) pattern.
* Design an efficient and scalable database schema.
* Define the classes and their relationships to ensure proper implementation.
* Visualize the system's activities and network interactions.
* Plan the deployment infrastructure to ensure smooth operation and security.

## Data Flow Diagrams

### Sign-Up Process

#### 0-level Data Flow Diagram



In this DFD, the focus is on the sign-up process for new users. An Anonymous/New User entity submits a filled sign-up form, which is then passed to the **Register process**.

The Register process validates the submitted data and either returns an error in case of validation issues or automatically creates the user account and passes the user login to the **Login process** that authenticates the user’s credentials, and logs in the new user.

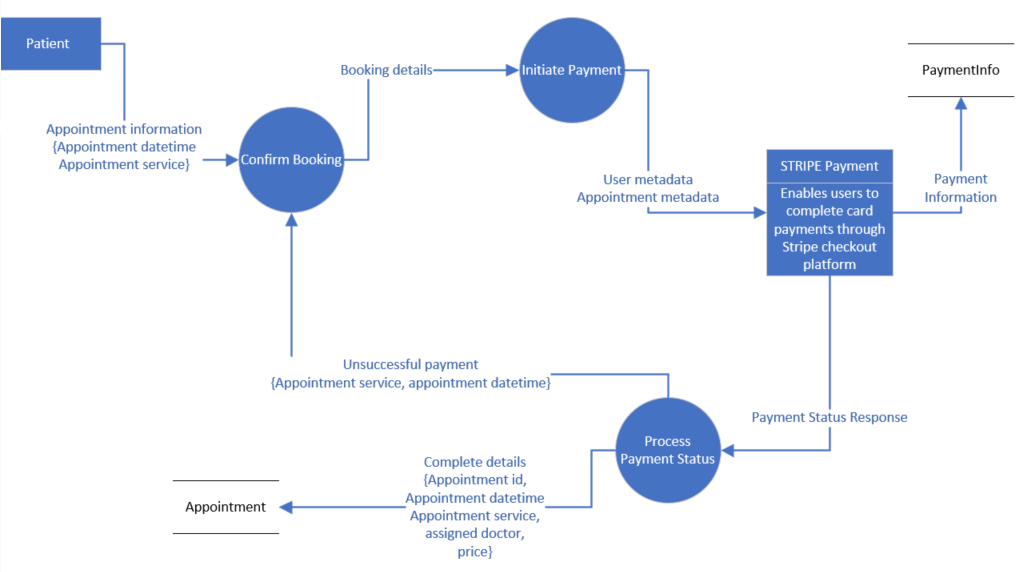
The overall objective is to ensure a seamless and secure sign-up process for users, where their information is validated and stored correctly for future authentication and account management.

### Booking Process

#### 0-level Physical DFD

In the physical DFD for the appointment booking process, the Patient visits the clinic's reception desk to schedule an appointment. The patient interacts with the receptionist, providing details such as the desired doctor, preferred date and time, and the specific health service required. The receptionist manually inputs this information into the system, which initiates the booking process. The system validates the availability of the doctor and generates an appointment confirmation, which is then provided to the patient. If applicable, the receptionist may also collect payment for the appointment. This physical interaction ensures a seamless and efficient booking experience for the patient at the clinic's front desk.

#### 0-level Logical DFD

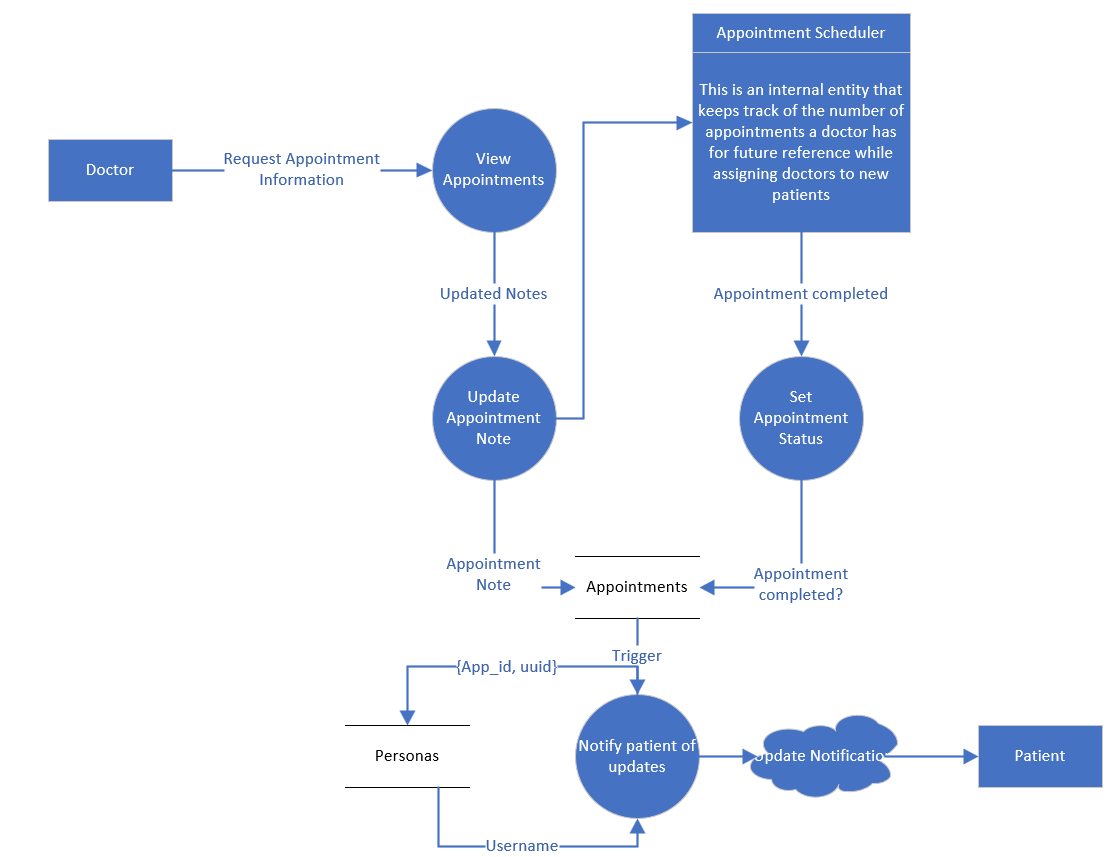


This DFD represents the booking process in the system. It captures the interaction between a patient and the clinic when requesting an appointment with a doctor. Two possible methods of booking are depicted: the patient physically visiting the clinic and requesting an appointment at the reception desk or the patient using a device to communicate with the clinic remotely.

In this process, the Patient entity initiates the booking by providing the necessary details such as the desired datetime and the specific service required. The Confirm Booking process receives the appointment request and passes it to the Payment process.

The Payment process initiates a payment gateway whose result determines the next operation. A successful payment confirms the appointment while any cancelled payment redirects to the Confirm Booking iteration. Each process involved in the booking process interacts with its appropriate data store to store or retrieve relevant information. The objective of this DFD is to ensure a smooth and efficient booking process for patients, whether they are present at the clinic or making remote bookings.

#### Level-1 Logical DFD: Appointment Management

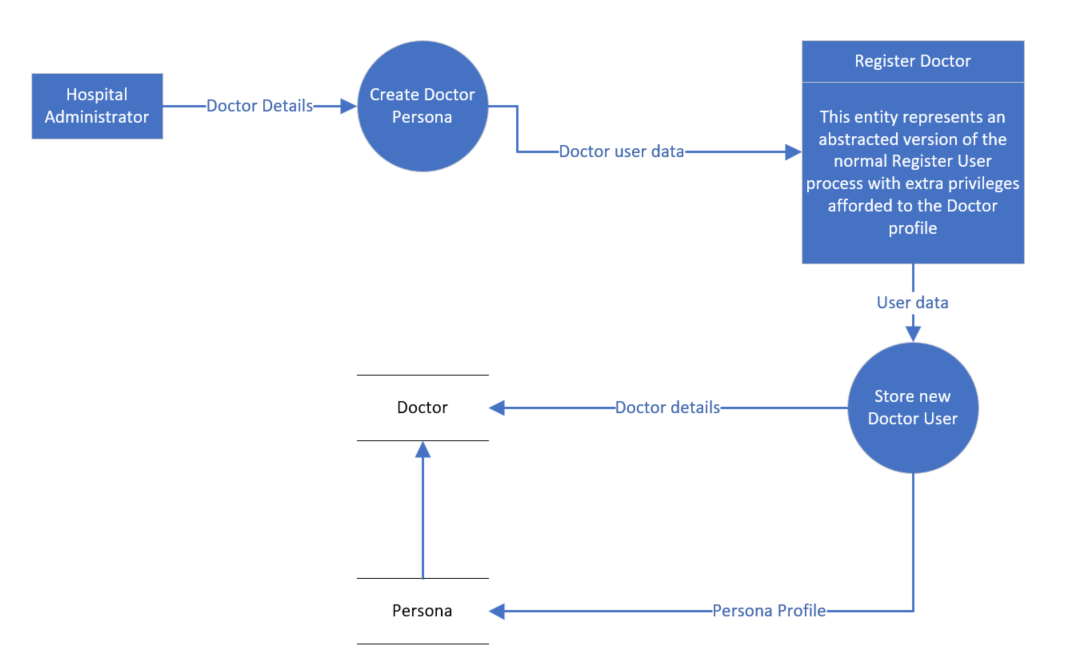


In the logical DFD for appointment management, the **key entity is the Doctor** who is **responsible for managing appointments** within the system. The Doctor interacts with the system through a user interface that provides the necessary functionalities for appointment management. The Doctor performs various actions, **such as viewing upcoming appointments, marking appointments as complete, and updating appointment notes**.

There is also a **Notification process** that plays a crucial role. The Notification process is responsible for sending notifications to patients regarding their appointments. When an appointment is scheduled or updated by the Doctor, the system triggers the Notification process to send a notification to the respective patient. This ensures that patients are promptly informed about any changes or updates to their appointments. The Notification process retrieves the necessary information from the appointment records and uses a communication channel, such as email or SMS, to send the notification to the patient. This ensures effective communication and helps in keeping patients informed and updated about their appointments

### Doctor Registration

#### 0-level DFD



The main entities involved are the **Hospital Administrator and the Registration module**. The Administrator initiates the registration process by providing inputting staff’s required identification details and professional details, such as name, contact information, specialty, and credentials.

The **Registration System** receives the provided information and **validates it for completeness and correctness**. Once the validation is successful, the Registration System creates a new Doctor account and assigns a unique identifier. The Doctor's information is stored in the Doctor Database and a Persona profile to enable Doctors receive same benefits as Patients.

### Medical Chatbot Interaction

#### 0-level Logical DFD

The main entities involved are the **User/Patient and the Chatbot System**.

The User initiates the interaction by sending a message or query to the Chatbot System.

The Chatbot System **receives the user's message and processes it using natural language understanding techniques**. It analyzes the user's intent and context to generate an appropriate response.

The Chatbot System **accesses the medical knowledge base and algorithms to provide relevant and accurate information or recommendations** to the User. The generated response is then sent back to the User, completing the interaction loop.

### Analytics Dashboard

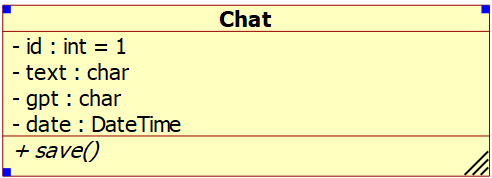
The key elements involved are the Administrator entity and the processes related to preparing and presenting the dashboard. The Administrator entity interacts with the system to access the Analytics Dashboard. The "Prepare Analytics Dashboard" process pulls data queries from the datastores, specifically the Persona Datastore and the Appointments Datastore. These queries are transformed into graphs or charts by the "Transform queries into graphs" process. The final step is the "Present charts in Analytics Dashboard" process, which displays the generated charts in the Analytics Dashboard for the Administrator to view and analyse. The data flows include the data querysets from the datastores, the trigger request, the queries themselves, the access to the Analytics data, and the data displayed in the Dashboard.

The Analytics Dashboard logical DFD captures the flow of data and processes involved in generating and presenting the analytics charts in the dashboard. The Administrator can access the dashboard, triggering the process of pulling data queries from the datastores. These queries are transformed into visual representations, such as graphs or charts, which are then presented in the Analytics Dashboard. This allows the Administrator to analyse and gain insights from the data stored in the Persona Datastore and the Appointments Datastore, facilitating data-driven decision-making.

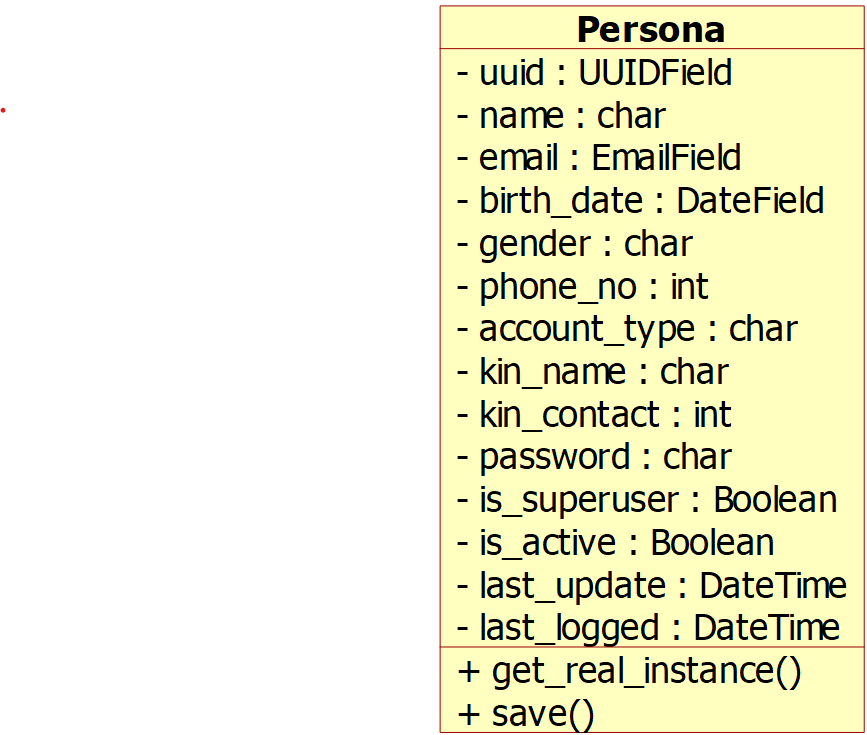
## Class Diagrams

In the class diagrams section, attributes, methods, and relationships play a crucial role in representing the structure and behavior of classes in the project. Attributes define the data associated with each class, methods represent the actions or behaviors that can be performed, and relationships establish connections and dependencies between classes, facilitating data flow and interactions within the system. Understanding these elements is essential for comprehending the design and functionality of the classes in developing the clinic application.

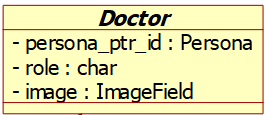
### The Chat Class

The Chat class represents user-input prompts and GPT-answers stored in the application. It has fields to store the text, GPT response, and date of the chat.

### The Persona Class

 The Persona class is a custom user model that extends Django's AbstractUser and PermissionsMixin classes. It represents users of the application and includes fields for personal information such as name, email, birth date, gender, phone number, account type, and more.

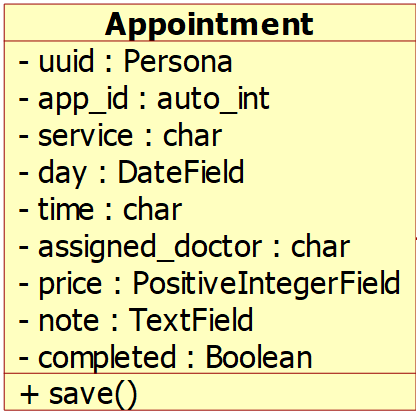
### The Doctor Class



**Inheritance:** Doctor inherits from Persona class.

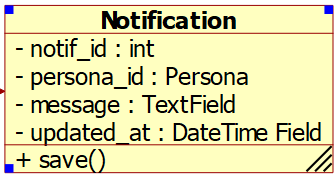
The Doctor class represents a specialized type of user, specifically doctors. It includes additional fields such as role and image for doctor-specific information.

### Appointment Class



The Appointment class represents appointments made by users. It includes fields for the user UUID, appointment ID, service type, appointment date and time, assigned doctor, price, notes, and completion status.

### Notification Class



The Notification class represents notifications sent to users. It includes fields for notification ID, user ID, message content, and the timestamp of when the notification was last updated.

## Overview Class Diagram

Here is a summary of the relationships between my abstracted classes:

1. Chat:
   * No explicit relationships with other classes.
2. Persona:
   * Doctor: Inherits from Persona model.
   * Appointment: ForeignKey relationship with Persona class (uuid field).
   * Notification: ForeignKey relationship with Persona class (persona\_id field).
3. Doctor:
   * Inherits from Persona.
4. Appointment model:
   * Persona: ForeignKey relationship with Persona (uuid field).
5. Notification model:
   * Persona: ForeignKey relationship with Persona (persona\_id field).

To summarize:

* The Persona model is the base model that represents users of the application.
* The Doctor model inherits from the Persona model, representing specialized users who are doctors.
* The Appointment model has a ForeignKey relationship with the Persona model, linking appointments to specific users.
* The Notification model also has a ForeignKey relationship with the Persona model, associating notifications with individual users.

## Database Schema and Entity-Relationship Diagrams

This phase of the project entails designing the underlying structure of the application's database and visualizing the relationships between different entities. The database schema defines the tables, fields, and relationships required to store and organize data effectively. It provides a blueprint for data storage and retrieval.

Additionally, the Entity-Relationship Diagram (ERD) visually represents the entities, attributes, and relationships within the database schema. It helps to identify the entities and their relationships, including one-to-one, one-to-many, or many-to-many relationships. The ERD serves as a powerful tool for understanding the data model and ensuring data integrity and consistency.

In this section, I define the database schema for each model, outlining the field names, field types, properties, and validation checks.

#### Persona table

|  |  |  |  |
| --- | --- | --- | --- |
| Field Name | Field Type | Properties | Validation Check |
| uuid | UUIDField (PK) | default=uuid.uuid4 |  |
| name | CharField | max\_length=50 |  |
| email | EmailField | max\_length=255, unique=True | Valid email address |
| birth\_date | DateField |  |  |
| gender | CharField | choices=GENDER\_CHOICES, max\_length=20 |  |
| phone\_no | IntegerField |  |  |
| account\_type | CharField | choices=ACCOUNT\_CHOICES, max\_length=10 |  |
| kin\_name | CharField (Nullable) | max\_length=50 |  |
| kin\_contact | IntegerField (Nullable) |  |  |
| password | CharField | max\_length=30 | Min length: 8, Cannot be "password" |
| is\_superuser | BooleanField | default=False |  |
| is\_active | BooleanField | default=True |  |
| last\_update | DateTimeField | auto\_now=True |  |
| last\_logged | DateTimeField | default=datetime.now |  |

The "Persona" model extends Django's AbstractUser and PermissionsMixin classes, providing a custom user model with additional fields tailored to the platform's requirements. It is converted to a SQL table that includes attributes such as name, email, birth date, gender, and phone number, along with validation checks to ensure data integrity. The table also incorporates fields for account type, password, and last update information. The Persona table plays a vital role in user management and authentication within the platform.

#### The Doctor table (Inherits from Persona)

|  |
| --- |
|  |
| Field Name | **Field Type** | **Properties** | **Validation Check** |
| role | CharField | max\_length=50 |  |
| image | ImageField | upload\_to='profiles', verbose\_name='Images' |  |

#### Appointment table

|  |  |  |  |
| --- | --- | --- | --- |
| Field Name | Field Type | Properties | Validation Check |
| uuid | ForeignKey (Persona) | on\_delete=models.CASCADE |  |
| app\_id | BigAutoField (PK) |  |  |
| service | CharField | max\_length=50 |  |
| day | DateField | default=datetime.now |  |
| time | CharField | choices=TIME\_CHOICES, max\_length=10 |  |
| time\_ordered | DateTimeField | default=datetime.now, blank=True |  |
| assigned\_doctor | CharField | max\_length=30, editable=False |  |
| Price | PositiveIntegerField |  |  |
| Note | TextField | default="" |  |
| Completed | BooleanField | default=False |  |

The Appointment table in the application plays a crucial role in managing and tracking appointments between users and doctors. It stores information such as the appointment ID, associated user (via a foreign key relationship), selected service, appointment date, time, and other details like the assigned doctor, price, and completion status. This model allows for efficient appointment scheduling and tracking within the platform.

The Appointment table’s attributes provide essential data points for managing appointments, ensuring accurate scheduling and tracking of medical consultations. The foreign key relationship with the Persona model establishes the connection between users and their respective appointments. Additionally, the completion status attribute allows for monitoring the progress of appointments, while the price attribute enables financial management related to appointment services.

#### Notification table

|  |  |  |  |
| --- | --- | --- | --- |
| Field Name | Field Type | Properties | Validation Check |
| id | UUIDField (PK) | default=uuid.uuid1 |  |
| notif\_id | IntegerField |  |  |
| persona\_id | ForeignKey (Persona) | on\_delete=models.CASCADE |  |
| message | TextField |  |  |
| updated\_at | DateTimeField | default=datetime.now |  |

The Notification table in the application serves as a vital component for delivering important messages and updates to users. It stores information such as the notification ID, associated user (via a foreign key relationship), message content, and the timestamp of when the notification was last updated. This model efficiently supports the notification functionality within the platform.

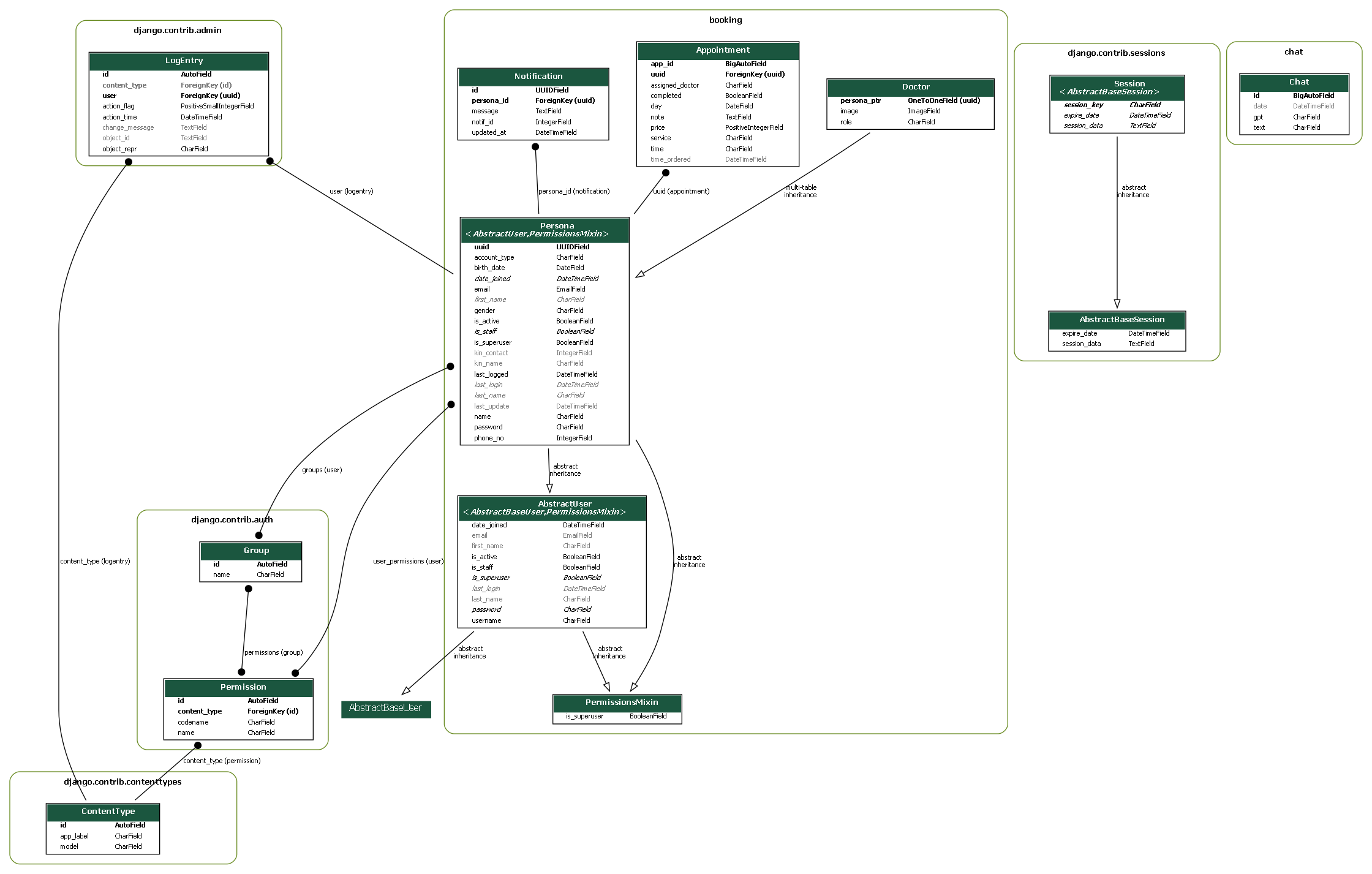
The platform can effectively communicate to users by sending timely notifications and alerts. The foreign key relationship with the Persona model allows for linking notifications to specific users, ensuring targeted message delivery. The message attribute holds the content of the notification, providing relevant information or updates to users. The timestamp attribute enables tracking and displaying the last updated time of the notification, allowing users to stay informed about the most recent developments.

#### Chat table

|  |  |  |  |
| --- | --- | --- | --- |
| Field Name | Field Type | Properties | Validation Check |
| id | Integer (PK) |  |  |
| text | CharField | max\_length=500 |  |
| gpt | CharField | max\_length=17000 |  |
| date | DateTimeField | auto\_now\_add=True, blank=True, null=True |  |

The Chat table is a fundamental component of the application, responsible for storing user interactions and GPT-generated answers. It consists of fields such as "text" to store the user input, "gpt" to store the corresponding GPT-generated response, and "date" to capture the timestamp of the interaction. This model efficiently supports the chat functionality within the platform.

The application can record and analyse user interactions with the chatbot. The "text" field stores the user's input, allowing for further analysis and understanding of user queries or requests. The "gpt" field captures the corresponding GPT-generated response, providing a record of the chatbot's output. The "date" field records the timestamp, enabling the platform to analyse the timing of user interactions.



### Entity-Relationship Diagram

## System Architecture

### Overview

This section highlights the various components of the application, how they are organized and interface with each other. The structure, behavior and deployment of the system are analysed, designed and optimized from a set of design options.

The following key aspects will be adopted and explored as the project’s development guide:

1. **Client-Server Architecture**: The application follows a client-server architecture, where the client (web browser) communicates with the server (Django application) to request and receive data.
2. **Django Framework**: The Django web framework provides a robust and scalable foundation for developing web applications. It follows the MTV (Model-Template-View) architecture, which is a variation of the MVC pattern specific to Django. The **Model** is the data manipulation logic of the application, the **View** is the presentation logic while the **Template** provides a semblance of user-contextualized contexts there making the illusion of personal sessions and interactivity veritable.

**This design pattern promotes code reusability especially in Rapid Application Development situations, modularity and maintainability.**

1. **Database Management System (DBMS)**: The application utilizes a DBMS, MySQL, to store and manage persistent data. The DBMS provides a reliable and efficient storage solution for the application's models and data.
2. **Third-Party Integrations**: The architecture may involve integrating third-party services or APIs for additional functionalities, such as chat completion with OpenAI and, payment processing with Stripe. While the limited budget may imply the use of demo implementations for certain features, the architecture provides flexibility for future integration and expansion.

### Architectural Diagram

#### High-Level Diagram

The architecture of the system follows the Model-View-Template (MVT) pattern, which provides a clear separation of concerns and promotes maintainability and scalability. At the core of the architecture is the **Model**, which represents the data entities and business logic of the application. The models define the structure of the database tables and handle interactions with the database through **data access operations**. The **View** component is responsible for handling user requests and rendering appropriate responses. It interacts with the models to fetch and manipulate data as needed. The **Template** component defines the presentation layer and provides the structure and layout for rendering the user interface.

One of the key components in the architecture is the **Custom Authentication Backend**, which enables the application to implement a tailored authentication mechanism. This component handles user authentication requests and performs necessary checks and validations using custom logic. It interfaces with the **View** component to receive authentication requests and provides authentication responses back to the view.

External APIs, such as **Stripe for payment processing** and **GPT 3.5 for chat completion**, play an important role in the system. These APIs are integrated into the architecture to extend the application's capabilities. For example, the Stripe API allows seamless payment processing by handling payment requests and responses. The GPT 3.5 API enables the chatbot functionality by processing user queries and generating appropriate responses.

By adopting the MVT architecture and incorporating components like the Custom Authentication Backend and External APIs, the system achieves a robust and scalable design. The clear separation of concerns enables efficient development and maintenance, while the integration of external services enhances the functionality and user experience. The architecture fosters extensibility, allowing for future enhancements and integration of additional components as needed.

The architecture of the platform can also be described as three separate microservices, each with high cohesion in terms of their specific business logic implementation and little coupling or dependency in terms of their own functioning.

### Booking App

The booking app handles most of the logic related to appointment booking and scheduling. It includes features such as:

* Allowing new users to sign up as patients
* Allowing patients to book appointments for specific health services
* Generating available time slots using an efficient scheduling algorithm that assigns the least-busy doctor
* Allowing patients to choose a time slot from the generated options
* Preventing users from editing appointments due within 24 hours
* Allowing users to receive notifications if their appointment note has been updated by a doctor
* Allowing users to view past or upcoming appointments
* Allowing doctors to access the staff panel and view their upcoming or past appointments
* Allowing doctors to mark an appointment as “complete” which sends a notification to the patient user and updates the appointment note
* Allowing doctors to undo marking an appointment as “complete”

This app has high cohesion, as all of its functionality is related to appointment booking and scheduling. It also has low coupling, as it does not depend heavily on other components of the project.

### Chat App

The chat app handles access to the medical chatbot and its integration with the OpenAI API. It includes features such as:

* Allowing users who have booked an appointment to access MediBot for a maximum of 24 hours from the time they submitted an appointment
* Providing a textbox-input webpage that runs OpenAI GPT-3.5 in the background
* Acting as a medical chatbot that only responds to medical-related questions

This app also has high cohesion, as all of its functionality is related to the medical chatbot. It has low coupling with other components of the project, but it does depend on external APIs such as the OpenAI API.

### Members App

The members app handles user authentication and account management. It includes features such as:

* Allowing only admin/superusers to create doctor accounts

This app has high cohesion, as all of its functionality is related to user authentication and account management. It also has low coupling with other components of the project.

## User Interface Design

### Design Principles and Consideration

When designing the wireframes for the platform, of note was the need to be consistent in adhering to certain design principles and considerations to ensure a user-friendly and visually appealing interface.

These are some key design tenets I intended to follow:

* Consistency: Maintaining a consistent design language throughout the wireframes, ensuring elements such as typography, colors, and layout are cohesive across all screens.
* Simplicity: Strive for a clean and minimalist design, focusing on essential elements and avoiding clutter. This will enhance usability and make the interface more intuitive for users.
* Visibility: Ensure that important elements and actions are prominently displayed, making them easily visible and accessible to users. This improves user engagement and reduces friction in completing tasks.
* Responsiveness: Design wireframes with a responsive layout, adapting to different screen sizes and devices. This guarantees a consistent and optimized user experience across desktop, tablet, and mobile devices.
* Accessibility: Consider accessibility guidelines, such as providing alternative text for images, using clear and legible fonts, and accommodating users with color blindness or other visual impairments.

### Wireframes and Mockups

From an early stage of the design process, I adopted use of low fidelity wireframes to prototype the ‘look and feel’ of key areas of the platform. The intention is to initially focus on users already within the platform as we reverse engineer towards the anonymous user’s perspective.

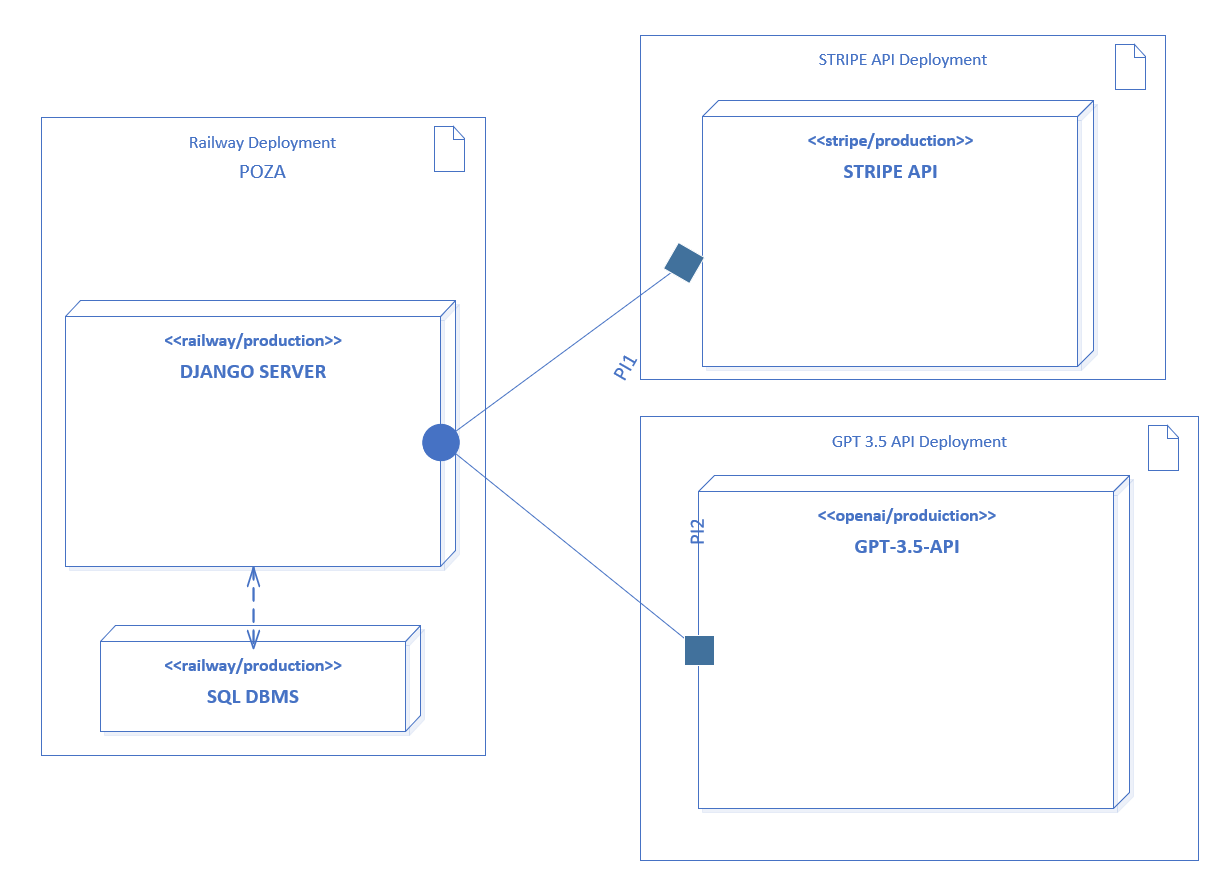
Below are the guidelines I will be sticking to in my usability design for the website:

* For the Booking Userflow Wireframe, prioritize a seamless and efficient booking experience, with clear form fields, error handling, and intuitive steps.
* For the Staff Panel Userflow Wireframe, focus on providing staff members with an intuitive and efficient interface to manage appointments, access patient information, and perform administrative tasks.
* For the User Panel Userflow Wireframe, emphasize user account management, appointment viewing, and a personalized experience to enhance user satisfaction and engagement.
* For the Analytics Dashboard Userflow Wireframe, concentrate on displaying data analysis and visualizations in a visually appealing and informative manner, allowing users to make data-driven decisions.

## Deployment and Network Infrastructure

One of the goals of this project is to develop a robust and scalable system architecture capable of efficiently managing high user traffic, ensuring reliable performance and uninterrupted availability, even during peak usage periods. To align with this goal, measures will be taken to prepare for a migration of the product from a local environment to a live production server where metrics of compute and memory usage can be better monitored for a diverse range of users.

First, the codebase of the Django project will be uploaded to a GitHub repository, which will serve as the central version control system for my project. This will allow for collaboration, easy code management and seamless deployment.

Railway plugins integrate with Github to enable automatic deployments and customizable features for deployments based on manual triggers.

The deployment architecture of project encompasses the arrangement and configuration of key components required for the successful deployment and operation of my application. At the core of this architecture is my Django application, which serves as the foundation for my web platform. It is deployed on a web server, acting as the entry point for users accessing my application over the internet. The web server handles incoming requests, processes them, and delivers the appropriate responses to users.

In conjunction with the web server, I utilize a robust database server to ensure efficient data storage and retrieval. The database server stores my application's data, enabling me to maintain data integrity and persistence. It plays a crucial role in managing user information, bookings, and other essential data that powers my platform.

To optimize the performance and scalability of my application, my deployment architecture also includes additional components. I employ static file servers to host and serve static files such as CSS, JavaScript, and images. This allows for faster loading times and a smoother user experience. Furthermore, I utilize load balancers to distribute incoming traffic across multiple servers, ensuring high availability and efficient resource utilization.

The deployment architecture diagram visually depicts the relationships and interactions among these components. It provides a clear overview of how my web server, database server, static file servers, and load balancers work together to deliver my application to users. This diagram serves as a valuable reference for system administrators, developers, and other stakeholders involved in the deployment and maintenance of my application.

By documenting my deployment architecture, I gain a comprehensive understanding of the configuration and dependencies of my deployment environment. This documentation facilitates troubleshooting, scalability planning, and system maintenance. It also serves as a vital resource for onboarding new team members and ensuring a consistent and reliable deployment process.

# Coding

# References

(Erin & Mugambi, 2022)