POZA SPECIALIST CLINIC APP

# Introduction

## Abstract

This project is a Django-based web application for a specialist clinic booking website. It combines traditional and emerging technologies to improve the medical sector in an effective and efficient manner. The application allows patients to book appointments for specific health services and interact with a medical chatbot powered by OpenAI’s GPT-3.5. Doctors and administrators can manage appointments and communicate with patients through the application. The site also features an analytics dashboard page that displays graphs over a dataframe containing all appointment data, including prices, services, and days.

## Background

The medical sector is constantly evolving, with new technologies and innovations being developed to improve patient care and outcomes. One such technology is telemedicine, which refers to the use of electronic information and telecommunication technology to deliver clinical care at a distance. Telemedicine has been shown to improve access to care, reduce wait times, and enhance communication between patients and healthcare providers

However, despite the potential benefits of telemedicine and other technologies, there are still many challenges that need to be addressed in the medical sector. These challenges include improving access to care, reducing wait times, and enhancing communication between patients and healthcare providers.

## Literature Review

Several studies have shown that technology can play a key role in addressing the challenges faced by health clinics. For example, online appointment booking systems have been found to improve access to care and reduce wait times. Medical chatbots powered by artificial intelligence have also been shown to enhance communication between patients and healthcare providers.

In addition to improving patient care, technology can also provide valuable insights for healthcare providers. For example, analytics dashboards can help doctors and administrators analyse appointment data to identify trends and make informed decisions about resource allocation.

## Problem Statement

Despite the potential benefits of technology in the medical sector, there is still a need for solutions that effectively combine traditional and emerging technologies to improve patient care. This project aims to address this need by developing a web application that allows patients to book appointments for specific health services using an efficient scheduling algorithm that auto-assigns the least-busy doctor. Patients can also interact with a medical chatbot powered by OpenAI’s GPT-3.5 to receive answers to medical-related questions.

A major functional requirement of the system is the ability to manage appointments and communicate with patients. Doctors and administrators can view and update appointment information, while patients can receive notifications about their appointments and view their past or upcoming appointments. The application also features an analytics dashboard page that displays graphs over a dataframe containing all appointment data.

The acceptance criteria for this project include:

* The application must allow patients to book appointments for specific health services.
* The application must provide an efficient scheduling algorithm that auto-assigns the least-busy doctor.
* The application must allow patients to interact with a medical chatbot powered by OpenAI’s GPT-3.5.
* The application must allow doctors and administrators to manage appointments and communicate with patients.
* The application must provide an analytics dashboard page that displays graphs over appointment data.

This project can be deployed in environments such as health clinics, hospitals, and other healthcare facilities.

## User Requirements

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

**Patients** require the ability to sign up for an account, book appointments for specific health services, interact with a medical chatbot, receive notifications about their appointments, and view their past or upcoming appointments.

**Doctors** require the ability to view their appointments, mark appointments as complete, update appointment notes, undo changes, and access the analytics dashboard page.

**Administrators** require the ability to create doctor accounts, manage appointments, and access the analytics dashboard page.

## Functional Requirements

The application must provide the following functionality:

* Allow patients to sign up for an account
* Allow patients to book appointments for specific health services
* Allow patients to interact with a medical chatbot powered by OpenAI’s GPT-3.5
* Send notifications to patients about their appointments
* Allow patients to view their past or upcoming appointments
* Allow doctors to view their appointments
* Allow doctors to mark appointments as complete
* Allow doctors to update appointment notes
* Allow doctors to undo changes
* Allow doctors and administrators to access the analytics dashboard page
* Allow administrators to create doctor accounts
* Allow administrators to manage appointments

## Non-Functional Requirements

The application must meet the following non-functional requirements:

* Usability: The application must be easy to use for all users.
* Performance: The application must provide fast response times and be able to handle a large number of users.
* Security: The application must protect user data and prevent unauthorized access.
* Scalability: The application must be able to scale to accommodate growth in the number of users and appointments.

## Constraints

The development of this application is subject to the following constraints:

* Time: The project must be completed within three months.
* Budget: The project has a limited budget for development costs. Therefore, extra features like payment integration will have to be implied or implemented as demos.
* Technology: The application will be developed using the Django web framework.

# Design

## Overview

This project was designed and developed using Rapid Application Development (RAD) principles. RAD is an agile software development methodology that emphasizes rapid prototyping and iterative delivery to quickly deliver functional software.

## Technologies

The project was developed using a combination of technologies, including Django, pandas, OpenAI API, and SQLite. Django is a high-level Python web framework that enables rapid development and clean, pragmatic design. Pandas is a powerful data analysis and manipulation library for Python. The OpenAI API provides access to advanced artificial intelligence models such as GPT-3.5. SQLite is a lightweight database engine that provides a simple and efficient way to store and retrieve data.

## Design Process

The design process for this project followed the key principles of RAD. The project began with a requirements-gathering phase, during which the functional and non-functional requirements of the system were identified. This was followed by a rapid prototyping phase, during which a working prototype of the system was developed using Django, pandas, OpenAI API, and SQLite.

The prototype was then iteratively refined through a series of user feedback and testing cycles. During each cycle, the prototype was evaluated by users and any necessary changes were made to improve its functionality and usability. This iterative process continued until the final version of the system was delivered.

## Architecture

The project follows a modular architecture with low coupling and high cohesion. This means that each component of the project is designed to be independent and self-contained, with minimal dependencies on other components. This allows for greater flexibility and maintainability in the development process.

The project has several key components including the booking, chat, and members apps. Each of these apps is responsible for a specific set of functionalities within the project.

## Database Schema and Entity-Relationship Diagrams

### 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.

# Coding

## User Profile Class Models and Interface Managers

This Persona extends Django’s AbstractUser and PermissionsMixin classes to provide custom fields and behavior for user profiles.

The Persona model defines several fields such as uuid, name, email, birth\_date, and gender to store information about each user. It also defines a custom manager called PersonaManager, which is the class defined in the first code snippet you provided.

The PersonaManager class provides custom methods for creating user profiles, including methods for creating regular users, doctor users, and superusers. These methods are used by the Django admin interface and other parts of the Django framework to create and manage user profiles.

The PersonaManager class extends Django’s BaseUserManager class and overrides several methods to provide custom behaviour for creating user profiles.

The \_create\_user method is a helper method that creates and saves a new user profile with the given email, password, and other fields. The create\_user method calls \_create\_user to create a new user profile with the is\_staff and is\_superuser fields set to False by default.

The create\_doctor method is similar to the create\_user method, but it sets the account\_type field to ACCOUNT\_CHOICES[1], which presumably represents a doctor account type.

The create\_superuser method calls \_create\_user to create a new user profile with the is\_staff and is\_superuser fields set to True. It also performs some validation to ensure that the is\_staff and is\_superuser fields are set to True.