# Project Report: **Hospital Management System**

## Author:


- **Name:** Shivansh Jhalani
- **Roll Number:** 24f3002824
- **Student Email:** 24f3002824@ds.study.iitm.ac.in
- **About Me:** I am a student learning Data Science from IIT Madras online BS degree programme. This project was a great way for me to practice building a complete web application using Python and Flask.


## AI/LLM Usage:

I used Google's Gemini (an AI/LLM) as a helper tool while working on this project.

My use of the AI was for tasks like:

- Helping to understand the project requirements and plan the structure.
- Answering my questions about technologies like Flask, Flask-WTF, and SQLite.
- Assisting with debugging code when I got stuck (for example, helping me solve a `CSRF token` error).
- Helping me organize my thoughts and write this project report.

The AI acted as a teaching assistant, but I wrote, tested, and finalized all the code for the project myself.

## Description:

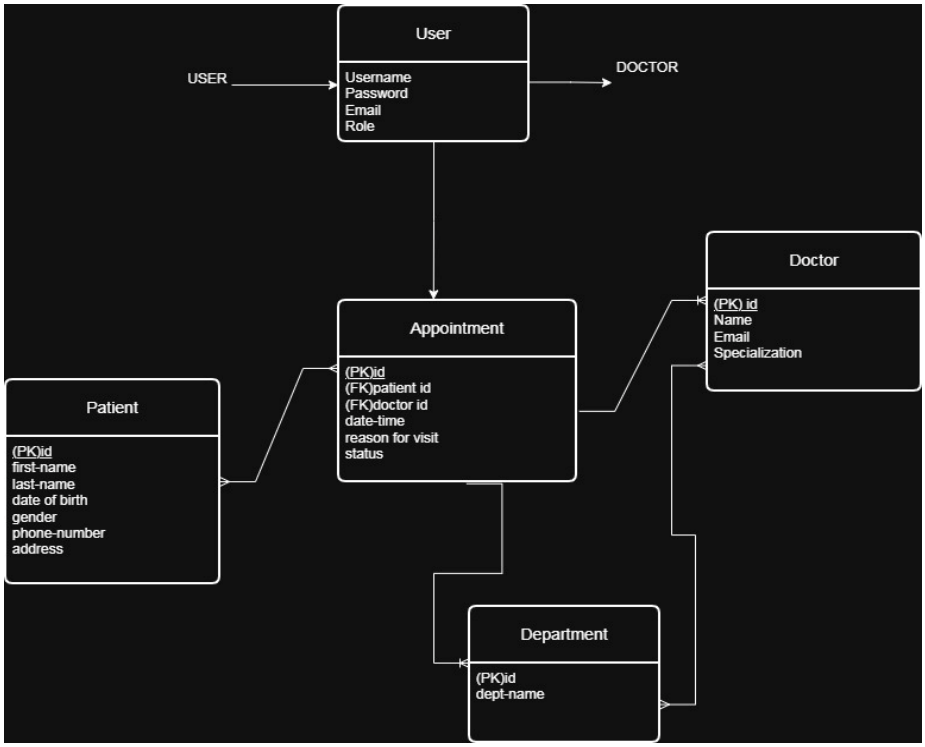
This project is a web application to help manage a hospital. It's designed for a user, like a receptionist, to easily handle information. The system allows the user to add, view, and manage details for doctors, patients, and their appointments.

- **AI/LLM Used:** I estimate I used the AI for about 15-20% of the support tasks.
- This help was mainly for getting ideas, solving problems, and documentation, not for writing the actual application code.

## Technologies Used:

- **Python:** The main programming language used to build the application's logic.
- **Flask:** A simple and flexible Python framework used to build the web server and handle web pages.
- **Jinja2:** A "templating engine" that works with Flask. It let me create HTML templates and easily insert Python data into them.
- **SQLite:** A lightweight, file-based database used to store all the project's data (like patient lists, doctor info, etc.).
- **Flask-SQLAlchemy:** A Flask extension that made it much easier to connect Flask to the SQLite database. It helped me create the database tables using Python code.
- **Flask-WTF:** A Flask extension used to create and protect web forms. This was important for security, like preventing CSRF attacks on my login and data entry forms.
- **HTML/CSS:** Used to structure and style the web pages to make them look good and easy to use.

## DB Schema Design:




**Reasons behind this design:** I designed it this way to be efficient and reduce repeated data. For example, a **Doctor now belongs to a `Department`**. Instead of typing the department name for every doctor, I just link them using the `department_id`. This is much cleaner.

Similarly, in the `Appointment` table, I only store the *IDs* of the patient and doctor. This "links" the tables together, saves space, and makes it easy to update a patient's or doctor's name in just one place.

## API Design:

In this project, the "APIs" are the routes that Flask uses to run the website. My Python code defines these routes, which handle requests from the user's browser.

- `/login` (GET and POST): Shows the login page (GET) and handles the form submission to check the username and password (POST).
- `/logout` (GET): Logs the user out.
- `/dashboard` (GET): The main page after login, which might show a summary of patients and appointments.
- `/patients` (GET): Shows a list of all patients.
- `/patient/add` (GET and POST): Shows the form to add a new patient (GET) and saves the new patient to the database (POST).
- `/doctors` (GET): Shows a list of all doctors.
- `/appointment/add` (GET and POST): Shows the form to book an appointment (GET) and saves the new appointment (POST).

These routes are the backbone of the application, connecting the HTML templates to the database logic.

## Architecture and Features:

**Architecture:** The project is organized in a way that is standard for Flask applications:

- `app.py`: This is the main file. It contains all the Flask routes (the "controllers"), sets up the database, and runs the application.
- `models.py`: (Or this code might be in `app.py`) This file defines the database tables (the "schema" like `User`, `Patient`, `Doctor`, `Department`, etc.) using Flask-SQLAlchemy.
- `forms.py`: This file defines all the web forms used to add or edit data, using Flask-WTF.
- `templates/` (Folder): This folder contains all the HTML files (the "views"). Jinja2 uses these files to build the pages the user sees.
- `static/` (Folder): This folder holds any static files, such as CSS files for styling.

**Features Implemented:**

- **Secure User Login:** (Based on the `User` table) A staff member must log in to use the system. Passwords are hashed for security.
- **Department Management:** (Based on the `Department` table) The user can add new hospital departments (like "Cardiology", "Orthopedics") and view a list of all existing departments.
- **Doctor Management:** (Based on the `Doctor` and `Department` tables) The user can add new doctors, edit their details, and **assign them to a specific department**.

- **Patient Management:** (Based on the `Patient` table) The user can add new patients, view a complete list of all patients.
- **Appointment Booking:** (Based on the `Appointment` table) The user can book a new appointment by **linking a `Patient` with a `Doctor`** and selecting a date and time.
- **Dashboard:** A central home page that shows key information at a glance.