ANUDIP FOUNDATION

**ITPR Project**

**Project Title**

**“Hospital Management System in Python Using Django”**

**By**

|  |  |
| --- | --- |
| **Name** | **Student id** |
| Deepesh Sarsar | AF0481788 |
| Omkar Maral | AF0481954 |

Under Guidance

Of

**Rajshri Thete**

**ABSTRACT**

The purpose of the project entitled as “HOSPITAL MANAGEMENT SYSTEM” is to computerize the Front Office Management of Hospital to develop software which is user friendly, simple, fast, and cost – effective. It deals with the collection of patient’s information, diagnosis details, etc. Traditionally, it was done manually. The main function of the system is register and store patient details and doctor details and retrieves these details as and when required, and also to manipulate these details meaningfully system input contains patient details, diagnosis details, while system output is to get these details on to the screen. The Hospital Management System can be entered using a username and password. It is accessible either by an administrator or receptionist. Only they can add data into the database. The data can be retrieved easily. The data are well protected for personal use and makes the data processing very fast.

**ACKNOWLEDGEMENT**

The project **“Hospital Management System in Python Using Django”** is the Project work carried out by

|  |  |
| --- | --- |
| **Name** | **Enrollment No** |
| **Rajshri Thete** |  |

Under the Guidance.

We are thankful to my project guide for guiding me to complete the Project.

His suggestions and valuable information regarding the formation of the Project Report have provided me a lot of help in completing the Project and its related topics.

We are also thankful to my family member and friends who were always there to provide support and moral boost up.

**CONTENT**

**OF T**

**HE PROJECT**

|  |  |
| --- | --- |
| **TOPIC OF THE PROJECT** | ***Page No*** |
| 1. ***Title of the Project*** 2. ***Introduction/Objective***     1. ***Introduction***    2. ***Objective*** 3. ***System Analysis***     1. ***Problem Definition***    2. ***Preliminary Investigation***    3. ***Feasibility Study***    4. ***Project Planning***    5. ***Project Scheduling***    6. ***Software Requirement Specification***    7. ***Functional Requirements***    8. ***Software Engineering Paradigm***    9. ***Data model description***      1. ***System Design***     1. ***Modularization Details***    2. ***Database Design***    3. ***Procedural Design***    4. ***User Interface Design***    5. ***Outputs of the Report***      1. ***Coding***     1. ***Complete Project Coding***    2. ***Error Handling*** | **6**    **7**  8    **9**  **10-11**  **12-14**  **15-19**  **20**  **21-24**  **25-26**  **27-32**  **33**      34-35  **36**  37  **38-39**  **40**      **41-52**  **53** |
| ***5.3. Code Improvement***  ***5.4. Parameters Passing***  ***5.5. Validation Check***   1. ***Testing***     1. ***Testing Strategies***    2. ***Conduction Test Cases*** 2. ***System Security Measure***     1. ***Security Strategies***    2. ***Interface Security***    3. ***Database Security*** 3. ***Cost Estimation of the Project***      1. ***Reports***      1. ***PERT Chart, Gantt Chart***     1. ***PERT Chart***    2. ***Gantt Chart***      1. ***Future Application of the Project***      1. ***Bibliography*** | **54**  **55-56**  **57**    **58**  **59**    **60**  **61**        **62**    **63**  **64-65**    **66-67**    **68** |

**INTRODUCTION OF THE PROJECT**

A **Hospital Management System (HMS)** is a comprehensive web-based application developed to automate and streamline the daily operations of a hospital or healthcare facility. Built using the powerful **Python Django framework**, this system offers a secure and scalable environment for managing various aspects of hospital administration. The application covers critical modules such as patient registration, appointment scheduling, medical history tracking, staff management, and billing. Django’s Model-View-Template (MVT) architecture allows for rapid development and clean separation of concerns, making it easier to maintain and expand the system. With Django's built-in admin interface, administrators can easily manage user roles, update records, and monitor system activity. This enhances the efficiency and accuracy of administrative tasks, reduces paperwork, and minimizes the chances of manual errors.

The backend of the system is supported by a **MySQL database**, which stores all the essential information in an organized and secure manner. This includes patient details, doctor profiles, staff information, appointment logs, and billing transactions. The use of MySQL ensures fast data retrieval, reliability, and support for complex queries, making it suitable for large-scale hospital operations. The system can be accessed by different users such as doctors, nurses, receptionists, and administrators, each with specific permissions and access levels. Features such as role-based login, medical report generation, prescription management, and real-time notifications further enhance the usability of the application. This project not only demonstrates practical implementation of Django and MySQL but also provides a valuable solution to improve hospital workflow, enhance patient care, and enable better decision-making through digitized data management.

4o

**OBJECTIVES**

1. Define hospital
2. Recording information about the Patients that come.
3. Generating bills.
4. Recording information related to diagnosis given to Patients.
5. Keeping record of the Immunization provided to children/patients.
6. Keeping information about various diseases and medicines available to cure them.

These are the various jobs that need to be done in a Hospital by the operational staff and Doctors. All these works are done on papers.

**Scope of the Project:-**

1. Information about Patients is done by just writing the Patients name, age and gender. Whenever the Patient comes up his information is stored freshly.
2. Bills are generated by recording price for each facility provided to Patient on a separate sheet and at last they all are summed up.
3. Diagnosis information to patients is generally recorded on the document, which contains Patient information. It is destroyed after some time period to decrease the paper load in the office.
4. Information about various diseases is not kept as any document. Doctors themselves do this job by remembering various medicines.

All this work is done manually by the receptionist and other operational staff and lot of papers are needed to be handled and taken care of. Doctors have to remember various medicines available for diagnosis and sometimes miss better alternatives as they can’t remember them at that time.

**PROBLEM DEFINATION**

To address these challenges, a **Hospital Management System** built using **Python (Django)** and a **MySQL database** is proposed. This web-based application aims to automate and integrate all hospital processes into a single platform. It ensures efficient handling of patient information, staff coordination, appointment scheduling, and billing while maintaining data accuracy, security, and accessibility for authorized users. The goal is to improve the overall workflow, reduce human error, and provide faster and better healthcare services.

**Key Areas:**

1. **Patient Management**
   * Registration of new patients
   * Viewing and updating patient medical history
   * Search and filter patient records
2. **Doctor and Staff Management**
   * Add/view doctor and staff profiles
   * Assign duties and manage schedules
   * Maintain specialization and contact details
3. **Appointment Scheduling**
   * Book, reschedule, or cancel appointments
   * Display available time slots for doctors
   * Send appointment reminders
4. **Medical Records and Reports**
   * Maintain detailed patient diagnosis and treatment records
   * Generate prescriptions and test reports
   * Secure access to past reports
5. **Billing and Payments**
   * Generate bills for consultation, tests, and treatments
   * Track payment history
   * Print or email invoices
6. **Authentication and Role Management**
   * Separate login access for Admin, Doctors, Nurses, and Receptionists
   * Role-based permissions and secure data access
   * Admin dashboard for monitoring activity
7. **Database Integration (MySQL)**
   * Structured and secure storage of hospital data
   * Efficient querying and data management
   * Backup and restore capabilities

**Preliminary Investigation**

The preliminary investigation is the initial phase of the system development process that helps identify the need for a new system and evaluates whether the proposed solution is feasible and beneficial. In the context of a Hospital Management System, this investigation aims to understand the current problems in hospital operations, gather requirements, explore alternatives, and determine the practicality of implementing a digital system using Django and MySQL.

**1. Problem Identification**

Most hospitals today still rely on manual or semi-digital systems for managing their daily operations, which leads to inefficiencies such as:

* Difficulty in maintaining patient records and retrieving them when needed
* Errors in billing, appointment scheduling, and report generation
* Lack of centralized data, leading to duplicated or lost information
* No proper access control for different users (e.g., admin, doctors, staff)

These issues affect the quality of patient care and result in operational delays and resource wastage.

**2. Objective of the System**

The primary objective of the proposed system is to develop a web-based Hospital Management System that:

* Automates and digitizes hospital operations
* Centralizes all patient, doctor, and staff data in a secure database
* Supports real-time appointment scheduling, medical records tracking, and billing
* Ensures role-based access for different users
* Provides an intuitive and user-friendly interface

**3. Feasibility Study**

* Technical Feasibility: Python and Django provide a powerful and scalable backend, while MySQL ensures efficient data storage and retrieval. These technologies are open-source and widely supported, making them suitable for development.
* Operational Feasibility: The system will significantly improve hospital workflow, reduce manual errors, and enhance the patient experience. It is easy to train hospital staff to use a simple, well-designed interface.
* Economic Feasibility: As the project uses open-source technologies, development costs are minimized. Over time, the system can reduce operational costs related to paperwork, delays, and mismanagement.

**4. Data Gathering Methods**

To gather relevant information for system development, the following methods were considered:

* Interviews with hospital staff and administrators to understand daily challenges
* Observation of current workflows to identify inefficiencies
* Review of existing software solutions to determine common features and gaps

**6. Feasibility Analysis**

| **Criteria** | **Analysis** |
| --- | --- |

OPERATING SYSTEM : Windows or any equivalent OS

FRONT END : HTML, CSS, JavaScript query

SERVER SIDE SCRIPT : Python

DATABASE : MySQL

FRAMEWORK : Django

1. **Expected Benefits**

 Automates daily hospital tasks such as appointment booking, billing, and patient record management.

 Reduces paperwork and manual data entry.

**9. Constraints and Assumptions**

* The system will be developed using Python (Django) and MySQL only.
* The application will be web-based, accessible through modern browsers only.

Users (admins, doctors, receptionists) have basic knowledge of how to operate a computer and use web applications.

**FEASIBILITY STUDY**

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are:

**3.3.1 Economic Feasibility**

This study is carried out to check the economic impact will have on the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products have to be purchased.

**3.3.2 Technical Feasibility**

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes for the implementing this system.

**3.3.3 Operational Feasibility**

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

**DESIGN**

4.1 SYSTEM DESIGN:

4.1.1INTRODUCTION TO UML:

**UML Design**

The Unified Modeling Language (UML) is a standard language for specifying, visualizing, constructing, and documenting the software system and its components. It is a graphical language, which provides a vocabulary and set of semantics and rules. The UML focuses on the conceptual and physical representation of the system. It captures the decisions and understandings about systems that must be constructed. It is used to understand, design, configure, maintain, and control information about the systems.

The UML is a language for:

* Visualizing
* Specifying
* Constructing
* Documenting

**Visualizing**

Through UML we see or visualize an existing system and ultimately we visualize how the system is going to be after implementation. Unless we think, we cannot implement. UML helps to visualize, how the components of the system communicate and interact with each other.

**Specifying**

Specifying means building, models that are precise, unambiguous and complete UML addresses the specification of all the important analysis design, implementation decisions that must be made in developing and deploying a software system.

**Constructing**

UML models can be directly connected to a variety of programming language through mapping a model from UML to a programming language like JAVA or C++ or VB. Forward Engineering and Reverse Engineering is possible through UML.

**Documenting**

The Deliverables of a project apart from coding are some Artifacts, which are critical in controlling, measuring and communicating about a system during its developing requirements, architecture, desire, source code, project plans, tests, prototypes releasers, etc...

**4. Coding**

* **Backend**:
  + Django server routes
  + MySQL database connection
  + Predict disease logic using SQL queries.
* **Frontend**:
  + Form for entering symptoms.
  + Buttons for submitting and speaking symptoms.
  + Table for showing results.
* **Speech Recognition**:
  + Using JavaScript’s SpeechRecognition API.

**5. Security**

* **User Authentication**:
  + Login/Signup using session management.
  + User passwords securely stored (in future: consider password hashing).
* **Data Protection**:
  + Sessions to prevent unauthorized access.
  + Database access through parameterized queries to prevent SQL injection.

**TYPES OF TESTING**:

**Unit testing:**

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

**Integration testing:**

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

**Functional test:** Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

**System Test**:

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

**White Box Testing:**

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

**Black Box Testing:**

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

**Test strategy and approach**

Field testing will be performed manually and functional tests will be written in detail.

**Test objectives**

* All field entries must work properly.
* Pages must be activated from the identified link.
* The entry screen, messages and responses must not be delayed.

**Features to be tested**

* Verify that the entries are of the correct format
* No duplicate entries should be allowed
* All links should take the user to the correct page.

# Integration Testing:

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

**Test Results:**

All the test cases mentioned above passed successfully. No defects encountered.

**Acceptance Testing:**

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

**Test Results:**

All the test cases mentioned above passed successfully. No defects encountered.

**Software Requirement Specification**

**1.Project Title: Hospital Management System**

**Technology Used: Python, Django, MySQL**

**1. Introduction**

**1.1 Purpose**

**The purpose of this document is to define the software requirements for the Hospital Management System. This system will streamline the operations of a hospital by automating the management of patients, doctors, appointments, billing, and medical records.**

**1.2 Scope**

**The system will provide interfaces for Admin, Doctors, Receptionists, and Patients. It will handle tasks such as patient registration, appointment scheduling, diagnosis recording, bill generation, and user role management.**

**1.3 Definitions, Acronyms, and Abbreviations**

**HMS: Hospital Management System**

**DBMS: Database Management System**

**GUI: Graphical User Interface**

**SRS: Software Requirements Specification**

**---**

**2. Overall Description**

**2.1 Product Perspective**

**The HMS is a web-based application developed using Django (backend) and MySQL (database). It integrates multiple modules into a single platform accessible via browsers.**

**2.2 Product Functions**

**Patient registration and record maintenance**

**Doctor schedule and patient diagnosis**

**Appointment scheduling**

**Billing and payment management**

**Admin dashboard for system monitoring**

**2.3 User Characteristics**

**Admin: Manages the system, users, and settings**

**Doctor: Views appointments, adds diagnoses and prescriptions**

**Receptionist: Handles patient registration and appointments**

**Patient: Books appointments and views medical records**

**2.4 Constraints**

**Only users with valid credentials can access the system**

**Internet connection required**

**Limited to CRUD operations for users and records**

**2.5 Assumptions and Dependencies**

**Users are familiar with web applications**

**Deployed on a server running Django and MySQL**

**Browser-based access**

**---**

**3. Specific Requirements**

**3.1 Functional Requirements**

**Login System: Secure authentication and role-based access**

**Patient Module: Add/update/view patient details**

**Doctor Module: View patients, write prescriptions**

**Appointment Module: Schedule, reschedule, and cancel appointments**

**Billing Module: Generate and manage invoices**

**Admin Module: Manage users, view reports**

**3.2 Non-Functional Requirements**

**Performance: Fast response time (≤ 2 seconds per action)**

**Scalability: Should support growing number of users and records**

**Security: Password hashing, input validation, and role-based access**

**Reliability: 24/7 availability with error recovery features**

**3.3 Interface Requirements**

**User Interface: Browser-based interface with HTML/CSS/Bootstrap**

**Database Interface: MySQL for data storage and retrieval**

**External Interfaces: Email/SMS APIs (for notifications – optional)**

**---**

**4. System Features**

**User Management: Create/edit/delete users with roles**

**Search Functionality: Quickly find patient/doctor records**

**Notifications: Reminders for appointments and billing (optional)**

**Reports: Generate patient, appointment, and billing reports**

**5. Software and Hardware Requirements**

**5.1 Software Requirements**

**Python 3.x**

**Django framework**

**MySQL database**

**HTML, CSS, JavaScript, Bootstrap**

**Web browser (Chrome, Firefox, etc.)**

**5.2 Hardware Requirements**

**4GB RAM minimum**

**Dual-core processor or higher**

**500MB disk space (for local development)**

**Functional Requirements Document (FRD)**

**Project Title: Hospital management System in python**

**1. User Authentication & Authorization**

* The system shall allow users to register and login securely.
* The system shall authenticate users based on username and password.
* The system shall authorize access based on user roles (Admin, Doctor, Patient, Receptionist).

**2. Patient Management**

* The system shall allow staff to register new patients with their personal and medical details.
* The system shall allow updating and viewing patient records.
* The system shall allow patients to view their own history, prescriptions, and appointments.

**3. Doctor Management**

* The system shall allow the admin to add, update, or remove doctor profiles.
* The system shall allow doctors to view their schedules and assigned patients.
* The system shall allow doctors to add diagnoses, prescriptions, and notes for each patient visit.

**4. Appointment Management**

* The system shall allow patients or receptionists to schedule, reschedule, or cancel appointments.
* The system shall check doctor availability before confirming appointments.
* The system shall notify users about confirmed or cancelled appointments.

**5. Medical Records Management**

* The system shall allow doctors to create, view, and update patient medical records.
* The system shall store historical data for each patient visit, including prescriptions and diagnoses.
* The system shall allow uploading and viewing of test reports and other documents.

**6. Billing and Payment**

* The system shall generate bills based on services availed by the patient (consultation, tests, etc.).
* a system shall display payment status (paid/pending).
* The system shall generate downloadable invoices.

**7. Admin Controls**

* The system shall allow the admin to manage departments, user roles, and system settings.
* The system shall allow the admin to view reports and monitor user activity.
* The system shall allow the admin to view and manage appointments, billing, and staff records.

**8. Notifications and Alerts**

* The system shall send alerts/notifications to patients for appointment confirmation or reminders.
* The system shall notify doctors of new appointments or test results.
* The system shall send billing reminders or updates to patients.

**9. Search and Filter**

* The system shall allow searching for patients, doctors, or appointments using filters (date, name, department, etc.).

**10. Reporting**

* The system shall generate daily, weekly, or monthly reports on patient visits, billing, and doctor schedules.

**Optional/Future Functional Requirements:**

* Pharmacy management: Add/manage medicine inventory and issue prescriptions.
* Lab test management: Allow lab staff to upload test results and track samples.
* Feedback module: Allow patients to submit feedback or ratings for doctors and services.

**4.2 UML Approach**

**USE CASE DIAGRAM:**

A use case diagram in the Unified Modeling Language (UML) is type of behavioral diagram defined by and created from a use-case analysis. its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals(represented as use cases) ,and any dependencies between those use cases.

Use case diagrams are formally included in two modeling languages defined by the OMG:theunfied modeling language (UML) and the systems modeling language(sysML)

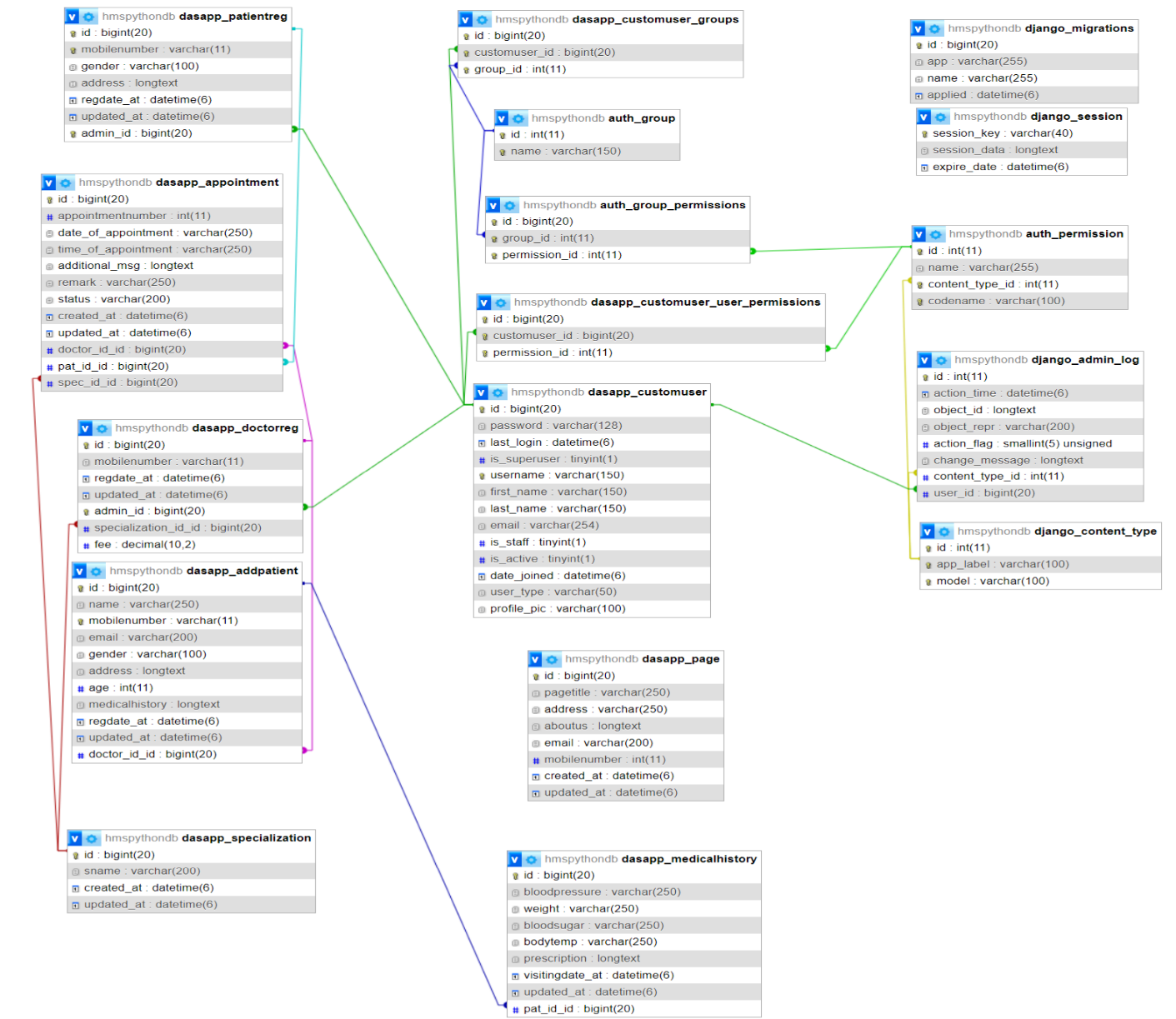
**Use Case Diagram Admin**

**Doctor Use Case Diagram**

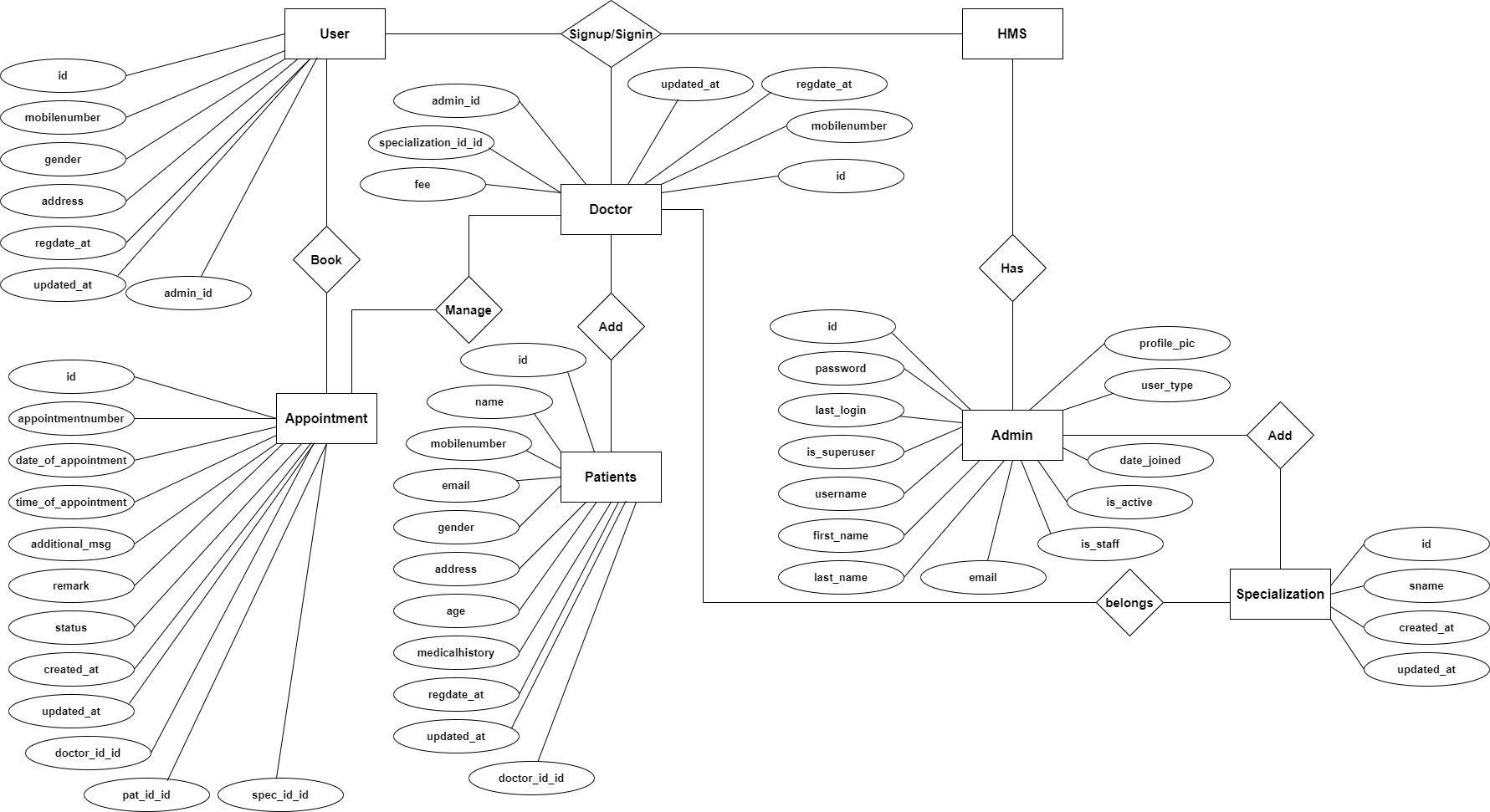
**Patient Use Case Diagram**

**Class Diagram**:

A Class is a category or group of things that has similar attributes and common behavior. A Rectangle is the icon that represents the class it is divided into three areas. The upper most area contains the name, the middle; area contains the attributes and the lowest areas show the operations. Class diagrams provides the representation that developers work from. Class diagrams help on the analysis side, too.

****

**ER Diagram**

****

**Deployment diagram:**

A **Deployment Diagram** shows the configuration of run-time processing nodes and the components that live on them. Deployment diagrams address the static deployment view of architecture. They are related to component diagrams in that a node typically encloses one or more components.

****

**Sequence Diagrams:**

The state diagram shows the states of an object and represents activities as arrows connecting the states. The Activity Diagram highlights the activities. Each activity is represented by a rounded rectangle-narrower and more oval-shaped than the state icon. An arrow represents the transition from the one activity to the next. The activity diagram has a starting point represented by filled-in circle, and an end point represented by bulls eye.



**Modules and Their Description for Hospital Management System in python System**

**1. Admin Module**

Description:  
The Admin is the superuser who has complete control over the system. This module allows the admin to manage all users (doctors, staff, patients), monitor system activities, and configure system settings.

Key Features:

* Add, update, and delete users
* Assign roles and permissions
* View system reports and logs
* Manage hospital departments and specialties

**2. Doctor Module**

**Description:**This module is designed for doctors to access and manage their patients' data. It helps them track medical history, schedule appointments, and write prescriptions.

Key Features:

* View and manage assigned patients
* Add medical records, diagnoses, and prescriptions
* Manage appointment schedules
* Access test reports and previous visit history

**3. Patient Module**

Description:  
Patients can register/login to the system and manage their interactions with the hospital. This module improves patient engagement and convenience.

Key Features:

* Online registration and login
* Book, view, or cancel appointments
* View medical history, prescriptions, and bills
* Receive alerts/notifications for upcoming visits

**4. Receptionist/Front Desk Module**

Description:  
Receptionists manage patient registration, appointments, and basic front-desk operations. This module simplifies check-ins and schedules.

Key Features:

* Register new patients and update existing profiles
* Schedule and reschedule appointments
* Search patient details and check availability
* Generate patient visit slips

**5. Appointment Management Module**

Description:  
A centralized system to handle the booking and tracking of appointments between patients and doctors.

Key Features:

* Doctor-wise time slot allocation
* Real-time availability status
* **SMS/email notifications for confirmation or reminders**
* **Calendar view for doctors and staff**

**6. Medical Records Module**

Description:  
Stores complete health data of patients including visit history, diagnosis, treatment, and lab reports.

Key Features:

* Add/view patient diagnosis
* Upload medical test reports
* Track patient vitals and history
* Confidential and role-based access

**7. Billing and Invoice Module**

Description:  
This module manages all financial transactions within the hospital system. It includes payments for consultation, treatment, and medicines.

Key Features:

* Generate bills and invoices
* Track payment status (paid/pending)
* Add consultation and treatment charges
* Download or print invoices

**8. Pharmacy Module *(optional/future scope)***

**Description:**Manages medicine inventory, prescriptions, and issue tracking.

Key Features:

* Maintain stock levels and expiry dates
* Issue medicine based on doctor prescriptions
* Record sales and generate pharmacy bills

**9. Laboratory Module *(optional/future scope)***

Description:  
Handles diagnostic tests and report management for patients.

Key Features:

* Request lab tests for patients
* Upload test results and reports
* Notify doctors and patients once results are ready

**10. Authentication and User Management Module**

Description:  
Manages secure login/logout functionality and user roles.

Key Features:

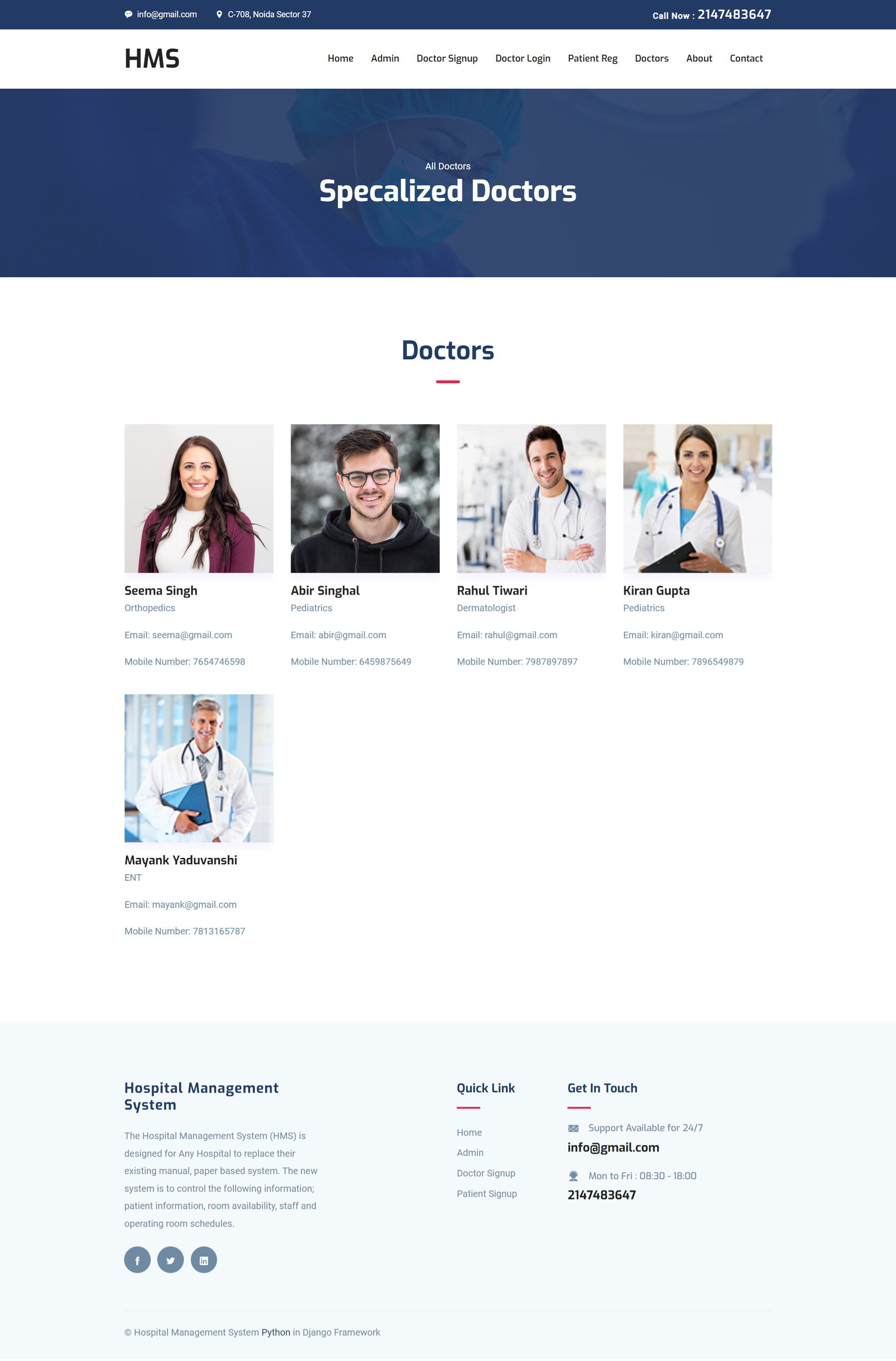
* User registration and login
* Password reset and update
* Role-based access control (Admin, Doctor, Receptionist, Patient)
* Secure session handling

**Input Screens**

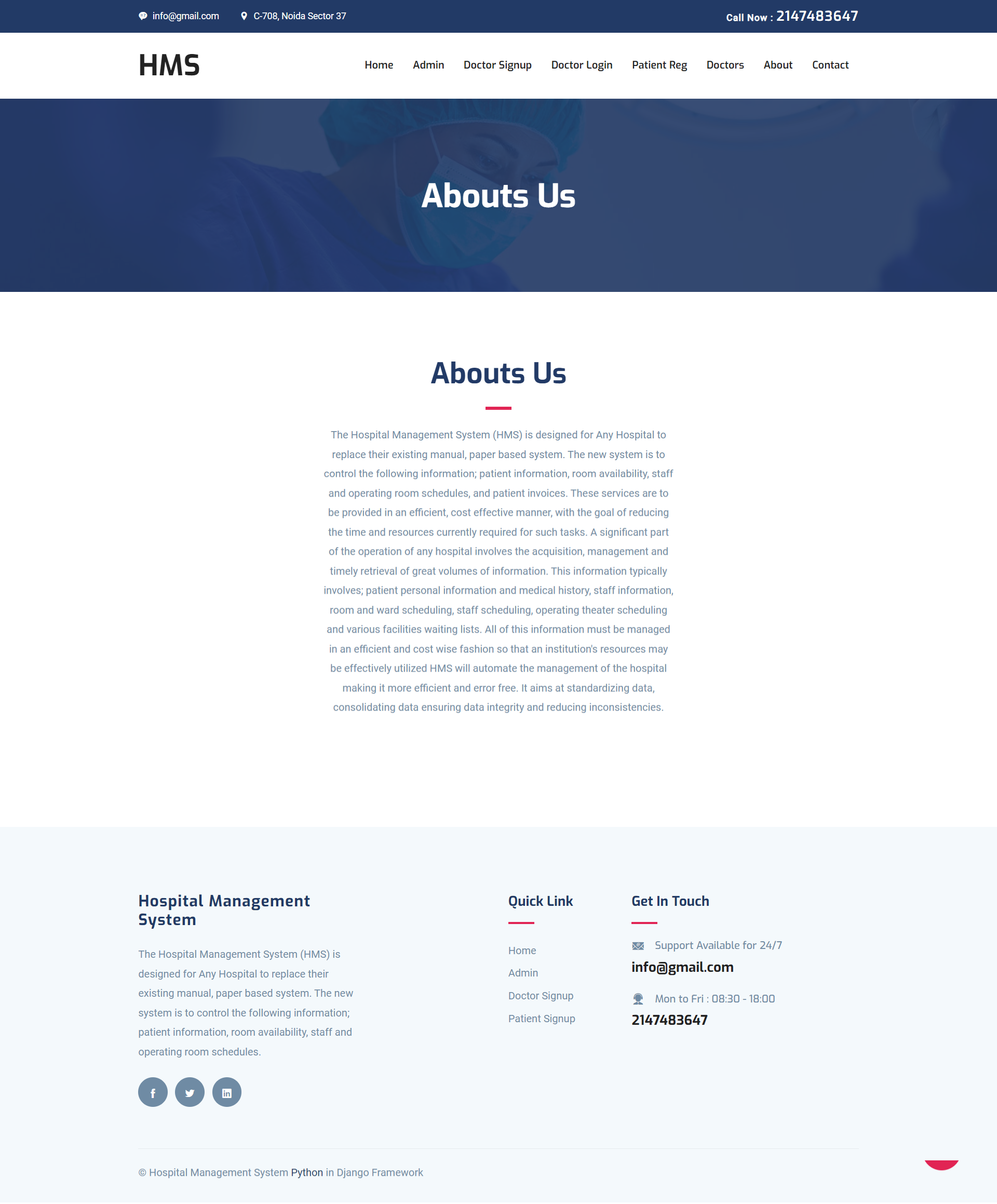
**Home Page**

****

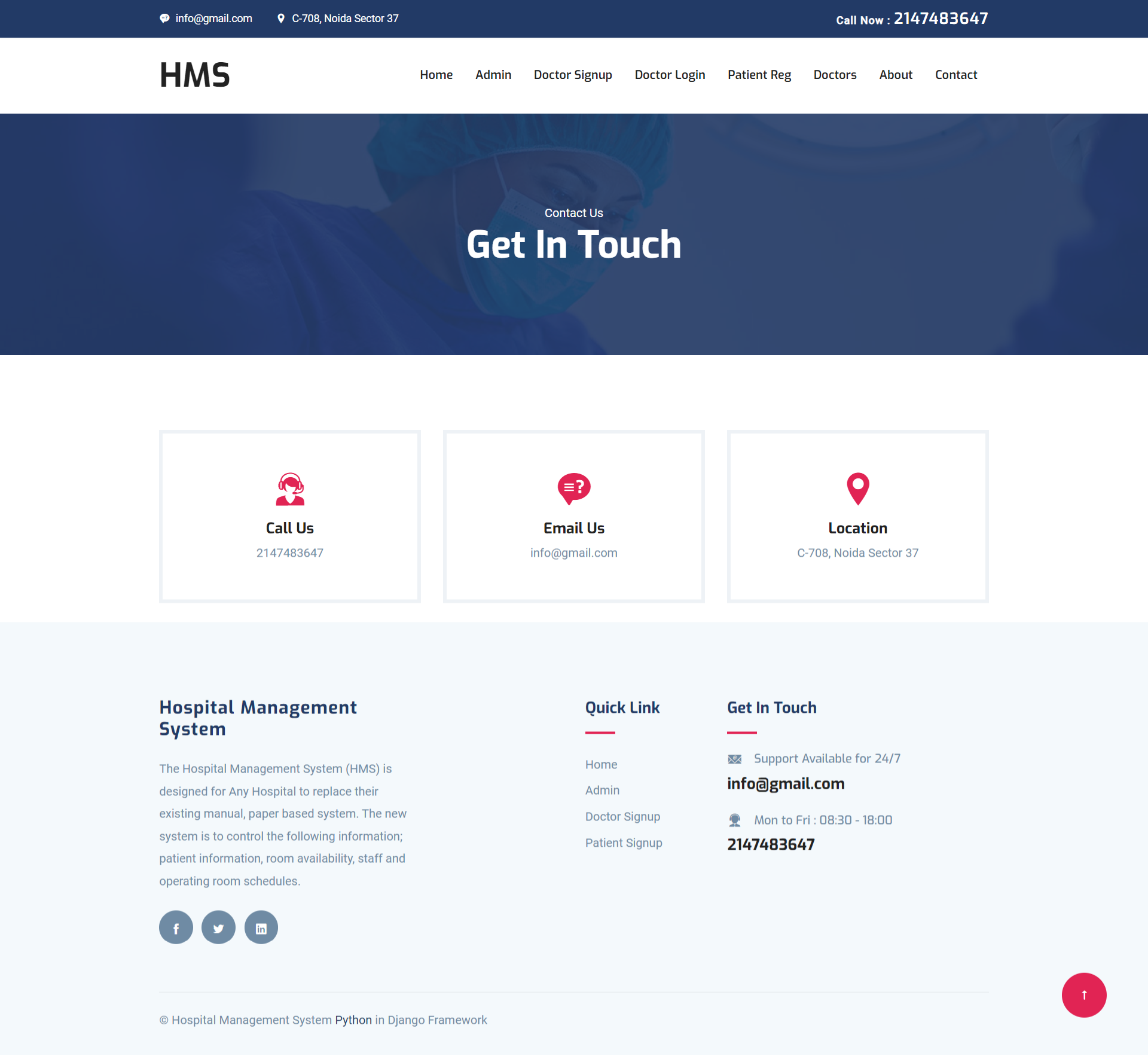
**Doctor Details**

****

**About Us**

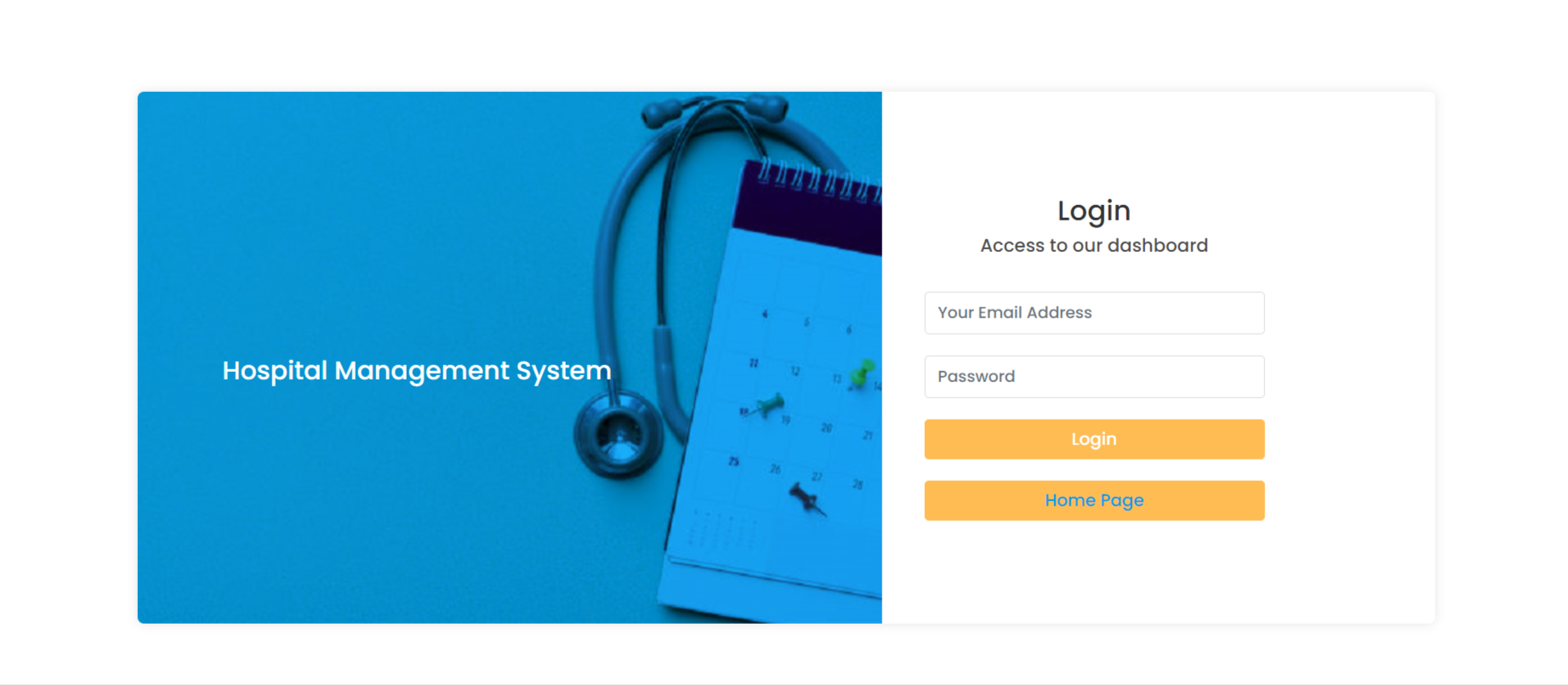
****

**Contact Us**

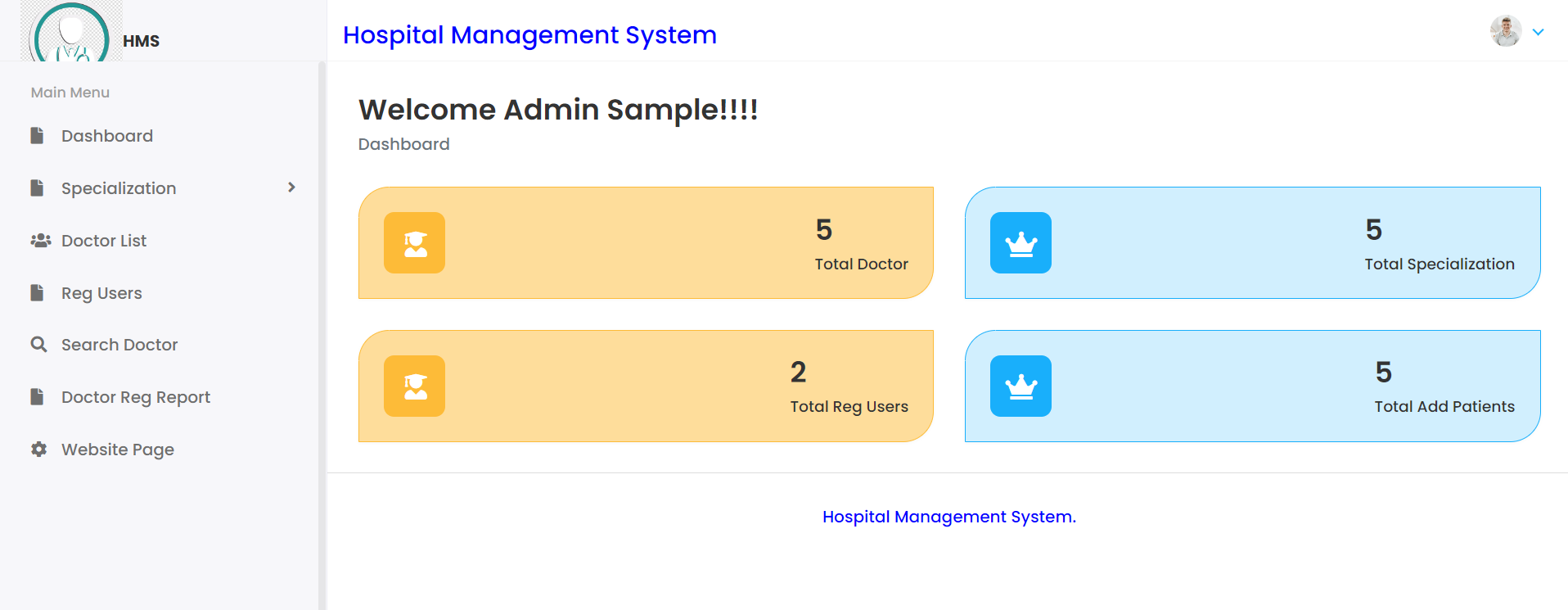
****

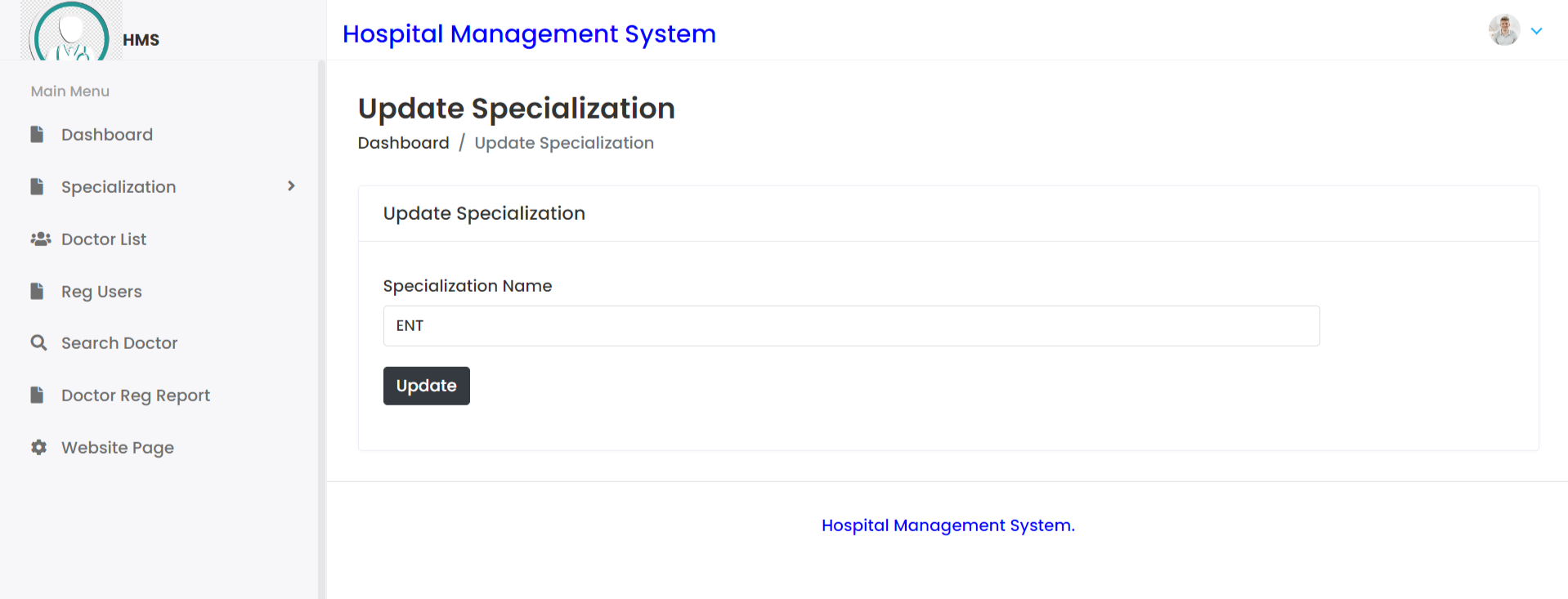
**Admin Panel**

**Login Page**

****

**Dashboard**

****



CODING

SAMLE CODE:

Index.html

{% extends 'userbase.html' %}

{% load static %}

{% block content %}

<section class="banner">

    <div class="container">

        <div class="row">

            <div class="col-lg-6 col-md-12 col-xl-7">

                <div class="block">

                    <div class="divider mb-3"></div>

                    <span class="text-uppercase text-sm letter-spacing ">Total Health care solution</span>

                    <h1 class="mb-3 mt-3">Your most trusted health partner</h1>

                </div>

            </div>

        </div>

    </div>

</section>

<section class="features">

    <div class="container">

        <div class="row">

            <div class="col-lg-12">

                <div class="feature-block d-lg-flex">

                    <div class="feature-item mb-5 mb-lg-0">

                        <div class="feature-icon mb-4">

                            <i class="icofont-surgeon-alt"></i>

                        </div>

                        <span>24 Hours Service</span>

                        <h4 class="mb-3">Online Appoinment</h4>

                        <p class="mb-4">Get ALl time support for emergency.We have introduced the principle of family medicine.</p>

                        <a href="{% url 'patreg' %}" class="btn btn-main btn-round-full">Make a appoinment</a>

                    </div>

                    <div class="feature-item mb-5 mb-lg-0">

                        <div class="feature-icon mb-4">

                            <i class="icofont-ui-clock"></i>

                        </div>

                        <span>Timing schedule</span>

                        <h4 class="mb-3">Working Hours</h4>

                        <ul class="w-hours list-unstyled">

                            <li class="d-flex justify-content-between">Sun - Wed : <span>8:00 - 17:00</span></li>

                            <li class="d-flex justify-content-between">Thu - Fri : <span>9:00 - 17:00</span></li>

                            <li class="d-flex justify-content-between">Sat - sun : <span>10:00 - 17:00</span></li>

                        </ul>

                    </div>

                    <div class="feature-item mb-5 mb-lg-0">

                        <div class="feature-icon mb-4">

                            <i class="icofont-support"></i>

                        </div>

                        <span>Emegency Cases</span>

                        <h4 class="mb-3">1-800-700-6200</h4>

                        <p>Get ALl time support for emergency.We have introduced the principle of family medicine.Get Conneted with us for any urgency .</p>

                    </div>

                </div>

            </div>

        </div>

    </div>

</section>

<section class="section service gray-bg">

    <div class="container">

        <div class="row justify-content-center">

            <div class="col-lg-7 text-center">

                <div class="section-title">

                    <h2>Award winning patient care</h2>

                    <div class="divider mx-auto my-4"></div>

                            </div>

            </div>

        </div>

        <div class="row">

            <div class="col-lg-4 col-md-6 col-sm-6">

                <div class="service-item mb-4">

                    <div class="icon d-flex align-items-center">

                        <i class="icofont-laboratory text-lg"></i>

                        <h4 class="mt-3 mb-3">Laboratory services</h4>

                    </div>

                    <div class="content">

                    </div>

                </div>

            </div>

            <div class="col-lg-4 col-md-6 col-sm-6">

                <div class="service-item mb-4">

                    <div class="icon d-flex align-items-center">

                        <i class="icofont-heart-beat-alt text-lg"></i>

                        <h4 class="mt-3 mb-3">Heart Disease</h4>

                    </div>

                    <div class="content">

                    </div>

                </div>

            </div>

            <div class="col-lg-4 col-md-6 col-sm-6">

                <div class="service-item mb-4">

                    <div class="icon d-flex align-items-center">

                        <i class="icofont-tooth text-lg"></i>

                        <h4 class="mt-3 mb-3">Dental Care</h4>

                    </div>

                    <div class="content">

                    </div>

                </div>

            </div>

            <div class="col-lg-4 col-md-6 col-sm-6">

                <div class="service-item mb-4">

                    <div class="icon d-flex align-items-center">

                        <i class="icofont-crutch text-lg"></i>

                        <h4 class="mt-3 mb-3">Body Surgery</h4>

                    </div>

                    <div class="content">

                    </div>

                </div>

            </div>

            <div class="col-lg-4 col-md-6 col-sm-6">

                <div class="service-item mb-4">

                    <div class="icon d-flex align-items-center">

                        <i class="icofont-brain-alt text-lg"></i>

                        <h4 class="mt-3 mb-3">Neurology Sargery</h4>

                    </div>

                    <div class="content">

                    </div>

                </div>

            </div>

            <div class="col-lg-4 col-md-6 col-sm-6">

                <div class="service-item mb-4">

                    <div class="icon d-flex align-items-center">

                        <i class="icofont-dna-alt-1 text-lg"></i>

                        <h4 class="mt-3 mb-3">Gynecology</h4>

                    </div>

                    <div class="content">

                    </div>

                </div>

            </div>

        </div>

    </div>

</section>

<!-- Appointment End -->

{% endblock %}

Login.html

<!DOCTYPE html>

<html lang="en">

    {% load static %}

   <!-- Mirrored from preschool.dreamguystech.com/html-template/login.html by HTTrack Website Copier/3.x [XR&CO'2014], Thu, 28 Oct 2021 11:11:39 GMT -->

   <head>

      <title>HMS - Login</title>

      <link rel="stylesheet" href="https://fonts.googleapis.com/css2?family=Poppins:ital,wght@0,500;0,600;0,700;1,400&amp;display=swap">

      <link rel="stylesheet" href="{% static 'assets/plugins/bootstrap/css/bootstrap.min.css'%}">

      <link rel="stylesheet" href="{% static 'assets/plugins/fontawesome/css/fontawesome.min.css'%}">

      <link rel="stylesheet" href="{% static 'assets/plugins/fontawesome/css/all.min.css'%}">

      <link rel="stylesheet" href="{% static 'assets/css/style.css'%}">

   </head>

   <body>

      <div class="main-wrapper login-body">

         <div class="login-wrapper">

            <div class="container">

               <div class="loginbox">

                  <div class="login-left">

                     <p style="text-align: left;">Hospital Management System</p>

                  </div>

                  <div class="login-right">

                     <div class="login-right-wrap">

                        <h1>Login</h1>

                        <p class="account-subtitle">Access to our dashboard</p>

                        {% if messages %}

                            {% for message in messages %}

                             {% if message.tags == 'error' %}

                           <div class="alert alert-warning alert-dismissible fade show" role="alert">

                          {{message}}

                         <button type="button" class="close" data-dismiss="alert" aria-label="Close">

                         <span aria-hidden="true">&times;</span>

                             </button>

                              </div>

                           {% endif %}

                            {% endfor %}

                           {% endif %}

                               {% if messages %}

                            {% for message in messages %}

                             {% if message.tags == 'success' %}

                           <div class="alert alert-warning alert-dismissible fade show" role="alert">

                          {{message}}

                         <button type="button" class="close" data-dismiss="alert" aria-label="Close">

                         <span aria-hidden="true">&times;</span>

                             </button>

                              </div>

                           {% endif %}

                            {% endfor %}

                           {% endif %}

                        <form action="{% url 'doLogin' %}"  method="POST" >

                            {% csrf\_token %}

                           <div class="form-group">

                            <input type="email" class="form-control" placeholder="Your Email Address" name="email">

                           </div>

                           <div class="form-group">

                            <input type="password" class="form-control" placeholder="Password" name="password">

                           </div>

                           <div class="form-group">

                              <button class="btn btn-primary btn-block" type="submit">Login</button>

                           </div>

                        </form>

                        <div class="btn btn-primary btn-block"><a href="{% url 'index' %}">Home Page</a></div>

                     </div>

                  </div>

               </div>

            </div>

         </div>

      </div>

      <script src="{% static 'assets/js/jquery-3.6.0.min.js'%}"></script>

      <script src="{% static 'assets/js/popper.min.js'%}"></script>

      <script src="{% static 'assets/plugins/bootstrap/js/bootstrap.min.js'%}"></script>

      <script src="{% static 'assets/js/script.js'%}"></script>

   </body>

   <!-- Mirrored from preschool.dreamguystech.com/html-template/login.html by HTTrack Website Copier/3.x [XR&CO'2014], Thu, 28 Oct 2021 11:11:40 GMT -->

</html>

profile.html

{% extends 'base.html' %}

{% block content %}

<div class="content container-fluid">

    {% if user.user\_type == '1' %}

    <div class="page-header">

    <div class="row">

    <div class="col">

    <h3 class="page-title">Profile Details</h3>

    <ul class="breadcrumb">

    <li class="breadcrumb-item"><a href="{% url 'admin\_home' %}">Dashboard</a></li>

    <li class="breadcrumb-item active">Profile Details</li>

    </ul>

    </div>

    </div>

    </div>

     {% elif user.user\_type == '2' %}

    <div class="page-header">

    <div class="row">

    <div class="col">

    <h3 class="page-title">Profile Details</h3>

    <ul class="breadcrumb">

    <li class="breadcrumb-item"><a href="{% url 'doctor\_home' %}">Dashboard</a></li>

    <li class="breadcrumb-item active">Profile Details</li>

    </ul>

    </div>

    </div>

    </div>

    {% else  %}

    <div class="page-header">

        <div class="row">

        <div class="col">

        <h3 class="page-title">Profile Details</h3>

        <ul class="breadcrumb">

        <li class="breadcrumb-item"><a href="{% url 'userhome' %}">Dashboard</a></li>

        <li class="breadcrumb-item active">Profile Details</li>

        </ul>

        </div>

        </div>

        </div>{% endif %}

    <div class="row">

    <div class="col-lg-12">

    <div class="card">

    <div class="card-header">

    <h5 class="card-title">Profile Details</h5>

    </div>

    <div class="card-body">

        {% if messages %}

                        {% for message in messages %}

                         {% if message.tags == 'error' %}

                       <div class="alert alert-warning alert-dismissible fade show" role="alert">

                      {{message}}

                     <button type="button" class="close" data-dismiss="alert" aria-label="Close">

                     <span aria-hidden="true">&times;</span>

                         </button>

                          </div>

                       {% endif %}

                        {% endfor %}

                       {% endif %}

                           {% if messages %}

                        {% for message in messages %}

                         {% if message.tags == 'success' %}

                       <div class="alert alert-warning alert-dismissible fade show" role="alert">

                      {{message}}

                     <button type="button" class="close" data-dismiss="alert" aria-label="Close">

                     <span aria-hidden="true">&times;</span>

                         </button>

                          </div>

                       {% endif %}

                        {% endfor %}

                       {% endif %}

                       <form method="POST" action="{% url 'profile\_update' %}" enctype="multipart/form-data">

                        {% csrf\_token %}

    <div class="form-group row">

    <label class="col-form-label col-md-2">Profile Pic</label>

    <div class="col-md-10">

        <input type="file" class="form-control" name="profile\_pic">

    </div>

    </div>

    <div class="form-group row">

    <label class="col-form-label col-md-2">First Name</label>

    <div class="col-md-10">

        <input type="text" class="form-control" name="first\_name" value="{{user.first\_name}}">

    </div>

    </div>

    <div class="form-group row">

    <label class="col-form-label col-md-2">Last Name</label>

    <div class="col-md-10">

        <input type="text" class="form-control" name="last\_name" value="{{user.last\_name}}">

    </div>

    </div>

    <div class="form-group row">

    <label class="col-form-label col-md-2">Email</label>

    <div class="col-md-10">

        <input type="email" class="form-control" readonly="True" name="email" value="{{user.email}}">

    </div>

    </div>

    <div class="form-group row">

    <label class="col-form-label col-md-2">Username</label>

    <div class="col-md-10">

        <input type="text" class="form-control" readonly="True" name="username" value="{{user.username}}">

    </div>

    </div>

    <div class="form-group row">

        <div class="col-sm-10">

            <button type="submit" class="btn btn-dark">Update</button>

        </div>

    </div>

    </form>

    </div>

    </div>

    </div>

    </div>

    </div>

{% endblock %}

"""

Django settings for docappsystem project.

Generated by 'django-admin startproject' using Django 5.0.2.

For more information on this file, see

https://docs.djangoproject.com/en/5.0/topics/settings/

For the full list of settings and their values, see

https://docs.djangoproject.com/en/5.0/ref/settings/

"""

from pathlib import Path

import os

# Build paths inside the project like this: BASE\_DIR / 'subdir'.

BASE\_DIR = Path(\_\_file\_\_).resolve().parent.parent

# Quick-start development settings - unsuitable for production

# See https://docs.djangoproject.com/en/5.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!

SECRET\_KEY = 'django-insecure-=u!!em!u$#9d=ew1uzeq&=90w(%62nx5b)9j66kbhh2\*ee\_\_il'

# SECURITY WARNING: don't run with debug turned on in production!

DEBUG = True

ALLOWED\_HOSTS = []

# Application definition

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'dasapp',

]

MIDDLEWARE = [

    'django.middleware.security.SecurityMiddleware',

    'django.contrib.sessions.middleware.SessionMiddleware',

    'django.middleware.common.CommonMiddleware',

    'django.middleware.csrf.CsrfViewMiddleware',

    'django.contrib.auth.middleware.AuthenticationMiddleware',

    'django.contrib.messages.middleware.MessageMiddleware',

    'django.middleware.clickjacking.XFrameOptionsMiddleware',

]

ROOT\_URLCONF = 'docappsystem.urls'

TEMPLATES = [

    {

        'BACKEND': 'django.template.backends.django.DjangoTemplates',

        'DIRS': ['templates'],

        'APP\_DIRS': True,

        'OPTIONS': {

            'context\_processors': [

                'django.template.context\_processors.debug',

                'django.template.context\_processors.request',

                'django.contrib.auth.context\_processors.auth',

                'django.contrib.messages.context\_processors.messages',

            ],

        },

    },

]

WSGI\_APPLICATION = 'docappsystem.wsgi.application'

# Database

# https://docs.djangoproject.com/en/4.2/ref/settings/#databases

# DATABASES = {

#     'default': {

#         'ENGINE': 'django.db.backends.sqlite3',

#         'NAME': BASE\_DIR / 'db.sqlite3',

#     }

# }

DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.mysql',

        'NAME': 'hospital',

        'USER': 'root',

        'PASSWORD': 'dipu123',

        'HOST': 'localhost',

        'PORT': '3306',

    }

}

# Password validation

# https://docs.djangoproject.com/en/5.0/ref/settings/#auth-password-validators

AUTH\_PASSWORD\_VALIDATORS = [

    {

        'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',

    },

    {

        'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',

    },

    {

        'NAME': 'django.contrib.auth.password\_validation.CommonPasswordValidator',

    },

    {

        'NAME': 'django.contrib.auth.password\_validation.NumericPasswordValidator',

    },

]

# Internationalization

# https://docs.djangoproject.com/en/5.0/topics/i18n/

LANGUAGE\_CODE = 'en-us'

TIME\_ZONE = 'UTC'

USE\_I18N = True

USE\_TZ = True

# Static files (CSS, JavaScript, Images)

# https://docs.djangoproject.com/en/5.0/howto/static-files/

STATIC\_URL = '/static/'

STATIC\_ROOT = '/static/'

STATICFILES\_DIRS = [

    os.path.join(BASE\_DIR,'static')

]

# Default primary key field type

# https://docs.djangoproject.com/en/5.0/ref/settings/#default-auto-field

DEFAULT\_AUTO\_FIELD = 'django.db.models.BigAutoField'

AUTH\_USER\_MODEL = 'dasapp.CustomUser'

**Error Handling :**

Error handling is crucial to ensure that the system functions reliably and provides clear feedback to users when something goes wrong. Proper error handling improves user experience, aids debugging, and prevents system crashes.

---

1. User Input Validation Errors

What it handles: Invalid or missing form fields (e.g., empty name, wrong email format).

How it is handled:

Django form validation is used to ensure correct input.

Custom error messages are shown next to form fields.

Required fields are marked, and input types are enforced (e.g., email, date, number).

---

2. Authentication Errors

What it handles: Incorrect login credentials, unauthorized access.

How it is handled:

Django displays specific messages like "Invalid username or password."

Unauthorized users are redirected to the login page.

Role-based access restrictions prevent unauthorized actions or page views.

---

3. Database Errors

What it handles: Issues like missing records, duplicate entries, or connection failures.

How it is handled:

Django ORM catches exceptions like DoesNotExist, IntegrityError, etc.

Custom error pages (404 for not found, 500 for server errors) are implemented.

Try-except blocks handle critical queries with fallback logic.

**Code Improvement**

**1. 1. Use Django Class-Based Views (CBVs) Instead of Function-Based Views (FBVs)**

**Why: CBVs reduce redundancy and allow reusability with mixins.**

**Example: Replace repeated logic for CRUD operations with Django's ListView, DetailView, CreateView, etc.**

**---**

**2. Modularize Code**

**Why: Keeps code organized, easier to maintain and scale.**

**How:**

**Separate concerns: use different apps for users, appointments, billing, etc.**

**Keep logic in views thin by moving business logic to services.py or utils.py.**

**---**

**3. Use Django Signals Appropriately**

**Why: Helps decouple logic like sending emails or creating logs when models are saved or deleted.**

**Example: Use post\_save for sending appointment confirmations.**

**---**

**4. Use Custom User Model**

**Why: Easier to manage roles (admin, doctor, patient, etc.) and add profile fields in future.**

**How:**

**Extend AbstractBaseUser or AbstractUser in your models.py.**

**---**

**5. Optimize Database Queries**

**Why: Avoid performance issues with large datasets.**

**How:**

**Use select\_related and prefetch\_related to reduce database hits.**

**Avoid nested loops that query the database inside them.**

**---**

**6. Implement Role-Based Access Control (RBAC) Using Decorators or Mixins**

**Why: Ensures users only access what they're allowed to.**

**How:**

**Create custom decorators or use Django’s UserPassesTestMixin for access restrictions.**

**---**

**7. Improve Forms Using Django’s ModelForms**

**Why: Reduces boilerplate and keeps validation clean.**

**How:**

**Use class Meta: in forms to bind them directly to models.**

**Summary Chart:**

VALIDATION CHECKS

**1. User Registration/Login Validation**

**Email Format Validation: Only valid email formats accepted (e.g., example@mail.com).**

**Password Strength Validation: Enforce minimum length (e.g., 8 characters), use of numbers, uppercase/lowercase, and special characters.**

**Unique Username/Email Check: Prevent duplicate registration.**

**Empty Field Check: No required field (name, email, password) should be left blank.**

**CSRF Token Validation: Protect forms from cross-site request forgery.**

**---**

**2. Patient Registration/Details Validation**

**Name Validation: Only allow alphabetic characters and spaces.**

**Phone Number Validation: Exactly 10 digits; no alphabets or symbols allowed.**

**Age Validation: Accept only reasonable age values (e.g., 0–120).**

**Date of Birth Format Check: Use standard formats (YYYY-MM-DD).**

**Gender Validation: Accept predefined choices only (Male, Female, Other).**

**---**

**3. Appointment Booking Validation**

**Doctor Availability Check: Prevent double-booking in the same time slot.**

**Time Slot Validation: Ensure the selected slot is within working hours and not in the past.**

**Required Field Check: Patient ID, doctor, date, and time must be filled.**

**Duplicate Appointment Check: Avoid duplicate entries for the same patient and doctor at the same time.**

**---**

**4. Medical Record Validation**

**Diagnosis Text Limit: Limit characters to prevent excessively long entries.**

**Prescription Format Validation: Ensure medicines, dosage, and duration are entered correctly.**

**Report Upload Validation: Accept only specific file types (PDF, JPG, PNG) with size limits (e.g., 5MB).**

**---**

**5. Billing/Invoice Validation**

**Service Charge Input Check: Only positive numeric values allowed.**

**Total Calculation Validation: Automatically compute totals based on service fees.**

**Duplicate Bill Check: Prevent multiple invoices for the same patient on the same date.**

**Payment Status Validation: Accept only predefined values (Paid, Unpaid, Pending).**

SYSTEM SECURITY

**Security is a critical aspect of any hospital management system, as it deals with sensitive data such as patient records, medical history, and billing information. The proposed system, built with Django and MySQL, incorporates multiple layers of security to ensure data protection, privacy, and secure access. Below are the key security measures and features included in the system:**

**1. Authentication and Authorization**

* **User Login System: Django's built-in authentication system provides secure login functionality for different roles (admin, doctor, staff, etc.).**
* **Role-Based Access Control: Each user has access only to specific modules based on their role. For example, doctors cannot access billing, and receptionists cannot access patient medical history.**

**2. Secure Password Handling**

* **Hashed Passwords: Django stores passwords using a strong hashing algorithm (PBKDF2 by default), ensuring that even if the database is compromised, raw passwords are not exposed.**
* **Password Validation: Enforces password complexity and strength rules to prevent weak passwords.**

**3. Session Management**

* **Session Timeout: User sessions expire after a period of inactivity to prevent unauthorized use.**
* **Secure Cookies: Session data is stored in secure cookies with flags like HttpOnly and Secure to prevent attacks like XSS and session hijacking.**

**4. Protection Against Common Web Vulnerabilities**

* **Cross-Site Request Forgery (CSRF): Django provides built-in CSRF protection to ensure that form submissions originate from trusted sources.**
* **Cross-Site Scripting (XSS): Django escapes output in templates by default, preventing malicious scripts from being executed.**
* **SQL Injection Protection: Django ORM abstracts direct SQL queries, reducing the risk of SQL injection attacks.**

**5. Database Security (MySQL)**

* **User Permissions: Database users have limited permissions, such as read/write access only to specific tables.**
* **Data Encryption: Optionally, sensitive fields in the database (like patient records) can be encrypted at rest.**
* **Backups: Regular, secure backups of the database to recover data in case of corruption or attack.**

**6. Admin Panel Security**

* **Restricted Access: Django's admin panel is accessible only by authorized superusers.**
* **Admin URL Obfuscation: Changing the default admin URL to a custom path to reduce exposure to automated attacks.**

COST ESTIMATION

**1. Academic/Prototype Version (Low-Budget Estimate)**

| **Component** | **Details** | **Estimated Cost (USD)** |
| --- | --- | --- |
| Development Tools | Python, Django, MySQL (all open-source) | $0 |
| IDE and Code Editor | VS Code, PyCharm Community Edition | $0 |
| Hosting (for testing) | Free Tier (Heroku, Render, or AWS Free Tier) | $0 |
| Domain (optional) | Temporary/Free subdomain (e.g., Heroku) | $0 |
| System Design & Documentation | Time investment only | $0 |
| Developer Time (self or team) | Student/academic contribution | $0 |

**Total Cost (Academic Use):** **~$0**  
*Note: The primary investment is time and effort.*

**2. Real-World Deployment Version (Basic Production Setup)**

| **Component** | **Details** | **Estimated Cost (USD)** |
| --- | --- | --- |
| Development Tools | Python, Django, MySQL (open-source) | $0 |
| Developer/Team Cost | Freelancer or team (est. 100–150 hours @ $10–$30/hr) | $1,000 – $4,500 |
| Hosting & Server | VPS (DigitalOcean, AWS, etc. – 1yr basic) | $100 – $300 |
| Domain Name | Annual registration (e.g., .com or .org) | $10 – $20 |
| SSL Certificate | Let’s Encrypt (Free) or paid SSL | $0 – $50 |
| Backup & Security Tools | Basic plan for backups/firewall | $50 – $150 |
| Testing & Maintenance | Ongoing updates/bug fixes (3-6 months support) | $200 – $500 |
| UI/UX Design (optional) | Paid templates or designer fee | $50 – $200 |

**Total Cost (Real-World Deployment):** **~$1,400 – $5,700+**

FUTURE SCOPE

The Hospital Management System (HMS) developed using Django and MySQL has significant potential for future expansion and improvement. As technology in healthcare continues to evolve, this system can be scaled and enhanced to meet the growing needs of hospitals and clinics. Below are some of the key areas for future development:

**1. Integration with Advanced Medical Technologies**

* Connect with medical devices (like ECG, X-ray, and diagnostic tools) to automatically store and analyze patient data.
* Enable integration with Electronic Health Record (EHR) and Health Information Exchange (HIE) systems.

**2. Mobile Application Support**

* Develop mobile apps for patients and doctors for better accessibility and convenience.
* Patients can book appointments, view reports, and receive notifications directly on their phones.

**3. Telemedicine and Virtual Consultation**

* Add video calling and chat features to enable remote consultations, especially beneficial for rural or remote patients.
* Allow doctors to prescribe medicine and monitor patients online.

**4. Pharmacy and Inventory Management**

* Include pharmacy module to manage medicine stock, expiry, and billing.
* Integrate inventory management for medical supplies and equipment.

**5. Insurance and Third-Party Billing Integration**

* Add integration with health insurance providers for claim processing.
* Automate billing systems for insured and non-insured patients.

**6. AI and Data Analytics**

* Use AI to analyze patient data and generate insights or alerts (e.g., detecting high-risk cases).
* Provide dashboards and visual analytics for hospital performance and patient trends.

**7. Multi-Hospital Support**

* Extend the system to handle multiple branches or hospitals under one network.
* Centralized admin panel to monitor performance across locations.

**8. Enhanced Security and Compliance**

* Implement encryption, audit logs, and role-based data access to comply with regulations like **HIPAA** or **GDPR**.
* Improve login security using **Two-Factor Authentication (2FA)** or biometric verification.

**9. Patient Portal**

* Allow patients to log in, view reports, treatment history, make payments, and communicate securely with doctors.
* Improve transparency and engagement between hospitals and patients.

BIBLIOGRAPHY

**For Python and Django**

[1] Django homepage. http://www.djangoproject.com/.

[2] Python documentation. <http://www.python.org/doc>.

[3] Django(web framework). <http://en.wikipedia.org/wiki/Django>.

[4] Django documentation. <http://docs.djangoproject.com>.

[5] Python(programming language). <http://en.wikipedia.org/wiki/Python>.

[6] Books: Web Development with Django by Ben Shaw, Saurabh,

Django 4 By Example by Antonio Mele.