NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY

GREATER NOIDA

(An Autonomous Institute Affiliated to AKTU, Lucknow)

SCHOOL OF COMPUTER APPLICATIONS

"HealthEase – Appointment Booking System"

Mini Project Report submitted by

Ayush Singh, 2401330140100

Saurav Singh, 2401330140336

Ritik Raj, 2401330140283

In partial fulfilment for the award of the degree of

MASTER OF COMPUTER APPLICATIONS

Under the supervision of

Ms. Jishu Varshney

DEC 2025

DECLARATION

I hereby declare that the work presented in this report entitled **"HealthEase – Appointment Booking System"**, was carried out by me. I have not submitted the matter embodied in this report for the award of any other degree or diploma of any other University or Institute. I have given due credit to the original authors/sources for all the words, ideas, diagrams, graphics, computer programs, experiments, and results that are not my original contribution. I have used quotation marks to identify verbatim sentences and given credit to the original authors/sources.

I affirm that no portion of my work is plagiarized, and the experiments and results reported in the report are not manipulated. In the event of a complaint of plagiarism and manipulation of the experiments and results, I shall be fully responsible and answerable.

Name: Ayush Singh
Roll Number: 2401330140100

(Candidate Signature)

CERTIFICATE

Certified that **Ayush Singh** (Enrolment Number: 2401330140100) has carried out the Mini Project in this Report entitled **"HealthEase – Appointment Booking System"** for the award of **Master of Computer Applications** from Dr. APJ Abdul Kalam Technical University, Lucknow under my supervision. The Project Report embodies results of original work, and studies are carried out by the students herself/himself.

(Signature)
Ms. Jishu Varshney
Assistant Professor, MCA
Date:29/12/2025

(Signature of Dean)
Prof. (Dr.) C. S. Yadav
Dean, MCA

(Signature of HOD)
Dr. Sakshi Kumar
HOD, MCA

Date: 29/12/2025

ACKNOWLEDGEMENTS

We would like to express my gratitude towards **Ms. Jishu Varshney** for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

Our thanks and appreciations to respected Dr. Sakshi Kumar, HOD & Ms. Alka Singh Dy. HOD, for their motivation and support throughout.

ABSTRACT

The **HealthEase** Doctor Appointment Booking System is a full-stack web application designed to digitize and streamline clinic and hospital appointment operations. The system provides features such as doctor listing, appointment scheduling, real-time slot availability checking, and administrative management.

The frontend of the system is built using **HTML, CSS, and JavaScript**, offering a fast, interactive, and responsive user experience. The backend is implemented using **Python Flask**, serving as a lightweight and efficient server, while **MySQL** is used as the primary database for secure and reliable data storage.

The project follows a modular architecture where the frontend communicates with backend REST APIs, ensuring scalability and flexibility. Patients can browse doctors by specialization, check real-time appointment availability, and complete bookings instantly. Administrators can manage doctor profiles, configure availability schedules, and view appointment reports. The system aims to reduce manual workload, eliminate scheduling errors, improve operational efficiency, and provide a seamless healthcare management workflow.

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	Full Form UI
	User Interface
UX	User Experience
API	Application Programming Interface
REST	Representational State Transfer **HTTP**
	Hypertext Transfer Protocol
CRUD	Create, Read, Update, Delete
DBMS	Database Management System
RDBMS	Relational Database Management System
SQL	Structured Query Language
ORM	Object-Relational Mapping
JSON	JavaScript Object Notation
JWT	JSON Web Token
CSS	Cascading Style Sheets
HTML	Hypertext Markup Language
SPA	Single Page Application
DOM	Document Object Model
IDE	Integrated Development Environment
PK	Primary Key
FK	Foreign Key
ER	Entity Relationship
DFD	Data Flow Diagram
QoS	Quality of Service
LAN	Local Area Network
OS	Operating System
CPU	Central Processing Unit
RAM	Random Access Memory
URL	Uniform Resource Locator
UPI	Unified Payments Interface

CHAPTER 1

INTRODUCTION:

HealthEase is a digital healthcare solution designed to provide hospital-like facilities at home, offering a convenient and efficient way for patients to access healthcare services without the need to visit a hospital or clinic. The system connects patients with a wide range of healthcare providers, including doctors, diagnostic service providers, physiotherapists, and other medical professionals, ensuring comprehensive care while maintaining the comfort, safety, and convenience of home treatment.

HealthEase leverages modern web technologies such as HTML, CSS, and JavaScript for the frontend, creating an interactive and responsive user interface. The backend is developed using Python (Flask), providing efficient server-side processing, secure API management, and seamless communication between the frontend and database. MySQL is used for reliable and structured data storage, managing patient records, appointment schedules, medical reports, and service provider details in a secure and scalable manner.

Health Ease is a comprehensive web-based **Doctor Appointment Booking System** developed to bridge the gap between patients and healthcare providers. This system enables patients to browse doctor profiles, view consultation fees, check real-time availability, and book appointments through an interactive online interface. On the other hand, administrators can manage doctor details, monitor appointment logs, and oversee system operations through a dedicated dashboard.

By integrating a **React-based frontend** with a **Python Flask backend** and **MySQL database**, the system ensures smooth communication, secure data management, and a user- friendly experience. The project follows a structured architecture where the frontend and backend communicate via REST APIs, ensuring scalability and reliability

1.1 Purpose of the Project:

The primary purpose of the **Health Ease** project is to:

- **Digitalize Healthcare Operations:** Transition from manual appointment booking to a digital platform for better speed and accuracy.
- **Reduce Waiting Times:** Eliminate the need for patients to stand in long physical queues or wait on phone lines to book appointments.

- **Centralized Information:** Provide a single platform where patients can view doctor specializations, experience, and fees.
- **Streamline Management:** Help clinic administrators manage doctors, patient records, and schedules in a structured way.
- **Ensure Data Security:** Maintain secure records of users and appointments using a reliable MySQL database.

1.2 Problems in the Existing System

The existing manual or semi-digital systems used in clinics and hospitals face several issues:

- **Long Queues & Delays:** Patients often must visit the clinic physically just to get an appointment token, leading to overcrowding.
- **Lack of Information:** Patients often lack information about doctor availability, consultation fees, or schedule changes until they reach the clinic.
- **Human Errors:** Manual record-keeping on paper or basic spreadsheets often leads to double bookings or lost appointment data.
- **Inconvenient Scheduling:** Booking appointments requires calling during working hours, restricting accessibility for patients.
- **No Centralized History:** It is difficult for doctors to track a patient's appointment history manually.

1.3 Solution to these Problems

The proposed **Health Ease** system resolves these issues through:

- **Online Appointment Booking: Allows patients to book appointments 24/7 from the comfort of their homes, eliminating physical queues.**
- **Real-Time Availability: Displays live slot availability to prevent double bookings and ensure accurate scheduling.**
- **Doctor Management: Admins can easily add or update doctor profiles, specializations, and images.**
- **Automated Workflow: The system automatically handles user registration, authentication, and appointment status updates (e.g., upcoming, completed).**

CHAPTER 2

2 FEASIBILITY STUDY:

The feasibility study is a critical phase of the project life cycle that evaluates whether the proposed **Health Ease** system can be developed effectively using available resources, technology, and time constraints. It analyses various factors to ensure that the solution is practical, viable, and beneficial for clinics, doctors, and patients. This chapter focuses on the Technical and Operational feasibility of the system.

2.1 Technical Feasibility:

Technical feasibility assesses whether the current technical resources and technologies are adequate for the development of the system and if they can support the project requirements.

2.1.1 Availability of Technology

The **Health Ease** system is built using standard, widely available, and open-source technologies:

- **Frontend: HTML5, CSS3, and JavaScript (Vanilla JS) for building the user interface**
- **Backend: Python Flask (a lightweight web framework) for handling API logic and server-side operations.**
- **Database: MySQL (Relational Database Management System) for secure data storage.**
- **Tools: Visual Studio Code (IDE), Postman, and Git. These free and open-source tools ensure the project is technically feasible and cost-effective.**

All these technologies are free, well-documented, and supported across multiple platforms.

2.1.2 Suitability of Technology

The selected technology stack is highly suitable for this doctor appointment system:

- **HTML, CSS, & JavaScript:** Standard web technologies ensuring a lightweight, fast, and cross-browser compatible interface.
- **Python Flask:** A lightweight and efficient framework ideal for building secure REST APIs and managing backend logic.
- **MySQL:** A robust database system that ensures data integrity and effectively manages complex relationships between patients and doctors.

2.1.3 Developer Skill Requirements

- **Frontend Development:** Proficiency in HTML, CSS, and JavaScript for UI design and API integration.

- Backend **Development:** Basic understanding of Python and the Flask framework structure.

- Database **Management:** Ability to write SQL queries and manage connections using SQL Alchemy.

- **Conclusion:** Since these are standard curriculum skills, the project is feasible for the development team.

2.1.4 System Performance Capability

The system is designed for high efficiency and speed:

- **Smooth UI:** JavaScript updates content dynamically without page reloads.
- **Rapid Processing:** Flask's lightweight structure ensures fast API responses.
- **Quick Search:** Optimized MySQL queries deliver instant doctor availability results.
- **Concurrency:** Effectively handles multiple simultaneous booking requests.
- **Efficiency:** The lightweight application runs smoothly on minimal hardware.

2.2 Operational Feasibility

This study determines if the system can be smoothly integrated into the daily operations of clinics and hospitals.

2.2.1 Ease of Use

- **Simple Interface:** Built with standard HTML/CSS, providing a clean and distraction- free layout.
- **Minimal Steps:** The booking process is streamlined to just three steps: **Login → Select Slot → Confirm**.
- **No Training Needed:** The UI is self-explanatory, requiring no specialized training for patients or receptionists.
- **Responsive Design:** The layout adjusts to different screen sizes, making it usable on laptops and tablets.

2.2.2 User Acceptance

The system addresses critical pain points for both patients and healthcare providers, ensuring high adoption rates:

- **24/7 Accessibility:** Patients can book appointments anytime, reducing dependency on clinic opening hours.
- **Queue Elimination:** Removes the need for patients to stand in long physical lines for tokens.
- **Staff Efficiency:** Reduces the manual burden on receptionists, allowing them to focus on patient care.
- **Transparency:** Patients can view doctor fees, experience, and available slots upfront before booking.

2.2.3 Operational Workflow Improvements

Implementing this system transforms the clinic's operations from manual processing to digital automation:

- **Paperless Operations:** Replaces physical appointment registers and manual slips with digital records.

- **Conflict Prevention:** Automated logic prevents double-booking of the same time slot, eliminating scheduling errors.
- **Centralized Data:** Patient history and doctor schedules are stored in a single, secure MySQL database.
- **Real-Time Updates:** Slot availability is updated instantly, ensuring accurate information for all users.
- **Quick Retrieval:** Admins can search for patient details or doctor schedules in milliseconds.

2.2.4 Maintenance and Scalability

The system is built on a robust architecture that supports long-term use and future growth:

- **Modular Design:** Separating Frontend and Backend simplifies debugging and updates.
- **Easy Maintenance:** Uses standard languages (Python, SQL) for effortless code management.
- **Future Ready:** New features like Online Payments can be added easily.

CHAPTER 3

SYSTEM ANALYSIS:

System analysis is a crucial phase that involves studying the existing workflow, identifying its limitations, and defining how the proposed solution will address these issues to improve efficiency. It evaluates data flow, user interactions, and the overall process of the system.

3.1 Study of the System:

Current healthcare systems rely on manual registers or basic spreadsheets, requiring patients to visit clinics physically or call receptionists to book appointments.

Limitations of the Existing System:

- **Physical Presence:** Patients must travel to the clinic just to book a slot.
- **Manual Errors:** High risk of double-booking and incorrect data entry.
- **Time-Consuming:** Retrieving patient history from physical files is slow and inefficient
- **Communication Gaps:** Patients are often unaware of doctor unavailability until they arrive.
- **No Real-Time Status:** Patients cannot check live seat availability remotely.

3.2 Proposed System

The proposed **Appointment Booking System** solves the limitations of the manual process by providing a fully automated platform built using **HTML/CSS/JavaScript (frontend)**, **Flask with JWT Authentication (backend)**, and **SQLite/MySQL (database depending on your project)**.

The system provides:

For Users (Patients/Customers)

- View available doctors/time slots in real time
- Check availability instantly before booking
- Book appointments through an online interface
- Receive confirmation details immediately on the screen

For Administrators (Reception/Staff)

- Add, update, or remove doctor schedules
- Manage available appointment slots
- Track and review all bookings from a dashboard
- Access well-organized data for better decision-making

Using **REST APIs**, the frontend communicates smoothly with the backend for login, slot retrieval, and booking operations. All data is stored in a centralized database, ensuring **accuracy**, **consistency, and secure access**.

3.3 Input and Output

Input

- **User login details** (email, password) submitted through the login form
- **Admin login details** (admin email, password) for accessing the admin dashboard
- **Appointment-related inputs** including doctor selection, slot selection, and confirmation
- **Backend inputs received from forms and sent via REST API calls (JSON requests)**

Output

- **Dashboard or home page display** after successful login
- **Available time slots or schedule information** shown to the user
- **Booking confirmation message** with appointment details
- **Admin panels** show user lists, booking data, and schedules
- **Error messages or alerts** (e.g., invalid login, slot unavailable, incorrect input)

User inputs are captured through the **HTML/JavaScript frontend**, sent to the backend via **API requests**, processed by the server, and returned as **JSON responses**. These responses are then rendered instantly on the frontend for smooth and interactive user experience.

3.4 Process Models Used with Justifications

Process Model Used: Waterfall Model (Recommended for Academic Projects)
The system development follows the **Waterfall Model** a linear and structured software development approach widely used in academic and small-to-medium scale projects.

Justification:

- **Requirements were clearly defined** (login, admin access, appointment booking, API integration) and did not require frequent changes.
- E Each phase—**Analysis, Design, Development, Testing, Deployment**—could be completed independently and sequentially.
- The model is **simple to plan, execute, and document**, making it suitable for academic submissionist.
- Ensures the project can be completed **within the allotted timeframe** with clear milestones.

Waterfall Phases Applied:

1. **Requirement Gathering:** Identified key features: user login, admin login, slot booking, token handling, and API communication.
2. **System Design:** Designed UI structure, API endpoints, workflow diagrams, DFDs, and database schema (users, appointments, admins).
3. **Implementation**: Coding the Frontend (HTML/JS) and Backend (Flask).
4. **Testing**: Verifying login, booking logic, and database connections.
5. **Deployment**: Running the application on a local server.

CHAPTER 4

SOFTWARE REQUIREMENT SPECIFICATIONS:

This chapter details the functional and non-functional requirements necessary for the successful development and deployment of the Health Ease system.

4.1 Functional Requirements:

Functional requirements describe what the system must do. These requirements define the core operations of both patient and admin modules.

4.1.1 User Module

- Patients must be able to register and log in securely.
- Patients can browse the list of available doctors and their specializations.
- Patients can view consultation fees and experience details for a selected doctor.
- Patients can check real-time appointment slot availability.
- Patients can select a date and time slot to book an appointment.
- Patients can view booking confirmation and appointment history.

4.1.2 Admin Module

- Admins must be able to log in securely to the dashboard.
- Admins can add, update, or delete doctor profiles.
- Admins can schedule availability by assigning time slots to doctors.
- Admins can manage doctor details, images, and specializations.
- Admins can view daily, weekly, or monthly appointment booking reports

4.1.3 Backend/API Requirements

- The backend must expose REST APIs for authentication, doctors, appointments, and reports.
- API responses must follow JSON format.

- The system must validate all user inputs before processing.

4.1.4 Database Requirements

- Store user information securely (hashed passwords).
- Maintain records for doctors, patients, and appointments.
- Ensure relational integrity with foreign keys.
- Maintain timestamps for all booking activities.

4.2 Performance Requirements

Performance requirements specify how efficiently the system should operate under various conditions.

- The frontend pages should load within **2 seconds** on standard devices to ensure a smooth user experience.
- API responses from the Flask backend should return results within **300–800 milliseconds**
- The system should support multiple patients accessing the booking service simultaneously without crashing.
- Appointment booking operations must handle concurrency effectively to prevent double-booking of the same doctor's time slot.
- Database transactions must ensure atomicity during the appointment allocation process.
- The system should operate smoothly on a standard hosting environment.

4.3 Hardware Requirements

4.3.1 Minimum Hardware for Development

- Processor: Dual-Core or above
- RAM: 4 GB or higher
- Storage: 250 GB HDD/SSD

- Monitor: 720p or higher resolution

4.3.2 Minimum Hardware for Deployment

- Processor: Quad-Core or above
- RAM: 8 GB or higher
- Storage: 500 GB HDD/SSD
- Stable internet connection (for frontend-backend communication)

4.4 Software Requirements

4.4.1 Frontend Software

- **HTML5** (Structure).
- **CSS3** (Styling and Layout)
- **JavaScript** (Client-side logic and API integration).
- **Web Browser:** Google Chrome or Mozilla Firefox.

4.4.2 Backend Software

- **Python** (Programming Language).
- **Flask** (Web Framework)
- **Python Libraries:** Flask-SQLAlchemy, MySQL, Flask-CORS.
- Postman (API testing tool)

4.4.3 Database Software

- **MySQL Server** (Relational Database Management System).
- **Optional:** MySQL Workbench (GUI for database management)

4.4.4 Development Tools

- Visual Studio Code (frontend)
- Git and GitHub (version control)
- Web browser (Chrome/Firefox)

CHAPTER 5

SYSTEM DESIGN:

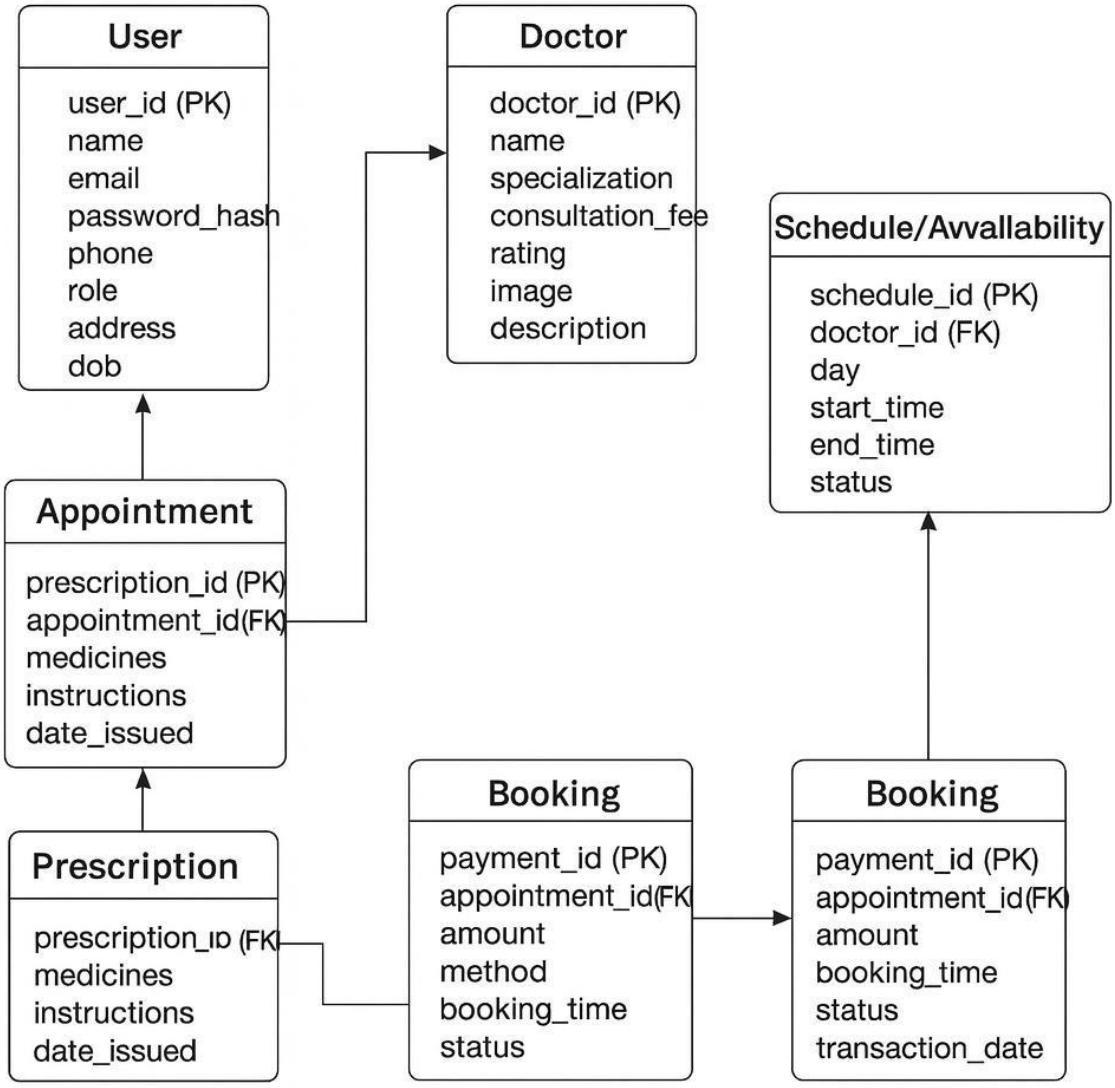
System Design defines the overall architecture, structure, data flow, and interactions of the HealthEase system. It translates requirements into technical blueprints that guide development, ensuring the system is modular, scalable, and easy to maintain.

5.1 Introduction

The **HealthEase** system follows a structured **three-tier architecture** consisting of:

1. **Frontend Layer – HTML, CSS, JavaScript:**
    - Handles UI rendering and user interaction.
    - Sends API requests to the backend using the Fetch API.
    - Displays doctor lists, appointment forms, and user dashboards.
2. **Backend Layer – Python Flask (REST APIs):**
    - Processes business logic and handles authentication.
    - Manages doctor schedules, appointment bookings, and admin operations.
    - Communicate with the MySQL database to store and retrieve data.
3. **Database Layer** – MySQL:
    - Stores user profiles (Patients/Admins) securely.
    - Maintains records for doctors, appointments, prescriptions, and payments.
    - Ensures relational integrity and data consistency

5.2 E-R Diagram



5.2.1 Entities and Their Relationships

5.2.1.1 USER

Represents the individuals who use the system, including patients seeking consultation and administrators managing the platform.

Attributes:

- user_id (PK)
- name
- email
- password_hash
- phone
- role
- address
- dob

Relationships:

- One user → many bookings (1:N)

5.2.1.2 Doctor

Represents the medical professionals available for consultation.

Attributes:

- doctor_id (PK)
- Name
- specialization
- consultation_fee
- rating
- image
- description

Relationships:

- **One** Doctor can have **Many** Appointments (1:N).
- **One** Doctor can have **Many** Schedule slots (1:N).

5.2.1.3 Appointment

Represents the booking transaction between a patient and a doctor.

Attributes:

- appointment_id (PK)
- patient_id (FK)
- doctor_id (FK)
- date
- time
- symptoms
- status

Relationships:

- **One** Appointment is linked to **One** Prescription (1:1)
- **One** Appointment is linked to **One** Payment (1:1)

5.2.1.4 Prescription

Represents the medical advice and medicines prescribed by the doctor after an appointment.

Attributes:

- prescription_id (PK)
- appointment_id (FK)
- medicines
- instructions
- date_issued

Relationships:

- Linked directly to a completed Appointment.

5.2.1.5 Schedule/Availability

Represents the time slots available for a doctor.

Attributes:

- schedule_id (PK)

- doctor_id (FK)
- day
- start_time
- end_time
- status

Relationships:

- Determines the valid inputs for creating an Appointment.

5.2.1.6 BOOKING

Represents the transaction details for the consultation fee.

Attributes:

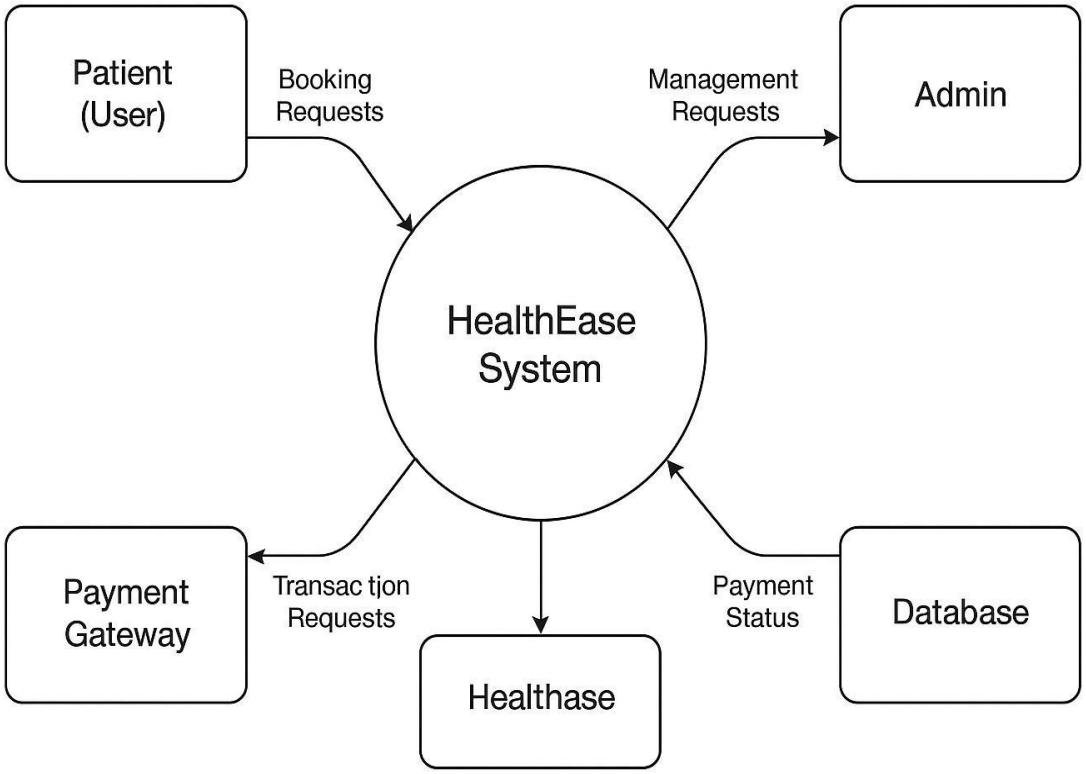
- payment_id (PK)
- appointment_id (FK)
- amount
- method
- booking_time
- status
- transaction_date

Relationships:

- **One** Payment is associated with **One** Appointment (1:1).

5.3 Data Flow Diagrams (DFD)

5.3.1 DFD Level 0



TDFD Level 0, also known as the Context Diagram, depicts the interaction between the system and external entities:

External Entities

- **Patient (User):** Sends booking requests, views doctor details, and receives appointment confirmations.
- **Admin:** Sends management requests (adding doctors/schedules) and receives system reports.
- **Payment Gateway:** Processes transaction requests and returns payment status
- **Database** → stores system data

Main Process

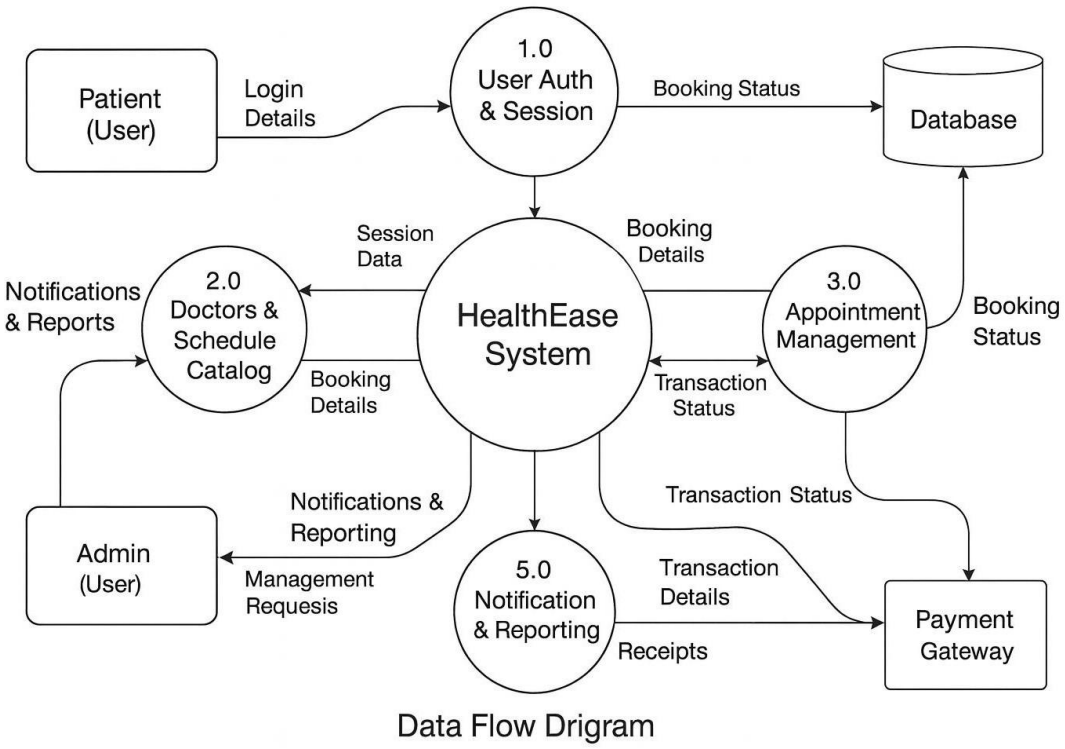
- **HealthEase System:** The central processing unit that handles all inputs and outputs.

Data Flow Summary

- Patients submit booking details → System → Database.
- System sends appointment receipts → Patients.

- Admin updates doctor profiles/schedules → System.
- Payment Gateway confirms transactions → System.

6.1.1 DFD Level 1



Data Flow Drigram

This diagram breaks down the main system into detailed modules:

1.0 User Auth & Session: Handles login, registration, and session storage for patients and admins.

2.0 Doctor & Schedule Catalog: Fetches doctor details and available slots from the database for display.

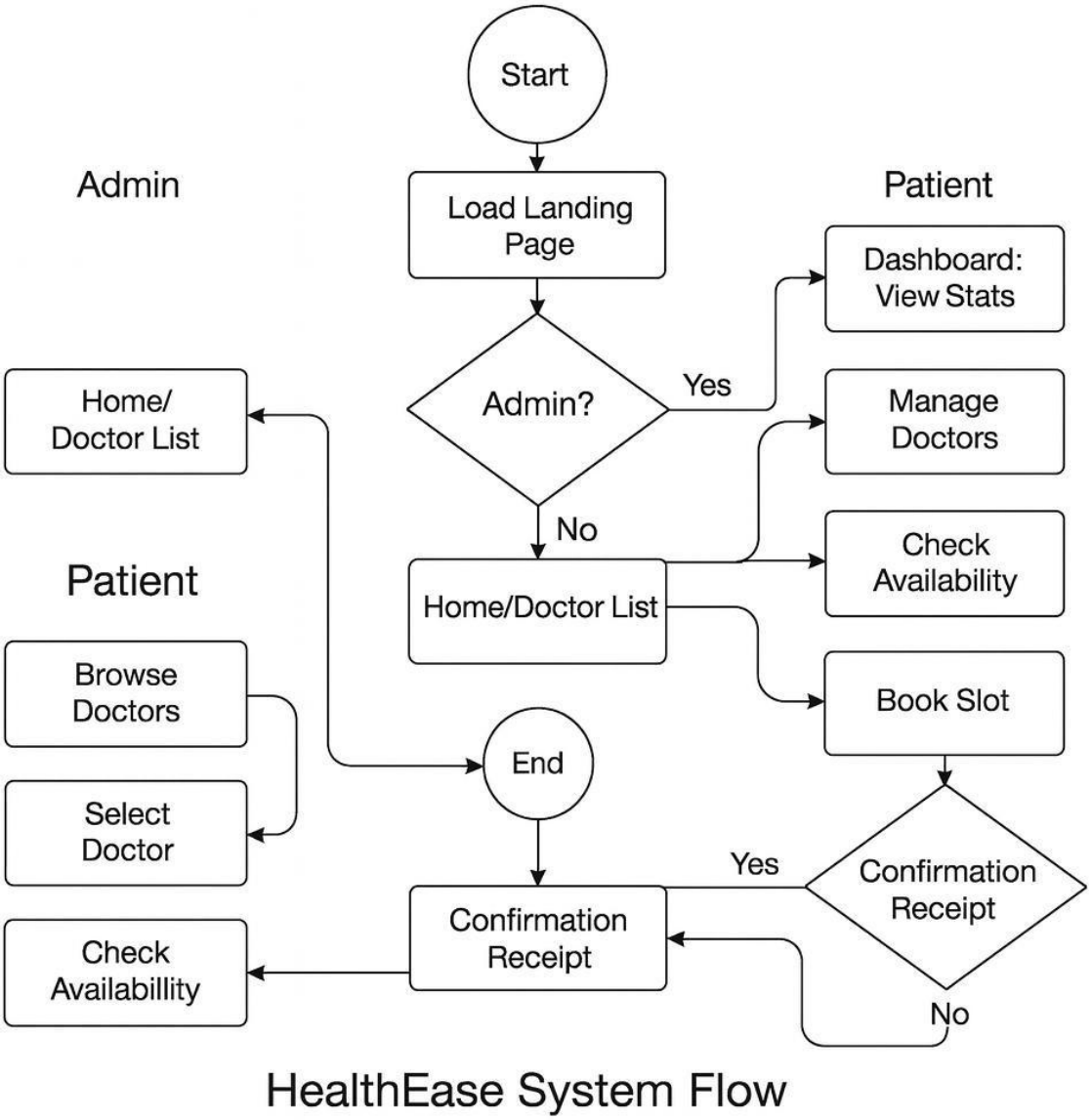
3.0 Appointment Management: Checks slot availability, reserves dates, and updates booking status.

4.0 Payment Processing: interacts with the payment gateway to finalize the booking fee.

5.0 Notification & Reporting: Generates appointment receipts for patients and logs data for admin reports**.**

6.0 Admin Tools: Allows administrators to manage doctors, view bookings, and update schedules.

6.2 Flowchart (System Flow)



The flowchart illustrates the sequential workflow of the HealthEase system for both patients and administrators.

The main flow of the system is as follows:

System Flow Description:

1. **Start:** The application loads the landing page.
2. **Login:** Users enter credentials. The system verifies role (Admin vs. Patient).
    - **If Admin:** Redirects to Admin Dashboard.
    - **If Patient:** Redirects to Home/Doctor List.
3. **Patient Flow:**
    - **Browse Doctors:** Patient views list of doctors.
    - **Select Doctor:** Patient views profile and fees.
    - **Check Availability:** Patient selects a date and views available time slots.
    - **Book Slot:** Patient selects a slot and confirms details.
    - **Payment**: System processes the consultation fee.
    - **Confirmation:** If payment is successful, the appointment is booked, and a receipt is shown.
4. **Admin Flow:**
    - **Dashboard:** View stats (Total Doctors, Active Bookings).
    - **Manage Doctors:** Add or update doctor profiles.
    - **Manage Schedules:** Assign shifts and time slots to doctors.
    - **View Reports:** Check daily appointments and revenue.
5. **End:** User logs out.

6.2.1 Users

Stores information for both patients and administrators.

Field	Type	Description
user_id	INT (PK)	Unique user identifier
name	VARCHAR(100)	Full Name
email	VARCHAR(120)	Unique Login Email

password_hash	VARCHAR(255)	Encrypted Password
role	VARCHAR(20)	'patient' or 'admin'
phone	VARCHAR(20)	Contact Number
address	TEXT	Residential Address

6.2.2 Doctors

Stores details of medical professionals.

Field	Type	Description
doctor_id	INT (PK)	Unique Doctor ID
name	VARCHAR(100)	Doctor's Name
specialization	VARCHAR(100)	Field (e.g., Cardiologist)
experience	INT	Years of practice
consultation_fee	INT	Fee per visit
rating	FLOAT	Average rating
image	VARCHAR(300)	Profile Image URL

6.2.3 Appointments

Field	Type	Description
id	INT (PK)	Unique Appointment ID
patient_id	INT (FK)	Links to User
doctor_id	INT (FK)	Links to Doctor
date	VARCHAR(20)	Appointment Date

time	VARCHAR(20)	Selected Time Slot
symptoms	VARCHAR(200)	Patient's description
status	VARCHAR(20)	'upcoming', 'completed'

6.2.4 Prescriptions

Field	Type	Description
id	INT (PK)	Unique Prescription ID
appointment_id	INT (FK)	Linked Appointment
medicines	TEXT	List of medicines
instructions	TEXT	Usage instructions
date_issued	DATE	Date of prescription

6.2.5 Schedules

Field	Type	Description
schedule_id	INT (PK)	Unique Schedule ID
doctor_id	INT (FK)	Links to Doctor
day	VARCHAR(20)	Day of week/Date
start_time	TIME	Shift start
end_time	TIME	Shift end
status	VARCHAR(20)	'Available' / 'Full'

6.2.6 Payments

Field	Type	Description
booking_id	SERIAL PK	Booking ID
user_id	INT FK	User
show_id	INT FK	Show
total_amount	NUMERIC(10,2)	Amount
booking_time	TIMESTAMP	Time
status	VARCHAR(20)	confirmed / cancelled
payment_id	INT FK	Payment ref
ticket_code	VARCHAR(50)	QR / ticket number

6.2.7 Booking_Seat

Field	Type	Description
booking_id	INT FK	Booking
seat_id	INT FK	Seat
price	NUMERIC(8,2)	Price per seat

6.2.8 Payment

Field	Type	Description
payment_id	**INT (PK)**	**Unique Payment ID**

Field	Type	Description
appointment_id	**INT (FK)**	**Linked Appointment**
amount	**DECIMAL**	**Fee Amount**

method	**VARCHAR(20)**	**Card / UPI / Cash**
status	**VARCHAR(20)**	**'Success' / 'Failed'**
transaction_date	**TIMESTAMP**	**Time of payment**

CHAPTER 6

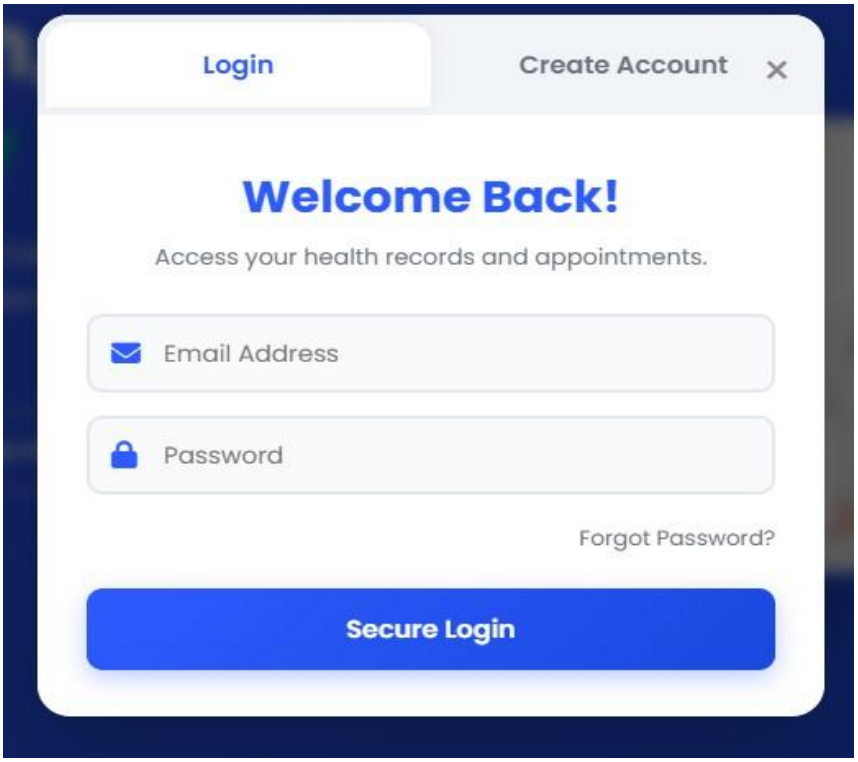
OUTPUT SCREENS:

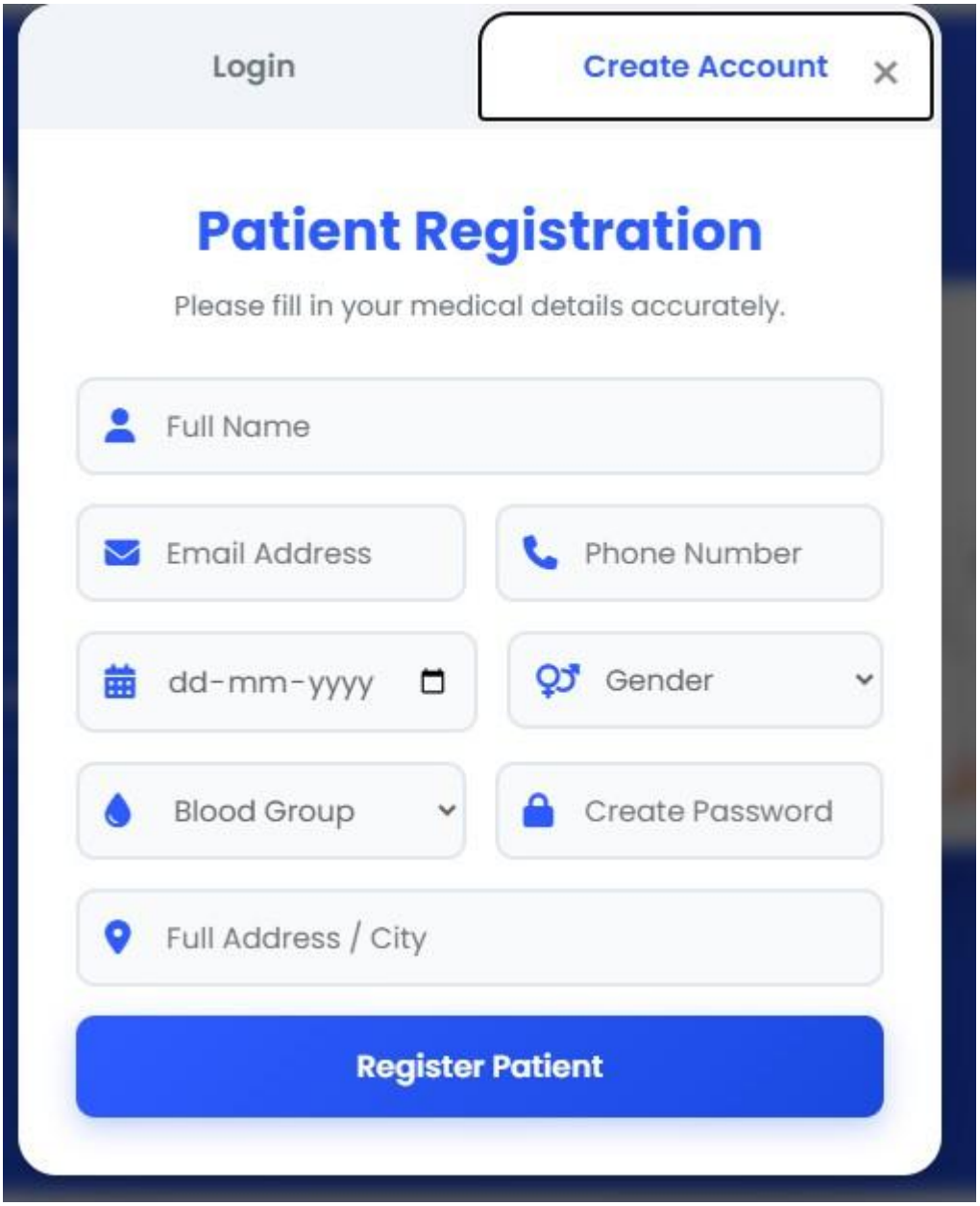
6.1 User Login / Registration Page

Description:

This is the entry point for the application. The system provides a secure interface for users to authenticate themselves.

Screenshot:



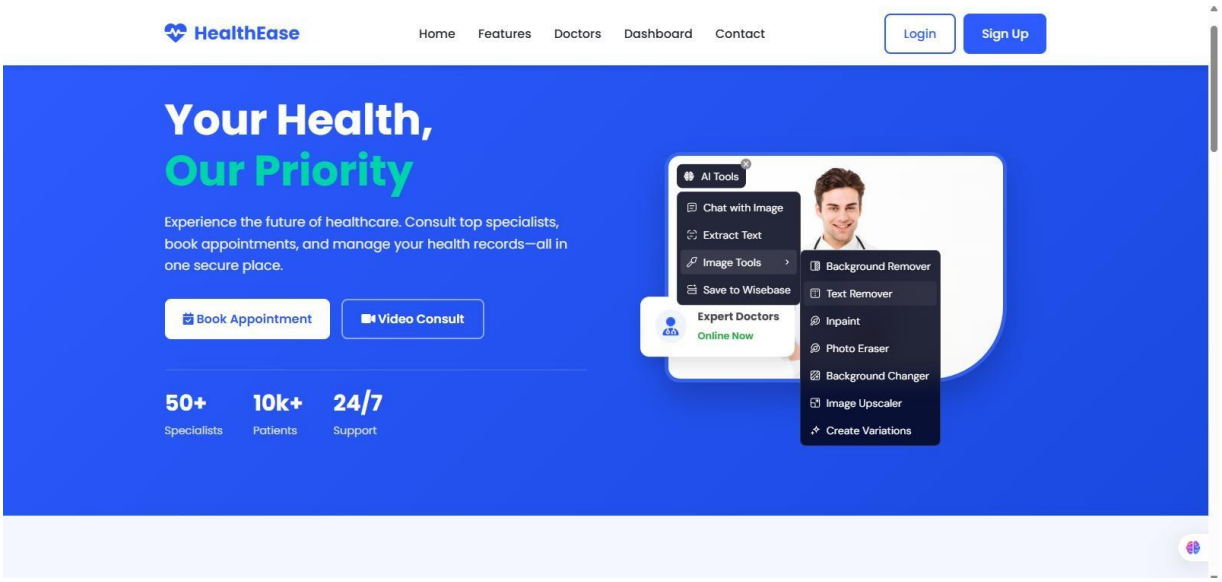


6.2 Home Page

Description:

The Home Page serves as the landing area, featuring a Hero section that highlights key services. The **Doctor List** section (displayed in the "Doctors" segment) dynamically fetches and displays doctor profiles from the database. Each Doctor Card includes.

Screenshot:



6.3 Appointment Booking Page / Confirmation

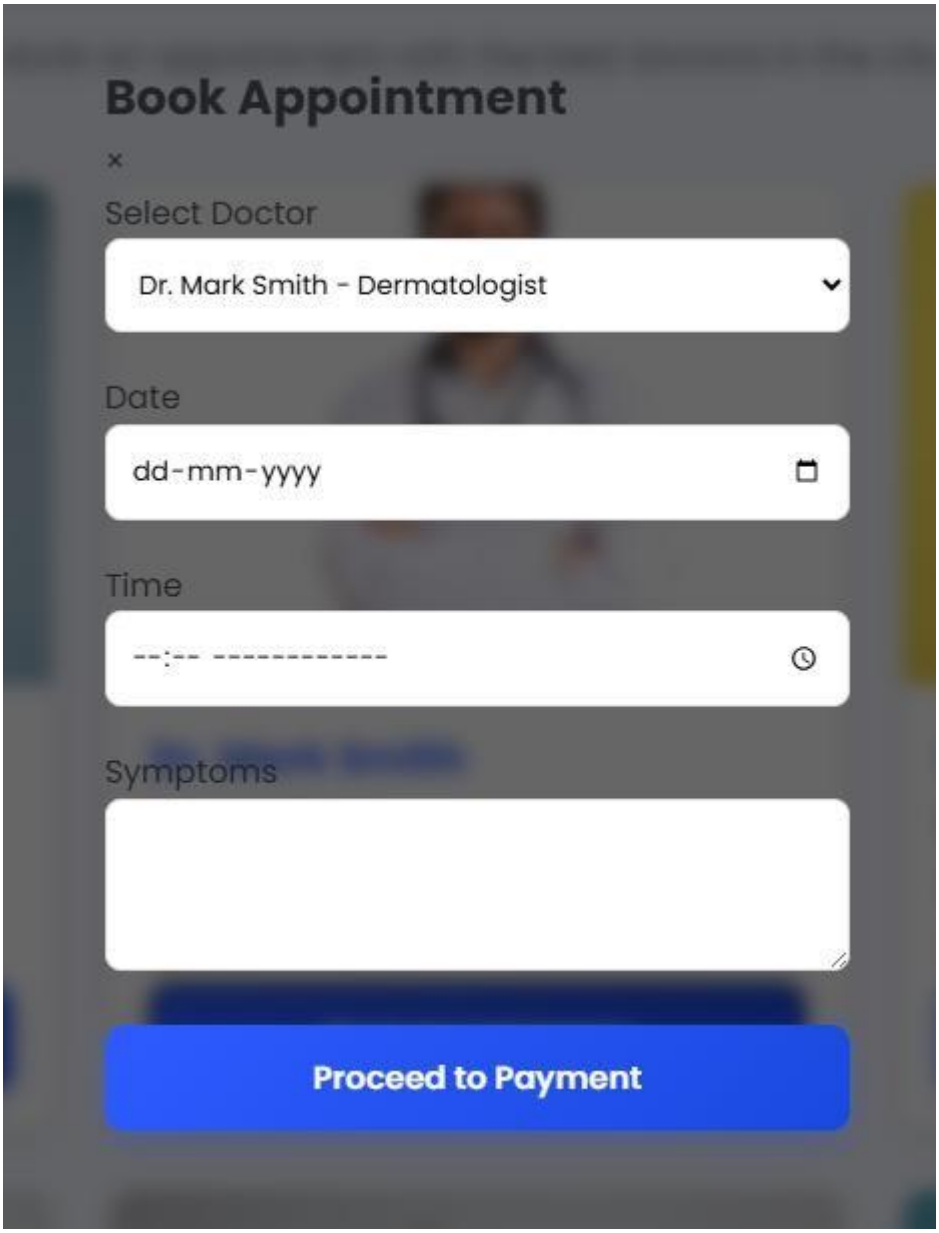
Description:

When a patient clicks "Book Now," the **Booking Modal** appears. This interface ensures a seamless booking experience.

- **Doctor Selection:** Automatically selects the chosen doctor (or allows selection from a dropdown).
- **Date & Time:** Patients use date and time pickers to schedule their visit.
- **Symptoms:** A text area allows patients to describe their symptoms beforehand.

Upon clicking "Book Appointment," the data is sent to the backend, and a success notification confirms the booking.

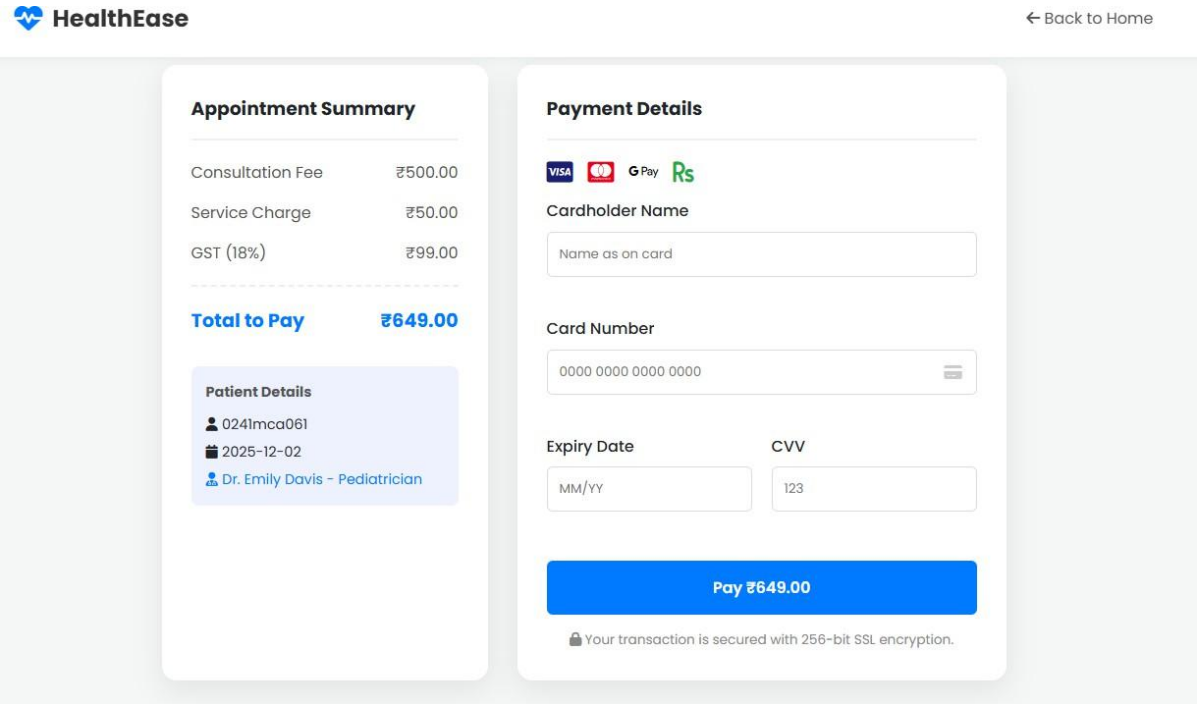
Screenshot:



6.4 Payment Page

Description:

(Note: As per the current code, payment is simulated/integrated into the booking flow). This screen represents the final step of the booking transaction. It displays the **Consultation Fee** associated with the selected doctor. In a real-world scenario, this would connect to a gateway; for this project, it confirms the financial transaction logic and finalizes the appointment status as "Confirmed" in the database.

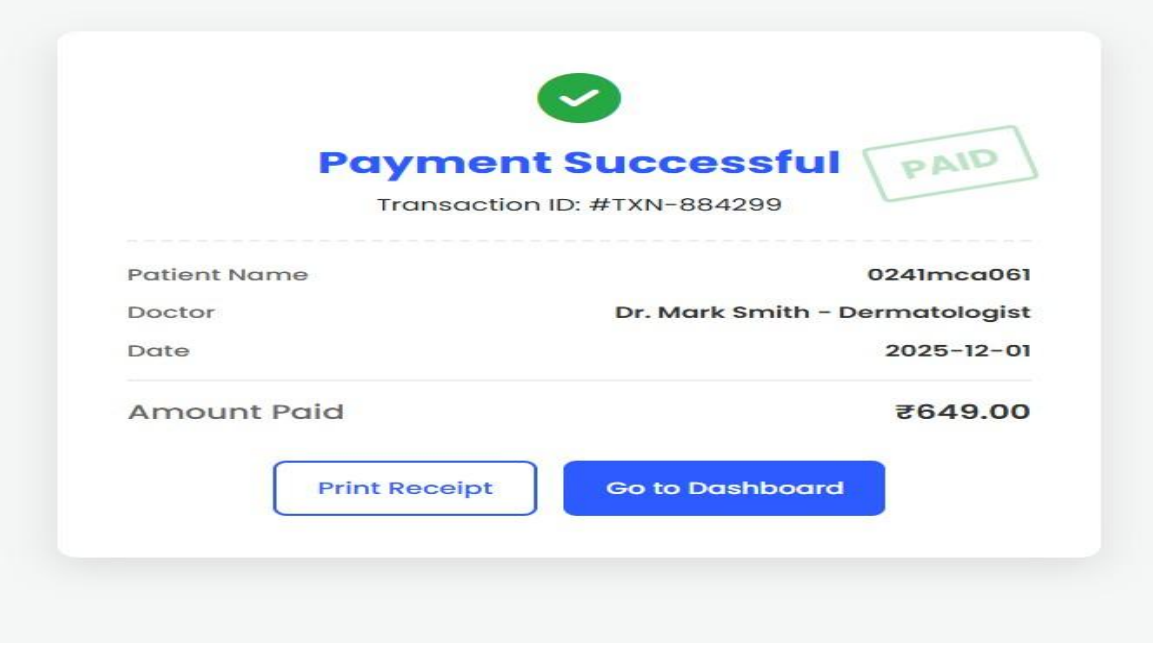


6.5 Prescription / Appointment Receipt

Description:

This screen serves as the digital receipt and official confirmation for the patient after a successful transaction. Once the payment is processed, the system generates this summary to verify the booking.

Screenshot:

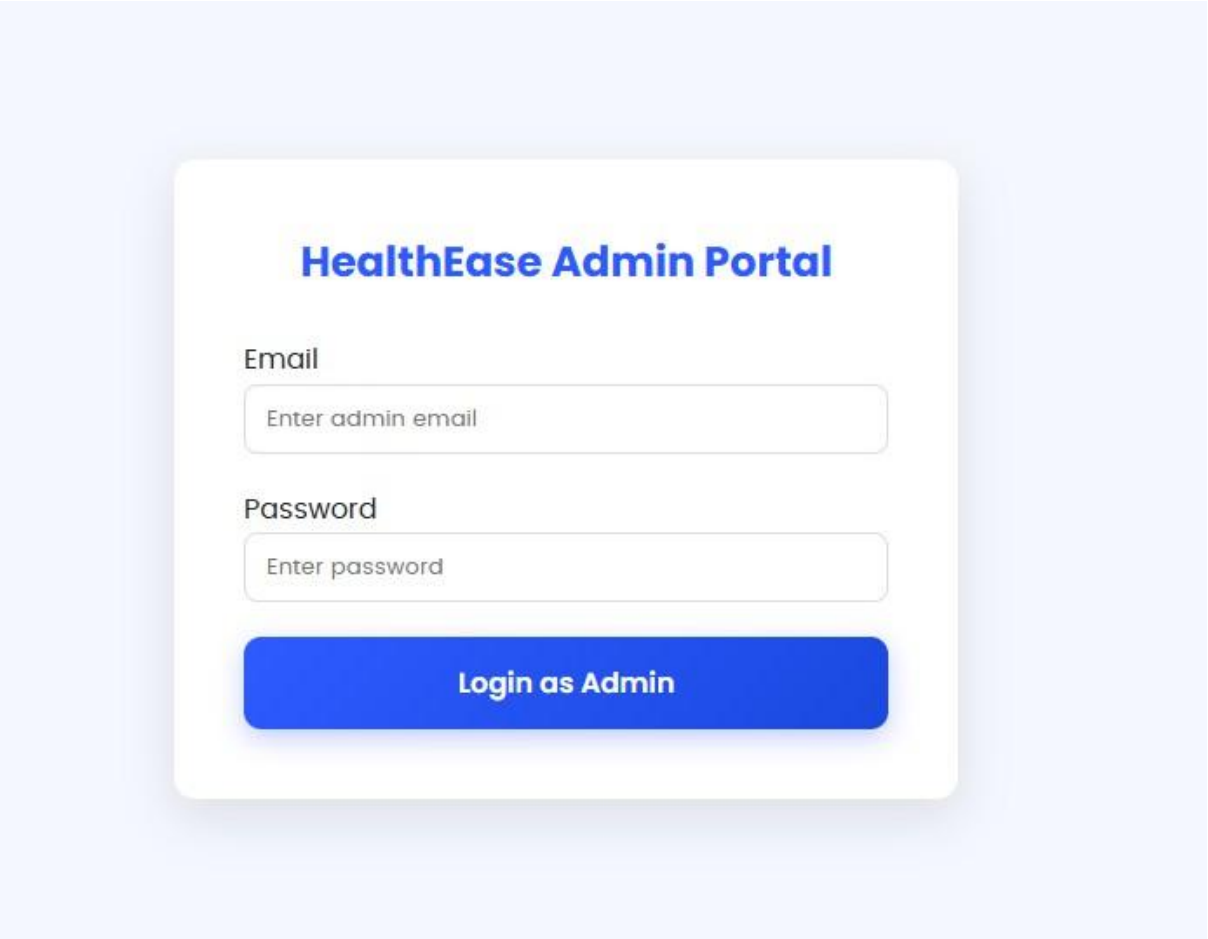


6.6 Admin Login Page

Description:

This interface is restricted to authorized personnel only. Administrators log in using specific credentials (e.g., admin@healthease.com). The backend verifies the user role as 'Admin' before granting access to sensitive management tools, ensuring the security of the system's data.

Screenshot:

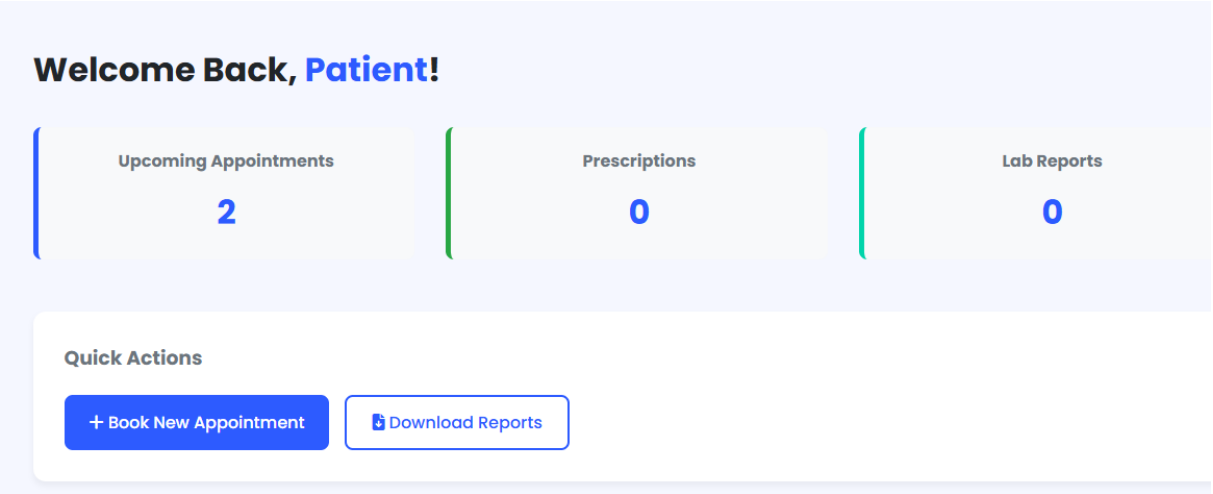


6.7 Dashboard

Description:

Upon logging in, the patient is directed to their personalized Dashboard. This screen serves as the central hub for managing their healthcare journey, offering a quick overview of their medical status and easy access to key actions.

Screenshot:

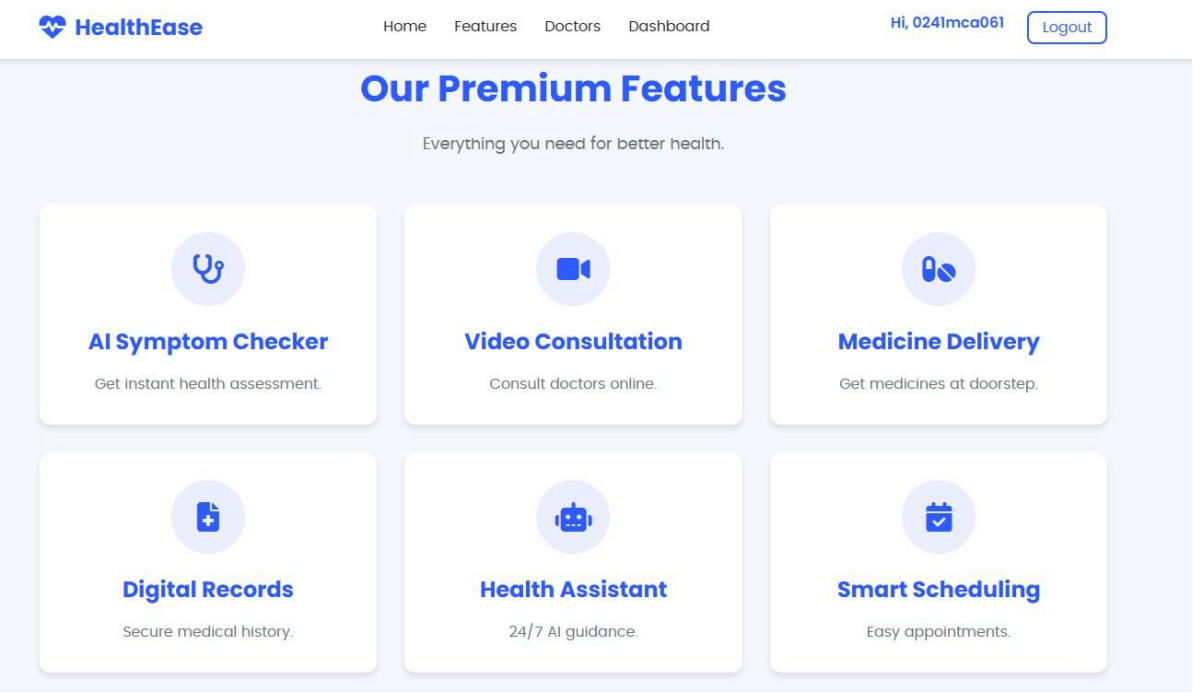


6.8 Meet Our Specialists

Description:

Upon successfully logging in, the patient is redirected to the Home Page, which features the **"Meet Our Specialists"** section. This interface acts as the central hub for finding and selecting healthcare providers.

Screenshot:



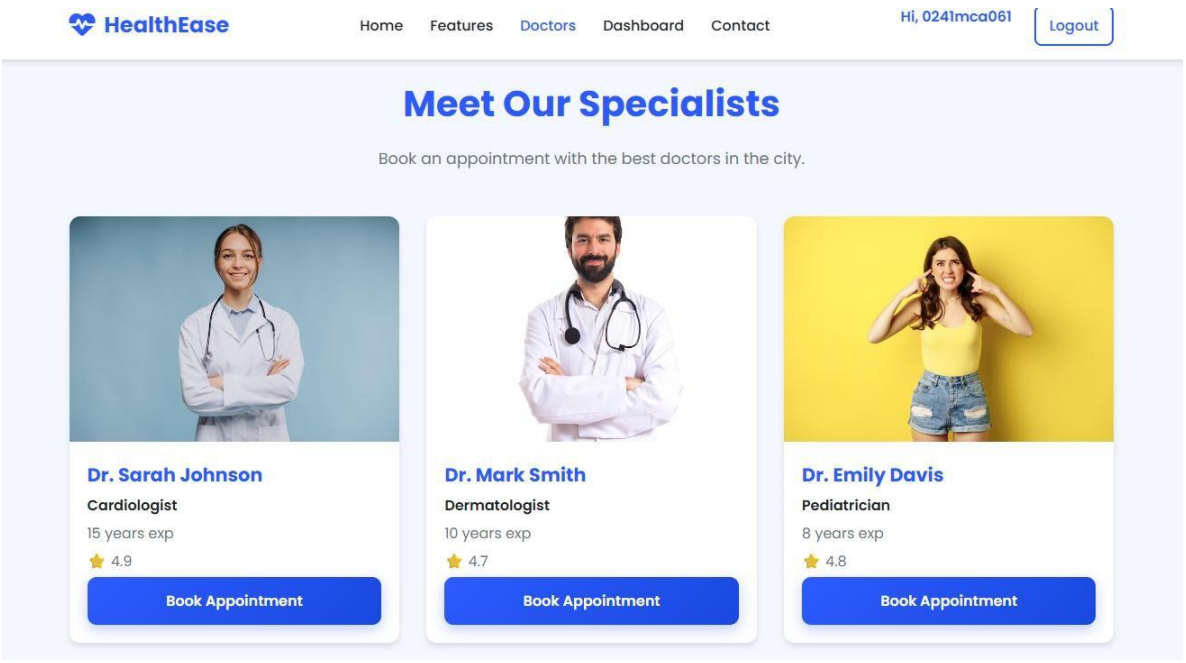
6.9 Doctor List

Description:

Upon successfully logging in, the patient is redirected to the Home Page, which features the

"Meet Our Specialists" section. This interface acts as the central hub for finding and selecting healthcare providers.

Screenshot:



CHAPTER 7

SYSTEM TESTING AND IMPLEMENTATION:

System testing is a critical phase of software development to ensure that the HealthEase application meets functional, performance, and security requirements before deployment.

7.1 Introduction to Testing:

Testing involves evaluating the system to identify errors, ensure correct functionality, and verify that user requirements are met. The purpose of testing in the HealthEase project is to confirm that:

The purpose of testing in this project is to confirm that:

- Patient registration and login work securely.
- Doctor searches return accurate results.
- Appointment bookings are processed without conflicts.
- Admin operations (adding doctors/schedules) execute successfully.
- API endpoints respond with valid JSON data.

Testing was conducted at multiple levels to thoroughly validate the system.

7.2 Types of Testing Performed

We conducted multiple levels of testing to thoroughly validate the system.

7.2.1 Unit Testing

We tested individual components in isolation to ensure they function correctly.

- **Backend Logic:** Verified Python functions like register(), login(), and book_appointment().
- **Database Models:** Tested if User, Doctor, and Appointment tables store data correctly.
- **API Responses:** Checked if API endpoints return the correct HTTP status codes (200 OK, 401 Unauthorized).

7.2.2 Integration Testing

This phase ensured proper communication between different modules:

- **Frontend-Backend:** Verified that the HTML forms correctly send data to Flask APIs using JavaScript fetch().
- **Backend-Database:** Confirmed that the Flask app successfully connects to MySQL and retrieves doctor records.
- **Authentication:** Tested if the JWT token generated by the backend allows access to protected routes.

Tools used: Postman, JMeter (partial load testing).

7.2.3 Functional Testing

We validated complete user flows to ensure the system meets business requirements:

- **Booking Flow:** A patient logs in, selects a doctor, books a slot, and receives a confirmation.
- **Admin Flow:** An admin logs in, adds a new doctor, and the doctor immediately appears on the patient's dashboard.
- **Validation:** Tested that the system rejects invalid emails or empty passwords.

7.2.4 UI/UX Testing

Since multiple patients might try to book appointments simultaneously, we tested:

- **Slot Locking:** Ensuring that if Patient A books a 10:00 AM slot, Patient B cannot book the same slot for the same doctor.
- **Database Stability:** Checking if the MySQL database handles multiple read/write requests without crashing.

7.2.5 Concurrency Testing

We ensured the interface is user-friendly and responsive:

- **Responsiveness:** Verified that the website looks good on laptops, tablets, and mobile screens.

- **Navigation:** Checked that buttons like "Book Now" and "Login" are easily accessible.
- **Feedback:** Ensured users see success messages ("Booking Confirmed") or error messages ("Invalid Credentials").

7.2.6 Security Testing

We implemented and tested security measures to protect user data.

- **SQL Injection:** Used SQLAlchemy ORM to prevent malicious SQL queries.
- **Password Security:** Verified that passwords are hashed (encrypted) using generate_password_hash before storage.
- **Access Control:** Confirmed that patients cannot access Admin-only pages

7.3 Implementation

7.3.1 Frontend Implementation

The user interface was built using standard web technologies for speed and compatibility.

Technologies Used:

- **HTML5** (Structure and layout architecture)
- **CSS3** (Custom styling and responsive design)
- **JavaScript** (Client-side logic and DOM manipulation)
- **Fetch API** (Asynchronous API communication)

Key Implementation Points

- **DoctorList, BookingForm, Login, and AdminDashboard** created as separate HTML pages or dynamic sections.
- **Dynamic Content Rendering** implemented using JavaScript to populate doctor lists and appointment slots from API data.
- **Event Handling** managed via event listeners for form submissions and button clicks.
- **Client-side Validation** ensures correct data format before sending requests to the backend.

Sample Code Snippet – Fetch Doctors

```
document.addEventListener('DOMContentLoaded', async () => {
 try {
  const response = await
  fetch('http://127.0.0.1:3000/api/doctors'); const data = await
  response.json(); displayDoctors(data.doctors);
 } catch (error) {
  console.error('Error:', error);
 }
});
```

7.3.2 Backend Implementation

Technologies Used**:**

- Python (Core programming language)
- Flask (Web framework for creating REST APIs)
- SQLAlchemy (ORM for database management)
- PyMySQL (MySQL connector for Python)
- JWT (JSON Web Tokens) (Secure user authentication)

Key Implementation Points

- API Endpoints created using Flask route decorators (@app.route) instead of separate files.
  - /api/auth/login
  - /api/doctors
  - /api/appointments
  - /api/admin/stats
- JSON Responses structured automatically using Flask's jsonify() function.
- Authentication handled via a custom @token_required decorator to protect sensitive routes.
- Input Validation applied directly within route functions before committing to the database.

Sample Code – Book Appointment API

```
@app.route('/api/appointments',
methods=['POST']) @token_required
```

```
def book_appointment(current_user):
    data = request.get_json()

    new_appt = Appointment(
        patientId=current_user.id,
        doctorId=data['doctorId'],
        date=data['date'],
        status='upcoming'
    )

    db.session.add(new_appt)
    db.session.commit()

    return jsonify({'success': True, 'message': 'Appointment booked successfully'})
```

7.3.3 Database Implementation

- **Technologies:** MySQL, SQLAlchemy (ORM).

Key Implementation Points

- **Tables:** Created tables for Users, Doctors, Appointments, and Schedules.
- **Relationships:** Linked Patients and Doctors via the Appointment table using Foreign Keys.
- **Auto-Creation:** The system automatically creates tables using db.create_all() if they don't exist.

7.3.4 Deployment Implementation

The system was deployed locally for testing purposes. Steps

1. Installed dependencies using pip install flask flask-sqlalchemy pymysql.
2. Configured the MySQL connection string in app.py.
3. Ran the server using the command python app.py.

4. The application runs on http://127.0.0.1:3000, making it accessible via any web browser.

The system is now accessible through a standard web browser.

CHAPTER 8

SYSTEM SECURITY:

Security is a crucial aspect of any web-based healthcare application, especially one involving sensitive patient data, medical records, and appointment schedules. The HealthEase system implements multiple layers of security across the frontend, backend, and database to protect against unauthorized access, data breaches, and malicious attacks.
This chapter explains the security principles, techniques, and mechanisms applied in the system.

8.1 Introduction:

The purpose of system security in HealthEase is to ensure:

- **Confidentiality:** Patient data and doctor records are protected from unauthorized access.
- **Integrity:** Medical records and appointment details remain accurate and cannot be tampered with.
- **Availability:** The system remains accessible for legitimate users (patients and doctors) without interruption.
- **Authentication:** Only valid users and administrators can log in to the system.
- **Authorization**: Users can only access functionality permitted to their specific role (e.g., Patients cannot access Admin settings).

The HealthEase system incorporates multiple security practices across different layers to maintain a secure environment.

8.2 Security in Software

This section describes the security mechanisms implemented in the software code to protect user data, appointment details, and system integrity.

8.2.1 Authentication Security

- Users (Patients) and Admins must log in using valid credentials (email and password).
- Passwords are never stored in plain text; they are hashed using secure algorithms before storage.
- The backend validates all login requests through REST APIs before issuing a secure token.

Purpose: Prevents unauthorized access to patient profiles and admin dashboards.

8.2.2 Authorization and Role-Based Access Control (RBAC)

- **Patients:** Can only book appointments, view their own history, and browse doctors.
- **Admins:** Have full access to manage doctors, schedules, and view all system reports.
- The backend (@token_required decorator) verifies the user's role before processing any sensitive request.

Purpose: Ensures that patients cannot modify doctor data or view other patients' records.

8.2.3 SQL Injection Prevention

- The system uses **SQLAlchemy ORM** (Object Relational Mapping) instead of raw SQL.
- This automatically handles parameter binding, ensuring that malicious SQL code injected into input fields (like login forms) is treated as data, not executable code.

Purpose: Prevents attackers from manipulating the database or stealing sensitive information

8.2.4 Input Validation & Sanitization

- User inputs (such as registration details, booking dates, and phone numbers) are validated on both the **Frontend** (JavaScript) and **Backend** (Python).
- The system checks for invalid characters, correct email formats, and mandatory fields before processing data.
- Backend checks for invalid characters, missing fields, and data format errors.

Purpose: Prevents malformed requests and reduces the risk of Cross-Site Scripting (XSS) attacks.

8.2.5 Session Management & Token Security

Session-based security (Servlet sessions)

- The system uses **JSON Web Tokens (JWT)** for secure session management
- Upon login, the server generates a signed token containing the user's ID and Role.
- This token must be included in the header of every subsequent API request (Authorization: Bearer <token>).

Purpose: Ensures stateless and secure user identity verification across different pages.

8.2.6 Password Security

- P Passwords are hashed using **PBKDF2** or **scrypt** (via Werkzeug security library) before being saved to the MySQL database.
- Hashing converts the password into a fixed-length string of characters that cannot be reversed to reveal the original password.

Purpose: Protects user accounts even if the database is compromised.

8.2.7 Secure API Communication

- All API endpoints enforce strict content-type checks (expecting JSON).
- **CORS (Cross-Origin Resource Sharing)** policies are configured to allow requests only from trusted frontend domains.

Purpose: Prevents unauthorized external websites from interacting with the Health Ease backend.

8.2.8 Concurrency Control in Seat Booking

To prevent double-booking (two patients booking the same doctor at the same time):

- The backend checks slot availability immediately before confirming the booking.

- Database transactions ensure that the booking process is atomic—either it completes fully, or it fails completely.

Purpose: Prevents race conditions and scheduling conflicts.

8.2.9 Error Handling & Logging

- The backend is configured to catch exceptions and return generic error messages to the user (e.g., "Internal Server Error") instead of exposing system details or stack traces.
- Failed login attempts and system errors are logged on the server for administrator review.

Purpose: Prevents attackers from gaining insights into the system architecture through error messages .

8.2.10 Database Security

- The MySQL database is protected with a strong root password.
- Access privileges are restricted; the application connects with a user that has only the necessary permissions (SELECT, INSERT, UPDATE)
- Regular database backups ensure data can be restored in case of corruption.

Purpose: Protects the core data repository from unauthorized direct access.

CHAPTER 9

CONCLUSION:

TThe **HealthEase** Doctor Appointment Booking System successfully automates and simplifies the operations of traditional clinics by integrating modern web technologies with a structured backend architecture. The system provides patients with an intuitive interface to browse doctor profiles, check real-time availability, select time slots, and complete appointment bookings seamlessly. For administrators, the system offers efficient tools to manage doctor details, configure schedules, and generate appointment reports.

The adoption of a **Frontend built with HTML, CSS, and JavaScript** ensures a lightweight and responsive user experience. The use of **Python Flask** in the backend provides robustness and reliability for business logic processing, while **MySQL** ensures secure and consistent data management for patient records and doctor schedules. The system's **REST API architecture** enhances scalability, allowing future integrations with mobile applications or third-party services like payment gateways.

Through structured system design, proper testing, secure implementation, and modular development, the project meets the objectives outlined at the beginning. It resolves the problems present in traditional manual scheduling systems by eliminating physical queues, reducing human errors, improving data accuracy, and providing real-time booking functionalities.

Overall, the HealthEase system demonstrates the benefits of combining standard frontend technologies with a stable backend framework to deliver a complete, efficient, and secure healthcare application. It provides a solid foundation for further enhancements and real-world deployment in medical facilities.

CHAPTER 10

FUTURE SCOPE:

The **HealthEase** system provides a strong and functional foundation for digitalizing clinic operations. However, there are several opportunities to enhance the system further by integrating advanced technologies, improving user experience, and expanding functionality. The following points outline the possible future enhancements of the system:

10.1 Integration of Online Payment Gateway:

Currently, the system records consultation fees but does not process actual transactions. In the future, the system can integrate real payment gateways such as **Razorpay**, **Stripe**, or **UPI APIs**. This would allow patients to pay consultation fees securely online while booking their appointment, reducing manual cash handling at the clinic.

10.2 Mobile Application (Android/iOS)

- A dedicated mobile application can significantly improve accessibility for patients. Using cross-platform frameworks like **React Native** or **Flutter**, the web system can be extended to mobile devices.

  Key features would include:

  - Mobile appointment booking.
  - Push notifications for appointment reminders.
  - Offline access to prescriptions.
  - GPS-based clinic locator.

10.3 AI-Based Disease Prediction/Chatbot

An AI-powered chatbot can be integrated to assist patients before they book an appointment. By analyzing reported symptoms, the AI can:

- Suggest the most appropriate **Doctor Specialization** (e.g., suggesting a Cardiologist for chest pain)
- Answer Frequently Asked Questions (FAQs) regarding clinic timings and services.
- Provide preliminary health tips based on symptoms.

10.4 Telemedicine / Video Consultation Integration

To make healthcare more accessible, the system can support **Virtual Consultations**. Secure video conferencing tools (like WebRTC or Zoom API) can be integrated, allowing doctors to consult with patients remotely. This is particularly useful for follow-up visits or for patients who cannot travel to the clinic physically.

10.5 Multi-Clinic / Hospital Chain Support

The existing system is designed for a single clinic or hospital. Future improvements can expand the backend architecture to support **multiple clinics or hospital chains** under a single brand:

- Centralized management for multiple branches.
- Unified patient records accessible across different hospital locations.
- Distributed database infrastructure for handling large-scale data.

10.6 Email and SMS Notifications for Appointments

The system can be enhanced to generate automated communication to reduce no-shows.

- **Confirmation:** Sending immediate email/SMS upon booking.
- Reminders**:** Automated alerts sent 24 hours and 1 hour before the scheduled time.
- **Updates:** Notifying patients immediately if a doctor cancels or reschedules.

10.6 Cloud Deployment & Scalability

Deploying the system on cloud platforms such as **AWS (Amazon Web Services)**, **Microsoft Azure**, or **Google Cloud** can significantly improve performance. Benefits would include:

- Automatic load balancing to handle high traffic during peak hours.
- Global accessibility for patients and doctor

Enhanced security compliance for medical data storage.

10.7 Enhanced Admin Analytics Dashboard

Future versions can include interactive visual dashboards for administrators to make data- driven decisions.

- **Patient Inflow:** Graphs showing peak visiting hours and days.
- **Revenue Tracking:** Daily, weekly, and monthly revenue reports.
- **Doctor Performance:** Analytics on the most visited doctors or departments.

REFERENCES

[1] Pallets Projects, "Flask: A micro web framework for Python," *Pallets Projects Documentation*, vol. 3, no. 1, 2024. Available Online: https://flask.palletsprojects.com

[2] Python Software Foundation, "The Python Language Reference, version 3.12," *Python Documentation*, 2024. Available Online: https://docs.python.org/3/

[3] Oracle Corporation, "MySQL 8.0 Reference Manual," *MySQL Technical Documentation*, 2023. Available Online: https://dev.mysql.com/doc/refman/8.0/en/

[4] Mozilla Developer Network (MDN), "Web technologies for developers (HTML, CSS, JavaScript)," *MDN Web Docs*, 2024. Available Online: https://developer.mozilla.org

[5] Bayer, M., "SQLAlchemy: The Database Toolkit for Python," *SQLAlchemy Documentation*, ver. 2.0, 2023. Available Online: https://www.sqlalchemy.org

[6] World Wide Web Consortium (W3C), "Architecture of the World Wide Web: Principles and guidelines," *W3C Technical Reports*, vol. 1, no. 1, 2022. Available Online: https://www.w3.org

[7] Fielding, R. T., "Architectural styles and the design of network-based software architectures," *University of California Technical Publications*, vol. 1, no. 1, 2000. Available Online: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[8] Pressman, R. S., "Software Engineering: A Practitioner's Approach," *McGraw-Hill Computer Science Series*, vol. 8, no. 1, 2014. Available Online: https://www.mhprofessional.com

[9] Silberschatz, A., Korth, H. F., & Sudarshan, S., "Database System Concepts," *McGraw-Hill Database Journal*, vol. 7, no. 1, 2020. Available Online: https://db-book.com

[10] Sommerville, I., "Software Engineering," *Pearson Education*, vol. 10, no. 1, 2016. Available Online: https://www.pearson.com