COLLEGE CODE :9534

COLLEGE NAME : V V COLLEGE OF ENGINEERING

DEPARTMENT : COMPUTER SCIENCE AND ENGINEERING

STUDENT NM-ID :

5F8D6B86BB865489E184CCCC3926D38E

6A6FFFB26826F05A4864CB31616E870A

F528A374A3B8E45858D4EB0C7013D31C

13AE651D2D1594BB5544D81ED76E57CB

ROLL NO :953423104078

DATE : 13/10/2025

TECHNOLOGY : FRONTEND

PROJECT NAME : HEALTHCARE APPOINTMENTSYSTEM

SUBMITTED BY,

NAME : J.SUYAMBURAJ (TEAM LEADER)

R.TENDULKAR (TEAM MEMBER)

Y SHAM DICKSON (TEAM MEMBER)

M.MUTHU RAJ (TEAM MEMBER)

MOBILE NO : 9952384719

Project Overview & Objectives

Problem Statement

The healthcare industry is facing several challenges due to the continued reliance on manual appointment systems. Patients often wait in long queues, hospitals experience scheduling conflicts, and doctors waste valuable time managing appointments rather than focusing on patient care.

Traditional appointment booking systems are inefficient, lack real-time updates, and provide no transparency between patients and doctors.

To solve these issues, this project proposes an online Healthcare Appointment System that digitizes the appointment booking process, simplifies doctor availability tracking, and provides a centralized platform for patients, doctors, and administrators to interact efficiently.

Project Objectives

- To develop a user-friendly web application that allows patients to book doctor appointments easily.
- To enable doctors to manage schedules, appointments, and patient details digitally.
- To allow administrators to monitor the system, manage users, and view statistics.
- To reduce manual workload, paperwork, and errors in appointment scheduling.
- To maintain data security and privacy using authentication and authorization.
- To provide real-time status updates (appointment confirmation, cancellation, or modification).

Key Features

1. **User Registration & Login**: Patients, doctors, and admins can register and securely log in.
2. **Role-Based Dashboards:** Separate panels for patients, doctors, and admins.
3. **Doctor Management**: Admin can add, update, and remove doctor details.
4. **Appointment Booking:** Patients can view available doctors and book appointments online.
5. **Doctor Availability**: Doctors can update available slots for consultation.
6. **Appointment Tracking**: View appointment status (Pending, Confirmed, Cancelled).
7. **Notifications**: Email or on-screen alerts for appointment status updates.
8. **Reports & Analytics:** Admin can view reports of total patients, appointments, and doctor activity

Expected Outcome

- After the successful implementation of this system, the following outcomes are expected:
- A fully functional, user-friendly web application for managing hospital appointments.
- Reduced patient waiting time and improved convenience through online booking.
- Automated and accurate appointment records for doctors and hospitals.
- Enhanced coordination between patients, doctors, and administrators.
- Secure data storage using Django's ORM and MySQL database.
- Scalable architecture capable of handling multiple hospitals, departments, and users.
- Improved hospital productivity and overall healthcare service quality.

Future Enhancements

In future versions, the Healthcare Appointment System can be expanded with more advanced features such as:

Online Payment Integration:

Allow patients to pay consultation fees securely through online payment gateways.

SMS and Email Notifications:

Send appointment reminders and confirmation messages to patients and doctors.

Video Consultation Module:

Include online consultation options through integrated video conferencing (for telemedicine support).

Electronic Medical Records (EMR):

Store patient medical histories, prescriptions, and reports securely within the system.

AI-Based Doctor Recommendation:

Suggest doctors based on patient symptoms, location, and previous records.

Hospital Resource Management:

Extend the system to manage hospital rooms, staff schedules, and inventory.

Mobile Application:

Develop an Android/iOS app version for better accessibility and real-time notifications.

Data Analytics and Insights:

Use data visualization tools to analyze hospital performance, patient trends, and doctor workloads.

Project Objectives

To develop a digital platform for healthcare appointments

Build a web-based system that allows patients to book, reschedule, or cancel doctor appointments online, eliminating the need for manual registration or long waiting queues.

To provide real-time doctor availability and scheduling

Enable patients to view doctors' available time slots instantly and allow doctors to manage their consultation schedules dynamically.

To ensure secure and role-based access control

Implement a role-based login system for Admin, Doctor, and Patient to maintain data privacy and control access permissions.

To automate hospital appointment management

Replace manual appointment tracking with an automated system that records, updates, and manages all bookings efficiently in a centralized database.

To enhance patient–doctor communication

Provide instant notifications, appointment confirmations, and reminders to improve coordination between patients and healthcare providers.

To create a centralized database using MySQL

Store all appointment, user, and doctor details securely in a single MySQL database, ensuring easy data retrieval and maintenance.

To reduce administrative workload and human errors

Automate repetitive tasks such as schedule updates, appointment logs, and data entry, minimizing manual errors and saving time.

To offer an intuitive and responsive user interface

Use HTML, CSS, and JavaScript to design a clean, mobile-friendly interface that ensures a smooth user experience across all devices.

Technology Stack

Frontend Technologies

The frontend is responsible for the user interface and user experience.

It allows patients, doctors, and admins to interact with the system easily.

HTML5: Used to design the structure of web pages, forms, and layout.

CSS3: Used for styling, colors, layouts, animations, and responsive design.

JavaScript (ES6): Adds interactivity to web pages, including form validation, dynamic data loading, and real-time updates.

Backend Technologies

The backend handles business logic, data management, and communication between the user interface and database.

Python 3.x: Core programming language used for server-side scripting.

Django Framework: Used to build a scalable and secure web backend using the MVC (Model–View–Controller) architecture.

Django ORM (Object Relational Mapper): Simplifies database queries and CRUD operations.

Database

The database stores and manages all data related to users, doctors, and appointments.

MySQL:

- Used as the primary relational database management system.
- Stores user credentials, appointment records, and doctor schedules.
- Provides fast query performance and supports Django integration**.**

Tools & Software Requirements

CategoryTool / Software Description

IDE / Code Editor -Visual Studio Code For writing and managing project code.

Framework/Django - Python-based web framework for backend.

Database MySQL / XAMPP **-**Database server and management interface.

Web Browser/Chrome / Edge -For testing and viewing the web application.

Version Control/Git & GitHub. -For source code management and collaboration.

Design Tools -Figma / Canva (optional) For designing UI mockups and layouts.

API Design & Data Model

Overview

The API design ensures smooth communication between the **frontend (HTML/CSS/JS)** and the **backend (Django)** using a RESTful architecture.
Each endpoint provides a structured way to perform CRUD (Create, Read, Update, Delete) operations on patients, doctors, appointments, reviews, and health records.
This allows data to flow securely and efficiently between different components of the system.

REST API Architecture

The system follows a RESTful design with JSON-based request and response formats.
Each resource (patient, doctor, appointment, record, review) has a unique endpoint for interaction.

Resource	HTTP Method	Endpoint	Description
Patient	GET	/api/patients/	Retrieve all patients
Patient	POST	/api/patients/	Add new patient
Doctor	GET	/api/doctors/	Retrieve all doctors

Resource	HTTP Method	Endpoint	Description
Doctor	GET	/api/doctors/{id}/	Get specific doctor details
Appointment	POST	/api/appointments/	Book a new appointment
Appointment	PUT	/api/appointments/{id}/	Update appointment details
Appointment	DELETE	/api/appointments/{id}/	Cancel appointment
Health Record	GET	/api/records/{patient_id}/	Get patient's medical records
Health Record	POST	/api/records/	Add new health record
Review	POST	/api/reviews/	Add a new doctor review
Review	GET	/api/reviews/{doctor_id}/	Get reviews for a doctor

Request & Response Format

Add Appointment (POST)

Request:


```
{
  "patient_id": 5,
  "doctor_id": 3,
  "appointment_date": "2025-10-20",
  "time_slot": "11:00 AM"
}
```

Response:


```
{
  "message": "Appointment booked successfully",
  "appointment_id": 22,
  "status": "Pending"
}
```


Add Health Record (POST)

Request:


```
{
  "patient_id": 5,
  "diagnosis": "High Blood Pressure",
  "prescription": "Amlodipine 5mg once daily",
  "doctor_notes": "Regular BP check advised"
}
```


Response:


```
{
  "message": "Health record added successfully",
  "record_id": 10
}
```


Add Doctor Review (POST)

Request:


```
{
  "doctor_id": 3,
  "patient_id": 5,
  "rating": 4.5,
```

```
  "comment": "Very attentive and helpful"
}
```

Response:

```
{
  "message": "Review submitted successfully",
  "review_id": 8
}
```

Data Validation & Error Handling

To ensure data accuracy and security, the API includes:

Input validation: Checks required fields, formats, and data types.

Error responses:

```
{
  "error": "Invalid data or missing field"
}
```

- **Authentication:** Token/session-based authentication to protect sensitive data.
- **Authorization:** Role-based access (Patient, Doctor, Admin) to control endpoint access.

Database Schema Design

Main Entities:

Patient Table

- patient_id (Primary Key)
- name
- date_of_birth
- gender
- email
- phone
- password (hashed)
- address

Doctor Table

- doctor_id (Primary Key)
- name
- specialization
- email
- phone
- rating (float)
- availability

Appointment Table

- appointment_id (Primary Key)
- patient_id (Foreign Key → Patient)

- doctor_id (Foreign Key → Doctor)
- appointment_date
- time_slot
- status (Pending / Confirmed / Cancelled)

HealthRecord Table

- record_id (Primary Key)
- patient_id (Foreign Key → Patient)
- doctor_id (Foreign Key → Doctor)
- diagnosis
- prescription
- report_file (optional upload)
- date_created

Review Table

- review_id (Primary Key)
- doctor_id (Foreign Key → Doctor)
- patient_id (Foreign Key → Patient)
- rating (1–5)
- comment
- review_date

Data Relationships

One-to-Many: A doctor can have many appointments and reviews.

One-to-One: Each appointment can generate one health record.

Many-to-One: Multiple patients can review the same doctor.

Security Considerations

- Enforce **CSRF protection** for all POST requests.
- Use **JWT or Django sessions** for authentication.
- **Password hashing** using Django's make_password() function.
- Validate all inputs to prevent SQL Injection and XSS attacks.

Database Query

Patients Table

```
CREATE TABLE Patients (
    patient_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    phone VARCHAR(15),
    date_of_birth DATE,
    gender ENUM('Male','Female','Other'),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Doctors Table

```
CREATE TABLE Doctors (
    doctor_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
```

```
    email VARCHAR(100) UNIQUE NOT NULL,
    phone VARCHAR(15),
    specialty VARCHAR(100),
    experience_years INT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Appointments Table

```
CREATE TABLE Appointments (
    appointment_id INT AUTO_INCREMENT PRIMARY KEY,
    patient_id INT NOT NULL,
    doctor_id INT NOT NULL,
    appointment_date DATETIME NOT NULL,
    status ENUM('Scheduled','Completed','Cancelled') DEFAULT 'Scheduled',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (patient_id) REFERENCES Patients(patient_id) ON
DELETE CASCADE,
    FOREIGN KEY (doctor_id) REFERENCES Doctors(doctor_id) ON DELETE
CASCADE
);
```

Health Records Table

```
CREATE TABLE HealthRecords (
    record_id INT AUTO_INCREMENT PRIMARY KEY,
    patient_id INT NOT NULL,
    doctor_id INT,
    record_date DATE NOT NULL,
    diagnosis TEXT,
```

```
    prescription TEXT,
    notes TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (patient_id) REFERENCES Patients(patient_id) ON DELETE CASCADE,
    FOREIGN KEY (doctor_id) REFERENCES Doctors(doctor_id) ON DELETE SET NULL
);
```

Doctor Reviews Table

```
CREATE TABLE DoctorReviews (
    review_id INT AUTO_INCREMENT PRIMARY KEY,
    doctor_id INT NOT NULL,
    patient_id INT NOT NULL,
    rating INT CHECK (rating >= 1 AND rating <= 5),
    comment TEXT,
    review_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (doctor_id) REFERENCES Doctors(doctor_id) ON DELETE CASCADE,
    FOREIGN KEY (patient_id) REFERENCES Patients(patient_id) ON DELETE CASCADE
);
```

Front-End UI/UX Plan

Purpose of Wireframe

The wireframe is a blueprint of your website/application that visually represents:

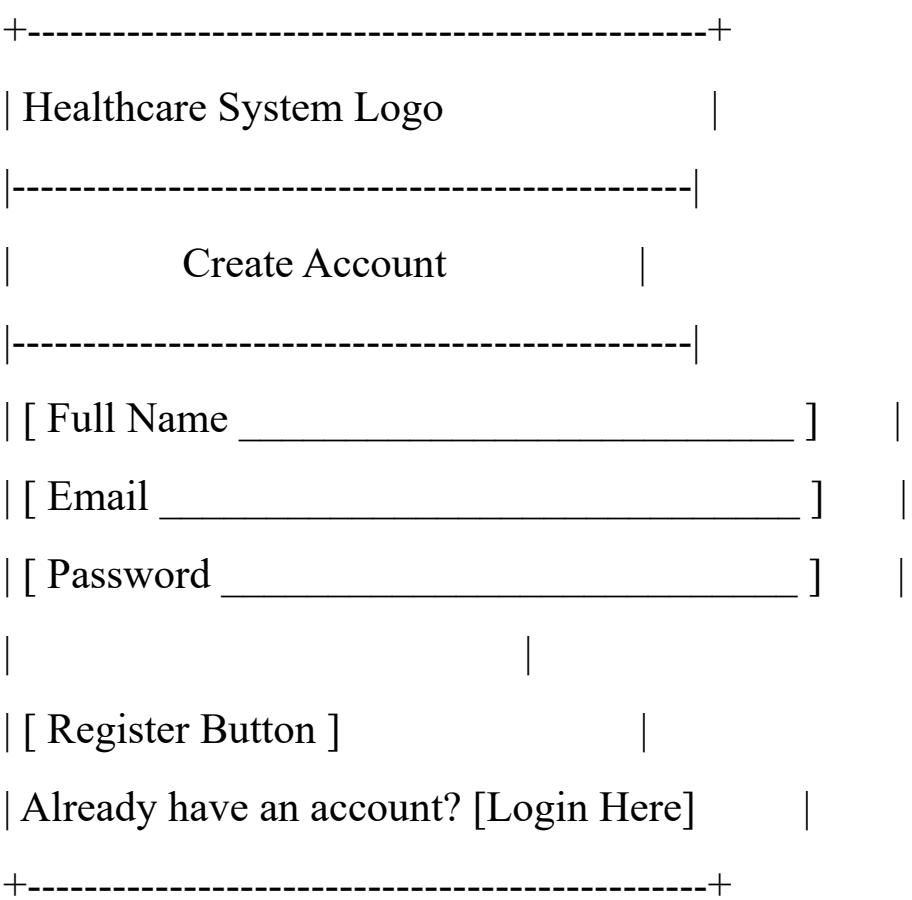
- Layout & hierarchy of elements
- Navigation between pages
- User interactions (buttons, forms, modals, etc.)
- Functional areas without focusing on colors or graphics

It ensures the UI/UX flow is clear before coding the frontend.

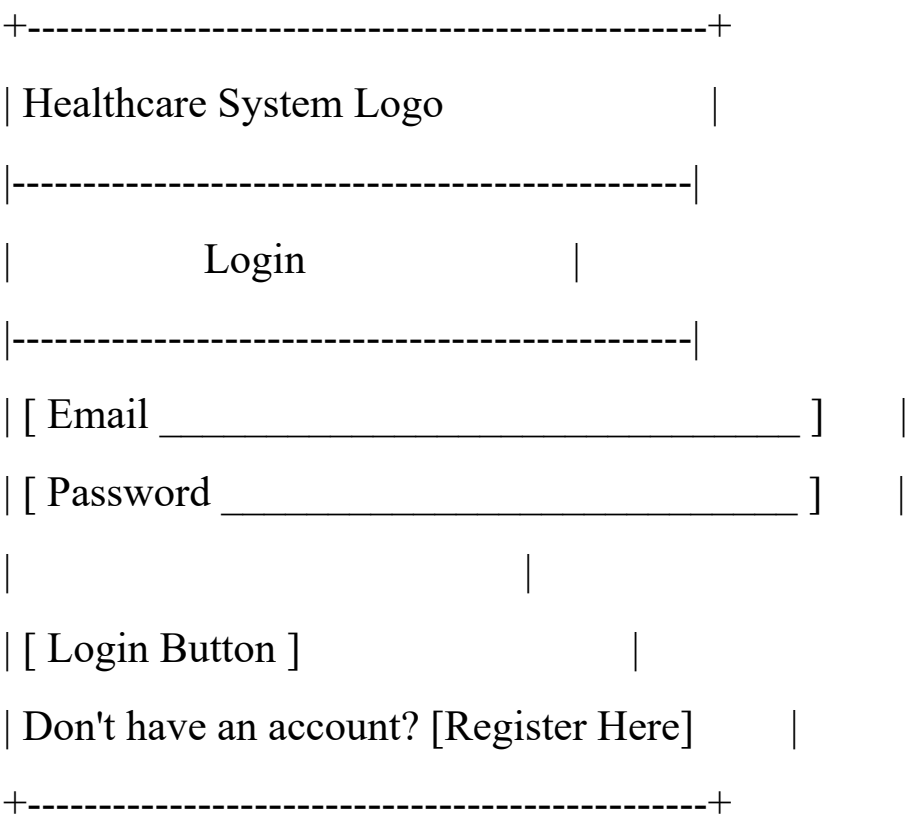
Design Philosophy

Principle	Description
Simplicity	Minimal clutter, clean input forms, and easy navigation
Clarity	Clear typography and call-to-action buttons
Consistency	Same header, footer, and button styles across all pages
Accessibility	Large text, color contrast, and readable layouts
Responsiveness	Mobile-first layout that adapts to different screen sizes

Register Page



Login Page



HOME

+--+

| LOGO | Home | Book Appointment | My Appointments | Profile | Logout |

|--|

| Hello, [User Name] |

|--|

| Welcome to Your Health Portal |

| [Card] Book Appointment |

| [Card] View Appointments |

| [Card] Doctor Reviews / Health Records |

|--|

| Footer © 2025 Healthcare System |

+--+

BOOK APPOINTMENT PAGE

+--+

| Header + Navigation Bar |

|--|

| Appointment Booking Form |

|--|

| [Full Name] |

| [Email] |

| [Phone] |

| [Select Doctor ▼] |

| [Date Picker] |

| [Time Picker] |

| [Book Appointment Button] |

VIEW APPOINTMENT

+--+

| Header + Navigation Bar |

|--|

| My Appointments Table |

|--|

| | Name | Doctor | Date | Time | Status | |

|--|

| [Edit] [Cancel] |

+--+

PROFILE PAGE

+--+

| Header + Navigation Bar |

|--|

| Profile Card |

|--|

| 👤 Name: [User Name] |

| 📧 Email: [User Email] |

| 📅 Joined: [Date] |

| [Edit Profile] [Logout] |

+---+

Navigation Flow

The **Navigation Flow** defines how users move between pages/screens, including logged-in and logged-out states. It ensures **logical transitions** and a smooth **user experience**.

High-Level User States

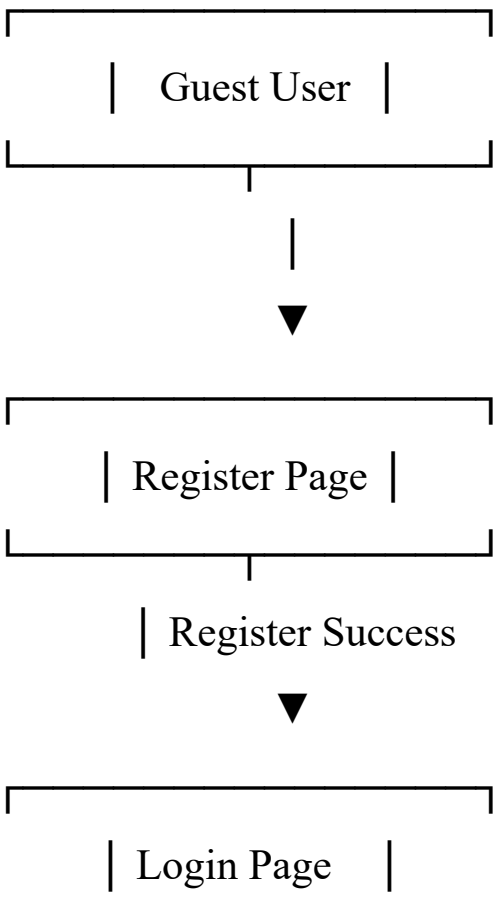
User State	Description
Guest / Not Logged In	Users who haven’t registered or logged in yet.
Registered & Logged In	Users who have an account and are authenticated.

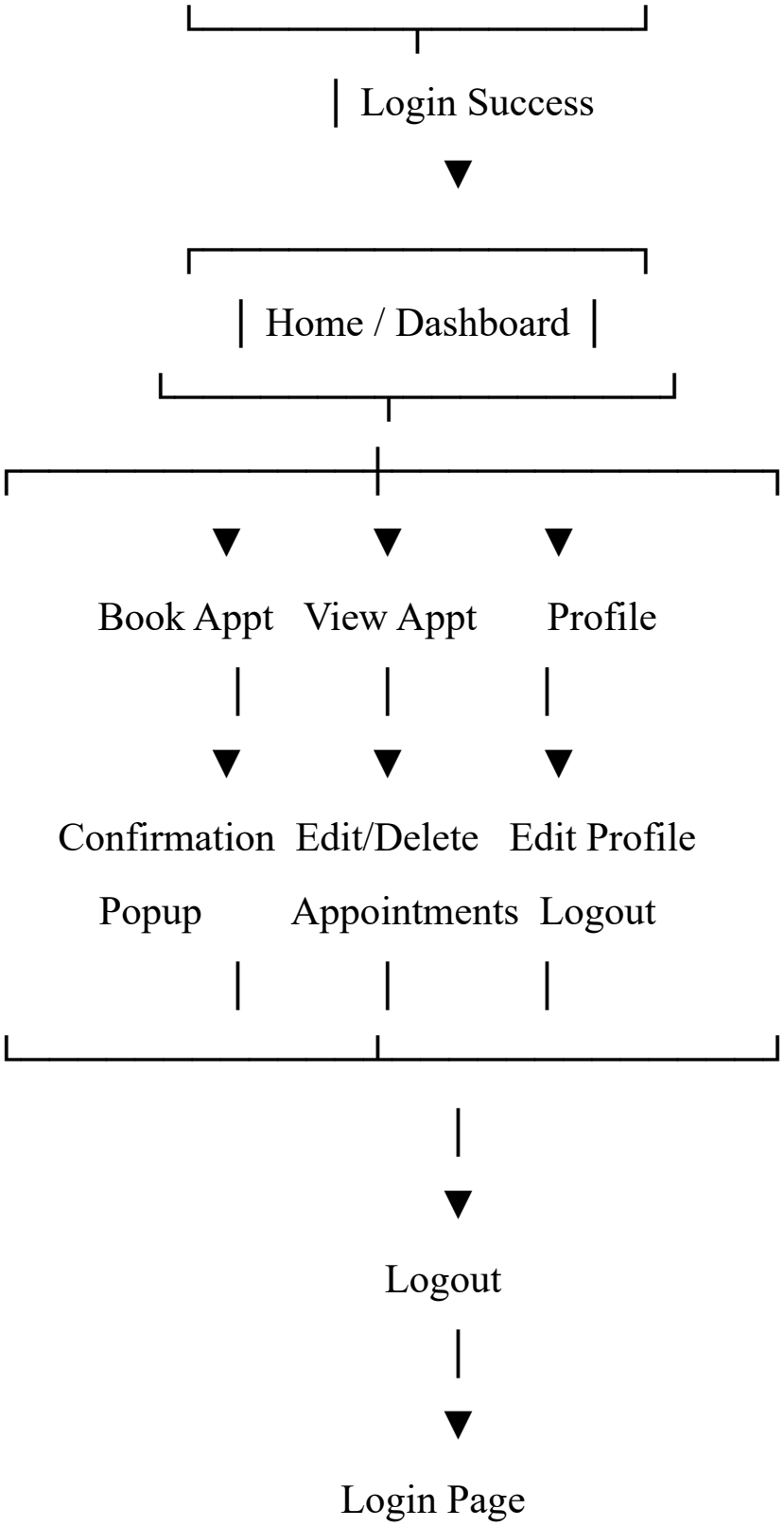
Page List & Functionality

Page	Description
`register.html`	Allows new users to create an account.
`login.html`	Allows users to log in with credentials.
`index.html`	Home page / Dashboard showing profile info, quick actions.

Page	Description
`appointments.html`	Book a new appointment.
`view.html`	View, edit, or cancel existing appointments.
`profile.html`	View and edit personal profile (optional separate page).
`doctor-reviews.html`	Browse and rate doctors (optional).
`health-records.html`	View, upload, download patient health records (optional).

Navigation Flow Diagram




Login Success
Home / Dashboard
Book Appt
View Appt
Profile
Confirmation Popup
Edit/Delete Appointments
Edit Profile
Logout
Logout
Login Page

Detailed Navigation Rules

Guest Users (Not Logged In)

- Can access: `register.html` and `login.html` only.
- Cannot access: Home, Appointments, Profile pages.
- Redirects to Login if they try to access protected pages.

Registered & Logged-In Users

- Can access all pages: `index.html`, `appointments.html`, `view.html`, `profile.html`.
- **Navbar links** change dynamically based on login status:
  - Show **Logout** button.
  - Show **Profile** link.
  - Hide **Login/Register** links.

Appointments Navigation

- From **Home / Dashboard**, user clicks **Book Appointment** → goes to `appointments.html`.
- After submitting a booking:
  - Display **confirmation popup**.
  - Redirect back to **View Appointments** (`view.html`) optionally.
- **View Appointments** table allows editing or canceling appointments.

Profile Navigation

- Users can view profile info at the top of the home page (or separate `profile.html`).
- Options:

- Edit profile info (name, email, phone).
- Change password.
- Logout.

Logout Navigation

- Clicking **Logout**:
  1. Clears `loggedIn` flag from localStorage.
  2. Redirects to **login.html**.
  3. Protected pages cannot be accessed until login again.

Mobile / Responsive Navigation

☰ Menu
├─ Home
├─ Book Appointment
├─ View Appointments
├─ Profile
└─ Logout/Login/Register

- Buttons / cards stack vertically for mobile view.
- Persistent Dark Mode toggle in navbar.

Navigation UX Best Practices

1. Highlight Active Page — navbar link is highlighted.
2. Breadcrumbs — optional for nested pages like appointment details.
3. Confirmation Modals — e.g., logout, cancel appointment.
4. Access Control — redirect unauthorized users to login.
5. Persistent State — logged-in session stored in `localStorage`

State Management Approach

Purpose of State Management

State management ensures the application **remembers user data and app state** across:

- Pages (Home, Appointments, Profile)
- Sessions (login persistence)
- User interactions (form submissions, dark mode toggle)

It helps to:

- Keep track of logged-in users
- Store appointments
- Store UI preferences like dark mode
- Manage temporary states like form validation

Types of State in the App

State Type	Example Use Case	Storage Mechanism
User Authentication	Logged-in user details, login status	`localStorage`
Appointment Data	Booked appointments, view/edit/cancel	`localStorage`
UI State	Dark mode toggle, active page highlighting	`localStorage` / JS
Form State	Temporary form inputs before submission	In-memory JS (variables)
Notifications	Alerts, success/error messages	In-memory JS / DOM

State Management Flow

User Authentication State

1. **Register / Login:**
   - On registration or login success:

     ```
     localStorage.setItem('user',
     JSON.stringify({name, email, password}));

     localStorage.setItem('loggedIn', 'true');
     ```

   - Redirect to `index.html`.
2. **Logout:**
   - Clear `loggedIn` flag:

     ```
     localStorage.setItem('loggedIn', 'false');

     window.location.href = 'login.html';
     ```

3. **Page Access Control:**
   - On page load, check `loggedIn`:

     ```
     const loggedIn =
     localStorage.getItem('loggedIn') === 'true';

     if (!loggedIn) window.location.href =
     'login.html';
     ```

Appointment State

- Store all appointments in **LocalStorage** as an array:

```
let appointments =
JSON.parse(localStorage.getItem('appointments')) ||
[];

appointments.push(newAppointment);

localStorage.setItem('appointments',
JSON.stringify(appointments));
```

On **View Appointments** page:

- Retrieve appointments from LocalStorage.
- Render in a table dynamically using JS.

Edit / Cancel:

- Update or remove the appointment in the array.
- Update LocalStorage and refresh table.

UI State (Dark/Light Mode)

- Toggle dark mode on navbar button click:

```
const dark = document.body.classList.toggle('dark-mode');
localStorage.setItem('darkMode', dark);
```

- On page load:

```
const isDark = localStorage.getItem('darkMode') === 'true';
document.body.classList.toggle('dark-mode', isDark);
```

Form State (Temporary)

- Input values are held in JS variables during typing.
- Upon submission, values are validated and either stored in LocalStorage or sent to backend (if integrated).

Optional Enhancement — Using JSON Objects

You can structure LocalStorage like this:

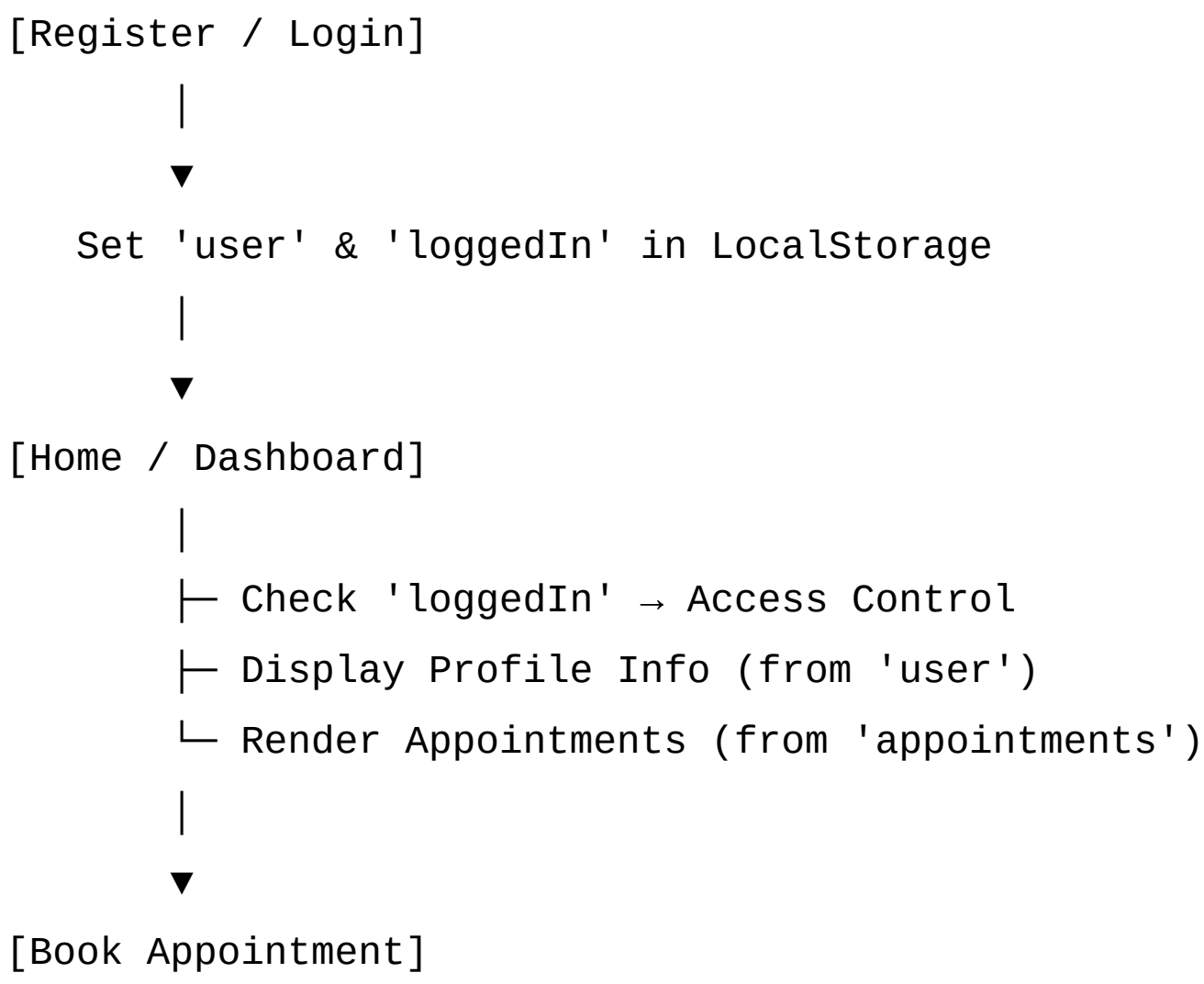
```
{
  "user": {
    "name": "John Doe",
    "email": "john@example.com",
    "password": "encryptedPassword"
  },
  "loggedIn": true,
  "appointments": [
    {
      "name": "John Doe",
      "email": "john@example.com",
      "phone": "1234567890",
      "doctor": "Dr. Smith",
      "date": "2025-10-15",
      "time": "10:00 AM"
    }
  ],
  "darkMode": true
}
```

This structure keeps all frontend state **centralized and easy to manage**.

Why LocalStorage / JS Variables

- **Pros:**
  - Simple and native, no backend needed.
  - Persistent across browser sessions.
  - Easy to implement for small apps.
- **Cons:**
  - Not secure for sensitive data (passwords, health info) — in production, a backend + secure storage is required.
  - Limited to browser storage limits (~5MB).

State Flow Diagram (Simplified)



```
        |
        └─ Add new appointment → Update LocalStorage
        ▼
[View Appointments]
        └─ Retrieve appointments → Display Table
        └─ Edit / Cancel → Update LocalStorage
        ▼
[Logout]
        └─ Set 'loggedIn' = false → Redirect to Login
```

Future State Management Options

Option	Use Case
Redux (React)	Large-scale app with complex interactions
Vuex (Vue.js)	Centralized store for Vue projects
Context API (React)	Lightweight global state
IndexedDB	Persistent storage for large datasets

Development & Deployment Plan

Team Roles & Responsibilities

Role	Responsibilities
Project Manager	Plan project timeline, track progress, coordinate tasks.
Frontend Developer	Build UI using HTML, CSS, JS; implement responsive design, forms, and navigation flow.
Backend Developer	Handle database, authentication, APIs (optional if backend is used).
UI/UX Designer	Create wireframes, mockups, color schemes, and user flows.
QA / Tester	Test functionality, performance, and UI responsiveness; report bugs.
DevOps / Deployment Engineer	Deploy the project to web hosting or cloud, manage CI/CD pipelines (optional for solo projects).

Git Workflow / Version Control

main (or master)

│

├─ feature/login-page

├─ feature/register-page

├─ feature/dashboard

├─ feature/appointments

Workflow Steps:

Initialize Repository

```
git init
git add .
git commit -m "Initial commit"
git branch -M main
git remote add origin <github-repo-url>
git push -u origin main
```

- Feature Branch
- Create a branch for each new feature:

  ```
  git checkout -b feature/appointments
  ```

- Implement feature → commit regularly:

  ```
  git add .
  git commit -m "Added appointment booking form"
  ```

- Pull Request & Merge
- Push branch → open Pull Request (PR) to `main`.
- Review → Merge into `main`.
- Best Practices
- Commit often with descriptive messages.
- Pull latest changes before starting a new feature.
- Keep `main` branch stable; all testing done in feature branches.

Testing Approach

Testing ensures the application is functional, responsive, and user-friendly.

A. Manual Testing

- Form Validation: Empty inputs, invalid email, invalid date/time.
- Navigation Testing: Ensure links, buttons, and redirects work.
- Login & Registration: Successful registration, login failure for wrong credentials.
- Appointment Flow: Book, view, edit, cancel appointments.
- Profile Update: Edit details and see the changes persist.

B. Responsive Testing

- Test on multiple devices (desktop, tablet, mobile).
- Check layout using Chrome DevTools or real devices.

C. Functional Testing

- Check LocalStorage persistence.
- Dark/Light mode toggle works on reload.

D. Optional Automated Testing

- Use Jest / Cypress / Playwright for frontend interaction tests if needed.

E. Bug Tracking

- Use GitHub Issues to track bugs and features.

Hosting & Deployment Strategy

Hosting Platform	Features / Notes
GitHub Pages	Free, simple, supports HTML/CSS/JS directly.
Netlify	Free plan, continuous deployment from GitHub repo, supports custom domains.
Vercel	Free, automatic deployment from GitHub, fast CDN.
Firebase Hosting	Free tier, HTTPS, supports JS SPA routing.

Deployment Steps (Example: GitHub Pages)

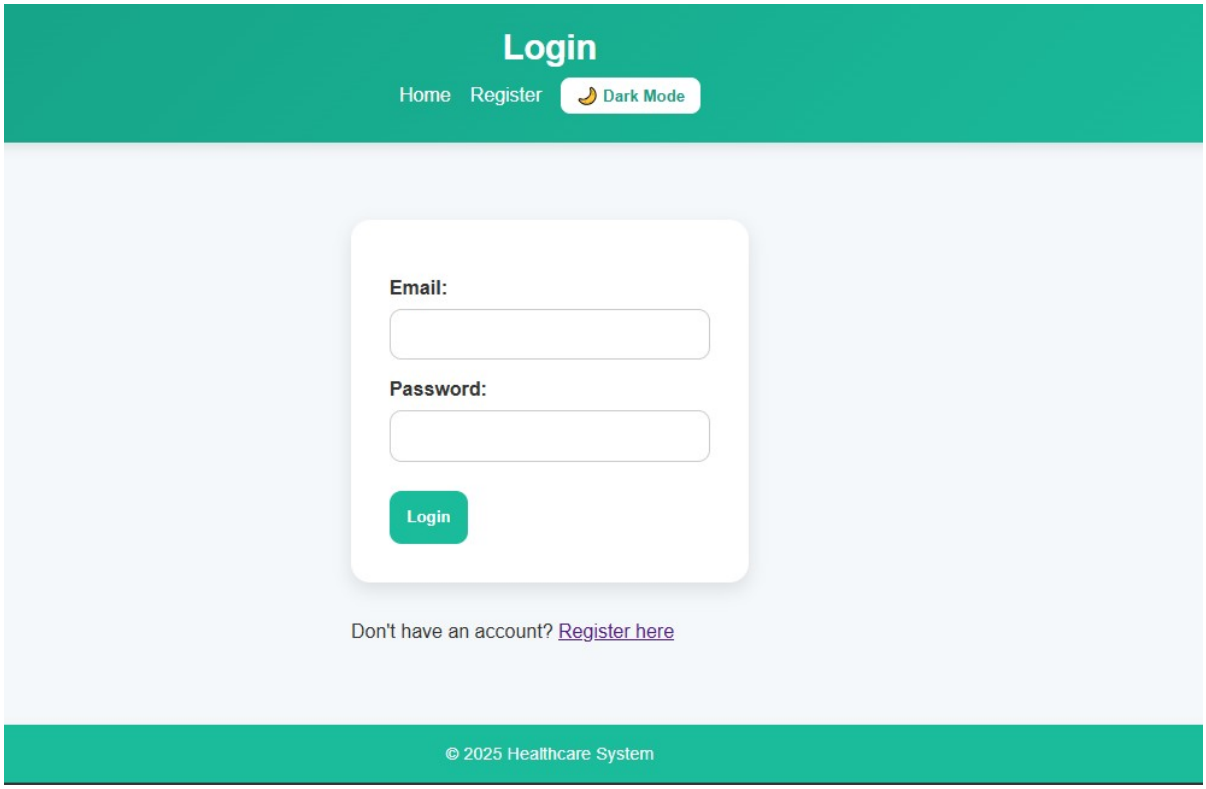
1. Push your project to GitHub repository.
2. Go to **Settings → Pages → Source → main branch / root**.
3. Save → GitHub generates a live URL.
4. Test live site → Ensure LocalStorage features work correctly.

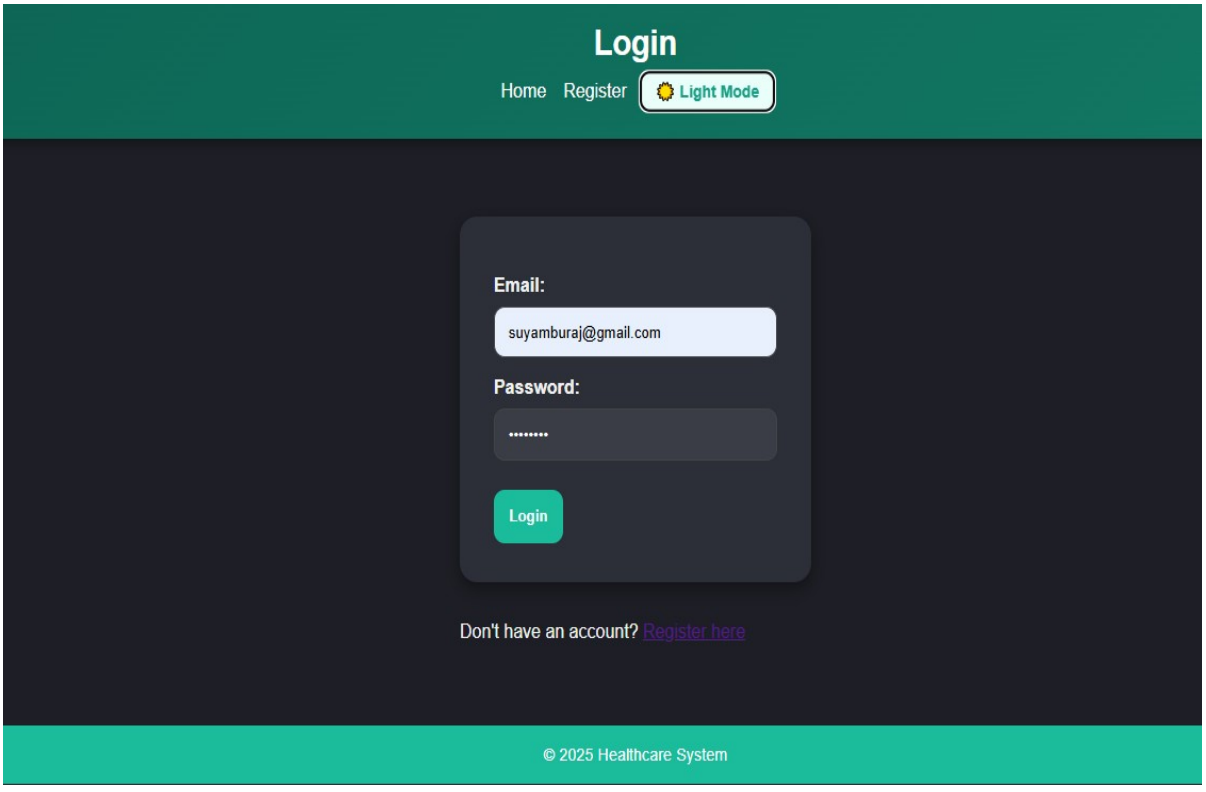
Deployment Steps (Example: Netlify)

1. Sign up at Netlify.
2. Connect GitHub repository.
3. Set build settings (for static HTML/CSS/JS, leave default).
4. Deploy → get a live URL.
5. Optional: Add **custom domain**.

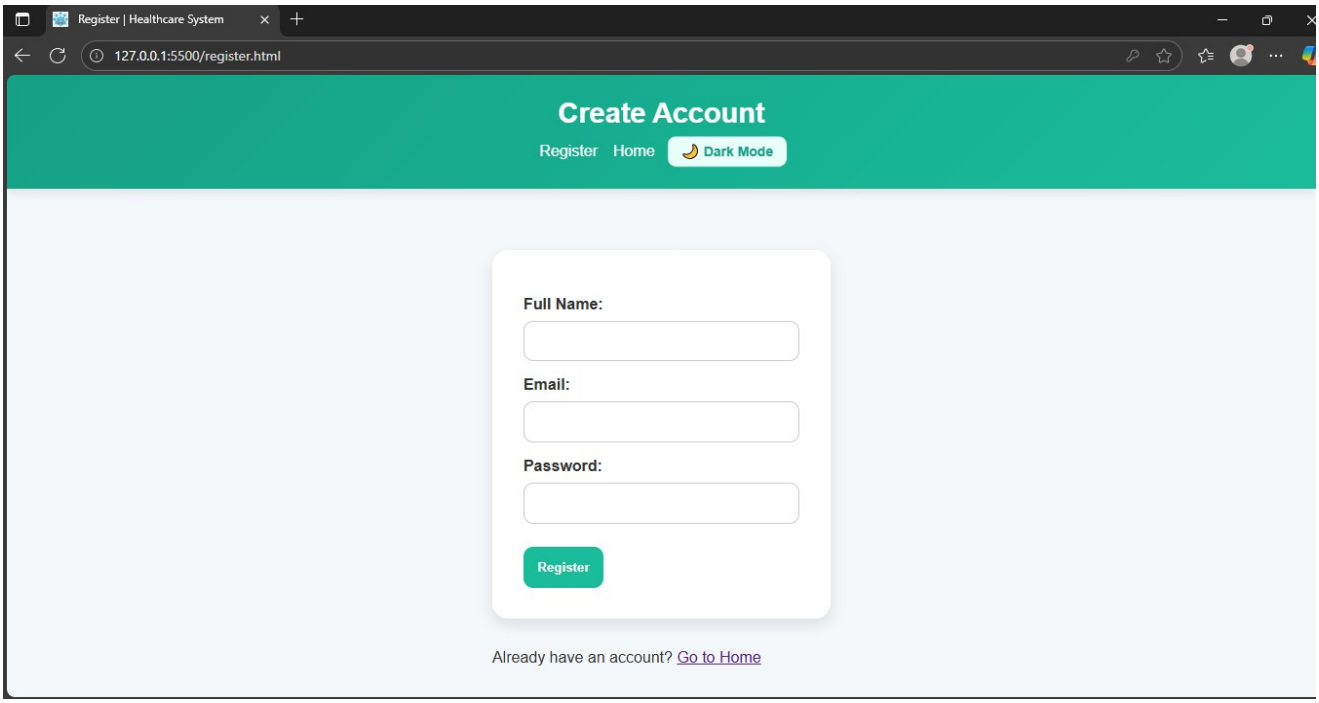
SCREENSHOT

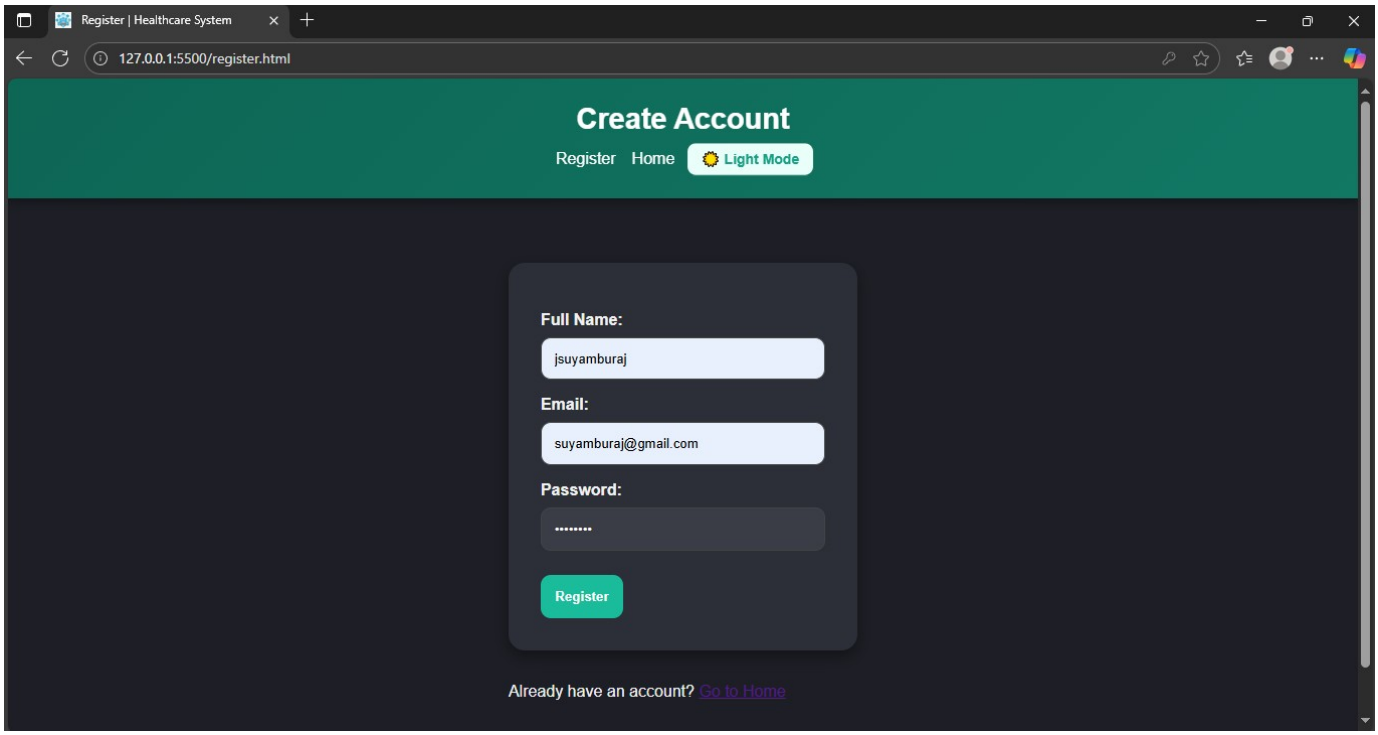
LOGIN PAGE



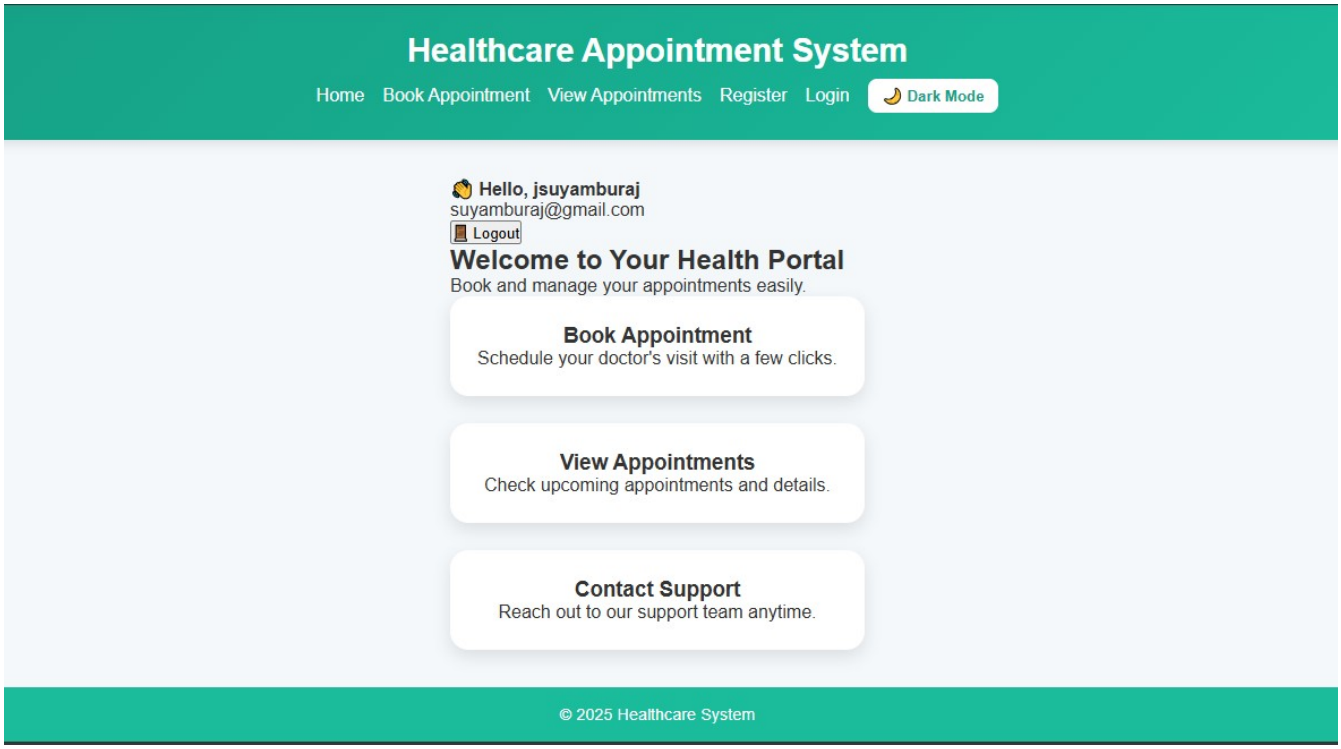


REGISTER PAGE

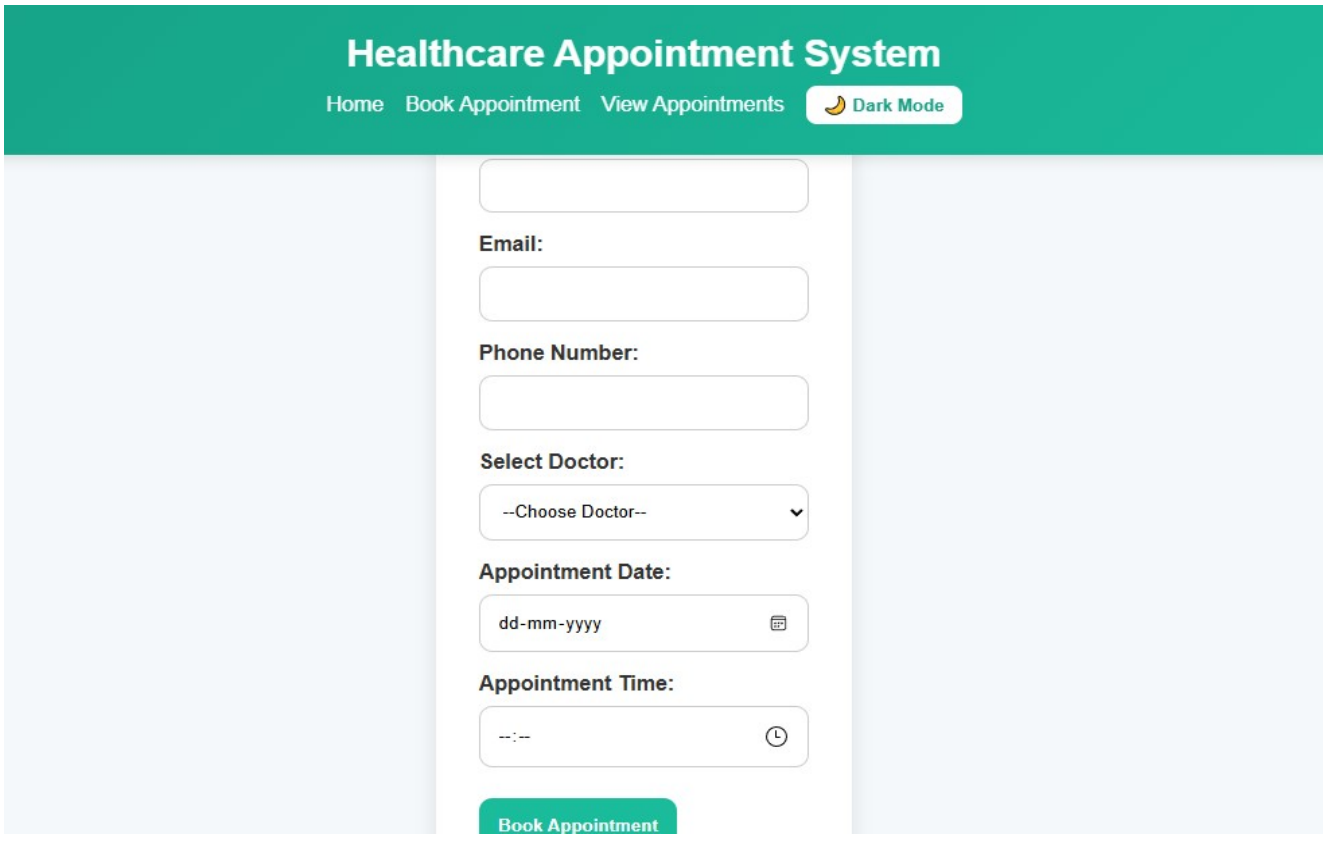




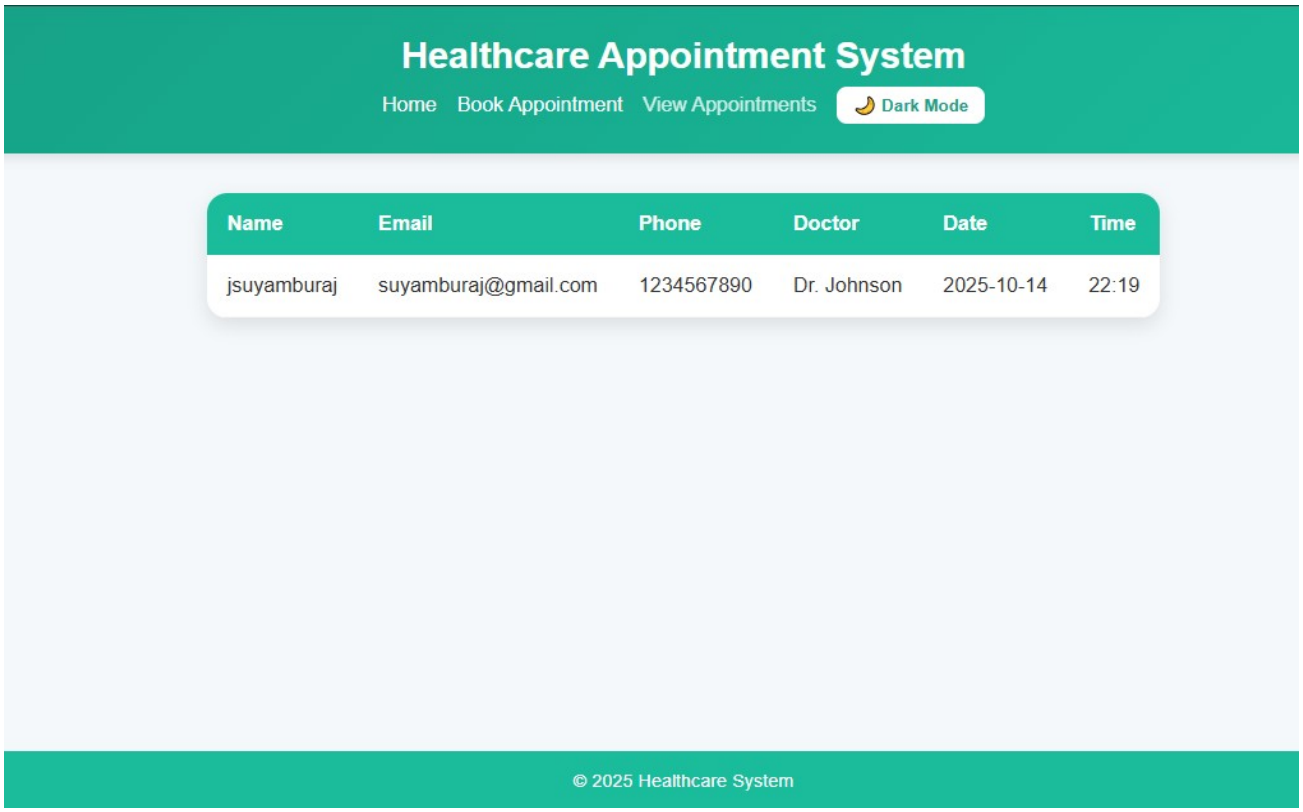
HOME PAGE



APPOINTMENT PAGE



VIEW PAGE



REGISTER SAVE