****

PROJECT REPORT ON **“HOSPITAL MANAGEMENT SYSTEM”**

Submitted in partial fulfilment of the requirements

of the award of the degree of

Bachelor of Computer Applications

Bengaluru North University



By

**NIJIN**

**U19BU22S0132**

Under the guidance of

**S Jayashree Ananth**

Assistant Professor

Department of Computer Applications

Koshys Institute of Management Studies

Bengaluru – 562149

****

Date:

**CERTIFICATE**

This is to certify that **NIJIN BIJU** bearing **U19BU22S0132** is the student of VI Semester Bachelor of Computer Applications of Koshys Institute of Management Studies. The project undertaken by the student and he has prepared Project report entitled “**Hospital Management System**”. The project report is submitted towards the partial fulfillment of the requirement of Bachelor of Computer Applications Degree of Bengaluru North University.

**Guide HOD**

**Vice- Principal**

**Examiner 1:**

**Examiner 2:**

## 

## STUDENT DECLARATION

I **NIJIN (U19BU22S0132)** hereby declare that this report entitled “**Hospital Management System**” submitted to the department of Computer Applications under the supervision and guidance of **Prof.** **Jayashree Ananth**, Assistant Professor, Department of Computer Applications, Koshys Institute of Management Studies, Bengaluru. The project is carried out by the student as per the requirement of Bengaluru North University.

**Date: Signature:**

**Place: NIJIN BIJU**

**U19BU22S0132**

## 

## ACKNOWLEDGEMENT

The successful completion of this Project report required significant guidance and assistance from many individuals, and I am truly grateful for their support throughout this journey.

I deeply grateful to our Director Dr. Prakash B Naik, Vice principal Dr. Rajasekaran S

and our HOD Dr. Reeja Jose Pavanathil for their unwavering support and for granting me the valuable opportunity to do the Project. I also express my sincere thanks to guide Prof. Jayashree Ananth**,** for her valuable guidance and timely suggestion at every stage of this project.

I would like to extend my heartfelt thanks to my parents for their support and constant encouragement throughout this project. Additionally, I am thankful to my friends for their support whenever I needed their assistance during this project.

Lastly, I would like to express my profound gratitude to all individuals who directly or indirectly contributed to the completion of this report.

**Place: Bengaluru**

**Date:**

**ABSTRACT**

The Hospital Management System (HMS) is a comprehensive, cloud-based web application designed to streamline the routine operations of a hospital through an intuitive digital interface. Developed with HTML, CSS, and JavaScript on the frontend and Firebase for backend services, HMS enables seamless interaction between administrators, doctors, and patients. The system supports three distinct user roles—Admin, Doctor, and Patient—each with role-specific dashboards and functionalities. Patients can securely register and log in to book appointments, manage personal information, and access their appointment history. Doctors can manage their availability, view scheduled appointments, and update patient notes following consultations. The Admin has full control over the system, including creating and managing departments, controlling user access, and overseeing overall operations.

HMS leverages Firebase Authentication to ensure secure login and role-based access control, while Cloud Firestore stores and manages real-time data, such as appointment details, user profiles, and medical records. This architecture allows efficient updates and reduces reliance on manual paperwork. The system significantly minimizes human error, enhances data accuracy, and improves coordination across departments. With centralized and real-time access to patient records, healthcare providers can deliver more timely and effective care. Additionally, HMS is modular and scalable, making it suitable for small clinics, mid-sized hospitals, or specialized care centers. Overall, the platform not only modernizes hospital workflows but also contributes to improved operational transparency, healthcare efficiency, and user satisfaction, establishing a strong foundation for digital transformation in the healthcare sector.

**EXECUTIVE SUMMARY**

The Hospital Management System (HMS) project was developed to streamline and digitize the essential operations of a healthcare facility. The primary objective was to create a web-based application that automates tasks such as appointment scheduling, patient record management, and communication between patients, doctors, and administrators. Traditional hospital systems often rely on manual processes, which can lead to inefficiencies, data loss, and miscommunication. HMS addresses these issues by providing a centralized, cloud-based solution that is accessible, secure, and easy to use.

The project utilizes modern web technologies such as HTML, CSS, and JavaScript for the front-end development, offering a responsive and intuitive user interface. On the backend, Firebase is integrated to handle user authentication and real-time database management. Firebase Authentication ensures secure, role-based access for three types of users: Admins, Doctors, and Patients, each with specific capabilities. For example, patients can register, book appointments, and view their history; doctors can manage their schedule and appointments; and admins oversee the entire system including user and department management.

The HMS platform was designed to be modular and scalable, enabling easy expansion of features and compatibility with larger healthcare environments. The system has undergone thorough testing, covering functionality, security, and usability. Test cases were created to validate user registration, login, appointment booking, data access control, and error handling.

In conclusion, the HMS project demonstrates a practical application of cloud-based technologies to improve healthcare service delivery. Its deployment is especially suited for small to mid-sized hospitals or clinics looking to modernize their operations while maintaining efficiency and accuracy.

**TABLE OF CONTENT**

|  |  |  |
| --- | --- | --- |
| Sl.No. | Topic |  |
| **1.**  1.1  1.2  **2**  2.1  2.2  2.3  **3**  3.1  3.2  3.3  **4**  4.1  4.2  **5**  5.1  5.2  5.3  **6**  **7**  **8**  **9** | **Introduction**  Problem Definition  Scope of Project  **System Study**  Existing System  Feasibility Study  Proposed System  S**ystem Design**  ER Diagram  DFD [level0, level1]  Input / Output Design  **System Configuration**  Hardware Requirement  Software Requirement  **Details of Software**  Overview of frontend  Overview of backend  About the platform  **Testing**  **Conclusion and Future Enhancement**  **Bibliography**  **Appendices A- Table structure**  **Appendices B- Screenshots**  **Appendices C- Sample Report of test cases** |  |

1. Introduction
   1. **Problem Definition**

In today’s fast-paced and technology-driven world, hospitals and healthcare institutions are expected to deliver services that are fast, reliable, and patient-centric. However, many hospitals, especially in developing regions or small towns, continue to rely on traditional manual systems for managing their operations, such as scheduling appointments, maintaining patient records, assigning doctors, and organizing departmental activities.

These outdated manual methods often result in disorganized data storage, lack of instant access to vital information, time-consuming processes, overlapping appointments, and critical delays in patient care. Furthermore, in a typical manual system, the chances of misplacing records, incorrectly scheduling appointments, or duplicating data entries are extremely high — leading to inefficiency, patient dissatisfaction, and sometimes even medical errors. Additionally, with increasing numbers of patients and the need for 24/7 service, the manual approach becomes extremely difficult to scale.

In such a scenario, there is an urgent need to design and implement an automated, cloud-based, real-time hospital management system that addresses the shortcomings of the manual method and provides a seamless experience to hospital staff, doctors, and patients. This project — the Hospital Management System (HMS) — aims to solve this problem by creating a complete, browser-based system that supports centralized data handling, remote access, appointment management, user profile tracking, and real-time doctor availability.

**1.2 Scope of Project**

The scope of the Hospital Management System (HMS) project is broad, scalable, and adaptable to the needs of various healthcare environments. It includes developing a fully functional, interactive, and real-time hospital management platform that supports key modules such as user authentication, role-based access (admin, doctor, patient), appointment scheduling, doctor availability tracking, medical record history, and departmental information management.

The system is designed using modern web development tools and frameworks — HTML and CSS are used to build a clean, responsive user interface; JavaScript is used to implement business logic and interactivity; and Firebase is used as the backend, offering secure authentication, real-time database updates, and cloud storage functionality. These technologies are selected not only for their performance and scalability but also because they offer low-cost and easily maintainable solutions.

HMS is intended to be used by:

* Patients, who can sign up, log in, manage their profile, book appointments, and view their appointment records.
* Doctors, who can view their schedules, check patient lists, and update appointment status.
* Admins, who can add or remove doctors, assign departments, and manage user access.

The system is fully browser-compatible, meaning users can access it from any device with internet access, without installing any software. This makes it ideal for small to mid-size hospitals, clinics, and medical institutions seeking digital transformation at minimal cost.

2.System Study

**2.1 Existing System**

In most conventional hospitals, especially in rural and semi-urban areas, the management of appointments, patient records, doctor schedules, departmental coordination, and user information is still carried out using manual procedures or semi-digital methods such as paper files, registers, and offline Excel sheets. These outdated systems often require human intervention at every step, which makes them extremely prone to human errors, inefficiencies, and delays. Furthermore, manual handling of such vast data — especially in a healthcare environment — compromises not only operational speed but also the accuracy and safety of critical patient information.

For example, if a patient walks into a hospital for a follow-up consultation, their previous medical history must be retrieved from physical storage systems, which can take a considerable amount of time, especially if files are misplaced, mislabelled, or stored improperly. There is also the risk of files being damaged due to environmental factors like fire, moisture, or general wear and tear. Additionally, appointment booking in such environments is often done over the phone or at the reception desk, which can lead to confusion, double bookings, and miscommunication between departments and doctors.

Doctors also face difficulties in accessing their daily schedules, as the lack of a digital platform prevents real-time synchronization between departments and the medical staff. There is no mechanism for alerting doctors to cancellations or newly booked appointments. Likewise, hospital administrators often struggle to keep an accurate record of the number of patients, departments, and doctors, and the amount of time spent on operational tasks increases significantly.

Moreover, data from one department is usually not easily accessible to another. For example, a doctor in the cardiology department may not be able to easily access a patient's previous visit to the neurology department, even though it might be relevant to current treatment. This fragmentation of data can lead to repetitive tests, delayed diagnoses, and frustration among patients and staff.

The lack of centralized data access, real-time updates, and system-wide integration in existing manual systems severely limits the hospital’s ability to provide fast, reliable, and quality healthcare. In the era of digital transformation, where cloud computing, big data, and AI are reshaping industries, hospitals that continue to depend on manual workflows are at a significant operational disadvantage.

Therefore, to meet modern healthcare standards and to fulfil the growing demands of scalability, reliability, and patient satisfaction, there is an urgent requirement for an automated Hospital Management System — one that can eliminate inefficiencies, ensure the accuracy of patient records, and bring every stakeholder (patients, doctors, administrators) onto a single digital platform

**2.2 Feasibility Study**

A feasibility study is one of the most important steps in the development of any software project. It is conducted to evaluate whether the proposed solution is technically achievable, economically viable, and operationally practical within the specific environment in which it is intended to be implemented. For the Hospital Management System (HMS), a thorough feasibility study was conducted to ensure that the system could be successfully deployed in hospitals without requiring large financial investments or complex technological infrastructure.

**a) Technical Feasibility**

The HMS project is technically feasible due to its use of modern yet lightweight web technologies that are highly compatible across platforms and devices. The frontend of the system is built using HTML, CSS, and JavaScript, which are the most widely adopted languages for developing responsive and interactive web applications. These technologies do not require high-end computers or operating systems and can run smoothly on any browser, making them accessible to users even in resource-constrained environments.

On the backend, the system uses Firebase, a powerful cloud-based platform offered by Google that provides real-time database services, user authentication, cloud hosting, and serverless architecture. Firebase’s modular and scalable services are ideal for web applications that require real-time synchronization, user management, and fast deployment. Since Firebase manages the backend infrastructure, developers do not need to worry about hosting servers, handling traffic spikes, or maintaining uptime — all of which are handled seamlessly by Google’s cloud network.

In essence, the use of Firebase eliminates the complexity of traditional server-side scripting and database management, thus making the HMS project not only technically feasible but also easily maintainable.

**b) Operational Feasibility**

Operational feasibility examines whether the system, once developed, can be used efficiently in the day-to-day activities of a hospital. The HMS is specifically designed to provide a smooth and intuitive user experience for all types of users — whether they are patients with no technical background, doctors who are busy and require quick access to data, or administrative staff managing hospital workflows.

The system uses role-based dashboards so that users only see what is relevant to them. Patients have a simplified dashboard where they can book appointments and view their records. Doctors see only their scheduled appointments and patient information. Admins have the ability to manage users, departments, and monitor hospital-wide activity.

Each user interface is clean, responsive, and easy to understand, minimizing the need for training. Additionally, since the system is web-based, it does not require any installations or updates — users only need a device with a web browser and internet access.

Hospitals can also use the system in parallel with their existing infrastructure, which allows for a smoother transition from manual to digital processes. This hybrid adaptability makes the HMS system operationally practical for immediate deployment.

**c) Operational Feasibility**

Operational feasibility examines whether the system, once developed, can be used efficiently in the day-to-day activities of a hospital. The HMS is specifically designed to provide a smooth and intuitive user experience for all types of users — whether they are patients with no technical background, doctors who are busy and require quick access to data, or administrative staff managing hospital workflows.

The system uses role-based dashboards so that users only see what is relevant to them. Patients have a simplified dashboard where they can book appointments and view their records. Doctors see only their scheduled appointments and patient information. Admins have the ability to manage users, departments, and monitor hospital-wide activity.

Each user interface is clean, responsive, and easy to understand, minimizing the need for training. Additionally, since the system is web-based, it does not require any installations or updates — users only need a device with a web browser and internet access.

Hospitals can also use the system in parallel with their existing infrastructure, which allows for a smoother transition from manual to digital processes. This hybrid adaptability makes the HMS system operationally practical for immediate deployment.

**d) Economic Feasibility**

The cost to implement the HMS project is minimal, making it economically feasible even for smaller hospitals and clinics. One of the key advantages of the technologies used is that they are free and open-source. HTML, CSS, and JavaScript have no licensing costs, and Firebase offers a generous free tier that includes:

* Firebase Authentication with up to 10k verifications/month
* Firebase with 50k reads/day and 1GB of storage
* Free hosting and SSL support
* Cloud Functions and analytics on-demand

With this free tier, most small to medium-scale hospitals can run the system without any upfront cost. If the hospital scales and requires more data or user access, Firebase offers flexible upgrade plans that are still much cheaper than building and maintaining traditional infrastructure.

Moreover, the HMS reduces long-term operational costs by minimizing paper use, improving staff efficiency, and preventing costly errors in appointment booking or patient records. It also reduces the administrative overhead typically required for managing hospital operations.

**2.3 Proposed System**

The Proposed System is a cloud-based, web-enabled Hospital Management System (HMS) developed to overcome the challenges of manual and semi-digital systems currently used in many hospitals. This system is designed to serve as a complete digital solution for managing the operational, clinical, and administrative needs of a hospital environment, while ensuring real-time access, minimal human error, and ease of use for patients, doctors, and administrators.

The goal of the proposed system is to bring together all core hospital functionalities — such as patient registration, appointment booking, doctor scheduling, and record maintenance — into a single, centralized platform that is easy to access and manage from any internet-enabled device. This ensures that patients can interact with the system even from their homes, doctors can manage their schedules on the go, and administrators can oversee hospital activities in real time.

User Roles in the Proposed System

The HMS project is built with role-based access, which means every user has a specific set of privileges and responsibilities:

**1. Patient Module**

Patients can create an account using a secure signup form. Once registered, they can:

* Log in to their personal dashboard using Firebase Authentication
* Update personal details such as name, age, gender, contact number
* Book appointments by selecting department, doctor, date, and time
* View appointment history and download booking records

All data entered by the patient is stored securely in the Firebase database and can be accessed or modified (with restrictions) at any time.

**2. Doctor Module**

Doctors are added to the system by the admin. Once their login credentials are active, doctors can:

* Log in and view a list of their scheduled appointments
* Check patient details for upcoming consultations
* Update appointment status to mark it completed or pending
* View past patient appointments for context-based diagnosis

Doctors have limited access, ensuring that patient data remains private and secure.

**3. Admin Module**

The administrator oversees the entire HMS and holds the highest level of access. Admin functions include:

* Adding, editing, or deleting doctor accounts
* Managing departments (Cardiology, Neurology, Pediatrics, etc.)
* Viewing a list of all patients, appointments, and activities
* Handling appointment rescheduling, cancellations, and system monitoring

**Core Features of the System**

The HMS includes several essential modules that work together to deliver a seamless experience:

* **Authentication Module** – Secures user access using Firebase Authentication.
* **Appointment Module** – Allows real-time appointment booking and management.
* **Dashboard Module** – Personalized views for patient, doctor, and admin roles.
* **Firebase Database** – Cloud-based storage that syncs in real time.
* **Responsive UI** – Mobile-friendly design using HTML, CSS, and JavaScript.
* **Modular Codebase** – Ensures easy future enhancements or module additions.

**Technology Stack Used**

| **Layer** | **Technology Used** |
| --- | --- |
| Frontend | HTML, CSS, JavaScript |
| Backend | Firebase Authentication & Firebase |
| Hosting | Firebase Hosting |
| IDE | Visual Studio Code |
| Versioning | GitHub |

**Security & Privacy**

Security is a top priority in this system. All user logins are verified using **Firebase Authentication**, which handles password encryption and session control. The Firestore database uses **role-based rules** to ensure that:

* Patients can access only their own data
* Doctors can view only their assigned appointments
* Admins can manage but not edit private patient records

This architecture ensures data privacy, integrity, and full compliance with common digital security practices.

3.SYSTEM DESIGN

System design is a vital phase in the software development life cycle that bridges the gap between requirements and implementation. It provides a structured blueprint of how different components in the Hospital Management System (HMS) will interact with one another, how data will be processed, and how users will interface with the application.

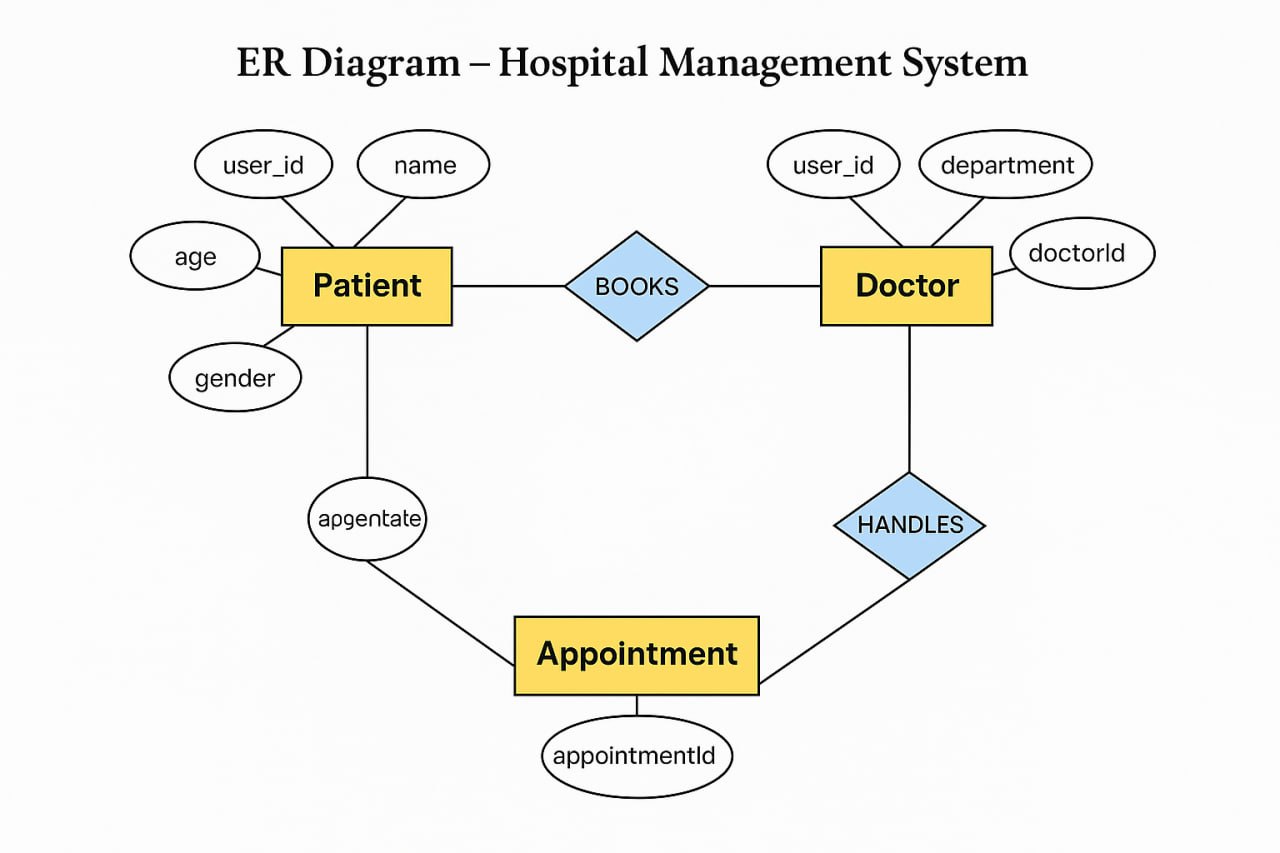
The objective of this section is to illustrate the internal structure and workflow of the system using industry-standard design tools such as Entity–Relationship (ER) Diagrams, Data Flow Diagrams (DFDs), and Input/Output Layouts. These representations help stakeholders — including developers, administrators, and evaluators — to understand the logic behind system operations and ensure that the final implementation aligns with the goals defined during planning.

The HMS follows a modular architecture based on three primary user roles: Patients, Doctors, and Admins. Each role has a dedicated interaction flow, dashboard interface, and data access rules, which are clearly visualized and explained in the design elements that follow.

**3.1 Entity–Relationship (ER) Diagram**

The Entity–Relationship Diagram is a visual representation of the major entities in the system and the relationships between them. It is essential for designing the database schema and ensuring logical connectivity between various data components.

In the HMS, the key entities are: **Patient**, **Doctor**, **Admin**, and **Appointment**.



**Entities and Attributes:**

1. **Patient**
   * Attributes: patient\_id (PK), name, email, age, gender, contact\_number
   * Description: Represents any individual user who registers and books appointments.
2. **Doctor**
   * Attributes: doctor\_id (PK), name, department, email, availability\_status
   * Description: Medical professionals who are assigned to handle appointments and view patient schedules.
3. **Admin**
   * Attributes: admin\_id (PK), name, email, role
   * Description: A superuser who oversees the platform’s functionality and manages doctors, patients, and departments.
4. **Appointment**
   * Attributes: appointment\_id (PK), patient\_id (FK), doctor\_id (FK), department, date, time, status
   * Description: This is a transactional entity that stores the interaction between patients and doctors.

**Relationships:**

* **One-to-Many**:
  + One patient can have multiple appointments
  + One doctor can attend to multiple appointments
  + Each appointment links exactly one doctor to one patient
* **Many-to-One**:
  + Many appointments are linked to a single department.

**ER Diagram Explanation**

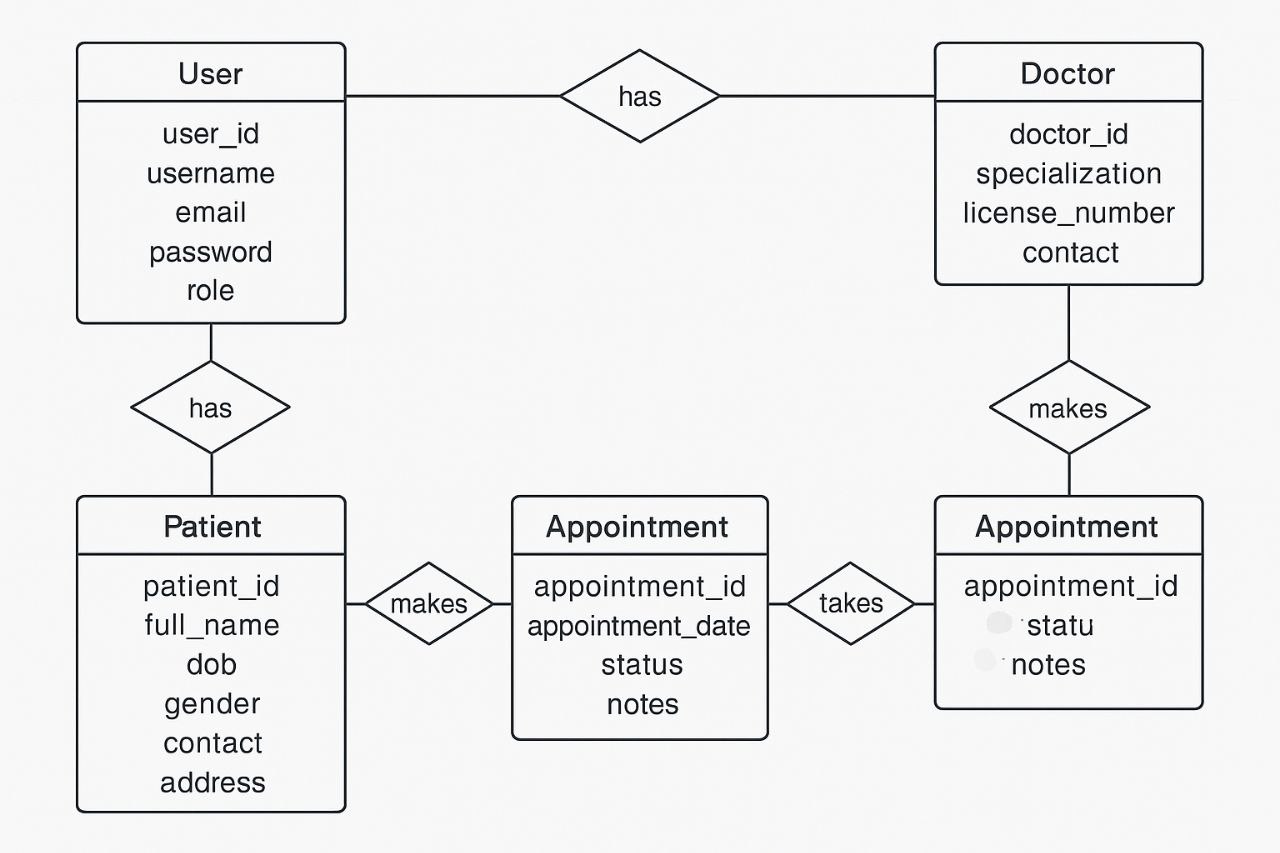
The ER Diagram simplifies the backend structure of the HMS by logically organizing how data is stored, accessed, and connected.

Whenever a patient books an appointment, the system:

* Captures the patient’s UID (from Firebase Authentication)
* Links it to the selected doctor UID
* Stores the data with a unique appointment\_id
* All of this is recorded in the **appointments** collection in Firebase

Doctors can log in and access only the appointments that reference their UID. Similarly, patients can only view appointments tied to their UID. This access control is designed using Firebase rules, ensuring data isolation and privacy.

The use of an ER Diagram helps prevent data redundancy, ensures referential integrity, and enables efficient data retrieval and updates during system operation.

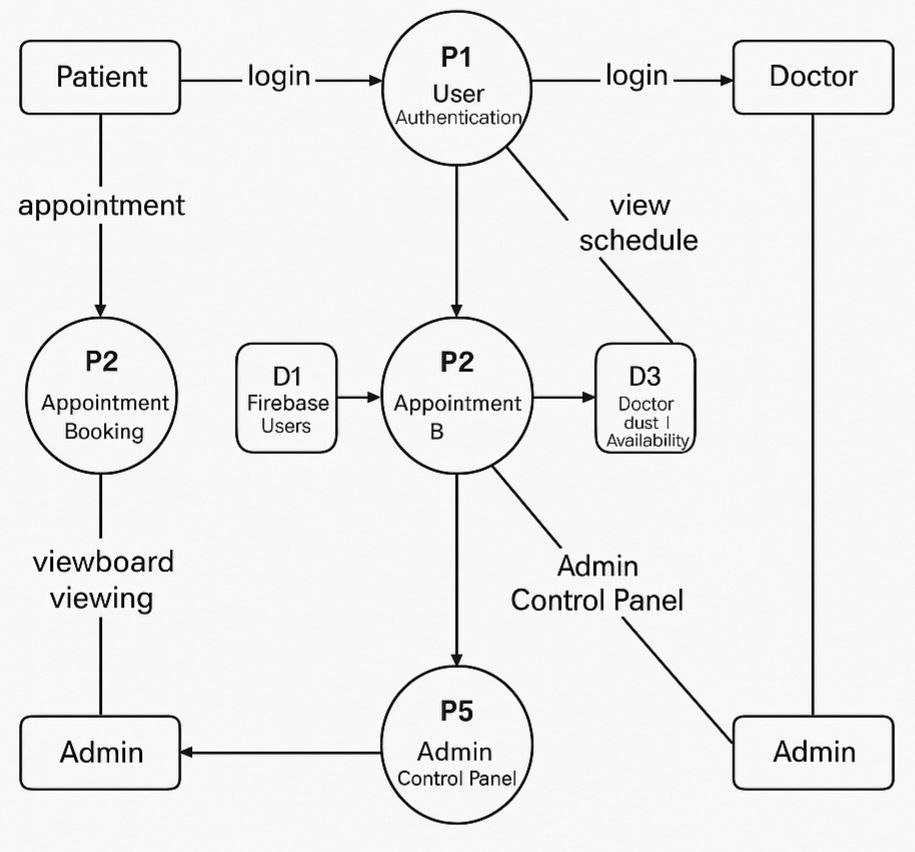


**3.2 Data Flow Diagram (DFD) – Level 0**

A Data Flow Diagram (DFD) is a visual representation of how data moves through a system — from external entities to internal processes and data stores. In this Hospital Management System (HMS), the Level 0 DFD gives a high-level overview of the entire system, showing how the main users (patients, doctors, admins) interact with various processes like login, booking, and viewing appointments.

This diagram helps developers and stakeholders understand:

* The core modules of the system
* The flow of information between users and the backend
* How external inputs are transformed into stored or displayed outputs



**Key Components of DFD Level 0**

**External Entities:**

* **Patient** – Registers, logs in, books appointments
* **Doctor** – Logs in, views scheduled appointments
* **Admin** – Manages users and departments

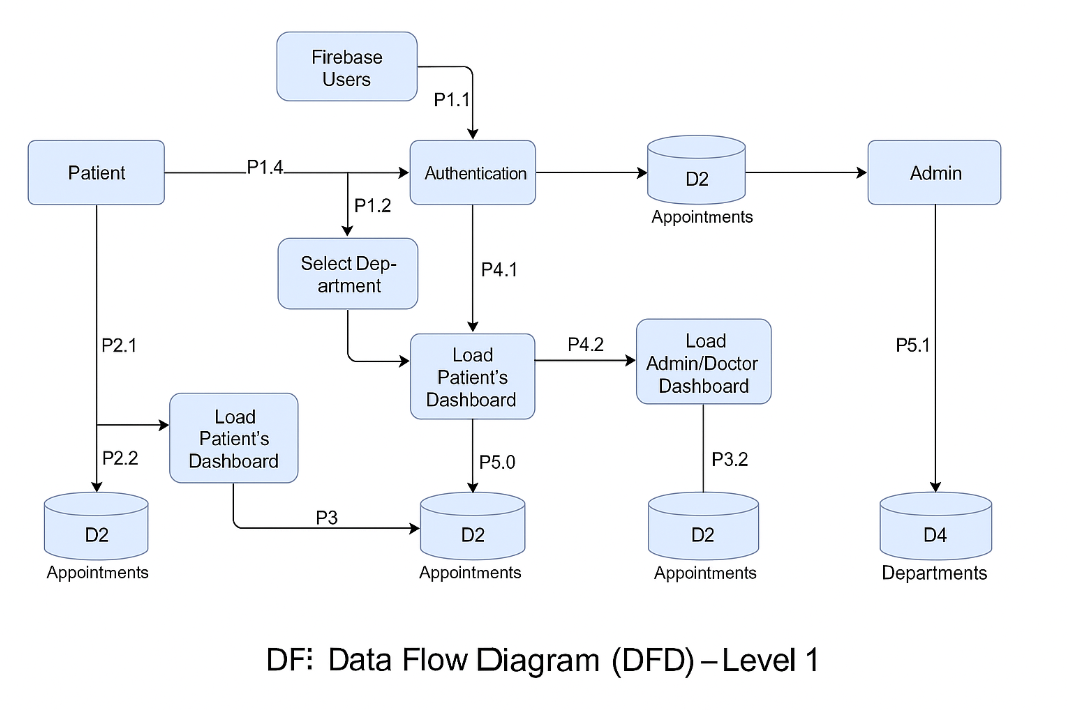
**Processes:**

* P1 – User Authentication
* P2 – Appointment Booking
* P3 – Schedule Management
* P4 – Dashboard Viewing
* P5 – Admin Control Panel

**Data Stores:**

* D1 – Firebase Users
* D2 – Appointments
* D3 – Doctor List / Availability
* D4 – Departments

**3.2 Data Flow Diagram (DFD) – Level 1**



The Level 1 Data Flow Diagram provides a more detailed look into how individual processes operate within the HMS and how data moves between them. It breaks down the high-level processes from DFD Level 0 into more specific sub-processes.

**Detailed Process Breakdown:**

1. **P1.1 – Login Authentication**
   * The user (patient, doctor, or admin) submits credentials.
   * Firebase Authentication checks the data against existing records in the Users data store.
   * Access is granted based on role.
2. **P2.1 – Book Appointment**
   * Patient selects department, doctor, date, and time.
   * Form is validated and data is saved in the Appointments collection.
   * System checks doctor availability in real time.
3. **P3.1 – View Dashboard Data**
   * Based on user role, relevant data is fetched from Firestore.
   * Patients see appointment history; doctors see upcoming appointments.
4. **P4.1 – Admin Panel Actions**
   * Admin can add or delete doctors, manage departments, or view all users.
   * Changes are updated live in the database.

**Data Stores:**

* D1 – Users (Patients, Doctors, Admins)
* D2 – Appointments
* D3 – Department List
* D4 – Doctor Availability

**3.3 Input and Output Design**

**Introduction:**

Input and output design is a crucial part of system development, especially in a user-centric platform like the Hospital Management System (HMS). The accuracy and efficiency of any system depend heavily on how clearly data is captured from the user and how effectively it is presented as output.

Input design focuses on how data is entered into the system. This includes ensuring that users can easily provide accurate and complete information through well-structured forms, dropdowns, and validations. Output design, on the other hand, is responsible for displaying the processed data in a clear, meaningful, and usable format — such as dashboards, confirmation messages, or reports.

In the HMS, each role — Patient, Doctor, and Admin — interacts with the system through a set of input and output components specifically designed to match their tasks and permissions.

**Input Design in HMS:**

The input interface is developed using HTML forms, styled with CSS, and validated using JavaScript to prevent incorrect or incomplete entries.

| **User Role** | **Form/Page** | **Fields** | **Validations Applied** |
| --- | --- | --- | --- |
| Patient | Signup | Name, Email, Password, Age, Gender | Required fields, valid email, password length |
| Patient | Login | Email, Password | Required fields |
| Patient | Book Appointment | Department, Doctor, Date, Time | Required, Date/Time logic |
| Doctor/Admin | Login | Email, Password | Required, Role check |
| Admin | Add Doctor / Department | Name, Department, Email, Availability Status | Format checks, dropdown controls |

All input is sent securely to Firebase using JavaScript, with real-time form updates, error messages, and field highlights for user guidance.

**Output Design in HMS:**

Output is rendered dynamically on the user’s dashboard after data is fetched from **Firebase**. Output content varies based on the user’s role:

| **User Role** | **Output Section** | **Displayed Content** |
| --- | --- | --- |
| Patient | Dashboard | Personal info, Appointment history, Booking confirmations |
| Doctor | Dashboard | List of scheduled appointments with patient names, dates, and times |
| Admin | Control Panel | Registered doctors, departments, appointments overview |
| All Users | Alert Boxes / Notifications | Login success, validation errors, appointment status |

Each dashboard page is developed to display real-time information pulled from the Firebase database and is structured using cards, tables, and coloured UI blocks for clarity.

**Summary:**

System design is a crucial and foundational phase in the software development life cycle (SDLC), serving as the bridge that transforms abstract user requirements and real-world healthcare challenges into a clearly defined technical blueprint that guides the development and implementation process. In the context of the Hospital Management System (HMS), the system design phase plays a vital role in defining the overall structure, functionality, data flow, component interactions, and user interfaces of the application, ensuring that all business logic, user needs, and technical specifications are accurately captured and represented.

This section is dedicated to presenting a comprehensive and structured visualization of the internal architecture and operational logic of the HMS. It utilizes widely accepted industry-standard design tools such as Entity–Relationship (ER) Diagrams, Data Flow Diagrams (DFDs) at multiple levels, and Input/Output Layouts, all of which collectively depict how various components of the system interact with one another, how data is created, stored, and retrieved, and how users engage with the platform through their respective interfaces. These design models not only serve as a communication tool between the development team and stakeholders, such as college project evaluators, mentors, or hospital IT teams, but also help verify that the final implementation will remain aligned with the original functional requirements and usability goals set during the planning phase.

The HMS is developed using a modular design architecture, which means the system is logically divided into self-contained units or modules, each responsible for a specific functionality. These modules include, but are not limited to, patient registration and login, doctor profile and availability management, appointment booking system, admin control panel, and dashboard interfaces. The modular approach enhances maintainability, scalability, and clarity in both design and implementation. The platform supports three major user roles — Patient, Doctor, and Admin — and each of these roles is associated with a clearly defined set of tasks, data access rights, user flows, and interface elements. These role-specific modules ensure that patients can seamlessly book appointments and view records, doctors can manage their consultation schedules and review patient data, and administrators can oversee the entire hospital workflow and system operations from a central control interface.

The visual and logical representations presented in this section will help the reader understand the core design philosophy, the flow of information, and the interactive structure that powers the HMS. By studying the ER diagrams, DFDs (Level 0 and 1), and input/output designs, one gains a thorough understanding of how the project was conceptualized, structured, and prepared for development. This detailed system design not only supports the technical development phase but also acts as documentation for future upgrades or integrations, such as billing modules, pharmacy systems, or analytics dashboards.

4. System Configuration

System configuration refers to the hardware, software, and platform setup required for the successful development, testing, deployment, and operation of the Hospital Management System (HMS). Since the HMS is a cloud-based web application, its configuration is lightweight and highly scalable. The system is designed to run smoothly on commonly available computer hardware, internet connectivity, and freely available software tools.

This section outlines the minimum system requirements necessary for:

* Developing the HMS platform
* Hosting and deploying the application
* Accessing and using the system by end users (patients, doctors, and admins)

The goal is to ensure that any educational institution, hospital, or clinic can easily adopt this project without the need for expensive or high-performance hardware.

**4.1 Hardware Requirements**

The hardware required for developing and running the HMS application is minimal. Since the entire system runs on web browsers and is hosted online, it does not need advanced processing power or expensive infrastructure.

| **Component** | **Minimum Requirement** |
| --- | --- |
| Processor | Intel Core i3 or equivalent (or higher) |
| RAM | Minimum 4 GB |
| Storage | At least 250 GB HDD or 128 GB SSD |
| Display | 14-inch monitor or laptop screen |
| Internet Connection | Stable broadband or mobile hotspot (4G+) |
| Peripheral Devices | Keyboard, Mouse, Webcam (optional) |

This configuration is suitable for both development and usage. Patients and doctors can access the system from any device that supports modern web browsers, including mobile phones and tablets.

**4.2 Software Requirements**

The software requirements for the Hospital Management System (HMS) project are selected based on ease of use, open-source availability, and compatibility with a wide range of systems. All tools used in this project are either free to use or come with community-supported versions, making the system affordable and accessible for hospitals, students, and developers alike.

The system is browser-based and uses web development technologies combined with Firebase as the backend. Hence, the setup does not demand installation of heavy IDEs or paid enterprise solutions. Below is the detailed list of software requirements:

**a) Operating System**

| **Requirement** | **Description** |
| --- | --- |
| OS | Windows 10/11, macOS, or any modern Linux distro |
| Note | Development was done on Windows, but platform-independent |

**b) Development Tools**

| **Software** | **Purpose** |
| --- | --- |
| **Visual Studio Code** | Code editor for HTML, CSS, JavaScript |
|  |  |
| **Git & GitHub** | Version control and collaborative development |
| **Firebase Console** | Backend setup, database access, and deployment |
| **Web Browser** | Chrome, Firefox, or Edge for testing UI |
| **Postman** (optional) | API testing tool if extensions were added |

**c) Programming Languages & Frameworks**

| **Language** | **Role** |
| --- | --- |
| **HTML5** | Structure of the web pages |
| **CSS3** | Styling and responsiveness |
| **JavaScript** | Logic and Firebase integration |
| **Firebase JS SDK** | Backend interaction & authentication |

**d) Backend & Hosting**

| **Component** | **Service Used** |
| --- | --- |
| **Authentication** | Firebase Authentication |
| **Database** | Firebase Firestore |
| **Hosting** | Firebase Hosting |
| **Cloud Functions** | Optional – for future features |

The selection of Firebase eliminates the need for traditional server-side scripting, SQL database management, or infrastructure provisioning. This makes the system lightweight, secure, and fast to deploy.

**4.3 Platform Details**

The Hospital Management System (HMS) is built using a cloud-native platform architecture, meaning it is developed, deployed, and accessed entirely via the internet. This design approach offers several advantages such as scalability, cost-efficiency, easy maintenance, and real-time access to data. The platform chosen for this project is the Firebase suite by Google, integrated with standard web technologies such as HTML, CSS, and JavaScript.

This section explains the reasons behind selecting Firebase as the core platform and describes how each service contributes to the functionality and reliability of the system.

**Why Firebase?**

Firebase is a Backend-as-a-Service (BaaS) platform that offers a range of services including real-time databases, authentication, hosting, cloud functions, and analytics. For a project like HMS, Firebase is an ideal choice because it removes the need to set up and maintain a traditional backend server or SQL database.

Firebase offers:

* **Secure user authentication** using email and password
* **Real-time Firestore database** for storing patient, doctor, and appointment records
* **Fast and reliable hosting** with built-in SSL encryption
* **Scalability** for future growth (e.g., SMS alerts, analytics, cloud functions)
* **Easy-to-use dashboard** for managing all backend components in one place

**Platform Architecture Summary**

| **Layer** | **Technology Used** | **Description** |
| --- | --- | --- |
| Frontend | HTML, CSS, JavaScript | User interface, form validation, and event handling |
| Backend (Auth) | Firebase Authentication | User signup/login and role-based access |
| Backend (Data) | Firebase Cloud Firestore | NoSQL database for all user and appointment data |
| Hosting | Firebase Hosting | Deployed to the web via Firebase CLI and GitHub |
| Storage (Optional) | Firebase Cloud Storage | For storing images or medical reports (future scope) |

**Hosting and Deployment**

The HMS application is hosted on Firebase Hosting, which provides:

* **Global content delivery** using a CDN (Content Delivery Network)
* **HTTPS by default**, ensuring secure access for all users
* **Version control and rollback options**
* One-click deploy using the Firebase CLI or GitHub Actions

This makes it suitable not only for student-level academic projects but also for real-world hospital implementations, especially for small- to mid-size clinics.

1. Details of Software

This section gives a comprehensive overview of the software components used in developing the Hospital Management System (HMS). It is divided into two major areas:

1. Frontend Technologies – responsible for creating the user interface
2. Backend Services – responsible for handling authentication, data storage, and deployment

Together, these components form a complete, lightweight, scalable system that is easy to use, quick to develop, and capable of being deployed for real-world applications.

**5.1 Overview of frontend**

The frontend is the part of the system that users directly interact with — including patients booking appointments, doctors viewing schedules, and admins managing users. For HMS, the frontend is built using:

* **HTML** for structure
* **CSS** for styling and layout
* **JavaScript** for logic and interactivity

Let’s explore each technology in detail.

**a) HTML (HyperText Markup Language)**

HTML is used to create the **structure and layout** of all web pages. Every page in the HMS — login forms, dashboards, appointment forms, and contact pages — is written in HTML. Elements like input fields, buttons, tables, and labels are all defined here.

**b) CSS (Cascading Style Sheets)**

CSS defines how the HTML content looks on the screen. It controls font styles, colours, spacing, layout, responsiveness, and animations. In HMS, CSS is used to create a professional and consistent look for all pages — including forms, dashboards, tables, and navigation bars.

**c) JavaScript**

JavaScript is used to make the web pages interactive and dynamic. It handles:

* Input validation
* Event listeners (e.g., button clicks)
* Firebase integration (Auth, data fetch/store)
* Real-time updates on the dashboard

For example, when a patient submits an appointment form, JavaScript first checks that all fields are filled, then sends the data to Firebase.

**Authentication**

**sign-up:**

createUserWithEmailAndPassword(auth, email, password)

.then((userCredential) => {

const user = userCredential.user;

const userData = {

email,

firstName,

address,

phoneNumber,

gender: Gender,

dateOfBirth

};

const docRef = doc(db, "users", user.uid);

setDoc(docRef, userData)

.then(() => {

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

})

.catch((error) => {

console.error("Error writing document", error);

});

})

.catch((error) => {

if (error.code === 'auth/email-already-in-use') {

showMessage('Email address already exists!', 'signUpMessage');

} else {

showMessage('Unable to create user', 'signUpMessage');

}});

**sign-in:**

signInWithEmailAndPassword(auth, email, password)

.then(async (userCredential) => {

const user = userCredential.user;

const userRef = doc(db, "users", user.uid);

const userDoc = await getDoc(userRef);

if (userDoc.exists()) {

localStorage.setItem('userData', JSON.stringify(userDoc.data()));

localStorage.setItem('userUID', user.uid);

window.location.replace('home.html');

} else {

showMessage('No user data found in Firestore.', 'signInMessage');

}

})

.catch((error) => {

showMessage('Login failed. Please try again.', 'signInMessage');

});

**Appointment:**

const appointmentData = {

patientName: document.getElementById('patientName').value,

phoneNumber: document.getElementById('phoneNumber').value,

gender: document.getElementById('gender').value,

age: document.getElementById('age').value,

department: document.getElementById('department').value,

doctorUID,

doctorName,

date: appointmentDate,

tokenNumber,

timestamp: firebase.firestore.FieldValue.serverTimestamp(),

userId

};

await db.collection('appointments').add(appointmentData);

**doctor-login:**

const userSnap = await getDoc(doc(db, "doctors", user.uid));

if (userSnap.exists() && userSnap.data().role === "doctor") {

localStorage.setItem("doctorUID", user.uid);

window.location.href = "doctor-appointments.html";

} else {

document.getElementById("loginMsg").textContent = "Unauthorized access.";

}

**Admin dashboard:**

const userSnap = await getDoc(doc(db, "doctors", user.uid));

if (userSnap.exists() && userSnap.data().role === "doctor") {

localStorage.setItem("doctorUID", user.uid);

window.location.href = "doctor-appointments.html";

} else {

document.getElementById("loginMsg").textContent = "Unauthorized access.";

}

**Home-page:**

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8" />

  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>

  <title>Infinity Care Hospital</title>

  <link href="style.css" rel="stylesheet" />

  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet"/>

  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/odometer.js/0.4.7/themes/odometer-theme-default.css" crossorigin="anonymous" />

</head>

<body>

  <!-- Header -->

  <header>

    <div class="container-fluid p-3 d-flex justify-content-between align-items-center">

      <a class="navbar-brand d-flex align-items-center" href="#">

        <div class="image">

          <img src="/images/logo.png" alt="Logo" width="100" />

        </div>

        <div class="logo-text">

          <img src="/images/Logotext.png" alt="Logo" width="180" />

        </div>

      </a>

      <div class="d-flex gap-4 align-items-center">

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

          <img src="/images/Call.png" alt="Phone Icon" width="20" class="me-2" />

          <div>

            <small>EMERGENCY</small><br />

            <a href="tel:+919876543210">+91-9876543210</a>

          </div>

        </div>

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

          <img src="/images/Email.png" alt="Email Icon" width="20" class="me-3" />

          <div>

            <small>EMAIL</small><br />

            <a href="mailto:info@infinitycarehospitals.com">info@infinitycarehospitals.com</a>

          </div>

        </div>

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

          <img src="/images/Location.png" alt="Location Icon" width="20" class="me-4" />

          <div>

            <small>LOCATION</small><br />

            Bangalore

          </div>

        </div>

      </div>

    </div>

    <!-- Navbar -->

    <nav class="navbar navbar-expand-lg" style="background-color: #1c2a69;">

      <div class="container-fluid d-flex justify-content-center">

        <ul class="navbar-nav mb-2 mb-lg-0">

          <li class="nav-item"><a class="nav-link text-white" href="about.html">About Us</a></li>

          <li class="nav-item"><a class="nav-link text-white" href="department.html">Departments</a></li>

          <li class="nav-item"><a class="nav-link text-white" href="doctor.html">Doctor</a></li>

          <li class="nav-item"><a class="nav-link text-white" href="contact.html">Contact</a></li>

          <li class="nav-item"><a class="nav-link text-white" href="patient.html">Record</a></li>

          <li class="nav-item"><a class="nav-link text-white" href="profile.html">Profile</a></li>

        </ul>

      </div>

        <ul class="navbar-nav mb-2 mb-lg-0 d-flex gap-4">

          <!-- Dropdown for Login/Sign Up -->

          <!-- Login/Signup Dropdown -->

<li class="nav-item dropdown" id="login-dropdown">

  <a class="nav-link dropdown-toggle text-white" href="#" id="loginDropdown" role="button" data-bs-toggle="dropdown" aria-expanded="false">

    Login/signup

  </a>

  <ul class="dropdown-menu dropdown-menu-end" aria-labelledby="loginDropdown">

    <li><a class="dropdown-item" href="signup.html">User Login</a></li>

    <li><a class="dropdown-item" href="doctor-login.html">Doctor Login</a></li>

    <li><a class="dropdown-item" href="adminLogin.html">Admin Login</a> </li>

  </ul>

</li>

<!-- Placeholder for Logout Button -->

<li class="nav-item d-none" id="logout-nav-item">

  <a class="nav-link text-white" href="#" id="logout-btn">Logout</a>

</li>

          <!-- User/Doctor logged-in info (will be displayed dynamically based on auth) -->

        </ul>

      </div>

    </nav>

  </header>

  <!-- Hero Section -->

  <div class="hero-container">

    <div class="hero-image-slider" id="heroSlider">/

      <div class="text">

        <p></p>

      </div>

    </div>

    <button class="appointment-button" onclick="window.location.href='appointment.html'">Appointment</button>

  </div>

  <!-- Services -->

  <div class="service-container">

    <div class="service-card">

      <img src="/images/fertility.png" alt="Fertility Care" />

      <h3>Fertility Care</h3>

      <p>Comprehensive infertility treatments, including IVF and ICSI, to achieve parenthood.</p>

      <a href="department.html" class="read-more">Read More</a>

    </div>

    <div class="service-card">

      <img src="/images/Obstetrics.png" alt="Obstetrics & Gynaecology" />

      <h3>Obstetrics & Gynaecology</h3>

      <p>Expert care in pregnancy, childbirth, and women's reproductive health.</p>

      <a href="department.html" class="read-more">Read More</a>

    </div>

    <div class="service-card">

      <img src="/images/Gynec.png" alt="Gynec Laparoscopy" />

      <h3>Gynec Laparoscopy</h3>

      <p>Minimally invasive surgical solutions for faster recovery and less pain.</p>

      <a href="department.html" class="read-more">Read More</a>

    </div>

  </div>

  <!-- Team -->

  <div class="team-section text-center">

    <h2>Our Experienced Team</h2>

    <div class="team-container">

        <div class="team-member">

            <img src="/images/Dr.1.jpg" alt="Dr. ABC">

            <p>Dr.Rajesh</p>

        </div>

        <div class="team-member">

            <img src="/images/Dr.2.jpg" alt="Dr. XYZ">

            <p>Dr.Ryan Thompson</p>

        </div>

        <div class="team-member">

            <img src="/images/Dr.3.jpg" alt="Dr. DEF">

            <p>Dr.logan John</p>

        </div>

        <div class="team-member">

            <img src="/images/Dr.4.jpg" alt="Dr. GHI">

            <p>Dr.Jackson lee</p>

        </div>

    </div>

    <a href="doctor.html" class="view-more-btn">View More</a>

</div>

  <!-- Stats -->

  <div class="stats-section">

    <div class="stats-container">

      <div class="stats-card">

        <img src="/images/bed.png" alt="Bed Icon" class="stats-icon" /><br />

        <div class="bed-odometer odometer plus">0</div>

        <p>Beds</p>

      </div>

      <div class="stats-card">

        <img src="/images/experience.png" alt="Experience Icon" class="stats-icon" /><br />

        <div class="experience-odometer odometer plus">0</div>

        <p>Years of Experience</p>

      </div>

      <div class="stats-card">

        <img src="/images/happy.png" alt="Happy Patients" class="stats-icon" /><br />

        <div class="happy-odometer odometer plus">0</div>

        <p>Happy Patients</p>

      </div>

    </div>

  </div>

  <!-- Insurance -->

  <div class="team-section text-center">

    <h2>Our Insurances</h2>

    <div class="insurance-container">

      <img src="/images/SBI.png" alt="SBI" />

      <img src="/images/hdfc.png" alt="HDFC" />

      <img src="/images/federal.png" alt="Federal" />

      <img src="/images/SIB.png" alt="SIB" />

    </div>

  </div>

  <!-- Footer -->

  <footer class="footer">

    <div class="footer-container">

      <div class="footer-logo">

        <img src="/images/logo.png" alt="Infinity Care Logo" class="logo-img" />

      </div>

      <div class="footer-contact">

        <h4>Contact Us</h4>

        <p>Infinity Care, 204 2nd Block, Kannuru, Bangalore</p>

        <p>Karnataka 560077</p>

        <p>📞 +91 4734 219500</p>

        <p>✉️ info@lifelinehospitalkerala.com</p>

      </div>

      <div class="footer-Shortcuts">

        <h5><a href="home.html">Home</a></h5>

        <h5><a href="about.html">About Us</a></h5>

        <h5><a href="department.html">Departments</a></h5>

        <h5><a href="doctor.html">Doctor</a></h5>

        <h5><a href="contact.html">Contact</a></h5>

        <h5><a href="patient.html">Record</a></h5>

      </div>

    </div>

    <div class="footer-copyright">

      &copy; 2020 Infinity Care Hospital. Developed by <a href="#">Codeface Technologies</a>.

    </div>

  </footer>

</body>

</html>

**Signup-page:**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Medical Signup Form</title>

</head>

<body>

    <div class="container">

        <div class="form-container">

            <h2>Give Your Details</h2>

            <form id="signupForm">

                <div id="signUpMessage" class="messageDiv" style="display: none;"></div>

                <div class="form-group">

                    <label>Name</label>

                    <input type="text" placeholder="Name" id="name" required>

                </div>

                <div class="form-group">

                    <label>Gender</label>

                    <select id="Gender" required>

                        <option value="">Select Gender</option>

                        <option value="Male">Male</option>

                        <option value="Female">Female</option>

                        <option value="Other">Other</option>

                    </select>

                </div>

                <div class="form-group">

                    <label>Address</label>

                    <input type="text" placeholder="Address" id="Address" required>

                </div>

                <div class="form-group">

                    <label>Date Of Birth</label>

                    <input type="date" placeholder="DD/MM/YYYY" id="DOB" required>

                </div>

                <div class="form-group">

                    <label for="phone">Phone</label>

                    <input type="tel" id="phone" name="phone" placeholder="Phone Number" pattern="[0-9]{10}" maxlength="10" required aria-required="true">

                </div>

                <div class="form-group">

                    <label>Email</label>

                    <input type="email" placeholder="Email" id="Email" required>

                </div>

                <div class="form-group">

                    <label>Password</label>

                    <input type="password" placeholder="Password" id="Password" required>

                </div>

                 <button type="button" class="btn" id="submitSignIn">Sign Up</button>

                 <div class="login-link">

                    Have an account? <a href="login.html">Log In</a>

                </div>

            </form>

        </div>

        <div class="image-container">

            <img src="/images/Nurse.jpg" alt="Doctor">

        </div>

    </div>

</body>

</html>

**Signin-page:**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Medical Login Form</title>

</head>

<body>

    <div class="container">

        <div class="form-container">

            <h2>Give Your Details</h2>

            <form id="loginForm">

                <div id="signInMessage" class="messageDiv" style="display: none;"></div>

                <div class="form-group">

                    <label for="Email">Email</label>

                    <input type="text" id="Email" placeholder="Email" required>

                </div>

                <div class="form-group">

                    <label for="Password">Password</label>

                    <input type="password" id="Password" placeholder="Password" required>

                </div>

                <button type="button" class="btn" id="submitSignIn">Log In</button>

                <div class="login-link">

                    Don't have an account? <a href="signup.html">Signup</a>

                </div>

            </form>

        </div>

        <div class="image-container">

            <img src="/images/Doctorr.png" alt="Doctor">

        </div>

    </div>

</body>

<script type="module" src="firebaseauth.js"></script>

</html>

**Profile-page:**

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>User Profile</title>

</head>

<body>

  <a href="home.html" class="back-button">← Back</a>

  <h2>User Profile</h2>

  <div class="profile-card">

    <div class="profile-header">

      <h3 id="profile-name">Name</h3>

      <p id="profile-email">example@gmail.com</p>

    </div>

    <div class="info-section">

      <div class="info-card">

        <p><span>Age:</span> <span id="profile-dob"></span></p>

        <p><span>Gender:</span> <span id="profile-gender"></span></p>

      </div>

      <div class="info-card">

        <p><span>Phone:</span> <span id="profile-phone"></span></p>

        <p><span>Address:</span> <span id="profile-address"></span></p>

      </div>

    </div>

  </div>

  <a href="edit-profile.html" style="display:block;text-align:center;margin-top:20px;color:#007bff;">Edit Profile</a>

</body>

</html>

**Appointment booking-page:**

<!DOCTYPE html>

<html>

<head>

  <meta charset="UTF-8">

  <title>Book Appointment</title>

</head>

<body>

<div class="appointment-form-container">

  <h2>Book an Appointment</h2>

  <form id="appointmentForm" class="appointment-form">

    <div class="form-group">

      <label for="patientName">Patient Name</label>

      <input type="text" id="patientName" required>

    </div>

    <div class="form-group">

      <label for="phoneNumber">Phone Number</label>

       <input type="tel" id="phoneNumber" name="phone" placeholder="Phone" pattern="[0-9]{10}" maxlength="10" required aria-required="true">

    </div>

    <div class="form-group">

      <label for="gender">Gender</label>

      <select id="gender" required>

        <option value="">Select Gender</option>

        <option value="Male">Male</option>

        <option value="Female">Female</option>

        <option value="Other">Other</option>

      </select>

    </div>

    <div class="form-group">

      <label for="age">Age</label>

      <input type="number" id="age" required min="0">

    </div>

    <div class="form-group">

      <label for="department">Department</label>

      <select id="department" required onchange="updateDoctors()">

        <option value="">Select Department</option>

      </select>

    </div>

    <div class="form-group">

      <label for="doctor">Doctor Name</label>

      <select id="doctor" required onchange="displayConsultingTime()">

        <option value="">Select Doctor</option>

      </select>

    </div>

    <div class="form-group">

      <label for="consultingTime">Consulting Time</label>

      <input type="text" id="consultingTime" readonly>

    </div>

    <div class="form-group">

      <label for="date">Select Date</label>

      <input type="date" id="date" required>

    </div>

    <button type="submit" class="submit-btn">Submit Appointment</button>

    <div class="messageDiv" id="messageDiv"></div>

    <div class="errorDiv" id="errorDiv"></div>

  </form>

</div>

</body>

</html>

**5.2 Overview of backend – Firebase**

The backend of the Hospital Management System (HMS) is fully developed using Firebase, a cloud-based Backend-as-a-Service (BaaS) platform by Google. Firebase allows seamless integration between frontend JavaScript code and backend services without the need to manage physical servers or write server-side code.

Firebase provides essential services like:

* **Authentication** for secure login/signup
* **Firestore** for real-time database management
* **Hosting** for deploying the web application
* (Optional) **Cloud Functions** for advanced features

Let’s explore the core services used in the HMS project:

**a) Firebase Authentication**

Firebase Authentication handles user identity verification and allows secure sign-up and login for different roles (Admin, Doctor, Patient).

**Key Features:**

* Supports email and password login
* Firebase handles password hashing and session tokens securely
* Integrates directly with Firestore for role-based access

**b) Firebase Firestore (Database)**

Firestore is a scalable, NoSQL cloud database used to store structured data in collections and documents. In HMS, Firestore is used for storing:

* Patient records
* Doctor details
* Appointment history
* Department data

**Collections in HMS:**

* users – Stores details for patients, doctors, and admins
* appointments – Contains all booked appointment data
* doctors – Details about each doctor and availability
* departments – List of hospital departments

**Real-Time Updates:**

Firestore allows real-time data sync across all devices. For example, when a patient books an appointment, the doctor’s dashboard updates instantly.

**c) Firebase Hosting**

Once the project is developed, it is deployed using **Firebase Hosting**, which:

* Serves the web app over HTTPS (secure)
* Supports fast global delivery via CDN
* Integrates directly with GitHub for deployment automation

**5.3 About the Platform**

In software development, a platform refers to the environment in which an application is developed, executed, tested, and deployed. For the Hospital Management System (HMS), the platform includes the tools, technologies, operating systems, and cloud services that work together to support the smooth functioning of the application.

The HMS platform is a cloud-based, browser-accessible system that can run on any device with a modern web browser and internet access. It is built using frontend web technologies (HTML, CSS, JavaScript) and Firebase as the backend cloud service, which provides real-time data management, user authentication, and secure hosting.

**Key Characteristics of the HMS Platform**

1. **Web-Based Architecture:**  
   HMS is a browser-based system that does not require installation. All users can access the platform from devices such as desktops, laptops, tablets, or smartphones.
2. **Cloud Integration (Firebase):**  
   Firebase provides backend services including Authentication, Firestore Database, and Hosting. These services are integrated with the JavaScript frontend, creating a fully serverless cloud platform.
3. **Cross-Platform Accessibility:**  
   The system is platform-independent and runs on all major operating systems — Windows, macOS, Linux — since it is accessed through the browser.
4. **Real-Time Operations:**  
   The Firestore database allows real-time updates to be reflected instantly on all connected clients. For example, when a patient books an appointment, it immediately appears on the doctor's dashboard.
5. **Security and Scalability:**  
   The platform uses Firebase Authentication to securely manage user access. It also supports scaling to thousands of users without major structural changes.

**Development & Deployment Environment**

| **Component** | **Platform Used** |
| --- | --- |
| Development OS | Windows 10 |
| Development IDE | Visual Studio Code |
| Browser | Google Chrome |
| Backend Platform | Firebase (Firestore + Auth) |
| Deployment Platform | Firebase Hosting |
| Version Control | Git & GitHub |

**Advantages of the Chosen Platform**

* **Low Cost**: Firebase offers a generous free tier, making it perfect for academic and startup projects.
* **Ease of Use**: Visual Studio Code and Firebase tools are beginner-friendly and well-documented.
* **Rapid Deployment**: Firebase CLI enables quick hosting with a single command.
* **No Server Management**: As a serverless platform, Firebase handles uptime, scalability, and performance optimization.
* **Secure Access**: HTTPS hosting, Firestore rules, and role-based access ensure privacy and protection of sensitive data.

**Firebase**

**Firebase authentication access:**

// Import necessary Firebase modules

import { initializeApp } from "https://www.gstatic.com/firebasejs/11.6.0/firebase-app.js";

import { getAuth, createUserWithEmailAndPassword, signInWithEmailAndPassword } from "https://www.gstatic.com/firebasejs/11.6.0/firebase-auth.js";

import { getFirestore, setDoc, doc, getDoc } from "https://www.gstatic.com/firebasejs/11.6.0/firebase-firestore.js";

// Initialize Firebase

const app = initializeApp(firebaseConfig);

const auth = getAuth();

const db = getFirestore();

// Function to display messages

function showMessage(message, divId) {

  const messageDiv = document.getElementById(divId);

  if (messageDiv) {

    messageDiv.style.display = "block";

    messageDiv.innerHTML = message;

    messageDiv.style.opacity = 1;

    setTimeout(() => {

      messageDiv.style.opacity = 0;

    }, 5000);

  }

}

// Wait for DOM content to fully load

window.addEventListener('DOMContentLoaded', function () {

    console.log("DOM fully loaded and parsed");

    // SIGN UP HANDLER

    const signUpButton = document.getElementById('submitSignIn');

    if (signUpButton) {

      signUpButton.addEventListener('click', (event) => {

        event.preventDefault();

        const firstName = document.getElementById('name')?.value;

        const Gender = document.getElementById('Gender')?.value;

        const email = document.getElementById('Email')?.value;

        const address = document.getElementById('Address')?.value;

        const phoneNumber = document.getElementById('phone')?.value;

        const dateOfBirth = document.getElementById('DOB')?.value;

        const password = document.getElementById('Password')?.value;

        if (!email || !password || !firstName || !Gender || !dateOfBirth) {

          showMessage('Please fill in all fields.', 'signUpMessage');

          return;

        }

        createUserWithEmailAndPassword(auth, email, password)

          .then((userCredential) => {

            const user = userCredential.user;

            const userData = {

              email,

              firstName,

              address,

              phoneNumber,

              gender: Gender,

              dateOfBirth

            };

            showMessage('Account Created Successfully', 'signUpMessage');

            const docRef = doc(db, "users", user.uid);

            setDoc(docRef, userData)

              .then(() => {

                window.location.href = 'login.html'; // Redirect to login page

              })

              .catch((error) => {

                console.error("Error writing document", error);

              });

          })

          .catch((error) => {

            const errorCode = error.code;

            if (errorCode === 'auth/email-already-in-use') {

              showMessage('Email address already exists!', 'signUpMessage');

            } else {

              showMessage('Unable to create user', 'signUpMessage');

            }

          });

      });

    }

    // SIGN IN HANDLER

    const signInButton = document.getElementById('submitSignIn');

    if (signInButton) {

      signInButton.addEventListener('click', (event) => {

        event.preventDefault();

        const email = document.getElementById('Email')?.value;

        const password = document.getElementById('Password')?.value;

        if (!email || !password) {

          showMessage('Please enter both email and password.', 'signInMessage');

          return;

        }

        signInWithEmailAndPassword(auth, email, password)

          .then(async (userCredential) => {

            const user = userCredential.user;

            showMessage('Login successful!', 'signInMessage');

            // Fetch user data

            const userRef = doc(db, "users", user.uid);

            const userDoc = await getDoc(userRef);

            if (userDoc.exists()) {

              const userData = userDoc.data();

              localStorage.setItem('userData', JSON.stringify(userData));

              localStorage.setItem('userUID', user.uid);

              // Redirect to home page after login

              window.location.replace('home.html');

            } else {

              console.log("No user data found for UID:", user.uid);

              showMessage('No user data found in Firestore.', 'signInMessage');

            }

          })

          .catch((error) => {

            const errorCode = error.code;

            if (errorCode === 'auth/invalid-credential') {

              showMessage('Incorrect email or password.', 'signInMessage');

            } else {

              showMessage('Login failed. Please try again.', 'signInMessage');

            }

          });

      });

    }

});

**Admin**

**Authentication access:**

<script>

    const adminCredentials = {

      username: "admin123",  // Static Admin ID

      password: "adminpass"  // Static Admin Password

    };

    // Function to handle admin login

    function handleAdminLogin() {

      const username = document.getElementById("admin-username").value;

      const password = document.getElementById("admin-password").value;

      if (username === adminCredentials.username && password === adminCredentials.password) {

        // Redirect to doctor signup page after successful login

        window.location.href = "adminDashboard.html";

      } else {

        alert("Incorrect admin credentials. Please try again.");

      }

    }

  </script>

**Dashboard:**

<script type="module">

    import { initializeApp } from "https://www.gstatic.com/firebasejs/11.6.0/firebase-app.js";

    import { getFirestore, collection, getDocs, doc, deleteDoc, updateDoc } from "https://www.gstatic.com/firebasejs/11.6.0/firebase-firestore.js";

    const app = initializeApp(firebaseConfig);

    const db = getFirestore(app);

    async function deleteDoctor(docId, rowElement) {

      if (confirm("Are you sure you want to delete this doctor?")) {

        try {

          await deleteDoc(doc(db, "doctors", docId));

          rowElement.remove();

          alert("Doctor deleted successfully.");

        } catch (error) {

          console.error("Error deleting doctor:", error);

          alert("Failed to delete doctor.");

        }

      }

    }

    async function getDoctorDetails() {

      const tableBody = document.getElementById("doctor-table-body");

      tableBody.innerHTML = "";

      try {

        const querySnapshot = await getDocs(collection(db, "doctors"));

        querySnapshot.forEach((docSnap) => {

          const doctor = docSnap.data();

          const uid = docSnap.id;

          const row = document.createElement("tr");

          row.innerHTML = `

            <td>${doctor.name}</td>

            <td>${doctor.email}</td>

            <td>${doctor.department}</td>

            <td>

              <select data-id="${uid}" class="status-select">

                <option value="On Duty" ${doctor.status === "On Duty" ? "selected" : ""}>On Duty</option>

                <option value="Off Duty" ${doctor.status === "Off Duty" ? "selected" : ""}>Off Duty</option>

              </select>

            </td>

            <td>

              ${doctor.status === "Off Duty" ? `

                <div>From: ${doctor.offDutyStart || "N/A"}<br>To: ${doctor.offDutyEnd || "N/A"}</div>

              ` : "N/A"}

            </td>

            <td>

              <input type="time" class="start-time" value="${doctor.consultingStart || ''}" />

              -

              <input type="time" class="end-time" value="${doctor.consultingEnd || ''}" />

              <button class="save-time-btn" data-id="${uid}">Save</button>

            </td>

            <td>${uid}</td>

            <td><button class="delete-btn" data-id="${uid}">Delete</button></td>

          `;

          // Delete doctor

          row.querySelector(".delete-btn").addEventListener("click", () => {

            deleteDoctor(uid, row);

          });

          // Status change handler

          row.querySelector(".status-select").addEventListener("change", async (e) => {

            const newStatus = e.target.value;

            const docId = e.target.getAttribute("data-id");

            const updates = { status: newStatus };

            if (newStatus === "Off Duty") {

              const startDate = prompt("Enter Off Duty Start Date (YYYY-MM-DD):");

              const endDate = prompt("Enter Off Duty End Date (YYYY-MM-DD):");

              if (!startDate || !endDate) {

                alert("Start and end dates are required.");

                return;

              }

              updates.offDutyStart = startDate;

              updates.offDutyEnd = endDate;

            } else {

              updates.offDutyStart = "";

              updates.offDutyEnd = "";

            }

            try {

              await updateDoc(doc(db, "doctors", docId), updates);

              alert("Doctor status updated.");

              getDoctorDetails(); // Refresh table

            } catch (error) {

              console.error("Error updating status:", error);

              alert("Failed to update doctor status.");

            }

          });

          // Consulting time save handler

          row.querySelector(".save-time-btn").addEventListener("click", async () => {

            const startTime = row.querySelector(".start-time").value;

            const endTime = row.querySelector(".end-time").value;

            if (!startTime || !endTime) {

              alert("Please enter both start and end times.");

              return;

            }

            try {

              await updateDoc(doc(db, "doctors", uid), {

                consultingStart: startTime,

                consultingEnd: endTime,

              });

              alert("Consulting time updated.");

            } catch (error) {

              console.error("Error updating consulting time:", error);

              alert("Failed to update consulting time.");

            }

          });

          tableBody.appendChild(row);

        });

      } catch (error) {

        console.error("Error getting doctor details:", error);

        alert("There was an error fetching doctor details.");

      }

    }

    window.onload = getDoctorDetails;

  </script>

Doctor

Signup:

<script type="module">

    import { getAuth, createUserWithEmailAndPassword } from "https://www.gstatic.com/firebasejs/11.6.0/firebase-auth.js";

    import { getFirestore, setDoc, doc } from "https://www.gstatic.com/firebasejs/11.6.0/firebase-firestore.js";

    import { initializeApp } from "https://www.gstatic.com/firebasejs/11.6.0/firebase-app.js";

    const firebaseConfig = {

      apiKey: "AIzaSyA33Px0Y-ThP9Wv7MO68sWg4FaMP\_Z89ZA",

      authDomain: "hospital-management-203c5.firebaseapp.com",

      projectId: "hospital-management-203c5",

      storageBucket: "hospital-management-203c5.appspot.com",

      messagingSenderId: "45154128779",

      appId: "1:45154128779:web:bfcd7f5f00cea5d0c6d58e"

    };

    const app = initializeApp(firebaseConfig);

    const auth = getAuth(app);

    const db = getFirestore(app);

    document.getElementById("doctorForm").addEventListener("submit", async (event) => {

      event.preventDefault();

      const name = document.getElementById("name").value;

      const email = document.getElementById("email").value;

      const password = document.getElementById("password").value;

      const department = document.getElementById("department").value;

      try {

        const userCredential = await createUserWithEmailAndPassword(auth, email, password);

        const user = userCredential.user;

        // Store data in Firestore under 'doctors' collection

        await setDoc(doc(db, "doctors", user.uid), {

          name: name,

          email: email,

          department: department, // Department field

          role: "doctor", // Role to distinguish doctor

          status: "active",

          createdAt: new Date().toISOString()

        });

        alert("Doctor signed up successfully!");

        window.location.href = "adminDashboard.html";

      } catch (error) {

        console.error("Error signing up doctor:", error);

        alert("Error: " + error.message);

      }

    });

  </script>

Authentication:

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8" />

  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>

  <title>Doctor Login - Infinity Care Hospital</title>

</head>

<body>

<div class="login-container">

  <h3>Doctor Login</h3>

  <form id="doctorLoginForm">

    <div class="mb-3">

      <label for="doctorEmail" class="form-label">Email address</label>

      <input type="email" class="form-control" id="doctorEmail" placeholder="Enter email" required />

    </div>

    <div class="mb-3">

      <label for="doctorPassword" class="form-label">Password</label>

      <input type="password" class="form-control" id="doctorPassword" placeholder="Enter password" required />

    </div>

    <button type="submit" class="btn">Login</button>

    <div id="loginMsg" class="mt-3 text-center"></div>

  </form>

</div>

<script type="module">

  // Firebase imports

  import { initializeApp } from "https://www.gstatic.com/firebasejs/11.6.0/firebase-app.js";

  import { getAuth, signInWithEmailAndPassword } from "https://www.gstatic.com/firebasejs/11.6.0/firebase-auth.js";

  import { getFirestore, doc, getDoc } from "https://www.gstatic.com/firebasejs/11.6.0/firebase-firestore.js";

  const app = initializeApp(firebaseConfig);

  const auth = getAuth(app);

  const db = getFirestore(app);

  // Handle doctor login

  document.getElementById("doctorLoginForm").addEventListener("submit", async (e) => {

    e.preventDefault();

    const email = document.getElementById("doctorEmail").value;

    const password = document.getElementById("doctorPassword").value;

    try {

      const userCredential = await signInWithEmailAndPassword(auth, email, password);

      const user = userCredential.user;

      // Check doctor role from 'doctors' collection

      const userRef = doc(db, "doctors", user.uid);

      const userSnap = await getDoc(userRef);

      if (userSnap.exists()) {

        const data = userSnap.data();

        if (data.role === "doctor") {

          // ✅ Store doctor UID in localStorage

          localStorage.setItem("doctorUID", user.uid);

          alert("Login successful as doctor!");

          window.location.href = "doctor-appointments.html";

        } else {

          document.getElementById("loginMsg").textContent = "You are not authorized as a doctor.";

        }

      } else {

        document.getElementById("loginMsg").textContent = "Doctor data not found in Firestore.";

      }

    } catch (error) {

      console.error("Login error:", error.message);

      document.getElementById("loginMsg").textContent = "Login failed: " + error.message;

    }

  });

</script>

</body>

</html>

Check appointments:

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <title>Doctor's Appointments</title>

</head>

<body>

  <div class="appointments-container">

    <h2 id="doctorHeading">Doctor's Appointments</h2>

    <label for="filterDate">Filter by Date:</label>

    <input type="date" id="filterDate" />

    <button id="approveSelected" class="approve-btn" disabled>Approve Selected</button>

    <table id="appointmentsTable">

      <thead>

        <tr>

          <th>Select</th>

          <th>Patient Name</th>

          <th>Phone Number</th>

          <th>Gender</th>

          <th>Age</th>

          <th>Appointment Date</th>

          <th>Status</th>

        </tr>

      </thead>

      <tbody>

        <!-- Dynamic appointment rows will be added here -->

      </tbody>

    </table>

  </div>

  <!-- Firebase Modular SDK -->

  <script type="module">

    import { initializeApp } from "https://www.gstatic.com/firebasejs/11.6.0/firebase-app.js";

    import { getFirestore, collection, query, where, getDocs, doc, getDoc, updateDoc } from "https://www.gstatic.com/firebasejs/11.6.0/firebase-firestore.js";

    import { getAuth, onAuthStateChanged } from "https://www.gstatic.com/firebasejs/11.6.0/firebase-auth.js";

    const app = initializeApp(firebaseConfig);

    const db = getFirestore(app);

    const auth = getAuth(app);

    onAuthStateChanged(auth, async (user) => {

      if (!user) {

        window.location.href = "login.html";

        return;

      }

      try {

        const docRef = doc(db, "doctors", user.uid);

        const docSnap = await getDoc(docRef);

        let doctorName = "Doctor";

        if (docSnap.exists()) {

          doctorName = docSnap.data().name || "Doctor";

          if (!doctorName.toLowerCase().startsWith("dr")) {

            doctorName = "Dr. " + doctorName;

          }

        }

        document.getElementById("doctorHeading").innerText = `${doctorName}'s Appointments`;

        const q = query(collection(db, "appointments"), where("doctorUID", "==", user.uid));

        const snapshot = await getDocs(q);

        const allAppointments = snapshot.docs.map(doc => ({ id: doc.id, ...doc.data() }));

        const tableBody = document.getElementById("appointmentsTable").getElementsByTagName("tbody")[0];

        const filterDateInput = document.getElementById("filterDate");

        const approveSelectedBtn = document.getElementById("approveSelected");

        // Function to render the appointments

        function renderAppointments(filteredAppointments) {

          tableBody.innerHTML = "";

          filteredAppointments.forEach((a) => {

            const isPending = (a.status || "Pending") === "Pending";

            const row = document.createElement("tr");

            row.innerHTML = `

              <td><input type="checkbox" class="appointment-checkbox" data-id="${a.id}" ${isPending ? "" : "disabled"} /></td>

              <td>${a.patientName}</td>

              <td>${a.phoneNumber}</td>

              <td>${a.gender}</td>

              <td>${a.age}</td>

              <td>${a.date}</td>

              <td>${a.status || "Pending"}</td>

            `;

            tableBody.appendChild(row);

          });

          // Enable/disable approve button based on selection

          const checkboxes = document.querySelectorAll(".appointment-checkbox");

          const selectedCheckboxes = Array.from(checkboxes).filter(checkbox => checkbox.checked);

          approveSelectedBtn.disabled = selectedCheckboxes.length === 0;

          // Event listener for checkboxes

          checkboxes.forEach(checkbox => {

            checkbox.addEventListener("change", () => {

              const selectedCheckboxes = Array.from(checkboxes).filter(checkbox => checkbox.checked);

              approveSelectedBtn.disabled = selectedCheckboxes.length === 0;

            });

          });

        }

        // Function to approve selected appointments

        async function approveSelectedAppointments() {

          const selectedCheckboxes = Array.from(document.querySelectorAll(".appointment-checkbox"))

            .filter(checkbox => checkbox.checked);

          for (const checkbox of selectedCheckboxes) {

            const appointmentId = checkbox.getAttribute("data-id");

            try {

              await updateDoc(doc(db, "appointments", appointmentId), { status: "Approved" });

              // Update the status locally as well

              allAppointments.forEach(app => {

                if (app.id === appointmentId) {

                  app.status = "Approved";

                }

              });

              renderAppointments(allAppointments.filter(filterFunc));

            } catch (err) {

              console.error("Error updating appointment:", err);

            }

          }

        }

        // Initial render of appointments

        function filterFunc(app) {

          const selectedDate = filterDateInput.value;

          return selectedDate ? app.date === selectedDate : true;

        }

        renderAppointments(allAppointments);

        // Event listener for filter by date

        filterDateInput.addEventListener("change", () => {

          renderAppointments(allAppointments.filter(filterFunc));

        });

        // Event listener for approving selected appointments

        approveSelectedBtn.addEventListener("click", approveSelectedAppointments);

      } catch (err) {

        console.error("Error loading data:", err);

      }

    });

  </script>

</body>

</html>

1. Testing

**6.1 Introduction to Testing**

Testing is a critical phase in the software development life cycle (SDLC), as it ensures that the application functions as intended, delivers accurate results, and meets user expectations. In the context of the Hospital Management System (HMS), testing was conducted extensively to validate the performance, security, user interaction, and functional modules across all user roles: patient, doctor, and admin.

The objective of testing is not just to find and fix bugs, but to guarantee the quality and reliability of the system. It helps verify that:

* Patients can log in and book appointments without errors
* Doctors can view and manage their schedule smoothly
* Admins can monitor system activity with real-time data
* All modules work correctly under various conditions

Testing also helps identify edge cases, such as missing form inputs, incorrect logins, invalid appointment times, or system behavior under unexpected actions.

**Types of Testing Used**

To thoroughly validate the HMS system, multiple types of testing were performed during development:

**1. Unit Testing**

Each function or module was tested individually to ensure that it produced the correct output when given expected input. For example:

* Firebase login functionality
* Appointment form validation
* Dashboard data rendering

**2. Integration Testing**

Modules were tested together to ensure they work correctly as a group. For example:

* Booking an appointment and displaying it on the doctor’s dashboard
* Admin adding a doctor and doctor appearing in the appointment list

**3. System Testing**

End-to-end testing of the complete HMS application. This included:

* Logging in as a patient
* Booking appointments
* Logging in as a doctor to verify appointments
* Logging in as admin to view records

**4. User Interface (UI) Testing**

Each page was tested for:

* Form alignment
* Mobile responsiveness
* Button functionality
* Proper form field validations and error messages

**5. Performance Testing (Basic)**

The system was tested with multiple dummy records to check loading time and database sync performance.

**Tools Used for Testing**

| **Tool/Method** | **Purpose** |
| --- | --- |
| **Manual Testing** | Used for UI and flow validation |
| **Firebase Console** | Checked real-time database entries |
| **Browser Developer Tools** | Checked JavaScript errors and API responses |

7. Conclusion and Future Enhancement

**7.1 Conclusion**

The development of the Hospital Management System (HMS) marks a significant step toward the digital transformation of hospital operations. Through this project, a complete cloud-based system was successfully designed and implemented to address common challenges in hospital administration, patient appointment scheduling, doctor availability, and centralized medical data management.

The system provides a user-friendly platform for three major roles — patients, doctors, and administrators. With real-time features powered by Firebase, users experience seamless data interaction, instant updates, and secure access from any device with internet connectivity. The integration of HTML, CSS, and JavaScript in the frontend ensures that the system is visually clean, mobile-responsive, and easy to use even for non-technical users.

Key features such as secure login, role-based dashboards, appointment booking, patient history tracking, and doctor schedule viewing were implemented successfully and tested thoroughly. The modular and scalable nature of the system makes it suitable for deployment in small to medium-sized hospitals and clinics with minimal setup cost and infrastructure requirements.

Overall, the HMS project has met its objectives of reducing manual workload, improving patient service quality, and increasing operational efficiency through digital automation. It also serves as a foundation for future academic or real-world improvements and expansion.

**7.2 Future Enhancements**

While the current version of HMS fulfils core requirements, there is substantial scope for adding more features to improve functionality, scalability, and usability in future versions.

**Suggested Enhancements:**

1. **Billing System Integration**  
   Implement a digital billing module to allow hospitals to generate and store bills for treatments and services directly through the platform.
2. **Pharmacy Module**  
   Include a pharmacy interface for prescription uploads, drug inventory tracking, and medicine distribution records.
3. **Notification System**  
   Add SMS or email reminders for patients regarding upcoming appointments or follow-up schedules.
4. **Multi-language Support**  
   Enhance accessibility by supporting regional languages for patients from diverse backgrounds.
5. **Reporting and Analytics**  
   Generate performance reports for doctors, track patient statistics, appointment trends, and system usage for administrators.
6. **Mobile App Version**  
   Build an Android/iOS version of the system for more convenient access via smartphones.
7. **AI-based Symptom Checker**  
   Introduce a basic AI tool for patients to enter symptoms and receive preliminary advice or suggested departments.

8. Bibliography

The following is a list of tools, platforms, websites, and references used during the development and documentation of the **Hospital Management System (HMS)** project. This section provides credit to open-source technologies, learning platforms, documentation sites, and academic resources that contributed to both the technical implementation and theoretical understanding.

**Web Technologies**

1. **HTML & CSS**  
   – MDN Web Docs  
   – <https://developer.mozilla.org/en-US/>
2. **JavaScript**  
   – W3Schools JavaScript Tutorial  
   – https://www.w3schools.com/js/  
   – JavaScript.info  
   – <https://javascript.info/>

**Firebase Platform**

1. **Firebase Documentation**  
   – Firebase Authentication, Firestore, Hosting  
   – https://firebase.google.com/docs
2. **Firebase YouTube Channel**  
   – Tutorials and deployment guidance  
   – <https://www.youtube.com/@Firebase>

**Development Tools**

1. **Visual Studio Code**  
   – <https://code.visualstudio.com/>
2. **GitHub (Version Control)**  
   – <https://github.com/>
3. **Google Fonts & Icons**  
   – <https://fonts.google.com/>  
   – https://fonts.googleapis.com/icons

**Design & Diagram Tools**

1. **Draw.io (ER & DFD diagrams)**  
   – <https://draw.io/>
2. **Lucid chart**  
   – <https://www.lucidchart.com/>

**Testing References**

1. **Software Testing Guide – Guru99**  
   – https://www.guru99.com/software-testing.html
2. **Postman API Testing Tool** *(Optional for advanced testing)*  
   – <https://www.postman.com/>

**Academic and Research References**

1. Project Guide Instructions – Internal mentor, Koshys Institute of Management Studies
2. Sample Reports provided by Bengaluru North University
3. Classroom notes, internal lab materials, and peer-reviewed student projects

**09. APPENDICES A-TABLE STRUCTURE**

**Table Name: User’s Collections**

|  |  |  |
| --- | --- | --- |
| Field Name | Data Type | Description |
| UID | String | Unique user Id (generated by Firebase) |
| Fullname | String | Full name of the user |
| Email | String | Email Address |
| Password | String | Encrypted Password |
| Role | String | Role of user: admin, patient, doctor |
| createdAt | Timestamp | Account creation time |

**Table Name: Appointment’s Collections**

|  |  |  |
| --- | --- | --- |
| Field Name | Data Type | Description |
| AppointmentId | String | Unique Id for appointment |
| PatientId | String | Reference to patient’s Id |
| DoctorId | String | Reference to doctor’s Id |
| Date | String | Appointment date(YYYY-MM-DD) |
| Time | String | Appointment time (HH:MM) |
| Status | String | Status: pending, confirmed, completed |
| CreatedAt | Timestamp | When appointment was booked |

**Table Name: Doctor’s Collections**

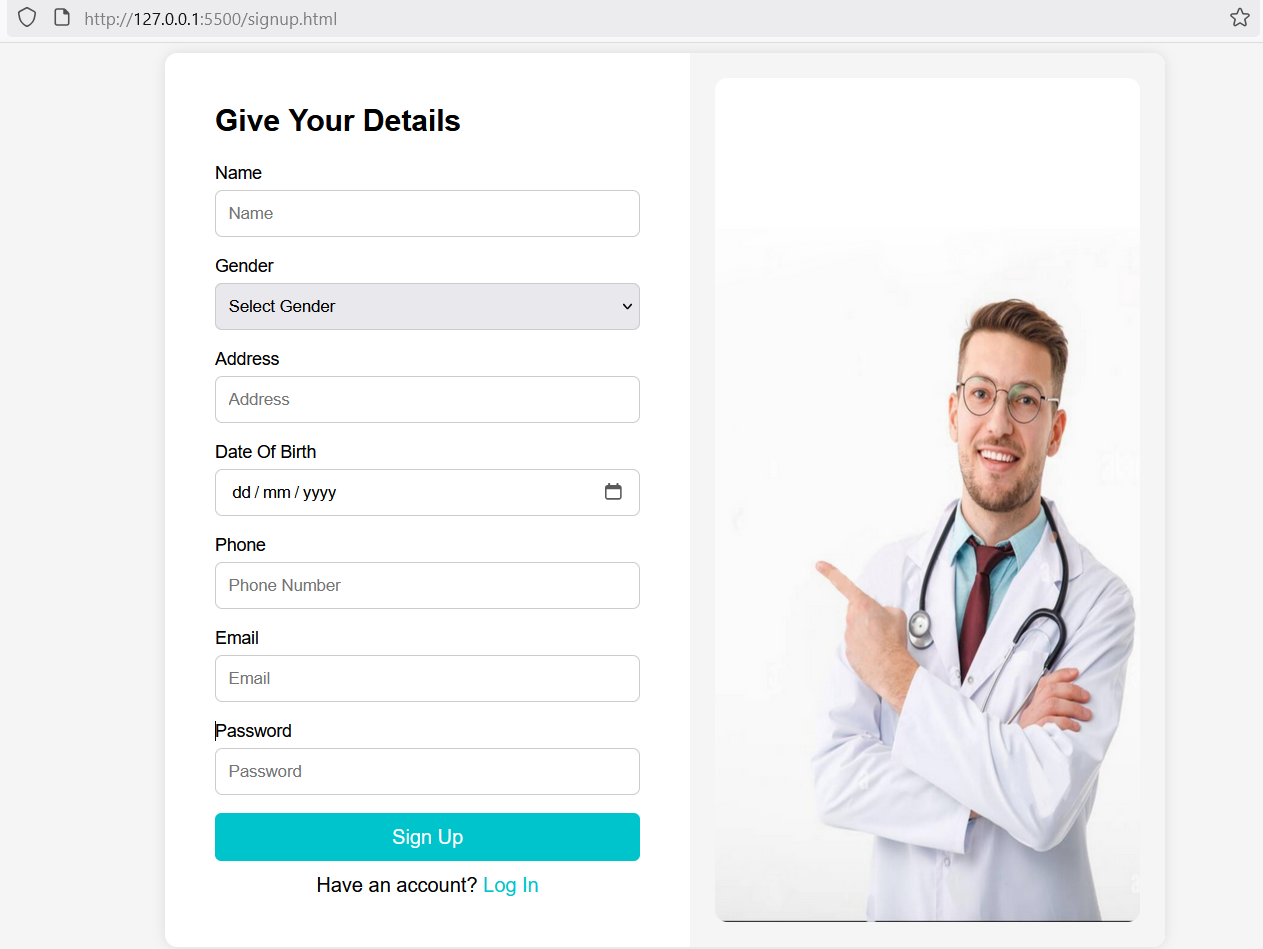
|  |  |  |
| --- | --- | --- |
| Field Name | Data Type | Description |
| DoctorId | String | Firebase User Uid |
| FullName | String | Doctor’s Name |
| DepartmentId | String | Reference to the department |
| Availability | Array | List of available days/times |
| Specialization | String | Medical Specialty |

**Table Name: Department’s Collections**

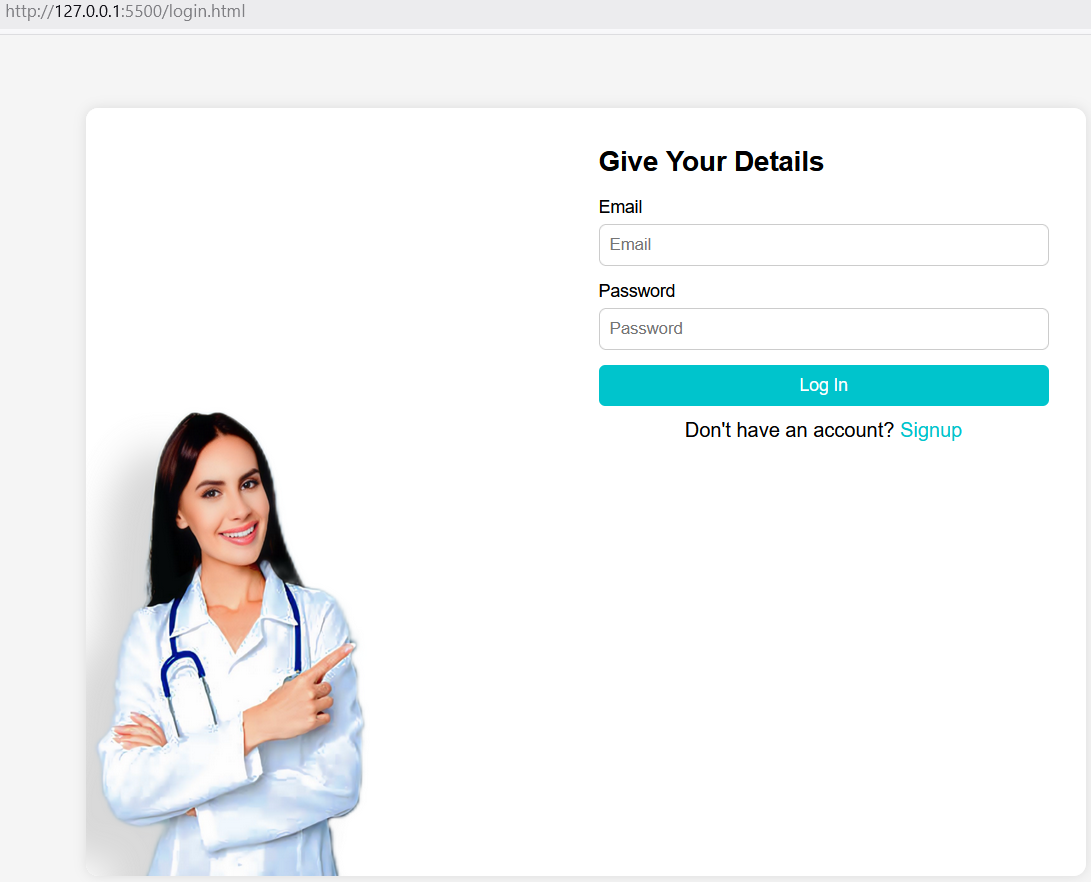
|  |  |  |
| --- | --- | --- |
| Field Name | Data Type | Description |
| DepartmentId | String | Unique department id |
| Name | String | Department name (eg., Cardiology) |
| Description | String | Brief about department |

**09. APPENDICES B-SCREENSHOTS**

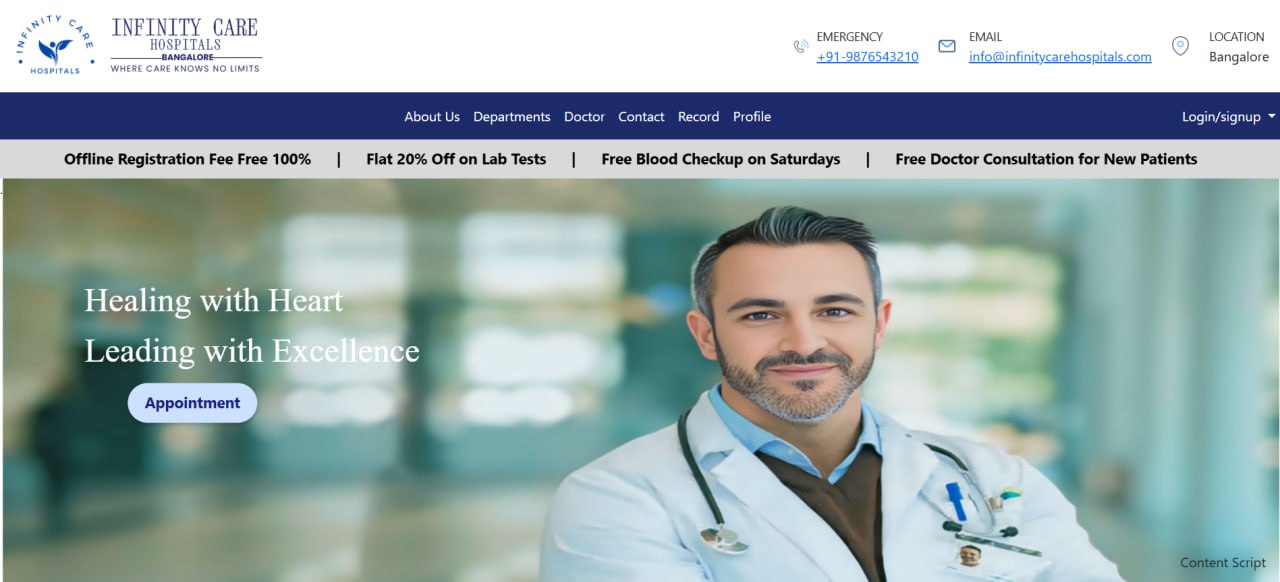
**User’s sign-up page**



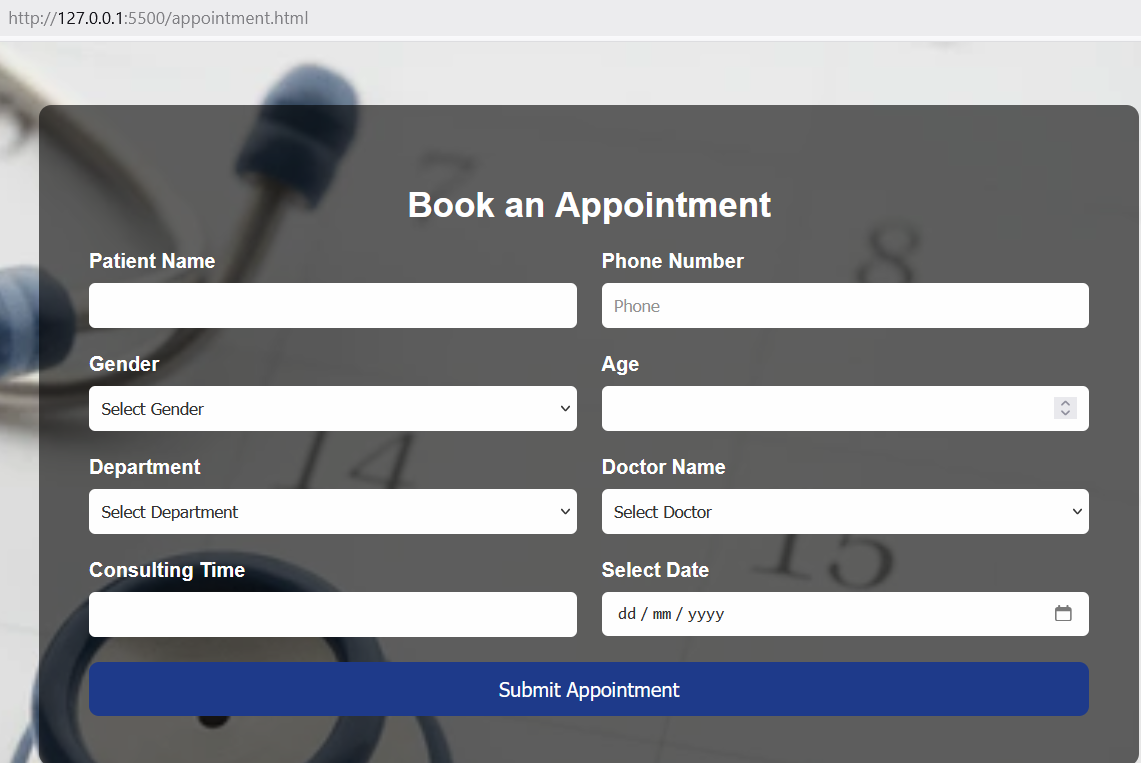
**User’s sign-in page**

****

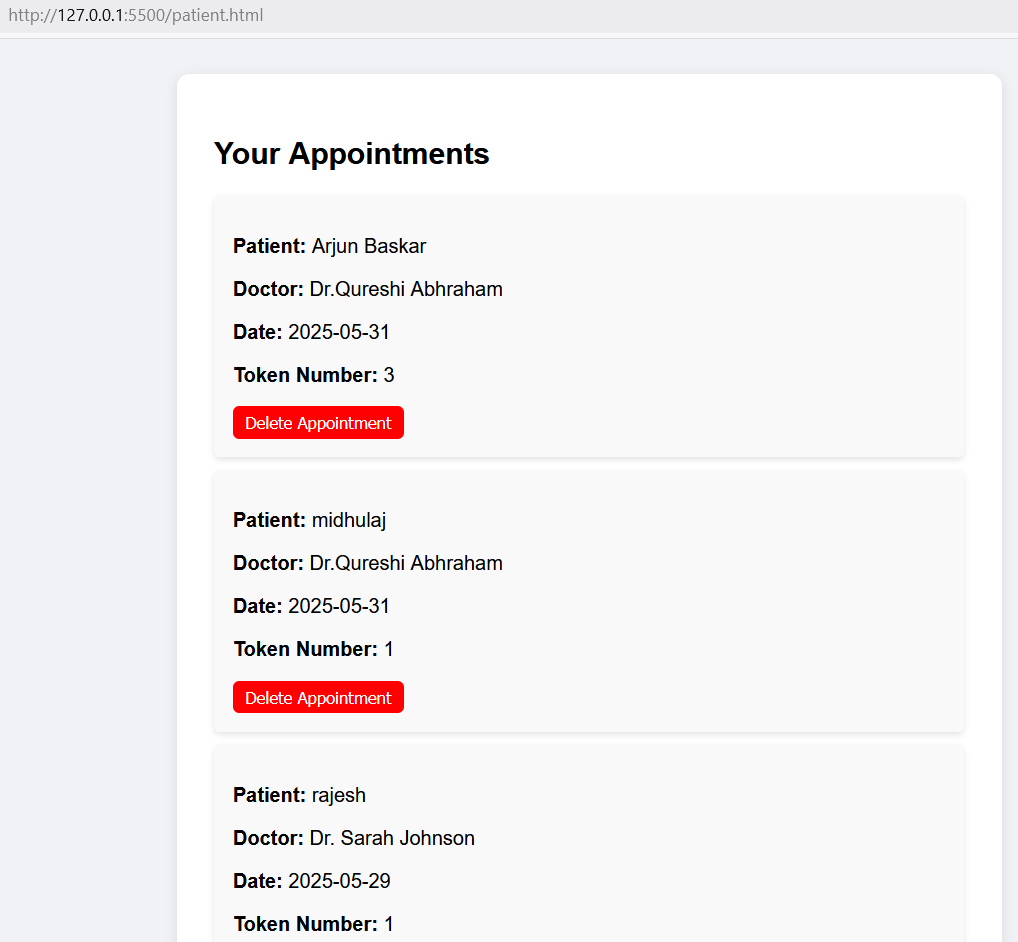
**Home-page**

****

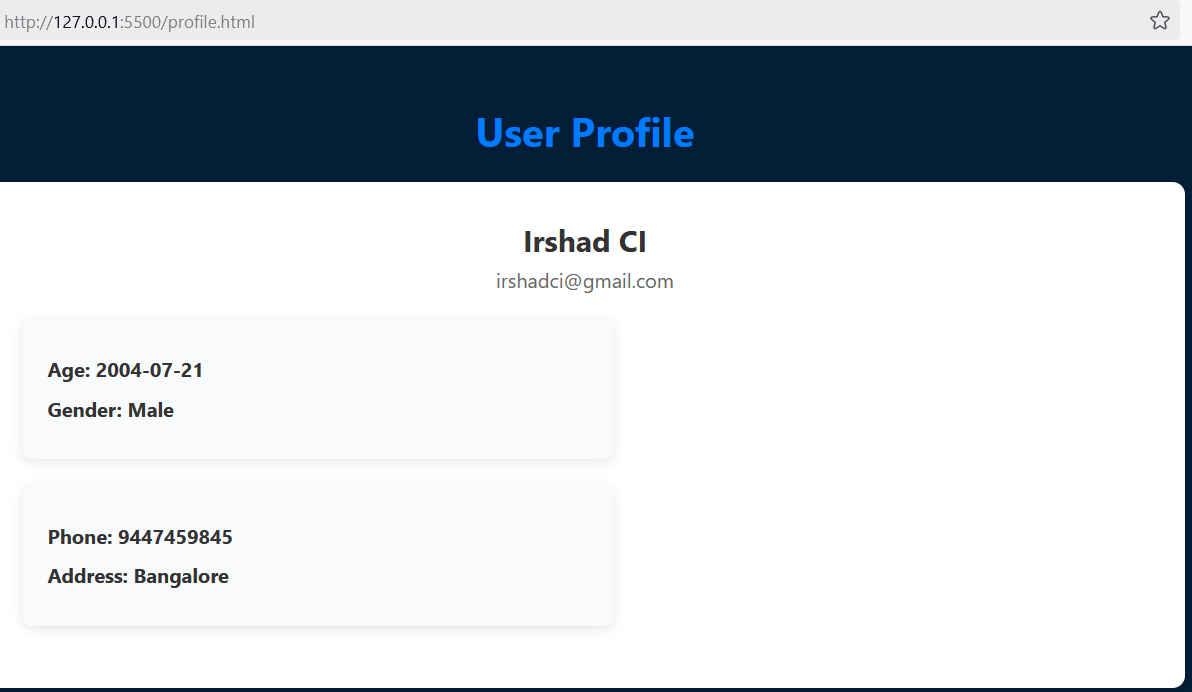
**Appointment booking-page**

****

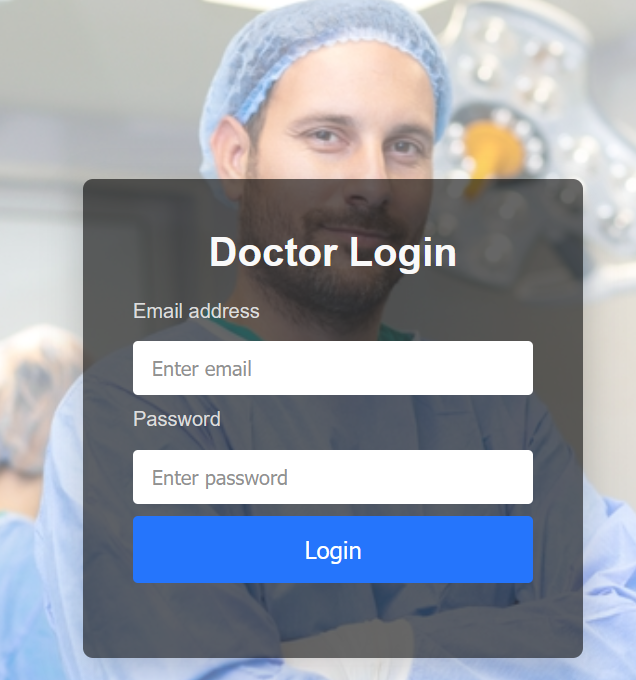
**Appointment list-page**

****

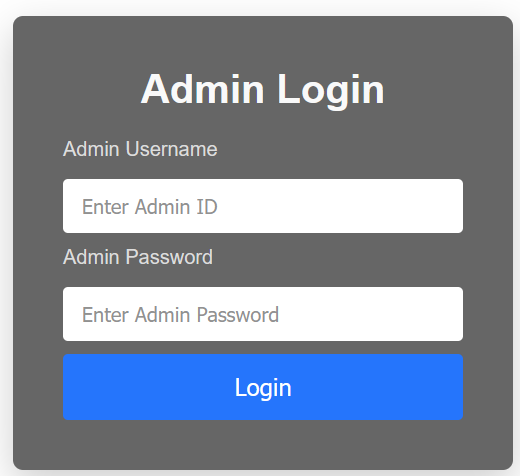
**Profile-page**

****

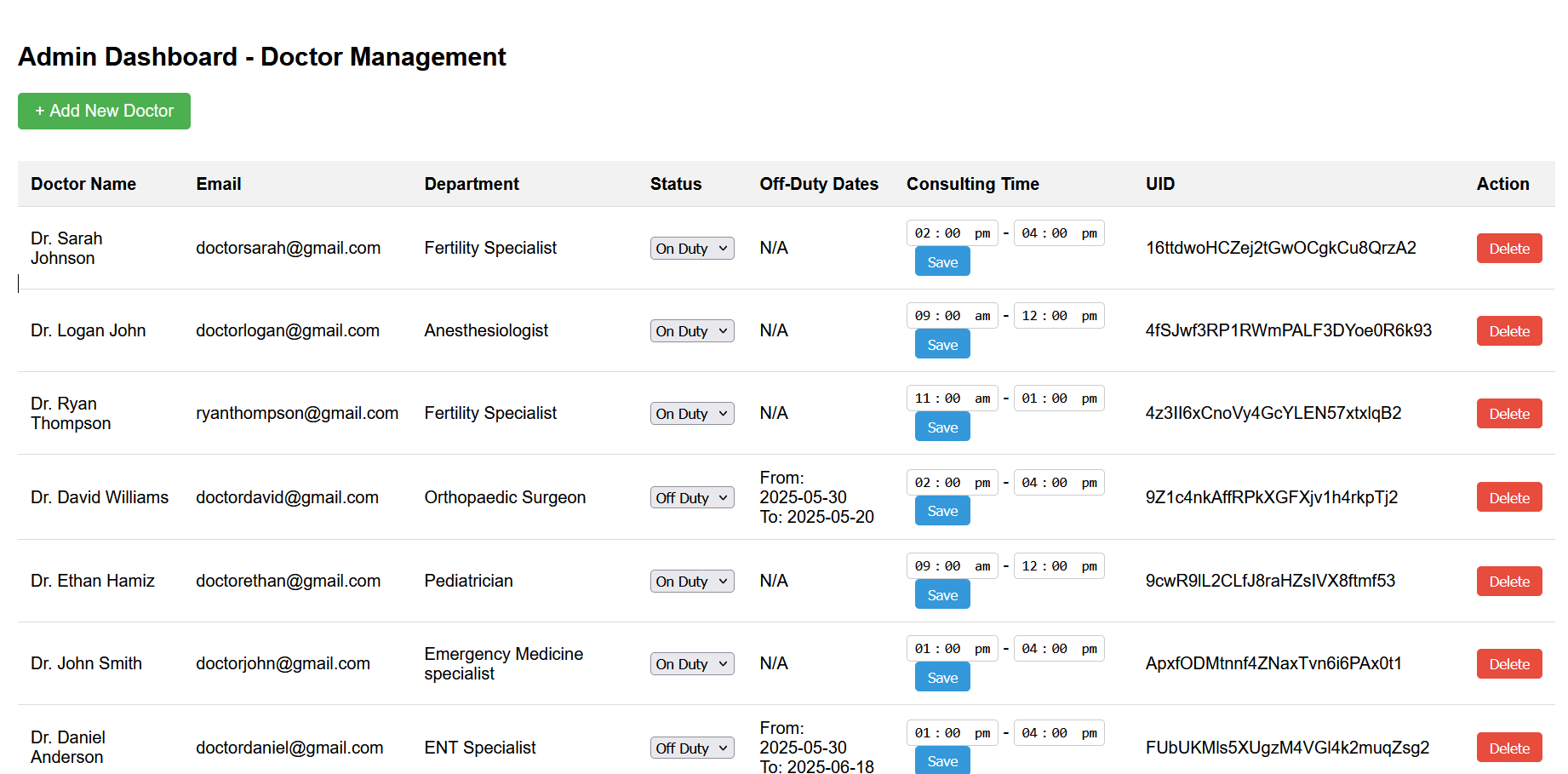
**Doctor login-page**

****

**Admin login-page**

****

**AdminDashboard-page**

****

**09. APPENDICES C-SAMPLE REPORT OF**

**TEST CASES**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Test Scenario** | **Input** | **Expected Output** | **Actual Output** | **Status** |
| TC001 | Patient Registration | Valid user details | Account Created | Account Created | Pass |
| TC002 | Patient Login | Valid email & password | Login Successful | Login Successful | Pass |
| TC003 | Doctor Login | Invalid password | Login Failed | Login Failed | Pass |
| TC004 | Book Appointment | Select doctor & time | Appointment booked | Appointment booked | Pass |
| TC005 | View Appointments | Logged-in Patients | List of appointments | List of appointments | Pass |
| TC006 | Admin Add Doctor | Valid doctor details | Doctor added to list | Doctor added to list | Pass |
| TC007 | Double Booking | Same slot for same doctor | Booking rejected | Booked rejected | Pass |
| TC008 | Doctor View Schedule | Logged-in doctor | View Schedule list | View Schedule list | Pass |
| TC009 | Patient Login | Wrong email format | Validation error | Login successful | Fail |
| TC010 | Booking Appointment | Missing time slot | Booking rejected | Appointment booking | Fail |
| TC011 | Admin Login | Correct Credentials | Login successful | Login failed | Fail |
| TC012 | View Department | Unauthorized access | Access denied | Data displayed | Fail |
| TC013 | Edit Profile | Valid updates | Profile updated | Profile updated | Pass |
| TC014 | Delete Appointment | Admin delete one | Appointment deleted | Appointment deleted | Pass |
| TC015 | Search Doctor | Keyword: Cardio | Matching doctors | Matching doctors | Pass |
| TC016 | Load Dashboard | After login | Dashboard displayed | Dashboard displayed | Pass |
| TC017 | View Doctor List | Patient accesses list | List down | List down | Pass |
| TC018 | Book Past Appointment | Date in the past | Error shown | Appointment booked | Fail |
| TC019 | Form Validation | Empty required fields | Error message | Error message | Pass |
| TC020 | Edit Doctor Availability | Valid data | Availability updated | Not updated | Fail |
| TC021 | Delete Patient Record | Admin deletes record | Record deleted | Record deleted | Pass |
| TC022 | Email Format Validation | Invalid email input | Error message | Error message | Pass |
| TC023 | Simultaneous Login | Same user on 2 devices | Login success | One session logged out | Fail |
| TC024 | View Reports | Admin user | Reports displayed | Reports displayed | Pass |
| TC025 | Security Test | Attempt SQL injection | Request denied | Request denied | Pass |