

Naan Mudhalvan Project
MONGODB With MERN STACK

Project Title: Book a Doctor using MERN

Submitted By

KEERTHIGA K.S– 212421104019

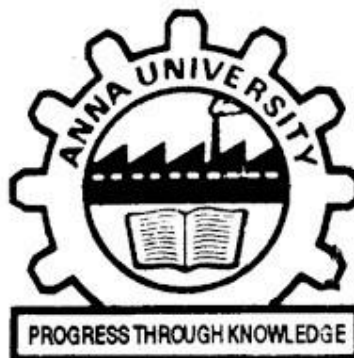
PREETHI N – 212421104028

AUGUSTIN SILVERSTAR D -212421104003

JOSHUA -212421104018

JAISEELAN- 212421104015

**SREE SASTHA INSTITUTE OF
ENGINEERING AND TECHNOLOGY
CHEMBARAMBAKKAM
CHENNAI - 6000123**



ANNA UNIVERSITY CHENNAI – 600025

NOV/DEC 2024

BONAFIDE CERTIFICATE

Certified that this report titled " Book a Doctor using MERN " for the Naan mudhalvan project is a bonafide work of (KEERTHIGA K.S, PREETHI N, AUGUSTIN SILVERSTAR D, JOSHUA, JAISEELAN) in MERN stack by Mongo DB -NM1016 who carried out the work under my supervision.

Certified further that to the best of my knowledge, the work reported here does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

FACULTY MENTOR

HEAD OF DEPARTMENT

SPOC

Submitted for the University Practical Examination held on _____

Internal Examiner

External Examiner

Index

Index	Section Title	Page No.
1	Abstract	1
2	Project Overview	2
3	Technology Stack & System Requirements	3
4	Project Architecture	4
5	Installation and Setup	5
6	Folder Structure	6
7	Workflow and Usage	8
8	Testing	9
9	Project Implementation & Execution	11
10	Challenges Faced	15
11	Future Enhancements	16
12	Conclusion	17
13	References	18

Abstract

The "Book a Doctor Using MERN" is a modern web application aimed at streamlining the doctor appointment booking process for patients, healthcare providers, and administrators. Built using the MERN stack (MongoDB, Express.js, React.js, Node.js), this platform offers a centralized solution for users to register, browse doctors, book consultations, and manage appointments effortlessly.

A key feature of the application is its role-based access model, distinguishing between three primary user roles—Patient, Doctor, and Admin. Patients can search for doctors based on specialty, location, and availability, and secure appointments through a simple and intuitive interface. Doctors, upon admin approval, manage their schedules, confirm appointments, and update patient records. Administrators oversee platform operations, including doctor verification, governance, and compliance, ensuring a secure and efficient system.

A key feature of the application is its role-based access model, distinguishing between three primary user roles—Patient, Doctor, and Admin. Patients can search for doctors based on specialty, location, and availability, and secure appointments through a simple and intuitive interface. Doctors, upon admin approval, manage their schedules, confirm appointments, and update patient records. Administrators oversee platform operations, including doctor verification, governance, and compliance, ensuring a secure and efficient system.

Future enhancements for the "Book a Doctor Using MERN" include advanced features like video consultations, AI-powered doctor recommendations, and enhanced filtering options. These developments aim to elevate user satisfaction, making the application a reliable and versatile tool in modern healthcare management, bridging the gap between patients and providers with ease and efficiency.

CHAPTER 1. Project Overview

The "Book a Doctor Using MERN" application is a web-based platform designed to simplify the process of scheduling doctor appointments for patients while enabling doctors to manage their consultations efficiently. Patients can register, search for healthcare providers based on specific preferences, book appointments in real-time, and upload necessary documents. Meanwhile, doctors can manage their schedules, approve appointments, and maintain patient records with ease.

To ensure a seamless user experience, the application incorporates an admin panel that oversees platform governance, verifies doctor registrations, and monitors activities. Administrators ensure compliance with policies, address disputes, and maintain the platform's quality and security.

The main goal of the "Book a Doctor Using MERN" application is to create a centralized platform where patients can effortlessly connect with suitable doctors while healthcare providers efficiently manage their appointments and patient interactions. By integrating essential features such as real-time appointment booking, advanced search filters, and secure authentication, the app enhances accessibility and convenience in healthcare.

The application also aims to modernize the healthcare booking process, offering a user-friendly and secure solution that caters to the growing need for online healthcare management.

Objectives

The primary objectives of the "Book a Doctor Using MERN" application are:

1. **Creating a Unified Platform:** Integrating patients, doctors, and administrators into a single application where each role can seamlessly perform its respective functions.
 2. **Facilitating Doctor Management:** Allowing doctors to efficiently manage their schedules, confirm or reschedule appointments, and update patient records post-consultation.
 3. **Simplifying Appointment Booking for Patients:** Enabling patients to browse doctors based on specialty, location, and availability, and book appointments in real-time with minimal effort.
 4. **Ensuring Platform Security:** With admin oversight, the application ensures a secure user experience by validating doctor registrations, monitoring platform activities, and enforcing compliance with privacy and data security standards.
 5. **Improving Communication:** Providing timely notifications and updates about appointments, confirmations, or cancellations to ensure transparent communication between patients and doctors.
-

CHAPTER 2. Technology Stack & System Requirements

The "Book a Doctor Using MERN" application is built using the MERN stack, a robust combination of technologies that enables the development of modern, dynamic, and scalable web applications. Core Components of the Stack:

- **MongoDB:** A NoSQL database used for storing user information, doctor profiles, appointment details, and medical records efficiently.
- **Express.js:** A flexible backend framework that powers the APIs and handles server-side logic for the application.
- **React.js:** A frontend framework used to build a dynamic, responsive user interface, ensuring seamless interaction for all users.
- **Node.js:** A runtime environment that executes backend JavaScript, handling server requests efficiently and reliably.

In addition to the MERN stack, several other technologies are utilized to enhance the user experience and application security:

- **Bootstrap and Material UI:** CSS frameworks utilized to design a responsive, intuitive, and aesthetically pleasing user interface.
- **JSON Web Token (JWT):** Implements secure user authentication and session management to ensure data protection.
- **Axios:** Used for handling API requests between the frontend and backend, ensuring smooth data exchange.

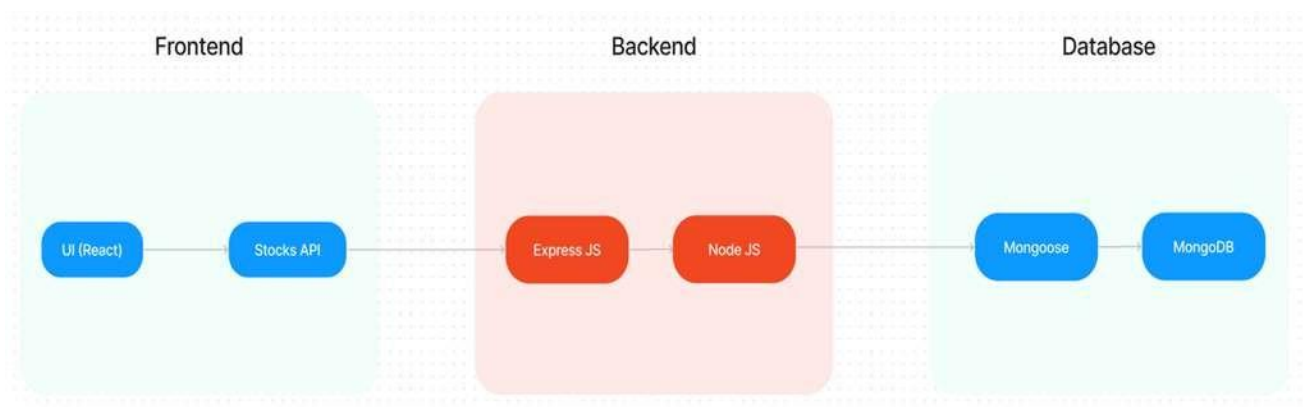
System Requirements:

To run the House Rent App, the following hardware and software are required:

- **Hardware:**
 - Windows 8 or higher machine with a stable internet connection (30 Mbps recommended).
 - **Software:**
 - **Node.js:** The latest version of Node.js should be installed to run the backend server.
 - **MongoDB Community Server:** Required for managing the database.
 - **Web Browsers:** At least two web browsers (e.g., Google Chrome, Mozilla Firefox) are recommended for cross-browser compatibility testing.
-

CHAPTER 3. Project Architecture

1. The application follows a modular architecture, split between frontend (client-side) and backend (server-side) components:
2. **Frontend (Client-Side):**
 - Built with React.js for dynamic rendering, creating an intuitive and interactive experience for users.
 - CSS frameworks like Bootstrap, Material UI, and Ant Design are used to style components, ensuring consistency in design.
 - Communicates with backend services using API endpoints, enabling real-time data handling and smooth user interactions.
3. **Backend (Server-Side):**
 - Express.js handles the core API and server functions, managing routes and ensuring that each request is processed efficiently.
 - MongoDB serves as the database, storing information related to users, doctors, and booking transactions.
 - JWT tokens secure user authentication and authorization, preventing unauthorized access.
4. **Database (MongoDB):**
 - MongoDB collections store all app-related data, including user profiles, doctor details, inquiries, and bookings.
 - The database is designed for optimal retrieval, supporting fast queries and handling real-time updates efficiently.



CHAPTER 4. Installation and Setup

Prerequisites

1. Install **Node.js** and **npm**.
2. Install **MongoDB Community Server**.

Step-by-Step Setup

1. Clone the Repository

https://github.com/KSKEERTHIGA/BOOK_A_DOCTOR_USING_MERN.git

2. Backend Setup:

- a. Navigate to the backend folder and install dependencies:
 - `cd book-a-doctor/backend`
 - `npm install`
- b. Create an `.env` file with MongoDB connection and JWT key.
- c. Modify the MongoDB connection string in `connect.js` in the `config` folder.
- d. Start the backend server:
 - `npm start`

3. Frontend Setup:

- a. Navigate to the frontend folder and install dependencies:
 - `cd ../frontend`
 - `npm install`
- b. Start the frontend server:
 - `npm start`

4. Access the App:

- a. Frontend: `http://localhost:3000`
 - b. Backend: `http://localhost:8000`
-

CHAPTER 5. FOLDER STRUCTURE

The folder structure of the Book a Doctor using MERN App is organized to separate frontend and backend components, ensuring modularity and maintainability.

book-a-doctor/

- | — frontend/ # Frontend (client-side) application folder
 - | | — public/ # Static files accessible to users
 - | | | — index.html # Main HTML file for the React app
 - | | | — assets/ # Images, icons, and other static assets
 - | | — src/ # Source code for the React app
 - | | | — components/ # Reusable React components
 - | | | | — Admin/ # Components for admin functionalities
 - | | | | — Common/ # Shared components (Navbar, Footer)
 - | | | | — User/ # Components for user pages
 - | | | | | — Doctor/ # Components for doctor-related pages
 - | | | — pages/ # Page-level components (e.g., Home, Login, Dashboard)
 - | | | — App.js # Main App component with routing
 - | | | | — index.js # Entry point for the React application
 - | | — package.json # Project metadata and dependencies for frontend
 - | | — .env # Environment variables for frontend

- └─ backend/ # Backend (server-side) application folder
 - └─ config/ # Environment and database configurations
 - └─ connectToDB.js # MongoDB connection setup
 - └─ controllers/ # Logic for handling API requests
 - └─ adminController.js # Handles admin-related requests
 - └─ doctorController.js # CRUD operations for doctors
 - └─ userController.js # Handles user-related functionalities
 - └─ middlewares/ # Middleware functions for request validation
 - └─ authMiddleware.js # JWT-based authentication middleware
 - └─ errorMiddleware.js # Error-handling middleware
 - └─ routes/ # API routes for different functionalities
 - └─ adminRoutes.js # Routes for admin operations
 - └─ doctorRoutes.js # Routes for doctor operations
 - └─ userRoutes.js # Routes for user operations
 - └─ models/ # Mongoose schemas for MongoDB collections
 - └─ User.js # Schema for user details (Admin, Doctor, Patients)
 - └─ Appointment.js # Schema for booking appointments
 - └─ Doctor.js # Schema for doctor profiles
 - └─ uploads/ # Folder for storing uploaded files (images, documents)
 - └─ .env # Environment variables for backend (MongoDB URI, JWT secret)
 - └─ server.js # Entry point for the Express server
 - └─ package.json # Project metadata and dependencies for backend
- └─ README.md # Documentation and instructions for the project

CHAPTER 6. Workflow and Usage

User Roles and Functionalities

The Book a Doctor using MERN App includes three primary user roles: Patient, Doctor, and Admin. Each role has specific responsibilities and functionalities to ensure smooth operation of the platform.

1. Patient:

- **Account Creation and Login:** Patients create an account and log in using their email and password.
- **Search and Book Appointments:** Patients can browse through available doctors based on specialization, location, or availability. They can book an appointment by filling out the required details.
- **Appointment Management:** Patients can view their upcoming and past appointments in the "My Appointments" section. They can also cancel or reschedule appointments based on their needs.

2. Doctor:

- **Account Approval:** Doctors must first receive approval from the admin to activate their accounts.
- **Profile Management:** Once approved, doctors can create and manage their profiles, including adding details like specialization, experience, availability, and consultation fees.
- **Appointment Management:** Doctors can view and manage their appointments. They can confirm or reject bookings and update their availability status.
- **Consultation Notes:** Doctors can add consultation notes or prescription details to completed appointments, which will be visible to the patient.

3. Admin:

- **User and Doctor Approval:** Admins review and approve user accounts for patients and doctors, ensuring only legitimate users gain access to the platform.
 - **Platform Monitoring:** Admins monitor all platform activities, including user interactions, appointment statuses, and doctor approvals, to maintain smooth operations.
 - **Policy Enforcement:** Admins ensure compliance with platform policies, terms of service, and privacy regulations to maintain a secure and trustworthy environment.
 - **Reports and Insights:** Admins can generate reports on appointment statistics, doctor activity, and platform usage to track performance and ensure quality service.
-

CHAPTER 7. Testing

Manual testing was conducted to ensure that the **Book a Doctor using MERN App** functions as expected and provides a seamless experience for users. The focus of manual testing included:

1. User Registration and Login:

- Tested the registration and login functionality for both patients and doctors, ensuring that the correct validation messages appear for invalid inputs (e.g., empty fields, incorrect credentials).
- Verified that only approved doctors (with admin authorization) could access the platform and manage their profiles.

2. Doctor Profiles:

- Verified that doctors could create and update their profiles, including adding specializations, availability, and consultation fees.
- Checked that profile details were correctly displayed to patients on the frontend.

3. Search and Filter Functionality:

- Tested the search and filter options to ensure that doctors could be filtered by criteria such as specialization, location, and availability.
- Ensured that applying filters resulted in relevant doctor profiles being displayed.

4. Appointment Booking and Management:

- Tested the booking process by submitting appointment requests as a patient and confirming that doctors could view, accept, or reject these requests.
- Verified that appointment statuses (e.g., "pending," "confirmed," "rejected") were updated correctly and reflected accurately on both the patient's and doctor's dashboards.

5. Admin Functionality:

- Checked if the admin could approve doctor accounts, ensuring that only authorized doctors could access the system.
- Verified the admin's ability to monitor platform activities, review user and doctor registrations, and manage any necessary updates.

6. Responsiveness:

- Tested the user interface on various devices (desktop, tablet, mobile) to ensure that the layout adapts appropriately to different screen sizes.
- Confirmed that all key functionalities remained accessible and user-friendly across devices.

7. Error Handling:

- Tested invalid inputs and error scenarios (e.g., unauthorized appointment booking, incorrect profile details) to ensure that the system provides appropriate error messages.
- Ensured the platform does not crash or behave unexpectedly in error scenarios.

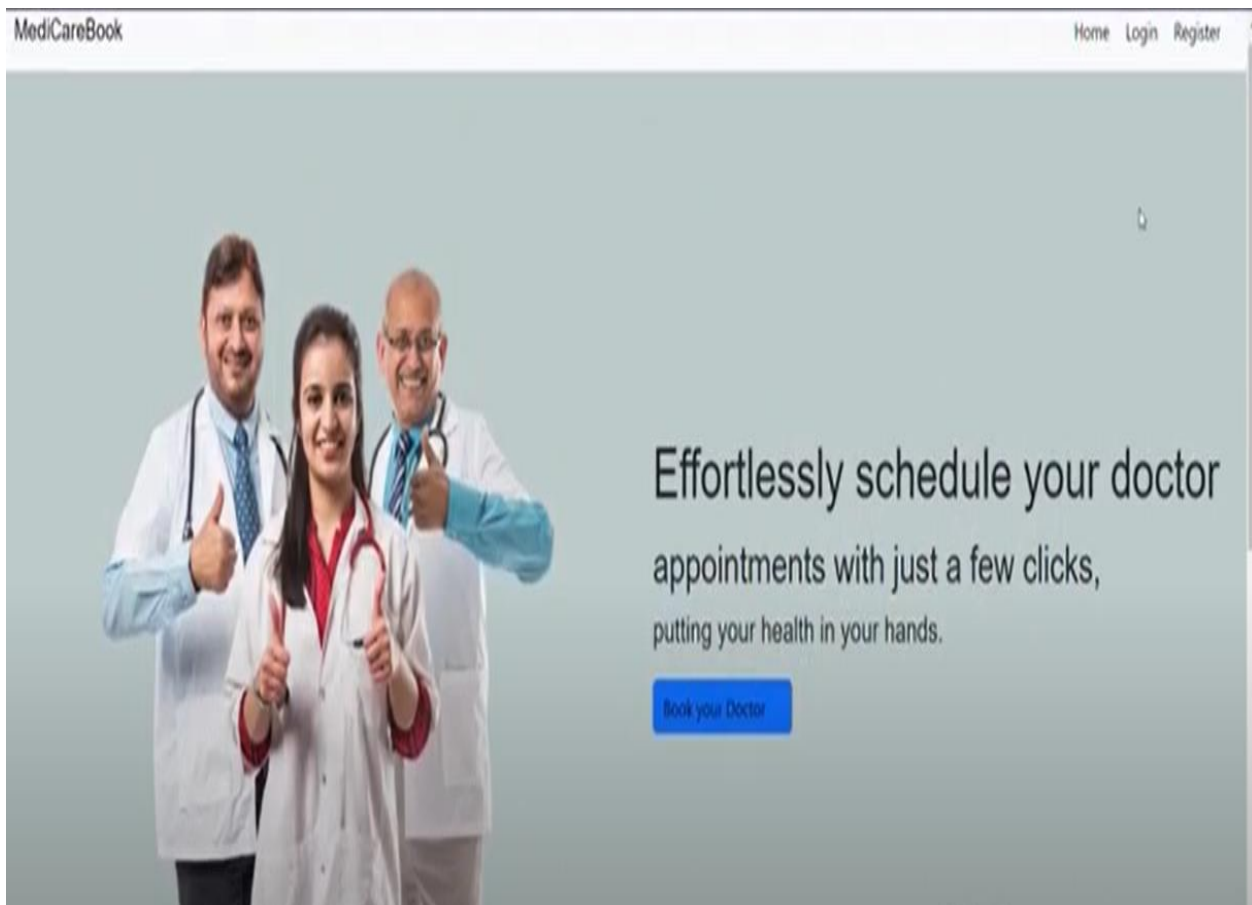
8. Real-Time Updates:

- Verified that appointment statuses and doctor availability were updated in real time and reflected correctly on both the patient's and doctor's dashboards.
-

CHAPTER 8. Project Implementation & Execution

On completing the development part, we then run the application one last time to verify all the functionalities and look for any bugs in it. The application's user interface looks a bit like the one provided below.

- **Landing page:**



- **Register page:**

MediCareBook Home Login Register

Sign up to your account

Full name


Email

Password

Phone

☐ Admin ☐ User

[Have an account? Login here](#)



- **Login page:**


MediCareBook Home Login Register

Sign in to your account

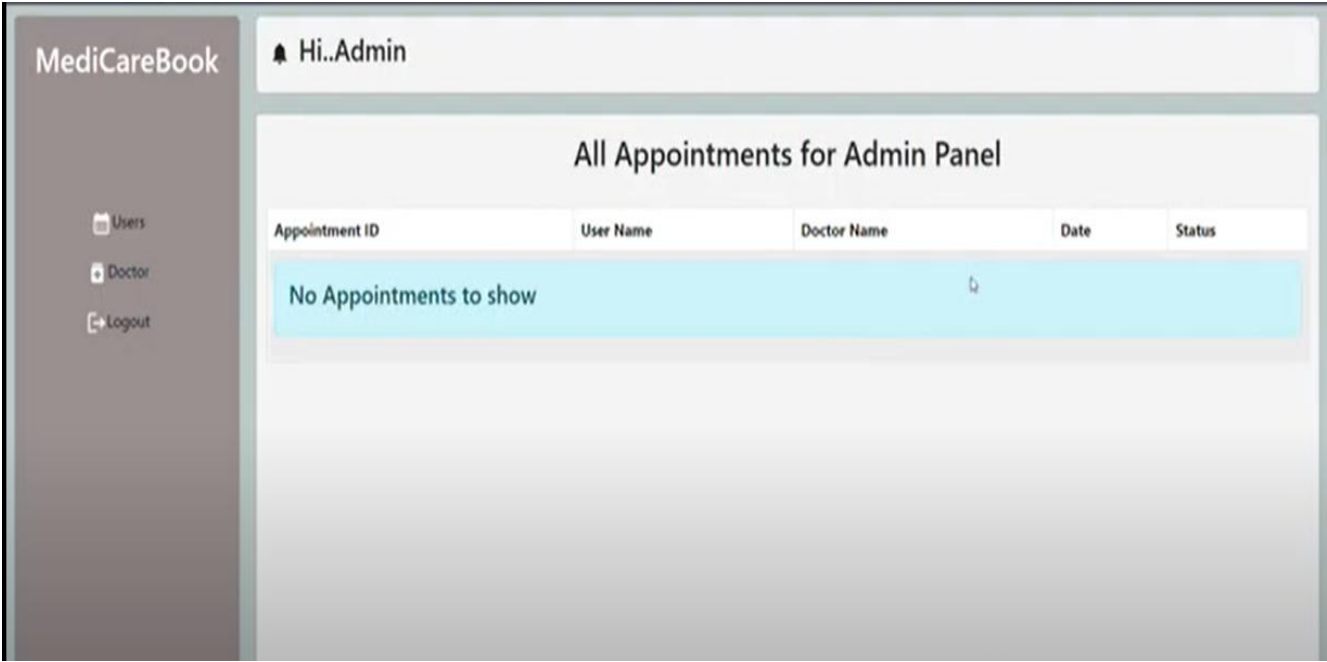
Email

Password

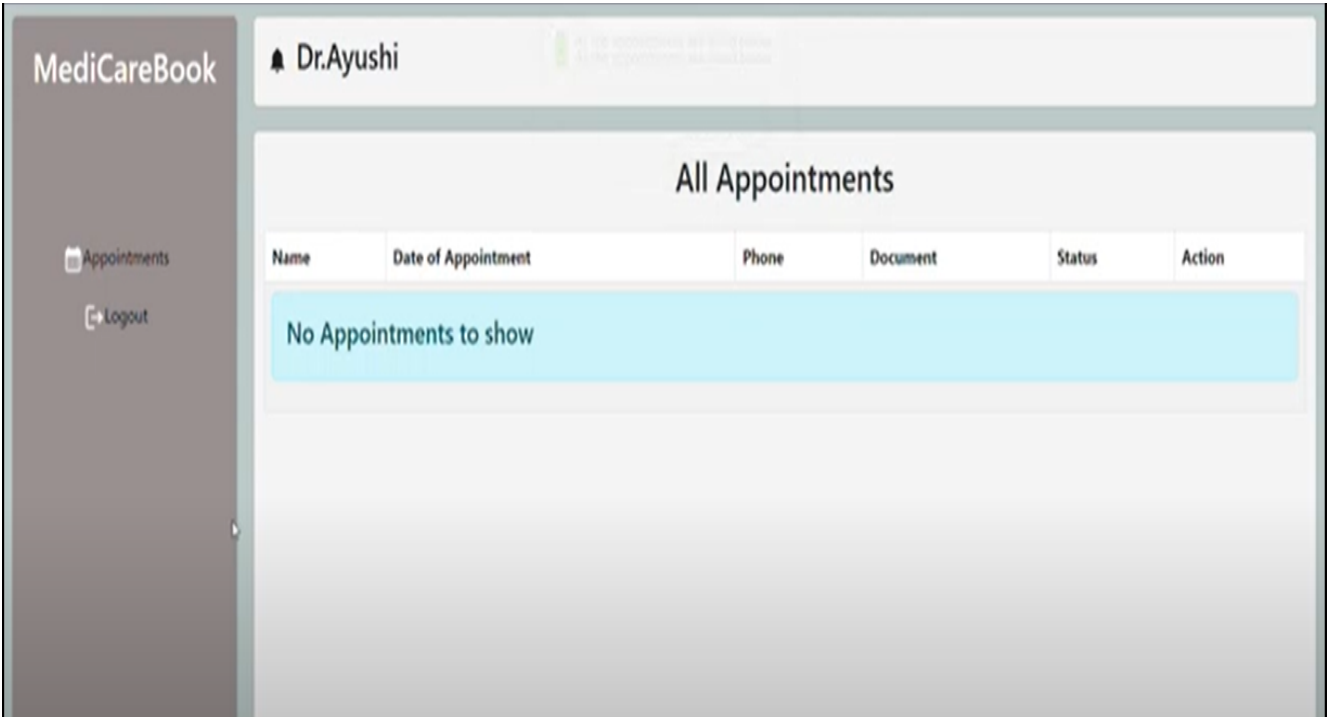
[Don't have an account? Register here](#)



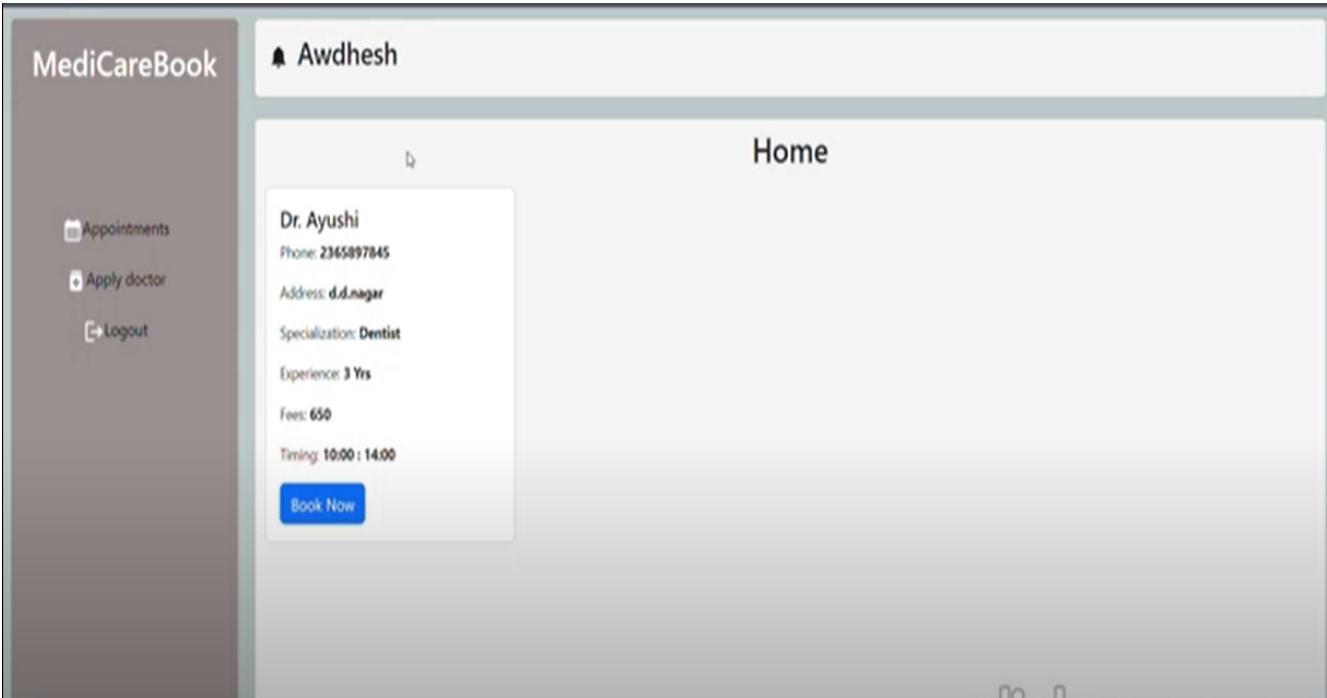
Admin Dashboard:



Doctor Dashboard:



User Dashboard:



CHAPTER 9. Challenges Faced

1. User Role Management:

- Developing a secure role-based access control system to ensure that users (patients, doctors, and admins) only have access to their permitted features.
- Handling edge cases where users might attempt to access features meant for other roles, requiring extensive testing and debugging.
- Implementing seamless transitions between roles (e.g., from a patient to a doctor) without affecting existing user data.

2. Database Design:

- Designing relationships between collections (e.g., Users, Appointments, Doctors, and Patients) to minimize redundancy and ensure scalability.
- Balancing schema flexibility with query performance to accommodate varying user and appointment details.
- Managing database migrations and updates as the application evolved during development.

3. Frontend Design:

- Creating a responsive design that adapts to different devices and screen sizes.
- Ensuring that the UI is intuitive and provides clear feedback to users, particularly during actions like booking appointments or viewing doctor details.
- Handling performance issues when rendering large datasets (e.g., doctor availability or appointment history) on the client side.

4. Backend Optimization:

- Building APIs that can handle high traffic efficiently, especially for appointment scheduling and user management.
- Ensuring proper error handling and debugging for asynchronous operations involving database queries and external APIs.

5. Testing and Debugging:

- Performing end-to-end testing for features like appointment booking, doctor availability, and user registration.
 - Identifying and resolving bugs, especially in edge cases like double-booking or overlapping appointment requests.
-

CHAPTER 10. Future Enhancements

1. Mobile Compatibility:

- Develop a dedicated mobile app for iOS and Android using frameworks like React Native or Flutter to provide better accessibility for patients and doctors on the go.
- Optimize the user interface for mobile devices, ensuring it is user-friendly and easy to navigate.
- Add offline functionality to allow users to view their appointment history, doctor profiles, and other essential data without an internet connection.

2. Advanced Search and Filter:

- Allow patients to search for doctors based on more specific criteria such as specialty, experience, available timings, and patient ratings.
- Integrate geolocation-based searching to display doctors near the user's current location.
- Implement dynamic filtering, where the results are updated in real-time as patients adjust search criteria like doctor availability or specialization.

3. Payment Integration:

- Integrate secure payment gateways such as Stripe or PayPal, enabling patients to pay for appointments, consultation fees, or deposits directly through the platform.
- Support multiple currencies for international users and provide automated receipt generation, including invoices and payment confirmation emails.

4. In-App Chat:

- Implement a messaging feature that allows patients to communicate directly with doctors in real-time, using WebSocket or libraries like Socket.io for seamless communication.
- Enable file sharing through chat for the exchange of medical records, prescriptions, or appointment-related documents.

5. AI-Powered Recommendations:

- Utilize machine learning algorithms to recommend doctors based on a patient's medical history, preferences, and previous appointment records.
 - Predict optimal consultation times and doctor availability, making suggestions based on both patient and doctor schedules.
 - Implement AI-based patient categorization to prioritize urgent cases or high-risk patients for faster responses and attention from medical professionals.
-

CHAPTER 11. Conclusion

The **Book a Doctor Using MERN** application successfully connects patients with healthcare providers, offering an efficient platform for browsing doctors, booking consultations, and managing appointments. Built on the MERN stack, the application ensures high performance, scalability, and a smooth user experience. Key features, including secure authentication, real-time notifications, and role-based access control, contribute to a secure and seamless environment for both patients and doctors.

Through thorough planning, development, and testing, the application delivers a reliable healthcare booking solution that reduces friction for patients seeking medical consultations. The user-friendly interface, advanced search filters, and appointment management tools enhance the overall experience, while the admin panel ensures smooth operation and security for the platform.

This project demonstrates the power of modern web technologies to address real-world challenges in the healthcare industry, improving access to medical services and streamlining the booking process. The integration of secure payment methods and in-app communication will further enhance user satisfaction and make the platform even more effective.

Future improvements, including mobile compatibility, machine learning-powered recommendations, and additional features like chat, are expected to expand the functionality of the app, making it even more accessible and efficient. Overall, **Book a Doctor Using MERN** is a successful implementation that paves the way for innovation in the online healthcare booking space.

CHAPTER 12.Reference

1. React.js Official Documentation

- <https://reactjs.org/>
Comprehensive documentation for building user interfaces using React.

2. Node.js Official Documentation

- <https://nodejs.org/en/docs/>
Detailed guides for building server-side applications with Node.js.

3. Express.js Guide

- <https://expressjs.com/>
Documentation for the Express framework, essential for handling backend routing and middleware.

4. MongoDB Manual

- <https://www.mongodb.com/docs/manual/>
Database design principles and best practices for managing collections and queries.

5. Mongoose.js Documentation

- <https://mongoosejs.com/docs/>
ORM for MongoDB to simplify schema creation and CRUD operations.

6. Material UI

- <https://mui.com/>
A React component library for creating modern and responsive UIs.

7. Ant Design

- <https://ant.design/>
A design system and component library for React-based applications.

8. Bootstrap

- <https://getbootstrap.com/>
A popular CSS framework for responsive design and styling.

9. JSON Web Tokens (JWT)

- <https://jwt.io/introduction/>
Overview and use cases of JWT for secure authentication.

10. W3Schools (HTML, CSS, JavaScript Basics)

- <https://www.w3schools.com/>
Beginner-friendly resources for learning web development basics.