



AALIM MUHAMMED SALEGH COLLEGE OF ENGINEERING

Approved by All India Council for Technical Education - New Delhi, Affiliated to Anna University, Chennai
NAAC Accredited Institution

"Nizara Educational Campus", Muthapudupet, Avadi - IAF, Chennai - 600 055.

ANNA UNIVERSITY COUNSELLING CODE : 1101

NBA ACCREDITED COURSES (Mech Engg, ECE, CSE & IT)



TITLE: BOOK A DOCTOR USING MERN

Department of Computer Science and Engineering

Submitted By

1.SATHISH KUMAR M (110121104085)

2.SYED TAHA HUSSAIN S (110121104099)

3.PRADEEP KUMAR D (110121104075)

4. MOHAMMED MAAZ T (110121104062)



AALIM MUHAMMED SALEGH COLLEGE OF ENGINEERING

Approved by All India Council for Technical Education - New Delhi, Affiliated to Anna University, Chennai
NAAC Accredited Institution

"Nizara Educational Campus", Muthapudupet, Avadi - IAF, Chennai - 600 055.

ANNA UNIVERSITY COUNSELLING CODE : 1101

NBA ACCREDITED COURSES (Mech Engg, ECE, CSE & IT)



BONAFIDE CERTIFICATE

Certified that this project report on “BOOK A DOCTOR USING MERN” is the Bonafide record of work done by Syed Taha Hussain.S(110121104099), Pradeep Kumar.D (110121104075), Mohammed Maaz.T(110121104062), Sathish Kumar.M (110121104085)

From the Department of Computer Science and Engineering by Anna University, Chennai

Internal Guide

Head of the Department

Internal Examiner

External Examiner

Abstract:

The Doctor Booking App is an advanced, web-based platform built using the MERN stack (MongoDB, Express.js, React.js, and Node.js), designed to streamline the process of booking doctor appointments and managing healthcare needs. With the growing demand for convenient and efficient healthcare services, this application aims to simplify the booking experience for patients while offering a comprehensive solution for healthcare providers to manage their appointments and patient interactions.

Core features of the application include user registration, browsing available doctors, detailed doctor profiles, and secure appointment scheduling. Patients can search for doctors based on specialty, location, and availability, while doctors can manage their schedules, patient records, and appointments. React.js provides an intuitive, dynamic, and responsive user interface, while Node.js and Express.js handle backend logic and API requests. MongoDB is used to store and retrieve user information, doctor profiles, appointment schedules, and patient histories.

The Doctor Booking App not only leverages the MERN stack to provide a fast and scalable solution for real-time appointment management but also aims to enhance the patient experience with features like appointment reminders, medical history tracking, and direct communication with doctors. This application serves as a robust foundation for a user-friendly healthcare management platform, supporting the modern, digital-driven approach to patient care and doctor availability.

With future possibilities for expanding the application to include features such as online consultations, patient reviews, and prescription management, the Doctor Booking App is poised to meet the evolving demands of both patients and healthcare providers in an increasingly digital healthcare landscape.

Introduction:

In today's fast-paced world, accessing healthcare services quickly and efficiently is more important than ever. Traditional methods of scheduling appointments, whether through phone calls or in-person visits, often involve long wait times and limited accessibility. With the increasing demand for convenient and streamlined healthcare solutions, the rise of digital platforms has provided a solution to these challenges, allowing patients to book appointments with healthcare professionals from the comfort of their homes.

The Doctor Booking App is designed to capitalize on this growing need for efficient healthcare scheduling by providing a seamless, web-based platform for patients to browse, book, and manage doctor appointments. This application leverages the power of the MERN stack (MongoDB, Express.js, React.js, and Node.js) to deliver a scalable, responsive, and user-friendly experience, making healthcare services more accessible and manageable for both patients and healthcare providers.

With a focus on user convenience, the Doctor Booking App brings together the convenience of online scheduling with the need for efficient healthcare management. By integrating key features such as secure appointment booking, detailed doctor profiles, and real-time availability, the platform ensures that users can easily find and schedule appointments with the right doctors at the right time.

The project not only showcases the flexibility and power of the MERN stack but also addresses the practical challenges faced by patients and healthcare providers in today's fast-evolving medical landscape. By offering a platform that supports a wide range of specialties, real-time appointment updates, and patient management tools, the Doctor Booking App aims to make healthcare more accessible, efficient, and user-friendly.

Through this platform, the project seeks to contribute to the digital transformation of the healthcare experience, empowering patients to take control of their healthcare needs while enabling healthcare providers to manage their appointments and patient interactions more effectively.

Scenario based Case study:

John is a busy professional working long hours in a fast-paced industry. His demanding job often leaves him with little time to manage personal tasks, including booking doctor's appointments. While he values his health, John finds it challenging to schedule doctor visits around his hectic work schedule. He often experiences frustration with traditional methods of booking appointments, such as calling offices during business hours, waiting on hold, and dealing with time-consuming paperwork at the clinic. This inefficiency results in delayed appointments, missed follow-ups, and unnecessary stress

Problem Statement:

The healthcare industry is increasingly moving towards digital solutions as patients and healthcare providers seek more efficient, accessible ways to manage appointments and medical care. However, existing doctor booking platforms often fall short in providing a seamless, user-friendly experience for both patients and doctors. Users face challenges such as complicated interfaces, lack of real-time availability, limited communication options, and cumbersome appointment management systems. Additionally, many platforms fail to scale effectively to accommodate a large user base and diverse healthcare needs.

This project addresses the need for an efficient and accessible **Doctor Booking App** that simplifies the appointment scheduling process for patients while streamlining the appointment management system for healthcare providers. The primary challenges include:

1. **User Accessibility:** Ensuring an intuitive, easy-to-navigate interface for patients of all backgrounds to browse doctor profiles, select specialties, and schedule appointments effortlessly.
2. **Seamless User Management:** Implementing a secure and efficient registration and login system to store patient details, appointment history, and preferences, as well as offering an easy way for doctors to manage their schedules and patient records.
3. **Real-time Availability and Scheduling:** Providing real-time, up-to-date doctor availability to avoid scheduling conflicts and ensure that patients can book appointments without delays.
4. **Efficient Appointment Management:** Developing a system that allows users to manage appointments, receive reminders, and easily reschedule or cancel appointments with minimal friction.
5. **Communication and Support:** Offering enhanced communication features, such as appointment reminders, in-app messaging between patients and doctors, and easy access to medical records or consultation histories.
6. **Responsive Performance:** Ensuring a responsive application that performs seamlessly across multiple devices, supports scalability to handle a growing number of users, and provides a smooth experience even during peak usage.

By addressing these challenges, the **Doctor Booking App** aims to provide a superior user experience, allowing patients to find and schedule appointments quickly and conveniently, while helping healthcare providers manage their appointments and patient interactions more effectively. This project will demonstrate the power of the MERN stack in creating a reliable, scalable, and user-centric digital healthcare platform.

Project Scope:

Target Audience:

Target Audience: The Doctor Booking App is designed to cater to patients, healthcare providers, and medical professionals seeking a more efficient, accessible, and streamlined way to manage medical appointments. The platform is ideal for:

- **Busy Professionals** who need to schedule doctor visits around their work schedules, often finding it difficult to book appointments during traditional office hours.
- **Parents and Caregivers** who need an easy way to manage appointments for their children or elderly relatives, ensuring timely access to healthcare services.
- **Chronic Patients** who require regular check-ups or treatments and need a convenient system to manage recurring appointments.
- **Tech-Savvy Patients** who are comfortable with digital platforms and prefer booking healthcare services online rather than calling clinics or waiting in long lines.
- **Healthcare Providers (Doctors, Specialists, Clinics)** who need a streamlined way to manage patient appointments, reduce no-shows, and communicate effectively with their patients.

Use Cases:

- ❑ **User Registration and Profile Creation:** A patient can create an account, providing essential personal details and preferences to streamline the appointment booking process.
- ❑ **Doctor Search and Filter:** A patient can search for doctors based on specialty, location, availability, and ratings, helping them make an informed choice quickly.
- ❑ **Appointment Scheduling:** Patients can book appointments in real-time, selecting available time slots that fit their schedule and receive immediate confirmation.
- ❑ **Appointment Reminders and Notifications:** Users receive automatic reminders and updates about upcoming appointments, cancellations, or schedule changes, ensuring they stay informed.
- ❑ **Doctor Dashboard:** Healthcare providers can manage their availability, view patient appointments, and maintain a digital record of their consultations, making it easier to manage their practice.
- ❑ **Secure Payment:** Patients can pay for consultations through a secure online payment system, reducing the need for physical payments at the clinic.
- ❑ **Appointment History and Medical Records:** Patients can view their appointment history, past consultations, and medical records, giving them easy access to important health information.

Project Setup and Configuration:

1. Install required tools and software:

- a) Node.js
- b) Mango DB
- c) Create-react-App

2. Create project folders and files:

- a. Client folders
- b. Server folders

3. Install Packages:

a. Frontend npm Packages

- i. Axios
- ii. React-Router –dom
- iii. Bootstrap
- iv. React-Bootstrap

b. Backend npm Packages

- i. Express
- ii. Mongoose
- iii. cors

Frontend Development:

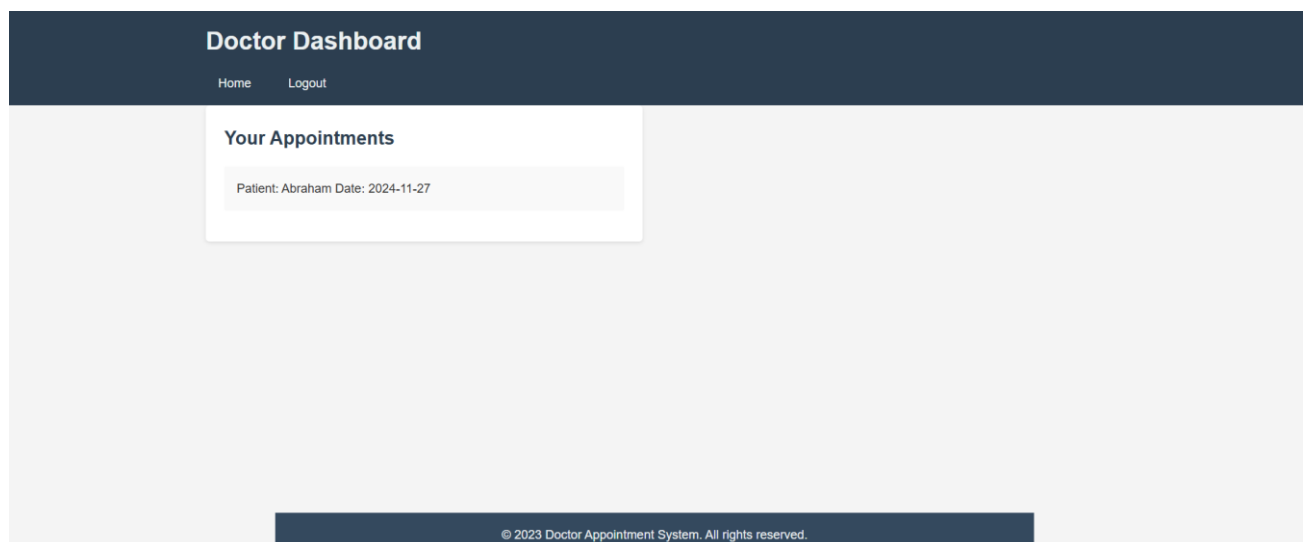
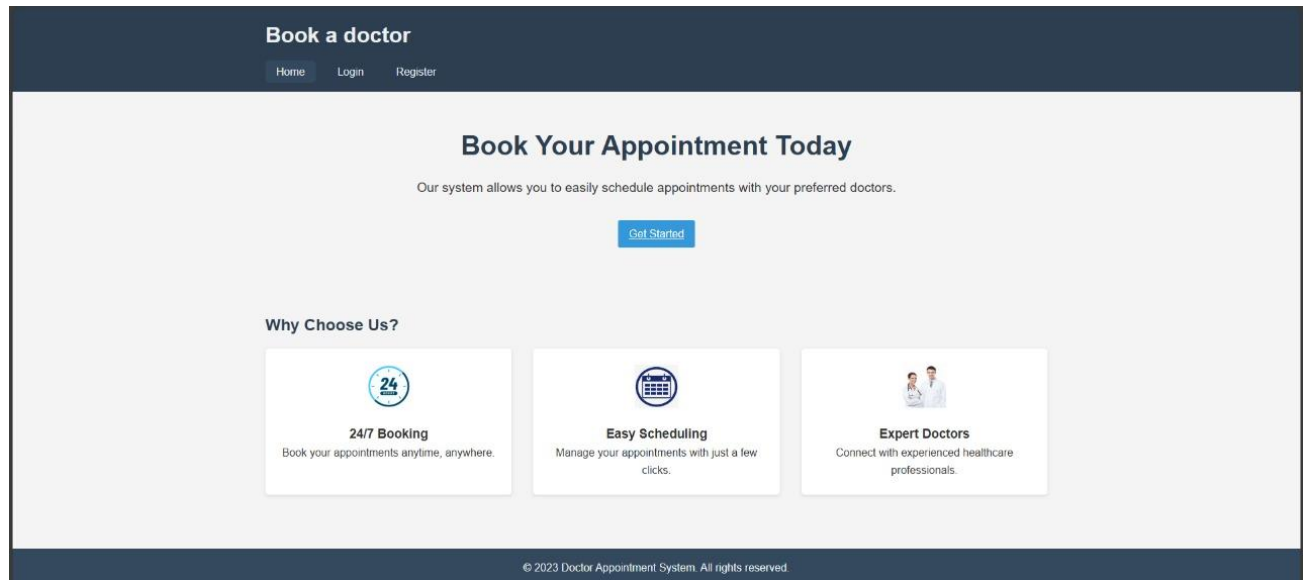
• Setup React Application:

- a. Create React application.
- b. Configure Routing.
- c. Install required libraries.

• Design UI components:

- a. Create Components.
- b. Implement layout and styling.
- c. Add navigation.

- **Implement frontend logic:**
 - a. Integration with API endpoints.
 - b. Implement data binding.



Backend Development:

- **Setup express server**
 - a. Create index.js file in the server (backend folder).
 - b. Create a .env file and define port number to access it globally.
 - c. Configure the server by adding cors, body-parser.
- **User Authentication:**
 - a. Create routes and middleware for user registration, login, and logout.
 - b. Setup authentication middleware to protect routes that require user authentication.
- **API Routes:**
 - a. Create separate route files for different API functionalities such as user's orders, and authentication.
 - b. Define the necessary routes for listing products, handling user registration and login, managing orders, etc.
 - c. Implement route handlers using Express.js to handle requests and interact with the database.
- **Implement Data Models:**
 - a. Define Mongoose schemas for the different data entities like products, users, and orders.
 - b. Create corresponding Mongoose models to interact with the MongoDB database.
 - c. Implement CRUD operations (Create, Read, Update, Delete) for each model to perform database operations.
- **Error Handling:**
 - a. Implement error handling middleware to catch and handle any errors that occur during the API requests.
 - b. Return appropriate error responses with relevant error messages and HTTP status codes.

Database:

- **Configure MongoDB:**

- a. Install Mongoose.
- b. Create database connection.
- c. Create Schemas & Models.

- **Connect database to backend:**

Now, make sure the database is connected before performing any of the actions through the backend. The connection code looks similar to the one provided below.

```
//  
const PORT = process.env.PORT || 6001;  
mongoose.connect(process.env.MONGO_URL, {  
  useNewUrlParser: true,  
  useUnifiedTopology: true  
}).then(()=>{  
  
  server.listen(PORT, ()=>{  
    console.log(`Running @ ${PORT}`);  
  });  
  
}).catch((err)=>{  
  console.log("Error: ", err);  
})
```

- **Configure Schema:**

Firstly, configure the Schemas for MongoDB database, to store the data in such pattern. Use the data from the ER diagrams to create the schemas. The Schemas for this application look alike to the one provided below.

```
JS User.js  X  
server > models > JS User.js > ...  
1  import mongoose from 'mongoose';  
2  
3  const UserSchema = new mongoose.Schema({  
4    username:{  
5      type: String,  
6      require: true  
7    },  
8    email:{  
9      type: String,  
10     require: true,  
11     unique: true  
12   },  
13   password:{  
14     type: String,  
15     require: true  
16   },  
17 });  
18  
19 const User = mongoose.model("users", UserSchema);  
20 export default User;
```

Software Requirement:

Code Editor:

Visual Studio Code (VS Code): A lightweight and powerful code editor that supports JavaScript, React.js, Node.js, and other web technologies. It includes useful features like IntelliSense, debugging, and integrated terminal, making it ideal for MERN stack development.

Node.js:

A runtime environment for JavaScript that allows developers to run JavaScript code outside the browser. It is used to build the backend of the application with Express.js.

MongoDB:

A NoSQL database to store user data, book details, and purchase history. MongoDB's flexible document-based storage is perfect for handling the dynamic structure of e-books and user information.

npm (Node Package Manager):

The default package manager for Node.js, which helps manage project dependencies like React, Express, and other libraries used in the project.

Express.js:

A web application framework for Node.js used to create the API that handles routing, requests, and interactions between the frontend and database.

React.js:

A JavaScript library for building user interfaces, particularly single-page applications. React will be used to build the frontend of the E-book Store, ensuring a dynamic and responsive user experience.

Postman:

A powerful API testing tool used for testing the Express.js API endpoints. It helps in debugging and ensuring that the backend logic is working as expected.

Git and GitHub:

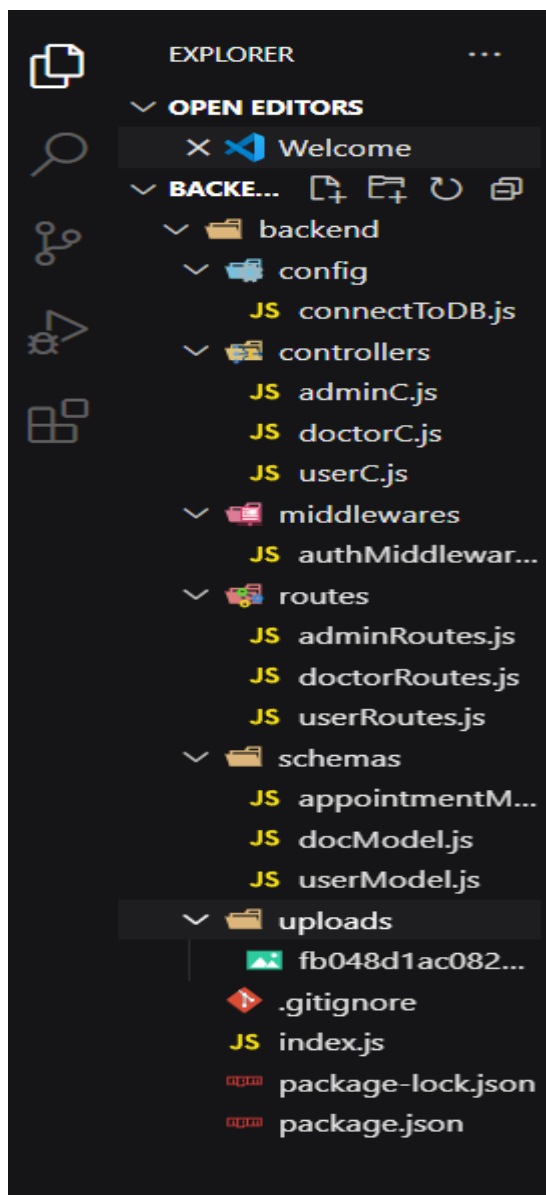
Git: A version control system used to manage and track changes to the project files.

GitHub: A cloud-based platform to host the project's code, enabling collaboration and version control.

Google Chrome / Mozilla Firefox:

These are popular web browsers for testing and running the application in real-time to ensure the frontend's responsiveness and functionality.

Project Structure:



▼ **BOOK-STORE**

▼ Backend

▼ db

> Seller

> Users

JS config.js

> node_modules

> uploads

{} package-lock.json

{} package.json

JS server.js

▼ frontend

> node_modules

> public

▼ src


> Admin

> Componenets


> Seller


> User


App.css

 App.jsx

index.css

 main.jsx

 .eslintrc.cjs

 .gitignore

<> index.html

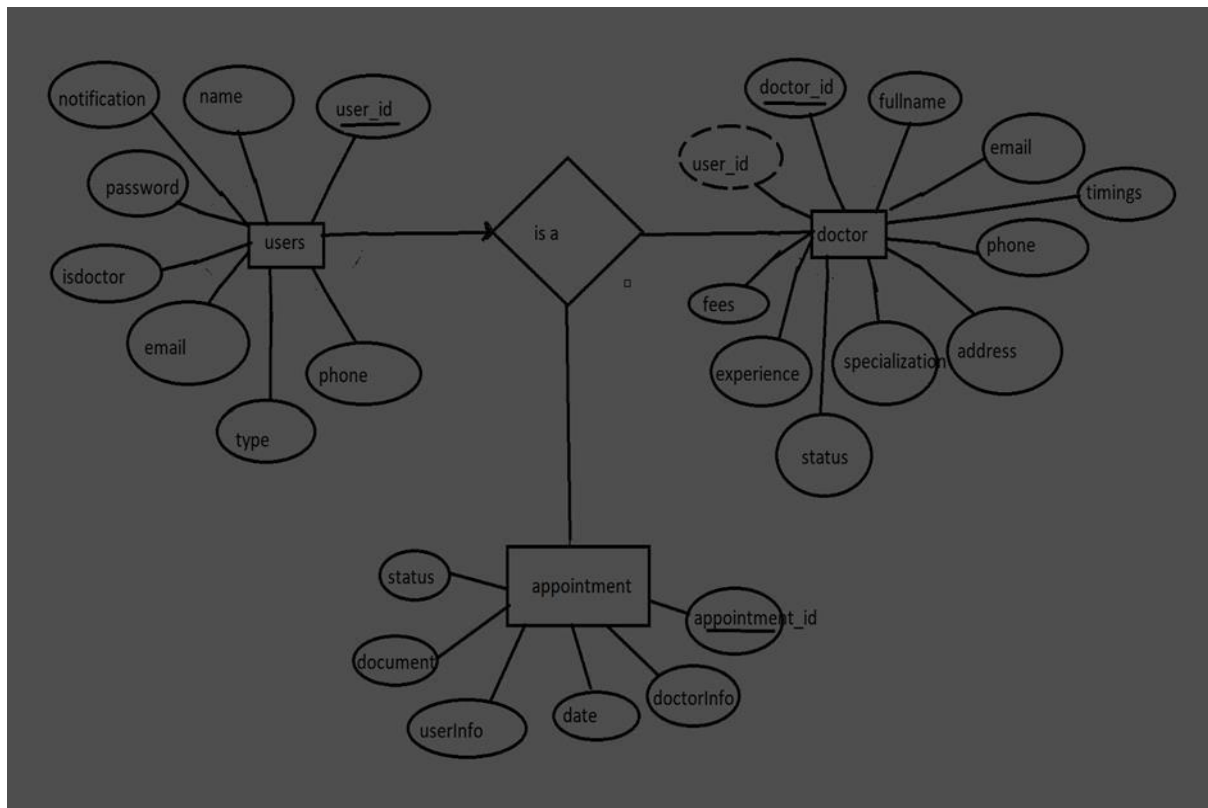
{} package-lock.json

{} package.json

 README.md

JS vite.config.js

E-R DIAGRAM:



User-Doctor Relationship:

- **Many-to-Many (M:M):** A single user (patient) can book appointments with multiple doctors, and a doctor can see many different patients.
- **Implementation:** Introduce an intermediate entity, "Appointment", with foreign keys to both the User and Doctor tables. This table could store additional information like appointment date, time, status (e.g., completed, canceled), and the reason for the visit.

Doctor-Specialty Relationship:

- **One-to-Many (1:M):** A single doctor can specialize in one or more specialties, but each specialty can have multiple doctors.
- **Implementation:** Maintain a separate "Specialty" table with fields like Specialty Name and Doctor ID (foreign key) to track the doctor's specialties.

User-Appointment Relationship:

- **One-to-Many (1:M):** A single user (patient) can have multiple appointments, but each appointment belongs to one user.
- **Implementation:** Keep the User ID as a foreign key in the Appointment table to track the patient's appointment history.

Doctor-Appointment Relationship:

- **One-to-Many (1:M):** A single doctor can handle multiple appointments, but each appointment is associated with only one doctor.
- **Implementation:** Store the Doctor ID as a foreign key in the Appointment table to track the appointments a doctor is responsible for.

User-Review Relationship:

- **One-to-Many (1:M):** A single user can leave reviews for multiple doctors, but each review is linked to one user.
 - **Implementation:** Maintain a "Review" table with fields like ReviewID, UserID (foreign key), DoctorID (foreign key), rating, and review text.
-

User-Payment Relationship:

- **One-to-Many (1:M):** A single user can make multiple payments for various appointments, but each payment belongs to one user.
 - **Implementation:** Store payment information in the "Payment" table, which links to the User table through a foreign key and includes payment details such as amount, payment method, and appointment ID.
-

Doctor-Schedule Relationship:

- **One-to-Many (1:M):** A single doctor can have multiple available time slots, but each time slot is linked to one doctor.
 - **Implementation:** Keep a separate "Schedule" table with fields like DoctorID (foreign key), date, start time, end time, and availability status.
-

Doctor-Clinic Relationship:

- **One-to-Many (1:M):** A doctor can practice in multiple clinics, but each clinic can have many doctors.
- **Implementation:** Maintain a "Clinic" table with fields like Clinic Name, Location, and DoctorID (foreign key) to associate doctors with clinics.

Working of the System:

1. User Registration & Authentication:

- **User Sign-up & Login:**
 - A user (patient) can sign up or log in through a secure registration process by submitting their email and password.
 - On successful registration, the user's data (email, password) is stored in MongoDB. Passwords are hashed using **Bcrypt.js** to ensure security.
 - A **JWT (JSON Web Token)** is generated upon successful login, which is sent to the frontend. This token is used for authenticating the user for subsequent requests.
-

2. Doctor Search & Browse:

- **Home Page:**
 - After logging in, the user is directed to the **home page**, which displays a list of available doctors.
 - The frontend (React.js) makes API calls to the backend to fetch doctor data (name, specialty, location, availability, and ratings).
 - The user can search and filter doctors by specialty, location, and availability to find the best match for their needs.
-

3. Doctor Profile Page:

- **Doctor Details:**
 - When a user clicks on a doctor, the system shows detailed information about the doctor (bio, specialties, clinic location, reviews, and available time slots).
 - The frontend fetches this data via an API call to display the detailed information to the user.
-

4. Appointment Scheduling:

- **Book Appointment:**
 - The user can select an available time slot and book an appointment with the doctor.
 - The system checks the availability and updates the backend to reflect the new appointment.
 - The frontend dynamically updates the user interface to show confirmation of the appointment, including the time and doctor details.
-

5. Appointment Confirmation & Notifications:

- **Confirmation & Reminders:**
 - After the appointment is booked, a confirmation page is displayed to the user with the details of their upcoming appointment.
 - The system sends **email/SMS reminders** to both the user and the doctor, reducing the chances of missed appointments.
 - Appointment details are saved to the user's profile in the backend, which can be accessed later.
-

6. User Profile & Appointment History:

- **Manage Profile:**
 - Users can view and manage their profile (personal details, medical history, etc.) through the app.
 - The system stores the patient's medical records, previous appointments, and doctor interactions in the backend, allowing users to track their healthcare journey.
-

7. Doctor Dashboard (Admin Panel for Doctors):

- **Manage Schedule:**
 - Doctors can log in to their dashboard and manage their available time slots.
 - The doctor can view patient appointments, update availability, and manage consultation history.
 - Doctors can also track appointments, patient feedback, and manage their schedules through the dashboard.
-

8. Payment (Optional Feature):

- **Payment Integration:**
 - After scheduling an appointment, patients can be directed to a payment page to complete the transaction.
 - **Payment gateway integration** (e.g., PayPal, Stripe) can be used to securely process payments.
 - Upon successful payment, an order (appointment) is confirmed, and the details are saved in the user's profile.
-

9. Technologies in Action:

- **Frontend (React.js):**
 - **React components** are used to create a dynamic and responsive user interface, with React Router to manage navigation between pages (home, doctor profile, booking, profile, etc.).
 - **State Management** using **React Context** or **Redux** is used for managing the cart (appointments), user data, and other application states.
 - **Backend (Node.js, Express.js, MongoDB):**
 - **Node.js** and **Express.js** are used to handle the backend logic, API routes, and user requests.
 - **MongoDB** stores user data, doctor profiles, appointments, reviews, and payment details in a scalable manner.
-

10. Security Features:

- **Authentication:**
 - **JWT (JSON Web Tokens)** are used for secure authentication. Once the user logs in, the token is sent to the frontend and used for verifying protected routes (such as booking appointments or managing profiles).
- **Password Security:**
 - User passwords are hashed with **Bcrypt.js** to ensure they are securely stored in the database.

Application Flow:

- 1. Start**
- 2. Search Doctors**
- 3. Select Specialty**
- 4. Select Doctor**
- 5. Add to Cart.**
- 6. Order Appointment**
- 7. View Bookings**
- 8. End.**

The project has 2 type of user – Customer and Doctor and other will be Admin which takes care to all the user. The roles and responsibilities of these two types of users can be inferred from the API endpoints defined in the code.

Here is a summary: Customer/Ordinary:

1. Create an account and log in to the system using their email and password.
2. They will be shown automatically all the doctors in their dashboard.
3. After clicking on the Book Now, a form will generate in which date of appointment and documents need to send.
4. They can sees the status of their appointment and can get a notification if the appointment is schedule or not.
5. The user can also cancel it`s booking in booking history page and can change the status of booking.

Admin:

1. Manage and monitor the overall operation of the appointment and the type of users and doctors to the application.
2. He monitors the applicant of all doctors and approve them and then doctors are registered in the app.
3. Implement and enforce platform policies, terms of service, and privacy regulations.

Doctor:

1. Gets the approval from the admin for his doctor account.
2. Manages all the appointments that are getting from the users

Conclusion:

The **Doctor Booking App** using the MERN Stack demonstrates the effective use of modern web technologies to build a comprehensive, user-friendly, and secure platform for booking doctor appointments. Leveraging the MERN stack—MongoDB, Express.js, React.js, and Node.js—the application ensures seamless communication between the frontend and backend, creating an efficient, responsive, and interactive experience for both patients and healthcare providers.

Seamless User Experience:

The **Doctor Booking App** offers an intuitive and responsive interface built with React.js, enabling users to easily search for doctors, book appointments, and manage their healthcare needs. The user-friendly design ensures that patients can effortlessly navigate through the platform, enhancing the overall satisfaction and accessibility of the service.

Efficient and Scalable Data Management:

Using **MongoDB** as the database provides flexibility and scalability, allowing the system to handle large volumes of doctor profiles, appointments, and patient data efficiently. The app is designed to scale with ease, ensuring a smooth experience even as the number of users and healthcare providers grows.

Secure User Authentication:

JWT authentication ensures that all user accounts and personal data are securely managed. The login system verifies user identities and protects sensitive medical information, maintaining privacy and safeguarding transactions for both patients and doctors.

Real-time Appointment Scheduling and Notifications:

The app enables users to check doctor availability, schedule appointments in real time, and receive timely reminders. This feature streamlines the appointment process, reducing no-shows and enhancing patient satisfaction.

Admin Control and Management:

The admin panel allows doctors and administrators to manage their schedules, monitor appointments, and track patient feedback. It offers complete control over the platform, ensuring that users and doctors have up-to-date, accurate information.

Innovative Features for Future Enhancements:

Telemedicine Integration:

- Incorporating video consultations and remote healthcare services to allow patients to consult with doctors virtually, expanding the app's functionality.

AI-Driven Health Recommendations:

- Integrating machine learning algorithms to suggest personalized health tips, doctor recommendations, and appointment reminders based on patient history and preferences.

Patient and Doctor Feedback Systems:

- Adding robust feedback mechanisms for both patients and doctors, enabling better service quality and helping users make informed decisions.

Mobile App Integration:

- Developing a mobile app version of the platform to offer greater flexibility and access for users on the go.

Advanced Analytics and Health Insights:

- Providing patients with detailed reports on their medical history, treatment progress, and appointment trends, helping them better manage their health.