Full Stack Development with MERN

Project Documentation

1. INTRODUCTION:

Project Title: Book a doctor

Team Members-NM ID

Nandhagopal.T	-	E1BCEC6B604F9D78565483DDB17D98D6
Praveen kumar. J	-	2D018F644D9087B9B074EE2B746DFE5E
Saran Raj. V	-	4E7ACE5BE42B8DCB57B4458D30815459
Venkatesh Prasath. A	-	1239A587BCEFEF296A54A3B73399BD5D

---

2. PROJECT OVERVIEW:

Purpose:

The Doctor Appointment Booking System is a web application built using the MERN stack (MongoDB, Express.js, React.js, and Node.js). The platform facilitates seamless communication between patients and doctors by enabling patients to book, reschedule, or cancel appointments while allowing doctors to manage their schedules efficiently. The application aims to digitize and simplify the traditional appointment booking process.

- To provide a user-friendly platform for booking doctor appointments.
- To facilitate doctors in managing their availability and appointments.
- To minimize manual errors and reduce time consumption in booking processes.
- To provide secure authentication and data management for users.
- To offer scalability for future enhancements such as video consultations.

FEATURES:

For patients:

- User Registration/Login: Secure authentication system.
- Search Doctors: Search by name, specialization, or location.

.

- Book Appointment: Select a date and time to book an appointment.
- Manage Appointments: View, reschedule, or cancel appointments.

2.For doctors:

- Schedule Management: Set availability and view booked appointments.
- Doctor Registration/Login: Create and manage profiles securely.
- Notification System: Get notifications for new bookings or cancellations.

3. Admin Panel:

- User Management: Monitor and manage registered users.
- Doctor Management: Verify and approve doctors' profiles.
- Appointment Logs: Track all appointments for auditing purposes.

---

3. ARCHITECTURE:

FRONTEND:

Framework: React.js

Tools: React Router, Axios, Material-UI, Bootstrap

BACKEND:

Framework: Node.js

Middleware: Express.js

Tools: bcrypt (password hashing), JSON Web Token (authentication)

DATABASE:

MongoDB for storing user data, doctor schedules, and appointments.

---

4. SETUP INSTRUCTION:

- **PREREQUISITE:**

  Before setting up the application, ensure the following software dependencies are installed:

  1. Node.js: Install the latest LTS version (recommended) from [Node.js official website](https://nodejs.org/).

     - Verify installation:

       ```bash

       node -v

       npm -v

       ```

  2. MongoDB: Install MongoDB Community Edition or use a cloud-hosted service like [MongoDB Atlas](https://www.mongodb.com/atlas).

     - Verify installation:

       ```bash

       mongodb --version

       ```

  3. Package Manager:

     - **NPM**: Comes with Node.js installation.

  4. Git: Install Git for version control from [Git official website](https://git-scm.com/).

     - Verify installation:

       ```bash
       ```

git --version

```

5. Code Editor: Install a code editor like [Visual Studio Code](https://code.visualstudio.com/) for development.

6. Environment Variables Management: Use a tool or library like `dotenv` for managing environment variables.

- Additional Tools

  Node Package Dependencies:

  Install required dependencies for the project after cloning the repository:

  ```bash
  npm install
  ```

  Common dependencies include:

  - express: Web framework.
  - mongoose: MongoDB object modeling.
  - dotenv: Environment variable management.
  - cors, helmet: Middleware for security.
  - bcrypt, jsonwebtoken: For authentication.

With these prerequisites in place, you can proceed to set up the project locally or deploy it on your desired platform.
  ```

- INSTALLATION GUIDE:

  Follow these steps to set up the Grocery project on your local machine:

  1. Clone the Repository

  Download the project repository to your local machine:

  ```bash

  git clone https://github.com/TSF-Solutions-24/Naan_Mudhalvan

  cd Naan_Mudhalvan

  ```

  2. Install Dependencies

  Install the required Node.js dependencies on both frontend and backend:

  ```bash

  npm install

  ```

  3. Set Up Environment Variables

  Create a `.env` file in the project root directory and add the following variables:

  ```plaintext

  MONGO_URI=<your-mongodb-connection-string>

  JWT_SECRET=<your-secret-key>

  PORT=<desired-port-number, default: 3000>

  ```

- Replace `<your-mongodb-connection-string>` with your MongoDB connection URI.

- Replace `<your-secret-key>` with a strong secret key for JWT authentication.

- Optionally set a custom `<desired-port-number>` for the server.

---

4. Start the Server

Run the development server:

```bash

npm run dev

```

The server will start, and the application will be accessible at `http://localhost:<PORT>` (default: `3000`).

5. Test the Application

Open a browser or API testing tool like Postman and interact with the application to ensure everything is working correctly.

---

5. FOLDER STRUCTURE:

1. Users:

Fields: id, name, email, password, role (patient/doctor), contact, etc.

2. Doctors:

Fields: id, name, specialization, experience, availability, appointments, etc.

3. Appointments:

Fields: id, doctor_id, patient_id, date, time, status (booked/cancelled), etc.

---

6. RUNNING THE APPLICATION:

Follow these commands to start the frontend and backend servers locally:

1. Frontend:

Navigate to the frontend directory (Frontend) and start the React development server:

```bash
cd frontend
npm start
```

- The frontend server will run on `http://localhost:3000` by default.

2. Backend

Navigate to the backend directory (backend) and start the Node.js server:

```bash
cd backend
npm start
```

- The backend server will run on `http://localhost:5000` by default (or as configured in the `.env` file).

- **Ensure Both Servers are Running**

- Frontend interacts with the backend through API endpoints.

- Confirm both servers are up to experience the full functionality of the application.

---

7. API DOCUMENTATION:

1. Authentication Endpoints

1.1 Register User

POST /auth/register
Create a new user account.

Request Body

{

"name": "John Doe",

"email": "johndoe@example.com",

"password": "securepassword",

"role": "patient"

}

Response

{

"message": "User registered successfully.",

"userId": "12345"

}

1.2 Login User

POST /auth/login
Authenticate a user and get a token.

Request Body

{

"email": "johndoe@example.com",

"password": "securepassword"

}

Response

{

"token": "eyJhbGci...exampletoken...",

"user": {

"id": "12345",

"name": "John Doe",

"role": "patient"

}

}

2. Doctors

2.1 **Get All Doctors**.

GET /doctors
Retrieve a list of all doctors.

Query Parameters

- specialization (optional): Filter by specialization.
- location (optional): Filter by location.
- availability (optional): Filter by date/time.

2.2 **Get Doctor Details**

GET /doctors/:id
Retrieve details of a specific doctor.

Path Parameters

- id: Doctor's unique ID.

3. Appointments

3.1 Book Appointment

POST /appointments
Book an appointment with a doctor.

Request Body

{

"patientId": "12345",

"doctorId": "d1",

"dateTime": "2024-11-20T10:00:00",

"mode": "online"

}

Response

{

"message": "Appointment booked successfully.",

"appointmentId": "a12345"

}

3.2. Get Appointments

GET /appointments
Retrieve all appointments for a user.

Query Parameters

- userId (optional): Filter by user ID.

3.3. Cancel Appointment

DELETE /appointments/:id
Cancel a specific appointment.

Path Parameters

- id: Appointment ID

4. Payments

4.1. Initiate Payment

POST `/payments/initiate`
Initiate payment for an appointment.

Request Body

{

"appointmentId": "a12345",

"amount": 50.00,

"paymentMethod": "credit_card"

}

Response

{

"paymentId": "p12345",

"status": "pending",

"redirectUrl": "https://paymentgateway.com/checkout?p=p12345"

}

4.2. Check Payment Status

GET /payments/:id
Check the status of a payment.

Path Parameters

- id: Payment ID.

{

"paymentId": "p12345",

"status": "success"

}

5. Notifications

5.1. Get Notifications

GET /notifications
Retrieve notifications for a user.

Query Parameters

- userId (optional): Filter by user ID.

Response

[

{

"id": "n1",

"message": "Your appointment with Dr. Smith is confirmed.",

"dateTime": "2024-11-18T12:00:00"

}

]

6. Admin

6.1. Get All Users

GET /admin/users
Retrieve a list of all registered users. **Admin only**.

---

8. AUTHENTICATION AND AUTHORIZATION IMPLENTATION

1. Authentication

Authentication ensures that users (patients, doctors, or admin) can securely log in and access their accounts.

Steps to Implement Authentication

1.1. User Registration

- Users provide details such as name, email, password, and role (patient, doctor, or admin).
- Password Hashing: Use a library like bcrypt to hash passwords before storing them in the database.

Example Workflow:

1. User sends a POST request to /auth/register with their details.
2. Backend hashes the password and saves the user data to the database.
3. Backend responds with a success message and user details (excluding password).

1.2. User Login

- Validate user credentials and issue a JSON Web Token (JWT) for authenticated sessions.

Example Workflow:

1. User sends a POST request to /auth/login with email and password.
2. Backend verifies the email and password.
3. If valid, a JWT is generated and returned to the user.

1.3. JSON Web Token (JWT)

- Token Structure:
    - Header: Metadata about the token (e.g., algorithm).
    - Payload: User-specific data (e.g., userId, role).
    - Signature: Ensures token integrity using a secret key.

---

2. Authorization

Authorization determines what resources a user can access based on their role (patient, doctor, or admin).

Steps to Implement Authorization

2.1. Role-Based Access Control (RBAC)

Assign specific permissions to roles:

- Patient: Book appointments, view their appointments, access doctor profiles.
- Doctor: View appointments, update availability, manage patient records.
- Admin: Manage users, monitor activity, generate reports.

2.2. Middleware for Authorization

Create middleware to validate JWTs and check user roles.

2.3. Applying Middleware

Protect Routes:

- Add authenticateToken to all secured routes.
- Add authorizeRole for role-specific access.

---

3. Security Best Practices

3.1. Password Management

3.2. Secure Token Storage

- Store JWTs in HTTP-only cookies for enhanced security (to prevent XSS attacks).
- If using localStorage or sessionStorage, ensure secure handling on the client side.

3.3. Protect Sensitive Endpoints

- Implement rate limiting to prevent brute force attacks:

---

4. Example Endpoints with Authentication and Authorization

Protected Endpoint Example

GET /appointments (Requires authentication)

javascript

```
router.get ('/appointments', authenticateToken, (req, res) => {
  const userId = req.user. userId;
  // Fetch appointments for the authenticated user
});
```

Role-Based Access Example

POST /doctor/availability (Requires doctor role)

```
javascript
router.post ('/doctor/availability', authenticateToken, authorizeRole('doctor'), (req, res) => {
  const doctorId = req.user. userId;
  // Update doctor's availability
});
```

---

9. USER INTERFACE:

---

10. Testing

1.1. Unit Testing

Tests individual components or functions of the project, such as:

- Authentication (e.g., password hashing, JWT generation).
- Database operations (e.g., saving user data, fetching appointments).
- API endpoints.

1.2. Integration Testing

Ensures that different modules interact correctly, such as:

- User registration flows.
- Booking an appointment and updating the doctor's schedule.

1.3. System Testing

Tests the application as a whole to validate end-to-end workflows:

- From user login to booking a doctor and receiving a confirmation.

1.4. User Interface Testing

Validates the responsiveness and behavior of UI components:

- Navigation menus.
- Form validations.
- Button actions.

1.5. Performance Testing

Tests the system under high load, such as:

- Many users searching for doctors simultaneously.
- High appointment booking activity.

1.6. Security Testing

Ensures protection against vulnerabilities like:

- SQL injection.
- Cross-Site Scripting (

---

11. Screenshots or Demo

- Provide screenshots or a link to a demo to showcase the application.

12. KNOWN ISSUES

1. Authentication and Authorization

Issues:

1. JWT Expiration Handling:
    - If a token expires during a session, the user is logged out abruptly without proper notification.
2. Role-based Access Bugs:
    - Users may accidentally access unauthorized pages due to incorrect middleware implementation.
3. Forgot Password Flow:
    - Password reset emails may not send due to SMTP configuration errors.

---

2. Doctor Booking

Issues:

1. Double Booking:
    - Race conditions may allow multiple users to book the same time slot for a doctor.
2. Search Performance:
    - Searching for doctors with extensive filters may result in slow response times, especially with a large database.
3. Appointment Overlap:
    - Doctors may accidentally overlap availability schedules due to inadequate validation.

---

3. Dashboard Features

Issues:

1. Data Syncing:
    - Real-time updates for appointment status may lag without proper WebSocket implementation.
2. Navigation Glitches:
    - Sidebar menu may not collapse/expand properly on certain screen sizes.
3. UI Inconsistencies:
    - Buttons, fonts, or spacing may appear inconsistent across pages due to CSS conflicts.

---

4. Payment System

Issues:

1. Payment Gateway Integration:
    - Payments may fail or be delayed due to integration issues with third-party gateways.
2. Receipt Generation:
    - In some cases, receipts may not generate if the payment process is interrupted.
3. Refund Handling:
    - Partial or full refunds might not sync correctly with appointment status changes.

---

5. User Interface

Issues:

1. Responsiveness:
    - Certain pages, like the doctor search or admin dashboard, may break on small screen sizes.
2. Browser Compatibility:
    - Some features (e.g., date pickers) may not work in older browsers like Internet Explorer.
3. Accessibility Issues:
    - Lack of screen reader support or missing ARIA labels may hinder usability for disabled users.

---

6. Performance

Issues:

1. Database Queries:
    - Complex queries (e.g., filtering doctors by specialization and location) may result in high latency.
2. Concurrent Users:
    - The system may slow down when multiple users are booking appointments simultaneously.
3. API Throttling:
    - APIs may not handle rate-limiting properly, allowing abuse of certain endpoints.

---

7. Notifications

Issues:

1. Delayed Notifications:
    - Notifications for appointment updates may not reach the user instantly due to queue failures.
2. Unreadable Notifications:
    - Long notification messages may overflow in the UI, especially on mobile devices.
3. Notification Spam:
    - Users may receive duplicate notifications due to redundant event triggers.

---

8. Security

Issues:

1. SQL Injection:
    - Improper sanitization of inputs in the search or login forms may leave the system vulnerable.
2. Cross-Site Scripting (XSS):
    - Inputs in the comments section or profile update forms may allow malicious scripts.
3. Token Exposure:
    - Tokens stored in localStorage might be exposed to XSS attacks.

---

9. Admin-Specific Issues

Issues:

1. Bulk User Operations:
    - Deleting or updating multiple users at once might fail due to lack of proper batching.
2. Error Handling:
    - Admin actions like "Add Doctor" may not provide meaningful error messages when an operation fails.
3. Data Overload:
    - Loading large reports or logs might cause the admin dashboard to crash.

---

10. Miscellaneous

Issues:

1. Email Notifications:
    - Emails (e.g., booking confirmations) may land in spam due to improper domain configuration (SPF, DKIM).
2. Timezone Handling:
    - Appointment timings may display incorrectly for users in different timezones.
3. Session Expiry:
    - Users might not get a warning before their session expires, causing frustration when filling forms.

---

13. FUTURE ENHANCEMENT:

1. User Experience Enhancements

1. **Mobile App Development**:
    - Create mobile apps for Android and iOS to offer a seamless experience for users on the go.
2. **Voice Search for Doctors**:
    - Integrate voice search functionality to allow users to find doctors by speaking instead of typing.
3. **Multilingual Support**:
    - Add support for multiple languages to cater to a broader audience.
4. **Dark Mode**:

- Implement a dark mode toggle for user comfort, especially during nighttime use.

---

2. Appointment Management

1. **Appointment Reminders**:
   - Send automated SMS or push notifications to remind users of their upcoming appointments.
2. **Waitlist Feature**:
   - Introduce a waitlist system where users can opt to take a canceled slot for popular doctors.
3. **Recurring Appointments**:
   - Allow users to schedule recurring appointments for regular check-ups or treatments.
4. **Time Slot Recommendations**:
   - Provide AI-driven suggestions for the best available time slots based on past booking patterns.

---

3. Teleconsultation

1. **Video Consultation Integration**:
   - Allow patients to book virtual appointments and consult doctors via secure video calls.
2. **File Uploads**:
   - Enable patients to upload medical reports or prescriptions for review during online consultations.
3. **In-App Chat with Doctors**:
   - Introduce a messaging system for follow-ups and clarifications after appointments.

---

4. Payment System Enhancements

1. **Insurance Integration**:
   - Allow users to link their health insurance policies and show covered services during booking.
2. **Subscription Plans**:
   - Offer subscription plans for regular check-ups or discounted consultation fees.

3. **Multiple Payment Gateways**:
    - Integrate additional payment gateways like PayPal, Apple Pay, or Google Pay for flexibility.
4. **Split Payments**:
    - Enable patients to split payments between themselves and their insurance providers.

---

5. Analytics and Insights

1. **Patient Health Dashboard**:
    - Provide patients with a dashboard summarizing their past appointments, prescriptions, and health trends.
2. **Doctor Analytics**:
    - Offer doctors insights into their practice, such as most frequent cases, average consultation time, etc.
3. **Admin Reports**:
    - Enhance admin reports with visual dashboards showing trends in bookings, earnings, and user activity.

---

6. AI and Machine Learning

1. **Doctor Recommendations**:
    - Use AI to suggest the best doctors based on patient reviews, location, and specialization.
2. **Symptom Checker**:
    - Add an AI-powered symptom checker to guide patients in selecting the appropriate specialist.
3. **Predictive Analytics**:
    - Use machine learning to predict user preferences or peak booking times for resource allocation.

---

7. Security and Privacy

1. **Biometric Login**:
    - Allow users to log in using facial recognition or fingerprint authentication for added security.
2. **Data Encryption Enhancements**:

- Upgrade to advanced encryption methods for sensitive patient data.

3. **Audit Logs**:
    - Maintain detailed logs of all user activities for transparency and accountability.

---

8. Scalability

1. **Cloud Migration**:
    - Move to a cloud-based infrastructure (e.g., AWS, Azure) to handle growing user bases efficiently.
2. **Load Balancing**:
    - Implement load balancers to ensure high availability during peak traffic.
3. **Microservices Architecture**:
    - Refactor the application into microservices for better scalability and maintainability.

---

9. Community Engagement

1. **Doctor Reviews and Ratings**:
    - Allow patients to leave reviews and rate doctors after their appointments.
2. **Health Blog or Tips Section**:
    - Provide a blog section with articles, health tips, and news written by certified professionals.
3. **Referral System**:
    - Introduce a referral program where users earn credits for inviting friends to the platform.

---

10. Accessibility

1. **Screen Reader Support**:
    - Make the platform accessible to visually impaired users by adding screen reader compatibility.
2. **Keyboard Navigation**:
    - Ensure all features can be accessed via keyboard for users with mobility challenges.
3. **Low Bandwidth Mode**:

- Introduce a lightweight version of the app for users in regions with limited internet connectivity.