# Doctor Patient Appointment Booking System

## Abstract

A sophisticated web tool called the Doctor Patient Appointment Booking System was created to make scheduling and organizing doctor visits easier. Users can effectively manage bookings, search for doctors by region or specialization, and register as patients or doctors using this system. The solution improves accessibility and convenience for patients and healthcare providers by including email notifications and video calling.

## Introduction

Managing doctor-patient interactions effectively is a difficulty for healthcare systems. By offering a user-friendly platform for connecting patients and doctors, this project seeks to alleviate these problems. The system guarantees a smooth experience with features like video consultations, JWT-based authentication, and real-time booking. The frontend provides an easy-to-use interface, while the backend makes use of Node.js, Express.js, and MongoDB.

## Key Features

1. **User Registration and Login**:
    - Doctors and patients can register with their roles, credentials, and location.
    - Login is secured using **JWT (JSON Web Tokens)**.
2. **Search Functionality**:
    - Patients can search for doctors based on location or specialty.
3. **Appointment Booking**:
    - Patients can book appointments with doctors by selecting a date and time slot.
4. **Video Consultation**:
    - Supports peer-to-peer video calls for remote consultations.
5. **Email Notifications**:
    - Sends confirmation emails upon successful booking.
6. **User Role Management**:
    - Doctors can manage their availability and view bookings.
    - Patients can view or delete their bookings.
7. **Secure Authorization**:
    - Access to sensitive endpoints is restricted based on user roles.

# Code Structure

### Backend:

- **index.js**: Entry point for the Node.js server, managing routes and middleware.
- **config/db.js**: Establishes connection to the MongoDB database.
- **models/**:
  - `userModel.js`: Mongoose schema for users (doctors and patients).
  - `bookingModel.js`: Mongoose schema for appointment bookings.
- **routes/**:
  - `userRoute.js`: Manages user-related endpoints (registration, login, search).
  - `bookingRoute.js`: Manages booking-related endpoints.
- **middlewares/**:
  - `authenticationMiddleware.js`: Verifies JWT tokens.
  - `authorizationMiddleware.js`: Restricts access based on user roles.

### Frontend:

• Offers an appointment scheduling UI that is responsive.
• Integrated location and specialist filter search capabilities.
• A contemporary JavaScript framework or library (React/Angular) was used in its development.

---

# API Endpoints

### User Endpoints

1. **Register User**:
   - `POST /user/register`
   - Registers doctors or patients.
2. **Login User**:
   - `POST /user/login`
   - Authenticates users and issues JWT tokens.
3. **Retrieve Doctors**:
   - `GET /user/doctors`
   - Retrieves all registered doctors.
4. **Retrieve Doctors by Location**:
   - `GET /user/doctors/:location`
   - Filters doctors by location.
5. **Retrieve Doctors by Specialty**:
   - `GET /user/doctors/specialty/:value`
   - Filters doctors by specialty.

### Booking Endpoints

1. **Create Booking**:
   - `POST /booking/create`
   - Books an appointment with a doctor.
2. **Retrieve User Bookings**:
   - `GET /booking/user`
   - Fetches all bookings made by the logged-in patient.
3. **Delete Booking**:
   - `DELETE /booking/:id`
   - Cancels a specific booking.

---

# Implementation Details

### Authentication

- **JWT Tokens**: Ensure secure authentication and session management.
- **Role-Based Access Control**: Restricts actions based on user roles (doctor/patient).

### Database Design

- **MongoDB** is used to store user data, bookings, and session tokens.
- Collections:
  - Users: Stores doctor and patient profiles.
  - Bookings: Tracks appointments.

### Email Notifications

- **Nodemailer** is used to send appointment confirmation emails.
- Environment variables are used to securely manage email credentials.

### Video Consultation

- **PeerJS** is used for establishing peer-to-peer video connections.
- The video calling feature requires the Peer server to be run locally (`peerjs --port 3001`).

---

# Deployment and Usage

## Installation

1. Clone the repository.
2. Navigate to the `backend` directory using `cd backend`.
3. Install dependencies:

   ```
   bash
   Copy code
   npm install
   ```

4. Set up a `.env` file with the following keys:
   - `mongoDbUrl`: MongoDB connection string.
   - `Key`: JWT secret key.
   - `emailPassword`: Password for email notifications.

## Usage

1. Start the server:

   ```
   bash
   Copy code
   npm run server
   ```

2. Start the Peer server for video calls:

   ```
   bash
   Copy code
   peerjs --port 3001
   ```

## Deployment

The application can be deployed using cloud services like **Heroku** or **AWS**. Note that video call functionality requires a local PeerJS server.

---

# Technologies Used

## Backend

- Node.js
- Express.js
- MongoDB
- JWT for authentication
- Nodemailer for email notifications

### Frontend

- React/Angular for UI
- Responsive design

### Other Tools

- PeerJS for video calls
- Bcrypt for password hashing

---

# Challenges Faced

1. **Integrating Video Calls**: Peer-to-peer connections required careful setup of the Peer server.
2. **Role-Based Access Control**: Ensuring security and restricting access to sensitive endpoints.
3. **Email Configuration**: Securing credentials and managing email notifications.

---

# Future Enhancements

1. **Enhanced Video Functionality**: Deploy a cloud-based PeerJS server for video calls in production.
2. **Advanced Filtering**: Add more filters for doctor search, such as ratings and availability.
3. **Payment Gateway**: Integrate online payment options for booking consultations.
4. **Mobile App**: Extend the system to mobile platforms for broader accessibility.

---

# Conclusion

The **Doctor Patient Appointment Booking System** successfully simplifies the process of managing doctor appointments. By integrating essential features like video calls, secure authentication, and email notifications, the system enhances the healthcare experience for patients and doctors alike.

---

# References


1. Node.js Documentation
2. MongoDB Documentation
3. Express.js Official Guide
4. PeerJS Library Documentation