Online Appointment Booking System

This repository contains the backend API for an Online Appointment Booking System for a clinic.

About the Booking System

This system allows users to book appointments with clinics and doctors. It includes the following entities:

- **Clinic:** Represents a medical clinic with a name, address, contact number, and email.
- **Doctor:** Represents a doctor with a name, specialization, contact number, and email.
- **Availability:** Represents the next availability of a doctor for booking appointments.
- **Fees:** Represents the fees charged by a doctor for different modes of appointments.
- **User:** Represents a user of the system with a name, email, and contact number.
- **Slot:** Represents an available time slot for booking appointments.

API Endpoints

Clinic

- **Add Clinic:** `POST /addclinic`
  - Create a new clinic.
  - Request body: { name, address, contactNumber, email }
- **Get Clinic Information:** `POST /getclinic`
  - Get information about a clinic.
  - Request body: { clinicId }

Doctor

- **Add Doctor:** `POST /adddoctor`
  - Create a new doctor.
  - Request body: { name, clinicID, specializations, degrees, contactNumber, email, nextAvailibility }
- **Get Doctor Information:** `POST /getdoctor`
  - Get information about a doctor.
  - Request body: { doctorId }

Availability

- **Set Next Availability:** `POST /setavailibility`
  - Set the next availability for a doctor.
  - Request body: { doctorId, nextAvailability }

Fees

- **Get Doctor Fees:** `POST /getfees`
  - Get fees information for a clinic and mode of consultation.
  - Request body: { clinicId,mode }
- **Set Doctor Fees:** `POST /setfees`
  - Set fees for a clinic.
  - Request body: { clinicId, mode, amount }

User

- **Get User Information:** `POST /getuser`
  - Get information about a user.
  - Request body: { userId }
- **Create User:** `POST /createuser`
  - Create a new user.
  - Request body: { name, email, contactNumber, username, password }
- **Login User:** `POST /login`
  - Authorizing user.
  - Request body: { username, password }

Slot

- **Create Slot:** `POST /createslot`
  - Create a new appointment slot.
  - Request body: { clinicId, time, date, mode }
- **Get Slots:** `POST /getslots`
  - Get information about available slots for a clinic.
  - Request body: { clinicId, date, mode }
- **Book Slot:** `POST /bookslot`
  - Book a slot for a user (Authorized users only). token must be included in request.
  - Request body: { userId, slotId }

Authentication

- **User Login:** `POST /login`
  - Login using email and OTP.
  - Request body: { email, otp }
- **Send OTP:** `POST /sendotp`

  - Send OTP for user verification.
  - Request body: { email }

Authorization

Authorization is implemented using JWT (JSON Web Token). Users need to log in using their email and OTP for authentication. Only authorized users can make bookings.

OTP Verification

Every time a user attempts to book an appointment, the system sends an OTP to the user's email for verification. The booking is only confirmed upon successful OTP verification.

Feel free to explore and use the provided APIs for your appointment booking needs!

Schemas for Entities

Clinic

Fields

- **Name:** (String) The name of the clinic.
- **Address:** (String) The address of the clinic.
- **Contact Number:** (String) The contact number for the clinic.
- **Email:** (String) The email address of the clinic.

Doctor

Fields

- **Name:** (String) The name of the doctor.
- **ClinicID:** (ObjectId) The id of the clinic with whom he is associated.
- **Specialization:** [String] The specialization or medical field of the doctor.
- **Degrees:** [String] The degrees of the doctor.
- **Contact Number:** (String) The contact number for the doctor.
- **Email:** (String) The email address of the doctor.
- **Next Availability:** (Time) The next available time for the doctor's appointments.

Fees

Fields

- **Clinic ID:** (ObjectId) The unique identifier of the clinic.

- **Mode:** (String) The mode of appointment (e.g., “In-Person”, “Telemedicine”).
- **Amount:** (Number) The fees amount for the specified mode.

User

Fields

- **Name:** (String) The name of the user.
- **Email:** (String) The email address of the user.
- **Password:** (String) The password of the user.
- **Token:** (String) The assigned token of the user.
- **OTP:** (String) The otp sent to the user.
- **Contact Number:** (String) The contact number for the user.

Slot

Fields

- **Clinic ID:** (ObjectId) The unique identifier of the clinic.
- **User ID:** (ObjectId) The unique identifier of the user who booked if booked.
- **Time:** (Date) The time of the appointment slot.
- **Date:** (Date) The date of the appointment slot.
- **Mode:** (String) The mode of appointment (e.g., “In-Person”, “Telemedicine”).
- **Is Booked:** (Boolean) Indicates whether the slot is booked or available.

Technologies Used

The backend of this system is built using the following technologies and npm packages:

- **Node.js:** A JavaScript runtime built on Chrome’s V8 JavaScript engine.
- **Express:** A fast, unopinionated, minimalist web framework for Node.js.
- **MongoDB:** A NoSQL database for storing data related to clinics, doctors, appointments, and users.
- **Mongoose:** An Object Data Modeling (ODM) library for MongoDB and Node.js.
- **Bcrypt:** A library for hashing passwords to enhance security.
- **Body-parser:** Middleware to parse incoming request bodies in a middleware before your handlers.
- **Cors:** Middleware for enabling Cross-Origin Resource Sharing in Express applications.
- **Dotenv:** A zero-dependency module that loads environment variables from a .env file.

- **Express-session:** A session middleware for Express.js to manage session data.
- **Jsonwebtoken:** A library to generate and verify JSON Web Tokens (JWTs).
- **Nodemailer:** A module for sending emails using Node.js.
- **Nodemon:** A tool that helps develop Node.js-based applications by automatically restarting the server on file changes.
- **Otp-generator:** A library for generating one-time passwords (OTPs).

How to Run Locally

Follow the steps below to run the project locally on your machine:

1. **Clone the Repository:** "'bash git clone https://github.com/shyam2672/vscodeassignment

   Navigate to the Project Directory.

2. **Install dependencies:**

```
cd your-repo
 npm install
```

3. **Configure .env:** make a .env file in the root directory and add necessary details into it

4. **Run the Development Server:**

```
npm run app.js
```

5. **Access the API Endpoints:** The API server will be running at http://localhost:3000 (or the port specified in your .env file). You can access the defined API endpoints locally.