Doctors Appointment Booking Web App - Project Documentation

---

1. Project Overview

The **Doctors Appointment Booking Web App** is a web-based platform that allows patients to book appointments with doctors, manage their schedules, and receive real-time updates. Doctors can manage their availability, view appointments, and update appointment statuses. This project is designed to teach integrated learning concepts such as user authentication, CRUD operations, real-time updates, and scheduling.

---

2. User Requirements

Patients:

- Register and log in to their accounts.
- View a list of available doctors.
- Book, reschedule, or cancel appointments.
- Receive real-time notifications about appointment status (confirmed, canceled, etc.).
- View their appointment history.

Doctors:

- Register and log in to their accounts.
- Set and update their availability (working hours, days off, etc.).
- View and manage their appointments (confirm, cancel, etc.).
- Receive notifications about new appointments or changes.

Admin (Optional):

- Manage doctors and patients (add, remove, or update profiles).
- Monitor system activity (appointments, user activity, etc.).

---

3. Tech Stack

Frontend:

- **React**: For building the user interface.
- **Bootstrap**: For styling and responsive design.
- **Axios**: For making API requests to the backend.
- **Socket.io Client**: For real-time updates.

Backend:

- **Node.js**: Runtime environment.
- **Express**: Framework for building the REST API.
- **Socket.io**: For real-time communication.

- **Firebase Auth** or **Passport.js**: For user authentication.
- **MongoDB**: Database for storing user data, appointments, and doctor profiles.
- **Mongoose**: For MongoDB object modeling.

Real-Time Updates:

- **Socket.io**: Enables real-time communication between the frontend and backend.

Hosting:

- **Vercel**: For hosting the frontend.
- **Render**: For hosting the backend.

---

4. Project File Structure

Copy

```
doctor-appointment-booking-app/
├── client/                # Frontend (React)
│   ├── public/
│   ├── src/
│   │   ├── components/    # Reusable components (Navbar, Footer, etc.)
│   │   ├── pages/         # Pages (Home, Login, Register, etc.)
│   │   ├── services/      # API service calls (Axios)
│   │   ├── App.js         # Main application component
│   │   ├── index.js       # Entry point
│   │   └── styles/        # CSS or SCSS files
│   └── package.json
│
├── server/                # Backend (Node.js + Express)
│   ├── controllers/       # Logic for handling routes
│   ├── models/            # Database models (User, Appointment, Doctor)
│   ├── routes/            # API routes
│   ├── utils/             # Utility functions (e.g., authentication)
│   ├── app.js             # Main application file
│   ├── server.js          # Server entry point
│   └── package.json
│
├── README.md              # Project documentation
└── .gitignore             # Files to ignore in Git
```

---

5. Step-by-Step Guide

Step 1: Set Up the Backend

1. **Initialize the Node.js project:**

bash

Copy

```
mkdir server
cd server
npm init -y
```

2. **Install dependencies:**

bash

Copy

```
npm install express mongoose socket.io firebase-admin passport passport-local cors dotenv
```

3. **Set up the Express server:**
    - Create server.js and app.js files.
    - Configure middleware (e.g., cors, express.json).
    - Set up routes for authentication, appointments, and doctors.
4. **Set up MongoDB:**
    - Create a MongoDB database (e.g., using MongoDB Atlas).
    - Define Mongoose models for User, Doctor, and Appointment.
5. **Implement authentication:**
    - Use Firebase Auth or Passport.js for user authentication.
    - Create endpoints for user registration, login, and profile management.
6. **Set up Socket.io:**
    - Integrate Socket.io for real-time updates (e.g., appointment status changes).

Step 2: Set Up the Frontend

1. **Initialize the React project:**

bash

Copy

```
npx create-react-app client
cd client
```

2. **Install dependencies:**

bash

Copy

npm install axios bootstrap react-router-dom socket.io-client

3. **Create components and pages:**
    - Create reusable components (e.g., Navbar, Footer).
    - Create pages for Home, Login, Register, DoctorList, AppointmentBooking, etc.
4. **Set up routing:**
    - Use react-router-dom to handle navigation between pages.
5. **Connect to the backend:**
    - Use Axios to make API calls to the backend.
    - Use Socket.io client to listen for real-time updates.

Step 3: Integrate Real-Time Updates

1. **Backend:**
    - Emit Socket.io events when appointment status changes (e.g., appointmentConfirmed, appointmentCanceled).
2. **Frontend:**
    - Listen for Socket.io events and update the UI in real-time.

Step 4: Hosting

1. **Frontend:**
    - Deploy the React app on **Vercel**:

bash

Copy

npm install -g vercel

vercel

2. **Backend:**
    - Deploy the Node.js app on **Render**:
        - Push your code to GitHub.
        - Connect your GitHub repository to Render and deploy.

---

6. Key Features to Implement

- **User Authentication:** Patients and doctors can register, log in, and manage their profiles.
- **CRUD Operations:**
    - Patients can create, read, update, and delete appointments.

    - Doctors can manage their availability and appointments.
- **Real-Time Updates:** Patients and doctors receive live updates about appointment statuses.
- **Scheduling:** Patients can view doctor availability and book appointments accordingly.

---

7. Additional Resources

- **React Documentation:** https://reactjs.org/docs/getting-started.html
- **Express Documentation:** https://expressjs.com/
- **MongoDB Documentation:** https://docs.mongodb.com/
- **Socket.io Documentation:** https://socket.io/docs/

---

8. Starter Code

Backend (Express + MongoDB):

javascript

Copy

```
// server.js
const express = require('express');
const mongoose = require('mongoose');
const app = require('./app');
const http = require('http');
const socketIo = require('socket.io');

const server = http.createServer(app);
const io = socketIo(server);

// Connect to MongoDB
mongoose.connect('mongodb://localhost:27017/doctor-appointment', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});

// Socket.io connection
io.on('connection', (socket) => {
  console.log('New client connected');
  socket.on('disconnect', () => {
```

```
    console.log('Client disconnected');
  });
});

const PORT = process.env.PORT || 5000;
server.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

Frontend (React):

javascript

Copy

```
// App.js
import React, { useEffect, useState } from 'react';
import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';
import Home from './pages/Home';
import Login from './pages/Login';
import Register from './pages/Register';
import DoctorList from './pages/DoctorList';
import AppointmentBooking from './pages/AppointmentBooking';
import Navbar from './components/Navbar';
import io from 'socket.io-client';

const socket = io('http://localhost:5000');

function App() {
  const [appointmentStatus, setAppointmentStatus] = useState('');

  useEffect(() => {
    socket.on('appointmentConfirmed', (data) => {
      setAppointmentStatus(data.message);
    });
  }, []);

  return (
```

```
    <Router>
      <Navbar />
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/login" element={<Login />} />
        <Route path="/register" element={<Register />} />
        <Route path="/doctors" element={<DoctorList />} />
        <Route path="/book-appointment" element={<AppointmentBooking />} />
      </Routes>
    </Router>
  );
}

export default App;
```

---

9. Conclusion

This project provides a comprehensive learning experience for students, covering essential web development concepts. By following this guide, students will build a fully functional Doctors Appointment Booking Web App and gain hands-on experience with modern web technologies.

Let me know if you need further assistance! 🚀