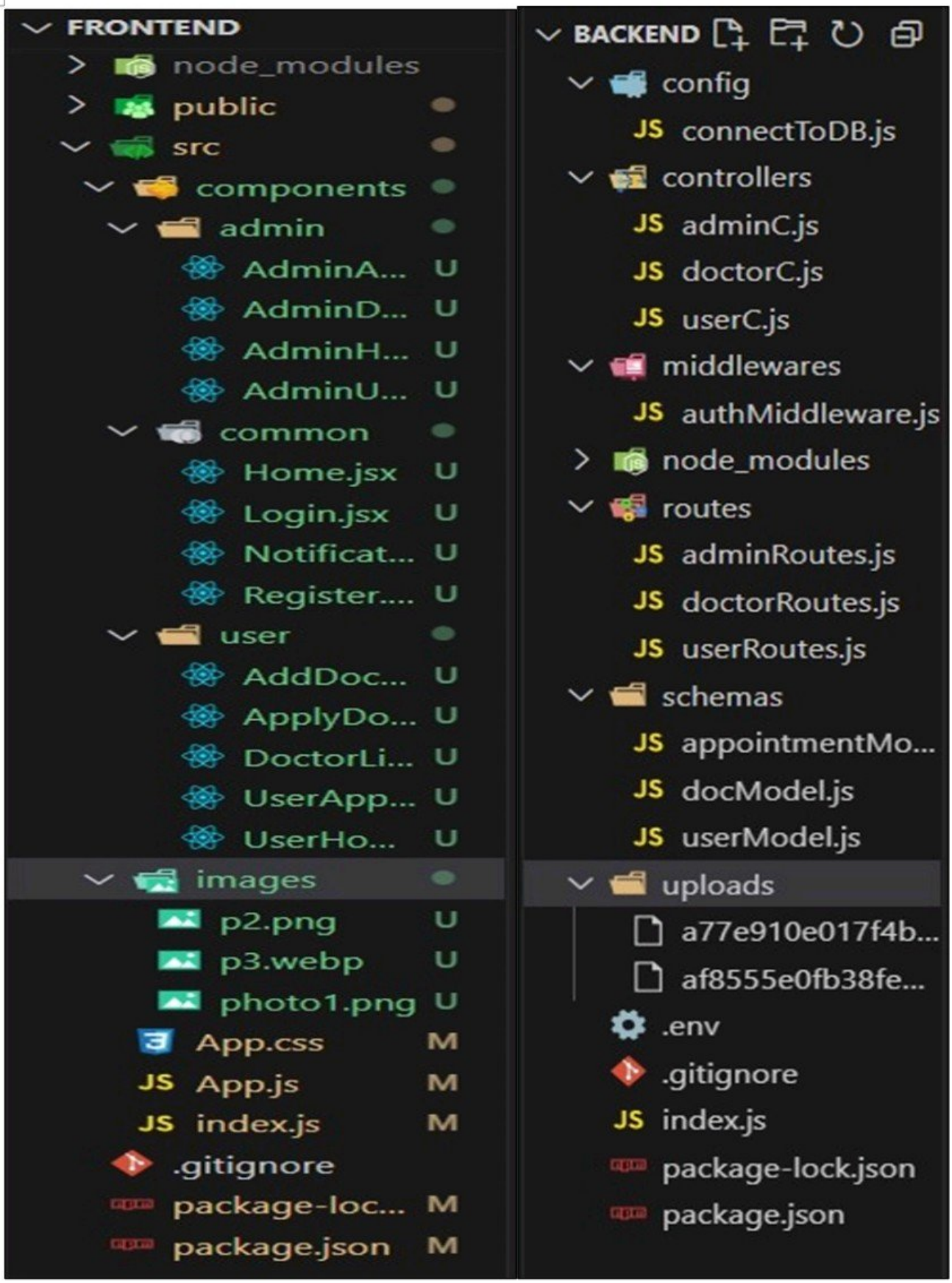
FRONTEND
node_modules
public
src
components
admin
AdminA... U
AdminD... U
AdminH... U
AdminU... U
common
Home.jsx U
Login.jsx U
Notificat... U
Register.... U
user
AddDoc... U
ApplyDo... U
DoctorLi... U
UserApp... U
UserHo... U
images
p2.png U
p3.webp U
photo1.png U
App.css M
App.js M
index.js M
.gitignore
package-loc... M
package.json M
BACKEND
config
connectToDB.js
controllers
adminC.js
doctorC.js
userC.js
middlewares
authMiddleware.js
node_modules
routes
adminRoutes.js
doctorRoutes.js
userRoutes.js
schemas
appointmentMo...
docModel.js
userModel.js
uploads
a77e910e017f4b...
af8555e0fb38fe...
.env
.gitignore
index.js
package-lock.json
package.json

FRONTEND PART :

- The frontend is responsible for creating and rendering the user interface that customers, doctors, and admins interact with. It consists of the following files and folders:

- REACT COMPONENTS – Each component is designed for user interactions such as displaying doctors, booking appointments, and viewing notifications.
- ROUTING – Handles navigation between pages like customer dashboard, booking form, history, etc.
- STATE MANAGEMENT – Keeps track of the logged-in user's session, doctor details, and appointment statuses.
- STYLING – Uses CSS and UI libraries (e.g., Ant Design) to style the components.

BACKEND PART :

The backend handles the server-side operations, including user authentication, data handling, and API responses. It contains the following files and folders:

- API ENDPOINTS – Defines the routes for handling customer, doctor, and admin functionalities, such as booking appointments, updating statuses, etc.
- DATABASE MODELS – Defines schemas for Users, Doctors, and Appointments using MongoDB and Mongoose.
- AUTHENTICATION & AUTHORIZATION – Manages login, registration, and access control for different user roles.
- NOTIFICATION SYSTEM – Sends notifications to users about appointment status updates.
- ADMIN FUNCTIONS – Admin-related routes for managing users, doctors, and appointment approvals.

APPLICATION FLOW:

This project has three main types of users: Customer, Doctor, and Admin. Each has specific roles and responsibilities that are defined by the API endpoints.

CUSTOMER/ORDINARY USER:

- CREATE AN ACCOUNT & LOGIN – Customers can register and log in using their email and password.
- VIEW DOCTORS – After logging in, customers will see a list of available doctors in their dashboard.
- BOOK APPOINTMENT – Customers can select a doctor and book an appointment by filling out a form with the appointment date and required documents.
- VIEW APPOINTMENT STATUS – Customers can track the status of their appointments (approved, pending, cancelled) and receive notifications when the appointment is scheduled.
- CANCEL BOOKING – Customers can cancel their appointments from their booking history page and change the status of the booking if needed.

ADMIN:

- MONITOR ALL OPERATIONS – The admin oversees the platform, including the management of users, doctors, and appointments.
- APPROVE DOCTOR APPLICATIONS – Admins can review and approve doctor applications, making them available in the app.
- MANAGE POLICIES – Admin enforces platform policies, terms of service, and privacy regulations.
- USER MANAGEMENT – Admins can manage the profiles of customers and doctors, monitor their actions, and maintain a secure environment.

DOCTOR:

- ACCOUNT APPROVAL – Doctors must receive approval from the admin before they can use the platform.
- MANAGE APPOINTMENTS – Once registered, doctors can manage the appointments they receive from customers, including confirming, rescheduling, or rejecting them.
- APPOINTMENT NOTIFICATIONS – Doctors are notified when new appointments are booked and when customers update their appointment details.

SETUP & CONFIGURATION :

Setting up the Doctor Appointment Webpage involves configuring both the Frontend (React.js) and Backend (Node.js, Express.js, MongoDB) to ensure the application runs smoothly. Below are the steps to set up and configure the environment for your project.

FRONTEND CONFIGURATION :

INSTALLATION :

Clone the Repository:

- Clone the project from GitHub to your local machine:
- bash
- Copy code
- git clone <your-repository-url>
- Replace <your-repository-url> with the URL of your project repository.

Navigate to Frontend Directory:

- After cloning, navigate to the frontend folder where the React.js app is located:
- bash
- Copy code
- cd book-a-doctor/frontend

Install Dependencies:

- Use npm (Node Package Manager) to install the necessary dependencies:
- bash
- Copy code
- npm install
- This will install all the required libraries, such as React, React Router, Ant Design, and others.

Run the React Development Server:

- To start the frontend server and run the React application:
- bash
- Copy code
- npm start
- The application will be available at http://localhost:3000 in your browser.

BACKEND CONFIGURATION :

INSTALLATION :
Navigate to Backend Directory:

- Move to the backend folder of your project:
- bash
- Copy code
- cd book-a-doctor/backend
- Install Dependencies:
- Install the necessary backend dependencies using npm:
- bash
- Copy code
- npm install
- This will install all required libraries for Node.js and Express.js, such as express, mongoose, bcryptjs, jsonwebtoken, etc.

Configure MongoDB :

- Ensure you have MongoDB installed on your local machine or use a cloud-based MongoDB service like MongoDB Atlas.
- In the backend, open the configuration file (e.g., config/db.js or .env) and configure the connection URL for MongoDB:
- bash
- Copy code
- MONGO_URI=mongodb://localhost:27017/doctor_appointment
- If you are using MongoDB Atlas, replace the localhost part with your MongoDB Atlas connection string.

Set Up Environment Variables:

- Create a .env file in the backend directory to store environment-specific variables such as:
- bash
- Copy code
- PORT=5000
- JWT_SECRET=your_jwt_secret
- Make sure to replace your_jwt_secret with a strong secret key for JWT authentication.

Run the Backend Server:

- Start the backend server by running:
- bash
- Copy code
- npm start
- The backend server will be running at http://localhost:5000.

- Start the backend server by running:
- bash
- Copy code
- npm start
- The backend server will be running at http://localhost:5000.

DATABASE CONFIGURATION (MONGODB) :

Install MongoDB (Local Installation):

- If you are using a local MongoDB instance, download and install it from the official MongoDB website: Download MongoDB

Set Up MongoDB Database:

- After installation, start the MongoDB service:
- bash
- Copy code
- mongod
- This will run MongoDB on the default port 27017.

MongoDB Atlas (Cloud-based MongoDB):

- If you prefer to use MongoDB Atlas, create an account on MongoDB Atlas and create a cluster.
- Once the cluster is set up, get the connection string and replace it in the backend's .env file:
- bash
- Copy code
- MONGO_URI=<your-mongodb-atlas-connection-string>

FINAL CONFIGURATION & RUNNING THE APP

Run Both Servers:

- The React frontend and Node.js backend servers should run simultaneously for the app to function correctly.
- You can open two terminal windows or use tools concurrently to run both servers together.
- To install concurrently, run:
- bash
- Copy code
- npm install concurrently --save-dev
-

In your package.json file, add a script to run both servers:

json
Copy code

```
"scripts": {
  "start": "concurrently \"npm run server\" \"npm run client\"",
  "server": "node backend/server.js",
  "client": "npm start --prefix frontend"
}
```

Start Both Servers:

- Now you can run the following command in the root directory to start both the backend and frontend:
- bash
- Copy code
- npm start
- The backend will be available at http://localhost:5000, and the frontend at http://localhost:3000.

VERIFYING THE APP :

Check Frontend:

- Open your browser and go to http://localhost:3000. The React.js application should load with the list of doctors, booking forms, and status updates.

Check Backend:

- Open Postman or any API testing tool to verify if the backend endpoints are working correctly, such as user login, doctor registration, and appointment creation.

ADDITIONAL SETUP :

- Version Control:
- If you haven't already done so, initialise a Git repository in the root of your project:
- bash
- Copy code
- git init

Add your project files and commit them:

- bash

DATABASE CONFIGURATION (MONGODB) :

Install MongoDB (Local Installation):

- If you are using a local MongoDB instance, download and install it from the official MongoDB website: Download MongoDB

Set Up MongoDB Database:

- After installation, start the MongoDB service:
- bash
- Copy code
- mongod
- This will run MongoDB on the default port 27017.

MongoDB Atlas (Cloud-based MongoDB):

- If you prefer to use MongoDB Atlas, create an account on MongoDB Atlas and create a cluster.
- Once the cluster is set up, get the connection string and replace it in the backend's .env file:
- bash
- Copy code
- MONGO_URI=<your-mongodb-atlas-connection-string>

FINAL CONFIGURATION & RUNNING THE APP

Run Both Servers:

- The React frontend and Node.js backend servers should run simultaneously for the app to function correctly.
- You can open two terminal windows or use tools concurrently to run both servers together.
- To install concurrently, run:
- bash
- Copy code
- npm install concurrently --save-dev
-

In your package.json file, add a script to run both servers:

json
Copy code

```
"scripts": {
  "start": "concurrently \"npm run server\" \"npm run client\"",
  "server": "node backend/server.js",
  "client": "npm start --prefix frontend"
}
```

Start Both Servers:

- Now you can run the following command in the root directory to start both the backend and frontend:
- bash
- Copy code
- npm start
- The backend will be available at http://localhost:5000, and the frontend at http://localhost:3000.

```
  "client": "npm start --prefix frontend"
}
```

Start Both Servers:

- Now you can run the following command in the root directory to start both the backend and frontend:
- bash
- Copy code
- npm start
- The backend will be available at http://localhost:5000, and the frontend at http://localhost:3000.

VERIFYING THE APP :

Check Frontend:

- Open your browser and go to http://localhost:3000. The React.js application should load with the list of doctors, booking forms, and status updates.

Check Backend:

- Open Postman or any API testing tool to verify if the backend endpoints are working correctly, such as user login, doctor registration, and appointment creation.

ADDITIONAL SETUP :

- Version Control:
- If you haven't already done so, initialise a Git repository in the root of your project:
- bash
- Copy code
- git init

Add your project files and commit them:

- bash
- Copy code
- git add .
- git commit -m "Initial commit"
- Deployment (Optional):
- If you want to deploy your app, consider using cloud platforms like Heroku, AWS, or Vercel for frontend hosting and backend API deployment.

FOLDER SETUP :

The folder structure for your Doctor Appointment Webpage project will include separate folders for the frontend and backend components to keep the code organised and modular. Here's how to set it up:

PROJECT ROOT STRUCTURE :

Create the Main Folders:

- In your project's root directory, create two main folders: frontend and backend.
- plaintext
- Copy code
- project-root/
  ├── frontend/
  └── backend/

BACKEND SETUP :

- Install Necessary Packages in the Backend Folder:
- Navigate to the backend folder and install the following essential packages:
- plaintext
- Copy code
- backend/

```
├── config/
├── controllers/
├── models/
├── routes/
├── middleware/
├── uploads/
├── server.js
└── .env
```

Packages to Install :

- cors: To enable cross-origin requests.
- bcryptjs: For securely hashing user passwords.
- express: A lightweight framework to handle server-side routing and API management.
- dotenv: For loading environment variables.
- mongoose: To connect and interact with MongoDB