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.

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
- 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.
- multer: To handle file uploads.
- nodemon: A utility to auto-restart the server upon code changes (for development).
- jsonwebtoken: To manage secure, stateless user authentication.
- Installation Commands

Run these commands in the backend folder:

- bash
- Copy code
- npm init -y
- npm install cors bcryptjs express dotenv mongoose multer jsonwebtoken
- npm install --save-dev nodemon

FRONTEND SETUP :

- React Project Initialization:
-
- Navigate to the frontend folder and initialise a new React application:
- plaintext
- Copy code
- frontend/
  ├── public/
  ├── src/
  │ ├── components/
  │ ├── pages/
  │ ├── services/
  │ └── App.js
  ├── .env
  └── package.json
- Setting Up the Frontend Project
- In the frontend folder, run the following commands to set up and install any initial dependencies:
- bash

PROJECT FLOW :

Project Demo :

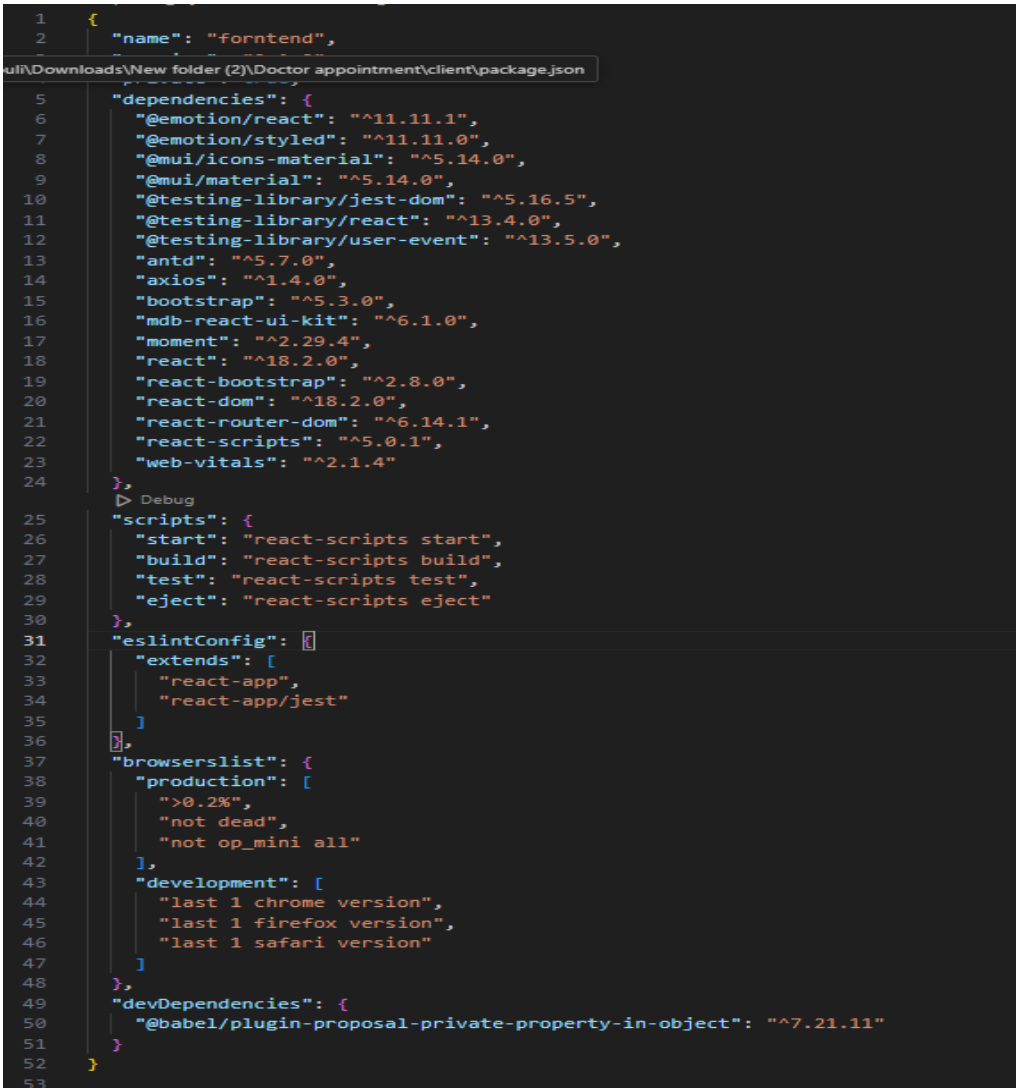
- Before diving into development, you can view a demo of the project to understand its functionality and user interactions.
- Project FlowDemo Video : https://drive.google.com/drive/folders/1pteT8STdObONWwELNDHRK9biItLuiJ-1?usp=sharing
- Project Code Repository : Doctor Appointment Booking Using MERN Source Code
- The source code for this project can be accessed and cloned from GitHub, providing a base structure and example code for further customization and understanding.
- GitHub Repository: Source Code

Video Tutorials :

- For a step-by-step guide on setting up and working with this project, follow the video tutorials below:
- Video Guide 1: Setting Up the Backend
- Video Guide 2: Configuring Frontend and API Endpoints
- Video Guide 3: Testing and Deployment
- These resources should give you a solid foundation and clear understanding of the project structure and workflow before development begins.

MILESTONE 1: PROJECT SETUP AND CONFIGURATION :

- Setting up a structured environment is crucial for any application. By organising the project into separate folders for the frontend and backend, we ensure that the codebase is manageable and scalable.




```
{
  "name": "forntend",
  "dependencies": {
    "@emotion/react": "^11.11.1",
    "@emotion/styled": "^11.11.0",
    "@mui/icons-material": "^5.14.0",
    "@mui/material": "^5.14.0",
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "antd": "^5.7.0",
    "axios": "^1.4.0",
    "bootstrap": "^5.3.0",
    "mdb-react-ui-kit": "^6.1.0",
    "moment": "^2.29.4",
    "react": "^18.2.0",
    "react-bootstrap": "^2.8.0",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.14.1",
    "react-scripts": "^5.0.1",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  },
  "devDependencies": {
    "@babel/plugin-proposal-private-property-in-object": "^7.21.11"
  }
}
```

```
{
  "name": "backend",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  ▷ Debug
  "scripts": {
    "start": "node index.js",
    "dev": "nodemon index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "cors": "^2.8.5",
    "dotenv": "^16.3.1",
    "express": "^4.18.2",
    "jsonwebtoken": "^9.0.1",
    "mongoose": "^7.3.2",
    "multer": "^1.4.5-lts.1",
    "nodemon": "^3.0.1"
  }
}
```

PROJECT FOLDERS:

- Frontend Folder: Contains all code related to the user interface, written in JavaScript using frameworks and libraries like React, Material UI, and Bootstrap. This setup helps maintain a clear boundary between UI logic and server logic.
- Backend Folder: Manages the server, API routes, and database interactions, typically handled through Node.js and Express.js. Using separate folders enables a modular structure, allowing changes in one area without affecting the other.

LIBRARY AND TOOL INSTALLATION:

Backend Libraries:

- Node.js: Provides a runtime environment to run JavaScript code on the server side.
- MongoDB: A NoSQL database, perfect for flexible and schema-less data storage, ideal for applications needing frequent updates and various data types.
- Bcrypt: Encrypts passwords for secure authentication, helping protect user data from potential breaches.
- Body-parser: Parses incoming request bodies, making it easy to access data in various formats like JSON.
- Frontend Libraries:
- React.js: Manages component-based UI creation, providing the flexibility to build reusable UI components.
- Material UI & Bootstrap: Provides styling frameworks, ensuring a consistent, responsive, and visually appealing design.
- Axios: Facilitates easy HTTP requests, allowing the frontend to communicate with the backend effectively.

MILESTONE 2: BACKEND DEVELOPMENT :

The backend forms the core logic and data management for the project. A well-structured backend ensures efficient data handling, security, and scalability.

EXPRESS.JS SERVER SETUP:

- Express Server: Acts as a hub for all requests and responses, routing them to appropriate endpoints. It's Essential for managing incoming requests from the frontend, processing them, and sending responses back.
- Middleware Configuration: Middleware like body-parser parses JSON data in requests, while cors enables cross-origin communication between the frontend and backend. Middleware makes it easy to add additional functionality, like error handling or data validation, without interfering with core application logic.

API ROUTE DEFINITION:

- Route Organization: Organizing routes by functionality (e.g., authentication, appointments, complaints) keeps the codebase readable and easy to maintain. For instance, all authentication-related routes can reside in an auth.js file, ensuring each file has a single purpose.
- Express Route Handlers: Route handlers manage the flow of data between client and server, such as fetching, creating, updating, and deleting records.

DATA MODELS (SCHEMAS) WITH MONGOOSE:

- User Schema: Defines structure for user data and includes fields for personal information and role-based access (e.g., doctor, customer, admin). Using a schema ensures consistent data storage.
- Appointment and Complaint Models: Models like Appointment and Complaint manage complex data interactions, including relationships between users and appointments, allowing efficient querying.
- CRUD Operations: CRUD functionalities (Create, Read, Update, Delete) provide a standard interface for handling data operations. They simplify data manipulation in a structured, predictable way.

USER AUTHENTICATION:

- JWT Authentication: JSON Web Tokens securely handle session management, ensuring that only verified users access protected routes. JWT tokens are embedded in request headers, verifying each request without storing session data on the server.

ADMIN AND TRANSACTION HANDLING:

- Admin Privileges: Administrators oversee user registrations, approve appointments, and manage doctor applications. Admin-specific routes ensure that these actions are isolated and secure.
- Transaction Management: This functionality allows customers to interact with appointments, booking history, and cancellations in real-time.

ERROR HANDLING:

- Middleware for Error Handling: Error-handling middleware catches issues and sends meaningful error responses with HTTP status codes (like 404 for Not Found or 500 for Server Error), enabling better debugging and user feedback.
-

MILESTONE 3: DATABASE DEVELOPMENT

- Using MongoDB as the database provides a flexible, schema-less structure, perfect for handling different types of user and appointment data.

SCHEMAS FOR DATABASE COLLECTIONS:

- User Schema: Defines fields for user information like name, email, password, and userType. This schema allows fine-grained control over user data and easy retrieval of information.
- Complaint and Assigned Complaint Schemas: These schemas manage complaint data, with fields linking complaints to users and statuses. They allow efficient tracking of complaints and status updates by linking agents to users.
- Chat Window Schema: This schema organises messages between users and agents, storing them by complaint ID for a streamlined user-agent communication flow.

DATABASE COLLECTIONS IN MONGODB:

- MongoDB collections, such as users, complaints, and messages, provide a structured, NoSQL approach to data management, making it easy to scale as data grows.
- MILESTONE 4: FRONTEND DEVELOPMENT
- Frontend development focuses on creating an interactive, intuitive user experience through a React-based user interface.

REACT APPLICATION SETUP:

- Folder Structure and Libraries: Setting up the initial React app structure and libraries ensures a smooth development workflow. By organising files into components, services, and pages, the project becomes easy to navigate and maintain.
- UI Component Libraries: Material UI and Bootstrap offer pre-built components, enabling rapid UI development and consistent design across all screens.

UI COMPONENTS FOR REUSABILITY:

- Reusable Components: Each UI element, like forms, dashboards, and buttons, is designed as a reusable component. This modularity allows efficient reuse across the app, reducing development time and ensuring consistency.
- Styling and Layout: Styling and layout components maintain a cohesive look and feel, contributing to the user experience with clean, intuitive visuals.

FRONTEND LOGIC IMPLEMENTATION:

- API Integration: Axios is used to make API calls to the backend, connecting UI components with data from the server.
- Data Binding and State Management: React's state management binds data to the UI, automatically updating it as the user interacts with the app.

MILESTONE 5: PROJECT IMPLEMENTATION AND TESTING:

- After completing development, running a final set of tests is crucial for identifying any bugs or issues