Documentation for Full-Stack Application(E-Commerce Website)

Overview

This full-stack application consists of a **frontend** (client) developed using React and a **backend** (server) built with Node.js and connected to a MongoDB Atlas database. The application supports various features, including category and product management, user authentication (register, login, logout), and token-based authorization.

Frontend (Client)

The frontend is a React application created with create-react-app. It provides the user interface for interacting with the backend services.

Key Files and Structure:

1. **src/index.js**:
   - Entry point for the React application.
   - Renders the App component into the root HTML element.
2. **src/App.js**:
   - Main application component.
   - Displays a simple layout with a header, logo, and links.
3. **src/App.css**:
   - Contains styles for the App component.
4. **public/index.html**:
   - HTML template for the React application.
   - Contains the root div element where the React components are mounted.

Running the Frontend:

1. Install dependencies: npm install
2. Start the development server: npm start

Backend (Server)

The backend is a Node.js application using Express.js as the web framework. It connects to a MongoDB Atlas database and provides RESTful APIs for the frontend.

Folder Structure:

1. **controllers/**:
   - Contains logic for handling requests and responses.
   - Example: categoryController.js, productController.js, authController.js.
2. **middleware/**:
   - Includes middleware functions for authentication and authorization.
   - Example: authMiddleware.js, tokenMiddleware.js.
3. **models/**:
   - Defines the data schema for MongoDB collections using Mongoose.
   - Example: CategoryModel.js, ProductModel.js, UserModel.js.
4. **routes/**:
   - Defines API routes for different functionalities.
   - Example: categoryRoutes.js, productRoutes.js, authRoutes.js.
5. **server.js**:
   - Main entry point of the server.
   - Sets up the Express application, connects to MongoDB, and configures routes and middleware.

Database Models:

1. **CategoryModel**:
   - Fields: name, description.
2. **ProductModel**:
   - Fields: name, price, description, categoryId.
3. **UserModel**:
   - Fields: username, email, password, roles.

API Endpoints:

1. **Authentication**:
   - POST /auth/register: Register a new user.
   - POST /auth/login: Log in a user.
   - POST /auth/logout: Log out a user.
   - POST /auth/refresh: Refresh authentication tokens.

2. **Category Management**:
    - GET /categories: Retrieve all categories.
    - POST /categories: Create a new category.
    - PUT /categories/:id: Update a category.
    - DELETE /categories/:id: Delete a category.
3. **Product Management**:
    - GET /products: Retrieve all products.
    - POST /products: Create a new product.
    - PUT /products/:id: Update a product.
    - DELETE /products/:id: Delete a product.

Key Middleware:

1. **authMiddleware.js**:
    - Verifies authentication tokens.
    - Ensures that only authorized users access protected routes.
2. **tokenMiddleware.js**:
    - Manages token generation and validation.

Running the Backend:

1. Install dependencies: npm install
2. Set up environment variables:
    - MONGO_URI: MongoDB Atlas connection string.
    - JWT_SECRET: Secret key for token generation.
3. Start the server: npm start

Integration

The frontend communicates with the backend via RESTful API calls. Authentication is managed using JSON Web Tokens (JWT).

Key Features:

1. **User Authentication**:
    - Secure registration, login, and logout processes.
    - Token-based authentication for protecting routes.
2. **Category and Product Management**:

    - Create, update, delete, and retrieve categories and products.

Deployment:

1. Deploy the frontend using a service like Netlify or Vercel.
2. Deploy the backend using services like Heroku or AWS, ensuring the MongoDB Atlas database is properly configured.

Conclusion

This full-stack application provides a robust architecture for managing categories, products, and user authentication. It is modular, scalable, and easy to extend with additional features.