Full Stack Development with MERN Project Documentation format

# Introduction

* + **Project Title:** E-Com
  + **Team Members:** 1] Chahak Jain -Back End

2] Om Deepak Patil – Front End

3] Mokshita Jain – Database

4] Sancharini Chakraborty – Management feature

# Project Overview

* + **Purpose:**  The purpose of the E-Com project is to develop a fully functional, scalable, and user-friendly e-commerce web application that enables users to browse products, search and filter items, manage a shopping cart, leave reviews, and complete secure purchases using Razor pay integration. The platform also provides a powerful admin dashboard to manage products, users, orders, and admin accounts effectively. Built using the MERN stack (MongoDB, Express.js, React.js, and Node.js), the goal of this project is to simulate a real-world online shopping experience while applying best practices in full-stack development, modern UI/UX design, and web security. The project not only serves as a technical milestone for the developers but also as a practical solution for real-world e-commerce needs.
  + **Features:** The E-Com project offers a comprehensive set of features that reflect the essential components of a modern e-commerce platform. Users can browse through a wide range of products, perform keyword-based searches, view product details, and manage items in a fully functional shopping cart. The platform supports user authentication, allowing users to register, log in, view their profiles, and track order history. It includes a review and rating system for products and showcases a top products carousel to highlight popular items. The checkout process is smooth and secure, with Razor pay integration for payment. For administrators, the system provides a robust admin dashboard to manage products, users, orders, and other admins. Features like product pagination, order status updates, and a database seeder for easy setup further enhance the platform's usability and maintainability.

# Architecture

* + **Frontend:** The frontend of the E-Com project is built using React.js, offering a dynamic and responsive user interface with efficient component-based architecture. Reacts virtual DOM enhances performance by updating only the components that need re-rendering. The application uses React Router for seamless navigation between pages such as Home, Product Listings, Product Details, Cart, Checkout, Login/Register, Profile, and Admin Dashboard. State management is handled using both Reacts Context API and local state, ensuring smooth interaction and data flow between components. Axios or Fetch API is used for communication with the backend RESTful APIs. The interface is designed with modern UI/UX principles, ensuring responsiveness across devices. Components are modular and reusable, making the codebase clean, maintainable, and scalable.
  + **Backend:** The backend of the E-Com project is developed using Node.js and Express.js, providing a scalable and efficient server-side architecture. Express.js is used to create RESTful APIs that handle all core functionalities such as user authentication, product management, order processing, and review submissions. The server uses JWT (JSON Web Tokens) for secure user authentication and authorization, ensuring protected access to sensitive routes. Data is stored and managed in MongoDB, with Mongoose as the ODM (Object Data Modeling) library for schema definition and database operations. Middleware functions are implemented for error handling, route protection, and request validation. The backend is modular, with separate route handlers and controllers for users, products, orders, and admins, making the codebase organized and easy to maintain. This architecture ensures smooth communication between the frontend and database, enabling dynamic and real-time data interaction.
  + **Database:** The E-Com project utilizes MongoDB as its primary database due to its flexibility, scalability, and JSON-like document structure, which aligns well with the MERN stack. The database is structured into multiple collections, primarily including Users, Products, Orders, and Reviews. Each collection follows a well-defined schema using Mongoose, allowing for structured data modeling with validation rules. The User schema stores details such as name, email, password (encrypted), role (admin/user), and order history. The Product schema includes product name, image, description, brand, category, price, stock count, ratings, and associated reviews. The Order schema captures buyer details, ordered items, shipping address, payment method, total price, and order status (paid/delivered). The Review schema is embedded within the Product model, storing user feedback and ratings. MongoDB interactions such as data creation, retrieval, update, and deletion are handled via Mongoose methods, with proper error handling and async operations ensuring efficient communication between the server and the database.

# Setup Instructions

* + **Prerequisites:** To successfully run the E-Com project, several software prerequisites and dependencies must be installed and configured. The core technologies include Node.js (v14 or above) and npm (Node Package Manager) for managing backend and frontend packages, along with MongoDB for the database. On the frontend, React.js is used, requiring tools like Create React App, React Router DOM, and Axios for API communication. The backend depends on Express.js, Mongoose for MongoDB interaction, dotenv for managing environment variables, and jsonwebtoken (JWT) for secure authentication. Additional tools like Razor pay Node SDK are required for payment integration. Development also requires Visual Studio Code or any suitable code editor, Postman for API testing, and optionally MongoDB Compass for database visualization. Version control is managed via Git and the application can be deployed using services like Render, Netlify, or Vercel.
  + **Installation:**

1. **Fork the repository:** Start by forking the repository to your GitHub account.
2. **Clone the repository:**

* git clone https://github.com/your-username/MERN-eCommerce.git
* cd MERN-eCommerce

1. **Create MongoDB Database:** Go to [MongoDB Atlas](https://www.mongodb.com/cloud/atlas) and create a new database, then get the MongoDB URI.
2. **Create Razorpay Account:** Sign up on [Razorpay](https://razorpay.com/), and retrieve your Key ID and Key Secret.
3. **Create Brevo Account**: Sign up at [Brevo](https://www.brevo.com/) and generate a new SMTP Key.

# Folder Structure

* + **frontend:**
  1. **frontend/**: Main directory for the React frontend.
  2. **public/**: Contains static files such as index.html and images.
  3. **src/**: The main source code directory, divided into:
  + **assets/**: Global resources like images and fonts.
  + **components/**: Reusable UI components (e.g., Navbar, Footer, Product Card).
  + **pages/**: Different views/pages (e.g., Home, Product Details, Cart, Checkout, Profile).
  + **context/**: Manages global state (e.g., Cart Context, AUTH Context for cart and user authentication).
  + **services/**: Handles API calls and interactions with external services (e.g., api.js, auth.js).
  + **utils/**: Helper functions for common tasks (e.g., format Price, paginate).
  + **App.js**: The root component of the app that renders the structure.
  + **index.js**: Entry point for rendering the React app into the DOM.
  1. **.env**: Stores environment variables (e.g., API URLs, keys).

The **frontend** of the MERN eCommerce platform is structured under the frontend/ directory, which contains all React-related files. The public/ folder holds static assets like index.html and images, while the src/ folder includes the main codebase, organized into subfolders for assets, reusable components, page views, global context for state management, API services, and utility functions. Key files like App.js define the main component tree, and index.js bootstraps the React app. Additionally, a .env file is used to manage environment variables like API URLs and keys.

**Backend:** MERN eCommerce platform, starting from the backend folder:

1. **backend/**: Main directory for the Node.js/Express backend.

* **config/**: Contains configuration files, such as database connection setup.
* **controllers/**: Holds logic for handling incoming requests (e.g., product, user, order controllers).
* **data/**: Includes sample data used for seeding the database (e.g., products and users).
* **middleware/**: Contains custom middleware like authentication, error handling, and request logging.
* **models/**: Defines MongoDB schemas using Mongoose (e.g., User, Product, Order models).
* **routes/**: Defines API routes for different functionalities (e.g., user routes, product routes).
* **utils/**: Utility functions like token generation, email handling, etc.
* **seeder.js**: Script for importing or destroying sample data in the database.
* **server.js**: Entry point for the backend server; initializes Express, connects to MongoDB, and sets up middleware and routes.

The **backend** resides in the backend/ directory and is built using Node.js and Express. It includes a config/ folder for database setup, controllers/ for request-handling logic, and data/ for sample product and user data. The middleware/ folder provides custom functions like authentication and error handling, while models/ defines Mongoose schemas. The routes/ folder maps API endpoints, and utils/ contains helper functions such as token generation. seeder.js is used for database seeding, and server.js serves as the backend entry point, managing server configuration and MongoDB connection.

# Running the Application

To run the MERN eCommerce application locally, you need to start both the frontend and backend servers:

* **Frontend**: Navigate to the frontend directory and run npm start. This will start the React development server on the default port (usually http://localhost:3000).
* **Backend**: Navigate to the backend directory and run npm start. This will start the Node.js/Express server, typically running on http://localhost:5000.

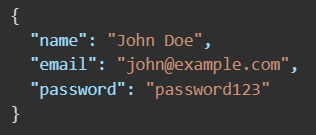
Make sure all dependencies are installed and environment variables are properly configured before starting the servers.

# API Documentation

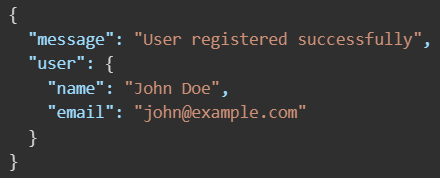
**1. User Endpoints:**

**1.1** **User Registration**

* + **Method**: POST
  + **Endpoint**: /api/users/register
  + **Request Body**:

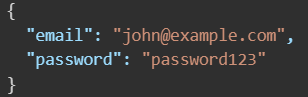


* + **Response Body:**

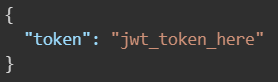
****

**1.2. User Login**

* **Method: POST**
* **Endpoint: /api/users/login**
* **Request Body:**

****

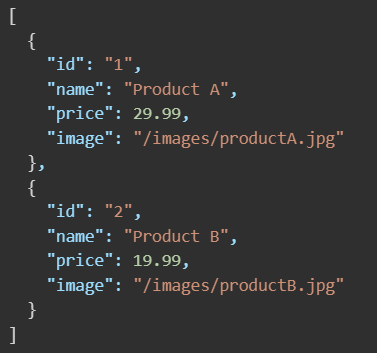
* **Response Body:**

****

**2. Product Endpoints**

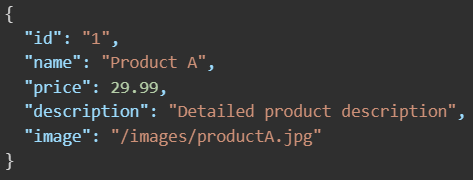
**2.1. Get All Products**

* **Method**: GET
* **Endpoint**: /api/products
* **Response**:



**2.2. Get Product by ID**

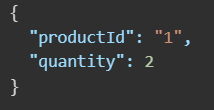
* **Method**: GET
* **Endpoint**: /api/products/:id
* **Response**:



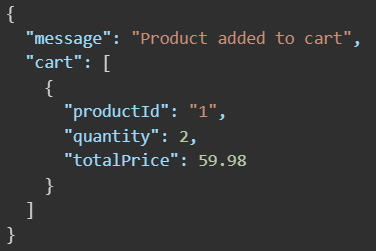
**3. Cart Endpoints**

**3.1. Add Product to Cart**

* **Method**: POST
* **Endpoint**: /api/cart
* **Request Body**:



* **Response Body:**

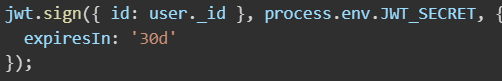
****

# Authentication

The project uses **JWT (JSON Web Tokens)** for handling authentication and authorization in a stateless and secure manner. Here's how it works:

* **Authentication Workflow**

1. **User Registration/Login**:
   * When a user registers or logs in, the server validates the credentials.
   * On success, a JWT token is generated using the user’s ID and a secret key defined in .env (JWT\_SECRET).
2. **Token Generation**:
   * The token is signed using the jsonwebtoken library.
   * Example:

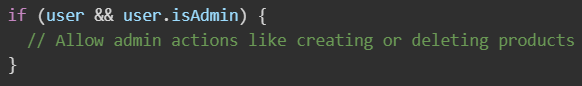


1. **Token Storage**:
   * The token is sent back in the response and stored by the frontend in local Storage or cookies (commonly local Storage for simplicity).
   * The token is then attached to the headers for subsequent requests:



* **Authorization Workflow**

1. **Protected Routes**:
   * Middleware (authMiddleware.js) is used to protect sensitive routes such as:
     + /api/users/profile
     + /api/orders/myorders
     + /api/products/:id/reviews
2. **Token Verification**:
   * The middleware extracts the token from the Authorization header.
   * It verifies the token using jwt.verify().
   * If valid, it attaches the decoded user ID to the request object (req.user) for access in controllers.
3. **Role-based Authorization**:
   * The project supports **admin-level access**.
   * Example:



# User Interface:

# 

# Screenshots:

# 

# 

# 

# 

# 

# 

# Testing

* **Tools Used:**
* **Jest**: Primary testing framework used for unit and integration testing on both frontend and backend.
* **Supertest**: Used to test backend API endpoints by simulating HTTP requests.
* **React Testing Library**: For testing React components in the frontend, focusing on user interactions and rendering logic.
* **Postman**: For manual testing of RESTful APIs during development.
* **MongoDB Memory Server (optional)**: For running backend tests without affecting the main database.

# Screenshots or Demo

* + Video Link: https://drive.google.com/file/d/1lxUJGpb3kek6FIit9icPfAT9WeVuR9\_6/view?usp=sharing

# Known Issues

The current version of the MERN eCommerce platform has a few known issues, including occasional loss of cart state on page reload, limited mobile responsiveness in some components, and lack of scroll restoration after navigation. Backend concerns include the absence of API rate limiting, no automatic redirection on JWT token expiry, and potentially verbose error messages in development mode. Additionally, image uploads and Razorpay webhook handling (if used) may encounter issues without proper configuration or error logging.

# Future Enhancements

Potential future improvements for the MERN eCommerce platform include adding real-time order tracking, implementing product recommendations using machine learning, integrating social login (Google, Facebook), enabling multi-language support, and improving mobile responsiveness. Additional features like wish list functionality, product comparison, coupon codes, admin analytics dashboard, and push notifications could further enhance user experience and platform functionality.