Full Stack Development with MERN Project Documentation format

# Introduction

* + **Project Title:** Desi Shop
  + **Team Members:** 1. Vaishnav Pramod Ahire - Database.

**2**. Om Somnath Pawar – Front End

3. Sanket Sanjay Pawar – Management features

4. Abhishek Taur – Back End

# Project Overview

* **Purpose**: The Online Shopping project is designed to provide a comprehensive platform for learning and practicing full-stack web development using the MERN stack (MongoDB, Express.js, React.js, Node.js). The purpose of this project is to simulate a real-world e-commerce experience, allowing users to browse products, manage their shopping cart, and go through a simulated checkout process. It is a hands-on learning tool that covers a wide array of concepts such as user authentication, API creation, database management, and the development of responsive and interactive user interfaces. The primary goal of the project is to help developers understand the workings of a modern e-commerce website by building both the client-side (using React) and server-side (using Node.js and Express) components. It will also give developers the opportunity to practice creating and managing MongoDB databases, handling secure user authentication, and implementing features like shopping carts, order management, and payment simulations. The project serves as an excellent introduction to full-stack development, enabling developers to build, test, and deploy a real-world application while gaining hands-on experience with both front-end and back-end technologies.  
  + **Features:** The Online Shopping project includes a variety of features designed for both customers and admins. For customers, it offers secure user authentication with JWT, product browsing and searching by categories or filters, detailed product pages, and a shopping cart where users can add, remove, or update items. The checkout process is simulated with an order summary and mock payment integration. Customers can also manage their profiles, view order history, and update account details. For admins, the platform includes secure login, product management (add, edit, delete), order management (view and update status), and user management (view and manage customer accounts). The backend provides RESTful APIs with secure routes, MongoDB database integration to store products, users, and orders, and error handling and logging for monitoring. The application is fully responsive, with smooth navigation and a clean admin dashboard, making it a complete e-commerce system built using the MERN stack.

# Architecture

* + **Frontend:** The frontend of the Online Shopping project is built using React.js, following a component-based architecture where each part of the user interface is encapsulated in reusable components like the header, footer, product listings, cart, and checkout pages. React Router is used for seamless client-side navigation between pages such as the homepage, product listings, shopping cart, checkout, and user profile. The app leverages React’s state management through the Context API or Redux for managing global states like user authentication and cart data. Data is fetched from the backend through RESTful APIs using Axios or Fetch, and is dynamically rendered in the UI. Components are styled using CSS-in-JS (e.g., Styled-Components) or traditional CSS/SCSS for responsive design across devices. Authentication is handled with JWT tokens stored in local Storage for secure access to protected routes. Form handling and validation are done with controlled components, with real-time validation for user inputs. The frontend also includes error handling, user feedback (via toast notifications), and performance optimization techniques like lazy loading and code splitting to ensure a smooth and responsive user experience.
  + **Backend:** The backend of the Online Shopping project is built using Node.js and Express.js, following a modular and scalable architecture. The core structure includes a central server that handles HTTP requests, routes them to appropriate controllers, and communicates with the database. The application uses Express.js to define routes and middleware for different functionalities, such as user authentication, product management, order processing, and user profile updates. The backend integrates with MongoDB, a NoSQL database, to store and manage data such as product details, user information, and order history. MongoDB collections are structured to handle products, users, orders, and cart data efficiently. JWT (JSON Web Tokens) is used for secure user authentication, ensuring that users can access protected routes like their profile and checkout. Middleware is employed for tasks such as request validation, error handling, and ensuring that routes are accessible only to authorized users. The server-side application uses RESTful API architecture to expose endpoints for interacting with the database, such as adding products, updating orders, and managing user data. The backend also includes features like logging, proper error handling, and environment configuration with a .env file for storing sensitive data like database credentials. This architecture ensures that the backend is modular, maintainable, and capable of handling requests efficiently while providing a secure and responsive user experience.
  + **Database**: The Online Shopping project uses MongoDB as its NoSQL database, storing key collections such as Products, Users, Orders, and Carts. The Product collection holds details like product name, description, price, stock, and images, while the User collection stores user information including email, password, role, and order history. The Order collection tracks user orders with references to products, order status, and payment information. The Cart collection stores the user’s selected products, including quantity. The backend interacts with the database using RESTful APIs to perform CRUD operations on these collections, like adding products, managing user authentication with JWT tokens, adding items to the cart, and processing orders. Data is validated with Mongoose schemas to ensure correct formats, and relationships between collections are established via ObjectId references, optimizing data retrieval. Indexes on frequently queried fields improve query performance, and MongoDB's built-in security ensures safe access and data handling. This setup provides a scalable, efficient database architecture for handling dynamic e-commerce data.

# Setup Instructions

* + **Prerequisites:** To run the Online Shopping project, you'll need the following software dependencies: Node.js (version 14 or higher), which is required to run the server-side application, and npm (Node Package Manager), which comes with Node.js to manage project dependencies. You'll also need MongoDB (either locally installed or using a cloud-based service like MongoDB Atlas) for the NoSQL database. If you choose to work with a local installation of MongoDB, ensure that it is up and running before starting the application. Additionally, Postman can be helpful for testing API endpoints, though it is optional for this project.
  + **Installation:** To set up the Online Shopping project, first clone the repository from GitHub using the command git clone https://github.com/tiendung8a6/Online-Shopping-MERN-Stack.git. Then, navigate to the server directory and run npm install to install the backend dependencies. Similarly, go to the client-admin and client-customer directories and run npm install to install the frontend dependencies for the admin and customer interfaces. Ensure that MongoDB is running locally or configure a cloud-based instance like MongoDB Atlas, and update the connection URL in the MyConstants.js file found in server/utils/. Next, create a .env file in the server directory with the necessary environment variables: MONGO\_URI for the MongoDB connection URL and JWT\_SECRET for the secret key used in JWT encoding. Once the configuration is complete, start the backend server by running npm start in the server directory, and then start the admin and customer frontends with npm start in the client-admin and client-customer directories. The Admin Interface will be available at http://localhost:3000, and the Customer Interface at http://localhost:3001. This setup will get the Online Shopping project up and running locally.

# Folder Structure

frontend/ – React Client App

1. Represents the admin interface of the application used for managing products, orders, users, and site content.
2. Contains the following subfolders and files:
   * public/  
     • Stores static files like index.html, images, and favicons used by the admin UI.
   * src/  
     • Main source code directory containing all frontend logic and UI components.
     + assets/  
       • Stores images, icons, and other static resources used in the admin dashboard.
     + components/  
       • Reusable UI components like tables, modals, forms, navigation bars, and buttons.
     + pages/  
       • Contains complete pages like Dashboard.js, ProductManagement.js, UserList.js, etc.
     + redux/ *(or store/)*  
       • Manages global state using Redux (actions, reducers, and store configuration).
     + services/  
       • API functions used to fetch or send data between the frontend and backend.
     + styles/  
       • Contains CSS or SCSS files for styling the admin interface.
     + utils/  
       • Utility/helper functions like formatters, validators, or authentication checkers.
   * App.js  
     • Root component managing routing and structure of the admin panel.
   * index.js  
     • Entry point of the React app, rendering the root App component into the DOM.

* **backend/ – Node.js + Express Server**
* middleware/ – Auth & validation logic.
* models/ – Mongoose schemas (User, Product, Order).
* routes/ – API route files:
* cartRoutes.js, orderRoutes.js, paymentRoutes.js, productRoutes.js, userRoutes.js
* server.js – Main backend entry point.
* seed.js – Adds sample data to MongoDB.
* .env – Stores secrets and DB URLs.
* package.json – Backend dependencies and scripts.

# Running the Application

To run the application locally, follow these steps:

* **Backend:**
  + Open a terminal.
  + Navigate to the backend directory.
  + Run the command: npm start
  + This will start the Express server on http://localhost:5000.
* **Frontend:**
  + Open a new terminal.
  + Navigate to the frontend directory.
  + Run the command: npm start
  + This will start the React app on http://localhost:3000.

Make sure MongoDB is running in the background before starting the backend server. Once both servers are up, the frontend and backend will work together to run the full application.

# API Documentation

Product API

* GET /api/products – Get all products
* GET /api/products/:id – Get product by ID
* POST /api/products – Create product (Admin)
* PUT /api/products/:id – Update product (Admin)
* DELETE /api/products/:id – Delete product (Admin)

User API

* POST /api/users/register – Register new user
* POST /api/users/login – Login user and get token
* GET /api/users/profile – Get logged-in user profile (Requires token)

Cart API

* GET /api/cart – Get user’s cart (Protected)
* POST /api/cart – Add product to cart
* DELETE /api/cart/:id – Remove item from cart

Order API

* POST /api/orders – Place a new order
* GET /api/orders – View order history

Payment API

* POST /api/payment – Simulate payment process

All protected routes require a valid Authorization: Bearer <token> header.

# Authentication In this project, authentication and authorization are handled using JWT (JSON Web Tokens) to ensure secure access to protected routes and user-specific operations.

# Authentication Workflow:

# User Registration & Login:

# When a user registers or logs in, the server validates the credentials and generates a JWT token.

# This token is sent back to the client and typically stored in localStorage or sessionStorage.

# Token-Based Access:

# For all protected routes (e.g., profile, cart, orders), the client must send the token in the Authorization header:

# Authorization: Bearer <JWT\_TOKEN>

# Backend Verification:

# A custom middleware function checks the token's validity using a secret key from .env.

# If the token is valid, it allows access and attaches the user’s data to the request object.

# Authorization:

# The token includes user role information (e.g., "admin" or "customer").

# Based on this role:

# Regular users can only access their own data (e.g., cart, orders).

# Admin-only routes (like adding or deleting products) are restricted using role-based checks in middleware.

# 

# User Interface

# Screenshots

# 

# Testing

We used a combination of unit, integration, and end-to-end testing to ensure the

reliability of our MERN stack Online Shopping project.

* **Frontend**: Tested using **Jest** and **React Testing Library** to verify component behaviour.
* **Backend**: Used **Mocha**, **Chai**, and **Super test** to test API endpoints and logic.
* **End-to-End**: **Cypress** was used to simulate real user flows like browsing, adding to cart, and checkout.

Key functionalities such as authentication, cart, and payment were prioritized during

testing.

# Screenshots or Demo

* + Video link.  
    https://drive.google.com/file/d/1WF0OOgV4BBunRcidq\_MGpbgtZckGtAmw/view?usp=drive\_link

# Known Issues

* + **Responsive Design**: Minor layout issues on some mobile screen sizes.
  + **Cart Sync**: Cart items occasionally don’t sync properly across multiple devices.
  + **Payment Gateway**: In sandbox mode, payment response delays may occur.
  + **Error Handling**: Some API errors lack detailed user feedback messages.

These issues are being tracked and will be addressed in future updates.

# Future Enhancements

* + **Wishlist Feature**: Allow users to save products for later.
  + **Product Reviews & Ratings**: Enable users to leave feedback on products.
  + **Admin Dashboard**: For better management of products, orders, and users.
  + **Live Chat Support**: Add real-time customer support functionality.
  + **Improved Analytics**: Track user behavior and sales trends with dashboards.
  + These features aim to enhance user experience and platform functionality