E-COMMERCE USING MERN STACK

NAAN MUDHALVAN PROJECT REPORT

Submitted by

PAVAN KUMAR U	**(311521243036)**
SARATHI J	**(311521243045)**
KEERTHANA V	**(311521243024)**
JANANI K	**(311521243017)**

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

MEENAKSHI SUNDARARAJAN ENGINEERING COLLEGE,

KODAMBAKKAM, CHENNAI-24

ANNA UNIVERSITY: CHENNAI 600 025

DEC 2024

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report "**E-COMMERCE USING MERN STACK** " is the bonafide work of **PAVAN KUMAR U (311521243036), SARATHI J(311521243013), KEERTHANA V (311521243024), JANANI K (311521243017)** who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
Mrs.N.MATHANGI, M.E., M.B.A, [Ph.D.]	Mrs. P. Muthulakshmi.,M.E
HEAD OF THE DEPARTMENT	**ASSISTANT PROFESSOR**
Artificial Intelligence and Data Science	Artificial Intelligence and Data Science
Meenakshi Sundararajan Engineering College	Meenakshi Sundararajan Engineering College No.
363, Arcot Road, Kodambakkam,	No. 363, Arcot Road, Kodambakkam,
Chennai -600024	Chennai – 600024

Submitted for the project viva voce of Bachelor of Technology in Computer Science and Engineering held on ________.

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

First and foremost, we express our sincere gratitude to our Respected Correspondent **Dr. K. S. Lakshmi,** our beloved Secretary Mr. N. Sreekanth, Principal Dr. S. V. Saravanan for their constant encouragement, which has been our motivation to strive towards excellence.

Our primary and sincere thanks goes to **Mrs.N.MATHANGI** M.E., M.B.A, [Ph.D.], Associate Professor Head of the Department, Department of Computer Science and Engineering, for her profound inspiration, kind cooperation and guidance.

We're grateful to **Mrs. P. Muthulakshmi.**,M.E,Internal Guide, Assistant Professor as our project coordinators for their invaluable support in completing our project. We are extremely thankful and indebted for sharing expertise, and sincere and valuable guidance and encouragement extended to us.

Above all, we extend our thanks to God Almighty without whose grace and Blessings it wouldn't have been possible.

ABSTRACT

This project aims to develop a dynamic and user-friendly e-commerce platform tailored to provide a seamless and personalized shopping experience for both customers and sellers. Designed with a user-friendly interface, the platform enables effortless navigation through a vast catalog of products, ensuring that customers can easily discover items that match their preferences. With detailed product descriptions, authentic customer reviews, and real-time updates on discounts and promotions, JAPS MART empowers users to make well-informed purchasing decisions. The intuitive search and filtering options further enhance the efficiency of product discovery, saving customers time and effort.

The platform features a secure and streamlined checkout process that prioritizes user convenience and trust. Customers benefit from encrypted transactions, multiple payment options, and instant order confirmations, ensuring a hassle-free experience from browsing to purchase. For sellers, JAPS MART offers a robust suite of tools through its advanced dashboard, enabling efficient order management and inventory tracking. Sellers can leverage insightful analytics to understand customer behaviour, monitor sales trends, and optimize their business strategies to drive growth and profitability.

JAPS MART also focuses on providing a personalized shopping journey for every user by incorporating data-driven recommendations based on individual preferences and browsing history. This approach not only enhances customer satisfaction but also increases engagement and retention. For sellers, the platform fosters scalability and success by offering actionable insights and tools for effective business management.

By combining advanced technology with a customer-centric approach, JAPS MART aims to revolutionize the online shopping experience. It bridges the gap between buyers and sellers, creating a dynamic ecosystem that promotes convenience, trust, and growth. Whether for personal shopping or business success, JAPS MART is the future of online marketplaces.

This project demonstrates the practical application of full-stack development using the MERN stack and highlights the potential of modern web technologies in e-commerce solutions.

TABLE OF CONTENTS

CHAPTER 1

1. PROJECT OVERVIEW

1.1 PURPOSE

The purpose of this project is to create a scalable and user-friendly online shopping experience by creating a platform that caters to the needs of both customers and sellers in a seamless and efficient manner. For customers, the platform aims to simplify product discovery and decision-making through a user-friendly interface, detailed product information, and personalized recommendations. By integrating features such as secure checkout processes and instant order confirmations, Japs Mart seeks to enhance customer trust and satisfaction, ensuring a smooth end-to-end shopping journey. For sellers, the purpose extends to providing a comprehensive business management toolkit that includes efficient order processing, inventory tracking, and access to insightful analytics. These tools empower sellers to make data-driven decisions, improve operational efficiency, and drive business growth. By bridging the gap between buyers and sellers, Japs Mart fosters a mutually beneficial ecosystem that not only meets but exceeds the expectations of modern e-commerce participants. Ultimately, the project seeks to establish a reliable, scalable, and innovative platform that sets new benchmarks for convenience, engagement, and value in the online shopping industry. while demonstrating the potential of full-stack web development for modern e-commerce solutions.

1.2 FEATURES

User-Friendly Interface:

A responsive and intuitive React-based frontend for seamless navigation and enhanced user experience across devices.

User Authentication and Authorization:

Secure login and registration using encrypted credentials.Role-based access control for users and administrators.

Effortless Product Discovery:

Intuitive product categorization and navigation for an enhanced user experience. Personalized product recommendations based on browsing history and preferences.

Insightful Business Analytics:

Visualized data on customer behavior, sales trends, and market demand. Actionable insights to optimize product offerings and marketing strategies

.

Shopping Cart and Wish-list:

Add items to the cart for immediate purchase or save them to a wish-list for future reference.

Order Management:

Track order history, status, and delivery updates for users .Admin panel for managing orders, cancellations, and returns.

Inventory Management:

Admin capabilities to add, update, or remove products from the inventory.

Payment Gateway Integration:

Secure payment processing through trusted gateways with support for multiple payment methods.

User Reviews and Ratings:

Allow users to provide feedback to help others make informed choices.

.

CHAPTER 2

2. ARCHITECTURE

The **e-commerce** application is built on the MERN stack (MongoDB, Express.js, React, Node.js), delivering a full-stack solution with efficient data handling, responsive user interactions, and real-time updates. By leveraging the strengths of each component in the stack, the architecture supports a seamless shopping experience while ensuring scalability and secure management of inventory and user data. The client-server model separates concerns between the frontend and backend. The frontend provides a dynamic user interface for customers and admins, while the backend manages business logic, data processing, and secure communication.

2.1 FRONTEND ARCHITECTURE

- **UI (React):** The user interface built using the React library.
- **Stocks API:** A frontend API that communicates with the backend API to fetch data

2.2 BACKEND ARCHITECTURE

- **Express JS:** A Node.js framework for building web applications and APIs**.**
- **Node JS:** A JavaScript runtime environment for building backend applications.

2.3 DATABASE ARCHITECTURE

- **Mongoose:** An Object Data Modeling (ODM) library for MongoDB.
- **MongoDB**: A NoSQL database that provides flexible data storage and retrieval.

CHAPTER 3

3. SETUP INSTRUCTIONS

3.1 PREREQUISITES

Before setting up the application, make sure the following prerequisites are met:

1. **Operating System**

   A Windows 8 or higher machine is recommended, though the setup should also work on macOS and Linux systems.

2. **Node.js**

   Download and install Node.js (version 14 or above). Node.js is required to run both the backend server (Node and Express) and the frontend server (React).

3. **MongoDB**

   Local MongoDB Installation: Install MongoDB Community Edition from MongoDB's official site if you prefer to use a local database.

   MongoDB Atlas (Optional): Alternatively, you can set up a free MongoDB Atlas account to use MongoDB in the cloud. MongoDB Atlas provides a connection string that will be needed to configure the backend.

4. **Two Web Browsers**

   The application works best with two web browsers installed for simultaneous testing (e.g., Google Chrome, Mozilla Firefox).

5. **Internet Bandwidth**

A stable internet connection with a minimum speed of 30 Mbps is recommended, especially if using MongoDB Atlas or deploying to a remote server.

6. **Code Editor**

   Visual Studio Code or any other preferred code editor for easy management of the codebase and environment configuration.

3.2 INSTALLATION

1. **Install Node.js and MongoDB**

   Node.js: Download and install Node.js (Version 14 or above).

   MongoDB: Download and install the MongoDB Community Edition and set up MongoDB on your machine. Alternatively, you can use MongoDB Atlas for cloud hosting.

2. **Clone the Repository**

   Open a terminal and clone the project repository:

   git clone <repository_url> cd

3. **Install Backend Dependencies**

   Navigate to the backend folder and install the necessary Node.js packages:

   cd backend npm

   install

4. **Install Frontend Dependencies**

   Open a new terminal window or navigate back to the root directory, then go to the frontend folder and install dependencies.

   cd ../frontend npm

   install

5. **Configure Environment Variables**

- In the backend folder, create a env file.
- Add the following environment variables:

MONGO_URI=<your_mongodb_connection_string>

PORT=8000

JWT_SECRET=<your_jwt_secret>

6. **Run MongoDB**

If using MongoDB locally, ensure the MongoDB server is running:

mongod

7. **Start the Backend Server**

In the backend folder, start the server with the following command:

npm start

This will start the backend server on http: / / local host : 8000 .

8. **Start the Frontend Server**

In the frontend folder, start the React development server:

npm start

This will start the frontend server on http: / / local host : 3000 .

9. **Access the Application**

Open your web browser and navigate to http: / / local host : 3000 to view and interact with the application

CHAPTER 4

4. FOLDER STRUCTURE

CLIENT: REACT FRONTEND STRUCTURE

The frontend is structured using the following folders:

```
frontend/
│
├── public/                    # Static files that are served directly
│   ├── index.html                # Main HTML template for the app
│   ├── favicon.ico               # Favicon for the app
│   ├── robots.txt               # Web crawlers instructions
│   └── manifest.json               # App metadata for PWA features
│
├── src/                       # All the source files for React application
│   ├── assets/                  # Images, fonts, icons, and other static assets
│   │   ├── images/                # Folder for images used in the app (e.g., logos, banners)
│   │   └── fonts/               # Folder for custom fonts
│   │
│   ├── components/                # Reusable UI components
│   │   ├── Header.js              # Header component (Navbar)
│   │   ├── Footer.js              # Footer component
│   │   ├── ProductCard.js           # Component to display individual products
│   │   ├── CartItem.js            # Component to display items in the cart
│   │   ├── ProductList.js           # List of products
│   │   └── Loader.js              # Loading spinner component
│   │
│   ├── context/                 # Global state management (e.g., Context API)
│   │   ├── CartContext.js           # Cart state management
│   │   ├── AuthContext.js           # User authentication state
│   │   └── ProductContext.js          # Product data management
│   │
│   ├── pages/                   # Components representing different pages of the app
│   │   ├── Home.js                # Home page component (product listings, promotions)
│   │   ├── Cart.js               # Cart page where users review their order
│   │   ├── ProductDetail.js          # Product details page
│   │   ├── Profile.js              # User profile page
│   │   ├── AdminDashboard.js          # Admin dashboard page (for sellers/admin users)
│   │   └── NotFound.js              # 404 page not found component
│   │
│   ├── services/                # API request logic (handling server-side communication)
│   │   ├── authService.js          # Functions for login, registration, and token management
│   │   ├── productService.js          # Functions for fetching product data
│   │   ├── orderService.js          # Functions for order-related actions
│   │   └── cartService.js          # Functions for managing cart items
│   │
│   ├── styles/                # Global CSS styles
│   │   ├── main.css              # Main global styles
│   │   ├── theme.css              # Application-wide theme styles (colors, fonts, etc.)
│   │   └── productCard.css          # Styles for product card component
│   │
│   ├── utils/                # Utility functions and helper methods
│   │   ├── formatPrice.js          # Format prices (e.g., currency formatting)
│   │   ├── validation.js          # Form validation logic
│   │   └── authUtils.js          # Helper functions related to user authentication
│   │
│   ├── hooks/                # Custom React hooks
│   │   ├── useAuth.js              # Custom hook for managing user authentication
│   │   ├── useCart.js              # Custom hook for managing cart state
│   │   └── useProducts.js          # Custom hook for fetching products
│   │
│   ├── App.js                # Main app component that holds the routing structure
│   ├── index.js              # Entry point of the application
│   ├── routes.js              # React Router routes configuration
│   ├── .env                # Environment variables for API URL, keys, etc.
│   ├── package.json              # Frontend dependencies and scripts
│   └── README.md              # Frontend project overview and setup instructions
│
└── .gitignore                # Ignore node_modules, build files, and other unnecessary files
```

4.1 SERVER: NODE.JS BACKEND STRUCTURE

The backend server is structured as follows: server/

```
│ backend/

│

├── config/                # Configuration files for the backend setup

│   ├── dbConfig.js              # MongoDB or MySQL connection configurations

│   ├── jwtConfig.js              # JWT token secret, expiration time, and related settings
│   └── envConfig.js              # Other environment-specific configurations
│
├── controllers/                  # Controller functions to handle business logic
│   ├── authController.js         # Handles login, signup, password reset logic
│   ├── cartController.js         # Cart-related logic (add, remove, update items)
│   ├── orderController.js        # Handles order creation, processing, and history
│   ├── productController.js      # Handles fetching and managing products
│   └── adminController.js        # Admin-related actions (e.g., product and user management)
│
├── middleware/                   # Custom middleware functions (e.g., for authentication)
│   ├── authMiddleware.js         # Verifies the JWT token and ensures user is authenticated
│   ├── adminMiddleware.js        # Ensures that the user has admin privileges
│   ├── errorMiddleware.js        # Global error handling middleware
│   └── validateMiddleware.js     # Middleware to validate request parameters or body data
│
├── models/                       # Database schemas/models
│   ├── User.js                   # User model/schema (e.g., name, email, password)
│   ├── Product.js                # Product model/schema (e.g., title, description, price, stock)
│   ├── Cart.js                   # Cart model/schema (e.g., userId, items, totalAmount)
│   ├── Order.js                  # Order model/schema (e.g., order details, status, shipping info)
│   └── Payment.js                # Payment schema (if applicable, for tracking payments)
│
├── routes/                       # API route definitions (e.g., product, user, order routes)
│   ├── authRoutes.js             # Routes for authentication (login, signup, logout)
│   ├── cartRoutes.js             # Routes for cart actions (add, update, remove items)
│   ├── orderRoutes.js            # Routes for order management
│   ├── productRoutes.js          # Routes for product management (fetch, add, update, delete)
│   └── adminRoutes.js            # Routes for admin (dashboard, manage users, products)
```

```
│
├── services/                  # Business logic related to specific domains (e.g., order service)
│   ├── authService.js             # Logic for handling authentication (token generation, password hashing)
│   ├── cartService.js             # Logic for managing the cart (adding items, calculating total)
│   ├── orderService.js            # Logic for processing orders, calculating order total, etc.
│   ├── productService.js          # Logic for product-related operations (CRUD actions)
│   └── paymentService.js          # Logic for payment processing (if needed)
│
├── utils/                     # Utility functions
│   ├── logger.js                  # Logger utility (for logging requests, errors, etc.)
│   ├── formatPrice.js             # Utility for formatting price information (e.g., currency formatting)
│   ├── validation.js              # Utility for validating input (e.g., email, password)
│   └── generateToken.js           # Utility for generating JWT tokens
│
├── validations/               # Input validation schemas (using Joi, Yup, etc.)
│   ├── authValidation.js          # Validation for login/signup forms
│   ├── productValidation.js       # Validation for product-related data
│   ├── orderValidation.js         # Validation for order creation
│   └── cartValidation.js          # Validation for cart data
│
├── .env                       # Environment variables (DB connection strings, JWT secret)
├── app.js                     # Main entry point for the app (Express setup and middlewares)
├── server.js                  # Server setup and initialization
├── package.json               # Backend dependencies and scripts
├── package-lock.json          # Lock file for dependencies
└── README.md                  # Project setup instructions
```

CHAPTER 5

5. RUNNING THE APPLICATION

To run both the frontend and backend servers locally, follow these steps:

5.1 SET UP THE FRONTEND (SERVER)

1. Open another terminal window and navigate to the cl i ent directory:

   cd client

2. Install the frontend dependencies:

   npm install

3. Start the frontend server:

   npm start

The frontend server will launch at http: / / local host : 3000

5.2 SET UP THE BACKEND (SERVER)

1. Navigate to the server directory:

   cd server

2. Install the necessary backend dependencies:

   npm install

3. Create a env file in the root of the server directory and add your environment variables (e.g., MongoDB URI, JWT secret, etc.)

   MONGODB_URI=your_mongo_connection_url JWT_SECRET=your_jwt_secret

   PORT=8000

4.Start the backend server:

   npm start

   The server will run on http: / / local host : 8000 by default (unless you specify a different port in the env file).

CHAPTER 6

6. API DOCUMENTATION

The following section documents the endpoints exposed by the backend server of the **Japs Mart** e-commerce application. Each endpoint includes details on the HTTP request methods, parameters, and example responses.

6.1 ORDER A PRODUCT BY THE USER:

- **Endpoint**: /api/order/create
- **Method**: post
- **Description**: Allows a user to place an order by specifying the products in their cart, shipping details, and payment method.

REQUEST BODY

```
{
  "userId": "609d1b6e5b2b2c001f8b4567",
  "cartId": "609d1b7e5b2b2c001f8b4678",
  "shippingAddress": "456 Oak St, Springfield, IL, 62701",
  "paymentMethod": "Credit Card",
  "totalAmount": 491.34
}
```

EXAMPLE RESPONSE:

```
{
  "orderId": "609d1c5e5b2b2c001f8b5678",
  "userId": "609d1b6e5b2b2c001f8b4567",
  "status": "success",
  "message": "Order placed successfully",
  "orderDetails": {
    "products": [
      {
        "productId": "609d1a2f8b2b2c001f8b2345",
        "quantity": 2,
        "price": 245.67
      }
    ],
    "shippingAddress": "456 Oak St, Springfield, IL, 62701",
    "paymentMethod": "Credit Card",
    "totalAmount": 491.34
  }
}
```

CHAPTER 7

7. TESTING

To ensure the robustness and reliability of the e-commerce platform, a comprehensive testing strategy has been implemented, covering both frontend and backend functionality. This strategy includes unit tests, integration tests, end-to-end (E2E) tests, and manual testing to ensure the application functions as expected

7.1 UNIT TESTING:

Description: Unit tests are written to verify individual functions and modules in isolation. This helps identify any issues at the component level early in the development process. Unit tests are essential for validating logic within functions, utility methods, and small components.

Tools Used:

- **Jest**: Primarily used for testing JavaScript functions and modules on the backend.
- **Mocha** and **Chai**: Used for testing API endpoints, backend logic, and services such as payment processing.

Example Tests:

- Testing the login function to ensure correct user authentication.
- Verifying that the cart logic calculates the correct total amount based on product prices and quantities.
- Checking if the order confirmation sends the correct response after placing an order.

7.2 INTEGRATION TESTING:

Description: Integration tests are conducted to ensure that various modules and services work together as expected. These tests verify the interactions between different components of the system, including frontend-to-backend communication and database interactions.

Tools Used:

- **Jest** and **Enzyme** (for React): Used to test interactions between frontend components and ensure the UI works as intended when connected to backend services.
- **Supertest** (with Mocha): Used to test the integration of API routes, checking responses, and ensuring data flows properly between the server and database.

Example Tests:

- Testing the end-to-end process from adding a product to the cart and proceeding to checkout.
- Verifying that user registration results in creating a new user in the database and returns the correct response.
- Checking the interaction between product APIs and ensuring product data is correctly stored and fetched from the database.

7.3 END-TO-END (E2E) TESTING:

Description: E2E tests simulate real-world user interactions to ensure the application’s entire user journey functions as expected. These tests cover the entire process, from logging in to browsing products, adding items to the cart, placing an order, and receiving order confirmation.

7.3.1 Tools Used:

- **Cypress**: Automated browser-based testing tool that simulates real-world user interactions. Cypress is used to test the user journey from start to finish.
- **Playwright**: A next-generation tool used for automating modern web applications. It can handle complex scenarios and is an alternative to Cypress.

Example Tests:

- Verifying that a user can successfully browse through products, add them to the cart, and place an order.
- Testing that a user can log in, view their profile, and update their account details.
- Simulating a complete checkout process, from adding a product to the cart, entering shipping information, selecting a payment method, and completing the order.

7.3.2 Manual Testing:

Description:
In addition to automated tests, manual testing is conducted to ensure the application’s usability, accessibility, and responsiveness across different devices and screen sizes.

Scope:

- Verifying the correct display of layout, buttons, and images on various screen sizes (mobile, tablet, desktop).
- Checking the behavior of interactive elements such as dropdowns, modals, and form validation.
- Validating error messages for incorrect form submissions and ensuring proper user guidance is provided.

CHAPTER 8

ADVANTAGES:

• Full-Stack Solution

The **Japs Mart** platform utilizes a full-stack solution by integrating the MERN stack (MongoDB, Express.js, React, Node.js), which ensures seamless communication between all components. The stack is highly suitable for creating scalable, efficient, and feature-rich e-commerce platforms.

• Scalability:

MongoDB's flexible NoSQL database architecture allows for easy scalability. As the product catalog grows or user data increases, MongoDB's schema-less structure can accommodate the addition of new fields and features without disrupting the existing data.

• Efficient Data Handling

MongoDB excels in handling complex, non-relational data, which is essential for an e-commerce platform like **Japs Mart**. **React's** virtual DOM optimizes the UI rendering process, improving performance for data-heavy interactions such as browsing large product catalogs, applying filters, or updating the shopping cart.

• Interactive User Experience

With **React**, **ShopEZ** offers a highly dynamic and interactive user interface (UI). The platform supports smooth browsing, searching, and cart management, ensuring a seamless customer experience. React's component-based architecture also facilitates easy updates and feature additions, making it ideal for an evolving e-commerce platform.

• Cost-Effective Development

As a fully open-source stack, the **MERN** technologies eliminate licensing fees, reducing operational costs. Additionally, using **JavaScript** across both the frontend (React) and backend (Node.js) simplifies development, as developers only need to master a single language. This uniformity minimizes learning curves, reduces code duplication, and accelerates the overall development process, thus saving time and resources.

.

CHAPTER 9

DISADVANTAGES

• Learning Curve for MERN Stack

While the MERN stack simplifies development by using JavaScript across the stack, it requires a good understanding of multiple technologies—**React**, **Node.js**, **Express.js**, and **MongoDB**. Developers who are not familiar with these tools may face a steep learning curve, particularly when working with advanced features such as state management in React or non-relational database design in MongoDB.

• Performance Limitations of Node.js

Node.js uses a single-threaded, event-driven model that works well for handling a large number of I/O-bound tasks. However, it can struggle with CPU-intensive tasks, such as complex computations or image processing, potentially leading to performance bottlenecks in certain scenarios, like analytics dashboards or personalized recommendations.

• Limited Support for Complex Queries

While **MongoDB** is excellent for storing unstructured or semi-structured data, it has limitations when handling complex queries or transactions. For example:

- Joining multiple collections is less efficient compared to relational databases.
- Handling multi-document transactions may require careful design to ensure consistency, which could complicate development.

• Frontend Complexity with State Management

React's component-based architecture is powerful but requires careful management of state, especially for larger applications. Managing global states, such as a user's shopping cart or product filters, often requires additional libraries like **Redux** or **Context API**, which can increase development complexity.

CHAPTER 10

FUTURE ENHANCEMENTS:

The **Japs Mart** platform is designed to be scalable and adaptable, allowing for continuous improvement and expansion. Below are potential enhancements aimed at improving user experience, ensuring robust security, and scaling to meet future demands.

Enhanced User Experience

1. Implement dynamic filters for product searches, such as price range, brand, category, discount percentage, and ratings.
2. Add keyword-based predictive search functionality.

Improved Security

1. Implement granular permissions for administrators, sellers, and customers, ensuring that each role can only access authorized actions and data.

Expanded Features

1. Enable users to create wishlists for future purchases.
2. Notify users about price drops, flash sales, and product restocks.

Scalability Enhancements

1.Optimize product listing pages with server-side pagination and infinite scrolling for faster loading of large inventories.