Full Stack Development with MERN

Project Documentation format

1. Introduction

· **Project Title:** ShopEZ:One-Stop Shop for Online Purchases

· **Team Members And Roles:**

Name	**Roles**
Thathireddy Lokitha	Full Stack Developers(Backend & Frontend)
Yadavakunta Pradeep Reddy	Full Stack Developers(Backend & Frontend)
M Sai Dhiraj Kumar	Database & Payment Integration

2. Project Overview

Purpose :

- To develop a secure ecommerce web application using the MERN stack.
- To provide a smooth and user-friendly online shopping experience.
- To implement secure authentication using JWT and password encryption.
- To enable efficient product, cart, and order management.
- To integrate a secure online payment system using Razorpay.
- To design a scalable architecture using MongoDB Atlas.

Goals:

- To build a fully functional full-stack ecommerce application using MERN.
- To ensure secure user authentication and authorization.
- To implement efficient product browsing and search functionality.
- To enable smooth cart and order processing.
- To integrate secure online payment processing.
- To provide an admin panel for product and order management.
- To design a scalable and maintainable system architecture.

Features and Functionalities:

Authentication:

- User registration with email validation
- Secure login with JWT token
- Password hashing using bcrypt
- Role-based access (Customer / Admin)
- Profile management
- Protected API routes

Product Management:

- Add new product (Admin)
- Update product details
- Delete product
- View product listing
- View product details
- Basic search and category filtering

Cart Management:

- Add product to cart
- Update product quantity
- Remove product from cart
- Real-time total calculation
- Cart data persistence

Order Management:

- Create order from cart
- Generate unique order ID
- Store order details
- View order history
- Order status tracking

Payment Processing:

- Razorpay order creation
- Payment verification
- Payment success handling
- Payment failure handling
- Update order payment status

Admin Features:

- Admin login
- Manage products
- View users
- Monitor orders

Review Functionality:

- Add product review
- View product ratings

3. Architecture

Frontend:

1. Technology Stack

- React.js (UI Development)
- Vite (Fast build tool)
- Tailwind CSS (Styling)
- Axios (API communication)
- React Router (Client-side routing)

2. Architecture Design

Component-Based Structure

- Reusable components (Navbar, ProductCard, CartItem, etc.)
- Page-level components (Login, Register, Home, Cart, Orders, Admin Dashboard)

3. State Management

- useState and useEffect hooks
- Local state for forms and UI updates
- Token stored in localStorage for authentication

4. Routing

- Public Routes (Login, Register)
- Protected Routes (Cart, Orders, Admin)
- Role-based access using JWT

5. API Communication

- Axios used for sending HTTP requests
- Backend endpoints connected through REST APIs
- Token attached in Authorization header

6. UI Design

- Responsive layout
- Mobile-friendly design
- Clean and minimal interface

Backend:

Server Layer

- Express server handles HTTP requests
- Middleware for JSON parsing and error handling
- API routes defined under /api

Route Layer

Separate route files for better modularity:

- Auth Routes
- Product Routes
- Cart Routes
- Order Routes

- Payment Routes
- Admin Routes

Each route connects to its respective controller.

Controller Layer

- Contains business logic
- Handles request and response
- Performs validation
- Communicates with database

Middleware Layer

- JWT Authentication Middleware
- Role-Based Authorization
- Error Handling Middleware

Ensures secure and controlled API access.

Database Layer

- MongoDB Atlas (Cloud Database)
- Mongoose schemas for:
  - Users
  - Products
  - Orders
  - Cart
  - Payments

Payment Integration Layer

- Razorpay SDK integration
- Order creation
- Payment verification
- Signature validation

Security Implementation

- JWT token validation on protected routes
- Password hashing using bcrypt
- Role-based access control (Admin / User)

· **Database:**

- MongoDB Atlas cloud database
- Collections: Users, Products, Cart, Orders, Payments
- Data stored in JSON format using Mongoose schemas
- CRUD operations: save(), find(), update(), delete()
- Relationships handled using ObjectId references
- Secured with JWT authentication & bcrypt password hashing

4. Setup Instructions

Prerequisites:

- **Node.js** (v16+ recommended)
- **MongoDB Atlas** account (or local MongoDB)
- **Razorpay Account** (for payment integration)
- **VS Code**
- **Git**

Installation Steps

1.Clone the Repository

```
git clone <repository-url>
cd shopez
```

2.**Backend Setup**

```
cd server
npm install
```

Create .env file inside server folder:

```
PORT=5000
MONGO_URI=your_mongodb_connection_string
JWT_SECRET=your_secret_key
RAZORPAY_KEY_ID=your_key
RAZORPAY_SECRET=your_secret
```

3.**Frontend Setup**

```
cd client
npm install
```

4.Run the Application

▶ Start Backend

```
npm start
```

▶ Start Frontend

```
npm run dev
```

Now application runs on:

- Backend → http://localhost:5000
- Frontend → http://localhost:5173

5. Folder Structure

Client (React Frontend Structure)

```
frontend/
├── src/
│   ├── pages/       → Page components (Home, Login, Cart, Orders)
│   ├── components/   → Reusable UI components (Navbar, ProductCard)
│   ├── api/         → Axios API calls
│   ├── context/      → Global state management
│   └── App.jsx       → Main routing file
├── package.json
└── vite.config.js
```

Server (Node.js Backend Structure)

```
server/
├── models/       → Mongoose schemas (User, Product, Order)
├── routes/       → API route definitions
├── controllers/  → Business logic
├── middleware/   → JWT auth & error handling
├── config/       → Database connection
└── server.js     → Main entry file
```

6. Running the Application

```
cd server && npm start
cd client && npm run dev   (or npm start if using CRA)
```

7. API Documentation

Backend API (Simple Version)

Auth

- POST /auth/register → Create account
- POST /auth/login → Login & get token

Products

- GET /products → View products
- POST /products → Add product (Admin)
- PUT /products/:id → Update
- DELETE /products/:id → Delete

Cart

- POST /cart → Add item
- GET /cart → View cart

Orders

- POST /orders → Place order
- GET /orders → View orders

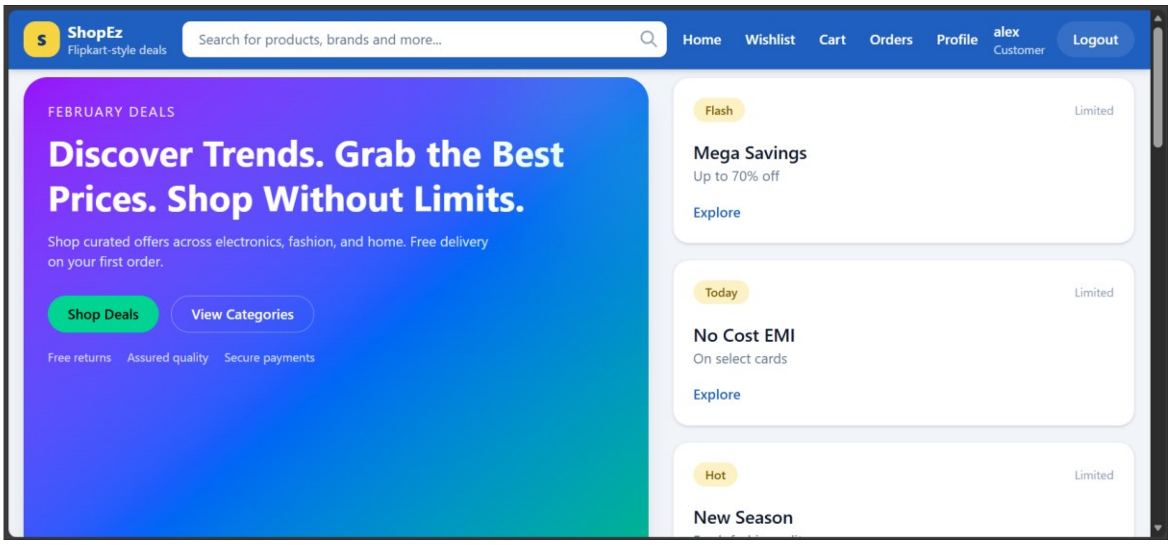
Payment

- POST /payment/create-order
- POST /payment/verify

8. Authentication

- Passwords are hashed using **bcrypt**.
- On login, backend generates a **JWT token**.
- Token is sent in request headers for protected routes.
- Backend verifies token before giving access.
- User roles (Admin/User) control permissions.

9. User Interface



10. Testing

- **Manual Testing** → All features (login, cart, order, payment) manually test chesam
- **API Testing** → Backend APIs ni **Postman** lo test chesam
- **Frontend Testing** → Browser lo UI validation & form checks
- **Authentication Testing** → Invalid token & role-based access test chesam
- **Payment Testing** → Razorpay test mode lo payment verification chesam

Tools Used

- Postman
- Chrome Browser
- MongoDB Atlas (DB verification)