**NAAN MUDHALVAN**

**PROJECT REPORT**

**Grocery Shop**

**Using MERN Stack**

TEAM MEMBERS

1.MAHESH J

2.MANIKANDAN K

3.MEGA VIGNESH

4.MANISHA

## 1. ****Introduction****

### 1.1. Project Overview

The Grocery Shop website is an online platform that allows users to browse, search, and purchase groceries. This project is developed using the MERN stack, which includes:

* **MongoDB**: A NoSQL database to store product details, user information, and order history.
* **Express.js**: A web application framework for Node.js used to handle server-side logic and routing.
* **React.js**: A JavaScript library for building the user interface and dynamic web pages.
* **Node.js**: A runtime environment for executing JavaScript on the server side.

### 1.2. Objective

The goal of this project is to build a seamless and efficient grocery shopping experience. Users can:

* Browse product categories.
* Search for groceries.
* Add items to their cart.
* Make secure payments.
* View order history and track shipments.

## 2. ****System Architecture****

The system follows a **client-server architecture**, with the frontend, backend, and database being connected.

### 2.1. ****Frontend (React.js)****

* **User Interface (UI)**: The frontend is built with React.js to create a responsive, interactive interface. Key pages include:
  + Home page with featured products.
  + Product listing page with filters (category, price, etc.).
  + Cart and Checkout pages.
  + Order confirmation and tracking.
* **State Management**: Redux or Context API is used for managing global state (e.g., user authentication, cart items, etc.).

### 2.2. ****Backend (Node.js + Express.js)****

* **RESTful APIs**: The server exposes REST APIs for various functionalities, including user authentication, CRUD operations for products, cart management, and order processing.
* **Authentication**: User authentication is handled using **JWT (JSON Web Tokens)** to secure API routes and ensure authorized access.
* **Payment Integration**: A payment gateway like **Stripe** or **PayPal** is integrated for handling transactions securely.

### 2.3. ****Database (MongoDB)****

* **Product Collection**: Stores information such as name, category, price, stock quantity, and image URL.
* **User Collection**: Stores user details, including username, email, password hash, and order history.
* **Order Collection**: Stores order details, including user ID, product list, payment status, and shipping information.

## 3. ****Technologies Used****

### 3.1. ****MongoDB****

MongoDB is used to store data in a flexible, JSON-like format. Collections are used to manage products, orders, and user data. MongoDB’s document-oriented nature makes it ideal for the variable product data.

### 3.2. ****Express.js****

Express.js serves as the backend framework that simplifies the routing and server-side logic, making it easier to define RESTful API endpoints and handle HTTP requests.

### 3.3. ****React.js****

React.js is used to create an interactive frontend. It enables building reusable components for the product listings, cart system, and checkout forms.

### 3.4. ****Node.js****

Node.js is used for running JavaScript on the server. It handles backend logic, API requests, and the interaction with the database.

### 3.5. ****JWT (JSON Web Tokens)****

JWT is used for implementing user authentication and securing the APIs. It allows users to log in, receive a token, and access protected routes.

### 3.6. ****Payment Integration (Stripe/PayPal)****

Stripe or PayPal is integrated to handle payments in a secure and user-friendly manner.

## 4. ****Features****

### 4.1. ****User Features****

* **Registration and Login**: Users can create an account or log in to access personalized services.
* **Product Search & Filter**: Users can search products by name, category, or price, and apply various filters to narrow down results.
* **Cart Management**: Users can add products to the shopping cart, update quantities, or remove items.
* **Checkout**: Secure checkout process where users can enter shipping details and make payments.
* **Order Tracking**: After checkout, users can view their order history and track delivery status.

### 4.2. ****Admin Features****

* **Product Management**: Admins can add, update, or delete products in the catalog.
* **Order Management**: Admins can view and manage orders, update order status, and manage shipment details.
* **User Management**: Admins can manage user details and roles.

## 5. ****API Endpoints****

Here’s an overview of the key RESTful API endpoints:

### 5.1. ****User Authentication****

* POST /api/auth/register: Register a new user.
* POST /api/auth/login: Log in and return a JWT token.
* GET /api/auth/user: Get user profile (protected route).

### 5.2. ****Product Management****

* GET /api/products: Fetch all products.
* GET /api/products/:id: Fetch a specific product by ID.
* POST /api/products: Add a new product (Admin only).
* PUT /api/products/:id: Update a product (Admin only).
* DELETE /api/products/:id: Delete a product (Admin only).

### 5.3. ****Cart Management****

* GET /api/cart: Get the user's cart.
* POST /api/cart: Add a product to the cart.
* PUT /api/cart/:id: Update product quantity in the cart.
* DELETE /api/cart/:id: Remove a product from the cart.

### 5.4. ****Order Management****

* POST /api/orders: Create an order from the cart (after checkout).
* GET /api/orders: Fetch order history.
* GET /api/orders/:id: Fetch details of a specific order.

## 6. ****Security Considerations****

* **Authentication**: The app uses JWT tokens to authenticate users and protect sensitive routes.
* **Data Validation**: Input data is validated both on the client side (React) and the server side (Express.js) to avoid invalid or malicious inputs.
* **Password Hashing**: User passwords are hashed using bcrypt before storing them in the database.
* **Payment Security**: Payment details are processed using secure APIs (Stripe/PayPal), ensuring PCI compliance.

## 7. ****Challenges Faced****

* **User Authentication**: Implementing secure authentication with JWT while ensuring that the user experience remains smooth.
* **State Management**: Managing the state of the cart and product filters across different pages and components was complex, requiring the use of Redux or Context API for efficient state management.
* **Payment Integration**: Integrating payment gateways like Stripe requires careful handling of sensitive user data and ensuring proper error handling during transactions.

## 8. ****Conclusion****

This Grocery Shop website demonstrates the effective use of the MERN stack in building a full-fledged e-commerce application. The system is scalable, easy to maintain, and provides a smooth shopping experience for users. By leveraging MongoDB, Express.js, React.js, and Node.js, this project showcases the power of JavaScript in both frontend and backend development.