# Full Stack Development with MERN

## 1. Introduction

- **Project Title:** GrocerGo: MERN Stack based Grocery Store Application
- **Team Members:**

| S.No | Name | NMID | Role |
|---|---|---|---|
| 1 | Shivani Suresh | 11A8EBB9579E75F9C2D46A29C937B0E3 | Backend |
| 2 | Shyamala R B | 1E352D1BBE931BE9376478F8D8893207 | Backend |
| 3 | Ranjini.S | B7794B70D35CE65BF334FDDC573F1590 | Frontend |
| 4 | Ranjana.S | 447799B371B066D810948D514AAD777E | Frontend |
| Final Year, Department of Computer Technology, MIT Campus (Code: 477), Anna University | | | |

## 2. Project Overview

- **Purpose:** GrocerGo is designed to provide a seamless and efficient e-commerce platform for online grocery shopping. Its primary goal is to deliver an intuitive interface for users and a robust backend system for managing products, orders, and users.
- **Features:**
  - ★ **User Registration and Login:** Secure user authentication using JWT.
  - ★ **Product Catalog:** Organized by categories with search and filter options for easy navigation.
  - ★ **Shopping Cart:** Add, update, and remove items in real-time with automatic price calculations.
  - ★ **Order Management:** Users can view their order history, track order status, and receive updates.

## 3. Architecture

- **Frontend:**
  - ★ Built using React for a modular, component-based architecture.
  - ★ Handles routing with React Router for seamless navigation between pages like the home page, product details, cart, and checkout.
  - ★ Uses state management libraries like Redux for handling global application state.
  - ★ Integrated with CSS Modules to ensure styling consistency and scalability.
- **Backend:**
  - ★ Built with Node.js and Express.js to expose RESTful APIs.
  - ★ Middleware implemented for authentication, authorization, and error handling.
  - ★ Service-oriented design with separation of concerns for controllers, services, and routes.
  - ★ APIs handle CRUD operations for products, users, and orders.

- **Database:**
  - ★ MongoDB is the database used to store all application data.
  - ★ Collections include users, products, orders, and sessions.
  - ★ Mongoose ORM facilitates schema validation and efficient data operations.
  - ★ Data is indexed where necessary for faster queries, especially for product search and user-specific data retrieval.

## 4. Setup Instructions

- **Prerequisites:**
  - ★ **Node.js:** Ensure v14 or higher is installed.
  - ★ **MongoDB:** Set up a local instance or connect to a MongoDB Atlas cloud database.
  - ★ **Git:** For version control to update project repository.
- **Installation:**
  1. **Clone the repository:**
     - git clone https://github.com/Shyami31/GrocerGo.git
  2. **Navigate to the project directory:**
     - cd GrocerGo
     - cd GrocerGo-main
  3. **Install dependencies for both frontend and backend:**
     - npm install
  4. **Configure environment variables in a .env file**
     a. CONNECTION_URL = <Your MongoDB Connection String>
     b. JWT_SECRET_KEY = <Your JSON Web Token>
     c. JWT_AUTH_TTL = <Authorization Time To Live in seconds - Typically 1 hour (3600s)>
     d. JWT_CHECKOUT_TTL = <Checkout Time To Live in seconds - Typically 15 minutes (900s)>
     e. STRIPE_PRIVATE_KEY = <Your Stripe Private API Key>
  5. **Database Initialization:** To seed the database with initial data
     - npm run seed
  6. **Run the application:**
     1) Open two terminals on the client and server side.
     2) Enter the following command on both terminals : npm start
     3) Run the server first, followed by the client.

## 5. Folder Structure

- **Client:**
  - ★ **/src/components:** Houses reusable UI components like Navbar, Footer, ProductCard, and CartSummary.
  - ★ **/src/pages:** Includes main application screens such as Home, ProductDetails, Cart, and Checkout.
  - ★ **/src/redux:** Redux slices for managing application state (e.g., user authentication, cart items).
  - ★ **/src/assets:** Contains static assets such as images, icons, and stylesheets

- **Server:**
  - ★ **routes/:** Defines API endpoints (e.g., /products, /users, /orders).
  - ★ **controllers/:** Contains logic for processing API requests and responses.
  - ★ **models/:** Mongoose schemas for MongoDB collections.
  - ★ **middleware/:** Includes authentication middleware for protected routes.
  - ★ **utils/:** Utility functions for token generation, hashing passwords, and error formatting.

## 6. Running the Application

- **Frontend:** Navigate to the client directory and run the following command to start the development server:
  - ★ npm start
- **Backend:** Navigate to the server directory and run the following command to start the server:
  - ★ npm start
- **Concurrent Development:**To run both servers simultaneously (requires a configured package.json)
  - ★ npm run dev

## 7. API Documentation

**Authentication Endpoints:**

- POST /auth/register
  Registers a new user.
  - ★ Request body: { "email": "user@example.com", "password": "password123" }
  - ★ Response: { "message": "User registered successfully" }
- POST /auth/login
  Authenticates a user and issues a JWT.
  - ★ Request body: { "email": "user@example.com", "password": "password123" }
  - ★ Response: { "token": "<jwt_token>" }

**Product Endpoints:**

- GET /products
  Retrieves all available products. Supports filtering by category.
  - ★ Query parameter: ?category=fruits
  - ★ Response: [ { "id": 1, "name": "Apple", "price": 2.5, "stock": 50 } ]
- POST /products (Admin only)
  Adds a new product to the catalog.
  - ★ Request body: { "name": "Mango", "price": 3.5, "stock": 100, "category": "fruits" }
  - ★ Response: { "message": "Product added successfully" }

**Order Endpoints:**

- POST /orders
  Places a new order.

    - ★ Request body: { "userId": 123, "items": [{ "productId": 1, "quantity": 2 }] }
    - ★ Response: { "orderId": 456, "status": "pending" }
- GET /orders/
  Fetches all orders placed by a specific user.
    - ★ Response: [ { "orderId": 456, "status": "completed" } ]

## 8. Authentication

Authentication is handled using **JSON Web Tokens (JWT)**:

- Tokens are generated during login and stored in the client's local storage or cookies.
- Protected routes validate the token in the authorization header.
- Passwords are hashed using **bcrypt** for secure storage.

## 9. User Interface

**Screens and Features:**

- **Home Page:** Displays featured products and promotions.
- **Product Details:** Offers comprehensive details about a product, including stock availability.
- **Shopping Cart:** Allows users to view and update items, with a real-time total price calculation.
- **Checkout:** Secure payment processing with integrated gateways.

## 10. Testing

**Testing Tools:**

- **Frontend:** React Testing Library for unit testing React components.
- **Backend:** Mocha and Chai for API testing.
- **API Testing:** Postman to validate all endpoints manually.

**Test Cases:**

- User authentication flow.
- Product search and filter functionality.
- Cart operations (add, update, remove).
- Order creation and retrieval.

## 11. Screenshots or Demo

Link to Demo:
https://drive.google.com/file/d/1pO2MV8O6H2ALr_V_vwAYDEaqQE318mKk/view?usp=sharing

## 12. Known Issues

- Product filtering could be optimized for better performance under high traffic.
- No support for real-time notifications (e.g., low stock alerts).

## 13. Future Enhancements

- Add user reviews and ratings for products.
- Real-time inventory updates using WebSockets.
- Multi-language and currency support for global users.