Full Stack Development with MERN - Grocery App Documentation

1. Introduction

Project Title: Grocery App

Team Members:

Ayesha Fathima A H – Backend Developer

Susi Azsaria W – Backend Developer

Gibson J – Frontend Developer

Varsha V – Frontend Developer

2. Project Overview

Purpose: The project is a grocery application that allows users to browse products, add them to their cart, place orders, and manage their user profile. Admins can manage products, orders, and reviews.

Features:

User authentication and profile management

Product search, filters, and product details

Shopping cart and wishlist management

Order placement and order history

Admin functionalities to add, update, delete products, and view orders

User reviews for products

3. Architecture

Frontend: The frontend is built using Angular, providing a responsive and user-friendly interface for interacting with the backend services. It consumes RESTful APIs exposed by the backend.

Backend: The backend is built using Node.js and Express.js, implementing RESTful services for product management, order management, user authentication, and reviews.

Database: MongoDB is used for storing data. The application schema includes collections for Users, Products, Orders, and Payments, with references between them using ObjectIds.

4. Setup Instructions

Prerequisites:

Node.js

MongoDB

Angular CLI (for frontend)

Postman (optional, for API testing)

Installation:

1. Clone the repository:

git clone https://github.com/AyeshaFathima2003/GroceryApp.git

2. Navigate to the server directory and install dependencies:

cd grocery-app-server
npm install

3. Set up environment variables in a .env file:

MONGO_URI=<your_mongo_database_uri>
JWT_SECRET_KEY=<your_jwt_secret_key>

4. Navigate to the client directory and install Angular dependencies:

cd grocery-app-interface
npm install

5. Folder Structure

Client:

src/app/components: Contains reusable Angular components (e.g., product list, cart).

src/app/services: Contains services for interacting with the backend API.

src/app/models: Contains models for data types like Product, User, and Order.

Server:

models/: Mongoose models for users, orders, products, etc.

routes/: Express routes for API endpoints

controllers/: Business logic for handling requests

middleware/: Token verification and other middleware

config/: Database connection and server configuration

6. Running the Application

Frontend: Navigate to the client directory and run:

ng serve

The frontend will be available at http://localhost:4200.

Backend: Navigate to the server directory and run:

npm start

The backend will be available at http://localhost:5000.

7. API Documentation

User Endpoints:

POST /api/user/signup: Register a new user

POST /api/user/signup
Registers a new user.

Request Body:

{ "name": "John", "email": "john@example.com", "password": "12345" }

Response:

{ "message": "User registered successfully" }

POST /api/user/login: Login a user

POST /api/user/login
Logs in a user and returns a JWT token.

Request Body:

{ "email": "john@example.com", "password": "12345" }

Response:

{ "token": "JWT_TOKEN" }

GET /api/user/profile: Get user profile details

GET /api/user/profile: Get user profile details

Request:

GET /api/user/profile
Authorization: Bearer <token>

Response:

```
{
  "status": "success",
  "data": {
    "name": "John Doe",
    "email": "john.doe@example.com",
    "phone": "123-456-7890",
    "addresses": [
      {
        "street": "123 Main St",
        "city": "New York",
        "state": "NY",
        "zip": "10001",
        "country": "USA"
      }
    ]
  }
}
```


POST /api/user/logout: Logout the user

POST /api/user/logout: Logout the user

Request:

```
POST /api/user/logout
Authorization: Bearer <token>
```

Response:


```
{
  "status": "success",
  "message": "User logged out successfully"
}
```


POST /api/user/address: Add user address

POST /api/user/address: Add user address

Request:

```
POST /api/user/address
Authorization: Bearer <token>
Content-Type: application/json
```

```
{
  "street": "456 Elm St",
  "city": "Los Angeles",
  "state": "CA",
  "zip": "90001",
  "country": "USA"
}
```


Response:


```
{
  "status": "success",
  "message": "Address added successfully"
}
```


GET /api/user/addresses: Get user addresses

GET /api/user/addresses: Get user addresses

Request:

GET /api/user/addresses
Authorization: Bearer <token>

Response:


```
{
  "status": "success",
  "data": [
    {
      "street": "123 Main St",
      "city": "New York",
      "state": "NY",
      "zip": "10001",
      "country": "USA"
    },
    {
      "street": "456 Elm St",
      "city": "Los Angeles",
      "state": "CA",
      "zip": "90001",
      "country": "USA"
    }
```

```
  ]
}
```

Order Endpoints:

POST /api/order/placeOrder: Place a new order

POST /api/order/placeOrder: Place a new order

Request:

POST /api/order/placeOrder
Authorization: Bearer <token>
Content-Type: application/json

```
{
  "items": [
    {
      "productId": "603c1f14f2b48e3d1c8b4567",
      "quantity": 2,
      "price": 15.99
    },
    {
      "productId": "603c1f14f2b48e3d1c8b4568",
      "quantity": 1,
      "price": 30.99
    }
  ],
  "totalAmount": 62.97,
  "paymentId": "603c1f14f2b48e3d1c8b4569"
}
```

Response:

```
{
  "status": "success",
  "message": "Order placed successfully",
  "orderId": "603c1f14f2b48e3d1c8b4570"
}
```

GET /api/order/userOrders: Get orders of the logged-in user

GET /api/order/userOrders: Get orders of the logged-in user

Request:

GET /api/order/userOrders
Authorization: Bearer <token>

Response:


```
{
  "status": "success",
  "data": [
    {
      "orderId": "603c1f14f2b48e3d1c8b4570",
      "status": "pending",
      "totalAmount": 62.97,
      "createdAt": "2024-11-10T14:30:00Z"
    },
    {
      "orderId": "603c1f14f2b48e3d1c8b4571",
      "status": "completed",
      "totalAmount": 45.50,
      "createdAt": "2024-11-05T10:15:00Z"
    }
  ]
}
```


GET /api/order/orderDetails: Get order details by order ID

GET /api/order/orderDetails: Get order details by order ID

Request:

GET /api/order/orderDetails?orderId=603c1f14f2b48e3d1c8b4570
Authorization: Bearer <token>

Response:


```
{
  "status": "success",
  "data": {
    "orderId": "603c1f14f2b48e3d1c8b4570",
    "items": [
      {
```

```
      "productId": "603c1f14f2b48e3d1c8b4567",
      "quantity": 2,
      "price": 15.99
    },
    {
      "productId": "603c1f14f2b48e3d1c8b4568",
      "quantity": 1,
      "price": 30.99
    }
  ],
  "totalAmount": 62.97,
  "status": "pending",
  "paymentId": "603c1f14f2b48e3d1c8b4569"
 }
}
```


Product Endpoints:

GET /api/product/allproducts: Get all products

GET /api/product/allproducts: Get all products

Request:

GET /api/product/allproducts

Response:


```
{
 "status": "success",
 "data": [
  {
   "productId": "603c1f14f2b48e3d1c8b4567",
   "name": "Grocery Item 1",
   "price": 15.99,
   "stock": 50,
   "category": "Fruits"
  },
  {
   "productId": "603c1f14f2b48e3d1c8b4568",
   "name": "Grocery Item 2",
   "price": 30.99,
   "stock": 20,
   "category": "Vegetables"
```

```
   }
 ]
}
```

GET /api/product/productid: Get a specific product by ID

POST /api/product/createproduct: Create a new product (admin only)

PUT /api/product/updateproduct: Update a product (admin only)

DELETE /api/product/deleteproduct: Delete a product (admin only)

GET /api/product/searchproduct: Search products by name or category

POST /api/product/addproductreview: Add a review for a product

GET /api/product/getproductreview: Get reviews for a product

GET /api/product/gettopratedproducts: Get top-rated products

8. Authentication

Authentication is handled using JWT (JSON Web Token). Upon successful login, a JWT token is generated and sent to the client. This token is used in subsequent API requests to verify the user's identity and grant access to protected resources.

Token Storage: Tokens are stored in the browser's local storage.

Authorization: Admin functionalities (e.g., adding, updating, deleting products) are protected by checking if the user's role is admin.

9. User Interface

The frontend provides a seamless user experience with the following key features:

Home page with product categories and search bar

Product details page with the option to add reviews and add to the cart

User profile management and order history

10. Testing

Testing is performed manually using Postman and browser tools to test the API endpoints and ensure the correct data is returned. Additionally, unit testing can be implemented with Jasmine/Karma for Angular components.

11. Screenshots or Demo

12. Known Issues

Some API endpoints might return null for certain invalid data (e.g., invalid product IDs).

There might be minor UI glitches on different screen sizes which are being resolved.

13. Future Enhancements

Implement user role management (e.g., admin, customer).

Add features like payment integration for placing orders.

Implement advanced product filters (e.g., by price range, rating).