

Project Report

Grocery App Documentation

1. Introduction

- **Project Title:** Grocery App
- **Team Members:**
 - Vishaal Maria Anto
 - Feby Yesudasan
 - Kinsmen
 - Rooban J

2. Project Overview

- **Purpose:**

The project is a grocery app that makes shopping easy and convenient. Users can browse products, add them to the cart, place orders, and manage their profiles. Admins can manage products, orders, and user accounts to ensure smooth operations.
- **Features:**
 - Easy product browsing and shopping.
 - Quick order placement and tracking.
 - Simple profile and inventory management.

3. Architecture

Frontend

- The frontend is built with React.js, providing a dynamic, responsive user interface. Key features include:
- 1. UI Structure: Components are organized into reusable units (e.g., headers, forms) for maintainability.
- 2. State Management: Global state is managed with React Context API or Redux, while local state is handled using useState.
- 3. Routing: React Router is used for smooth navigation between different pages.
- 4. API Integration: The frontend communicates with the backend via Axios or Fetch API to fetch and display data.
- 5. Responsive Design: The UI adapts to different screen sizes for a consistent experience across devices.

Backend

- The backend is built using Node.js and Express.js to provide a reliable and scalable server environment. It includes the following key functionalities:
- 1. API Routes: Handles user authentication, data management, and CRUD operations through structured endpoints.
- 2. Middleware: Implements authentication (via JWT), request validation, and error handling for consistent responses.
- 3. Database Integration: Uses MongoDB with Mongoose to interact with the database, defining schemas and managing data.
- 4. Authentication and Security: Provides secure login using JWT, encrypts passwords with bcrypt, and includes role-based access control.
- 5. Scalability: Follows a modular design, enabling easier updates and future feature expansions.

Database

- **MongoDB** was used to manage data.
- Database schema includes: [Detail key collections, fields, and relationships.]

4. Setup Instructions

Prerequisites

- Node.js (version X.X.X or higher)
- MongoDB (locally installed or cloud instance)
- [Other software dependencies]

Installation

Clone the repository:

bash

Copy code

```
git clone https://github.com/VishaalMariaAnto/SmartInternz-NM.git
```

1.

Navigate to the project directory:

bash

Copy code

```
cd SmartInternz-NM
```

2.

Install dependencies for the frontend:

bash

Copy code

```
cd client
```

```
npm install
```

3.

Install dependencies for the backend:

bash

Copy code

```
cd ../server
```

```
npm install
```

4.

5. Set up environment variables:

Add `.env` file in the `server` directory with the following keys:

makefile

Copy code

```
MONGO_URI=<Your MongoDB URI>
```

```
JWT_SECRET=<Your Secret Key>
```

```
PORT=<Backend Port>
```

5. Folder Structure

Client (Frontend)

- `/src/components`: [Description of key components]
- `/src/pages`: [Page structure and routing details]

Server (Backend)

- `/routes`: Contains API route definitions.
- `/models`: Database schema definitions.
- `/middleware`: Middleware functions for validation/authentication.

6. Running the Application

Frontend

Navigate to the `client` directory:

bash

Copy code

```
cd client
```

1.

Start the React development server:

bash

Copy code

```
npm start
```

2.

Backend

Navigate to the `server` directory:

bash

Copy code

```
cd server
```

1.

Start the backend server:

bash

Copy code

```
npm start
```

2.

7. API Documentation

Endpoint: `/api/resource`

- **Method:** GET
- **Description:** [Brief endpoint functionality]

Request Parameters:

bash

Copy code

```
{ id: <resource_id> }
```

-

Example Response:

json

Copy code

```
{  
  "id": "123",  
  "name": "Example Resource",  
  "details": "Additional details here."  
}
```

-

(Repeat for other endpoints.)

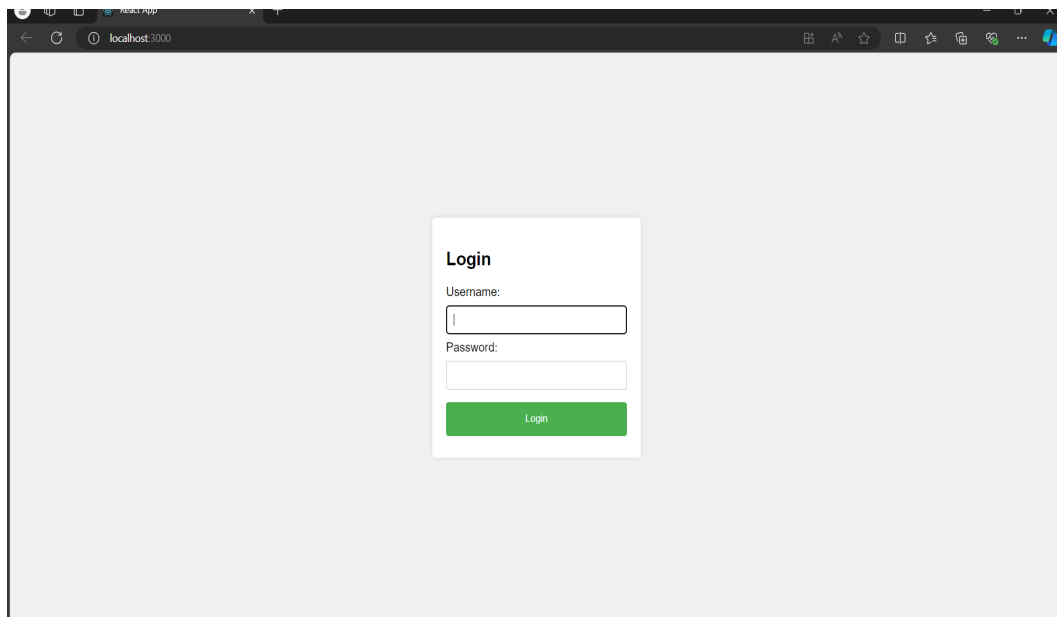
8. Authentication

- **Method:** JWT-based authentication.
- Access tokens are issued upon successful login and validated for every protected route.
- Refresh tokens/expiry details: [Details here.]

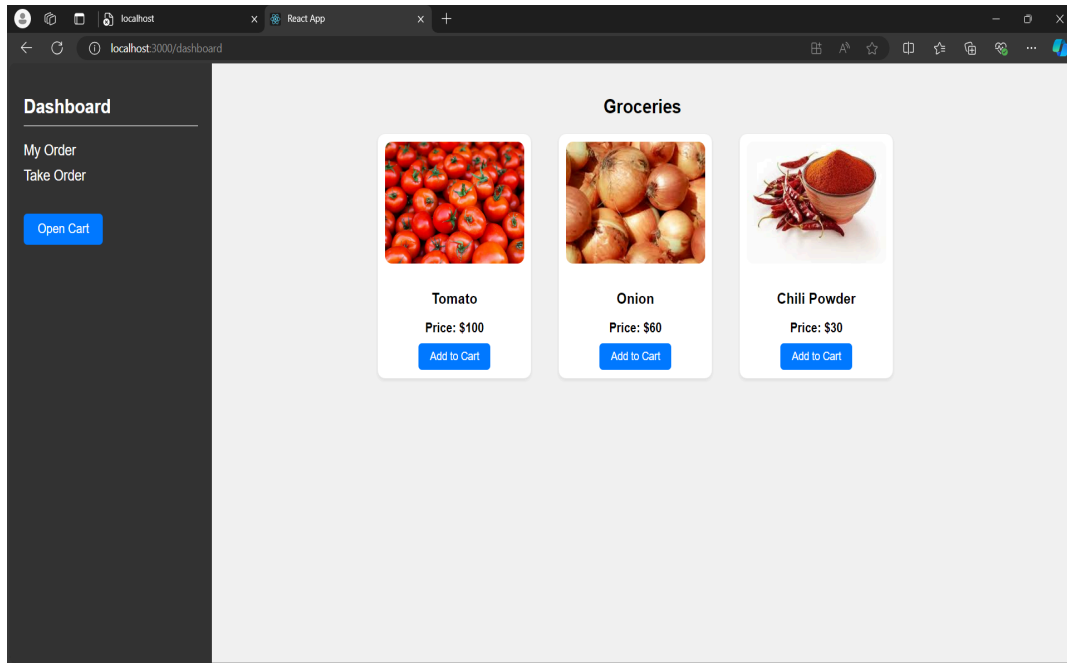
9. User Interface

(Screenshots/GIFs showcasing the UI.)

- **Login Page:** [Screenshot/description]

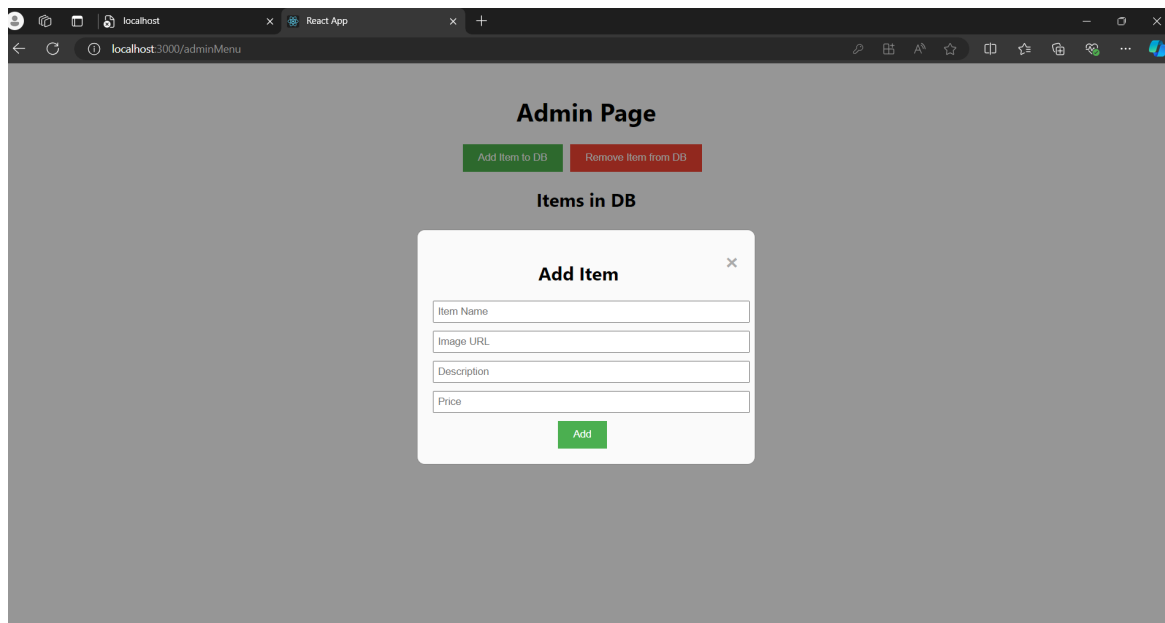


- **Dashboard:** [Screenshot/description]

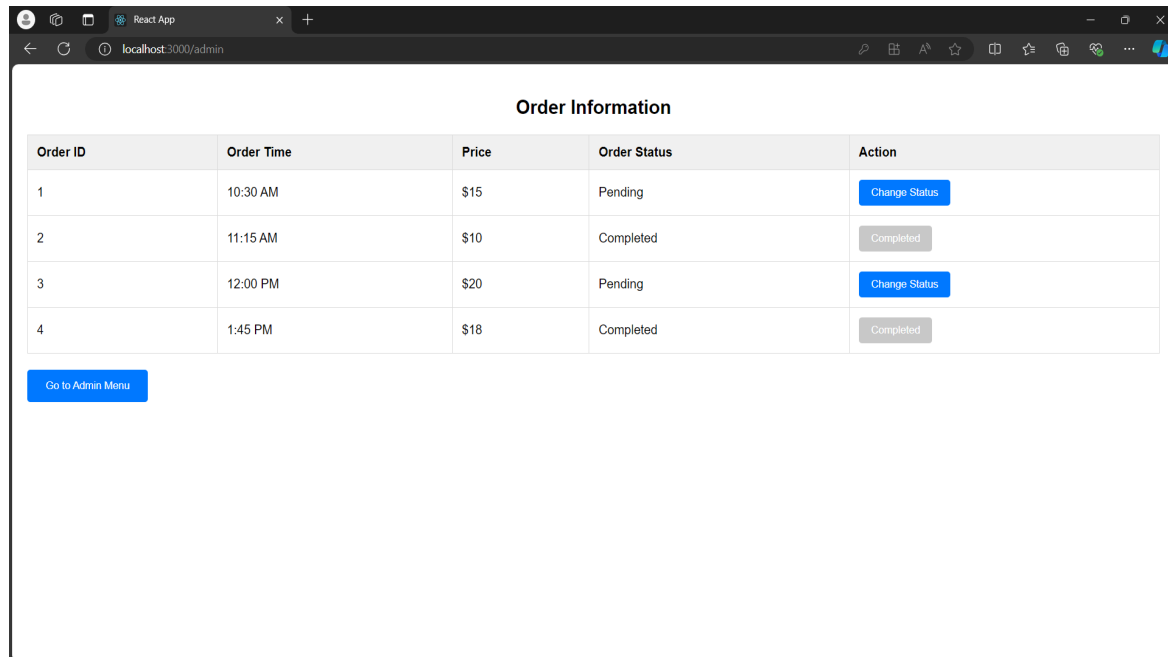


- **Other Features:** [Screenshot/description]

Admin Dashboard 1 :



Admin Dashboard 2 :

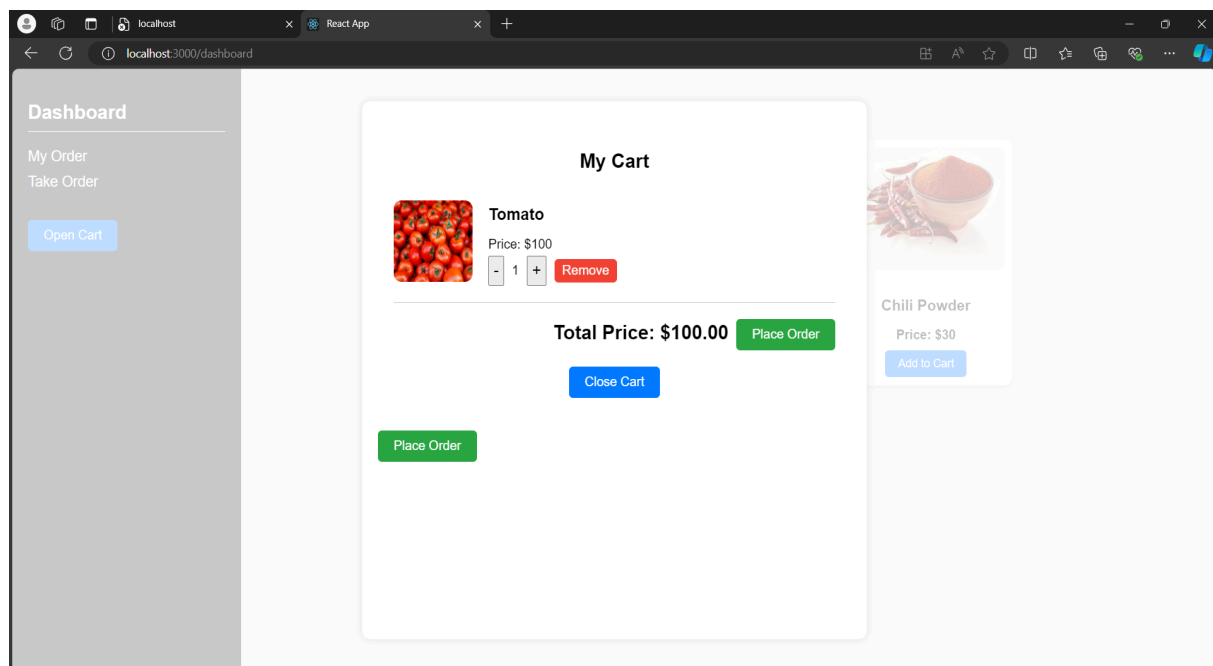


The screenshot shows a web browser window with the URL `localhost:3000/admin`. The page displays a table titled "Order Information" with the following data:

Order ID	Order Time	Price	Order Status	Action
1	10:30 AM	\$15	Pending	Change Status
2	11:15 AM	\$10	Completed	Completed
3	12:00 PM	\$20	Pending	Change Status
4	1:45 PM	\$18	Completed	Completed

Below the table, there is a button labeled "Go to Admin Menu".

Cart :



10. Testing

- **Strategy:** Unit testing with [tool], Integration testing with [tool].
- **Tools Used:** Jest, Mocha, Selenium.

- Example test cases: [Details here.]

1. User Registration

Test Case 1.1: Successful Registration

- **Description:** Verify that a new user can register with valid credentials.
- **Steps:**
 - Navigate to the registration page.
 - Enter a valid username, email, and password.
 - Submit the registration form.
- **Expected Result:**
 - The user is successfully registered.
 - A confirmation message is displayed.
 - The user is redirected to the login page.

Test Case 1.2: Registration with Existing Email

- **Description:** Verify that registration fails when using an email that's already registered.
 - **Steps:**
 - Navigate to the registration page.
 - Enter a username, an email already in use, and a password.
 - Submit the registration form.
 - **Expected Result:**
 - Registration fails.
 - An error message is displayed indicating that the email is already in use.
-

2. User Login

Test Case 2.1: Successful Login

- **Description:** Verify that a user can log in with correct credentials.
- **Steps:**
 - Navigate to the login page.
 - Enter a registered email and correct password.
 - Click the "Login" button.
- **Expected Result:**
 - The user is logged in successfully.
 - The user is redirected to the dashboard or home page.

Test Case 2.2: Login with Incorrect Password

- **Description:** Verify that login fails when an incorrect password is entered.
- **Steps:**
 - Navigate to the login page.
 - Enter a registered email and an incorrect password.
 - Click the "Login" button.
- **Expected Result:**

- Login fails.
 - An error message is displayed indicating incorrect credentials.
-

3. Product Browsing and Shopping

Test Case 3.1: Browse Products by Category

- **Description:** Verify that users can filter and browse products by category.
- **Steps:**
 - Log in as a user.
 - Navigate to the product browsing page.
 - Select a specific category from the filter options.
- **Expected Result:**
 - Only products belonging to the selected category are displayed.
 - The filter is applied correctly.

Test Case 3.2: Search for a Product

- **Description:** Verify that the search functionality returns relevant products.
 - **Steps:**
 - Log in as a user.
 - Use the search bar to enter a product name.
 - Submit the search.
 - **Expected Result:**
 - Products matching the search query are displayed.
 - Search results are relevant and accurate.
-

4. Cart Functionality

Test Case 4.1: Add Product to Cart

- **Description:** Verify that users can add products to the cart.
- **Steps:**
 - Log in as a user.
 - Browse products and select a product.
 - Click the "Add to Cart" button.
- **Expected Result:**
 - The selected product is added to the cart.
 - The cart count is updated accordingly.

Test Case 4.2: Remove Product from Cart

- **Description:** Verify that users can remove products from the cart.

- **Steps:**
 - Add a product to the cart.
 - Navigate to the cart page.
 - Click the "Remove" button for the selected product.
 - **Expected Result:**
 - The product is removed from the cart.
 - The cart updates to reflect the removal.
-

5. Order Placement

Test Case 5.1: Place an Order with Valid Cart

- **Description:** Verify that users can place an order with items in the cart.
- **Steps:**
 - Add products to the cart.
 - Proceed to checkout.
 - Enter valid delivery and payment details.
 - Click the "Place Order" button.
- **Expected Result:**
 - The order is placed successfully.
 - An order confirmation is displayed.
 - The order appears in the user's order history.

Test Case 5.2: Place an Order with Empty Cart

- **Description:** Verify that the system prevents order placement when the cart is empty.
 - **Steps:**
 - Ensure the cart is empty.
 - Attempt to proceed to checkout.
 - **Expected Result:**
 - The system prevents the order from being placed.
 - An error message is displayed indicating that the cart is empty.
-

6. Profile Management

Test Case 6.1: Update Profile Information

- **Description:** Verify that users can update their profile details.
- **Steps:**
 - Log in as a user.
 - Navigate to the profile page.
 - Edit profile information (e.g., name, address, contact number).
 - Save the changes.
- **Expected Result:**
 - Profile updates are saved successfully.

- The updated information is displayed correctly.

Test Case 6.2: Change Password

- **Description:** Verify that users can change their account password.
 - **Steps:**
 - Log in as a user.
 - Navigate to the "Change Password" section.
 - Enter the current password and a new password.
 - Confirm the new password and submit.
 - **Expected Result:**
 - Password is changed successfully.
 - The user can log in with the new password.
 - An appropriate confirmation message is displayed.
-

7. Admin - Manage Products

Test Case 7.1: Add New Product

- **Description:** Verify that the admin can add a new product to the inventory.
- **Steps:**
 - Log in as an admin.
 - Navigate to the product management page.
 - Click on "Add New Product".
 - Enter valid product details (e.g., name, price, stock, category).
 - Submit the form.
- **Expected Result:**
 - The new product is added to the inventory.
 - The product appears in the product list with correct details.

Test Case 7.2: Edit Existing Product

- **Description:** Verify that the admin can edit details of an existing product.
 - **Steps:**
 - Log in as an admin.
 - Navigate to the product management page.
 - Select a product to edit.
 - Modify the product details.
 - Save the changes.
 - **Expected Result:**
 - The product details are updated successfully.
 - The changes are reflected in the product list.
-

8. Admin - Manage Orders

Test Case 8.1: View All Orders

- **Description:** Verify that the admin can view all orders placed by users.
- **Steps:**
 - Log in as an admin.
 - Navigate to the order management page.
- **Expected Result:**
 - A comprehensive list of all orders is displayed.
 - Each order shows relevant details such as user information, products ordered, and order status.

Test Case 8.2: Update Order Status

- **Description:** Verify that the admin can update the status of an order.
- **Steps:**
 - Log in as an admin.
 - Navigate to the order management page.
 - Select an order to update.
 - Change the order status (e.g., from "Processing" to "Shipped").
 - Save the changes.
- **Expected Result:**
 - The order status is updated successfully.
 - The updated status is visible to both the admin and the user.

11. Demo

- **Live Demo:**

https://drive.google.com/file/d/1GxAawi-YSIJjnLpBX9iFdM7BcWIHEuDX/view?usp=drive_link

12. Known Issues

- **Issue 1:** Some components may not render perfectly on smaller screen sizes or devices with lower resolutions.
- **Issue 2:** Pages with complex queries or large datasets may experience slower load times.

13. Future Enhancements

- **Feature 1:** Enhance UI components to provide a seamless experience across all screen sizes and devices.
- **Feature 2:** Implement detailed analytics and reporting features to provide insights into user behavior and application performance.