GROCERY WEBAPP

1. Introduction

Project Title: Grocery Webapp

TEAM MEMBERS:

|  |  |  |  |
| --- | --- | --- | --- |
| S.NO | NAME | ROLE | RESPONSIBILITIES |
| 1. | Anitha D | Full-Stack  Developer | Responsible for overall development,  Including front-end, back-end, server-side logic, and database design |
| 2. | Ajith Kumar | Frontend  Developer | Responsible for UI/UX design using Angular, Material UI, and Bootstrap. |
| 3. | Aiswarya | Database  Administrator | Responsible for MongoDB setup and ensuring data integrity. |
| 4. | Aastin Raj | Backend Developer | Responsible for Express.js setup and API development. |

2.Project Overview

Purpose: A grocery web app allows users to browse, order, and manage grocery items online. Users can add items to a virtual cart, schedule delivery or pick-up, and make secure payments. The app will cater to individual customers and potentially offer features for local grocery stores to manage their inventory and reach online customers.

Features:

* ****User Authentication and Profiles:****
* Account creation, login, and profile management.
* Options to save favorite items, order history, and saved addresses.
* Product Browsing and Search:
* Search bar with filtering by category, brand, price, and more.
* Detailed product pages with images, descriptions, nutritional information, and prices.
* **Shopping Cart:**
* Ability to add/remove items, adjust quantities, and view cart contents.
* Display of cart total and estimated delivery fees.
* **Checkout and Payment:**
* Secure checkout process with options for delivery or pick-up.
* Payment integration for credit/debit cards, digital wallets, and coupons.
* Delivery Scheduling and Tracking:
* Select delivery windows based on availability.
* Real-time order tracking and notifications for updates on delivery status.
* **Inventory Management (for Store Admins):**

* Interface for store owners/admins to add, edit, or remove items.
* Stock and pricing management.
* **Order History and Reordering:**
* View previous orders and reorder items directly from past orders.

3.Architecure

Frontend:

* The frontend is built using a JavaScript framework such as React, Vue.js, or Angular, providing a single-page application (SPA) experience where users can browse products, add items to their cart, and proceed to checkout seamlessly.
* To manage state, libraries like Redux (for React) or Vuex (for Vue.js) keep data consistent across components, allowing for features such as real-time cart updates and personalized recommendations.
* The frontend uses Axios or the Fetch API to make HTTP requests to the backend, enabling interactions like searching for items, retrieving product details, and submitting orders.
* Progressive Web App (PWA) features, such as offline access and push notifications, can be implemented to enhance usability.

Backend:

* Node.js with Express or Django (Python) are common choices for backend frameworks due to their scalability and support for building robust APIs.
* For user authentication, JWT (JSON Web Tokens) are often used to maintain secure sessions, and OAuth integration can enable third-party login options (e.g., Google or Facebook).
* To handle orders, the backend integrates with payment gateways like Stripe or PayPal, managing transactions securely. WebSocket or Server-Sent Events (SSE) can be used to enable real-time updates for inventory levels, order status, or notifications.

Database:

* The application uses MongoDB, a NoSQL database, to store and manage the data. MongoDB is chosen for its flexibility and scalability.
* Mongoose is used to define the application's schemas and models. The key tables are:
* User Table: Stores user account information, such as personal details, email, hashed passwords, and addresses.
* Product Table: Contains product data, including names, descriptions, prices, categories, and inventory levels.
* Category Table: Groups products into various categories, like produce, dairy, or pantry items, and has a one-to-many relationship with the Product table.
* Order Table: Captures customer orders, with details like total amount, status, and delivery date.

.

4. Setup Instructions

Prerequisites:

* + Node.js: Make sure Node.js is installed. This is needed to run the application locally and install necessary dependencies.
  + MongoDB: Ensure that you have a running instance of MongoDB. You can use either a local MongoDB setup or a cloud-based solution like MongoDB Atlas.
  + npm: The Node Package Manager (npm) is required to install dependencies.

Installation:

1. Clone the Repository:

git clone [https://github.com/Bharath136/grocery-webapp](https://github.com/Bharath136/grocery-webapp%20)

1. Install Frontend Dependencies: Navigate to the client folder and run:

Cd client

npm install

1. Install Backend Dependencies: Navigate to the server folder and run:

cd server

npm install

1. Set up Environment Variables: In the server folder, create a .env file to store sensitive information such as:

MONGODB\_URI: The connection string for your MongoDB database.

JWT\_SECRET: A secret key used for signing JWT tokens.

PORT: Port on which the backend server will run (default is 5000).

1. Start the Backend Server: Run the following command in the server directory:

npm start

1. Start the Frontend Server: Run the following command in the client directory:

npm start

The application will be available at http://localhost:3000 (frontend) and http://localhost:5000 (backend).

5. Folder Structure

Client:

* public/: Static assets that do not change dynamically (like the HTML file and images used across the app).
  + src/: Core source code for the app
  + api/: Functions that interact with your backend API (e.g., fetch products, handle cart, etc.).
  + assets/: Static files like images, icons, and fonts.
  + components/: UI components that are reusable across pages (e.g., ProductCard, Navbar, Footer).
  + config/: Holds configuration files like API URLs, environment variables, or constants used across the app.
  + App.js: The main app component where routing and other top-level logic are managed.
  + index.js: The entry point for the React app, where you would render the App component into the DOM.
  + services/: Business logic that can be reused across multiple components (e.g., cartService, checkoutService).

Server:

* server/: Contains all the backend-specific logic.
* controllers/: Functions that handle HTTP requests and responses (e.g., handling adding/removing items from the cart, fetching products, etc.).
* models/: Data models that represent your database structure (e.g., Product, User, Order models, possibly using Mongoose for MongoDB).
* routes/: API routes that connect to controllers and define the app's endpoints (e.g., GET /products, POST /cart).
* middlewares/: Custom middleware for tasks like user authentication, logging, and error handling.
* config/: Configuration files (e.g., database connection setup, secret keys).
* server.js: Entry point for the server (setting up Express, middleware, routes, etc.).

6. Running the Application

Frontend: To start the frontend server, run the following command in the client folder:

npm start

Backend: To start the backend server, run the following command in the server folder:

npm start

7. API Documentation

1Description: Logs in a user with credentials

* Request Body:

{

"email": "user@example.com",

"password": "userpassword123"

}

* Response:

{

"token": "jwt\_token\_here",

"user": {

"id": "user\_id"

"email": "user@example.com",

"name": "John Doe"

}

}

2. Description: Fetch a list of all grocery products

* Response:

{

"page": 1,

"totalPages": 5,

"totalItems": 100,

"items": [

{

"id": "product1",

"name": "Apple",

"description": "Fresh red apples",

"price": 2.99,

"image": "/images/apple.jpg",

"category": "Fruits",

"stock": 150

},

{

"id": "product2",

"name": "Carrot",

"description": "Fresh organic carrots",

"price": 1.50,

"image": "/images/carrot.jpg",

"category": "Vegetables",

"stock": 200

}

]

}

3.Description: Create a new product in the catalog.

* Request Body:

{

"name": "Banana",

"description": "Fresh yellow bananas",

"price": 1.20,

"category": "Fruits",

"stock": 500,

"image": "/images/banana.jpg"

}

* Response:

{

"id": "product3",

"name": "Banana",

"description": "Fresh yellow bananas",

"price": 1.20,

"category": "Fruits",

"stock": 500,

"image": "/images/banana.jpg"

}

4. Description: Retrieve the user's shopping cart.

* Request Body:

{

"userId": "user\_id",

"items": [

{

"productId": "product1",

"name": "Apple",

"quantity": 3,

"price": 2.99,

"totalPrice": 8.97

},

{

"productId": "product2",

"name": "Carrot",

"quantity": 2,

"price": 1.50,

"totalPrice": 3.00

}

],

"totalAmount": 11.97

}

Description: Place an order using the items in the cart.

* Request Body

{

"address": "123 Main St, Cityville, NY 12345",

"paymentMethod": "credit\_card",

"items": [

{

"productId": "product1",

"quantity": 3,

"price": 2.99

},

{

"productId": "product2",

"quantity": 2,

"price": 1.50

}

]

}

Description: Retrieve user profile details.

* Response:

{

"userId": "user\_id",

"name": "John Doe",

"email": "user@example.com",

"address": "123 Main St, Cityville, NY 12345"

}

8. Authentication

JWT Authentication: Implementing authentication in a grocery web app is a key step for controlling access to user-specific features, such as managing a shopping cart, viewing past orders, or updating personal information. You can implement authentication in various ways, such as with traditional session-based authentication or modern token-based methods like JWT (JSON Web Tokens).

Authorization Middleware:

Authentication Middleware: The authentication middleware ensures that only users with valid credentials can access the app.

Steps:

* Token-Based Authentication: Use tokens (e.g., JWT) for user sessions. Upon login, the server generates a token and sends it to the user, who stores it (usually in cookies or local storage).
* Session Management:  For each request, validate the token or session ID to confirm that the user is authenticated.
* Authorization Middleware: Authorization middleware verifies that the user has permission to perform specific actions base on their role.

Steps:

* Role-Based Access Control: Define roles (e.g., customer, admin, staff) and assign permissions to each role.
* Resource-Based Access Control: Create access rules that determine which resources (e.g., products, orders) a role can access or modify

9. User Interface

Designing an effective user interface for a grocery web app involves creating a clean, user-friendly layout that allows customers to easily browse, search, and purchase products. key components for an intuitive grocery web app UI

Home Page

* Hero Section: Showcase a banner with promotional offers or seasonal products.
* Product Categories: Display popular categories like "Fruits & Vegetables," "Dairy," "Bakery," "Snacks," etc.
* Featured Products: Highlight best-sellers or trending items.
* Search Bar: Prominently place a search bar for quick access to specific products.
* User Navigation Options: Quick links to “Sign In,” “Register,” and “Cart.”

Product Listing Page

* Product Grid: Display products in a grid layout with each product showing its image, name, price, weight/quantity, and an "Add to Cart" button.
* Filters & Sorting Options: Allow users to filter by category, price range, brand, or dietary preferences (e.g., gluten-free, vegan). Sort by price, popularity, and new arrivals.
* Category Sidebar: A sidebar with expandable category options for quick browsing.
* Pagination or Infinite Scroll: Use pagination or infinite scroll to load products, keeping the page load time manageable.

Product Detail Page

* Product Image Gallery: Show multiple images of the product to help users see it from different angles.
* Product Info: Display product name, price, quantity options, detailed description, and nutritional information.
* Add to Cart/Buy Now: Prominent buttons for adding items to the cart or proceeding to purchase directly.
* Customer Reviews: Allow users to see and add reviews to inform their purchasing decisions.
* Related Products: Suggest similar or complementary items.

Shopping Cart Page

* Cart Summary: Show items added to the cart, including name, image, price, quantity, and subtotal.
* Update Options: Enable users to adjust quantities or remove items from the cart.
* Order Summary: Display a total cost summary, including any discounts, taxes, and delivery fees.
* Checkout Button: Include a clear call-to-action button to proceed to checkout.

Checkout Page

* Address Form: Collect delivery information with address fields (auto-suggestions for faster input can be helpful).
* Delivery Options: Provide options for delivery times or types (e.g., express delivery, standard delivery).
* Payment Options: Offer a variety of payment methods like credit/debit cards, digital wallets, and cash on delivery.
* Order Summary: Show the final order summary for review before payment.
* Confirmation: A "Place Order" button to complete the purchase.

User Profile Page

* Personal Information: Display user details with an option to edit profile information.
* Order History: List past orders with details like order date, items, total amount, and status.
* Address Book: Allow users to add, edit, or delete saved delivery addresses.
* Wishlist: Let users save items they’re interested in purchasing later

Admin Dashboard (For Staff and Admins)

* Product Management: Interface to add, edit, or delete products, set prices, and manage inventory levels.
* Order Management: View and manage orders, including order status updates.
* User Management: Access user profiles to manage users or handle customer inquiries.
* Analytics Dashboard: Display charts or graphs on sales trends, popular products, and user demographics.

10. Testing

Testing Strategy:

Unit Testing:

* + - Objective: Validate individual functions or components in isolation.
    - Tools: Jest (JavaScript), Mocha, JUnit (Java)..

Integration Testing:

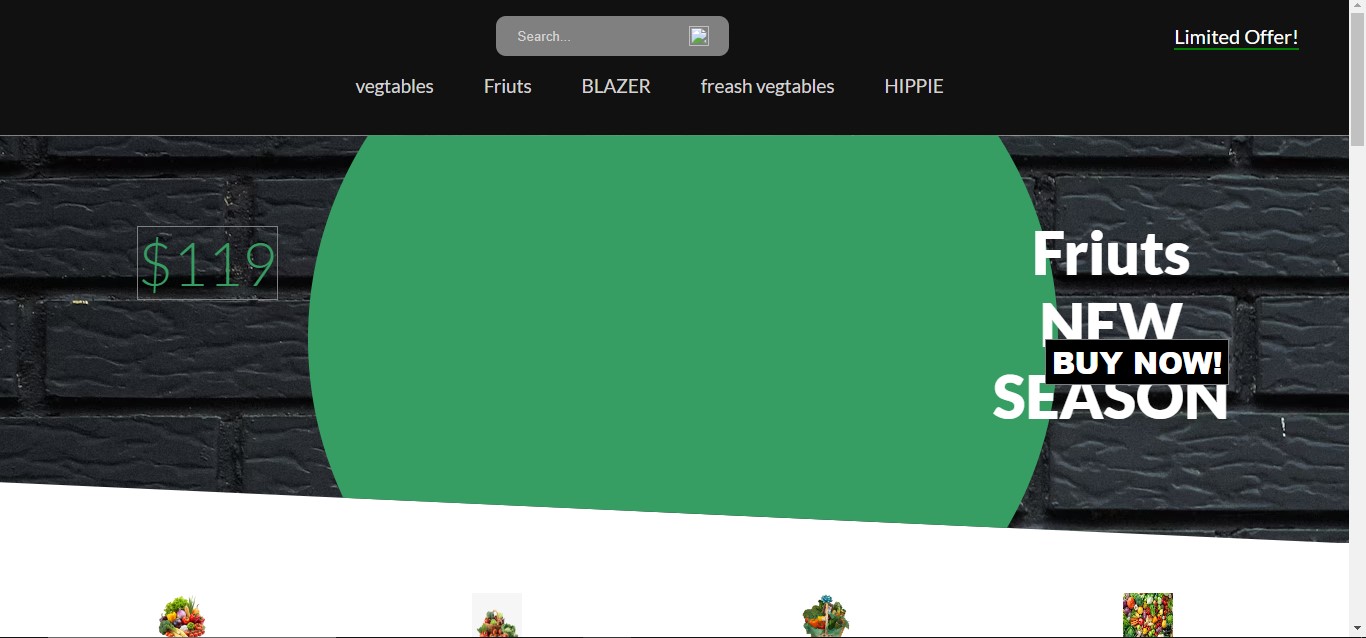
* Objective: Check that components work together correctly
* Tools: Postman (for API testing), Supertest, or integration frameworks in Jest or Mocha..

End-to-End Testing:

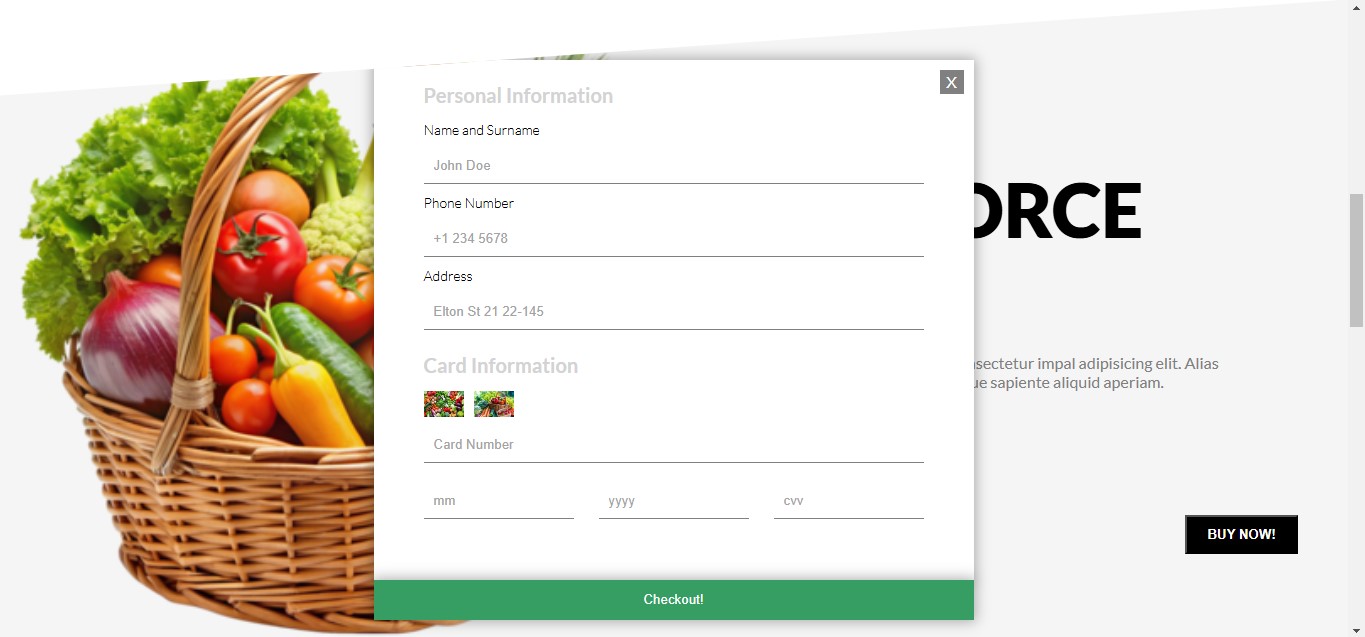
* Objective: Simulate user behavior from start to finish in the app.
* Tools: Cypress, Selenium, Puppeteer..

11. Screenshots

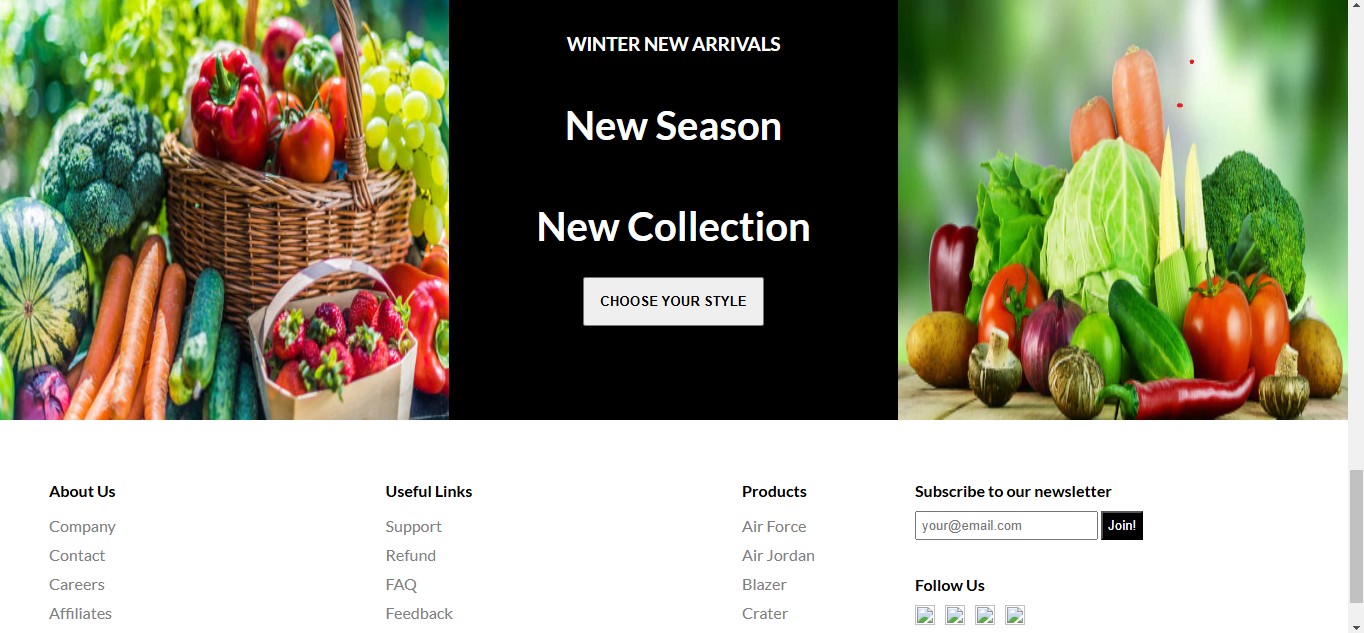
Landing page



Register page



Home page



12. Known Issues

While most of the application is fully functional, there are a few issues that developers or users should be aware of:

Issue with order Confirmation:

* Occasionally, the order confirmation status may take a few moments to update due to delays in backend processing.

Responsive Layout on Mobile:

* Some UI components may not fully adjust for extremely small screen sizes (e.g., older smartphones).

File Upload Issue:

* The file upload feature for appointment documents may fail intermittently with larger files (greater than 10MB).

Workarounds:

* For order confirmation delays, refreshing the page can help load the updated status.
* Mobile users can zoom out to view the full content if it's cut off on smaller screens.
* For file uploads, users are encouraged to compress files before uploading.

13. Future Enhancements

There are several potential improvements and features that could be added to the Grocery Web App with MERN application:

1. Personalized Shopping Experience

* AI-Driven Recommendations: Use AI to recommend products based on purchase history, dietary preferences, or browsing behavior.
* Dynamic Pricing and Discounts: Personalized offers and discounts based on loyalty, shopping frequency, and cart contents.

2. Improved Search and Navigation

* Voice and Visual Search: Integrate voice search for hands-free shopping or image recognition to find items by snapping a picture.
* Enhanced Filtering Options: Filter products by brand, dietary restrictions, eco-friendliness, or other preferences.

3. Enhanced Delivery Options

* Real-Time Delivery Tracking: Users can track their order in real time and get accurate delivery ETAs.
* Scheduled and Express Delivery: Options to schedule delivery times or opt for same-day/express delivery for select items.

4. Augmented Reality (AR) Integration

* In-Store Navigation: Help users find products more easily in physical stores with an AR-based navigation map.
* Product Visualizations: AR to preview products, read nutritional info, or view recipes that incorporate items in their cart.

5. Seamless Payments and Checkout

* One-Click Checkout: Fast and convenient checkout with payment and delivery details pre-filled.
* Multiple Payment Options: Integration with new payment methods like digital wallets, cryptocurrency, or buy-now-pay-later (BNPL) options.