

Full Stack Development with MERN

1.Introduction:

Project title: Grocery Web App

Team ID: NM2024TMID11807

Team Members:

Kiruthiga E – Project Lead & Frontend Developer

Led the project from initiation to completion, managing timelines, task allocation, and ensuring seamless communication. Additionally, designed and developed the frontend interface using AngularJS to ensure a dynamic and responsive user experience.

Komala B – Frontend Developer

Worked closely with the project lead on the AngularJS frontend, building interactive features such as product display, shopping cart, and checkout system. Optimized the UI for performance, usability, and responsiveness across different devices.

Manjushree V – UI/UX Designer

Led the design process, focusing on creating a user-friendly interface and ensuring the app's design was intuitive and visually appealing. Worked on wireframes, prototypes, and high-fidelity designs to guide the frontend development, ensuring the overall user experience was seamless and engaging.

Anish Fathima S – Backend Developer

Developed the backend using Node.js and Express, creating robust APIs for user authentication, product management, and order processing. Integrated the MongoDB database to handle dynamic data retrieval and optimized the server for scalability.

Karthiga P – Database Administrator & Quality Assurance

Managed MongoDB database setup and configuration, designed schemas for products, users, and orders, and ensured the database was efficient and scalable. Also performed testing across both frontend and backend components, identifying bugs and ensuring the app's performance and user experience met the required standards.

2. Project description:

Purpose:

The Grocery Web App built using the MERN stack (MongoDB, Express, React, Node.js) aims to provide a seamless online shopping experience for groceries by allowing users to browse, select, and purchase items easily. It offers personalized features like wishlists, order history, and product recommendations, while ensuring fast and scalable performance through Node.js, Express, and MongoDB. The app integrates real-time updates, secure payment gateways, and order tracking for a reliable shopping experience. An admin panel allows easy management of products, orders, and users, making the app both efficient and user-friendly, meeting the needs of modern grocery shoppers.

Features:

1. User Authentication:

The app will allow users to sign up and log in using their email/password or social media accounts such as Google or Facebook. Users can also reset their password if they forget it, ensuring smooth access to their accounts.

2. Product Catalog:

The product catalog will display items in different categories like fruits, vegetables, and dairy. Users can search for specific products, apply filters, and sort items based on price, popularity, or ratings for a streamlined shopping experience.

3. Shopping Cart:

Users can easily add products to their cart, view the total, and adjust item quantities or remove products. A cart summary will display the subtotal, taxes, and estimated shipping costs, and users can apply discount codes during checkout.

4. Checkout Process:

The checkout process will collect user shipping details and offer secure payment options like PayPal or Stripe. After payment, users will receive an order confirmation along with the estimated delivery date.

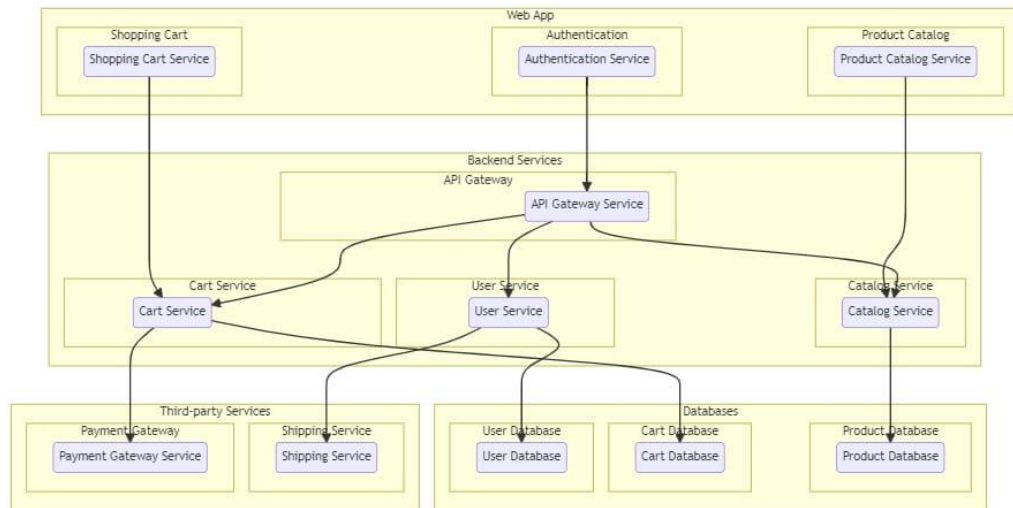
5. Order History & Tracking:

Users can view their order history and track current orders in real-time. They will be able to see the status of each order (pending, shipped, delivered), and get timely updates about their order progress.

6. Admin Panel

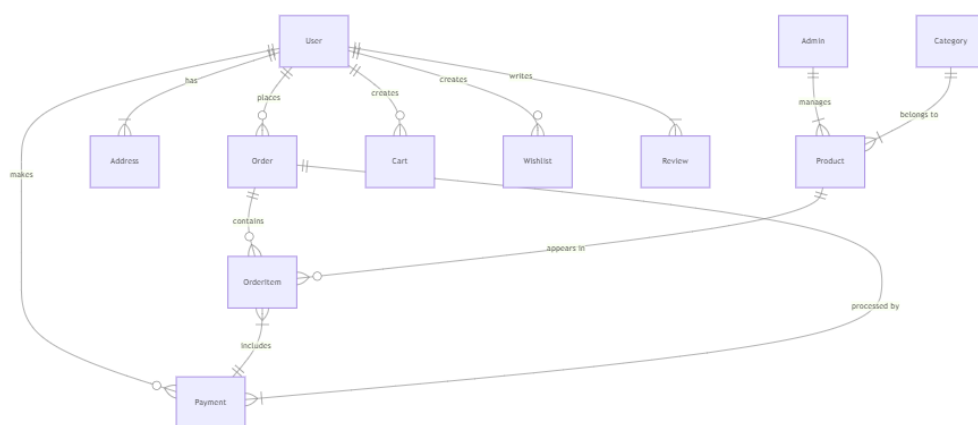
The admin panel allows administrators to manage products, view and update customer orders, and oversee user accounts. This helps in maintaining smooth operations, ensuring up-to-date product listings and timely order processing.

3.Architecture:



The technical architecture of an flower and gift delivery app typically involves a client-server model, where the frontend represents the client and the backend serves as the server. The frontend is responsible for user interface, interaction, and presentation, while the backend handles data storage, business logic, and integration with external services like payment gateways and databases. Communication between the frontend and backend is typically facilitated through APIs, enabling seamless data exchange and functionality.

ER Diagram



The Entity-Relationship (ER) diagram for an flower and gift delivery app visually represents the relationships between different entities involved in the system, such as users, products, orders, and reviews. It illustrates how these entities are related to each other and helps in understanding the overall database structure and data flow within the application.

4. Project flow:

Frontend development:

The frontend will focus on creating a visually appealing and user-friendly interface using modern design principles. Tools like Adobe XD, Sketch, Figma, or InVision will be used to create wireframes and mockups, ensuring a consistent design with attention to typography, color schemes, and spacing. The app will be fully responsive, utilizing CSS media queries and frameworks like Bootstrap or Tailwind CSS to ensure it looks great on all devices.

The product catalog will feature a listing page displaying product images, titles, descriptions, and prices, with search functionality and filters for easy browsing. A shopping cart will allow users to add items, update quantities, and remove products. The checkout process will include steps for shipping, payment selection, and order review.

User authentication will include a registration and login system, with profile pages for users to manage personal info, addresses, payment methods, and order history. Payment integration will use services like Stripe or PayPal to provide a secure and smooth transaction process, including handling success and failure scenarios.

Backend development:

The backend setup for the Grocery Web App begins by organizing the project with npm init and installing dependencies like Express.js. MongoDB is configured locally or via MongoDB Atlas, with collections created for products, users, and orders. An Express.js server handles HTTP requests, using middleware like body-parser and cors, and defines routes for functionalities like user management and order processing.

API routes are developed to handle user authentication, product catalog, shopping cart, and order placement, ensuring security and following best practices. Payment gateway integrations like Stripe or PayPal are added for secure transactions.

Shipping and logistics integrations are implemented for calculating costs, generating labels, and tracking deliveries. External services such as email and analytics are integrated. Security is ensured through encryption, access control, and input validation, while logging and error handling mechanisms are set up for smooth operation.

Database:

MongoDB is the database used for our grocery web application, selected for its flexible document-based structure and scalability.

MongoDB effectively manages various types of data, such as product information, user profiles, order histories, and shopping carts, supporting the dynamic needs of a grocery platform.

With its ability to adapt to different data models, MongoDB handles complex grocery-specific data, including item categories, inventory status, and seasonal promotions. The database also ensures fast retrieval of critical data, enabling smooth browsing, efficient order processing, and a seamless shopping experience for users.

5. Project structure:

Pre-requisites:

To develop a full-stack Grocery web app using AngularJS, Node.js, and MongoDB, there are several prerequisites you should consider. Here are the key prerequisites for developing such an application:

Node.js and npm: Install Node.js, which includes npm (Node Package Manager), on your development machine. Node.js is required to run JavaScript on the server side.

- Download: <https://nodejs.org/en/download/>
- Installation instructions: <https://nodejs.org/en/download/package-manager/>

MongoDB: Set up a MongoDB database to store hotel and booking information. Install MongoDB locally or use a cloud-based MongoDB service.

- Download: <https://www.mongodb.com/try/download/community>
- Installation instructions: <https://docs.mongodb.com/manual/installation/>

Express.js: Express.js is a web application framework for Node.js. Install Express.js to handle server-side routing, middleware, and API development.

- Installation: Open your command prompt or terminal and run the following command:

`npm install express`

Angular: Angular is a JavaScript framework for building client-side applications. Install Angular CLI (Command Line Interface) globally to create and manage your Angular project.

Install Angular CLI:

- Angular provides a command-line interface (CLI) tool that helps with project setup and development.
- Install the Angular CLI globally by running the following command:

`npm install -g @angular/cli`

Verify the Angular CLI installation:

- Run the following command to verify that the Angular CLI is installed correctly:

`ng version`

You should see the version of the Angular CLI printed in the terminal if the installation was successful.

Create a new Angular project:

- Choose or create a directory where you want to set up your Angular project.
- Open your terminal or command prompt.
- Navigate to the selected directory using the `cd` command.
- Create a new Angular project by running the following command:

`ng new client`

Wait for the project to be created

- The Angular CLI will generate the basic project structure and install the necessary dependencies
- **Navigate into the project directory:**
- After the project creation is complete, navigate into the project directory by running the following command:

`cd client`

Start the development server:

- To launch the development server and see your Angular app in the browser, run the following command:

`ng serve / npm start`

- The Angular CLI will compile your app and start the development server.
- Open your web browser and navigate to <http://localhost:4200> to see your Angular app running.

You have successfully set up Angular on your machine and created a new Angular project. You can now start building your app by modifying the generated project files in the `src` directory.

Please note that these instructions provide a basic setup for Angular. You can explore more advanced configurations and features by referring to the official Angular documentation:

<https://angular.io>

HTML, CSS, and JavaScript: Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.

Database Connectivity: Use a MongoDB driver or an Object-Document Mapping (ODM) library like Mongoose to connect your Node.js server with the MongoDB database and perform CRUD (Create, Read, Update, Delete) operations.

Front-end Framework: Utilize Angular to build the user-facing part of the application, including products listings, booking forms, and user interfaces for the admin dashboard.

Version Control: Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

- Git: Download and installation instructions can be found at: <https://git-scm.com/downloads>

Development Environment: Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.

- **Visual Studio Code:** Download from <https://code.visualstudio.com/download>
- **Sublime Text:** Download from <https://www.sublimetext.com/download>
- **WebStorm:** Download from <https://www.jetbrains.com/webstorm/download>

To Connect the Database with Node JS go through the below provided link:

- Link: <https://www.section.io/engineering-education/nodejs-mongoosejs-mongodb/>

Git Repository Cloning

To run the existing grocery-web app project downloaded from github:

Follow below steps:

1. Clone the Repository:

- Open your terminal or command prompt.
- Navigate to the directory where you want to store the grocery-webapp app.
- Execute the following command to clone the repository:

`git clone https://github.com/your-username/grocery-web app`

2. Install Dependencies:

- Navigate into the cloned repository directory:

`cd grocery-webapp`

- Install the required dependencies by running the following command:

`npm install`

3. Start the Development Server:

- To start the development server, execute the following command:

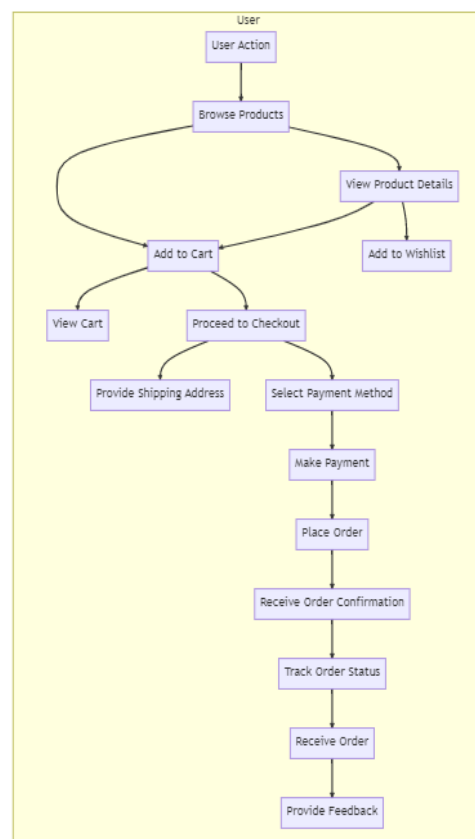
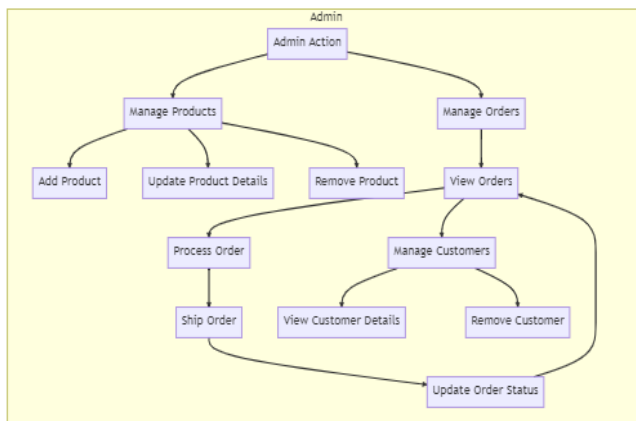
`npm run dev or npm run start`

- The e-commerce app will be accessible at <http://localhost:5100> by default. You can change the port configuration in the .env file if needed.

4. Access the App:

- Open your web browser and navigate to <http://localhost:5100>.
- You should see the grocery-webapp app's homepage, indicating that the installation and setup were successful.

6.Role based access:



1-Admin Role:

- Responsibilities:** The admin role has full control and administrative privileges over the system.

- **Permissions:**

- Manage: Admins can add, edit, and delete shop information along with products.
- Manage product bookings: Admins can view and manage all product bookings made by users and agents, including cancelling or modifying product bookings.
- Manage users: Admins can create, edit, and delete user accounts, as well as manage their roles and permissions.
- Generate reports: Admins have access to generate reports and analytics on product booking details, booking counts, and sales reports.

2-User Role:

- **Responsibilities:** Users are the customers of the online shopping web application who can search for products, and make product bookings.

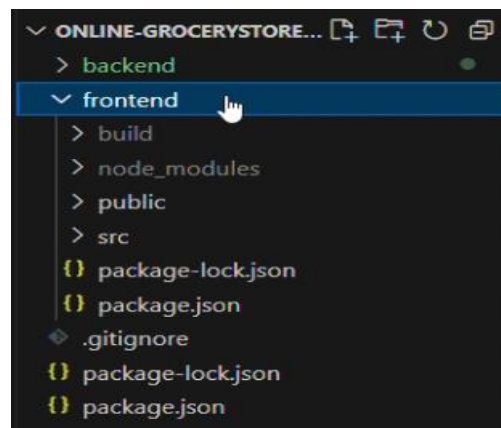
- **Permissions:**

- View products: Users can search for products, based on interest.
- Product bookings: Users can select products that are available and complete the order process.
- Manage product booking: Users can view their own product order bookings, modify booking details, track booking details, and cancel their bookings
- Manage cart: Users can view their cart details and modify them if needed.

7.Folder structure:

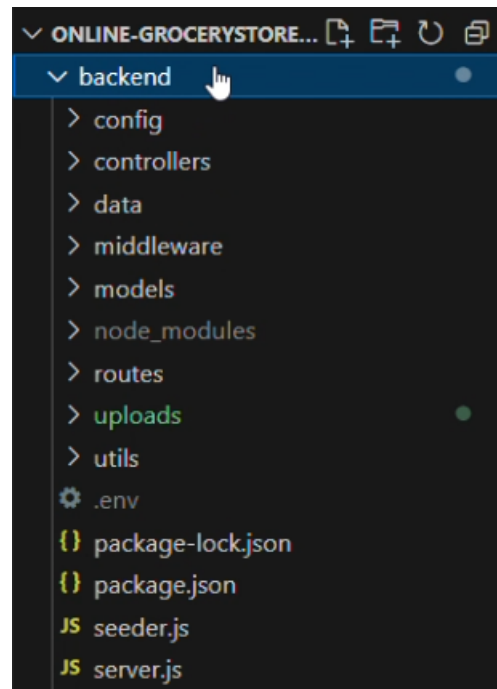
Client:

The client folder contains the React application with components, pages, hooks, and assets like images and styles. It includes configuration files such as package.json and webpack.config.js, as well as frontend-related logic for routing, state management, and UI rendering.



Server:

The server folder holds the backend logic, including Express.js routes, controllers, and models. It contains files for database configuration, authentication mechanisms, and environment variables, as well as the server setup configuration.



8. Running the application:

1. Running the Frontend:

The frontend of the Grocery Web App is built with AngularJS and handles the user interface and all client-side operations. This includes displaying product listings, managing shopping cart interactions, and handling API requests to the backend for data retrieval and updates. Follow these steps to run the frontend:

Step 1: Navigate to the Client Directory

- Open your terminal or command prompt.
- Navigate to the directory where the frontend code is located. This directory is typically named `client`.

`cd client`

Step 2: Install Dependencies

- Before starting the application, ensure that all dependencies are installed. Run the following command to install any missing packages specified in `package.json`:

`npm install`

Step 3: Start the Frontend Server

- After installing the dependencies, start the AngularJS development server with the following command:

`npm start`

- This command will launch the application on <http://localhost:4200> by default. You should see the home page of the Grocery Web App in your browser if everything is set up correctly.
- The frontend server supports live reloading, so any changes made to the code will automatically update in the browser.

Step 4: Check Frontend Components

- Verify that key components, such as the home page, product listing page, shopping cart, and user profile page, are rendering correctly.
- Ensure that the navigation system works smoothly, allowing users to access different sections of the app, and confirm that actions like adding items to the cart are functional.

2. Running the Backend:

The backend of the Grocery Web App is powered by Express.js, handling server-side logic, data processing, and interactions with the MongoDB database. It provides APIs that the frontend can call for operations like product retrieval, user authentication, and order management. To run the backend, follow these steps:

Step 1: Navigate to the Server Directory

- Open a new terminal window or tab.
- Move to the directory where the backend code is located, typically named `server`.

`cd server`

Step 2: Install Backend Dependencies

- Like the frontend, the backend requires certain packages to function correctly.
- Use the following command to install all required dependencies as listed in `package.json`:

`npm install`

Step 3: Configure Environment Variables

- Ensure that environment variables, such as the database URI and authentication keys, are correctly configured in a `.env` file within the backend directory. This file should define:
 - `DB_URI`: MongoDB connection string.
 - `JWT_SECRET`: Secret key for JWT authentication.
 - `PORT`: Port on which the backend server will run (default is 5000).

Step 4: Start the Backend Server

- After all configurations are in place, start the backend server by running:

`npm start`

- This command will initiate the server on <http://localhost:5000> (or the specified port in .env).
- The backend server will listen for API requests from the frontend and respond with data or process updates, depending on the request type.
-

Step 5: Verify API Endpoints

- Once the backend server is running, verify key API endpoints, such as `/api/products`, `/api/users`, and `/api/orders`, to ensure they are accessible and working correctly. These endpoints are crucial for functionalities like product listing, user management, and order handling.

3. Testing the Full Application

With both the frontend and backend running, open your browser and navigate to <http://localhost:4200> to access the Grocery Web App. Test the following core functionalities to ensure everything is functioning as expected:

User Authentication:

- Confirm that user registration, login, and profile management work correctly.
- Ensure that only authorized users can access certain sections of the app, such as the user profile and order history.

Product Interaction:

- Verify that product listings load from the backend to the frontend.
- Test the shopping cart functionality by adding, updating, and removing items. Ensure that the cart updates accurately with each action.

Order Processing:

- Complete the checkout process to verify that orders are submitted correctly to the backend.
- Confirm that order details, including shipping and payment information, are processed accurately.

9.API Documentation:

1. Authentication API

User Registration

Endpoint: </api/auth/register>

Method: `POST`

Description: Allows a new user to register by providing their username, email, and password. A successful registration returns a confirmation message.

User Login

Endpoint: </api/auth/login>

Method: **POST**

Description: Allows a registered user to log in by providing their email and password. Returns a JWT token for authentication.

2. Product API

Get All Products

Endpoint: </api/products>

Method: **GET**

Description: Retrieves a list of all available products in the grocery store, including product names, descriptions, prices, and images.

Get Single Product

Endpoint: </api/products/:id>

Method: **GET**

Description: Fetches detailed information for a specific product based on its unique ID.

3. Shopping Cart API

Get Cart Items

Endpoint: </api/cart>

Method: **GET**

Description: Retrieves all items currently in the user's shopping cart.

Add Item to Cart

Endpoint: </api/cart>

Method: **POST**

Description: Adds a product to the user's shopping cart, specifying the product ID and quantity.

Update Cart Item

Endpoint: </api/cart/:id>

Method: **PUT**

Description: Updates the quantity of a specific product in the shopping cart.

Remove Item from Cart

Endpoint: </api/cart/:id>

Method: **DELETE**

Description: Removes a specific product from the shopping cart.

4. Order API

Place Order

Endpoint: </api/orders>

Method: **POST**

Description: Allows the user to place an order by providing shipping address, payment method, and order items from the cart.

Get Order Details

Endpoint: </api/orders/:id>

Method: **GET**

Description: Retrieves details of a specific order, including shipping address, payment method, and order status.

5. Admin API

Add Product

Endpoint: </api/admin/products>

Method: **POST**

Description: Allows an admin user to add a new product to the store.

Update Product

Endpoint: </api/admin/products/:id>

Method: **PUT**

Description: Allows an admin user to update an existing product's details.

Delete Product

Endpoint: </api/admin/products/:id>

Method: **DELETE**

Description: Allows an admin user to delete a product from the store.

6. Error Handling

The API will respond with appropriate error messages in case of invalid requests or issues. Common response codes include:

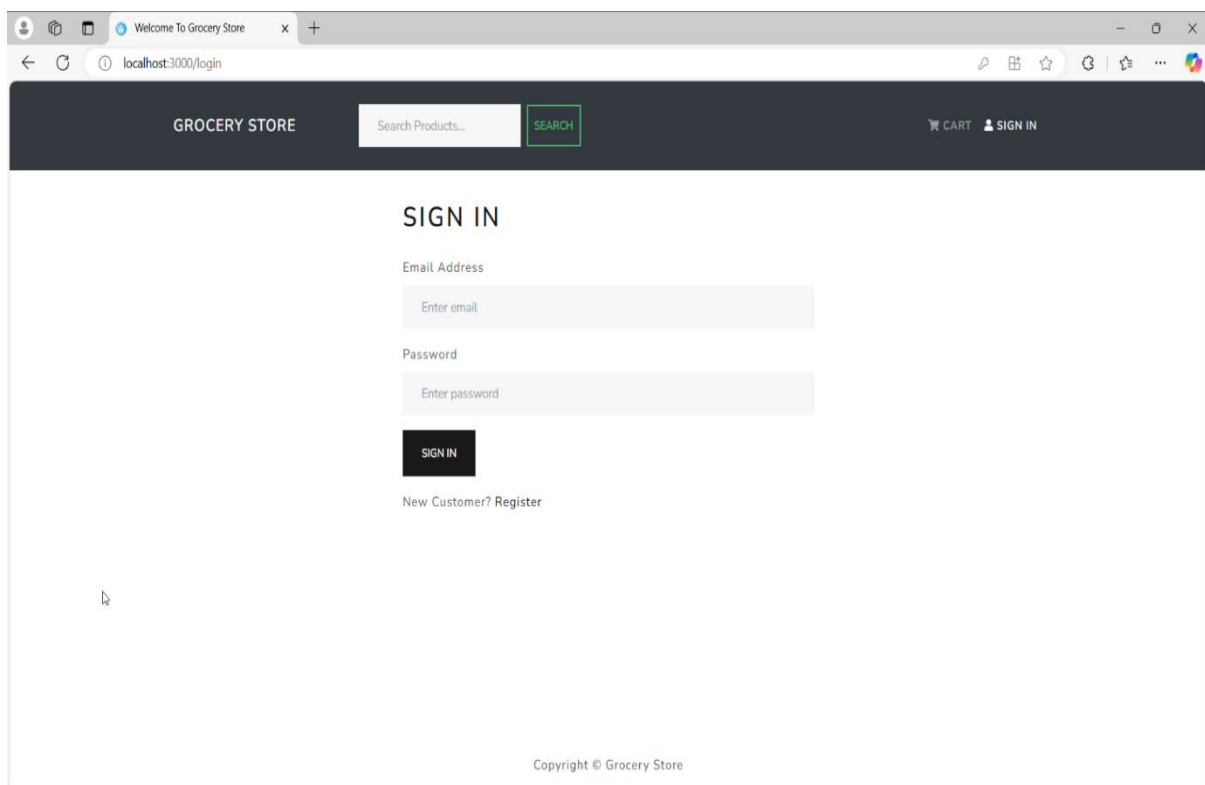
- 400 Bad Request: Invalid or missing parameters.
- 401 Unauthorized: User is not authenticated.
- 403 Forbidden: Insufficient permissions to perform the requested action.
- 404 Not Found: Resource not found.
- 500 Internal Server Error: General server error.

This API documentation ensures the Grocery Web Application functions efficiently, allowing both customers and administrators to interact with the system effectively.

10.Screenshots:

Sign in:

The secure entry point for registered users, allowing them to access their account with email and password credentials. It may also offer social media login options and a "Forgot Password" feature for easier access.



The screenshot displays a web browser window with the title "Welcome To Grocery Store". The address bar shows "localhost:3000/login". The page features a dark header with the text "GROCERY STORE", a search bar with the placeholder "Search Products..." and a green "SEARCH" button, and links for "CART" and "SIGN IN". The main content area is titled "SIGN IN" and contains two input fields: "Email Address" with the placeholder "Enter email" and "Password" with the placeholder "Enter password". Below these fields is a black "SIGN IN" button. At the bottom of the form, there is a link that reads "New Customer? Register". The footer of the page states "Copyright © Grocery Store".

Signup Page:

A user registration page where new users can create an account by providing basic information such as name, email, password, and possibly phone number. Clear instructions and validation messages ensure the signup process is smooth and secure.

Browser: Welcome To Grocery Store | localhost:3000/register/redirect=/
GROCERY STORE | Search Products... | SEARCH | CART | SIGN IN

SIGN UP

Name
Enter name

Email Address
Enter email

Password
Enter password

Confirm Password
Confirm password

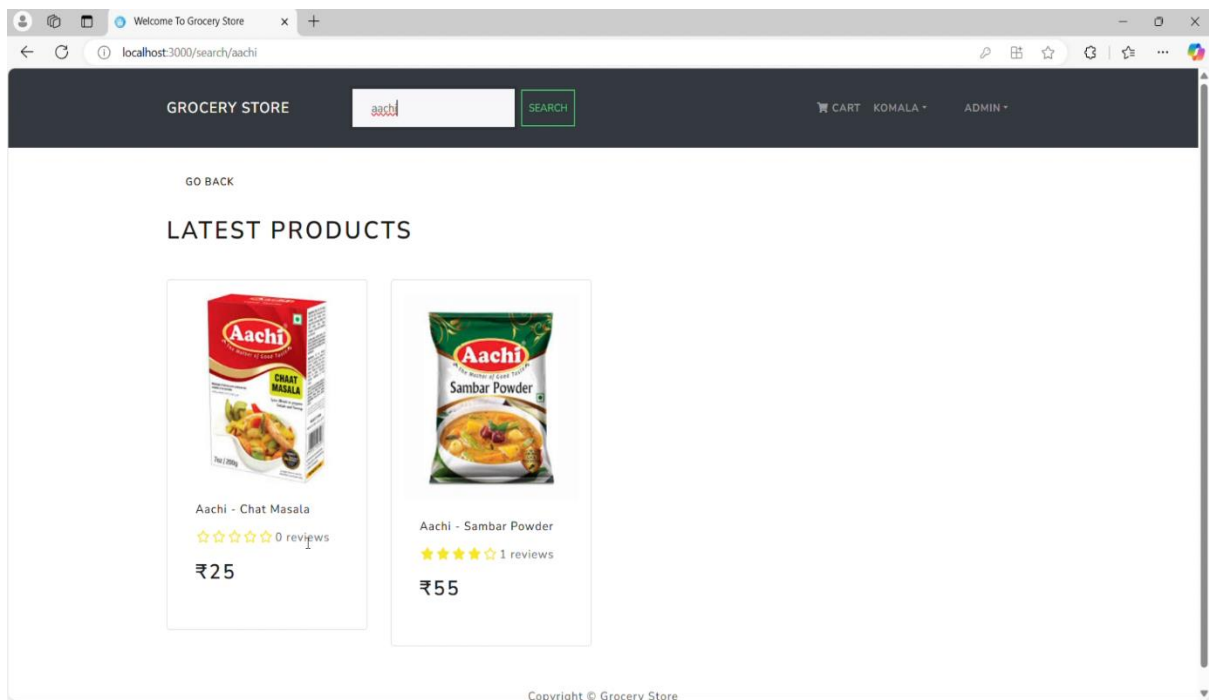
REGISTER

Have an Account? [Login](#)

Copyright © Grocery Store

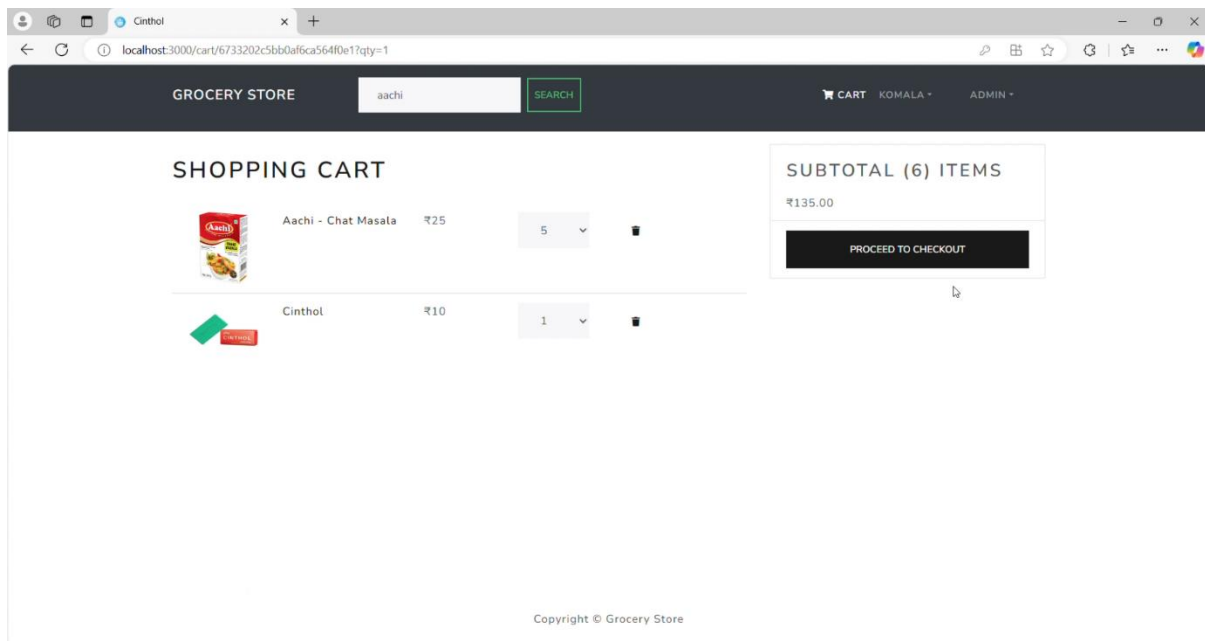
Search Page:

A powerful search interface that allows users to find specific groceries by name or category. Filters and sorting options enhance the search experience, helping users quickly locate their desired products.



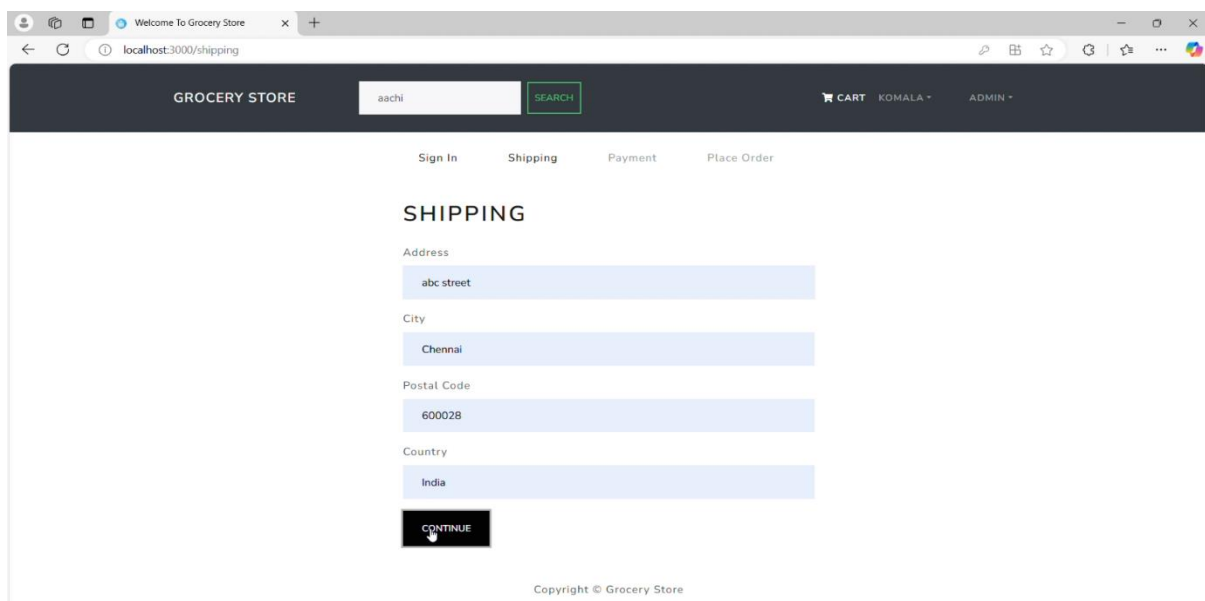
Cart Page:

A summary of items selected for purchase, where users can review, edit quantities, or remove items. The cart displays the total cost, applied discounts, and estimated shipping fees. It also has a button to proceed to checkout.



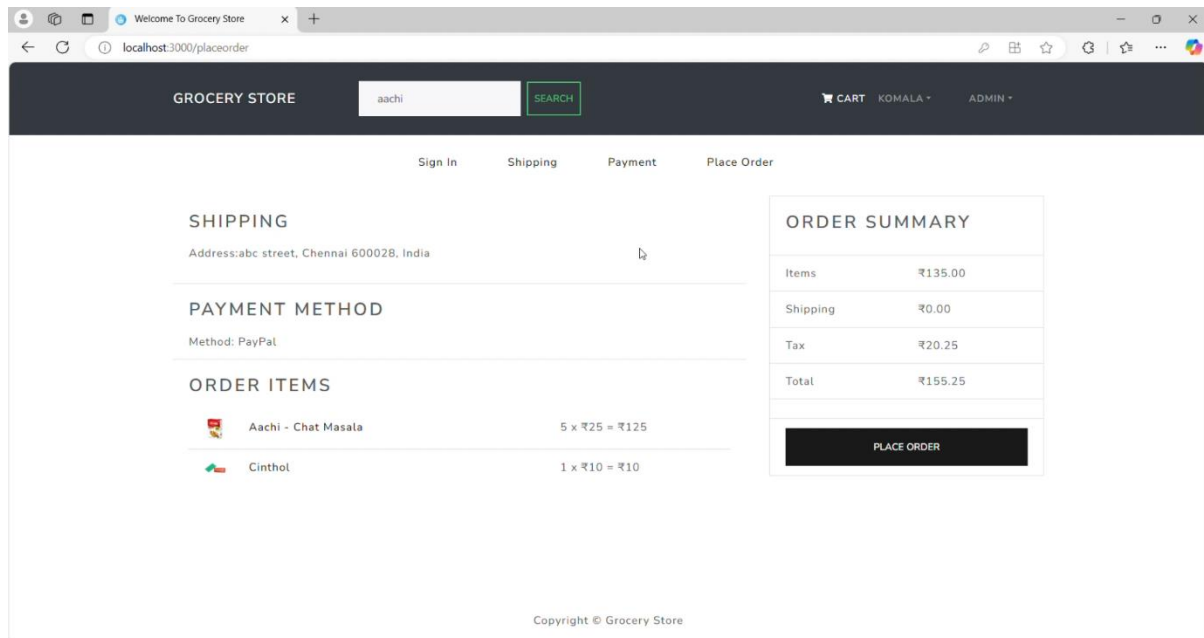
Shipping Page:

The page where users enter or confirm their shipping address and choose a delivery method. It includes options for expedited shipping or standard delivery, with clear information on delivery timelines and any associated fees.



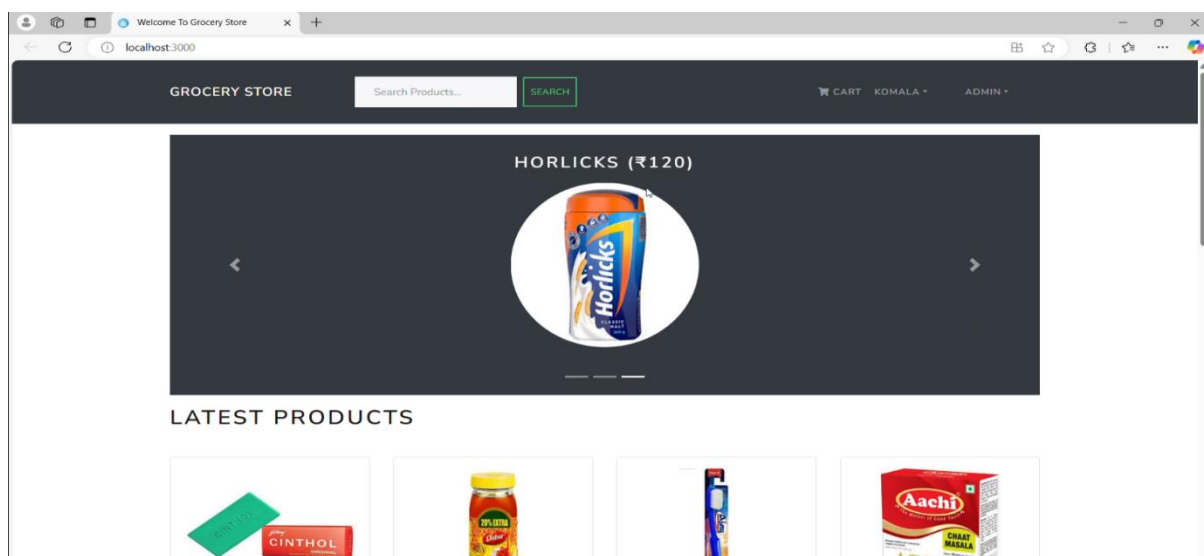
Order Page:

A final review page before purchase, showing a detailed order summary, selected items, total cost, shipping address, and payment options. After confirmation, users can track their order status, with regular updates on processing, shipping, and delivery.



Landing Page:

The welcome page that showcases the highlights of the grocery store. It includes promotional banners, featured products, and an easy-to-use navigation bar to explore product categories. This page aims to provide a user-friendly, visually appealing first impression to visitors.



11.Upcoming features

As our grocery web application continues to grow, several potential upgrades can enhance user experience, streamline operations, and boost competitiveness in the online grocery market. Here are some future enhancement ideas:

1. AI-Powered Product Recommendations

Feature:

Integrate AI algorithms to suggest personalized grocery items based on browsing patterns, purchase history, and individual preferences.

Benefits:

Enhanced Shopping Experience: Tailored suggestions make it easier for customers to discover relevant products.

Increased Sales: Targeted recommendations encourage customers to add more items to their cart, increasing the average order value.

Implementation:

Use machine learning to analyze customer behavior and predict items of interest. Display recommendations on the homepage, product pages, and within the cart for maximum visibility.

2. Real-Time Order Tracking

Feature:

Provide detailed, real-time order tracking so users can monitor their grocery orders from preparation to doorstep delivery, with notifications at key points.

Benefits:

Improved Transparency: Customers can track their order progress, enhancing trust and satisfaction.

Higher Engagement: Notifications via SMS or email keep customers informed and engaged throughout the delivery process.

Implementation:

Integrate with delivery partners' APIs for live tracking data. Design a user-friendly tracking interface that displays estimated delivery times and updates in real-time.

3. Chatbots for Customer Support

Feature:

Deploy AI-driven chatbots for real-time assistance, answering common questions about products, orders, and general inquiries.

Benefits:

24/7 Availability: Chatbots provide immediate responses, even outside business hours.

Efficient Support: Frees up human agents for complex issues, while the chatbot manages routine inquiries.

Implementation:

Utilize platforms like Dialogflow or Microsoft Bot Framework to develop a chatbot trained to handle queries on order tracking, product availability, delivery, and returns.

4. Multi-Language and Multi-Currency Support

Feature:

Introduce multi-language and multi-currency functionality to reach a broader customer base.

Benefits:

Wider Reach: Makes the platform accessible to users from various regions.

Localized Experience: Enhances customer comfort by allowing shopping in their preferred language and currency.

Implementation:

Use localization frameworks to support multiple languages and currencies, adjusting content dynamically. Integrate multi-currency payment gateways for seamless transactions.

5. Enhanced Admin Dashboard with Analytics

Feature:

Upgrade the admin dashboard with detailed analytics and reporting capabilities to empower admins with data-driven insights.

Benefits:

Informed Decisions: Admins gain insights into user behavior, sales trends, and inventory, helping them make strategic choices.

Operational Efficiency: Real-time access to data supports quick decision-making on stock, promotions, and customer support.

Implementation:

Incorporate tools like Google Analytics or build custom analytics dashboards for data visualization. Include features such as sales tracking, inventory alerts, and user engagement metrics.

6. Voice Search Integration

Feature:

Allow customers to search for products using voice commands, making it easier to find groceries hands-free.

Benefits:

Convenience: Voice search speeds up product discovery, particularly useful for mobile and smart device users.

Enhanced Accessibility: Adds a modern touch to the platform and improves usability for diverse customer groups.

Implementation:

Use speech recognition APIs (like Google Speech or Amazon Lex) to support voice search.

Add a voice-enabled search option on key pages like the homepage and search results.

7. Subscription Model for Regular Groceries

Feature:

Implement a subscription-based model for staple grocery items, allowing users to schedule recurring deliveries.

Benefits:

Customer Loyalty: Subscription plans encourage repeat orders and long-term customer retention.

Stable Revenue Stream: Regular subscriptions contribute to predictable revenue, reducing reliance on single-purchase orders.

Implementation:

Enable subscriptions for frequently purchased items, like milk, eggs, or cleaning supplies.

Offer incentives such as discounts or free shipping for users who subscribe to regular deliveries.

These future enhancements aim to enhance user engagement, streamline operations, and create a seamless and personalized shopping experience on our grocery platform.

The demo of the app is available at:

<https://drive.google.com/file/d/14Fj88E0UZY2vhBgdW91DtP5Mg7jkjKSY/view?usp=sharing>