NAAN MUDHALVAN PROJECT REPORT

TITLE : GROCERY WEBAPP - MERN

COLLEGE: AALIM MUHAMMED SALEGH COLLEGE OF ENGINEERING

MEMBERS:

1.ARSATH FARWESE M (110121104013)

2.BAROSH M (110121104901)

3.MAHESH TS (110121104035)

4.MOHAMED ABDUL RASIK S (110121104040)

Grocery WebApp MERN

Introduction:

Our basic grocery-web app! Our app is designed to provide a seamless online shopping experience for customers, making it convenient for them to explore and purchase a wide range of products. Whether you are a tech enthusiast, a fashionista, or a homemaker looking for everyday essentials, our app has something for everyone.

With user-friendly navigation and intuitive design, our grocery-webapp app allows customers to browse through various categories, view product details, add items to their cart, and securely complete the checkout process. We prioritize user satisfaction and aim to provide a smooth and hassle-free shopping experience.

For sellers and administrators, our app offers robust backend functionalities. Sellers can easily manage their product listings, inventory, and orders, while administrators can efficiently handle customer inquiries, process payments, and monitor overall app performance.

With a focus on security and privacy, our grocery-webapp app ensures that customer data is protected, transactions are secure, and personal information remains confidential. We strive to build trust with our customers and provide a safe platform for online shopping.

We are excited to have you on board and look forward to providing you with a delightful shopping experience. Happy shopping with our grocery-webapp!

Description:

The grocery webapp is a full-stack web application designed to revolutionize the way customers shop for groceries online. With the increasing demand for convenient, reliable, and secure online shopping platforms, our app aims to bridge the gap between customers and sellers, providing a seamless and intuitive shopping experience.

Key objectives:

For customers:

- Offer a user-friendly interface for browsing and purchasing products across various categories.
- Enable features like product search, category filtering, cart management, and secure checkout.
- Ensure a smooth and hassle-free shopping journey with responsive and reliable functionalities.

For sellers and administrators:

- Provide tools for sellers to manage product listings, update inventory, and track orders.
- Empower administrators with the ability to monitor application performance, handle customer inquiries, and manage transactions efficiently.

Core features:

Customer-centric design:

- Simple navigation and an intuitive layout to enhance user experience.
- Personalized recommendations and quick access to product details.

Backend robustness:

- Tools for sellers to add, edit, and update product details dynamically.
- Administrative controls for handling operations and ensuring platform efficiency.

Security and privacy:

Our grocery webapp prioritizes the security and privacy of its users. By employing technologies like json web tokens (jwt) for authentication and secure payment methods, the platform ensures:

- Protection of customer data.
- Confidentiality of personal and transactional information.

Vision:

This project is designed not just to facilitate online grocery shopping but also to build trust and provide a scalable solution for the growing e-commerce market. It serves as a platform for buyers and sellers to interact efficiently, creating a delightful experience for all stakeholders involved.

Scenario based case-study:

Filtering by Category and Producers

The Grocery WebApp's ability to filter items by categories and producers provides a streamlined shopping experience for users. For John, a busy professional with little time to browse, this feature ensures he can quickly locate essentials like fresh vegetables or snacks without searching through unrelated products. Similarly, Lisa, a homemaker, finds this functionality invaluable for managing her grocery runs efficiently, as it helps her identify items from trusted producers, ensuring quality for her family. For sellers like Sarah, filtering by producers helps her promote her locally-sourced products to a targeted customer base, increasing visibility and sales.

Dynamic Cart System

The app's real-time cart system enhances user convenience by enabling instant updates to the shopping cart. For John, this means he can quickly add, remove, or modify items on the go, making his online shopping as efficient as possible. For Lisa, it provides clarity on her spending as she adjusts her purchases based on her household budget.

On the seller side, Sarah benefits from accurate and immediate updates on customer orders, streamlining the fulfillment process.

Admin Panel for Sellers and Administrators

The admin panel empowers Sarah, a small-scale seller, by providing tools to manage inventory, update product listings, and track sales performance with ease. It eliminates the need for complex third-party software, allowing her to focus on expanding her customer base. For administrators, the panel facilitates efficient platform oversight, including monitoring user activity, resolving customer queries, and maintaining the app's overall functionality.

Secure Checkout System

The app's secure and efficient checkout process benefits both customers and sellers. For Lisa, this feature ensures that her transactions are completed safely, giving her peace of mind when making online payments. Similarly, John appreciates the quick and seamless checkout, allowing him to wrap up his shopping in no time. For Sarah, a smooth checkout process reduces the chances of cart abandonment, leading to increased sales and customer satisfaction.

Saved Lists for Easy Reordering

The saved lists feature is particularly helpful for Lisa, who frequently orders the same set of groceries. By saving her regular shopping list, she can complete her orders with minimal effort, freeing up time for other household tasks. For busy professionals like John, it provides a quick way to reorder essentials without starting from scratch each time, ensuring convenience and time savings.

Technical architecture:

1. Frontend Layer

The frontend is built using **React.js**, providing a dynamic and responsive user interface.

Technologies Used:

- React.js for component-based UI.
- Redux for state management to handle global states like the cart and user authentication.
- Tailwind CSS/Bootstrap for styling and responsiveness.

Responsibilities:

- Rendering the user interface, including product categories, cart, and checkout.
- Interacting with backend APIs to fetch data (e.g., product lists) and send user actions (e.g., add to cart).
- Managing routes using React Router for navigation between pages.

2. Backend Layer

The backend is implemented using **Node.js** and **Express.js**, providing RESTful APIs for communication between the frontend and database.

Technologies Used:

- Node.js for server-side logic.
- Express.js for building APIs.
- JSON Web Tokens (JWT) for secure user authentication.

Responsibilities:

- Handling user authentication and authorization.
- Exposing APIs for CRUD operations (e.g., managing products, orders, and user data).
- Processing business logic, such as calculating cart totals and verifying order details.
- Managing middleware for input validation and error handling.

3. Database Layer

The app uses **MongoDB**, a NoSQL database, to store and manage application data.

Technologies Used:

- MongoDB for data storage.
- Mongoose for object data modeling (ODM).

Responsibilities:

- Storing user information, including login credentials and order history.
- Maintaining a catalog of products with categories, producers, and pricing details.
- Recording cart details and order transactions.

4. Communication Flow

Frontend Request:
The user interacts with the app through the React.js frontend, which triggers an HTTP request via Axios to the backend API endpoints.

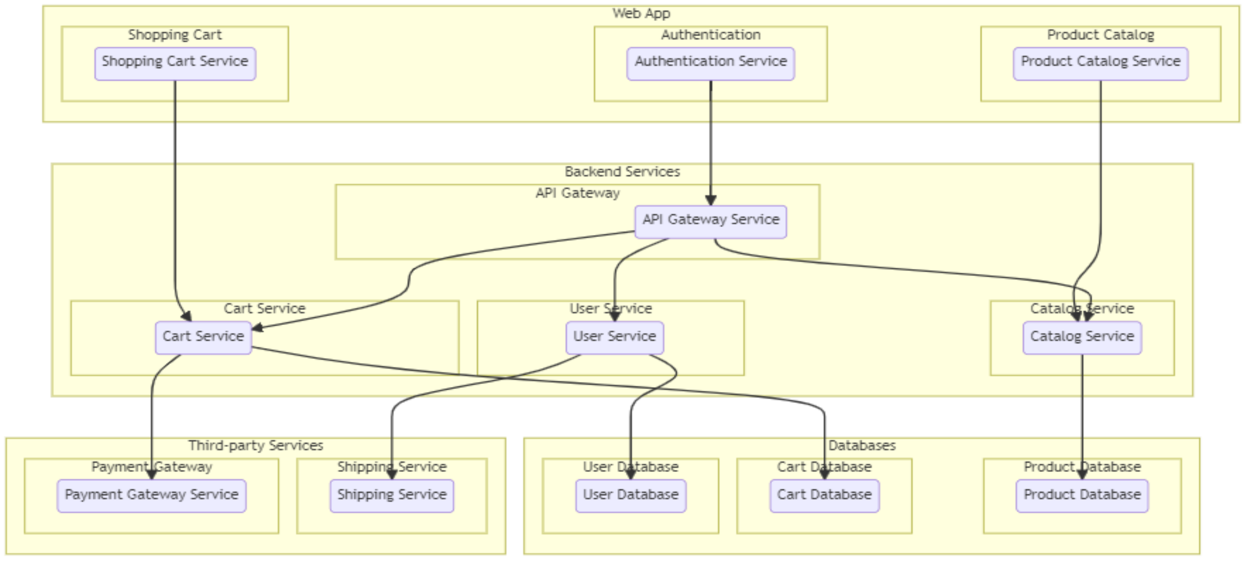
Example: John clicks "Add to Cart," sending a POST request to the `/cart` endpoint.

Backend Processing:
The Node.js backend receives the request, processes the logic (e.g., updating the cart), and interacts with MongoDB to update or retrieve data.

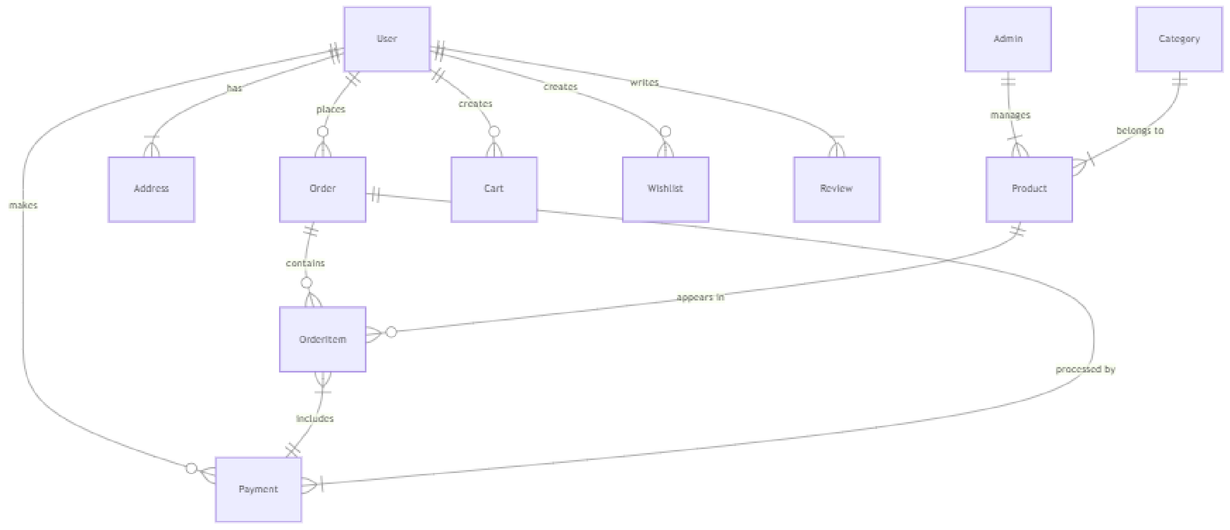
Example: The server updates the cart collection in MongoDB to reflect the added product.

Database Query:
The backend queries MongoDB through Mongoose to retrieve or update data. Once the operation is complete, the result is sent back to the backend.

Response to Frontend:
The backend responds with the result (e.g., a success message or the updated cart details) in JSON format. The frontend updates the UI accordingly


Web App
Shopping Cart
Shopping Cart Service
Authentication
Authentication Service
Product Catalog
Product Catalog Service
Backend Services
API Gateway
API Gateway Service
Cart Service
Cart Service
User Service
User Service
Catalog Service
Catalog Service
Third-party Services
Payment Gateway
Payment Gateway Service
Shipping Service
Shipping Service
Databases
User Database
User Database
Cart Database
Cart Database
Product Database
Product Database

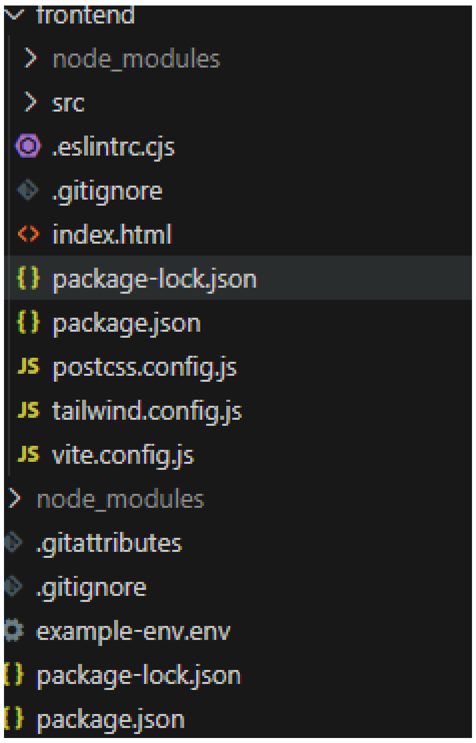
Er diagram:


User
Admin
Category
has
places
creates
creates
writes
manages
belongs to
Address
Order
Cart
Wishlist
Review
Product
makes
contains
appears in
processed by
OrderItem
includes
Payment

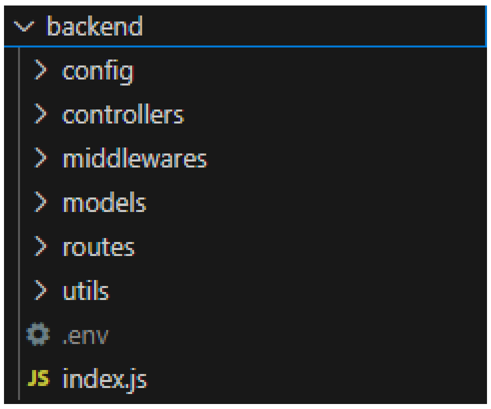
The Entity-Relationship (ER) diagram for the Grocery WebApp illustrates the relationships between key entities such as Users, Admins, Products, Orders, and Payments. Users can place orders, create shopping carts and wishlists, write product reviews, manage multiple addresses, and make payments. Admins oversee product and category management. Products are organized into categories, appear in carts, wishlists, and orders, and are subject to user reviews. Each order contains multiple order items, representing the individual products purchased, and is linked to a payment. The diagram highlights the interconnectedness of these entities, providing a clear structure for the app's functionality, focusing on user interaction, order processing, and product management.

Project structure:

The project is divided into two main sections: the **backend** (server-side code) and the **frontend** (client-side code). Each section has its own organized structure to ensure maintainability, scalability, and a clear separation of concerns.



```
frontend
  node_modules
  src
  .eslintrc.cjs
  .gitignore
  index.html
  package-lock.json
  package.json
  postcss.config.js
  tailwind.config.js
  vite.config.js
node_modules
.gitattributes
.gitignore
example-env.env
package-lock.json
package.json
```



```
backend
  config
  controllers
  middlewares
  models
  routes
  utils
  .env
  index.js
```

Backend Structure:

The **backend/** directory contains all server-side code necessary for managing the database, handling API requests, and processing business logic.

- **config/**: This folder contains configuration files such as `db.js`, which sets up the MongoDB database connection and manages environment variables like API keys, JWT secrets, and database URIs.

controllers/: This folder contains JavaScript files that define the business logic for handling requests. For instance, `userController.js` handles user registration and login, `orderController.js` processes the details of user orders, and `paymentController.js` integrates payment systems like Stripe or PayPal.

models/: The **models/** folder defines Mongoose schemas that structure data in the MongoDB database. For example, `User.js` defines the structure for user data, `Product.js` outlines the details of products available in the system, and `Order.js` defines the structure for orders placed by users.

routes/: The **routes/** directory holds API route definitions for different functionalities, such as `userRoutes.js` for user authentication, `productRoutes.js` for managing products, and `orderRoutes.js` for handling order processing.

middleware/: Middleware files in this folder handle essential operations like data validation and authentication. For instance, `authMiddleware.js` ensures that only authenticated users can access specific routes, and `validationMiddleware.js` validates incoming data for correctness before processing.

server.js: This is the entry point for the backend Express server. It initializes the server and ties together all the routes, middleware, and models, allowing the backend to handle client requests.

utils/: The **utils/** folder contains utility functions that can be used across different parts of the application, such as email handling for sending order confirmation emails.

Frontend Structure:

The **frontend/** directory houses the client-side React application that users interact with. It is responsible for the user interface, including product listings, user authentication, and order placement.

public/: This folder contains static files like the `index.html` template, which serves as the entry point for the React app. Other static resources like images and fonts are also stored here.

src/: This is the core of the React application, containing components, pages, and services that control the application's behavior and appearance.

components/: The **components/** folder includes reusable React components, such as `Navbar.js` for the navigation bar, `ProductCard.js` for displaying individual product details, and `Cart.js` for managing the user's shopping cart. These components can be combined to create different pages and views.

pages/: The **pages/** folder contains the different views of the application, such as `Home.js`, which displays product categories and featured items, `ProductList.js` for listing products with filters and search, and `CartPage.js`, which shows the contents of the user's shopping cart.

services/: The **services/** directory contains files like `userService.js` and `productService.js`, which are responsible for making API requests to the backend. These services handle actions like logging in users, fetching product details, and submitting orders.

App.js: This is the root React component that includes the main layout and structure of the application, routing between pages, and rendering various components based on the current route.

styles/: The **styles/** folder contains the CSS or SCSS files used to style the components of the React app. These styles define the appearance of the app, including layout, typography, and color schemes.

Additional Files:

.env: This file contains environment variables for both the frontend and backend, such as database connection strings, API keys, and JWT secrets.

package.json: This file manages the project's dependencies for both the frontend and backend, and contains scripts for running and building the application.

.gitignore: This file defines which files and directories should be excluded from version control. It typically includes directories like `node_modules` and environment files like `.env`.

README.md: The README file provides an overview of the project, including setup instructions, features, and usage.

Pre-requisite

Here are the key prerequisites for developing a full-stack application using express js,

Mongodb, and react.js:

✔*node.js and npm:*

- ➢ Node.js is a powerful javascript runtime environment that allows you to run javascript code on
- ➢ The server-side. It provides a scalable and efficient platform for building network applications.
- ➢ Install node.js and npm on your development machine, as they are required to run javascript on the server-side.
- ➢ Download: https://nodejs.org/en/download/
- ➢ Installation instructions: https://nodejs.org/en/download/package-manager/

✔*express.js:*

- ➢ Express.js is a fast and minimalist web application framework for node.js. It simplifies the process of creating robust apis and web applications, offering features like routing, middleware support, and modular architecture.

- ➢ Install express.js, a web application framework for node.js, which handles server-side routing,
- ➢ Middleware, and api development.
- ➢ Installation: open your command prompt or terminal and run the following command: npm install express.

✔mongodb:

- ➢ Mongodb is a flexible and scalable nosql database that stores data in a json-like format. It
- ➢ Provides high performance, horizontal scalability, and seamless integration with node.js, making
- ➢ It ideal for handling large amounts of structured and unstructured data.
- ➢ Set up a mongodb database to store your application's data.
- ➢ Download: https://www.mongodb.com/try/download/community
- ➢ Installation instructions: https://docs.mongodb.com/manual/installation/

✔react.js:

- ➢ React.js is a popular javascript library for building user interfaces. It enables developers to
- ➢ Create interactive and reusable ui components, making it easier to build dynamic and responsive Web applications.
- ➢ Install react.js, a javascript library for building user interfaces.
- ➢ Follow the installation guide: https://reactjs.org/docs/create-a-new-react-app.html

✔Html,css,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 express js server with the mongodb database and perform crud (create, read, update, delete) operations.

✔*front-end framework:*

Utilize react js to build the user-facing part of the application,including entering booking room, status of the booking, and user interfaces for the admin dashboard. For making better ui we have also used some libraries like material ui and bootstrap.

✔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*
- To set up the development environment for the Grocery WebApp, follow these necessary steps. First, clone the repository by running the command

```
git clone https://github.com/yourusername/grocery-webapp.git
```

- navigate into the project folder using `cd grocery-webapp`.
- Once in the project directory, you need to install the dependencies for both the backend and frontend. To do this, first, navigate to the `backend` directory with ‘`cd backend`’ and run ‘`npm install` to install the backend dependencies. Afterward, switch to the `frontend` directory by running `cd frontend` and execute `npm install` to install the frontend dependencies.
- Next, set up the required environment variables by creating a `.env` file in both the `backend` and `frontend` directories (if not already provided) and populate them with the following details: `MONGO_URI` (your MongoDB connection string), `JWT_SECRET` (a random string used for JWT token generation), and `PAYPAL_CLIENT_ID` (your PayPal client ID, if you are using PayPal integration).

- Once the dependencies are installed and the environment variables are configured, you can start the development servers.
- First, start the backend server by navigating to the `backend` directory and running `node index.js`.
- Then, start the frontend server by going to the `frontend` directory and executing `npm run dev`. After both servers are running, you can open your browser and navigate to `http://localhost:5173/` to access the application.
- These steps will set up the development environment and allow you to begin working with the application locally.

Application flow

The Grocery WebApp follows a simple yet effective application flow that guides the user from browsing products to completing orders. Upon accessing the homepage, users are presented with product categories and featured items, which they can browse through. Users can search for specific products or filter results based on categories such as vegetables, dairy, etc. When a user finds a product they wish to purchase, they can view its details, including price, description, and available stock.

Users have the option to add products to their cart or wishlist. The cart represents the items the user intends to purchase, while the wishlist allows users to save items for future consideration. Users can view their cart, update quantities, or remove items. When ready to checkout, the user proceeds to the order page, where they review the items in their cart and select a delivery address.

Once the user confirms the order, they are prompted to make a payment. Payments are processed via integrated payment gateways such as PayPal or Stripe, depending on the setup. Upon successful payment, the order is confirmed, and users receive an order confirmation along with an estimated delivery time.

For registered users, the application also supports authentication, allowing them to log in and view their order history, update personal details, and manage their addresses. New users can create accounts, while existing users can log in to access their account details.

Admin users have access to a different set of functionalities, such as managing product listings, viewing order details, and overseeing the application’s operation. They can add, edit, or delete product categories and manage inventory to ensure the smooth running of the platform.

In summary, the application flow guides users through the process of discovering products, managing their cart and wishlist, completing orders, making payments, and managing user details, while also providing admins with tools to manage the application’s content and operations.

Project flow

The Project Flow of the Grocery WebApp outlines the overall process from start to finish, covering both the user and administrative actions as well as the system's backend operations. The project flow begins when a user accesses the platform, where they are greeted with product categories, featured items, and a search function to explore the available grocery items.

The user can then browse products, filter by categories, or search for specific items. Once a user finds a product they want to purchase, they can add it to their cart or wishlist. The cart holds products that the user intends to buy, while the wishlist allows the user to save products for future purchases.

Once the user is ready to purchase, they proceed to the checkout process where they review the items in their cart, select a delivery address, and choose a payment method. The payment is processed through external services like PayPal or Stripe for secure transactions. Upon successful payment, an order is created, and the user receives an order confirmation, including an estimated delivery time.

For users with accounts, login authentication allows them to track previous orders, update personal details, and manage multiple delivery addresses. New users can create accounts to get started. The backend is responsible for managing user authentication, storing and retrieving product data, processing orders, handling payments, and managing user data, with MongoDB as the database for storing relevant information.

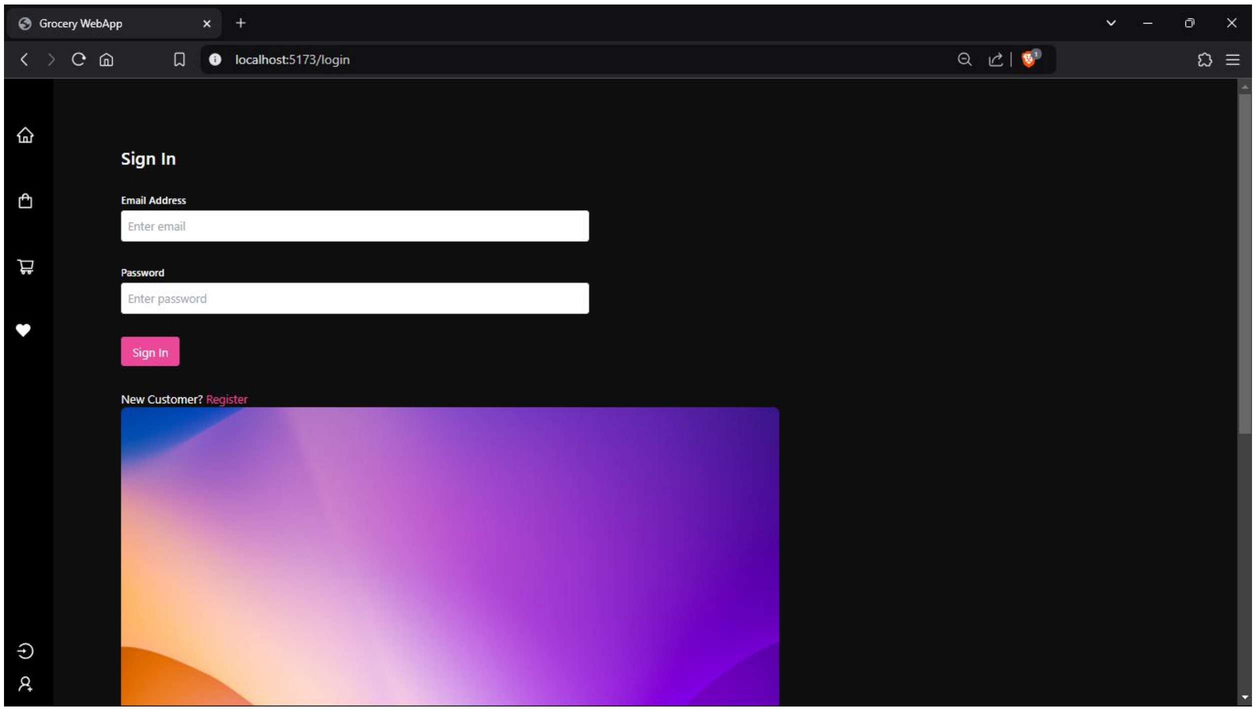
From the administrative side, admins have a broader control over the platform. They can add new products, modify existing ones, categorize products, and oversee order statuses. Admins also have access to user activity, including order details, to monitor the overall system and ensure smooth operation. The admin dashboard allows them to perform actions such as updating inventory, managing categories, and responding to user inquiries.

The **backend** consists of APIs built with Node.js and Express.js, responsible for handling requests, data processing, and interacting with the MongoDB database. The **frontend** is built using React.js and connects to the backend APIs to retrieve and display data to users. The frontend is also responsible for managing user interactions, such as logging in, viewing products, adding items to the cart, and completing the checkout process.

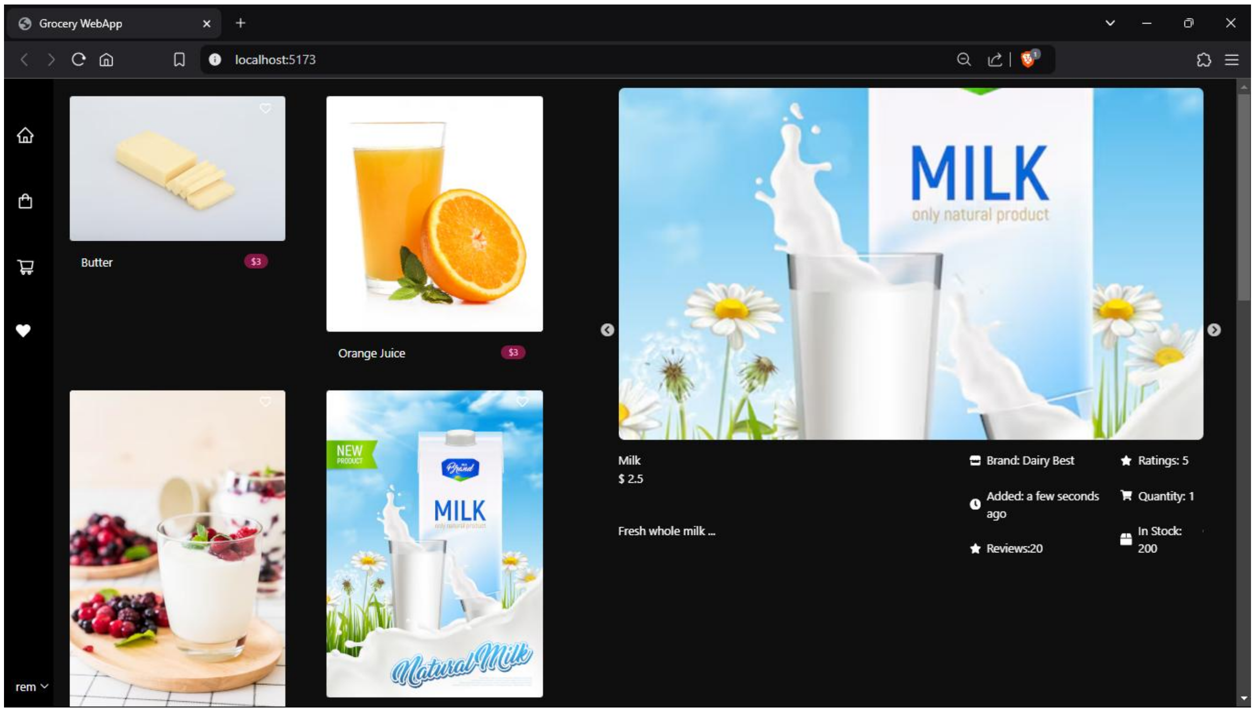
In summary, the Project Flow outlines the complete journey from user interaction with the platform to backend processing and administrative management. It ensures that the system runs efficiently, providing a seamless experience for users while allowing administrators to manage and control the platform.

Screenshots:

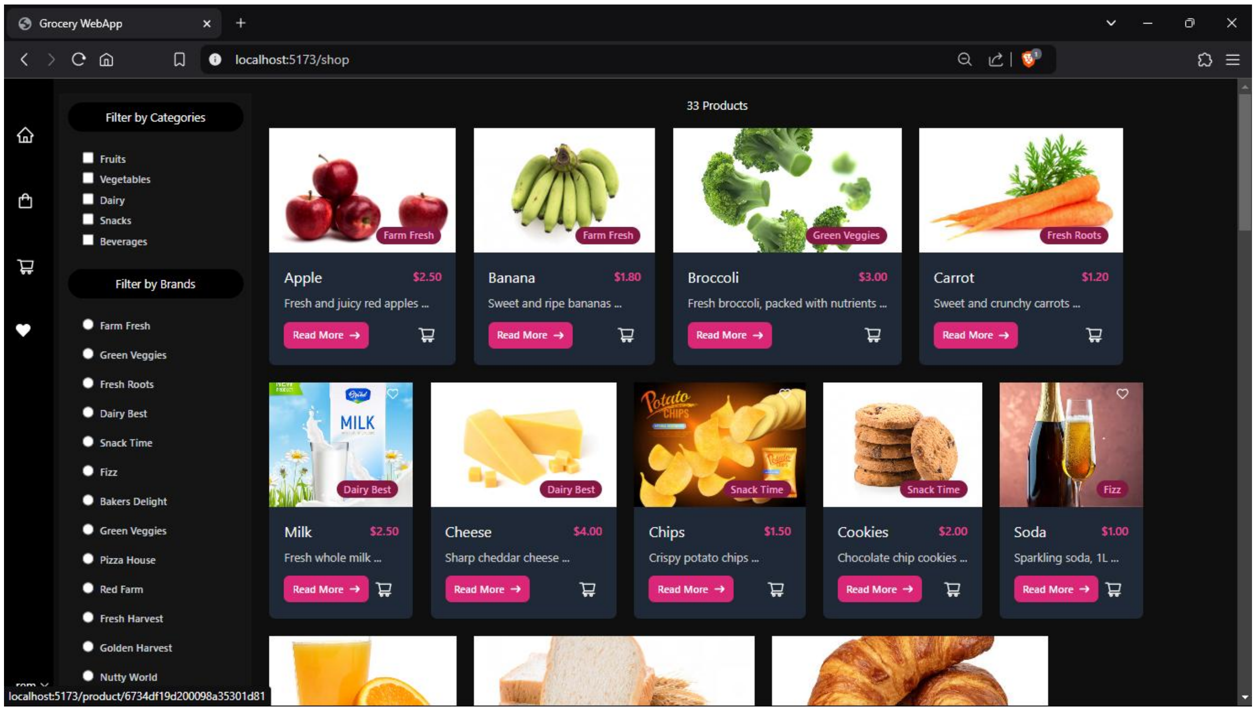
User Login:



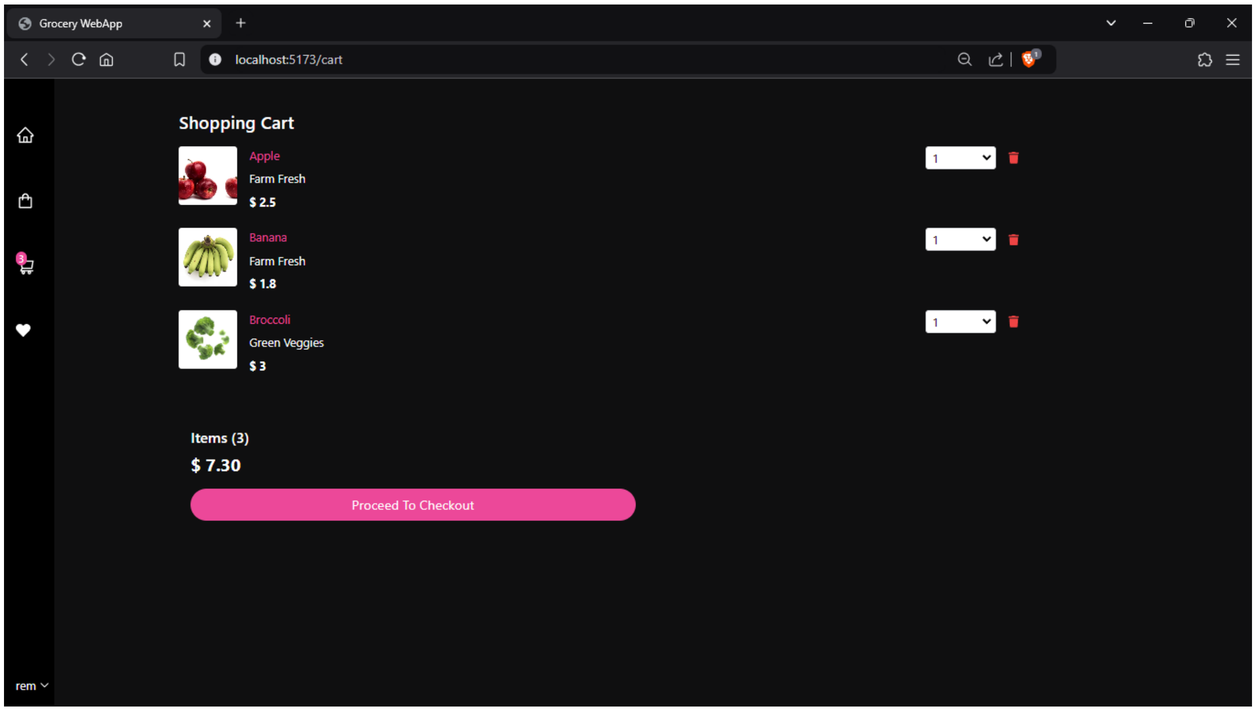
Home page:



Products:



Cart:



Shopping Cart

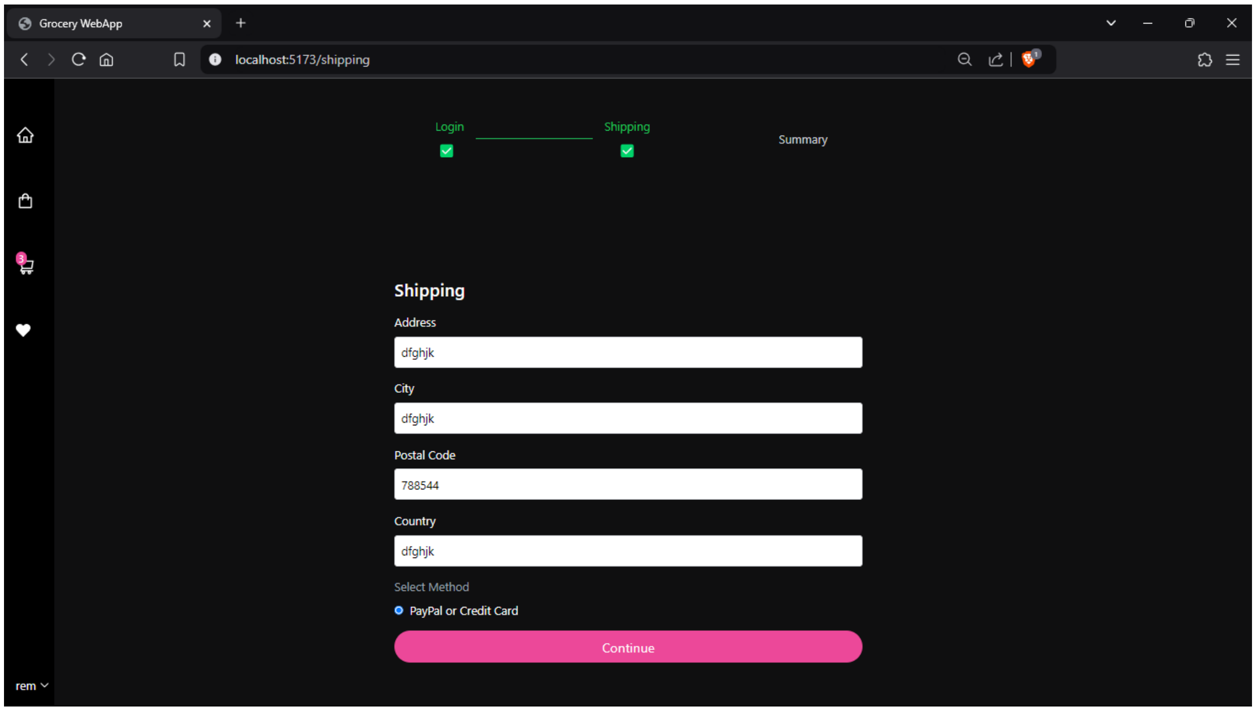
Apple	Farm Fresh	$ 2.5
Banana	Farm Fresh	$ 1.8
Broccoli	Green Veggies	$ 3

Items (3)

$ 7.30

Proceed To Checkout

User Information:



Shipping

Address: dfghjk

City: dfghjk

Postal Code: 788544

Country: dfghjk

Select Method

[✓ PayPal or Credit Card]

Continue

Checkout:

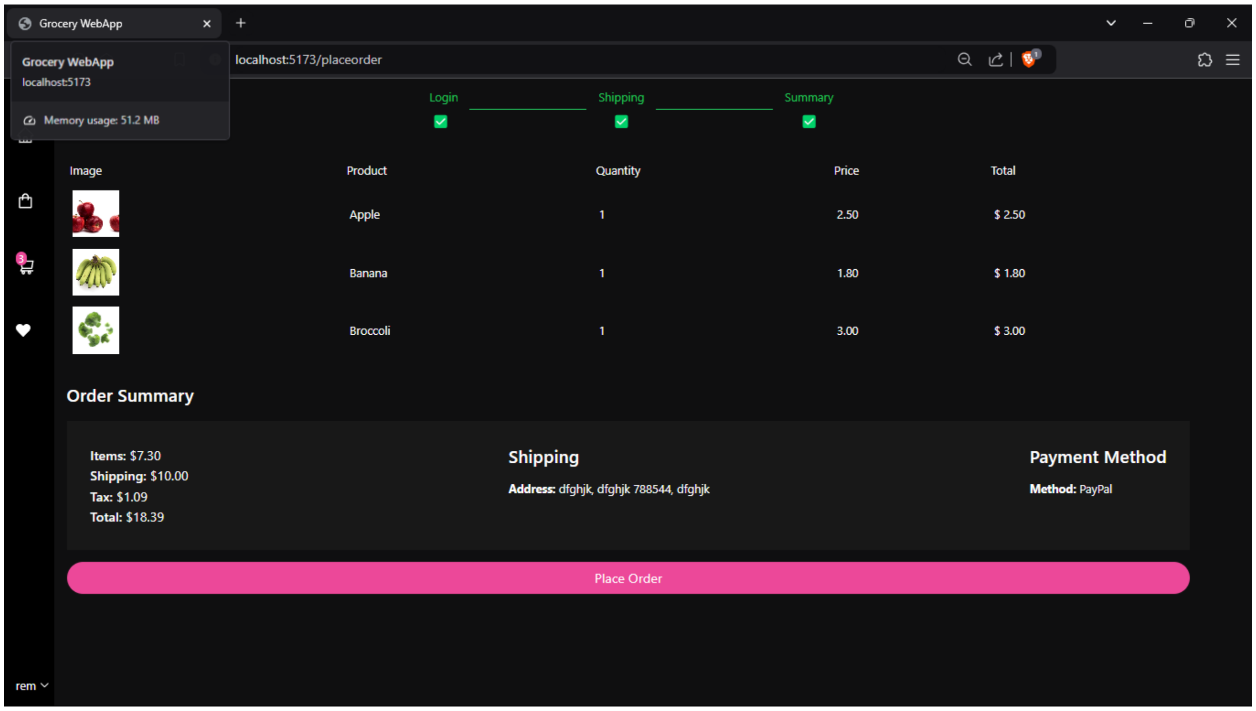


Image	Product	Quantity	Price	Total
	Apple	1	2.50	$ 2.50
	Banana	1	1.80	$ 1.80
	Broccoli	1	3.00	$ 3.00

Order Summary

Items: $7.30
Shipping: $10.00
Tax: $1.09
Total: $18.39

Shipping

Address: dfghjk, dfghjk 788544, dfghjk

Payment Method

Method: PayPal

Place Order

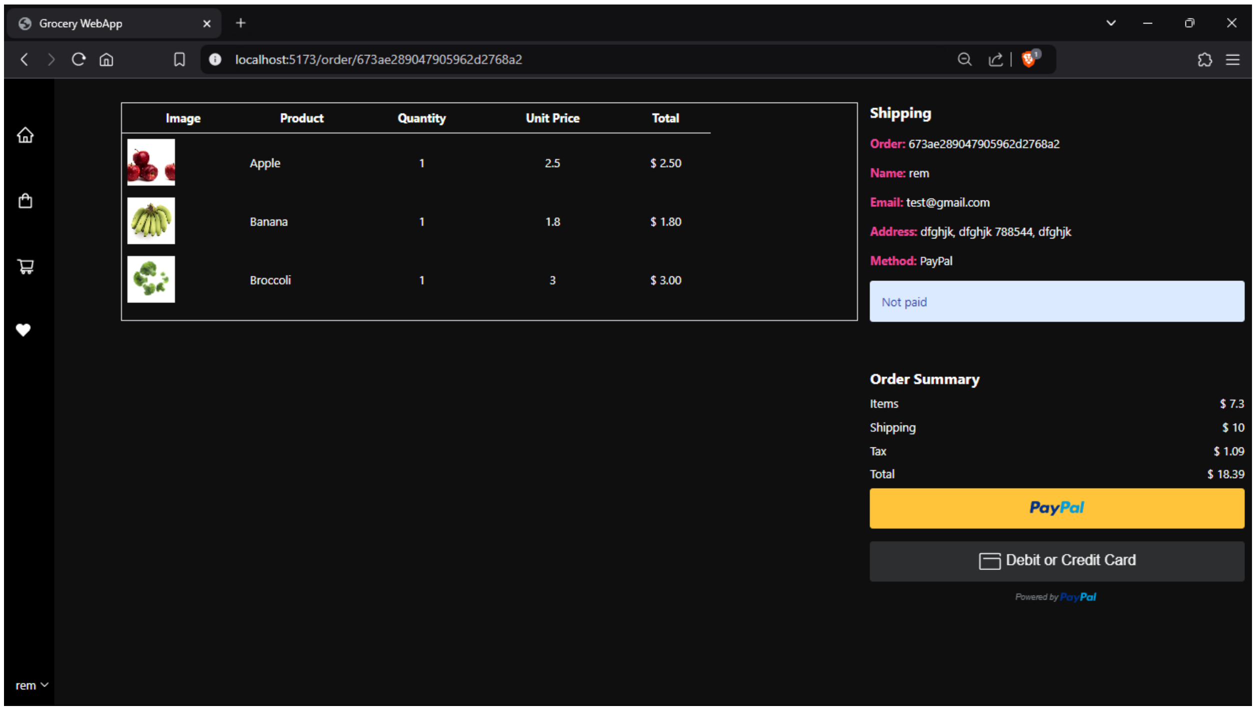


Image	Product	Quantity	Unit Price	Total
	Apple	1	2.5	$ 2.50
	Banana	1	1.8	$ 1.80
	Broccoli	1	3	$ 3.00

Shipping

Order: 673ae289047905962d2768a2
Name: rem
Email: test@gmail.com
Address: dfghjk, dfghjk 788544, dfghjk
Method: PayPal

Not paid

Order Summary

Items	$ 7.3
Shipping	$ 10
Tax	$ 1.09
Total	$ 18.39

Known Issues

Payment Integration Issues: Occasionally, users may experience delays or errors during the payment process when using third-party payment gateways like PayPal or Stripe. This could be due to connectivity issues with the payment services or API response time.

Product Availability Mismatch: Sometimes, the stock quantity displayed on the product page may not accurately reflect the available inventory due to lag in syncing between the database and the frontend.

Mobile Responsiveness: Although the web app is designed to be mobile-friendly, certain elements on small screens may not display optimally, particularly in the **cart** and **checkout** views. Further adjustments are needed for better responsiveness.

User Authentication Issues: Some users have reported issues with logging in or registering new accounts under certain network conditions, which may be related to server-side session handling or cookie storage in browsers.

Slow Performance with Large Data Sets: When browsing categories with a large number of products, the frontend may experience performance degradation due to insufficient data pagination or optimization in the backend.

Order Status Updates: In some cases, the order status may not update in real-time, leading to confusion about the delivery timeline. This is caused by delays in communication between the frontend and backend systems.

Future Enhancements

Real-Time Order Tracking: Implementing real-time order tracking for users to view the live status of their orders would enhance the user experience and provide transparency during the delivery process.

Advanced Search Filters: Adding more advanced search filters (e.g., price range, ratings, brand filters) would help users find products more quickly and efficiently, especially as the product catalog grows.

Recommendation System: Implementing a recommendation engine based on user preferences and browsing history would improve the user experience by suggesting products tailored to individual tastes.

Multi-language Support: Expanding the platform to support multiple languages would make the application more accessible to a broader audience, especially in international markets.

Guest Checkout Option: Currently, users are required to create an account before completing a purchase. Implementing a guest checkout option would streamline the purchasing process for users who prefer not to create an account.

Admin Dashboard Improvements: Enhancing the admin dashboard with more powerful analytics and reporting features would allow administrators to gain insights into sales performance, user behavior, and inventory management.

Subscription Service: Introducing a subscription model for users who wish to receive regular deliveries of their favorite items (e.g., weekly grocery orders) could increase user engagement and sales.

Mobile Application: Developing a mobile version of the app for both iOS and Android platforms would make the app more accessible and convenient for users who prefer to shop on their mobile devices.