MERN STACK POWERED BY MONGODB

GROCERY WEBAPP

A PROJECT REPORT

Submitted by

DHANUSH J	113321104016
MOHANKUMAR V S	113321104061
PAVANKUMAR V	113321104073
VIGNESH B	113321104114

BACHELOR OF TECHNOLOGY

COMPUTER SCIENCE AND ENGINEERING

VELAMMAL INSTITUTE OF TECHNOLOGY

CHENNAI 601 204

ANNA UNIVERSITY: CHENNAI 600 025

ANNA UNIVERSITY CHENNAI: 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"GROCERY WEB APP"** is the Bonafide work of **"DHANUSH J - 113321104016, MOHANKUMAR V S - 113321104061, PAVANKUMAR V - 113321104073, VIGNESH B - 113321104114"** who carried out the project work under my supervision.

SIGNATURE	SIGNATURE
Dr.V.P.Gladis Pushparathi	**Mrs. Joice Ruby J**
PROFESSOR,	**ASSISSTANT PROFESSOR ,**
HEAD OF THE DEPARTMENT,	**NM COORDINATOR ,**
Computer Science and Engineering, Velammal Institute of Technology, Velammal Gardens, Panchetti, Chennai-601 204.	Computer Science and Engineering, Velammal Institute of Technology, Velammal Gardens, Panchetti, Chennai-601 204

ACKNOWLEDGEMENT

We are personally indebted to many who had helped us during the course of this project work. Our deepest gratitude to the **God Almighty.**

We are greatly and profoundly thankful to our beloved Chairman **Thiru.M.V.Muthuramalingam** for facilitating us with this opportunity. Our sincere thanks to our respected Director **Thiru.M.V.M Sasi Kumar** for his consent to take up this project work and make it a great success.

We are also thankful to our Advisors **Shri.K.Razak**, **Shri.M.Vaasu**, our Principal **Dr.N.Balaji** and our Vice Principal **Dr.S.Soundararajan** for their never ending encouragement that drives ustowards innovation.

We are extremely thankful to our Head of the Department **Dr.V.P.Gladis Pushparathi** and NaanMudhalvan Coordinator **Mrs. Joice Ruby J**, for their valuable teachings and suggestions.

The Acknowledgment would be incomplete if we would not mention word of thanks to our Parents.Teaching and Non-Teaching Staffs, Administrative Staffs and Friends for their motivation and support throughout the project.

Finally,we thank all those who directly or indirectly contributed to the successful completion of this project .Your contributions have been a vital part of our success

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

The **Grocery-Web App** reimagines the traditional shopping experience by introducing a dynamic online platform tailored to meet the needs of both customers and sellers. Its goal is to provide a seamless, secure, and efficient solution for purchasing and managing groceries and other essential items, ensuring convenience for users across diverse lifestyles, from tech enthusiasts to homemakers.

At its core, the app emphasizes a customer-first approach. It features a sleek and intuitive interface that allows users to navigate effortlessly through various product categories, review detailed product information, manage shopping carts, and complete transactions through a secure and straightforward checkout process. Designed with user satisfaction in mind, the app offers a smooth and enjoyable shopping experience.

For sellers and administrators, the app integrates robust backend functionalities. Sellers can efficiently manage product inventories, update listings, and oversee order processing. Administrators are equipped with tools to facilitate customer service, manage payment transactions, and monitor the app's overall performance, ensuring continuous optimization and reliability.

Security and privacy are foundational to the Grocery-Web App's design. Advanced data protection measures safeguard user information and transactions, creating a trustworthy platform for online shopping. By prioritizing these aspects, the app builds confidence among users and establishes itself as a dependable solution in the digital marketplace.

This project embodies technical expertise and a commitment to user-centered design, delivering a high-quality e-commerce experience that caters to a broad and dynamic audience. The Grocery-Web App contributes to the expanding digital ecosystem by providing convenience, security, and satisfaction for all participants in the shopping process.

To ensure long-term relevance and adaptability, the app is built with scalability in mind. Regular updates accommodate an expanding user base and incorporate new features. Exclusive perks, such as discounts and loyalty rewards, coupled with secure payment integration, enhance the overall shopping journey. By bridging the gap between local businesses and digital consumers, the platform not only strengthens small enterprises but also nurtures a sense of community growth and engagement.

The app’s commitment to sustainability is reflected in its eco-friendly practices, such as offering digital invoices and promoting sustainable packaging solutions. Multilingual support and accessibility features make the platform inclusive and user-friendly for diverse demographics. Its forward-thinking design ensures that the app evolves alongside changing customer expectations and market trends, empowering businesses to thrive in the digital economy.

By merging modern technology with user-centric features, the Grocery-Web App sets a new standard in e-commerce, creating an impactful and innovative solution that transforms the way people shop and sell groceries.

CHAPTER 2

PROJECT OVERVIEW

The Grocery-Web App is designed as a modern online shopping platform, offering users a seamless and efficient way to explore and purchase groceries and other essentials. Whether you're a tech-savvy professional, a fashion-forward individual, or a homemaker managing daily needs, the app caters to diverse lifestyles, ensuring convenience for everyone.

PURPOSE AND GOALS

The purpose of the Grocery-Web App is to simplify the grocery shopping experience by digitizing the entire process. It tackles common issues such as the hassle of in-store shopping, limited information on available deals, and inefficient grocery management. By leveraging technology, the app enables users to effortlessly browse a wide array of products, add items to their carts, and complete secure transactions—all from the comfort of their homes.

KEY GOALS

- Enhance user satisfaction through a simple, intuitive interface.
- Provide a smooth shopping journey with features like personalized promotions and real-time order tracking.
- Empower users with order history management for better planning and reordering.

On the seller side, the app offers robust management tools to oversee inventory, process orders, and analyze sales performance. These tools not only improve operational efficiency but also enable superior customer service with timely updates and optimized stock management.

Security and data privacy are at the heart of the app's design. Features like secure checkout and stringent data protection measures ensure user trust and compliance with modern privacy standards.

In summary, the Grocery-Web App bridges the gap between traditional and digital grocery shopping by creating a versatile platform. Its goal is to establish a sustainable, user-friendly ecosystem that saves time, simplifies shopping, and enhances user control over grocery needs.

FEATURES

1. **User Registration and Authentication:**

- Secure sign-up and login using JWT authentication.
- Social media integration for faster onboarding.

2. **Product Browsing and Search:**

- Browse items by categories such as Fruits, Vegetables, and Beverages.
- Use advanced filters for price, ratings, and brands.

3. **Cart and Wishlist:**

- Add items to the cart for immediate purchase.
- Save favorite items to a wishlist for future reference.

4. **Checkout and Payments:**

- Multiple payment options, including credit/debit cards, UPI, and digital wallets.
- Integration with payment gateways like PayPal and Razorpay ensures secure transactions.

5. **Order Management:**

- Real-time tracking from order placement to delivery.
- Notifications for status updates.

6. **Admin Dashboard:**

- Tools to add, update, or delete products.
- Monitor user activity and efficiently manage orders.

7. **Data Analytics:**

- Visual dashboards for sales trends, inventory usage, and customer preferences.

8. **Mobile Responsiveness:**

- Fully optimized design for desktops, tablets, and smartphones.

Unique Selling Points:

- **Personalized Experience**: AI-based recommendation engine for suggesting products based on user behavior.
- **User-Friendly Interface**: Intuitive design with easy navigation for all age groups.
- **Efficiency**: Swift loading times and minimal latency to enhance user experience.
- **Security**: End-to-end encryption for secure user data handling and payment processing.

Target Audience:

- **Busy Professionals**: Individuals seeking quick and convenient grocery solutions.
- **Retail Grocery Stores**: Businesses looking to transition into digital operations.
- **Urban and Semi-Urban Customers**: Users with internet access and mobile devices, seeking ease in shopping.

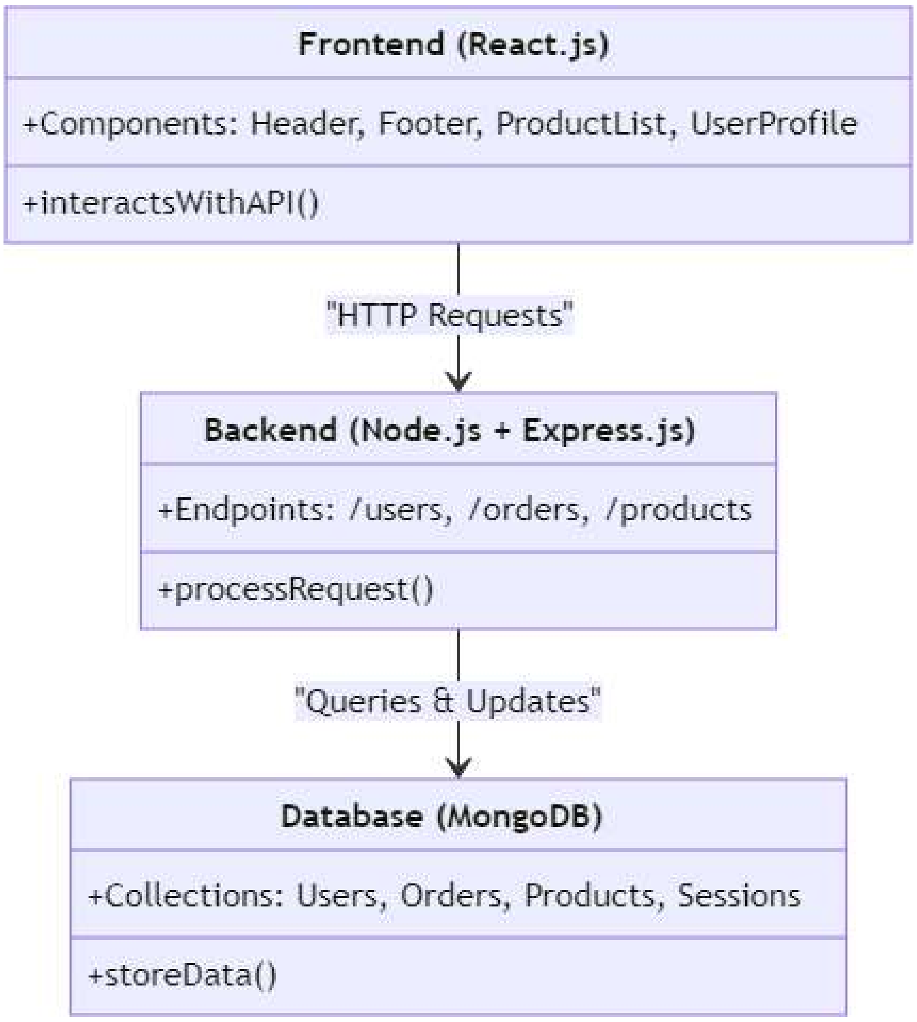
Business Benefits:

- **Increased Market Reach**: Digitizing operations opens new avenues for grocery stores to attract customers.
- **Improved Operations**: Centralized dashboards streamline inventory and order management.
- **Enhanced Customer Satisfaction**: A smoother shopping experience leads to higher retention and repeat business.

CHAPTER 3

ARCHITECTURE

The architecture of the Grocery Web App is structured to prioritize scalability, efficiency, and ease of use. Built using the **MERN stack** (**MongoDB, Express.js, React.js, Node.js**), the app adopts a modular and performance-driven design. Its architecture is organized into three core components—**Frontend**, **Backend**, and **Database**—ensuring seamless interaction and integration between them.

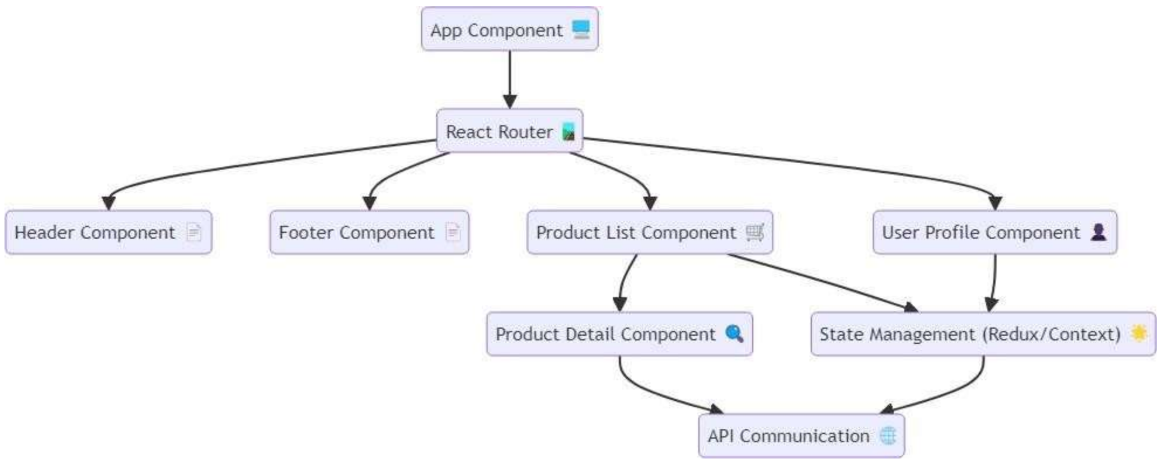

Frontend (React.js)
+Components: Header, Footer, ProductList, UserProfile
+interactsWithAPI()
"HTTP Requests"
Backend (Node.js + Express.js)
+Endpoints: /users, /orders, /products
+processRequest()
"Queries & Updates"
Database (MongoDB)
+Collections: Users, Orders, Products, Sessions
+storeData()

FRONT END

The frontend of the Grocery Web App is crafted using **React.js**, a powerful JavaScript library renowned for its speed and reusable component-based architecture. The focus of the design is on delivering a responsive, intuitive, and user-friendly interface that simplifies navigation and enhances the overall user experience.

Key Features:

- **Component-Based Design**:
  Each UI element, such as the product list, shopping cart, and user profile, is implemented as an independent, reusable component.
  - Example Components:
    - ProductCard for displaying individual products.
    - Cart for managing items added by users.
    - Navbar for seamless navigation across pages.
- **State Management**:
  - **Redux** or React Context API is used to manage global states, such as user authentication, cart data, and order status.
  - Ensures smooth data flow across components without redundancy.
- **Routing**:
  - **React Router** is used for efficient navigation between pages (e.g., Home, Product Details, Checkout).
  - Implements dynamic routes for viewing individual product details.
- **Styling and Responsiveness**:
  - CSS frameworks like **Bootstrap** and custom SCSS are used for styling.
  - Ensures mobile-first design for optimal performance across devices.



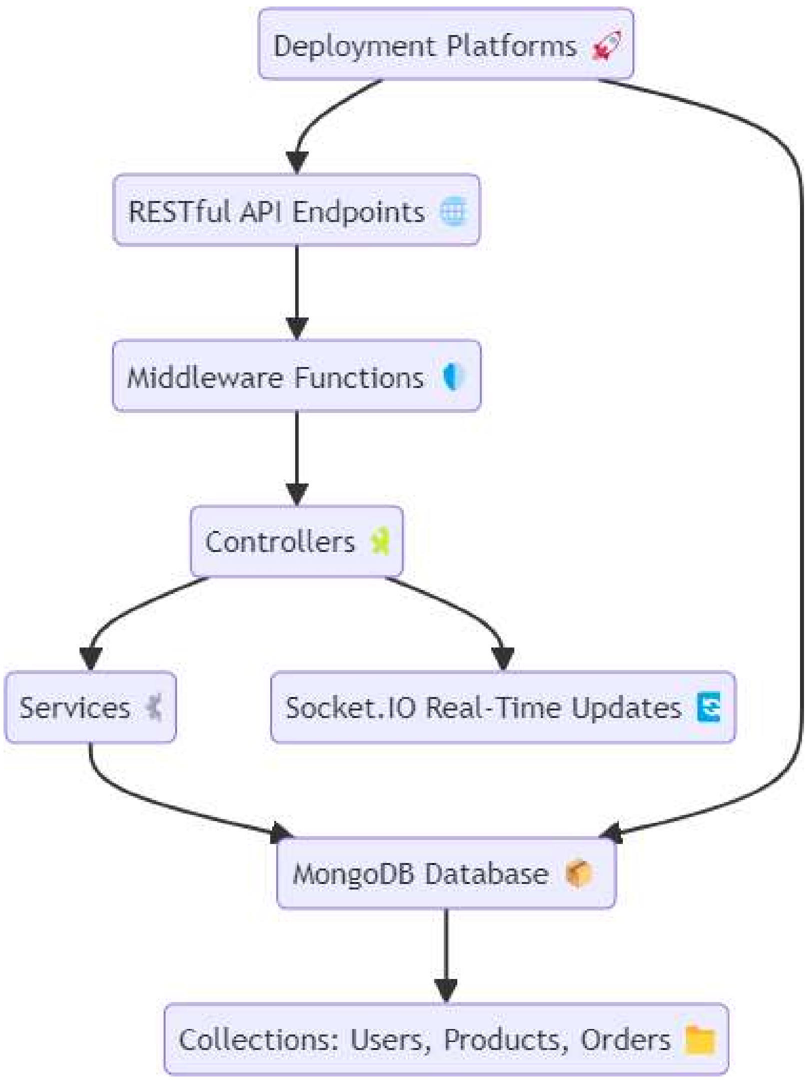
BACK END

The backend of the Grocery Web App is built using **Node.js** and **Express.js**, forming a strong and efficient server-side framework. It handles the core business logic, manages data processing, and facilitates seamless communication with the frontend to ensure a responsive and reliable user experience.

Key Features:

- **API-Driven Design**:
  RESTful APIs are designed for each functionality, enabling seamless interaction between the frontend and backend.
  - Example Endpoints:
    - POST /api/register for user registration.
    - GET /api/products for fetching all products.
    - POST /api/orders for placing an order.
- **Middleware**:
  - Middleware functions are used to handle tasks like authentication, request validation, and error handling.
  - Example: A JWT-based middleware ensures that only authenticated users can access protected routes.
- **Scalability**:
  - Modular structure with separate folders for routes, controllers, and services.
  - This separation of concerns makes it easy to scale the application by adding new features.

- **Real-Time Updates**:
  - Integrates **Socket.IO** for real-time notifications, such as order status changes or inventory updates



DATABASE

The database layer uses **MongoDB**, a NoSQL database known for its flexibility and scalability, to store and manage application data.

Key Features:

- **Schema Design**:
  MongoDB's flexible schema design allows for quick adjustments as new requirements emerge. The main collections include:
  - **Users**: Stores user profiles, authentication details, and order history.
  - **Products**: Stores product details, stock levels, and pricing.
  - **Orders**: Tracks order placement, status, and payment information.
- **Relationships**:
  - Logical relationships between collections (e.g., linking orders to users and products using ObjectIDs).
- **Indexing**:
  Indexes are created on fields like productName and category to optimize search queries.
- **Scalability**:
  - MongoDB's sharding feature can be leveraged to handle large datasets and concurrent users.

Integration and Communication

- **Frontend-Backend Communication**:
  - The frontend communicates with the backend through RESTful APIs.
  - Asynchronous requests are handled using **Axios** or **Fetch API**, ensuring non-blocking operations.
- **Backend-Database Communication**:
  - The backend uses **Mongoose**, an Object Data Modeling (ODM) library for MongoDB, to interact with the database.
  - Mongoose schemas enforce data consistency and validation at the application level.

Security

Security measures are implemented at every layer of the architecture to ensure the safety of user data and transactions:

- **Frontend**:
  - Input validation using React hooks and libraries like Formik or Yup.
- **Backend**:
  - JWT-based authentication and role-based access control.
  - Rate-limiting middleware to prevent brute force attacks.
- **Database**:
  - Data encryption for sensitive fields like passwords (hashed using bcrypt).

Deployment

The architecture supports smooth deployment to cloud platforms like **AWS**, **Heroku**, or **Vercel**. Key components:

- **Frontend**: Deployed on **Netlify** or **Vercel**.
- **Backend**: Deployed on **AWS EC2** or **Heroku**.

  Database: Hosted on **MongoDB Atlas**, a fully managed cloud database service

Technology Stack:

Layer	**Technology**
Frontend	React.js, Redux, Bootstrap
Backend	Node.js, Express.js
Database	MongoDB, Mongoose

State Management	Redux or Context API
Authentication	JWT
Deployment	AWS, Heroku, MongoDB Atlas

CHAPTER 4

SETUP INSTRUCTIONS

The following detailed setup instructions will guide you through the process of installing and running the Grocery Web App locally on your system.

PREREQUISITIES

1. **Node.js** (v14 or later):
   - Download and install from Node.js Official Website.
   - Verify installation by running:

     ```
     node -v

     npm -v
     ```

2. **MongoDB** (v4.4 or later):
   - Download and install MongoDB from MongoDB Official Website.
   - Start MongoDB service:

     ```
     mongodb
     ```

3. **Git**:
   - Download and install Git from Git Official Website.
   - Verify installation by running:

     ```
     git --version
     ```

4. **Code Editor**:
   - Use a code editor such as **Visual Studio Code** (recommended).

5. **Browser**:
   - A modern browser like Google Chrome or Mozilla Firefox.

Cloning the Repository

1. Open a terminal and navigate to the directory where you want to clone the project.
2. Clone the repository:

```
git clone https://github.com/kishan-kanth/NMprojects-grocery-webapp
```

3. Navigate to the project directory:

```
cd grocery-webapp
```

Environment Variables

1. Create a .env file in both the **backend** and **frontend** directories.
2. Add the following environment variables in the respective .env files:

Backend (/backend/.env):

```
MongoURI=mongodb+srv://kishankanthds:mongodbpass@cluster0
```

Frontend (/frontend/.env):

```
PORT=5000

LOCAL_HOST=  http://localhost:5173/
```

INSTALLATION

Install the required dependencies for both the client (frontend) and server (backend).

1. **Frontend**:

Navigate to the client directory:

```
cd frontend
```

Install dependencies using npm:

```
npm install
```

2. **Backend**:

Navigate to the server directory:

```
cd ../backend
```

Install dependencies using npm:

```
npm install
```

CHAPTER 5

RUNNING THE APPLICATION

1. **Start the MongoDB Service:**
    - Ensure the MongoDB service is running before starting the application.

```
mongodb
```

2. **Start the Backend Server**:

Navigate to the backend directory:

```
Cd backend
```

Start the backend server:

```
npm run start
```

- The backend server will run at http://localhost:5173.

3. **Start the Frontend Server**:

Open a new terminal and navigate to the client directory:

```
cd frontend
```

Start the frontend development server:

```
npm run dev
```

- The frontend will run at http://localhost:5173.

Verify the Setup

1. Open a browser and navigate to http://localhost:5173 to access the frontend of the Grocery Web App.
2. Test key functionalities:
    - User registration and login.
    - Product browsing and adding items to the cart.

Admin functionalities if applicable .

ADDITIONAL NOTES :

Data Seeding:
If the application requires initial data for products or users, run a seed script (if provided) in the backend directory:

```
npm run seed
```

1. **Using External APIs**:
   If the project integrates external APIs (e.g., payment gateways), ensure their keys are added to the `.env` file.
2. **Debugging**:
   - Use the terminal to monitor logs from the frontend and backend for troubleshooting errors.

Use browser developer tools for inspecting frontend issues

1. **Product Catalog:** Pre-load a diverse range of grocery products, categorized by type (e.g., fruits, vegetables, dairy, etc.), along with essential attributes like product name, description, price, SKU, and stock status. This ensures users have a rich selection when they first interact with the app.
2. **User Accounts:** Seed sample user data for testing purposes, including user profiles, account details, and order history. This helps ensure that user authentication and account management features are functioning correctly.
3. **Promotions and Offers:** Include default promotional offers, discounts, and loyalty points information so users can access deals immediately upon signup. This is essential for testing notification systems and discount applications.
4. **Categories and Subcategories:** Populate predefined categories (e.g., fruits, vegetables, snacks) with relevant products. This allows users to filter and search effectively from day one.
5. **Stock Levels and Availability:** Seed data reflecting initial stock levels for products, enabling real-time updates for users and testing the stock tracking and notification system.
6. **Payment Methods:** Configure a set of sample payment methods and payment gateway test data for the initial stages of development and testing.

CHAPTER 6

API DOCUMENTATION

This document outlines the key API endpoints for the **Grocery Web App**, which allows users to browse products, manage their cart, place orders, and track delivery. The API is RESTful, with JSON responses.

Base URL

arduino
https://api.groceryapp.com/v1

Authentication

All requests require an authentication token. The token is provided upon successful login and should be included in the Authorization header for every request.

Authorization Header

makefile
Authorization: Bearer <access_token>

Endpoints

1. User Authentication

POST /auth/signup

- **Description:** Create a new user account.
- **Request Body:**

```
json
{
  "email": "user@example.com",
  "password": "securepassword",
  "name": "Kishan"
}
```

- **Response:**

```
json
{
  "message": "User created successfully",
  "user_id": "2730",
  "token": "<access_token>"
}
```

POST /auth/login

- **Description:** Login and retrieve authentication token.
- **Request Body:**

```
json
{
  "email": "user@example.com",
  "password": "securepassword"
}
```

- **Response:**

```
json
{
  "message": "Login successful",
  "token": "<access_token>"
}
```

POST /auth/logout

- **Description:** Logout the current user.
- **Response:**

```
json
{
  "message": "Logout successful"
}
```

2. Product Management

GET /products

- **Description:** Retrieve a list of all products.
- **Query Parameters:**
  - category (optional) – Filter products by category.
  - search (optional) – Search products by name or description.
- **Response:**

```
json
 [
  {
    "id": "101",
    "name": "Apple",
    "category": "Fruits",
    "price": 1.99,
    "stock": 50,
    "image_url": "https://example.com/apple.jpg"
  },
  {
    "id": "102",
    "name": "Banana",
    "category": "Fruits",
    "price": 1.49,
    "stock": 20,
    "image_url": "https://example.com/banana.jpg"
  }
]
```

GET /products/{id}

- **Description:** Get details of a specific product.
- **Response:**

```
json
{
  "id": "101",
  "name": "Apple",
  "category": "Fruits",
  "price": 1.99,
  "stock": 50,
  "description": "Fresh red apples from local farms.",
  "image_url": "https://example.com/apple.jpg"
}
```

GET /categories

- **Description:** Retrieve all product categories.
- **Response:**

  json
  ```
   [
    "Fruits",
    "Vegetables",
    "Dairy",
    "Snacks",
    "Beverages"
  ]
  ```

3. Shopping Cart

GET /cart

- **Description:** Retrieve the current user's shopping cart.
- **Response:**

  json
  ```
  {
    "items": [
      {
        "product_id": "101",
        "name": "Apple",
        "quantity": 3,
        "price": 1.99
      },
      {
        "product_id": "102",
        "name": "Banana",
        "quantity": 2,
  ```

```
      "price": 1.49
    }
  ],
  "total_price": 9.94
}
```

POST /cart

- **Description:** Add a product to the cart.
- **Request Body:**

```
json
{
  "product_id": "101",
  "quantity": 2
}
```

- **Response:**

```
json
{
  "message": "Product added to cart",
  "cart": {
    "items": [
      {
        "product_id": "101",
        "name": "Apple",
        "quantity": 2,
        "price": 1.99
      }
    ],
    "total_price": 3.98
  }
}
```

DELETE /cart/{product_id}

- **Description:** Remove a product from the cart.
- **Response:**

```
json
```

```
{
 "message": "Product removed from cart",
 "cart": {
  "items": [
   {
    "product_id": "102",
    "name": "Banana",
    "quantity": 2,
    "price": 1.49
   }
  ],
  "total_price": 2.98
 }
}
```

4. Order Management

POST /orders

- **Description:** Place a new order.
- **Request Body:**

  ```
  json
  {
   "address": "123 Main St, City, Country",
   "payment_method": "credit_card",
   "cart_id": "abc123"
  }
  ```

- **Response:**

  ```
  json
  {
   "message": "Order placed successfully",
   "order_id": "order123",
   "status": "pending"
  }
  ```

GET /orders/{order_id}

- **Description:** Retrieve order details.
- **Response:**

```
json
{
  "order_id": "order123",
  "status": "pending",
  "items": [
    {
      "product_id": "101",
      "name": "Apple",
      "quantity": 3,
      "price": 1.99
    }
  ],
  "total_price": 5.97,
  "shipping_address": "123 Main St, City, Country"
}
```

GET /orders

- **Description:** Retrieve all orders for the authenticated user.
- **Response:**

```
json
 [
  {
    "order_id": "order123",
    "status": "shipped",
    "total_price": 9.94,
    "date": "2024-11-24T14:30:00Z"
  }
]
```

5. Stock and Notification System

GET /stock/{product_id}

- **Description:** Get real-time stock availability for a product.
- **Response:**

```
json
{
  "product_id": "101",
  "stock": 50
}
```

POST /notifications

- **Description:** Subscribe to notifications for deals, promotions, or stock updates.
- **Request Body:**

```
json
{
  "product_id": "101",
  "notification_type": "stock_update"
}
```

- **Response:**

```
json
{
  "message": "Subscribed to stock updates",
  "subscription_id": "sub123"
}
```

Error Handling

All API responses include an error code and message in the event of a failure.

Example Error Response:

```
json
{
  "error": {
    "code": "400",
    "message": "Product not found"
  }
}
```

Rate Limiting

- API requests are rate-limited to prevent abuse. The limits are as follows:
  - **500 requests per minute** per user
  - If exceeded, you will receive a 429 Too Many Requests error.

CHAPTER 7

AUTHENTICATION & AUTHORIZATION

Authentication is a critical feature in the Grocery Web App as it ensures secure and personalized access to the platform. Here's a detailed breakdown of the authentication process:

1. **User Registration (Sign-Up):**
    - o Users are required to create an account with personal details such as name, email, phone number, and a secure password.
    - o Validation checks should be in place to ensure the email is unique and the password meets security requirements (e.g., minimum length, special characters).
    - o An email verification process can be implemented to ensure the legitimacy of the user's email address.
2. **Login:**
    - o Users can log in using their registered email and password.
    - o A "Remember Me" feature can be added for a more convenient login experience on returning visits.
    - o Multi-Factor Authentication (MFA) can be integrated for enhanced security, prompting users for an additional verification step (e.g., SMS code or email link).
3. **Password Recovery:**
    - o A "Forgot Password" option allows users to reset their password through a secure process, typically involving sending a password reset link to the registered email.
    - o Security questions or additional identity verification can be added to ensure the user is the rightful account holder.
4. **Session Management:**
    - o After successful login, users are assigned a session token, stored securely (e.g., in cookies or local storage), allowing them to remain logged in while browsing the site.
    - o Session expiry should be managed to automatically log out inactive users, improving security.
5. **Profile Management:**
    - o Users can update their personal details, such as name, address, phone number, and payment information through their account settings.

    - Password changes can also be handled through this section with re-authentication for security.
6. **Role-Based Authentication:**
    - Different user roles, such as **Admin**, **Customer**, or **Seller**, can be defined with varying levels of access.
        - **Admins** manage the backend, approve listings, handle inventory, and monitor orders.
        - **Customers** can browse products, place orders, and view order history.
        - **Sellers** can add products, track orders, and manage their inventory.
7. **Social Media Login:**
    - Integrating options like Google, Facebook, or other OAuth-based login methods can provide an easier and faster authentication process for users.
8. **Security Measures:**
    - Use encryption protocols like HTTPS to ensure the security of user credentials and sensitive information.
    - Implement rate limiting, CAPTCHA, and other mechanisms to prevent brute-force attacks and unauthorized access.

CHAPTER 8

TESTING

1. Unit Testing with Jest (for Cart Functionality)

cart.js (Functionality)

```
// cart.js

let cart = [];


const addToCart = (product) => {

  cart.push(product);

};


const getTotalPrice = () => {

  return cart.reduce((total, item) => total + item.price, 0);

};


const getCart = () => cart;


module.exports = { addToCart, getTotalPrice, getCart };
```

Running Unit Tests with Jest

1. Install jest

```
npm install --save-dev jest
```

2. Run the test

```
npm test
```

2. End-to-End Testing with Cypress (for Checkout Process)

```
// checkout.spec.js

describe('Grocery Web App - Checkout Flow', () => {

  beforeEach(() => {

    cy.visit('http://localhost:5173');  // Make sure the app is running locally

  });


  it('should allow a user to add an item to the cart and proceed to checkout', () => {

    // Add an item to the cart

    cy.get('.product').first().click(); // Assume the first product on the list

    cy.get('.add-to-cart').click();  // Button to add to cart


    // Verify that the cart shows the added product

    cy.get('.cart').should('contain', 'Apple'); // Example product name


    // Proceed to checkout

    cy.get('.checkout-button').click();  // Checkout button click


    // Fill in checkout details (mock data)

    cy.get('.checkout-form input[name="name"]').type('John Doe');

    cy.get('.checkout-form input[name="address"]').type('123 Main St');

    cy.get('.checkout-form input[name="payment"]').type('4111111111111111'); //
Test credit card number
```

```
    // Submit checkout form
    cy.get('.checkout-form     button[type="submit"]').click();

    // Verify the order confirmation page
    cy.url().should('include', '/order-confirmation');
    cy.get('.order-summary').should('contain', 'Thank you for your purchase');
  });
});
```

3. Security Testing: Checking for SQL Injection Vulnerability

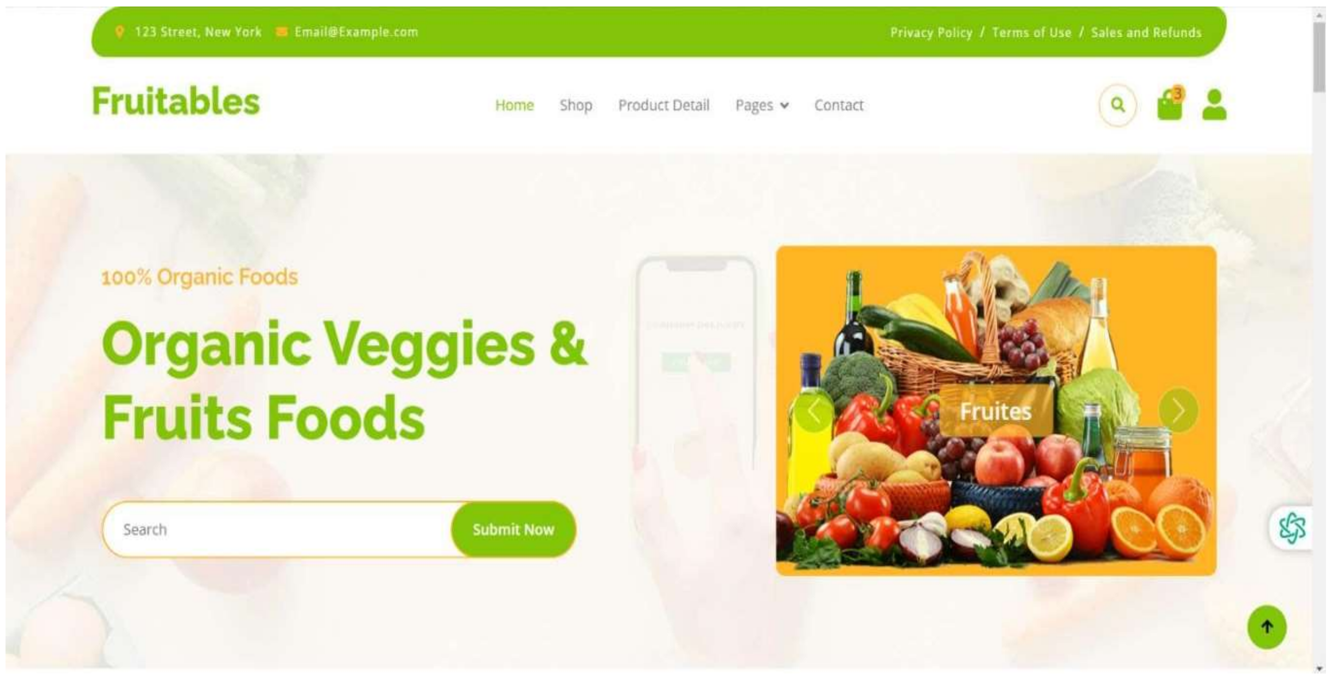
```
const { queryDatabase } = require('./db');  // Simulate a database query function

test('should not allow SQL injection in queries', async () => {
  const maliciousInput = "1 OR 1=1";  // SQL Injection attempt
  const result = await queryDatabase(`SELECT * FROM products WHERE id =
'${maliciousInput}'`);
  expect(result).toBeNull();  // Ensure no results or proper handling of malicious
input
});
```

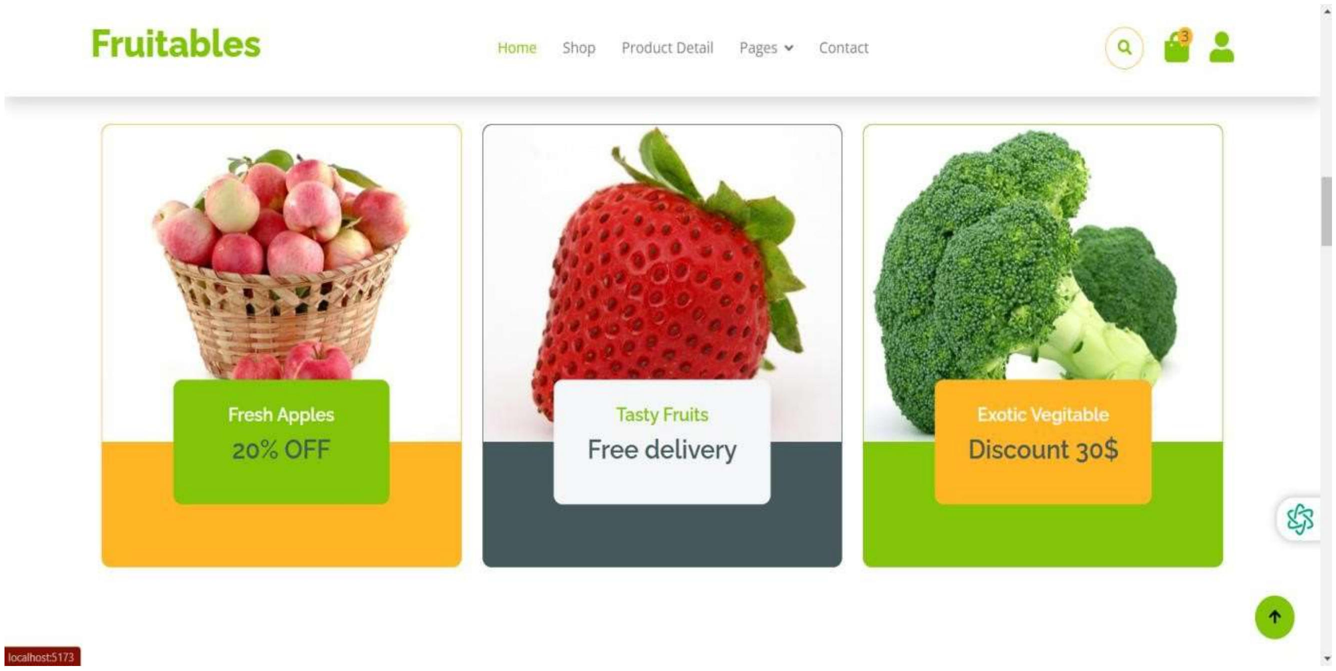
4. Performance Testing with Lighthouse (CLI)

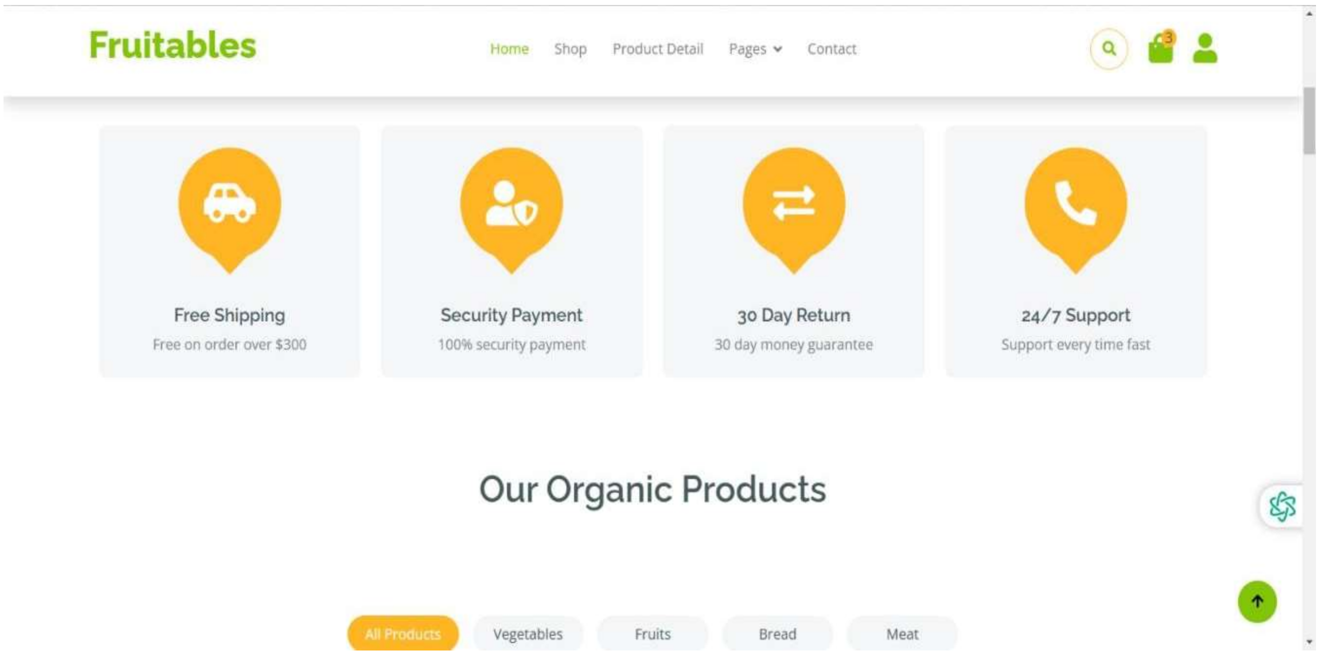
```
npx lighthouse http://localhost:5173 --output html --output-path ./lighthouse-
report.html
```

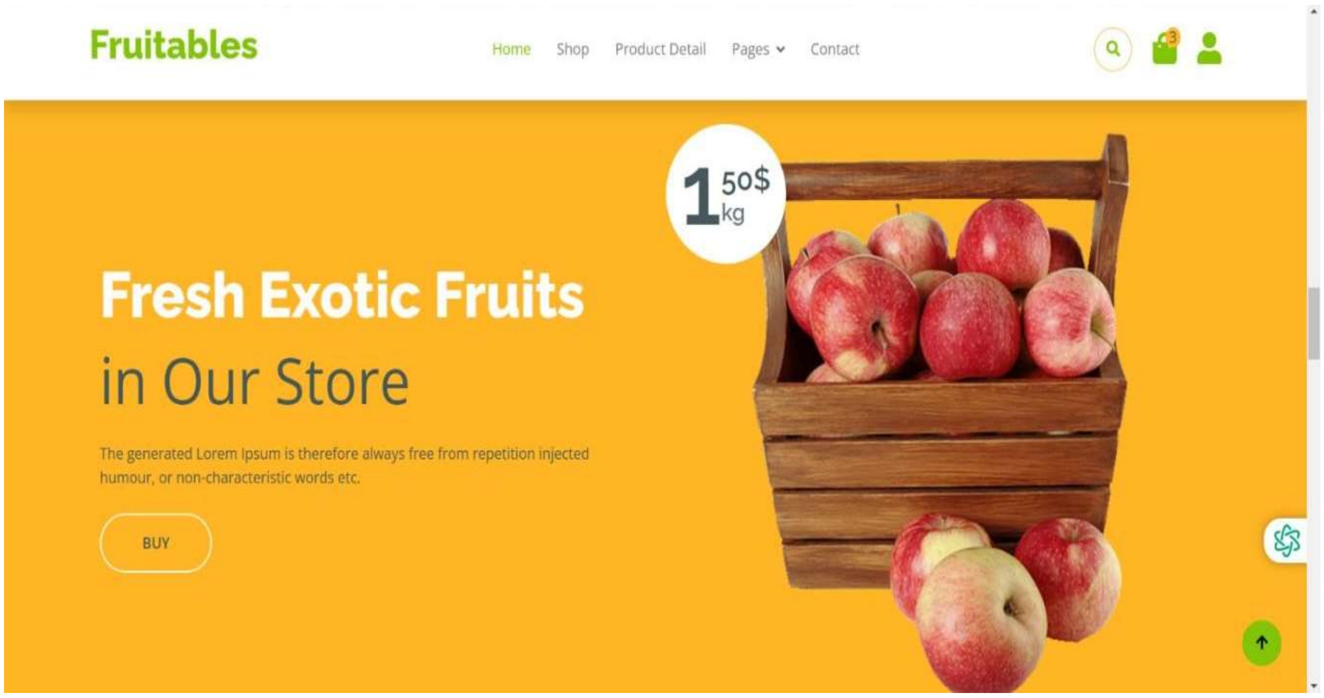
CHAPTER 9

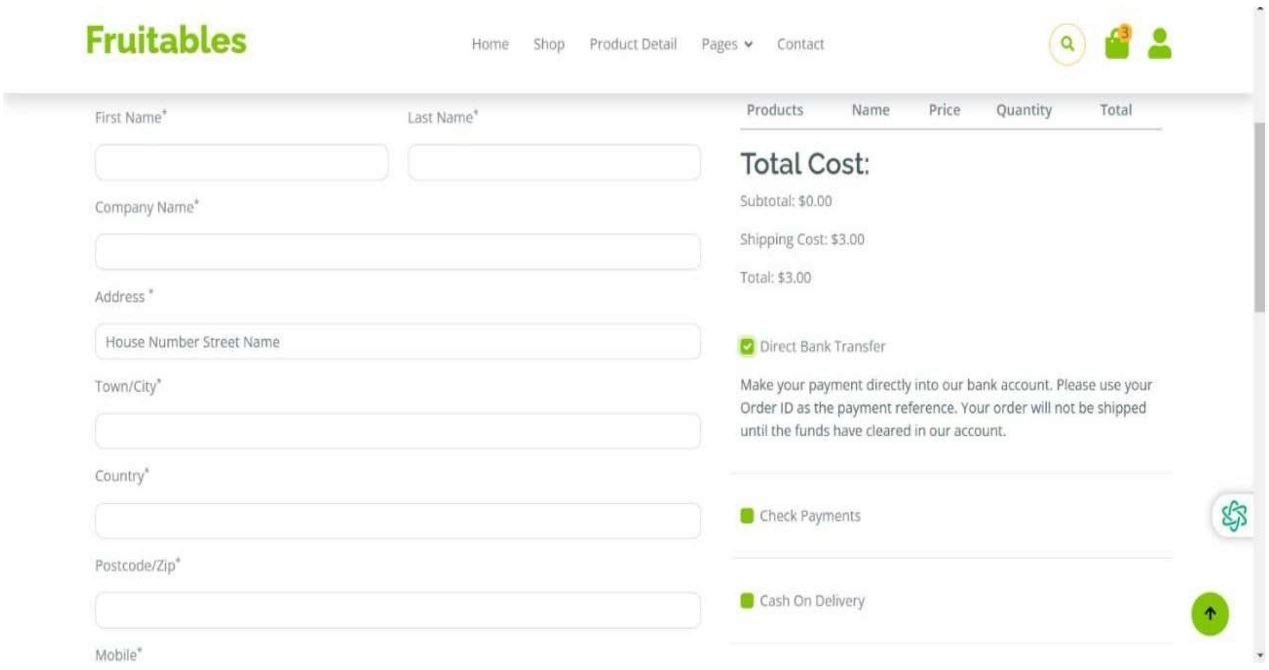
SCREENSHOTS






Fruitables
Home
Shop
Product Detail
Pages
Contact
Free Shipping
Free on order over $300
Security Payment
100% security payment
30 Day Return
30 day money guarantee
24/7 Support
Support every time fast
Our Organic Products
All Products
Vegetables
Fruits
Bread
Meat


Fruitables
Home
Shop
Product Detail
Pages
Contact
1 50$ kg
Fresh Exotic Fruits
in Our Store
The generated Lorem Ipsum is therefore always free from repetition injected humour, or non-characteristic words etc.
BUY



Fruitables

First Name*

Last Name*

Company Name*

Address *

House Number Street Name

Town/City*

Country*

Postcode/Zip*

Mobile*

Products	Name	Price	Quantity	Total

Total Cost:

Subtotal: $0.00

Shipping Cost: $3.00

Total: $3.00

[✓ Direct Bank Transfer]

Make your payment directly into our bank account. Please use your Order ID as the payment reference. Your order will not be shipped until the funds have cleared in our account.

Check Payments

Cash On Delivery

DEMO:

https://drive.google.com/file/d/1c999dNIjIQYMkYle6xf1euZ9kld8crtY/view?usp=drive_link

CHAPTER 10

KNOWN ISSUES

While developing the Grocery Web App, there are several potential or known issues that might arise. These can range from UI glitches to functionality problems. Below are common issues and challenges that may need to be addressed during development, testing, or post-launch.

1. Slow Page Load Times

- **Issue:** The web app may experience slow loading times due to heavy images, inefficient JavaScript, or unoptimized assets.
- **Possible Causes:**
  - Large image sizes or uncompressed assets.
  - JavaScript that is not optimized for performance.
  - Lack of lazy loading for images or asynchronous loading for resources.
- **Solution:**
  - Use image compression and optimize assets for the web.
  - Implement lazy loading for images and resources.
  - Leverage browser caching and content delivery networks (CDNs).

2. Authentication Errors

- **Issue:** Users may face login or sign-up issues, especially if credentials are not correctly validated or there is a session timeout issue.
- **Possible Causes:**
  - Improper session management or token expiration.
  - Errors in the login API or user authentication flow.
- **Solution:**
  - Ensure session tokens are securely stored and have appropriate expiration times.
  - Implement multi-factor authentication (MFA) if necessary.
  - Provide clear error messages for login failures and password resets.

3. Cart Synchronization Across Devices

- **Issue:** The cart may not be synced correctly across different devices or browsers.

- **Possible Causes:**
  - Lack of real-time data sync or improper session management.
  - Cart data is not saved in the cloud or is stored locally only.
- **Solution:**
  - Use cloud storage (e.g., Firebase, AWS) to save cart data and sync it across devices.
  - Implement WebSockets or other real-time synchronization methods to keep the cart updated across sessions.

4. Payment Gateway Failures

- **Issue:** Users may face errors during checkout, such as failed payment transactions or incorrect billing details.
- **Possible Causes:**
  - Misconfigured payment gateway API.
  - Invalid data being sent (e.g., incorrect card details or expired token).
- **Solution:**
  - Test payment APIs thoroughly in sandbox mode before production.
  - Handle errors gracefully with user-friendly messages and fallback options (e.g., alternative payment methods).

5. Order Status Updates Delay

- **Issue:** There might be delays in order status updates (e.g., "Shipped" or "Delivered").
- **Possible Causes:**
  - Backend services might not update the status immediately after processing.
  - Integration issues between the front-end and backend systems.
- **Solution:**
  - Set up real-time updates for order statuses.
  - Implement background jobs to track order status and send notifications.

6. Stock Availability Issues

- **Issue:** The app may show incorrect stock availability, leading to errors when users attempt to purchase out-of-stock items.
- **Possible Causes:**

    - Incorrect stock data in the database or errors in syncing between the frontend and backend.
    - Lack of proper inventory management.
- **Solution:**
    - Implement real-time stock checks and update inventory status frequently.
    - Show accurate product availability on the product page, and notify users if stock runs low.

7. Broken Links or Navigation Issues

- **Issue:** Some links or navigation buttons may not work, leading to dead ends or broken pages.
- **Possible Causes:**
    - Incorrect URL routing or outdated links.
    - Missing pages or resources on the server.
- **Solution:**
    - Use proper URL routing with a framework like React Router or Vue Router.
    - Implement 404 pages with helpful navigation suggestions when users hit broken links.

8. User Interface (UI) Bugs

- **Issue:** The UI might not be responsive, with elements overlapping or misaligned, particularly on mobile devices.
- **Possible Causes:**
    - Inconsistent CSS styles or media queries not covering all screen sizes.
    - Overuse of fixed-width elements or poorly implemented responsive design.
- **Solution:**
    - Use CSS frameworks like Bootstrap or Tailwind CSS for better responsiveness.
    - Test the app across various devices and screen sizes to identify and fix layout issues.

CHAPTER 11

FUTURE ENHANCEMENT

1. Personalized Recommendations

- **Enhancement:** Implement a recommendation engine that suggests products based on user behavior, preferences, or purchase history.
- **Benefit:** Increases customer engagement by providing tailored product suggestions, boosting sales and user satisfaction.
- **Technology:** Machine learning algorithms, collaborative filtering, and recommendation systems.

2. Subscription and Auto-Replenishment Service

- **Enhancement:** Allow users to subscribe to regular deliveries for frequently purchased items (e.g., milk, bread).
- **Benefit:** Encourages repeat business and improves convenience for users who need regular restocks.
- **Technology:** Subscription management system, recurring billing, and automated notifications.

3. Advanced Search Functionality

- **Enhancement:** Implement an advanced search with filters like price range, brand, dietary preferences (e.g., gluten-free), and delivery time slots.
- **Benefit:** Enhances user experience by making it easier to find specific products, improving overall shopping efficiency.
- **Technology:** ElasticSearch or Algolia for fast and customizable search.

4. Mobile App Development

- **Enhancement:** Launch a mobile app for iOS and Android to provide a more optimized, native experience for users.
- **Benefit:** Mobile apps offer a faster, more responsive interface, and are more accessible for on-the-go users.
- **Technology:** React Native or Flutter for cross-platform mobile app development.

5. **Voice Command Integration**

- **Enhancement:** Integrate voice recognition for hands-free shopping. Users can add items to their cart or search for products via voice commands.
- **Benefit:** Improves accessibility, convenience, and user experience, particularly for visually impaired users.
- **Technology:** Voice assistants like Google Assistant, Amazon Alexa, or built-in speech-to-text APIs.

6. **AI-Powered Customer Support (Chatbot)**

- **Enhancement:** Implement an AI-driven chatbot to assist customers in real-time with order tracking, product information, and general inquiries.
- **Benefit:** Provides 24/7 support and reduces the load on human customer service representatives.
- **Technology:** Natural Language Processing (NLP) with tools like Dialogflow, Microsoft Bot Framework, or IBM Watson.

7. **Dynamic Pricing and Discounts**

- **Enhancement:** Implement dynamic pricing where prices change based on demand, stock levels, or promotional periods (e.g., flash sales).
- **Benefit:** Maximizes revenue by adjusting prices based on real-time market conditions and customer demand.
- **Technology:** Pricing algorithms and integration with backend systems for real-time price updates.

8. **Augmented Reality (AR) for Product Visualization**

- **Enhancement:** Integrate AR technology to allow customers to visualize certain products (e.g., kitchen items, or grocery packaging) in their own environment.
- **Benefit:** Helps users make better purchase decisions by seeing how products will look in real life before buying.
- **Technology:** AR libraries such as ARKit (iOS), ARCore (Android), or WebXR for web-based AR.

9. Delivery Scheduling and Real-Time Tracking

- **Enhancement:** Allow users to schedule deliveries at their preferred time slots and provide real-time tracking for deliveries.
- **Benefit:** Increases customer satisfaction by offering more control over delivery timing and improving transparency.
- **Technology:** GPS and real-time tracking systems integrated with a delivery management platform.

10. Blockchain for Transparent Supply Chain

- **Enhancement:** Utilize blockchain to track the entire supply chain of products (from farm to table), providing transparency to customers about the origin and quality of the products.
- **Benefit:** Builds trust with customers, especially those concerned about product sourcing, sustainability, and ethical practices.
- **Technology:** Blockchain platforms like Ethereum or Hyperledger for supply chain management.