

# Full Stack Development with MERN

## Project Documentation

---

### 1. Introduction

#### Project Title:

##### **Shop Smart - Your Digital Grocery Store Experience**

Shop Smart is a revolutionary digital grocery shopping platform designed to bridge the gap between traditional grocery shopping and modern e-commerce convenience. The platform caters to both individual consumers and businesses, offering a seamless, efficient, and enjoyable shopping experience. By leveraging advanced technologies such as AI, machine learning, and cloud computing, Shop Smart aims to transform the way people buy groceries, making the process faster, safer, and more personalized.

#### Team Members:

- |                                   |            |
|-----------------------------------|------------|
| • <b>Mohammed Arshadulla khan</b> | 218X1A0468 |
| • <b>Shaik Abdul Khaim -</b>      | 218X1A04D1 |
| • <b>Pittu Narendra Reddy -</b>   | 218X1A04C7 |
| • <b>Shaik Yaseen</b>             | 218X1A04A5 |
| • <b>Shaik Soheb –</b>            | 218X1A04A4 |

#### Project Scope:

Shop Smart is designed to cater to a wide range of users, from individual consumers to businesses that require bulk purchases. The platform offers a comprehensive set of features, including product browsing, secure payments, order tracking, and personalized recommendations. Additionally, the platform includes an admin dashboard that allows businesses to manage inventory, track sales, and analyze customer behavior. The project scope also includes future enhancements such as AI-powered recommendations, IoT integration, and third-party API integrations.

#### Objectives:

The primary objectives of the Shop Smart project are:

- **User-Friendly Experience:** Provide an intuitive and easy-to-navigate interface that caters to users of all technical levels.
- **Secure Transactions:** Ensure that all payment transactions are secure and encrypted, protecting user data from potential threats.
- **Efficient Order Management:** Enable admins to manage inventory, track orders, and analyze sales data efficiently, ensuring smooth operations.
- **Personalization:** Use AI-powered recommendations to enhance customer engagement and satisfaction by offering personalized deals and product suggestions.

- **Scalability:** Build a robust system capable of handling high user traffic and future growth, ensuring that the platform remains reliable and performant as the user base expands.
- 

## 2. Project Overview

### Purpose:

The primary purpose of Shop Smart is to provide a highly efficient and convenient online grocery shopping experience. The platform is designed to cater to the needs of modern consumers who value speed, convenience, and security in their shopping experience. By offering a wide range of features, including product browsing, secure payments, and real-time order tracking, Shop Smart aims to make grocery shopping as seamless as possible.

### Features:

#### User Features:

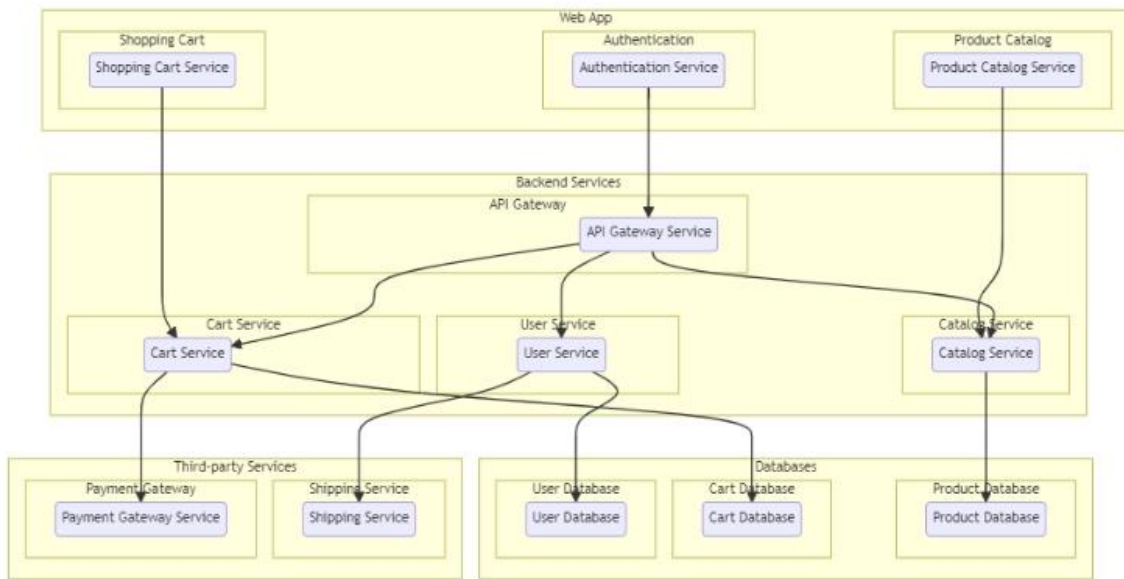
- **User Authentication and Authorization:** Shop Smart uses JWT (JSON Web Tokens) for secure user authentication. This ensures that user data is protected and only authorized users can access their accounts.
- **Product Browsing and Search:** The platform offers advanced filters and AI-powered recommendations, making it easy for users to find the products they need quickly.
- **Shopping Cart and Order Placement:** The shopping cart feature allows users to add, remove, and manage items with ease. The checkout process is streamlined, ensuring a smooth user experience.
- **Secure Payment Integration:** Shop Smart supports multiple payment options, including credit/debit cards, PayPal, and UPI, ensuring that users can choose the payment method that suits them best.
- **Order History and Tracking:** Users can view their order history and track the status of their current orders in real-time, providing transparency and peace of mind.
- **Subscription Model:** Users can subscribe to recurring grocery deliveries, ensuring that they never run out of essential items.
- **Wishlist Feature:** The wishlist feature allows users to save items for future purchases, making it easy to keep track of products they are interested in.
- **Push Notifications:** Users receive real-time updates on promotions, order status, and personalized deals, ensuring that they are always informed.

#### Admin Features:

- **Inventory Management:** Admins can track product availability, update stock levels, and manage suppliers, ensuring that the platform always has the products users need.
- **Order Management:** Admins can monitor order status, process refunds, and handle customer queries, ensuring that the platform operates smoothly.

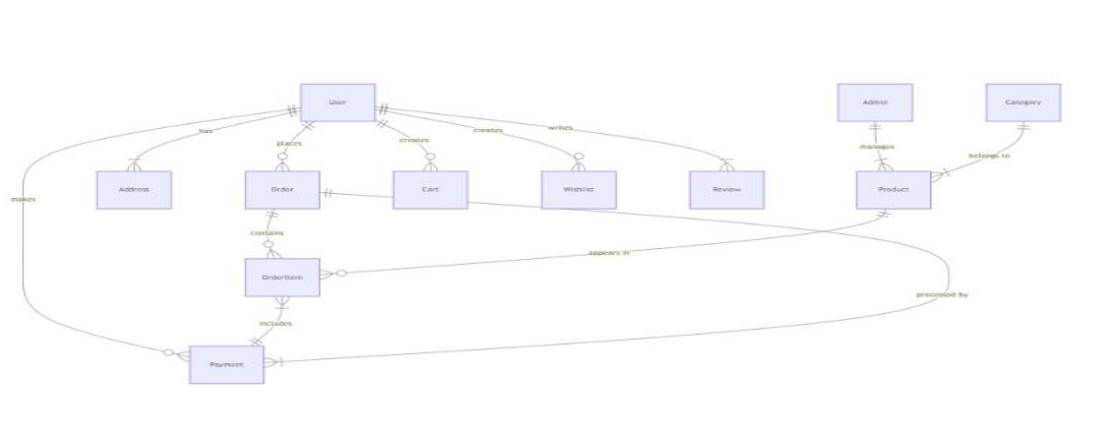
- **Promotional Campaigns:** Admins can create and manage discount offers, loyalty programs, and other promotional campaigns, helping to attract and retain customers.
- **Admin Dashboard Analytics:** The admin dashboard provides detailed sales reports, customer insights, and predictive analytics, helping admins make data-driven decisions.

### 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.

## Frontend:

The frontend of Shop Smart is built using **React.js**, a popular JavaScript library for building user interfaces. React's component-based architecture ensures that the UI is modular, scalable, and easy to maintain. The frontend also uses **Redux** for state management, ensuring that the application's state is consistent and predictable across different components. For styling, the platform uses **Material UI** and **Tailwind CSS**, which provide a responsive and elegant design.

## Backend:

### *Set Up Project Structure:*

- Create a new directory for your project and set up a package.json file using npm init command.
- Install necessary dependencies such as Express.js, Mongoose, and other required packages.

### *Database Configuration:*

- Set up a MongoDB database either locally or using a cloud-based MongoDB service like MongoDB Atlas.
- Create a database and define the necessary collections for hotels, users, bookings, and other relevant data.

### *Create Express.js Server:*

- Set up an Express.js server to handle HTTP requests and serve API endpoints.
- Configure middleware such as body-parser for parsing request bodies and cors for handling cross-origin requests.

### *Define API Routes:*

- Create separate route files for different API functionalities such as hotels, users, bookings, and authentication.
- Define the necessary routes for listing hotels, handling user registration and login, managing bookings, etc.
- Implement route handlers using Express.js to handle requests and interact with the database.

### *Implement Data Models:*

- Define Mongoose schemas for the different data entities like hotels, users, and bookings.
- Create corresponding Mongoose models to interact with the MongoDB database.
- Implement CRUD operations (Create, Read, Update, Delete) for each model to perform database operations.

#### *API Design and Development:*

- Identify the necessary functionality and data required by the frontend.
- Design a set of RESTful APIs using a framework like Express.js or Django REST Framework.
- Define API endpoints for user management, product catalog, shopping cart, order management, payment gateway integration, shipping integration, etc.
- Implement the API routes, controllers, and data models to handle the corresponding operations.
- Ensure that the APIs follow best practices, are secure, and provide appropriate responses.

#### *User Management and Authentication:*

- Implement user registration and login functionality.
- Choose an authentication mechanism like session-based authentication or token-based authentication (e.g., JWT).
- Store and hash user credentials securely.
- Implement middleware to authenticate API requests and authorize access to protected routes.

#### *Product Catalog and Inventory Management:*

- Design the database schema to store product details, pricing, availability, and inventory levels.
- Create APIs to retrieve product information, update inventory quantities, and handle search and filtering.
- Implement validations to ensure data integrity and consistency.

#### *Shopping Cart and Order Management:*

- Design the database schema to store shopping cart details and order information.
- Create APIs to handle cart operations like adding items, modifying quantities, and placing orders.
- Implement logic to calculate totals, apply discounts, and manage the order lifecycle.

#### *Payment Gateway Integration:*

- Choose a suitable payment gateway provider (e.g., Stripe, COD).

- Integrate the payment gateway SDK or API to handle secure payment processing.
- Implement APIs or callback endpoints to initiate transactions, handle payment callbacks, and receive payment confirmation.

#### **Shipping and Logistics Integration:**

- Identify shipping and logistics providers that align with your application's requirements.
- Utilize the APIs provided by these providers to calculate shipping costs, generate shipping labels, and track shipments.
- Implement APIs or services to fetch rates, generate labels, and obtain tracking information.

#### **Database Integration:**

- Choose a suitable database technology (e.g., MySQL, PostgreSQL, MongoDB) based on your application's requirements.
- Design the database schema to efficiently store and retrieve flower and gift delivery data.
- Establish a connection to the database and handle data persistence and retrieval.

#### **External Service Integration:**

- Identify third-party services like email service providers, analytics services, or CRM systems that are required for your application.
- Utilize the APIs or SDKs provided by these services to exchange data and perform necessary operations.
- Implement the integration logic to send order confirmations, track user behavior, or manage customer relationships.

#### **Security and Data Protection:**

- Apply appropriate security measures like encryption techniques for secure data transmission and storage.
- Implement input validation and sanitization to prevent common security vulnerabilities.
- Implement access control to ensure authorized access to sensitive data.

#### **Error Handling and Logging:**

- Implement error handling mechanisms to handle exceptions and provide meaningful error messages to the frontend.
- Use logging frameworks to record application logs for monitoring and troubleshooting purposes.

The backend of Shop Smart is built using **Node.js** with **Express.js**, a robust framework for handling API requests. The backend follows a **RESTful API architecture**, ensuring efficient communication between the frontend and backend. **JWT authentication** is used to manage

secure user sessions, and middleware functions are implemented for security, logging, and error handling.

### Database:

The platform uses **MongoDB**, a NoSQL database, to store user details, products, orders, and transactions. MongoDB's flexible schema makes it ideal for handling diverse data types. **Mongoose ORM** is used to simplify database interactions, and **Redis caching** is implemented to optimize product searches and reduce server load.

### Deployment:

Shop Smart is deployed using **cloud-based hosting** services such as AWS, Firebase, or Heroku, ensuring high availability and scalability. **Docker containerization** is used for scalable deployment, and **CI/CD pipelines** are set up for automated testing and deployment. **Load balancing** is implemented to handle high traffic and ensure smooth performance.

---

## 4. Setup Instructions

### 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/>

Before setting up the project, ensure that the following software is installed:

- **Node.js** (Latest LTS version)
- **MongoDB Community Server**
- **npm or yarn**
- **Postman** (for API testing)
- **Docker** (for containerized deployment)

## **Installation Steps:**

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/Bharath136/grocery-webapp>

### **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.

**Video Tutorial Link to clone the project:**

- <https://drive.google.com/file/d/1KTGK0XZj0XWOiDeNKJVRKQHXLyVWZYLM/view?usp=sharing>

**Project Repository Link:** <https://github.com/Bharath136/grocery-webapp>

Congratulations! You have successfully installed and set up the grocery-webapp app on your local machine. You can now proceed with further customization, development, and testing as needed.

**1. Clone the Repository:**

bash

Copy

```
git clone [repository_url]
```

**2. Navigate to the Project Directory:**

bash

Copy

```
cd grocery-web-app
```

**3. Install Dependencies:**

- For the frontend:

bash

Copy

```
cd client && npm install
```

- For the backend:

bash

Copy

```
cd ../server && npm install
```

**4. Set Up Environment Variables:**

Create a .env file in the root directory and add the necessary environment variables (e.g., database connection strings, API keys).

**5. Start MongoDB:**

Start the MongoDB server locally or use a cloud database like MongoDB Atlas.

**6. Run the Application:**

- For the frontend:

bash

Copy

```
npm start
```

- For the backend:

bash

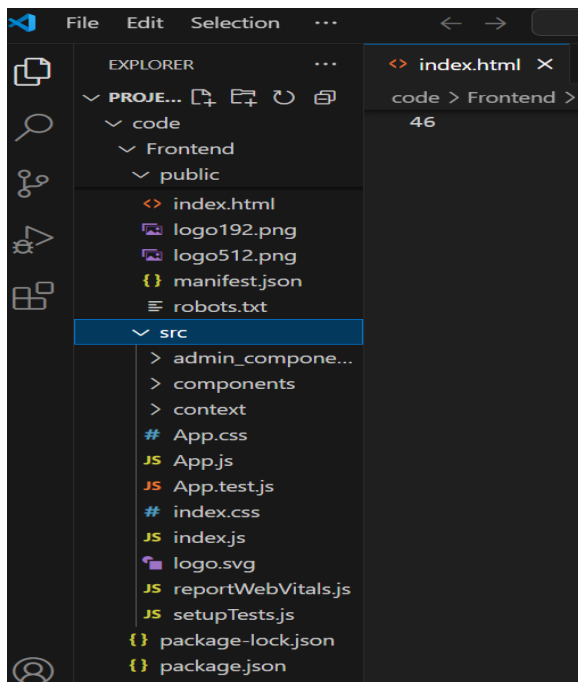
Copy

npm start

---

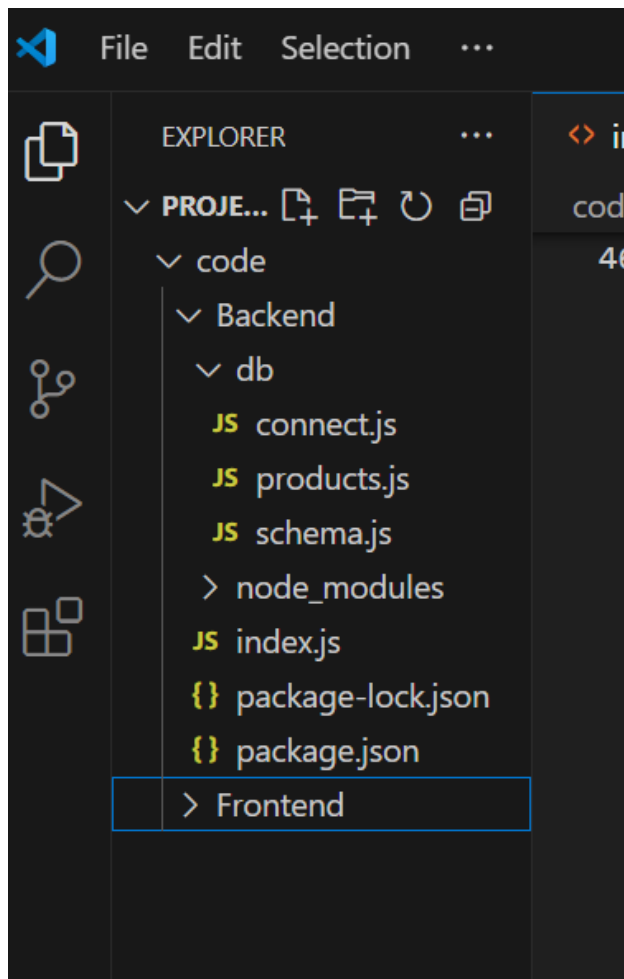
## 5. Folder Structure

Client:



- **src/components/**: Contains reusable UI components such as buttons, modals, and cards.
- **src/pages/**: Defines different pages of the application, including Home, Login, Checkout, and Product details.
- **src/redux/**: Handles state management using Redux, ensuring a global state for authentication, products, and cart.
- **src/assets/**: Stores static images, icons, and stylesheets for branding and UI design.
- **src/context/**: Contains React context providers to manage global states such as user authentication and theme settings.
- **src/components/products/**: Manages product-related components like Product List, ProductItem, and ProductDetails.

Server:



- **routes/**: Defines API endpoints for user authentication, product management, orders, and admin controls.
- **models/**: Contains Mongoose schemas defining the database structure for Users, Products, Orders, and Payments.
- **controllers/**: Implements business logic for handling requests and responses, including authentication and order processing.
- **middlewares/**: Houses authentication and error-handling middleware functions for securing API routes.
- **utils/**: Stores utility functions such as logging, email services, and token generation.
- **db/connect.js**: Establishes the connection with MongoDB and handles database setup.

- **db/schema.js:** Defines database schemas and relationships between different collections.
- 

## 6. Running the Application

### Frontend:

To run the frontend, navigate to the client directory and run:

```
bash
```

```
Copy
```

```
npm start
```

### Backend:

To run the backend, navigate to the server directory and run:

```
bash
```

```
Copy
```

```
npm start
```

### Development vs. Production Mode:

- **Development Mode:** In development mode, the application runs with additional debugging tools and hot-reloading for faster development.
- **Production Mode:** In production mode, the application is optimized for performance, with minified code and reduced logging.

### Troubleshooting:

- **Common Issues:** If the application fails to start, ensure that all dependencies are installed and that the MongoDB server is running.
  - **Logs:** Check the console logs for any error messages or warnings that may indicate the source of the problem.
- 

## 7. API Documentation

The API documentation will include detailed routes, request/response examples, and explanations of each endpoint. This section will cover:

- **User Authentication:** Login, registration, and token refresh endpoints.
- **Product Management:** Endpoints for adding, updating, and deleting products.
- **Order Management:** Endpoints for placing, tracking, and canceling orders.

- **Admin Controls:** Endpoints for managing inventory, promotions, and customer queries.
- 

## 8. Authentication

### Security Mechanisms:

- **JWT Authentication:** JSON Web Tokens are used to securely authenticate users and manage sessions.
- **Password Hashing:** User passwords are hashed using bcrypt before being stored in the database.
- **Role-Based Access Control:** Different user roles (e.g., admin, customer) have different levels of access to the platform.

#### **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 canceling 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.

#### **User Role**

##### 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.

#### Token-Based Authentication:

- **Token Generation:** When a user logs in, a JWT is generated and sent to the client.
- **Token Validation:** The token is validated on each request to ensure the user is authenticated.
- **Token Expiry:** Tokens have an expiry time, after which the user must log in again.

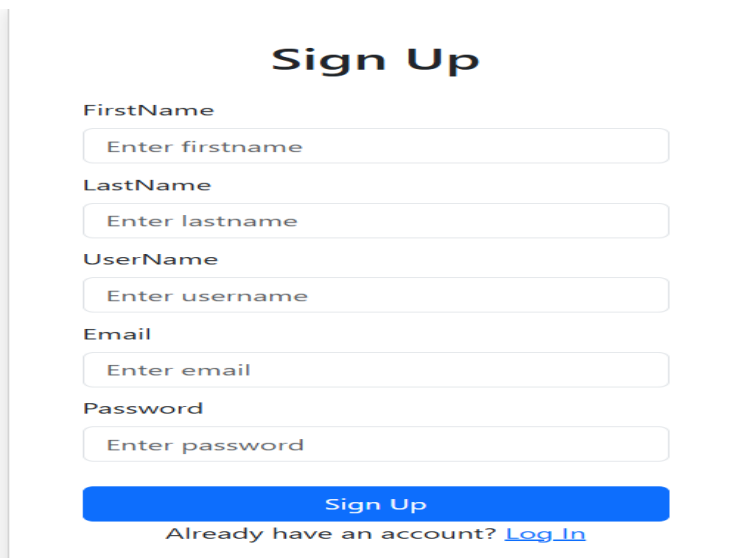
---

## 9. User Interface

#### Screenshots:

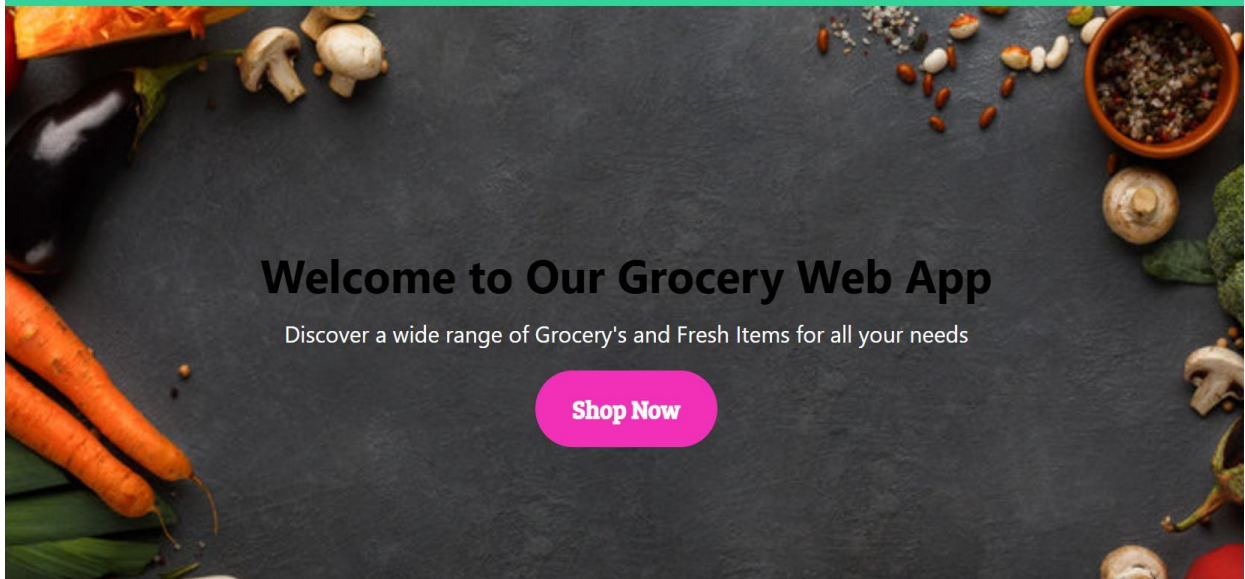
Screenshots of key pages, including the Home page, Product Details page, Checkout page, and Admin Dashboard, will be provided.

#### *Registration page:*

A screenshot of a 'Sign Up' form. The form is centered on a white background with a light gray border. It features a title 'Sign Up' in bold black text. Below the title are five input fields, each with a label above it: 'FirstName' (placeholder: 'Enter firstname'), 'LastName' (placeholder: 'Enter lastname'), 'UserName' (placeholder: 'Enter username'), 'Email' (placeholder: 'Enter email'), and 'Password' (placeholder: 'Enter password'). At the bottom of the form is a blue button with the text 'Sign Up'. Below the button is a link that says 'Already have an account? [Log In](#)'.

#### *Home page:*





**Login page:**

## Login

Email

Password

Login

Don't have an account? [Sign Up](#)

**Admin Dashboard:**

Grocery Web App

Dashboard Users Products Add product Orders Logout

## Dashboard

Product Count

0 Products

View Products

User Count

1 Users

View Users

Order Count

1 Orders

View Orders

Add Product

Add

### UI Flow Diagrams:

Flow diagrams will illustrate the user journey through the application, from browsing products to completing a purchase.

### Design Mockups:

Design mockups created using tools like Figma or Adobe XD will be included to showcase the visual design and layout of the platform.

---

## 10. Testing

### Testing Strategies:

- **Unit Testing:** Individual components and functions are tested to ensure they work as expected.
- **Integration Testing:** Different parts of the application are tested together to ensure they work seamlessly.
- **End-to-End Testing:** The entire application is tested from start to finish to simulate real user interactions.

### Automation Workflows:

Automated testing workflows using tools like Jest, Cypress, and Selenium will be implemented to ensure consistent and reliable testing.

---

## 11. Screenshots or Demo

A live demo of the platform will be provided, along with recorded walkthroughs of key features such as product browsing, order placement, and admin dashboard usage.

---

## 12. Known Issues

A detailed list of known issues and their resolutions will be provided. This section will include:

- **Bug Reports:** Descriptions of bugs encountered during development.
  - **Resolutions:** Steps taken to resolve each bug.
- 

## 13. Future Enhancements

### Integrations:

- **Third-Party APIs:** Integrate with third-party services for additional features such as recipe suggestions based on purchased items.
  - **IoT Integration:** Explore integration with IoT devices for smart grocery management (e.g., smart fridges that automatically reorder items).
-