

Shop Smart

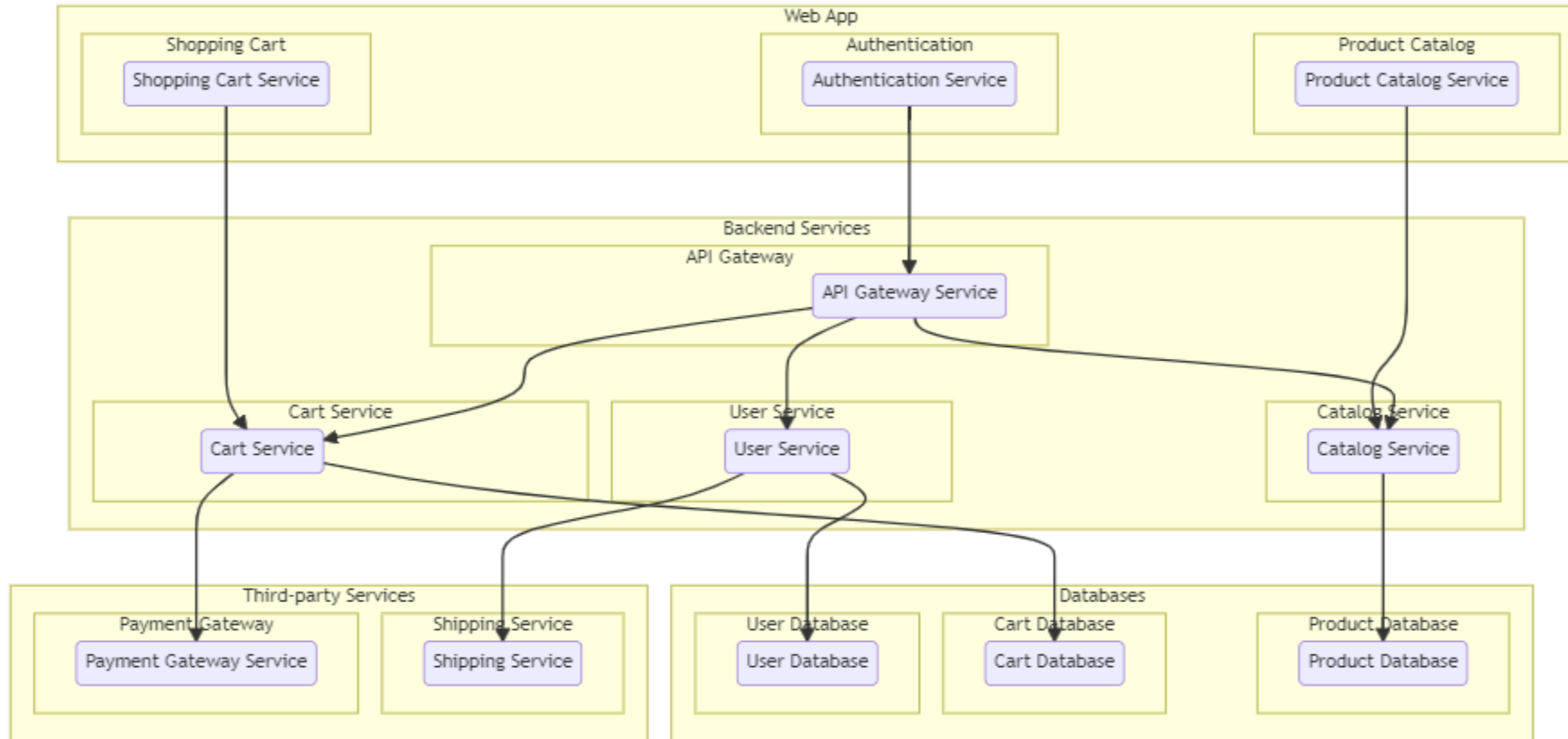
Your Digital Grocery Store Experience

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.

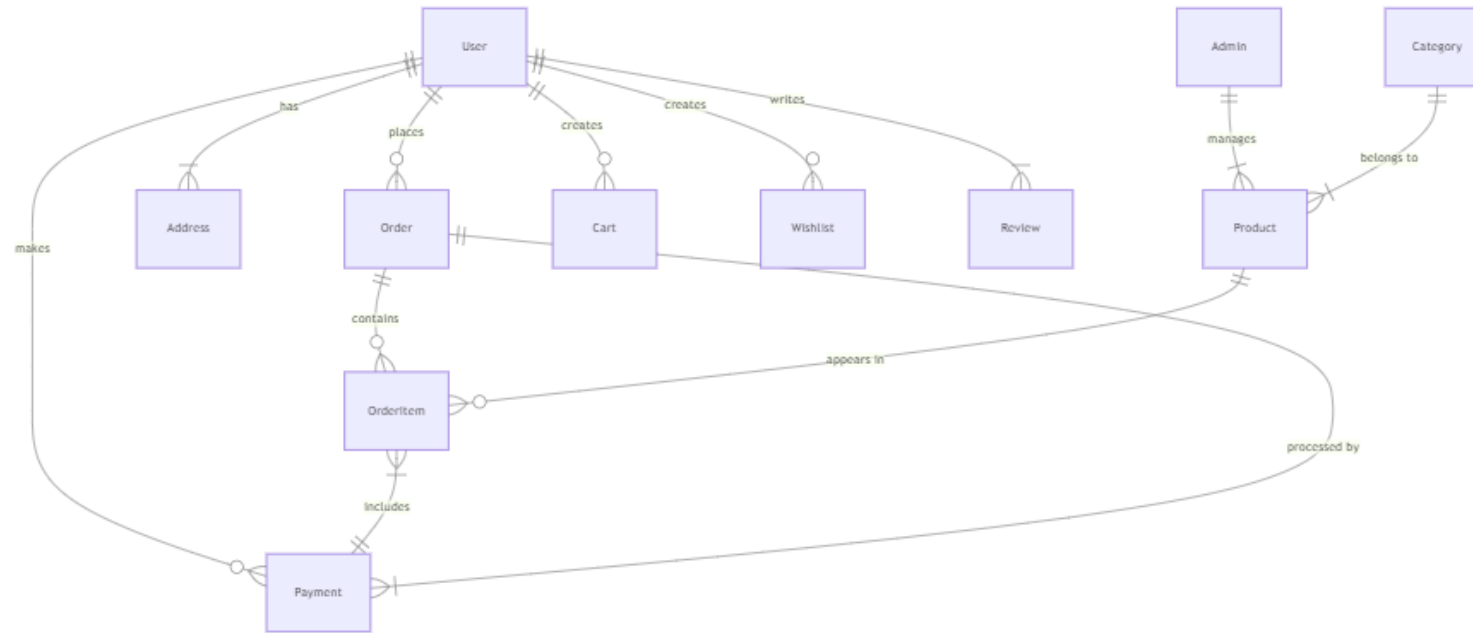
- For sellers and administrators, our app offers robust backend functionalities. Sellers can easily manage their product listings, inventory, and orders, while administrators can efficiently handle customer inquiries, process payments, and monitor overall app performance.
- With a focus on security and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

Architecture



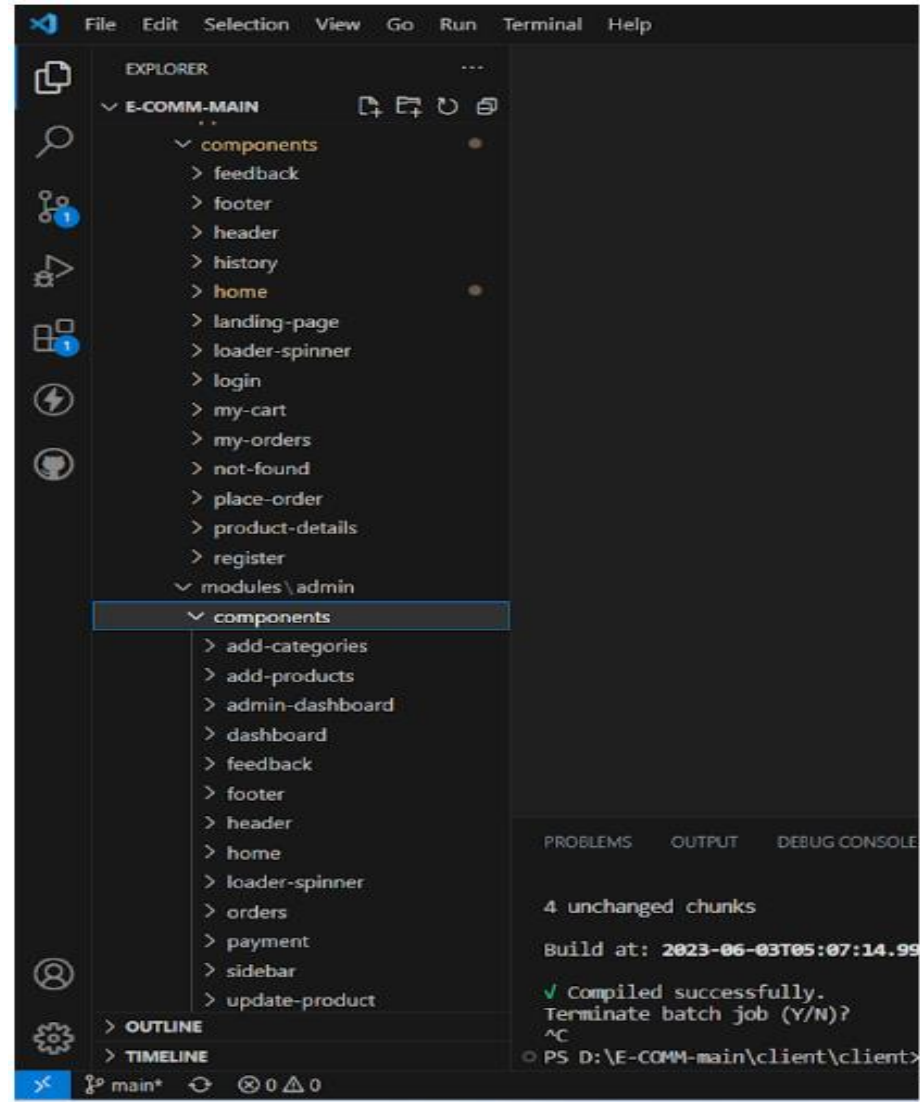
The technical architecture of a flower and gift delivery app typically involves a client-server model, where the frontend represents the client and the backend serves as the server. The frontend is responsible for user interface, interaction, and presentation, while the backend handles data storage, 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.

Project Structure



- **This structure assumes an Angular app and follows a modular approach. Here's a brief explanation of the main directories and files:**
- `src/app/components`: Contains components related to the customer app, such as register, login, home, products, my-cart, my-orders, placeorder, history, feedback, product-details, and more.
- `src/app/modules`: Contains modules for different sections of the app. In this case, the admin module is included with its own set of components like add-category, add-product, dashboard, feedback, home, orders, payment, update-product, users, and more.
- `src/app/app-routing.module.ts`: Defines the routing configuration for the app, specifying which components should be loaded for each route.
- `src/app/app.component.ts`, `src/app/app.component.html`, ``src``.

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 setup Angular on your machine and created a new Angular project.**

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>
- **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>

Git Repository Cloning

- To run the existing grocery-web app project downloaded from github:
- Follow below steps:
 1. Clone the Repository
- Open your terminal or command prompt.
- Navigate to the directory where you want to store the grocery-webapp app.
- Execute the following command to clone the repository:
- **git clone** <https://github.com/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`

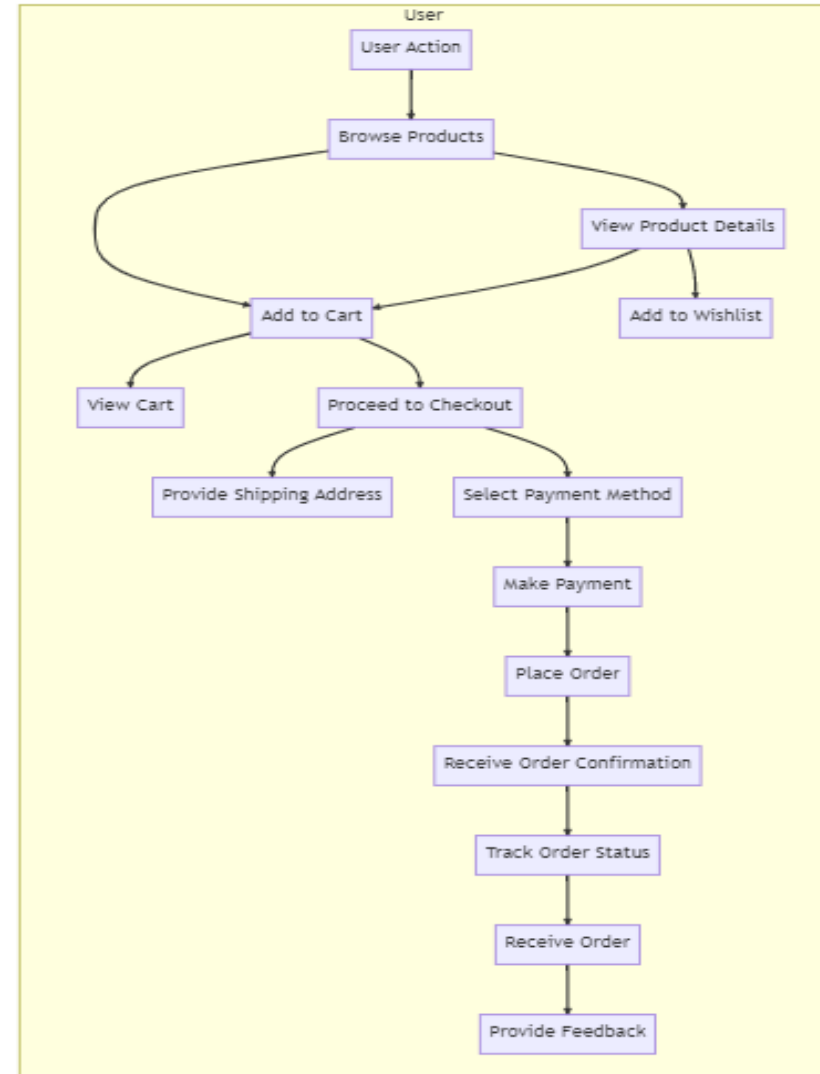
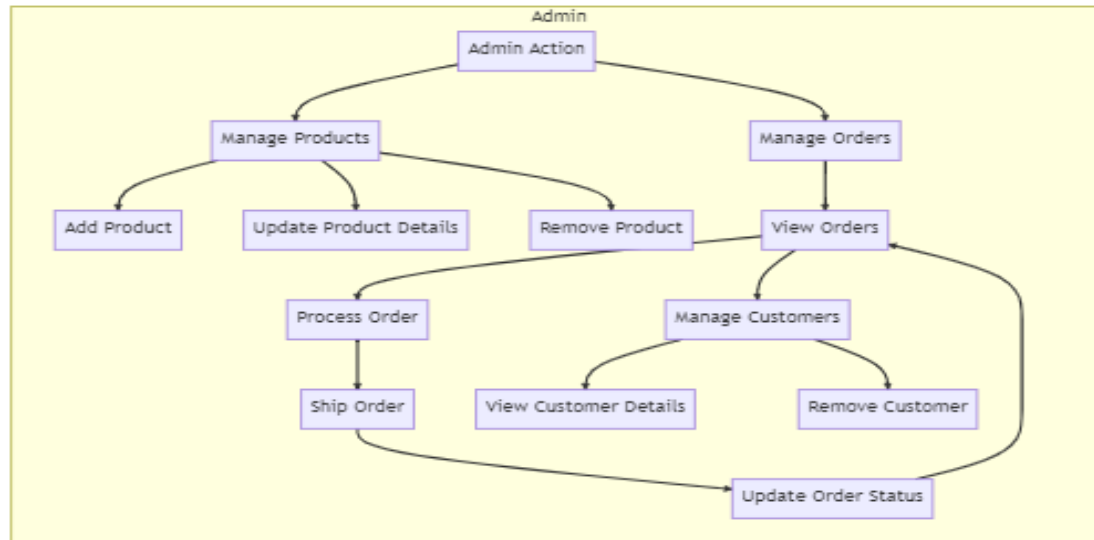
3. Start the Development Server:

- 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

Role Based Access



1 - Admin Role:

- Responsibilities: The admin role has full control and administrative privileges over the system.
- Permissions:
 - Manage: Admins can add, edit, and delete shop information along with products.
 - Manage product bookings: Admins can view and manage all product bookings made by users and agents, including 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.

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.

Project Flow

- Frontend Development
- Backend Development
- Integration
- **Demo link:-**
- https://drive.google.com/file/d/1HLowclqs2d8lxTprS2jqPmR4AOnUW8xD/view?usp=drive_link

Use the code in:-

- https://drive.google.com/drive/folders/1RP-29p9mf-bbLAK5r7S4HSF_Itai0i1G?usp=drive_link

Frontend Development

- Frontend development involves building the user interface (UI) and implementing the visual elements of the online shopping web application. It focuses on creating an intuitive and engaging user experience that allows users to interact with the application seamlessly.
- **To Setup the frontend development and to connect node.js with MongoDB Database Go through this video link: -**
https://drive.google.com/file/d/1b5bMvnqmASXLnSZ74B2t3EzNjuWHj63g/view?usp=drive_link

User Interface (UI) Design

- Create a visually appealing and consistent design using modern design principles.
- Use a UI design tool like Adobe XD, Sketch, Figma, or InVision to create wireframes and mockups.
- Pay attention to typography, color schemes, spacing, and visual hierarchy.
- Use responsive design techniques to ensure the app looks great on different devices.

Responsive Design

- Utilize CSS media queries and responsive design frameworks like Bootstrap or Tailwind CSS to create a responsive layout.
- Test your app on various devices and screen sizes to ensure a seamless user experience.

Product Catalog

- Design and implement a product listing page that displays product images, titles, descriptions, prices, and other relevant details.
- Implement search functionality to allow users to find products easily.
- Include filters and sorting options to enhance the browsing experience.

Shopping Cart and Checkout Process

- Design and develop a shopping cart component to allow users to add products, view cart contents, update quantities, and remove items.
- Create a checkout process with multiple steps, including shipping information, payment selection, and order review.

User Authentication and Account Management

- Design and implement a user registration and login system.
- Create user profile pages where users can view and edit their personal information, addresses, payment methods, and order history.
- Implement authentication guards to restrict access to certain pages or features.

Payment Integration

- Integrate with a payment gateway service like Stripe, PayPal, or Braintree.
- Implement a secure and seamless payment flow that allows users to enter payment details and complete transactions.
- Handle transaction success and failure scenarios and provide appropriate feedback to the user.

Backend Development

- Backend development involves building the server-side components and logic of the online shopping web application. It focuses on handling the business logic, processing requests from the front end, and interacting with the database. The following activities are part of the backend development process:

```
const mongoose = require("mongoose");

const db= 'mongodb://127.0.0.1:27017/grocery'
// Connect to MongoDB using the connection string

mongoose.connect(db, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
}).then(() => {
  console.log(`Connection successful`);
}).catch((e) => {
  console.log(`No connection: ${e}`);
});
```

Set Up 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 cores 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.
- Connect database to backend:

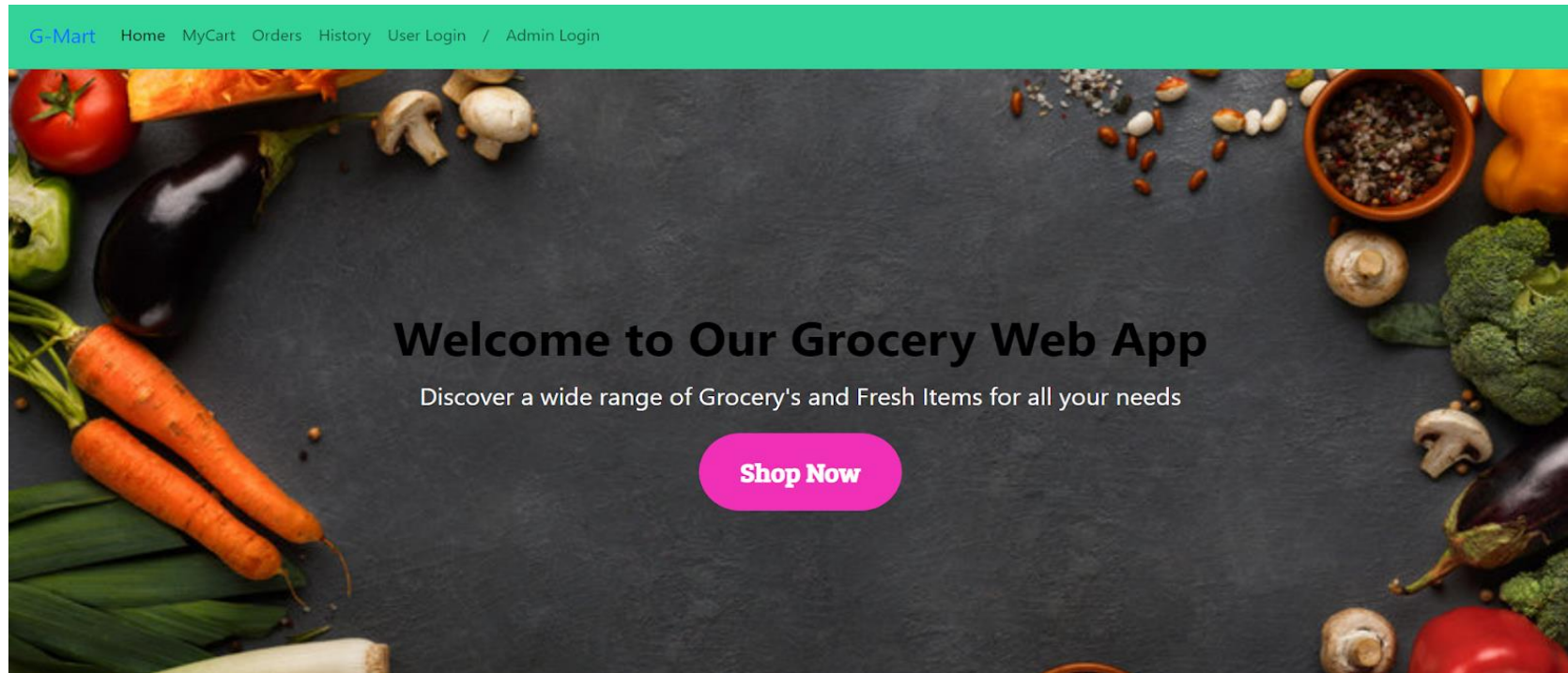
```
const mongoose = require("mongoose");

const db= 'mongodb://127.0.0.1:27017/grocery'
// Connect to MongoDB using the connection string

mongoose.connect(db, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
}).then(() => {
  console.log(`Connection successful`);
}).catch((e) => {
  console.log(`No connection: ${e}`);
});
```

Integration

- Integration is the process of combining and connecting the frontend and backend components of the online flower shop web application to create a unified and fully functional system. It involves establishing communication channels, exchanging data, and ensuring seamless interaction between the frontend UI and backend APIs. The following activities are part of the integration process:
- Landing page:-



Login Page:

[G-Mart](#) [Home](#) [MyCart](#) [Orders](#) [History](#) [User Login](#) / [Admin Login](#)

Login

Email

Password

Login

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

Item's Page:

[G-Mart](#) [Home](#) [MyCart](#) [Orders](#) [History](#) [Logout](#)

Search By Product Name

Filter By Category

all



Products



Apple
\$200

Buy Now

Add to Cart



orange
\$120

Buy Now

Add to Cart



Milk
\$80

Buy Now

Add to Cart



Cashew
\$800

Buy Now

Add to Cart



Chicken
\$250


Buy Now

Add to Cart

My Cart:


[G-Mart](#) [Home](#) [MyCart](#) [Orders](#) [History](#) [Logout](#)

My Cart




Apple
\$200

[Remove from Cart](#) [Buy this](#)



Milk
\$80

[Remove from Cart](#) [Buy this](#)



Chicken
\$250

[Remove from Cart](#) [Buy this](#)

My Order's Page:

[G-Mart](#) [Home](#) [MyCart](#) [Orders](#) [History](#) [Logout](#)

My Orders

Order ID: 6614b085c30b51d3c700f20a

Name: syed arshad

Phone: 9505221870

Date: 2024-04-09T03:05:41.563Z

Price: 400

Status: Pending

Payment Method: credit

My History Page:

[G-Mart](#) [Home](#) [MyCart](#) [Orders](#) [History](#) [Logout](#)

My History

Order ID: 6614a8ddc30b51d3c700f1b4

Name: syed arshad

Phone: 9505221870

Date: 2024-04-09T02:36:47.498Z

Price: 400

Status: Delivered

Payment Method: debit

Place Order Page:

[G-Mart](#) [Home](#) [MyCart](#) [Orders](#) [History](#) [Logout](#)

Order Details

First Name:

Last Name:

Phone:

Quantity:

Address:

Payment Method:

Cash on Delivery (COD) ▼


Submit

User Page:

Grocery Web App

[Dashboard](#) [Users](#) [Products](#) [Add product](#) [Orders](#) [Logout](#)

Users

sl/no	UserId	User name	Email	Operation
1	6614a859c30b51d3c700f1a0	syed	syed@gmail.com	 view

Add Product Page:

Grocery Web App

[Dashboard](#) [Users](#) [Products](#) [Add product](#) [Orders](#) [Logout](#)

Add Product

Product Name	Rating	Price
<input type="text" value="Enter product name"/>	<input type="text" value="Enter product rating"/>	<input type="text" value="Enter product price"/>
Image URL	Category	Count in Stock
<input type="text" value="Enter image URL"/>	<input data-bbox="1108 763 1439 792" type="text" value="Select Category"/>	<input type="text" value="Enter count in stock"/>
Description		
<input type="text" value="Enter product description"/>		
<input type="button" value="Add Product"/>		

Admin Order's Page:

Orders

<p>Order ID: 6614a8ddc30b51d3c700f1b4</p> <p>Fullname: syed arshad</p> <p>Phone: 9505221870</p> <p>Product ID: 660e84641c659d0b2b10e135</p> <p>Quantity: 2</p> <p>Total price: 400</p> <p>Payment Method: debit</p> <p>Address: hyderabad</p> <p>Created At: 2024-04-09T02:36:47.498Z</p> <p>Status: Delivered</p>	<div>Delivered</div>
<p>Order ID: 6614b085c30b51d3c700f20a</p> <p>Fullname: syed arshad</p> <p>Phone: 9505221870</p> <p>Product ID: 660e84641c659d0b2b10e135</p> <p>Quantity: 2</p> <p>Total price: 400</p>	

Frontend-Backend Integration

- Integrate the frontend UI components with the backend APIs, ensuring proper communication and data exchange.
- Implement API calls from the front end to retrieve the data.
- Handle data validation and error responses between the frontend and backend components.
- Conduct thorough testing to ensure seamless integration and compatibility between frontend and backend.

