

**TO DEVELOP A PROJECT ON**

**GROCERY WEBAPP USING (MERN stack by MongoDB)**

A PROJECT REPORT

*Submitted by*

**MARKESHWARAN M - 110521104018**

**MOHAMED ISMAIL N - 110521104020**

**NAVEEN KUMAR E - 110521104021**

**NAVIN BALAJI S - 1105211044022**

*In partial fulfilment for the award of the degree*

*Of*

BACHELOR OF ENGINEERING

IN

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GOJAN SCHOOL OF BUSINESS AND TECHNOLOGY

ANNA UNIVERSITY, CHENNAI-600 025.

**ABSTRACT:**

1. **Introduction**
2. **Technologies Used (MERN Stack)**
3. **System Architecture and Structure**
4. **Frontend Development (React.js)**
5. **Backend Development (Node.js & Express.js)**
6. **Database Design (MongoDB)**
7. **Key Features and Functionalities**
8. **Security Measures**
9. **Testing & Deployment**
10. **Conclusion**

**Executive Summary**

This report provides a comprehensive overview of the development of a Grocery Web App using the MERN stack, which stands for MongoDB, Express.js, React.js, and Node.js. The goal of the app is to provide an online platform for users to browse, add, and purchase groceries. It offers an intuitive user interface for customers and a robust backend for managing products, users, orders, and payments. The app will use modern web technologies to ensure a responsive, scalable, and secure application.

**1. Introduction**

The purpose of this project is to create a full-featured grocery web application to allow users to shop for groceries online, with a focus on simplicity, speed, and responsiveness. The MERN stack provides an ideal combination of technologies that work well together for building modern, scalable web applications. The app will cater to both customers who wish to buy groceries and administrators who need to manage product listings, orders, and user information.

**2. Technologies Used (MERN Stack)**

**MongoDB**

MongoDB is a NoSQL database that stores data in flexible, JSON-like documents, allowing us to represent complex, hierarchical relationships and data structures. It is highly scalable and ideal for managing unstructured data, such as user profiles, product catalogs, and transaction histories.

**Express.js**

Express.js is a minimal and flexible Node.js web application framework that provides a robust set of features to build web and mobile applications. It simplifies the backend API routing and management of HTTP requests and responses, enabling efficient communication between the frontend and the database.

**React.js**

React.js is a JavaScript library for building user interfaces, specifically single-page applications (SPAs). It is component-based, which allows for modularity and reuse of code. React enables efficient updating and rendering of UI components in response to data changes.

**Node.js**

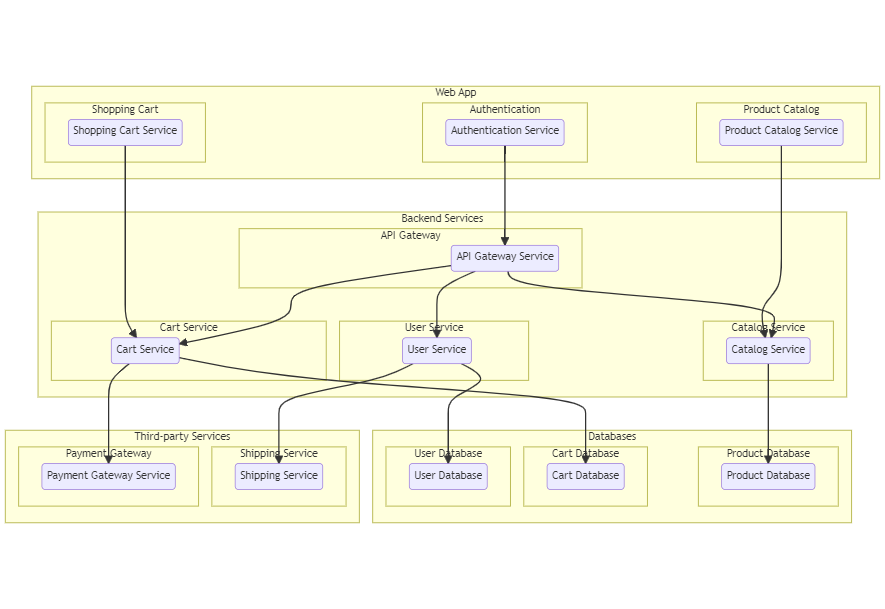
Node.js is a JavaScript runtime built on Chrome's V8 engine that enables server-side scripting. It allows developers to use JavaScript for both the frontend and backend of the application, leading to a unified development environment.

**3. System Architecture and Structure**

The architecture follows a client-server model where the frontend (React.js) communicates with the backend (Node.js/Express) via HTTP APIs, and the backend communicates with MongoDB to retrieve or manipulate data.

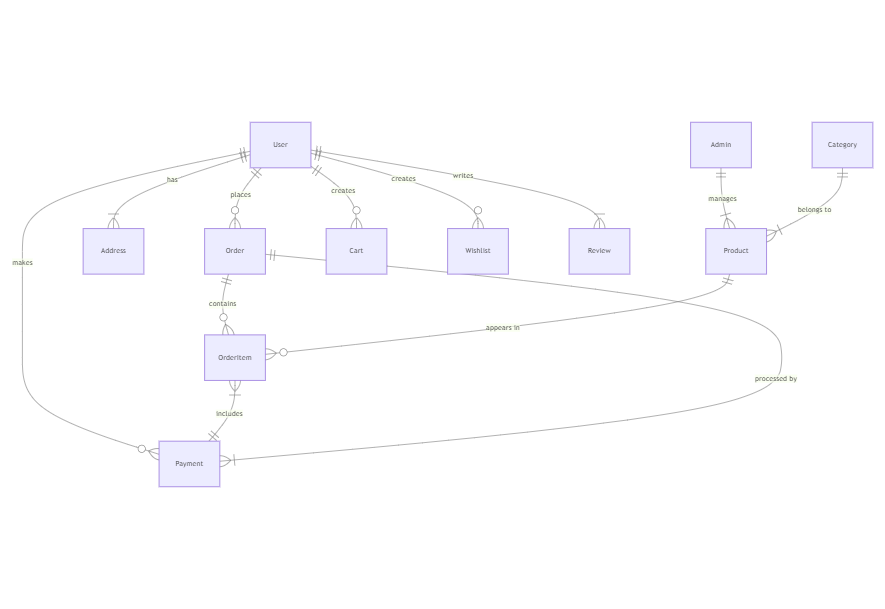
**Overall System Architecture:**

1. **Frontend (Client Side):**
   * React.js: Manages the presentation layer, handles user interactions, and communicates with the backend through RESTful APIs (using axios or fetch).
   * Redux: Optionally used for managing global state, especially for handling user authentication and managing the shopping cart.
   * CSS/Bootstrap/Tailwind: For responsive UI design and styling.
2. **Backend (Server Side):**
   * Node.js with Express.js: Handles routing and business logic.
   * Authentication: Uses JWT (JSON Web Tokens) for user authentication and session management.
   * REST APIs: CRUD operations for managing users, products, orders, and payments.
3. **Database:**
   * MongoDB: Stores data related to products, orders, users, and cart items. It uses collections for each type of entity, with schemas defining the structure of documents.



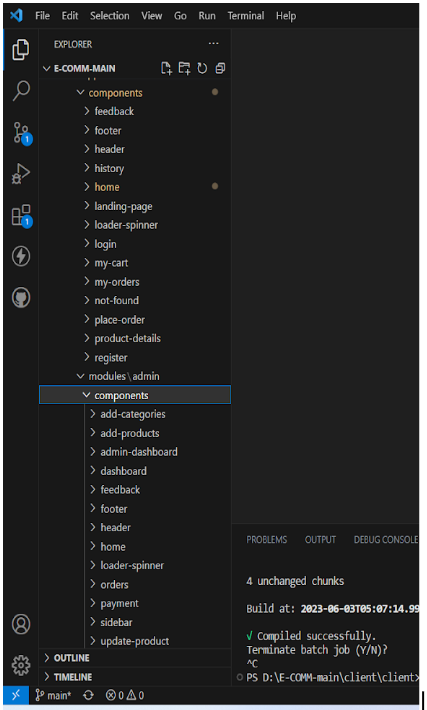
The technical architecture of an flower and gift delivery app typically involves a client-server model, where the frontend represents the client and the backend serves as the server. The frontend is responsible for user interface, interaction, and presentation, while the backend handles data storage,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

**ER Diagram:**



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

**4.Project 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.

**5.Frontend Development (React.js)**

**Components**:

1. **Home Page:** Displays featured products, promotions, and categories. React components like <ProductCard> can be reused to display individual products.
2. **Product Detail Page:** Shows detailed information about a specific product, including price, description, and available quantity.
3. **Shopping Cart:** Displays the items added to the cart, their quantities, and total price. Users can update quantities or remove items.
4. **Checkout:** Allows users to enter shipping information, review their order, and complete the purchase. Integration with payment gateways like Stripe or PayPal is often included here.
5. **Admin Dashboard**: For managing products, users, and orders. Admin can add, update, or delete products and view customer orders.

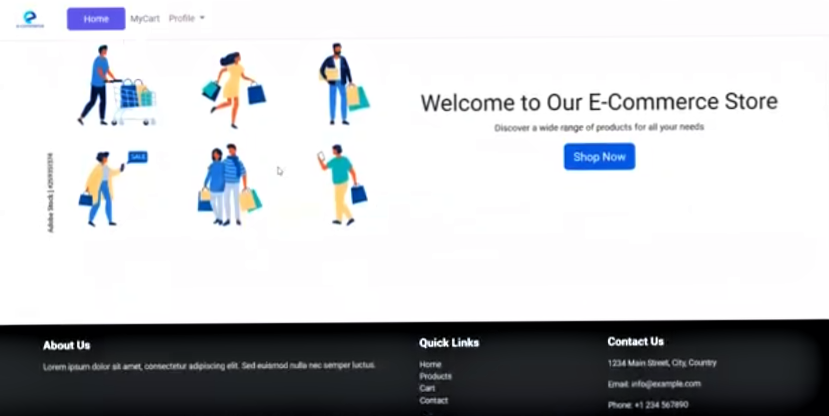
**State Management:**

**State management in React can be handled using:**

* useState and useEffect for local state within individual components.
* React Context API or Redux for global state (e.g., cart, user authentication state).

**Routing:**

React Router is used to handle routing between different pages such as the homepage, product pages, cart, checkout, and user profile.



**6. Backend Development (Node.js & Express.js)**

**API Design**:

The backend consists of several RESTful endpoints to handle different operations. Common routes include:

* **User Authentication:**
  + **POST /api/auth/register**: Register a new user.
  + **POST /api/auth/login**: Authenticate and login the user.
  + **GET /api/auth/logout**: Logout the user.
* **Product Management:**
  + **GET /api/products**: Retrieve a list of products.
  + **GET /api/products/:id:** Retrieve a single product by ID.
  + **POST /api/products**: Add a new product (Admin only).
  + **PUT /api/products/:id**: Update product details (Admin only).
  + **DELETE /api/products/:id**: Delete a product (Admin only).
* **Order Management:**
  + **GET /api/orders:** Retrieve all orders for an authenticated user.
  + **POST /api/orders**: Place a new order.
  + **GET /api/orders/:id:** Retrieve a specific order by ID.

**Middleware:**

Middleware functions are used to:

* **Authentication**: Verify if the user is authenticated via JWT tokens.
* **Authorization**: Check if the user has admin privileges for certain routes.
* **Validation**: Ensure input data is valid before passing it to the database.

**7. Database Design (MongoDB)**

**MongoDB's flexible schema allows for easy modification and scaling. Key collections in the database are:**

* **Users**: Stores user profiles with fields such as name, email, passwordHash, and orderHistory.
* **Products**: Contains product data including name, description, price, category, and imageURL.
* **Orders:** Stores orders with fields like userId, productList, totalPrice, shippingAddress, and status.
* **Payments**: Holds payment transaction details, including payment status, transaction ID, and method (credit card, PayPal, etc.).

**8. Key Features and Functionalities**

**1.User Authentication**: Allows users to sign up, log in, and manage their profile.

**2.Product Browsing**: Users can browse products by category, search for specific items, and view product details.

**3.Shopping Cart**: Users can add products to the cart, update quantities, and proceed to checkout.

**4.Order Management**: Users can view their order history and track the status of ongoing orders.

**5.Admin Dashboard:** Admins can add, update, delete products, and manage user orders.

**6.Payment Integration**: Integrates with third-party services (Stripe/PayPal) for secure payments.

**9. Security Measures**

The grocery web app implements several security measures to protect user data and ensure secure transactions:

**1.HTTPS:** All communication between the client and server is encrypted using SSL/TLS.

**2.Password Hashing:** User passwords are hashed using bcrypt before being stored in the database to prevent unauthorized access.

**3.JWT Authentication:** User sessions are authenticated using JWT tokens, which are stored securely in HTTP-only cookies.

**4.Input Validation:** All user inputs, especially payment details, are validated and sanitized to prevent SQL injection and other malicious attacks.

**5.Payment Security:** Payment gateways like Stripe and PayPal are used to ensure that sensitive payment information is handled securely.

**10. Performance and Scalability**

The web app is designed for performance and scalability, with the following considerations:

* **Database Indexing:** MongoDB collections are indexed for faster queries, particularly for frequently accessed data such as product listings and user order histories.
* **Caching:** Redis or similar caching mechanisms can be used to cache frequent queries (e.g., product listings) to reduce server load.
* **Horizontal Scaling:** The application can be scaled horizontally by deploying additional instances of the back-end server to handle increased traffic.
* **Load Balancing:** A load balancer is implemented to distribute traffic evenly across multiple server instances.

**11.Testing and Deployment**

**Testing**

* **Unit Tests:** Use Jest or Mocha for testing backend logic (e.g., API endpoints).
* **UI Testing:** Use Cypress for end-to-end testing of frontend components.
* **Integration Tests:** Test end-to-end flows such as user registration, checkout, and order creation.

**Deployment**

* **Frontend:** Deploy React frontend using platforms like Vercel or Netlify.
* **Backend:** Deploy the backend API on platforms such as Heroku or AWS.
* **Database:** Use MongoDB Atlas for cloud-based database hosting.

**10. Conclusion**

This Grocery Web App, built with the MERN stack, offers a robust and scalable solution for online grocery shopping. By using MongoDB for data storage, Express.js for handling backend routing, React.js for dynamic frontend rendering, and Node.js for backend logic, the app is designed to provide a seamless and secure shopping experience. The application includes essential features like user authentication, product management, shopping cart functionality, and order processing. With additional security measures and testing, the app is ready for deployment in a production environment.