***Vegitable E-commerce Website***

Creating architectural documentation for a vegetable eCommerce website using React.js, Node.js, and MySQL involves detailing the components, technologies, workflows, and interactions that enable the system to function. Here’s a comprehensive outline for your architecture:

**1. High-Level Architecture Overview**

The eCommerce website consists of three main components:

* **Frontend (React.js)**: Handles the user interface and user experience (UI/UX).
* **Backend (Node.js)**: Manages business logic, API endpoints, authentication, and communication with the database.
* **Database (MySQL)**: Stores data like user information, product inventory, orders, and payment details.

**2. Technology Stack**

* **Frontend**:
  + **React.js**: JavaScript library for building the UI.
  + **Redux** or **Context API**: State management.
  + **Axios** or **Fetch API**: For making HTTP requests to the backend.
* **Backend**:
  + **Node.js**: JavaScript runtime for server-side logic.
  + **Express.js**: Framework for building RESTful APIs in Node.js.
  + **JWT (JSON Web Token)**: For authentication and authorization.
  + **Bcrypt.js**: For password hashing and security.
  + **Nodemailer**: For sending email notifications (e.g., order confirmation).
* **Database**:
  + **MySQL**: Relational database to store structured data.
  + **Sequelize ORM** (optional): An ORM for managing MySQL database queries from Node.js.
* **Other Tools**:
  + **Docker**: Containerization for easy deployment.
  + **AWS S3** (optional): For storing product images.
  + **Stripe/PayPal API**: For handling payments.

**3. Architecture Components**

**Frontend (React.js)**

* **Components**:
  + **Home Page**: Displays the list of available vegetables and products.
  + **Product Page**: Detailed page showing individual vegetable/product information.
  + **Shopping Cart**: Allows users to view items they plan to purchase.
  + **Checkout**: Handles order confirmation, billing, shipping info, and payment gateway integration.
  + **User Authentication Pages**: Sign up, login, profile management.
* **State Management**: Use **Redux** or **Context API** to manage user state (e.g., logged-in status), cart data, and order information across different components.
* **Routing**: React Router is used to navigate between different pages (e.g., home, product details, cart, checkout).
* **API Calls**: Axios will be used to interact with backend APIs, for example, fetching product details or submitting order data.

**Backend (Node.js with Express.js)**

* **API Endpoints**:
  + **User Authentication**:
    - POST /api/auth/register: Registers a new user.
    - POST /api/auth/login: Authenticates a user.
    - GET /api/auth/logout: Logs out a user.
  + **Product Endpoints**:
    - GET /api/products: Lists all available products.
    - GET /api/products/:id: Gets product details by ID.
    - POST /api/products: Admin only, to add new products.
    - PUT /api/products/:id: Admin only, to update product details.
    - DELETE /api/products/:id: Admin only, to delete a product.
  + **Cart and Order Endpoints**:
    - POST /api/cart: Adds items to the shopping cart.
    - GET /api/cart: Fetches cart items for the logged-in user.
    - POST /api/order: Submits a new order.
    - GET /api/orders: Fetches order history for a user.
  + **Payment Integration**:
    - Stripe/PayPal endpoints for processing payments after checkout.
* **Authentication**:
  + JWT-based authentication. The backend will issue a JWT upon successful login, and the token must be included in the headers of subsequent requests to protected routes.
* **Middleware**:
  + **Authentication Middleware**: Ensures routes are protected and users must be logged in to access certain resources (e.g., placing an order).
  + **Error Handling Middleware**: Catches and handles errors globally.
  + **Body Parsing Middleware**: For parsing incoming request bodies (e.g., express.json()).

**Database (MySQL)**

* **Entities**:
  + **Users Table**: Stores user information (ID, name, email, hashed password, etc.).
  + **Products Table**: Stores product data (ID, name, description, price, stock, image URL, etc.).
  + **Cart Table**: Stores temporary cart data for logged-in users (cart item ID, product ID, quantity).
  + **Orders Table**: Stores order details (order ID, user ID, product IDs, quantities, order status, total price).
  + **Order\_Items Table**: Stores individual items in an order (order ID, product ID, quantity).
* **Schema Example**:

sql

Copy code

CREATE TABLE users (

id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(255),

email VARCHAR(255) UNIQUE,

password VARCHAR(255),

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE products (

id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(255),

description TEXT,

price DECIMAL(10, 2),

stock INT,

image\_url VARCHAR(255),

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE orders (

id INT PRIMARY KEY AUTO\_INCREMENT,

user\_id INT,

total DECIMAL(10, 2),

order\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

status ENUM('pending', 'completed', 'shipped', 'cancelled')

);

CREATE TABLE order\_items (

id INT PRIMARY KEY AUTO\_INCREMENT,

order\_id INT,

product\_id INT,

quantity INT,

price DECIMAL(10, 2)

);

**4. Detailed Flow**

**User Flow:**

1. **Browse Products**: The user browses the homepage, filters through different vegetable categories, and adds products to the cart.
2. **Cart Management**: The cart shows all selected products with their quantities and allows the user to update or remove items.
3. **Checkout Process**:
   * User provides shipping details and selects a payment method.
   * The backend processes the payment using Stripe/PayPal API.
   * After payment confirmation, the order is placed, and inventory is updated.
4. **Order Confirmation**: The system sends an email confirmation with the order details.

**Admin Flow:**

1. **Manage Products**: Admin can add, update, or delete products from the database.
2. **View Orders**: Admin can view a list of placed orders and their status.
3. **Inventory Management**: Admin ensures that the product stock is updated after every order.

**5. Security Considerations**

* **JWT Tokens**: Ensure tokens are securely stored in HTTP-only cookies or in local storage with proper security headers.
* **Password Hashing**: Use bcrypt.js to hash passwords before storing them in the database.
* **SQL Injection Prevention**: Use ORM (e.g., Sequelize) or parameterized queries to avoid SQL injection.
* **Data Validation**: Validate all user inputs to prevent invalid data from being processed.

**6. Deployment and Scalability**

* **Docker**: Containerize the application for easy deployment across different environments.
* **Cloud Services**: Use services like **AWS**, **Heroku**, or **DigitalOcean** to deploy the app.
* **Load Balancing and Auto-scaling**: Use load balancing and auto-scaling for handling high traffic, especially during peak seasons.

**7. Future Enhancements**

* **Real-Time Features**: Integrate real-time updates (e.g., order tracking, inventory updates) using WebSockets or similar technologies.
* **Mobile App**: Build a companion mobile app with React Native for broader reach.
* **AI/Recommendation Engine**: Suggest related products based on user behavior.

This architecture will allow you to scale, secure, and maintain the application efficiently while providing an optimal user experience.