# ShopSmart: Your Digital Grocery Store Experience

| | |
|---|---|
| DATE | 23-02-2026 |
| TEAM ID | **LTVIP2026TMIDS37243** |
| PROJECT NAME | ShopSmart: Your Digital Grocery Store Experience |
| COLLEGE | Ideal Institute Of Technology Kakinada |

## TEAM MEMBERS


1. **Team Leader :** Mosa Dhanush
   Email : mosamani35@gmail.com
2. **Team member :** Mangipudi V Srinivasa Subrahmanya Sarma
   Email : mvsss7150@gmail.com
3. **Team member :** Pilli Haritha
   Email : haritha9126@gmail.com
4. **Team member :** Thalatam Satya Harini
   Email : talatamsatyaharini@gmail.com
5. **Team member :** Sathi Grishmanjana
   Email : anjanaram219@gmail.com

# 2.Project Overview

### E-Cart: Digital Grocery Store Application

## INTRODUCTION

E-Cart is your one-stop digital destination for effortless grocery shopping. With a user-friendly interface and a comprehensive catalog of fresh produce and household essentials, finding everything you need for your home has never been easier.

Seamlessly navigate through detailed product descriptions, nutritional information, and available discounts to make healthy and informed decisions. Enjoy a secure checkout process and receive instant order confirmation for your weekly supplies. For vendors and store managers, our robust dashboard provides efficient inventory management and insightful analytics to drive business growth. Experience the future of grocery shopping with e-Cart today.

- **Seamless Checkout Process**
- **Fresh Product Discovery**
- **Personalized Grocery Lists**
- **Efficient Order Management for Vendors**
- **Inventory Analytics for Business Growth**

## Scenario: Mark’s Weekly Meal Prep

Mark, a fitness enthusiast with a busy work schedule, is planning his weekly meal prep. He needs fresh vegetables, lean proteins, and specific organic pantry staples, but he doesn't have time to visit multiple local markets. Feeling overwhelmed by his long list, Mark turns to **e-Cart** to simplify his shopping.

1. **Effortless Product Discovery:** Mark opens e-Cart and navigates to the "Organic Produce" category. He is greeted with a diverse range of options, from farm-fresh kale to seasonal fruits. Using filtering options, Mark selects "High Protein" and refines his search based on his dietary preferences and budget.

2. **Personalized Recommendations:** As Mark scrolls through his selection, he notices a "Frequently Bought Together" section. He discovers a discounted bundle of quinoa and almond milk that perfectly complements his meal plan. Impressed by the suggestion, he adds them to his cart.

3. **Seamless Checkout Process:** With his groceries in the cart, Mark proceeds to checkout. He selects his preferred delivery time slot and enters his payment details. Thanks to e-Cart's secure process, he completes the transaction in seconds.

4. **Order Management for Vendors:** On the other end, the local organic farm receives a notification of Mark's purchase through the e-Cart vendor dashboard. They quickly pack the fresh items to ensure quality, confident in the system's streamlined management.

## TECHNICAL ARCHITECTURE

The application follows a modern MERN stack architecture to ensure scalability and performance.

- **Frontend (React):** Manages the user interface, including User Authentication, Grocery Cart, Product Categories, and the Admin Dashboard.
- **Backend (Node.js & Express):** Consists of API endpoints for handling Users, Grocery Orders, and Product Inventory.
- **Database (MongoDB & Mongoose):** Stores collections for Users, Carts, Orders, and Grocery Products.

## CORE ENTITIES (ER MODEL)

The e-Cart system is built upon several interconnected entities:

- **USER:** Stores details of registered shoppers and their delivery addresses.
- **PRODUCTS:** A collection of all grocery items including descriptions, stock levels, and expiration dates.
- **CART:** Stores products added by users, differentiated by their unique User ID.
- **ORDERS:** Tracks all completed transactions, including payment methods and delivery statuses.
- **ADMIN/VENDOR:** Manages the platform's banners, categories, and inventory updates.

## KEY FEATURES

1. **Comprehensive Grocery Catalog:** e-Cart boasts an extensive catalog of items, from daily essentials to gourmet imports. Users can explore products with detailed nutritional facts, pricing, and active discounts.

2. **"Add to Cart" Functionality:** Every item features a convenient button for immediate selection, allowing users to build their weekly list effortlessly.

3. **Order Summary Page:** Before finalization, users are directed to a summary page to review their items, apply coupons, and confirm delivery details.

## 3.ARCHITECTURE

The ShopSmart application follows a modern **MERN** stack architecture, designed to ensure high performance, scalability, and a seamless grocery shopping experience.

- **Frontend (React):** Manages the user-facing interface, including User Authentication, the Grocery Cart, Product Categories (e.g., Produce, Dairy, Pantry), and the Admin/Vendor Dashboard.
- **Backend (Node.js & Express):** Provides the logic and API endpoints for managing Users, processing Grocery Orders, and updating Product Inventory.
- **Database (MongoDB & Mongoose):** A NoSQL database that stores collections for Users, Carts, Orders, and Grocery Products.

## 4. SETUP INSTRUCTIONS

To run the ShopSmart Digital Grocery Store locally, follow these comprehensive steps to configure your development environment and launch the application.

**I. Prerequisites**

Before beginning the installation, ensure your machine has the following software installed:

- **Node.js & npm:** The runtime environment and package manager required to run JavaScript on the server and install dependencies. (Recommended: Node.js v16 or higher).
- **MongoDB:** The NoSQL database used to store grocery products, user data, and orders. You can use MongoDB Community Server (local) or MongoDB Atlas (cloud).
- **Git:** Version control system to clone the project repository.
- **VS Code:** A recommended IDE for code editing and terminal management.
- **Web Browser:** A modern browser like Google Chrome or Mozilla Firefox for testing the application interface.

### II. Installation Steps

**Step 1: Clone the Repository** Open your terminal or command prompt and run the following command to download the source code to your local machine:

```
git clone https://github.com/your-username/shopsmart-grocery-store.git
```

**Step 2: Navigate to the Project Directory** Enter the root folder of the cloned project:

```
cd shopsmart-grocery-store
```

**Step 3: Install Dependencies** The project uses various libraries (Express, Mongoose, React, etc.). Install all necessary packages by running:

```
npm install
```

*Note: If the project has separate folders for frontend and backend, you may need to run npm install inside both the /backend and /frontend directories.*

**Step 4: Configure Environment Variables** Create a .env file in the backend root directory and add your MongoDB connection string and Port details:

```
Code snippet

PORT=5100

MONGO_URI=mongodb://localhost:27017/shopsmart

JWT_SECRET=your_secret_key
```

**III. Running the Application**

**Step 5: Start the Backend Server** Launch the Node.js server. Using dev mode (via Nodemon) allows the server to restart automatically when you make code changes:

```
npm run dev
```

*Wait for the console to log: "Server running on port 5100" and "MongoDB Connected".*

**Step 6: Start the Frontend Client** Open a new terminal tab, navigate to the frontend directory, and start the React development server:

```
npm start
```

**Step 7: Access the Application** Once the compilation is successful, your default browser should open automatically. If not, manually enter the following URL in your browser's address bar:

**http://localhost:5100** (or the port specified in your configuration)

You can now browse the grocery catalog, add items to your cart, and test the ShopSmart experience!

## 5. PROJECT FOLDER STRUCTURE

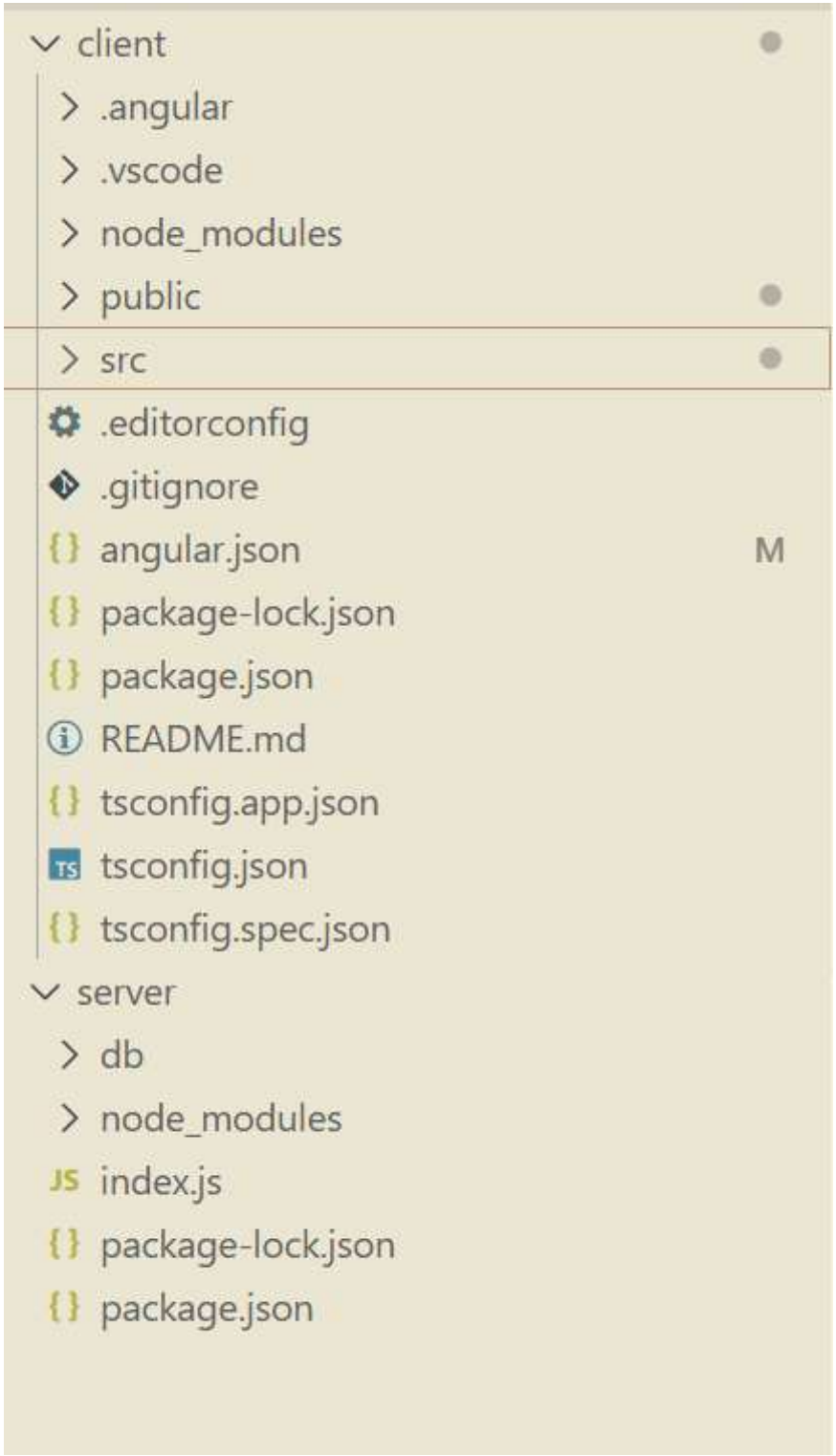
The ShopSmart project is organized into two primary directories: client (Frontend) and server (Backend). This separation ensures a clean "Separation of Concerns," making the code easier to maintain and scale.

### I. Client Folder (Frontend - React.js)

The client directory contains all the code responsible for the user interface and browser-side logic.

- **src/**: The main working directory for the React application.
- **components/**: Houses reusable UI elements used across multiple pages, such as the Navbar, Footer, ProductCard, and GroceryCategoryBar.
- **pages/**: Contains the main functional views of the application.
    - *Examples:* HomePage.js, ProductDetails.js, CartPage.js, and CheckoutPage.js.
- **services/**: Contains API calling functions (usually using Axios) that communicate with the backend server to fetch grocery data or process payments.
- **assets/**: Stores static files like the ShopSmart logo, grocery category icons (fruits, dairy, etc.), and CSS stylesheets.
- **App.js**: The root component that sets up the application routing (using React Router) and global state providers.

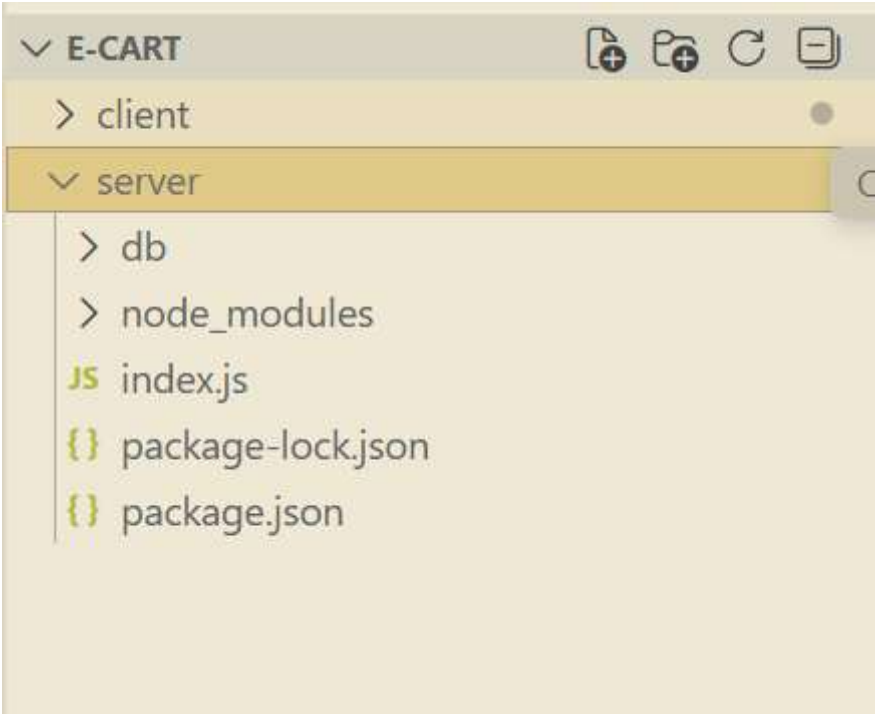
- **index.js**: The entry point that renders the React application into the HTML DOM.



### II. Server Folder (Backend - Node.js & Express)

The server directory handles the business logic, database interactions, and authentication.

- **models/**: Defines the structure of the data sent to MongoDB using Mongoose schemas.
    - *Examples:* User.js, Product.js (with fields for price, stock, and expiry), and Order.js.
- **routes/**: Defines the URL endpoints (APIs) for the application.
    - *Examples:* /api/products for fetching groceries and /api/users/login for authentication.
- **controllers/**: Contains the actual functions executed when a route is hit. For instance, the productController.js would contain the logic to "Get All Organic Vegetables."
- **middleware/**: Functions that run during the request-response cycle. Used for tasks like authMiddleware.js to check if a user is logged in before they can view their order history.
- **config/**: Contains configuration files, such as the database connection logic (db.js) and environment variable setups.
- **server.js**: The main entry point for the backend. It initializes the Express app, connects to MongoDB, and listens for incoming requests on Port 5100.




## 6. API DOCUMENTATION

The ShopSmart API is a RESTful service that enables communication between the React frontend and the MongoDB database. All requests and responses are handled in **JSON** format.

### I. Authentication & User Management

These endpoints handle user security and session management.

- **POST /api/register — Register User**
  - **Description:** Creates a new user account in the system.
  - **Input:** name, email, password, mobile, address.
  - **Response:** Success message and the created user object (excluding the password).
- **POST /api/login — Login User**
  - **Description:** Authenticates a user and starts a session.
  - **Input:** email, password.
  - **Response:** A unique **JWT (JSON Web Token)** used for authorizing subsequent requests like placing orders.

```
'/add-products', async (req, res) => {

/ Extract the product information from the request body
onst { productname, description, price, image, category, countInStock, rating } = req.body;

/ Validate if all required fields are provided
f (!productname || !description || !price || !image || !category || !countInStock || !rating)
  return res.status(400).send({ message: 'Missing required fields' });


/ Assuming models.Product and models.Category are defined and imported properly
/ Create a new product document
onst product = new models.Product({
  productname,
  description,
  price,
  image,
```

### II. Product Management

These endpoints manage the grocery catalog. Admin routes are protected and require administrative privileges.

- **GET /api/products — Get All Products**
  - **Description:** Fetches the list of all grocery items available in the store.
  - **Features:** Supports filtering by category (e.g., "Fruits", "Dairy") and search queries.
- **POST /api/products — Add Product (Admin Only)**
  - **Description:** Allows store managers to add new grocery items.
  - **Input:** productName, description, price, category, stockQuantity, expiryDate.

server > JS index.js > mongoose

```
113  app.post('/add-category', async (req, res) => {
114      try {
115          const { category, description } = req.body;
116          if (!category) {
117              return res.status(400).send('Category and description are required');
118          }
119          const existingCategory = await models.Category.findOne({ category });
120          if (existingCategory) {
121              return res.status(400).send('Category already exists');
122          }
123          const newCategory = new models.Category({
124              category,
125              description
126          });
127          const savedCategory = await newCategory.save();
128          console.log(savedCategory, 'category created');
```

- **PUT /api/products/:id — Update Product**
  - **Description:** Modifies existing product details (e.g., updating a discount or restocking an item).
  - **Parameter:** :id (The unique MongoDB ID of the product).
- **DELETE /api/products/:id — Delete Product**
  - **Description:** Removes a product from the catalog permanently.

```
app.delete('/remove-from-cart/:id', async (req, res) => {
    const id = req.params.id;
    try {
        const result = await models.AddToCart.deleteOne({ productId: id });
        if (result.deletedCount === 0) {
            res.status(404).json({ message: `Product with id ${id} not found in the cart` });
        } else {
            res.status(200).json({ message: `Removed product with id ${id} from cart` });
        }
    } catch (error) {
        console.error(error);
        res.status(500).json({ message: 'Internal server error' });
    }
});
```

### III. Order & Transaction Management

These endpoints handle the purchasing logic and history.


Shutterstock

- **POST /api/orders — Place Order**
  - **Description:** Finalizes the checkout process. It records the items purchased, calculates the total price, and clears the user's cart.
  - **Input:** cartItems, shippingAddress, paymentMethod, totalAmount.
  - **Validation:** Checks if the requested items are still in stock before confirming.
- **GET /api/orders — Get Orders**
  - **Description:** Retrieves order history.
  - **User View:** Shows the user's personal past purchases and delivery status.
  - **Admin View:** Shows all customer orders for fulfillment and logistics management.

## 7. AUTHENTICATION & SECURITY

ShopSmart implements a robust, industry-standard security protocol to protect user data and ensure that sensitive administrative actions are restricted to authorized personnel.

### I. JSON Web Tokens (JWT)

The application uses **JWT** for maintaining stateless user sessions. This process eliminates the need for the server to store session data in memory.

- **Token Generation:** Upon a successful login, the server generates a unique, digitally signed token containing the user's ID and their role (User or Admin).
- **Token Storage:** The frontend stores this token (usually in localStorage or a HttpOnly cookie).
- **Authorization Header:** For every protected request—such as viewing order history or adding a product—the frontend sends this token in the Authorization header. The server verifies the token's validity before granting access.

### II. Password Hashing (Bcrypt)

To ensure that user passwords are never stored in plain text, ShopSmart utilizes the **Bcrypt** library.

- **Salting & Hashing:** When a user registers, Bcrypt adds a "salt" (random data) to the password and then runs it through a cryptographic hashing algorithm.
- **One-Way Protection:** Even if the database were compromised, the actual passwords cannot be reversed or read. During login, the system compares the hashed version of the entered password with the one stored in MongoDB.

### III. Role-Based Access Control (RBAC)

The system distinguishes between different user levels to maintain platform integrity. This is handled via custom **Middleware** functions in the backend.

- **User Role:**
  - **Access:** Can browse the grocery catalog, manage their own profile, add items to the cart, and view their personal order history.
  - **Restriction:** Cannot access the admin dashboard or modify the global product inventory.
- **Admin Role:**
  - **Access:** Has full control over the platform. This includes adding new grocery items, updating prices/stock, deleting products, and managing orders from all customers.
  - **Implementation:** The isAdmin middleware checks the decoded JWT payload. If the role field is not "Admin", the server returns a 403 Forbidden status code, blocking the action.

## 8. TESTING AND QUALITY ASSURANCE

To ensure that ShopSmart provides a reliable and bug-free experience for both shoppers and administrators, a multi-layered testing approach was adopted. This involved verifying individual API endpoints and validating complete end-to-end user journeys.

### I. API Testing with Postman

Before integrating the backend with the React frontend, all REST API endpoints were rigorously tested using **Postman** to ensure data integrity and correct status code responses.

- **Endpoint Validation:** Each route (e.g., GET /api/products, POST /api/orders) was tested to ensure it returns the correct JSON structure.
- **Authentication Testing:** Verified that protected routes (like the Admin dashboard) correctly reject requests that do not include a valid **JWT Bearer Token**.
- **Edge Case Handling:** Tested how the system responds to invalid inputs, such as attempting to register with an existing email or placing an order for a product that is out of stock.
- **Environment Variables:** Used Postman environments to switch seamlessly between Localhost and Production testing.

### II. Manual UI/UX Testing

Manual testing was conducted from the perspective of a real user to ensure the interface is intuitive and the transitions are smooth.

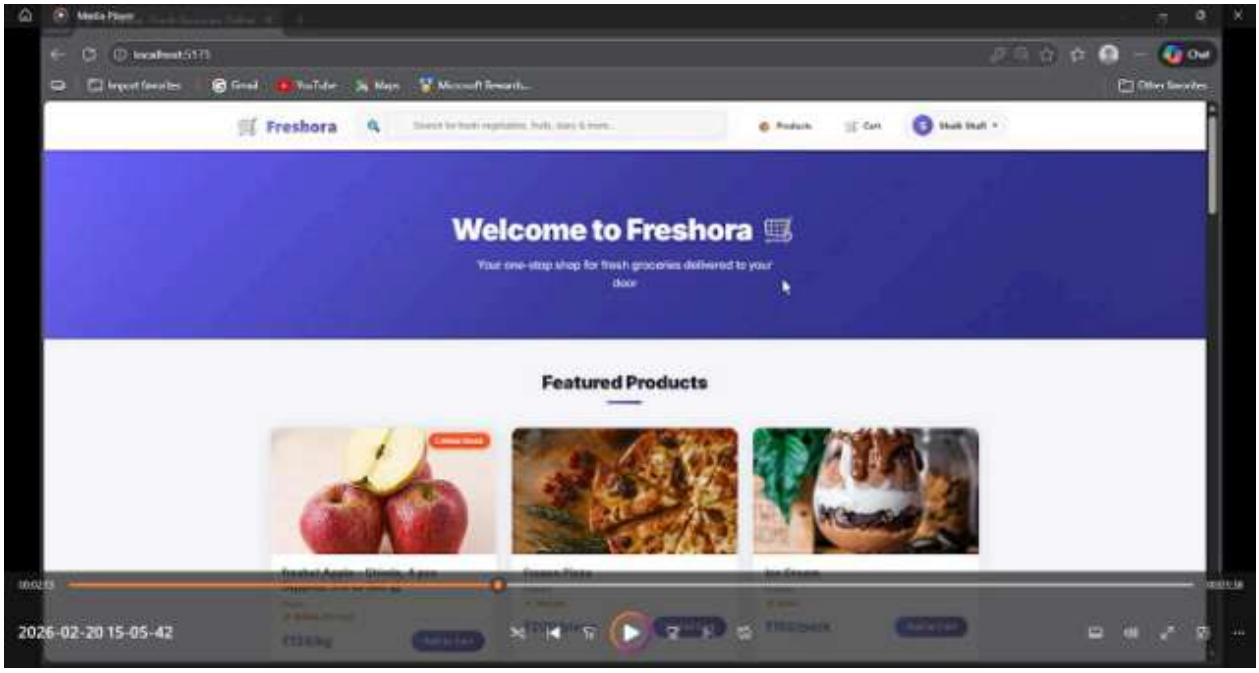
1. **Authentication Flow:** * Verified that a new user can register and is immediately redirected to the login page.
    - Confirmed that incorrect credentials trigger appropriate error messages (e.g., "Invalid Password").
2. **Cart & Inventory Management:** * Tested the "Add to Cart" functionality across multiple categories.
    - Ensured that the cart persists during the session and correctly calculates the subtotal, taxes, and discounts.
3. **Order Placement & Fulfillment:** * Simulated a complete purchase from product selection to the "Order Success" screen.
    - Verified that the stock quantity of a grocery item automatically decreases in the database once an order is confirmed.

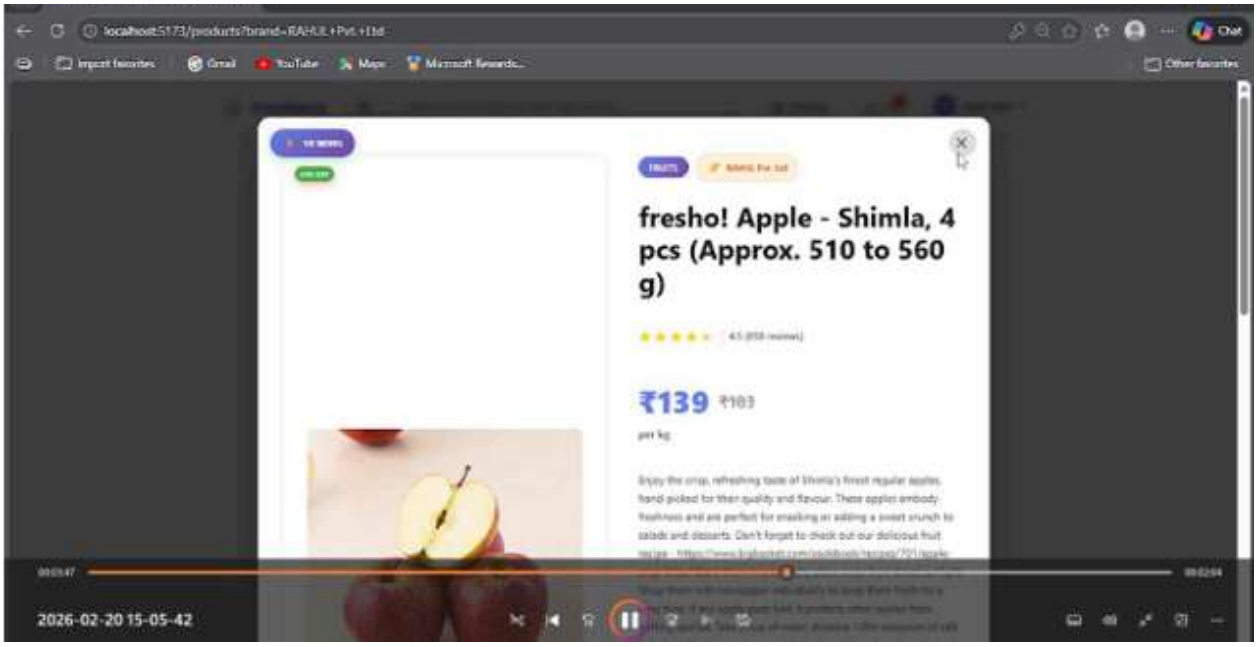
### III. Responsiveness & Compatibility Testing

Since grocery shopping often happens on the go, ShopSmart was tested across various devices and browsers.

- **Browser Compatibility:** Validated that the layout remains consistent on Chrome, Firefox, and Safari.
- **Responsive Design:** Used browser developer tools to ensure the grocery grid and navigation menu adapt correctly to mobile, tablet, and desktop screens.

# 9.Screenshots / DEMO



## 10. **FUTURE ENHANCEMENTS**

To further elevate the user experience and expand the business capabilities of ShopSmart, the following enhancements are planned for future development phases:

**I. Online Payment Integration (Stripe/Razorpay)**

Currently, the system focuses on order placement and "Cash on Delivery" or manual simulation.

- **Seamless Transactions:** Integration with gateways like **Stripe** or **Razorpay** will allow users to pay securely via Credit/Debit cards, UPI, and Digital Wallets.
- **Refund Management:** Automated refund triggers for cancelled grocery orders or out-of-stock items will improve customer trust.

**II. Automated Email Notifications (Nodemailer)**

Communication is key in grocery delivery. Integrating an email service will keep users informed at every stage.

- **Order Confirmations:** Automated emails sent immediately after a successful checkout containing the digital invoice.
- **Status Updates:** Real-time alerts when an order is "Dispatched," "Out for Delivery," or "Delivered."
- **Promotional Campaigns:** Newsletter integration to inform users about weekly discounts on fresh produce.

**III. Product Reviews and Ratings**

Social proof is essential for quality assurance in groceries.

- **Customer Feedback:** Users will be able to rate products (1-5 stars) and leave written reviews regarding the freshness of produce or the quality of packaged goods.
- **Admin Insights:** A dashboard for admins to monitor low-rated products, allowing them to switch vendors or improve quality control.

**IV. Mobile Application Version (React Native)**

To cater to the "on-the-go" shopper, a cross-platform mobile application will be developed.

- **Native Experience:** Using **React Native** to share the existing MERN logic while providing a smooth mobile UI.
- **Push Notifications:** Direct alerts to the user's phone for time-sensitive grocery deals and delivery tracking.

**V. Performance Optimization & PWA**

As the product catalog grows, maintaining speed is critical for user retention.

- **Caching:** Implementing **Redis** caching for frequently accessed product categories to reduce database load.
- **Image Optimization:** Automated compression of grocery high-resolution images to ensure fast loading on slower mobile networks.
- **Progressive Web App (PWA):** Enabling "Add to Home Screen" functionality and basic offline browsing for the grocery list.