Full Stack Development with MERN

Project Documentation

1. Introduction

This section provides an overview of the project title and team members.

- **Project Title:**
  ShopSmart – a full-stack web application aimed at showcasing trending products in an interactive, responsive storefront interface.
- **Team Members:**
  Contributors of the project:
  - Kavali Gnananjali
  - Alekhya Ravuri
  - Kasiadennagari Muni Lakshmi
  - Bodala Tejasree

---

2. Project Overview

✅ Purpose

- The goal is to provide a digital storefront experience.
- Users can browse, filter, and shop trending items.
- Built as a demonstration of modern full-stack development.

✅ Features

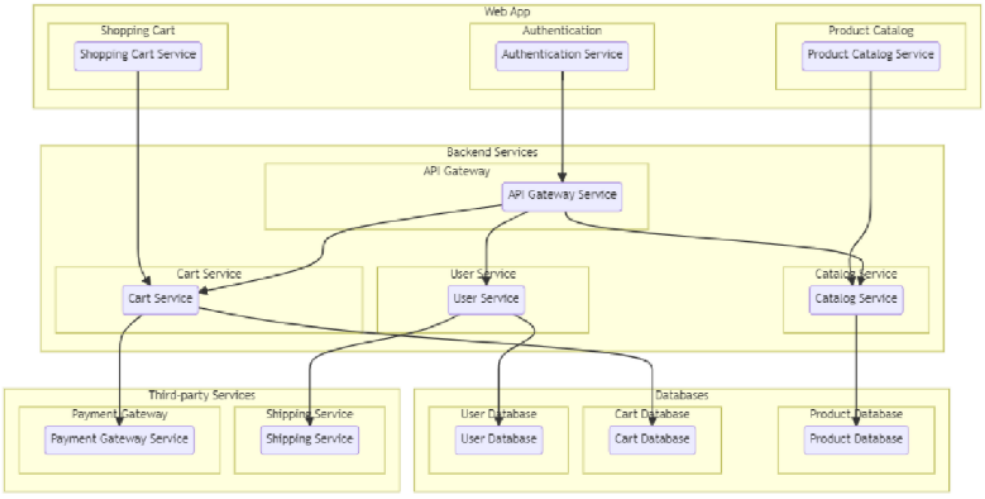
A summary of functionality:

- Responsive UI with Tailwind CSS
- Product browsing grid with filters
- Floating shopping cart with modal view
- Checkout and login pages
- API backend to serve data
- Frontend and backend separated for scalability

---

3. Architecture

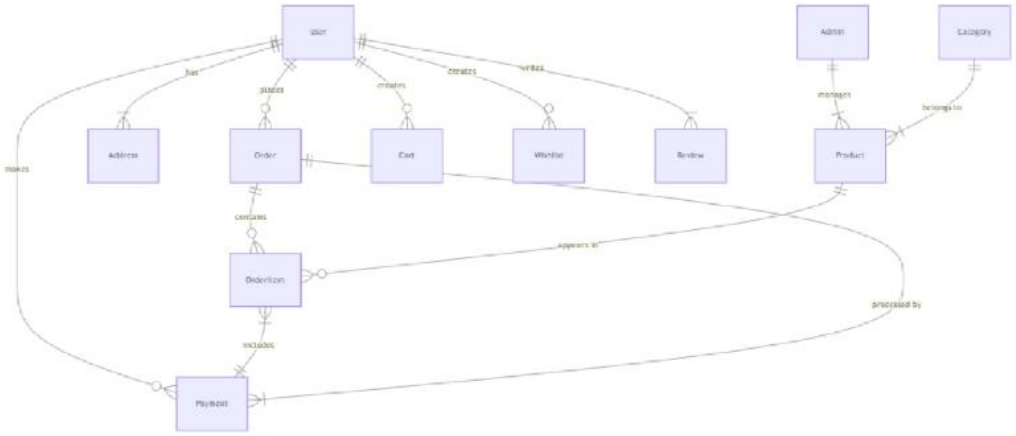
Technical Architecture:-



The technical architecture of an grocery-webapp 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 technical architecture of an grocery-webapp 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.

✅ Frontend

- Built using **React** + **TypeScript**
- **Vite** is used for faster development and builds.

- **Tailwind CSS** provides utility-first styling.
- UI is component-driven with context management (cart-context.tsx)
- Routes and views are organized in pages/

✅ **Backend**

- Built with **Node.js** and **Express**
- Written in **TypeScript**
- Core logic and routes located in server/
- APIs handle cart, product listings, etc.
- **Drizzle ORM** is used for database interaction

✅ **Database**

- The schema is defined in shared/schema.ts
- Supports relational data (PostgreSQL or SQLite likely)
- Managed via **Drizzle**, a typesafe ORM

---

4. Setup Instructions

✅ **Prerequisites**

Before running the project, you should install:

- Node.js
- npm
- Git
- Database system (if applicable)

✅ **Installation Steps**

```
git clone https://github.com/Gnananjali/ShopSmart-Your-Digital-Grocery-Store-Experience/tree/main
cd ShopSmart

# Set up frontend
cd frontend
npm install
npm run dev

# Set up backend
cd ../backend
npm install   # Or npm init -y if no package.json exists
```

---

5. Folder Structure

📁 frontend/

- **client/** – Contains all source code
  - components/ – Reusable UI elements (e.g., product cards, modals)
  - pages/ – Main screens like home, login, checkout
  - contexts/ – Global state like cart
  - hooks/ and lib/ – Custom hooks and utilities
- **Config files**:
  - vite.config.ts, tailwind.config.ts, postcss.config.js

📁 **backend/**

- **server/** – Main Express server and route logic
- **shared/** – Shared schema for database with Drizzle
- **drizzle.config.ts** – ORM configuration

📁 **common/**

- .gitignore, .replit, replit.md – Environment configs
- components.json – May relate to UI config or external libraries
- attached_assets/ – Includes documentation or text explanations

---

6. Running the Application

To run the project on your local machine:

➤ Frontend

cd frontend

npm run dev

Runs a local development server (default: http://localhost:5173)

➤ Backend

cd backend

npm start

You must have a package.json in place, or create one with npm init -y.

---

7. API Documentation

The backend of ShopSmart exposes a set of RESTful API endpoints to interact with products, the shopping cart, and user authentication. Below is a list of documented endpoints with their methods, request formats, parameters, and sample responses.

🛍️ Product APIs

GET /api/products

- **Description:** Fetch all available products.
- **Method:** GET
- **Response:**

```
[
  {
    "id": 1,
    "name": "Organic Apples",
    "price": 2.99,
    "category": "Fruits",
    "image": "/images/apples.png"
  },
  ...
]
```

GET /api/products/:id

- **Description:** Fetch a single product by its ID.
- **Method:** GET
- **Params:**
  - id (integer): Product ID
- **Response:**


```
{
 "id": 1,
 "name": "Organic Apples",
 "price": 2.99,
 "category": "Fruits"
}
```


🛒 Cart APIs

POST /api/cart

- **Description:** Add an item to the shopping cart.
- **Method:** POST
- **Body:**


```
{
 "productId": 1,
 "quantity": 2
}
```


- **Response:**


```
{
```

"message": "Product added to cart."
}

GET /api/cart

- **Description:** Retrieve the current user's cart.
- **Method:** GET
- **Response:**

[
{
"productId": 1,
"name": "Organic Apples",
"quantity": 2,
"total": 5.98
}
]

🔐 **Authentication APIs *(future enhancement)***

These are planned for future development.

POST /api/login

- **Description:** Authenticates a user and returns a token.
- **Body:** { "email": "user@example.com", "password": "******" }

POST /api/register

- **Description:** Registers a new user.

- **Body:** { "email": "user@example.com", "password": "******", "name": "John" }

---

8. Authentication

Currently:

- The frontend includes a login page (login.tsx)
- No authentication logic is implemented on the backend

Future Recommendation:

- Use **JWT** (JSON Web Tokens) for user sessions
- Implement login/register routes in Express
- Store tokens in cookies or local storage

---

9. User Interface

The UI is:

- Built with **Tailwind CSS** (responsive by default)
- Component-driven (product-card.tsx, cart-modal.tsx)
- Includes:
    - Header with navigation
    - Floating cart with modal functionality
    - Filter bar for browsing categories
    - Pages: home, login, checkout, not-found

---

10. Testing

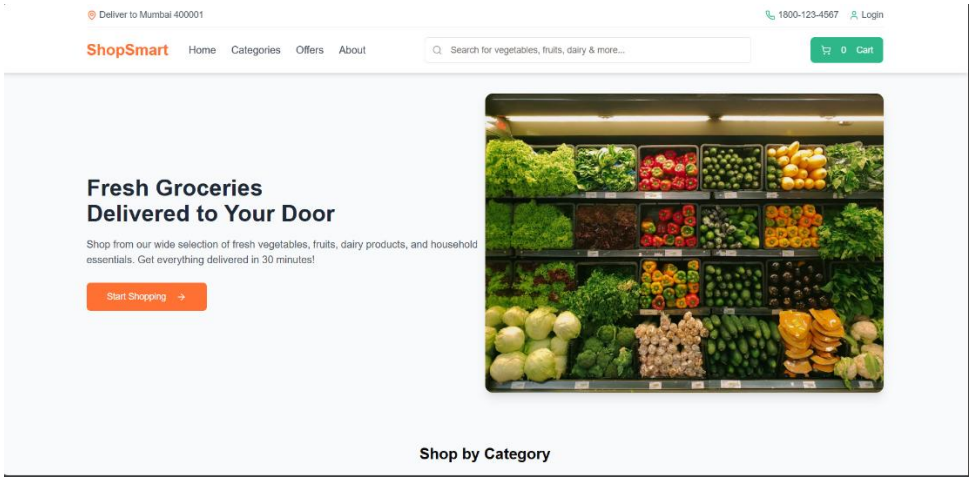
Currently not implemented, but suggestions:

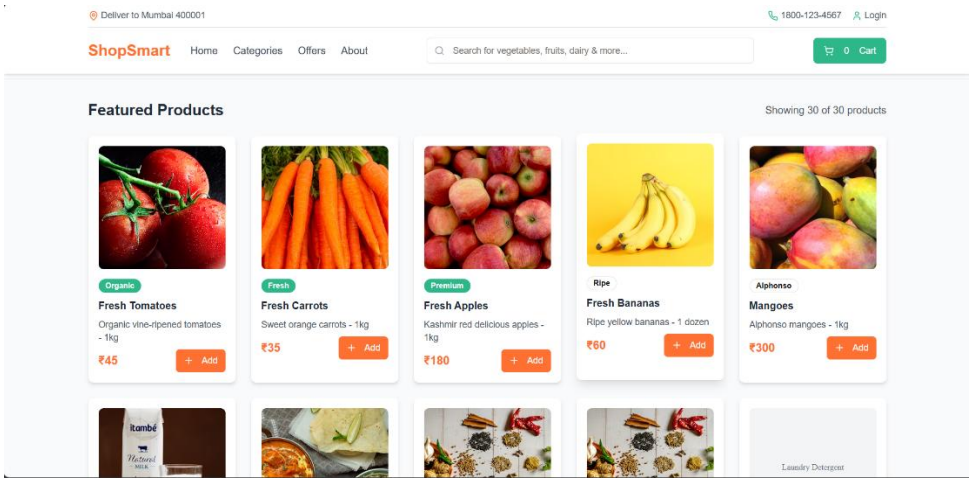
- **Frontend:** Jest + React Testing Library
- **Backend:** Supertest (for route testing), Mocha/Chai or Jest

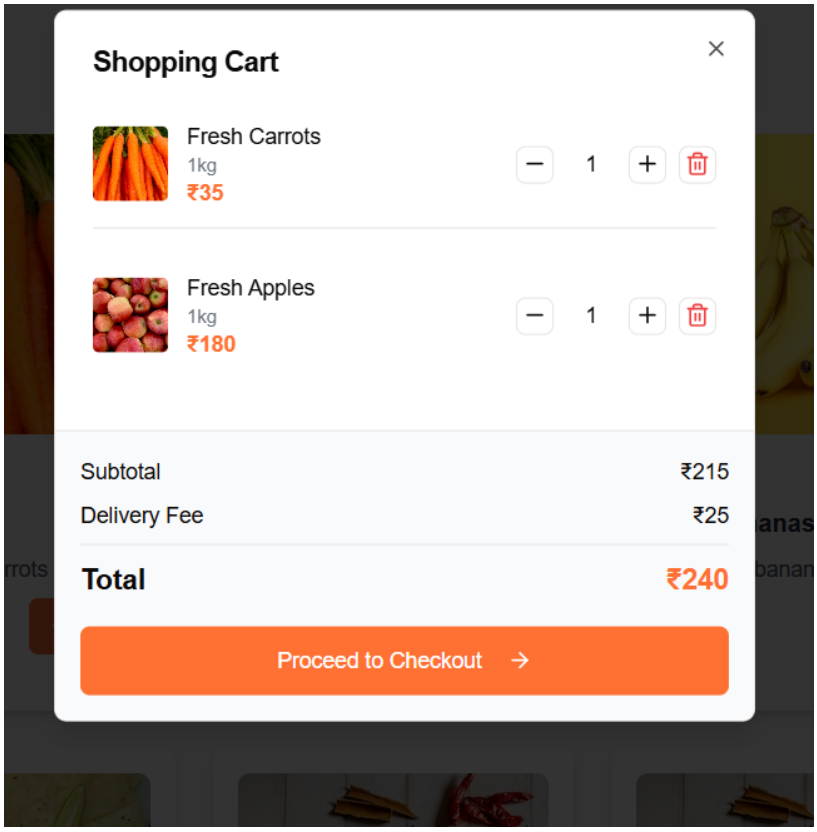
---

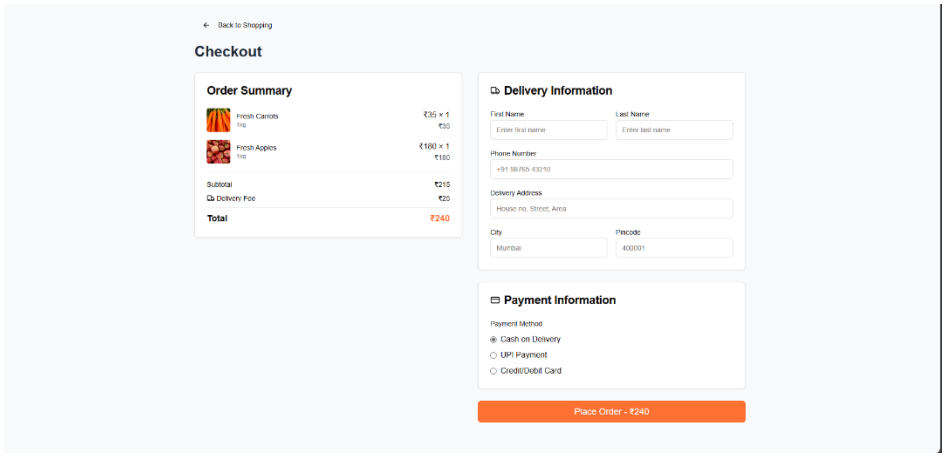
11. Screenshots or Demo

Screenshots of homepage, product grid, cart, and checkout









Demo Video Link: https://drive.google.com/file/d/1tmpoXrKH-zKP-UxBg_cRzp2zvTuK8_1I/view?usp=drive_link

---

12. Known Issues

- No package.json in backend/ by default

- Backend lacks authentication and input validation
- API responses not fully documented or standardized

---

13. Future Enhancements

- Add authentication (JWT + bcrypt or OAuth)
- Build admin dashboard for product uploads
- Enable real payment integration (e.g., Stripe)
- Implement search, sorting, and filter logic
- Write automated tests
- Set up CI/CD pipelines (GitHub Actions, Netlify, etc.)