Full Stack Development with MERN

Project Documentation

1. Introduction

The Grocery WebApp – Blinkit Clone is a full-stack e-commerce platform developed to replicate the core features of Blinkit, a popular grocery delivery service. Built using the MERN stack, the application allows users to browse products, manage carts, place orders, and make secure payments.

- **Project Title:** Grocery Webapp
- **Team Members:**

Team members	Roles
Neelam Kushwaha	Project Planning & Documentation
Somya Nigam	Documentation
Riya Mishra	Frontend & Backend, Demonstration
Deorshi Nishant	Assisted in Frontend

2. Project Overview

- **Purpose:**

  The primary goal of this project is to create an efficient and feature-rich grocery shopping web application. The app provides customers with the ability to browse grocery items, add products to a cart, and perform secure online transactions. Simultaneously, it provides sellers and administrators tools to manage inventory, track orders, and view real-time insights into user and product behavior.
- **Features:**
  - User registration and login using JWT authentication
  - Product browsing with search and category filters
  - Add-to-cart and dynamic cart updates
  - Razorpay payment gateway integration
  - Seller interface to manage product listings and stock
  - Admin dashboard for monitoring users and orders
  - Responsive design compatible with desktops, tablets, and mobile devices

3. Architecture

- **Frontend:**

  The frontend is developed using **React.js** with routing support via React Router DOM. UI components are designed to be reusable and styled using CSS modules. Axios is used to handle communication with backend APIs. React Hooks like `useState` and `useEffect` help manage application state and lifecycle events

efficiently.

- **Backend:**
  The backend is developed using **Node.js** and **Express.js**. It handles routing, middleware configuration, and RESTful API logic. The application includes user authentication, role-based authorization, cart processing, and order handling. Modular controller files improve maintainability.

- **Database:**
  MongoDB Atlas is used for cloud-based storage. Collections include `users`, `products`, `orders`, and `cartItems`. **Mongoose** is used to define schemas and interact with the database securely. It allows for validations, schema-level hooks, and efficient data querying.

4. Setup Instructions

- **Prerequisites:**
  - Node.js v16+
  - MongoDB Atlas Account
  - Git
  - Code Editor (VS Code)

- **Installation:**
- Clone the GitHub repository.
- Navigate to `client` and `server` folders.
- Install dependencies:

```
npm install
```

4. Set up environment variables in `server/.env`:
   - `MONGO_URI=your_mongo_connection_string`
   - `JWT_SECRET=your_jwt_secret`
   - `RAZORPAY_KEY_ID=your_razorpay_key_id`
   - `RAZORPAY_KEY_SECRET=your_razorpay_secret`
5. Start the backend server:

```
cd server
npm start
```

6. Start the frontend client:

```
cd client
npm start
```

7. Navigate to `http://localhost:3000` in the browser.

5. Folder Structure

Client (React Frontend):

- `src/`
  - `components/` – Contains reusable UI components like Navbar, Footer, ProductCard, and Loader.
  - `pages/` – Contains screen-level components like Home, Cart, Login, Register, and Seller Dashboard.
  - `services/` – Includes helper functions for making API requests using Axios.
  - `context/` – Manages global states for user authentication and cart using React Context API.
  - `App.js` – Root component with routing setup.
  - `index.js` – Entry point for rendering the app into the DOM.

Server (Node.js Backend):

- `routes/` – Contains all Express route definitions for users, products, cart, and admin.
- `controllers/` – Handles the logic for each route, separates concerns from routing logic.
- `models/` – Defines Mongoose schemas for User, Product, Order, and Cart collections.
- `middleware/` – Authentication middleware and error handling utilities.
- `config/` – MongoDB connection and Razorpay integration setup.
- `server.js` – Main entry point to start the backend server.

6. Running the Application

- Provide commands to start the frontend and backend servers locally.
  - **Frontend:** `npm start` in the client directory.

```
cd client
npm install
npm start
```

  - **Backend:** `npm start` in the server directory.

```
cd server
npm install
npm start
```

7. API Documentation

Endpoint	Method	Description
`/api/auth/register`	POST	Register new users
`/api/auth/login`	POST	Login and return JWT token

`/api/products`	GET	List all products
`/api/products/:id`	GET	Get single product details
`/api/cart/add`	POST	Add item to user's cart
`/api/cart/remove`	DELETE	Remove item from user's cart
`/api/checkout`	POST	Initiate payment via Razorpay
`/api/admin/users`	GET	Admin-only: View all registered users

8. Authentication

- JWT-based login system
- After successful login, token is stored in localStorage
- Each protected route requires `Authorization: Bearer <token>` header
- Middleware checks user roles for `admin`, `seller`, or `customer`
- Passwords stored as bcrypt hashes.

9. User Interface

The UI is designed to offer ease of navigation and accessibility:

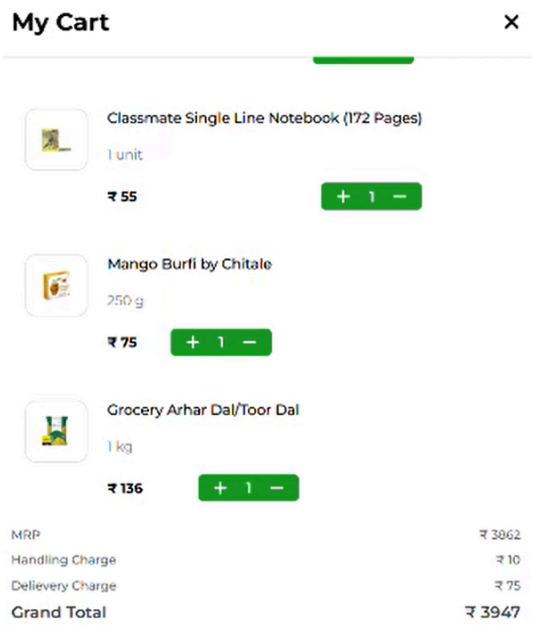
- **Navbar** – Dynamic based on user login status
- **Homepage** – Lists all groceries by category
- **Search Bar** – Filter by name
- **Cart** – Preview, update quantities, remove items
- **Checkout** – Address input, order summary, Razorpay popup
- **Seller Dashboard** – Add/edit/delete products
- **Admin Panel** – View users, monitor orders, system usage

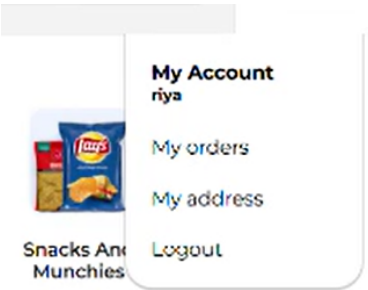
10. Testing

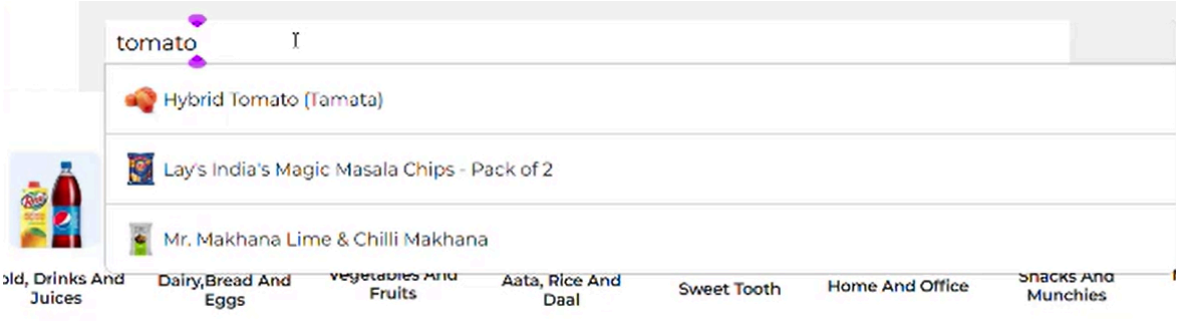
Testing Approaches:

- **Functional Testing** – Manual verification of flows
- **API Testing** – Used Postman collections to verify endpoint integrity
- **Performance Testing** – Load testing with Apache JMeter
- **Frontend Audit** – Google Lighthouse for speed and accessibility scores

11. Screenshots or Demo







https://drive.google.com/file/d/1UkKI_hIoIsxFNwyq0zidZ9EqUfoO9K1l/view?usp=sharing

12. Known Issues

- Voice search depends on browser capabilities.

13. Future Enhancements

To make the Grocery WebApp more scalable, intelligent, and competitive with leading grocery platforms, the following enhancements are proposed for future development. These upgrades aim to increase user engagement, streamline backend operations, and improve the overall performance and accessibility of the system.

- **Mobile App:** Develop native mobile app using React Native
- **Order Tracking:** Real-time delivery tracking and status updates
- **Notification System:** Email/SMS confirmation post checkout
- **Promo Engine:** Add coupon codes and discounts
- **Advanced Roles:** Fine-grained access control for admins and sellers
- **Analytics:** Dashboard for sales insights and visual reports
- **Chatbot:** Integrate AI-based support assistant for customers