

# FULL STACK DEVELOPMENT WITH MERN

## INTRODUCTION

### FRESH MART- Grocery WebApp

(Revolutionizing Your Online Food Ordering Experience)

#### TEAM MEMBERS:

MAGESH M	311421104044
GOUTHAM J	311421104023
HARISH K	311421104029
ANANDHA SHANMUGA SUNDARAM V	311421104004
BAVITHRAN V M	311421104010

# PROJECT OVERVIEW

## Purpose

Fresh Mart (grocery web app) is developed using the MERN stack (MongoDB, Express.js, React.js, and Node.js) aims to provide a seamless experience for customers to shop for groceries online and for administrators to manage inventory and orders efficiently.

## Features

- **User Authentication** : Secure user registration and login for both customers and administrators.
- **Product Listings** : A grid layout displaying categorized grocery items with essential details (name, price, image).
- **Carousel Banner** : A visually appealing image slider on the homepage to highlight deals or featured products.
- **Responsive Web Design** : A modern web design strategy that aims to provide a consistent experience across all devices.
- **Shopping Cart** : Add, update, or remove items in the cart with real-time calculations.
- **Checkout Process** : Simplified checkout workflow for placing orders.
- **Admin Panel** : Role-based features to manage inventory, categories, and user accounts.
- **User accounts & Profile** : Users can create accounts, save cart item details for a smoother experience.

## Skills Developed

- Backend API development with Node.js and Express for user authentication and CRUD operations.
- Frontend design and interactivity with React, focusing on reusable components and state management (e.g., using useContext hook).
- Integration of MongoDB for managing a dynamic database of products, users, and orders.
- Implementation of responsive and minimalistic UI for an optimal user experience across devices.
- Role-based access control for secure admin operations.

# ARCHITECTURE

The Fresh Mart App is designed with a modular architecture, consisting of three main components: Frontend, Backend, and Database. This structure ensures scalability, flexibility, and ease of maintenance.

## 1. Frontend (React + Vite)

The frontend, developed with React, is responsible for delivering a responsive, intuitive, and dynamic user experience. While Vite is implemented for faster and efficient build time and compiling efficiency.

Features and Design:

- **Component-Based Architecture:**
  - Key reusable components include Navbar, Cart Items Display, Product Display, Footer, etc
  - Promotes modularity, scalability, and easier debugging.
- **State Management:**
  - `useContext` react hook is used for managing global state across components, such as user authentication, shopping cart, and order history.
  - Local state is used for individual component interactions like input handling or UI toggles.
- **Routing:**
  - React Router handles multi-page navigation seamlessly without page reloads, supporting key routes:

> /	-	Home page route
> /sign-in	-	Sign In/Up page route
> /cart	-	Cart page route
> /my-orders	-	Orders page route
> /profile	-	Viewing User Profile route
> /item-view/:id	-	Dynamic route for Item page view
- **API Integration:**
  - Uses Axios to interact with backend APIs for fetching product data, submitting orders, and managing cart contents.
- **Responsive Design:**
  - Built using CSS frameworks like Tailwind CSS to ensure mobile-first and desktop-friendly layouts.

## 2. Backend (Node.js and Express.js)

The backend handles all business logic, API routing, and interactions with the MongoDB database. Here, we also use several packages such as mongoose, multer, jwt to handle complex application logic

Core Features:

- **RESTful API Design:**
  - Exposes endpoints to manage users, products, carts, and orders.
  - Example API endpoints:
    - GET /api/products - Fetch all products.
    - POST /api/user - User login.
    - POST /api/payments - Payment processing.
- **Authentication & Authorization:**
  - JWT (JSON Web Tokens) ensures secure user authentication.
  - Role-based access control:
    - Admin users can manage products and orders.
    - Regular users can browse, shop, and place orders.
- **Middleware:**
  - Custom middleware handles:
    - User and Admin Authorization : Verifies user and admin tokens for secure access.
    - Validation : Validates API input data.
    - Error Handling : Provides consistent error responses.
    - Multer : Handles file uploads.
- **Scalable Modular Design:**
  - Routes (/routes) define API paths and map them to specific controllers.
  - Controllers (/controllers) handle business logic and database interactions.
  - Middleware (/middlewares) handles communications and allows to work-together.
  - Config (/config) consists a hub where you can gather all the important configurations and settings.
  - Models (/models) defines the properties, behaviour, and interactions of a specific entity.

### 3. Admin (React + Vite)

The admin panel, developed as a standalone application using React and Vite, focuses on providing administrators with tools to manage the grocery store efficiently. Its independent build and deployment ensure modularity and optimized performance for admin-specific tasks.

#### Features and Design:

##### 1. Component-Based Architecture

- Key reusable components include:
  - Sidebar: Navigation menu for managing different sections of the admin dashboard.
  - Product Management: Displays a list of products with options to add, edit, or delete items.
  - Order Management: Tracks and updates customer orders and statuses.
  - User Management: Displays user accounts and roles for administrative oversight.

##### 2. Routing

- React Router enables multi-page navigation with key admin-specific routes:
  - > /admin/list - Overview and Managing of store products.
  - > /admin/add - Add a new product to the store.
  - > /admin/orders - Monitor and update customer orders.

##### 3. API Integration

- Axios is used for secure communication with backend APIs to:
  - Manage product inventory (add, update, delete).
  - Track and update order statuses.
  - Perform user management functions, including role-based access.

##### 4. Responsive Design

- Built using CSS frameworks such as Tailwind CSS, Bootstrap, or Material-UI to ensure a consistent and responsive experience across devices.
- Optimized for desktop screens, where most admin tasks are performed.

## 4. Database (MongoDB Atlas)

The database, powered by MongoDB, is designed to handle flexible, dynamic, and hierarchical data structures.

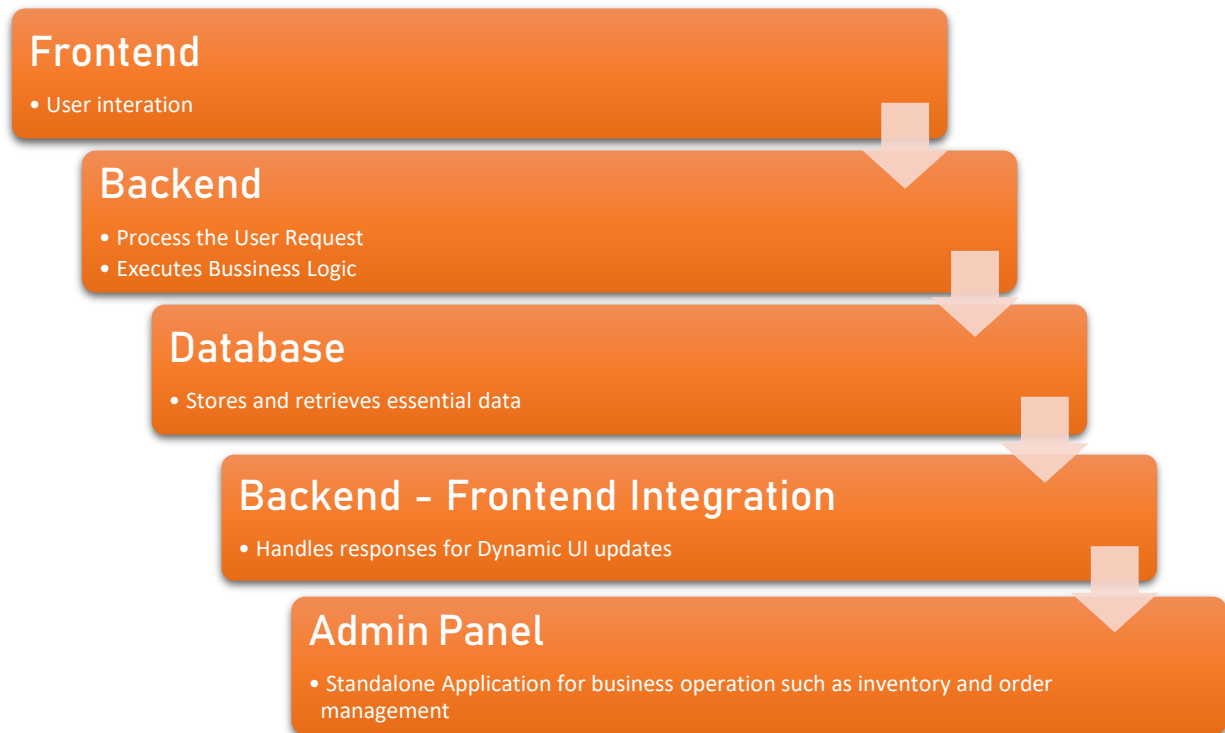
Key Features:

- Collections:
  - Users : Stores user information, hashed passwords, and roles (e.g., admin or customer).
  - Products : Contains product details like name, price, description, category, stock, and images.
  - Orders : Tracks user orders, including items, quantities, total price, and order status.
- Relationships:
  - Documents reference one another using Object IDs, e.g.:
    - Orders reference User and Product collections.
    - Cart references individual user data.
- Data Operations:
  - CRUD operations are managed with Mongoose, which ensures schema validation and efficient interaction:
    - Create: Add new users, products, or orders.
    - Read: Fetch product lists, user details, or order history.
    - Update: Modify cart items, product stock, or order status.
    - Delete: Remove obsolete items or clear user carts.

High-Level Flow:

1. Frontend Requests: Users interact with the React-based UI to browse products, add items to their cart, or place orders.
2. API Calls: The frontend sends HTTP requests to backend APIs using Axios.
3. Backend Processing: Node.js/Express.js handles requests, performs business logic, and interacts with MongoDB for data operations.
4. Database Operations: MongoDB executes CRUD actions and sends results back to the backend.
5. Frontend Updates: The backend returns data, which the frontend displays to the user in real-time.

## SYSTEM FLOW



## SETUP INSTRUCTIONS

### Setup Instructions for Fresh Mart grocery webapp

This guide outlines the steps to install, configure, and run the Fresh Mart app on your local machine. The project includes separate builds for frontend, backend, and admin panel, each running on different ports.

### Prerequisites

Before starting, ensure the following software is installed:

- Node.js (LTS version recommended, includes npm or Yarn)
- MongoDB (Using MongoDB Atlas)
- Git
- Code Editor (e.g., Visual Studio Code)

## Installation

### Step 1: Clone the Repository

1. Open a terminal and navigate to your desired directory.
2. Clone the below repository:

```
https://github.com/mightbegood12/Groceries
```

### Step 2: Install Dependencies

#### Frontend (React + Vite)

- > Navigate to the frontend directory: `cd frontend`
- > Install the required packages: `npm install`

#### Backend (Node.js + Express.js)

- > Navigate to the backend directory: `cd ../backend`
- > Install the required packages: `npm install`

#### Admin Panel (React + Vite)

- > Navigate to the admin directory: `cd ../admin`
- > Install the required packages: `npm install`

## Starting the Application

#### Frontend

- > `npm run dev`
- > The frontend will run on <http://localhost:3000>

#### Backend

- > `npm run server`
- > The backend will run on <http://localhost:4000>

#### Admin Panel

- > `npm run dev`
- > The admin panel will run on <http://localhost:5000>



# FOLDER STRUCTURE

The project is divided into three main parts: the Client (React frontend), the Server (Node.js backend). Below is the breakdown of the folder structure:

## Client (React + Vite Frontend)

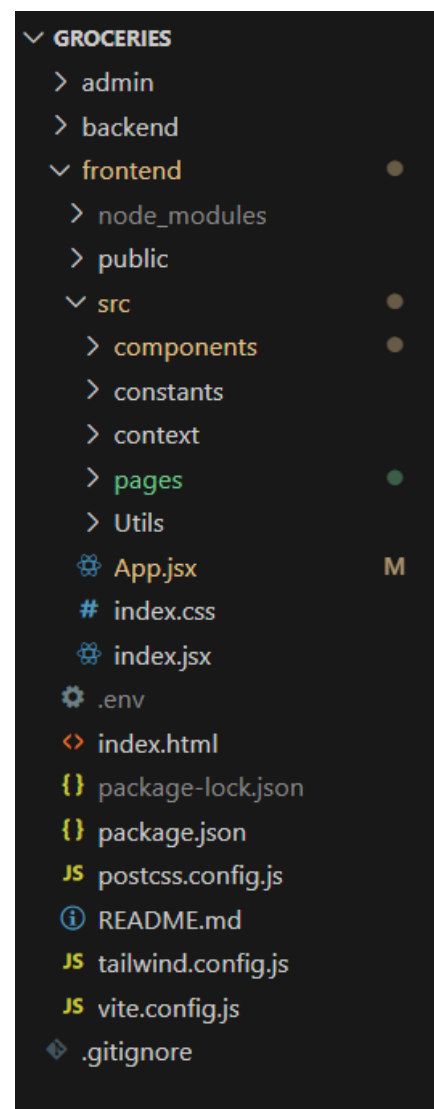
The React frontend handles the user interface and interaction with the backend API. The main structure is as follows:

### ❖ public/:

- Contains the `index.html` file, which serves as the entry point for the React app.
- Static assets like the favicon and manifest for Progressive Web Apps (PWA) are stored here.

### ❖ src/:

- `components/`
  - Houses reusable UI components like Navbar, Footer, and `ProductDisplay.jsx`.
- `pages/`
  - Contains page-level components for each route (e.g., `Home.jsx`, `Product.jsx`).
  - Each page corresponds to a feature or view in the app.
- `context/`
  - Implements React Context for state management (`CartContext` for sharing cart data across components).
- `constant/` and `Utils/`
  - Consists of constants used throughout the app (such as Currency, Site Charges, etc)
  - Also consists of necessary utility functions used (such as `ScrollToTop`, etc).
- `App.jsx`
  - The main component that wraps the app with necessary providers and renders routes.
- `index.jsx`
  - The entry point for the React app that renders `App.js` into the DOM.



## Server (Node.js + Express Backend)

The Node.js and express backend handles API requests, business logic, and database interactions. The structure is organized for clarity and modularity.

### ❖ controllers/

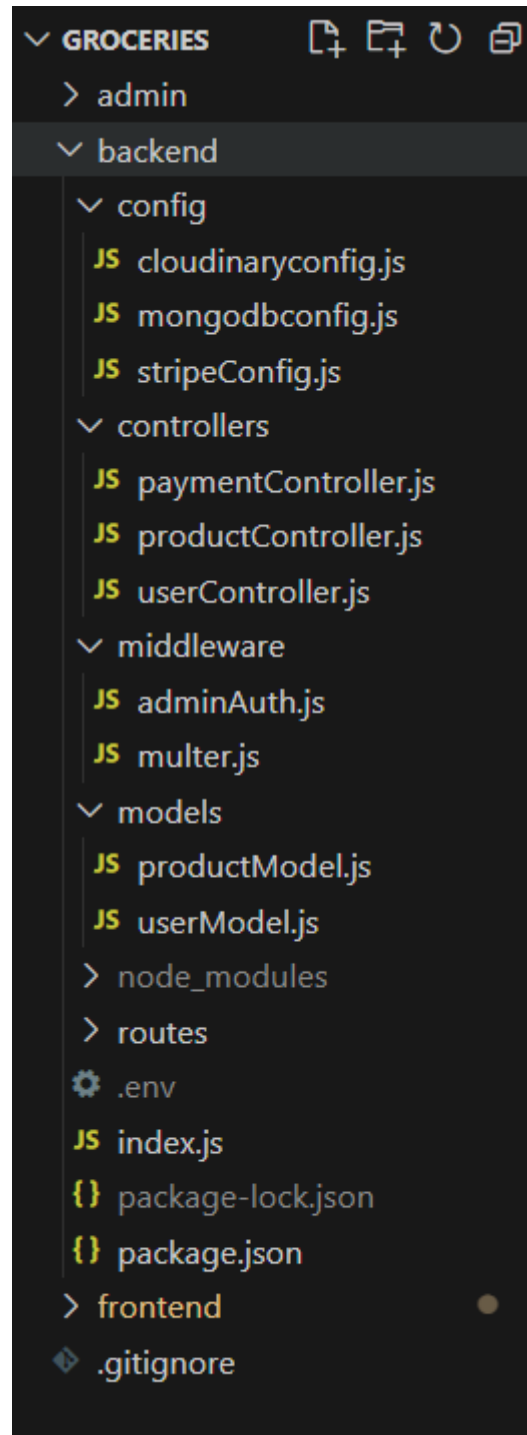
- Contains logic for handling API requests and responses.
- Example:
  - `UserController.js`: Handles user authentication (signup, login).
  - `productController.js`: Manages product-related operations (create, update, fetch).
  - `orderController.js`: Processes orders and manages their status.
- Keeps route definitions clean by separating business logic.

### ❖ models/

- Defines Mongoose schemas for MongoDB collections.
- Example:
  - `User.js`: Schema for user data (name, email, password, etc.).
  - `Product.js`: Schema for product data (name, price, stock, etc.).
  - `Order.js`: Schema for orders (user, product list, status, etc.).

### ❖ routes/

- Contains route definitions for different endpoints.
- Example:
  - `authRoutes.js`: Defines routes for `/api/user` (e.g., login, signup).
  - `productRoutes.js`: Defines routes for `/api/products` (e.g., fetch products, search by category).
  - `orderRoutes.js`: Defines routes for `/api/orders` (e.g., place an order, fetch user orders).



### ❖ middleware/

- Holds middleware functions for handling authentication, error handling, etc.
- Example:
  - adminAuth.js: Verifies JWT tokens for protected routes.
  - multer.js: For handling file uploads in the application.

### ❖ config/

- Stores configuration files.
- Example:
  - mongodbconfig.js: Handles the MongoDB connection.
  - clouldinaryconfig.js & stripeconfig.js : Manages external keys

### ❖ server.js

- The entry point for the backend server.
- Sets up the Express app, connects to MongoDB, and defines middleware.

### ❖ .env:

- Stores sensitive environment variables (e.g., database URI, JWT secret).

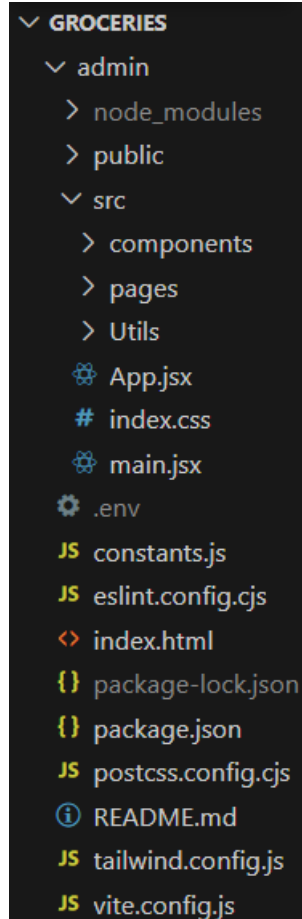
## Admin (React + Vite Frontend)

### ❖ public/:

- Contains the index.html file, which serves as the entry point for the React app.
- Static assets like the favicon and manifest for Progressive Web Apps (PWA) are stored here.

### ❖ src/:

- components/
  - Houses reusable UI components like Navbar, Sidebar, and ProductUnit.jsx.
- pages/
  - Contains page-level components for each route (e.g., AdminHome.jsx, Add.jsx, etc).
  - Each page corresponds to a feature or view in the app.



```

GROceries
├── admin
│   ├── node_modules
│   ├── public
│   └── src
│       ├── components
│       ├── pages
│       └── Utils
├── App.jsx
├── index.css
├── main.jsx
├── .env
├── constants.js
├── eslint.config.cjs
├── index.html
├── package-lock.json
├── package.json
├── postcss.config.cjs
├── README.md
├── tailwind.config.js
└── vite.config.js

```

# RUNNING THE APPLICATION

## Start the Frontend Client

- > npm run dev
- > The frontend will run on <http://localhost:3000>

## Start the Backend Server

- > npm run server
- > The server will run on <http://localhost:4000>

## Expected Output in the Browser

- You should see the Fresh Mart homepage with options to browse products, log in, or sign up.

## Simultaneous Backend and Frontend Execution

Ensure both servers are running simultaneously:

- Backend: <http://localhost:4000> (API server)
- Frontend: <http://localhost:3000> (React app)

## Testing the Application Locally

- Home Page: Verify that product listings load correctly (data fetched from the backend).
- User Authentication: Test signup, login, and token-based authentication.
- Cart and Checkout: Add products to the cart and place an order.
- Admin Features: Access the admin panel (if applicable)

## Troubleshooting

- If the backend server fails to connect to MongoDB:
  - Ensure MongoDB is running locally or the MongoDB Atlas URI is correctly configured in the MONGO\_URI variable in the .env file.
- If the frontend doesn't start:
  - Check for dependency installation errors in the frontend directory (npm install).
- Cross-Origin Issues (CORS):
  - Ensure the backend includes proper CORS headers using the cors middleware.
- Toast or Console logs:
  - Check for any logs or error messages in the application to pinpoint the error.

# API DOCUMENTATION

This project provides a RESTful API for managing products, orders, and user authentication. The API follows a consistent structure with standard HTTP methods and status codes.

## 1. Authentication Endpoints

### User Registration

- Endpoint: `/api/user/register`
- Method: POST
- This endpoint allows users to register by providing their name, email, and password.
- Response: On successful registration, a message confirms the user has been registered along with user details like name, email, and role.

### User Login

- Endpoint: `/api/user/login`
- Method: POST
- This endpoint allows users to log in by providing their email and password.
- Response: Upon successful login, the server returns a JWT token that the user can use for authentication on future requests.

## 2. Product Endpoints

### Get All Products

- Endpoint: `/api/product/list`
- Method: GET
- This endpoint fetches all products available in the store. Users can optionally filter by product category or search for a specific product.
- Response: A list of products with details such as product name, description, price, and category.

### Get Product by ID

- Endpoint: `/api/product/:id`
- Method: GET
- This endpoint fetches the details of a specific product by its unique ID.
- Response: Detailed information of the product including name, description, price, and category.

### 3. Cart Endpoints

#### Add Item to Cart

- Endpoint: /api/cart
- Method: POST
- This endpoint allows users to add a product to their cart by specifying the product ID and the desired quantity.
- Response: A confirmation message stating that the item has been added to the cart.

#### Remove Item from Cart

- Endpoint: /api/cart/:id
- Method: DELETE
- This endpoint allows users to remove a product from their cart by providing the cart item ID.
- Response: A message confirming that the item has been removed from the cart.

### 4. Order Endpoints

#### Place Order

- Endpoint: /api/orders/create
- Method: POST
- This endpoint allows users to place an order after reviewing their cart. The request should include the shipping address and payment method.
- Response: A message confirming the order has been placed, along with order details like total price, shipping address, and payment method.

#### Get All Orders (Admin)

- Endpoint: /api/orders/list-order
- Method: GET
- This endpoint fetches all orders for admin users, displaying order details like status, total price, and shipping address.
- Response: A list of orders placed by customers with order details.

#### Get Order by ID (User)

- Endpoint: /api/orders/getOrder
- Method: POST
- This endpoint fetches details of a specific order by order ID.
- Response: Order details for the specified order, such as status, total price, and shipping information.

# TESTING

## Testing Strategy for the Fresh Mart Grocery App using MERN Stack

### Testing Types:

1. **Unit Testing:** Focuses on testing individual components or functions.
2. **Integration Testing:** Ensures proper interaction between different parts of the system (frontend, backend, and database).
3. **End-to-End (E2E) Testing:** Simulates real user scenarios to check the entire workflow of the app.
4. **User Acceptance Testing (UAT):** Validates the app against user requirements.

### Testing Tools:

- **Frontend:**
  - **Jest:** For unit and integration tests of React components.
  - **React Testing Library:** For testing the UI and simulating user interactions.
  - **Cypress:** For E2E testing, simulating user flows from start to finish.
- **Backend:**
  - **Mocha & Chai:** For testing API endpoints and backend logic.
  - **Supertest:** For testing HTTP requests and responses.
- **Database:**
  - **MongoDB Memory Server:** For in-memory database testing.
  - **Sinon:** For mocking database or external service calls.

### Testing Workflow:

1. **Unit Tests:** Test components and API functions.
2. **Integration Tests:** Test interactions between frontend and backend.
3. **E2E Tests:** Test user flows like browsing products, adding to cart, and placing an order.
4. **Continuous Integration:** Automatically run tests using CI tools like GitHub Actions or Jenkins.

# SCREENSHOTS & DEMO

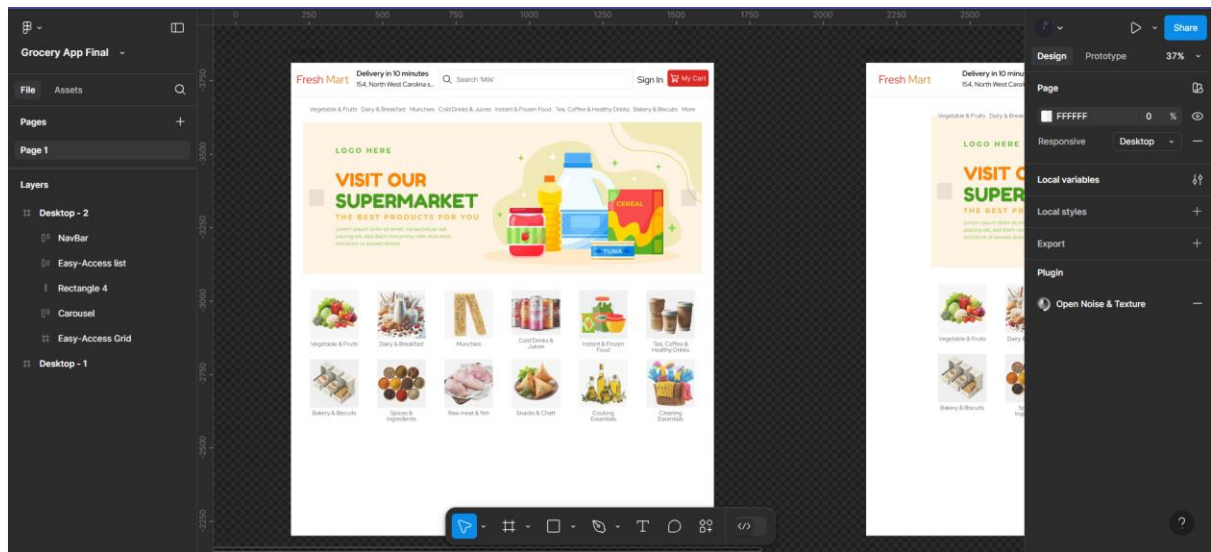


Fig 0 UI/UX Figma Design

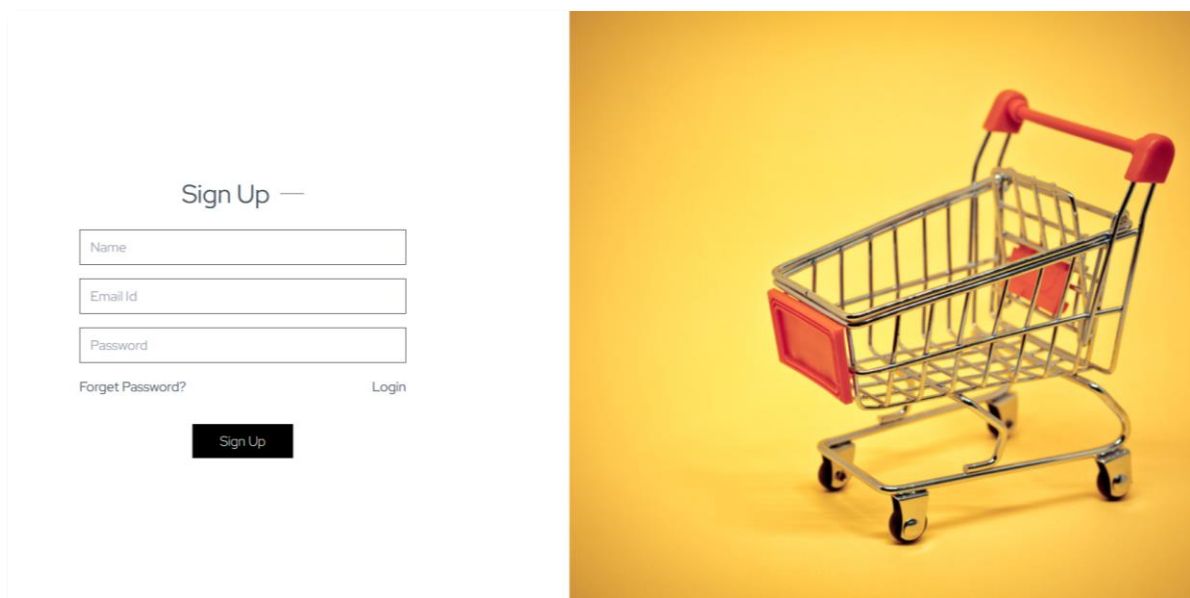


Fig 1 Sign Up / Register Page



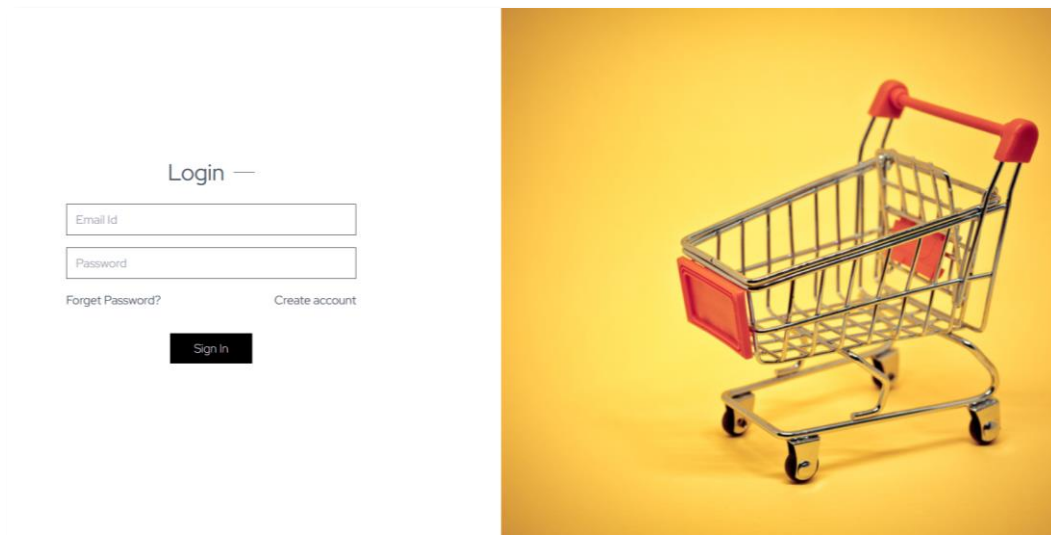


Fig 2 Login Page

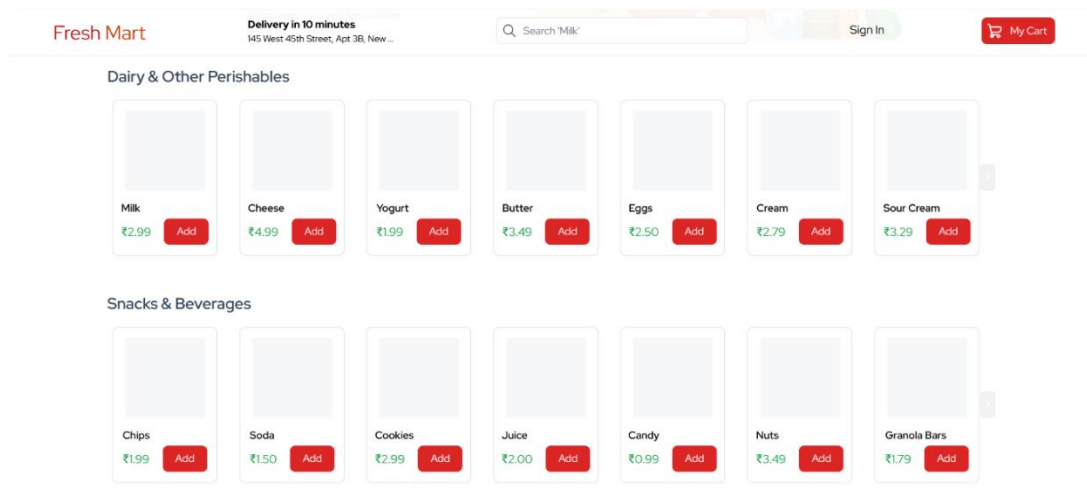


Fig 3 Home Page

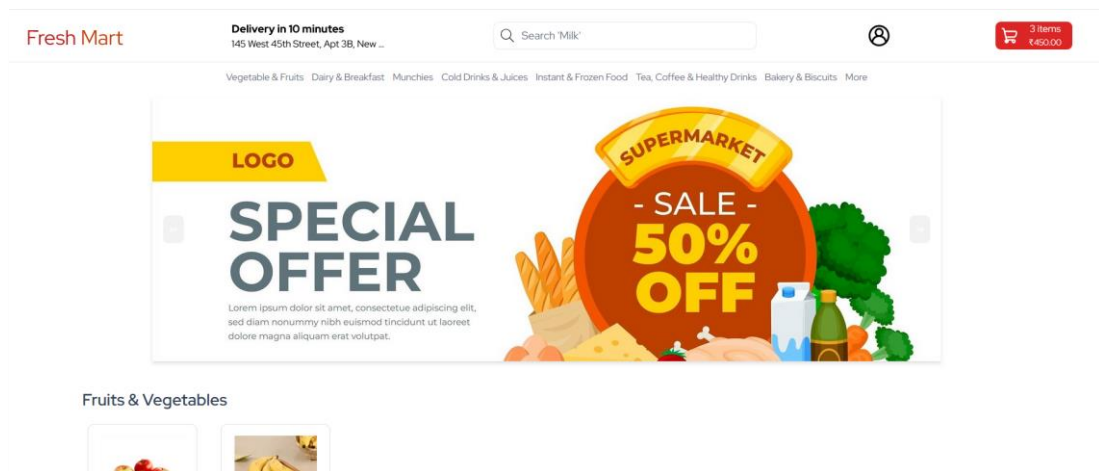


Fig 4 Landing Page

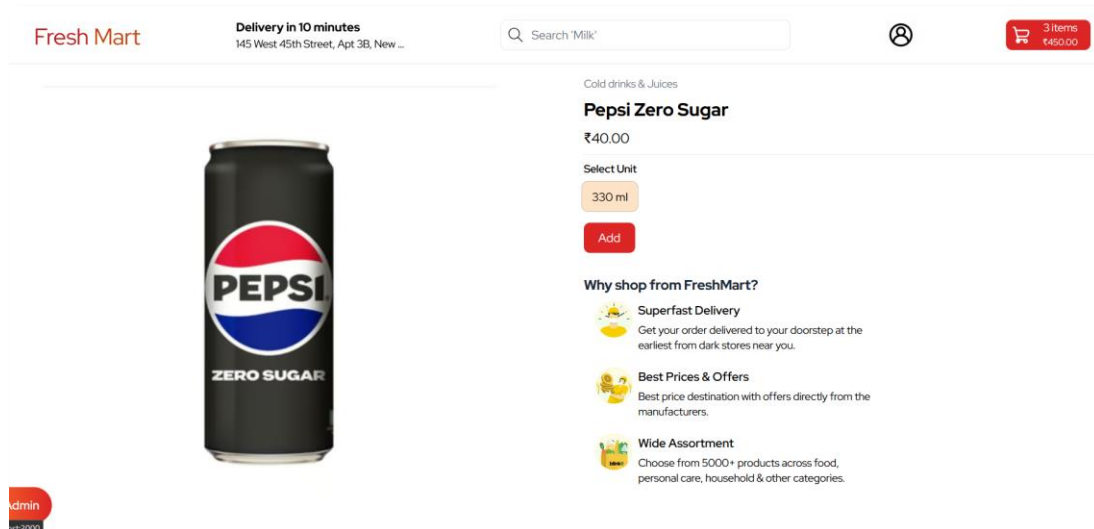


Fig 6 Item/Product View Page

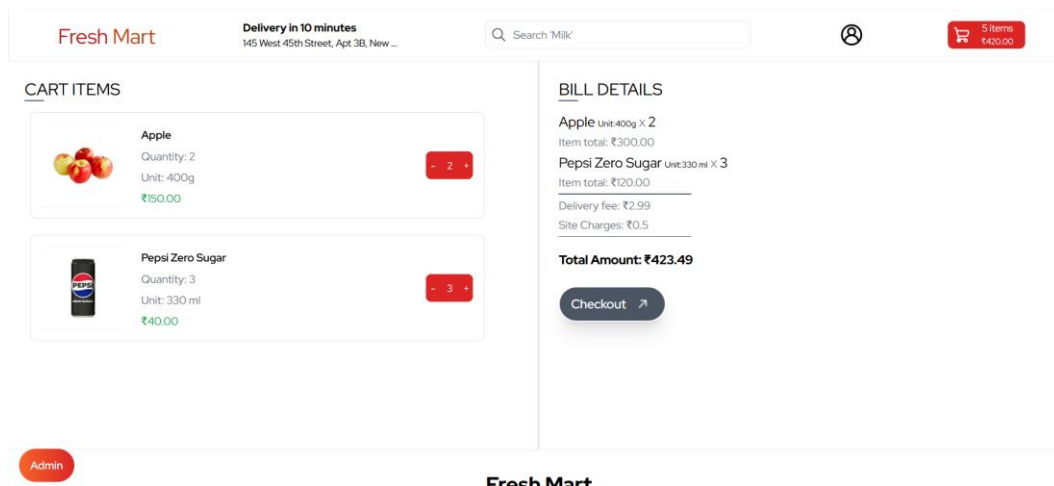


Fig 7 Cart Page

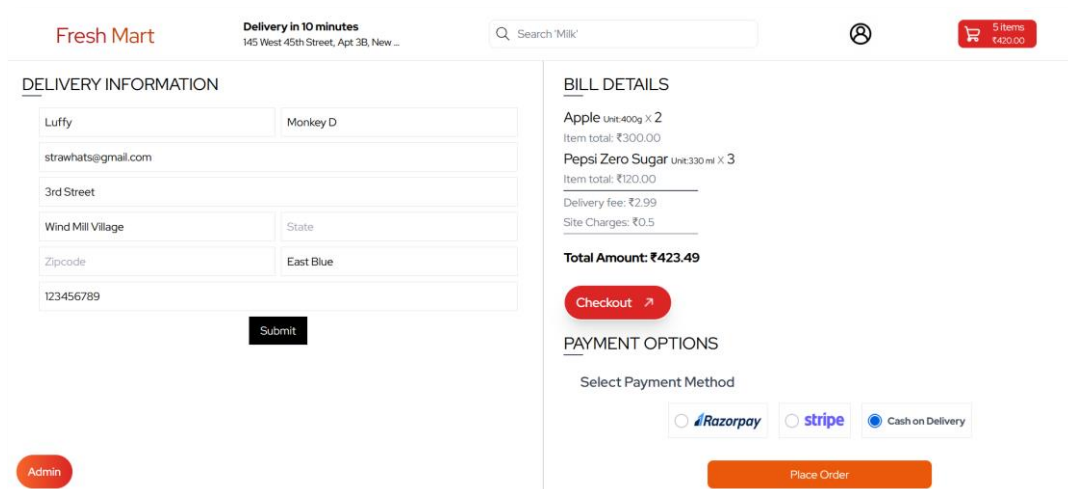


Fig 8 Checkout Page

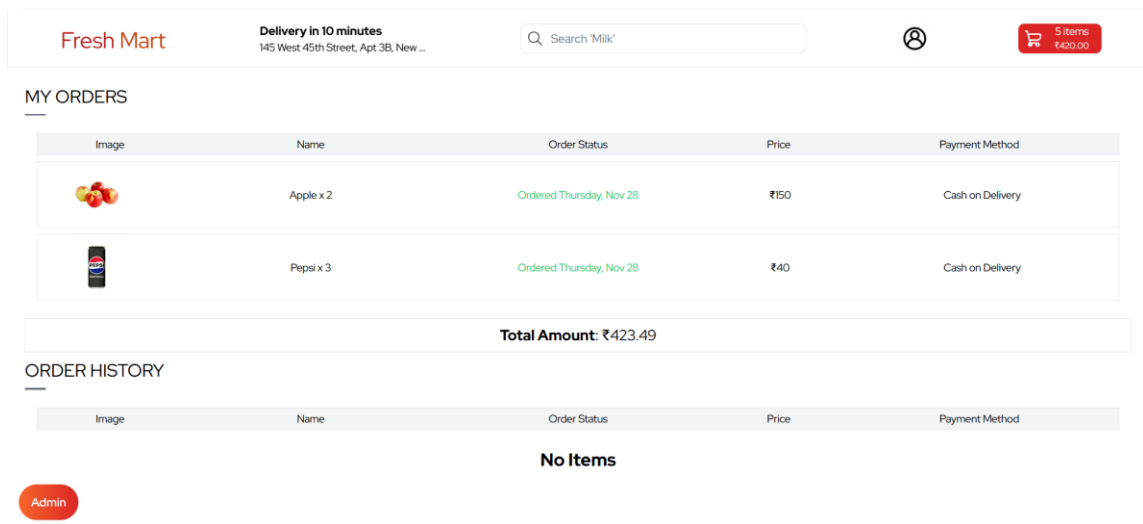


Fig 9 Orders Page

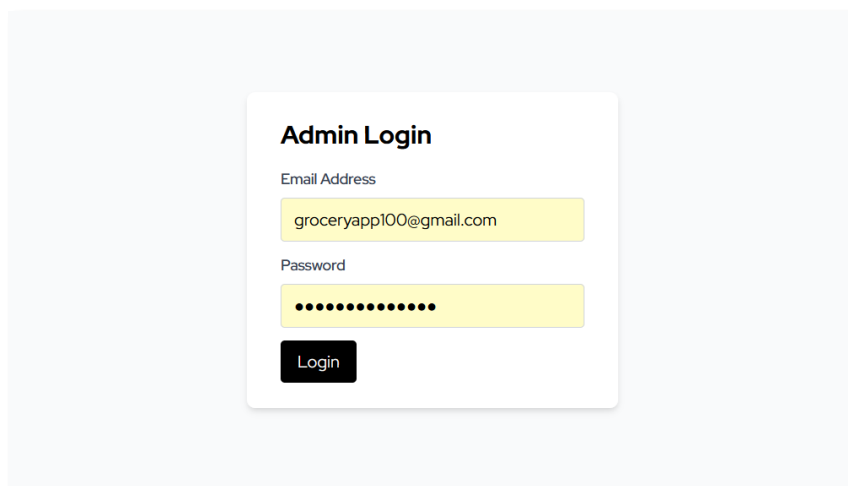


Fig 10 Login Page - Admin

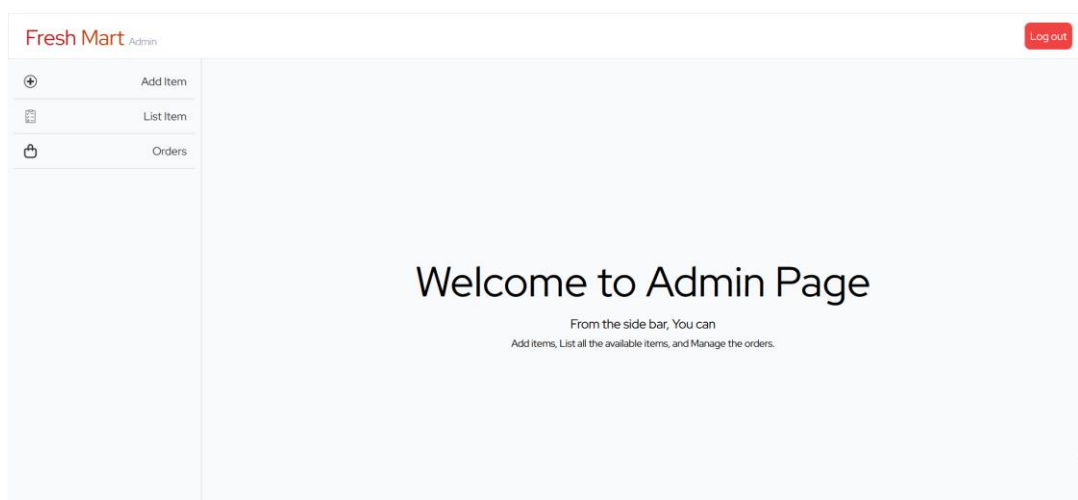


Fig 11 Admin Dashboard

Fresh Mart

Admin

Log out

+

Add Item

📁





List Item

🛒

Orders

Add Products

Upload Photos

Product Details

Cheetos

Snacks

1 pack

5 packs

Add Unit

20

Dive into a delightful cheesy treat with Cheetos Cheese Puffs Crisps. Crunchy, light and delicious in taste, these are perfect to satisfy those light hunger.

Add Product

Fig 12 Add Products - Admin

Fresh Mart

Admin

Log out

+

Add Item

📁

List Item

🛒

Orders

All Products List





Image	Name	Category	Price	Action
	Apple	Fruits & Vegetables	₹150	×
	Pepsi Zero Sugar	Cold drinks & Juices	₹40	×
	Banana	Fruits & Vegetables	₹120	×
	Sprite	Cold drinks & Juices	₹40	×

Fig 13 List Inventory - Admin

Fresh Mart

Admin

Log out

+

Add Item

📁

List Item

🛒

Orders

Order Page

User Details	Item Info	Price	Order Status
Apple x 2 Pepsi x 3 <b>Monkey D Luffy</b> Wind Mill Village, East Blue	Items : 5 Method: COD Payment: Pending Date : 28/11/2024	₹423.49	<div>Ordered</div> <div>Ordered</div> <div>Shipped</div> <div>Out for Delivery</div>

Fig 14 List orders - Admin

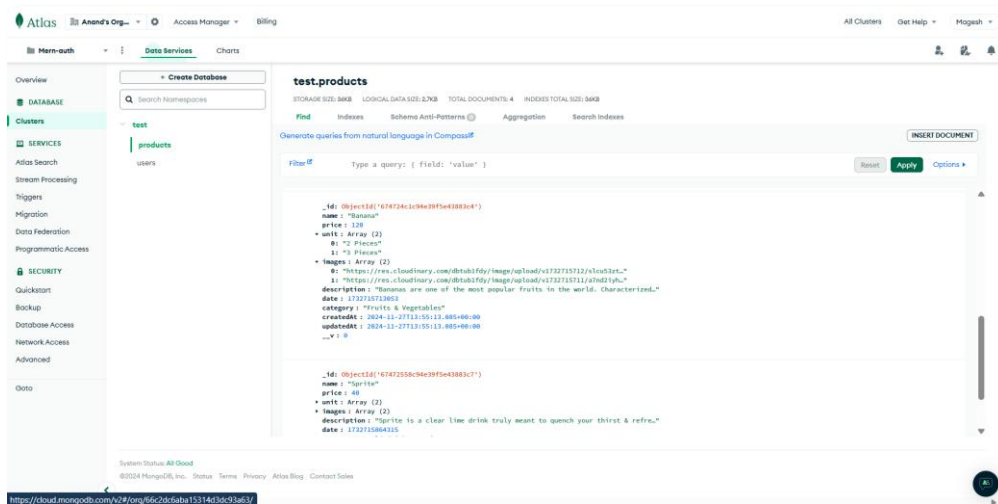


Fig 15 Product Collection - MongoDB Atlas

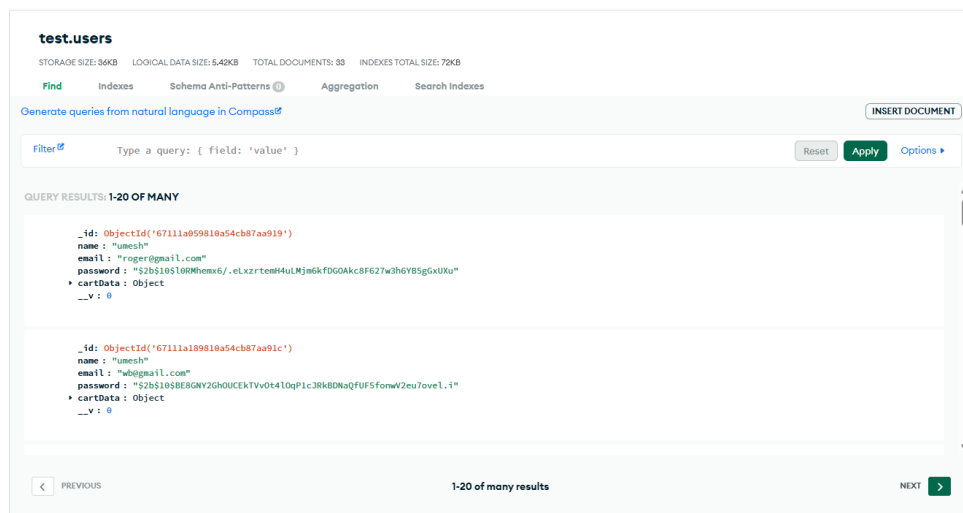


Fig 16 Users Collection - MongoDB Atlas

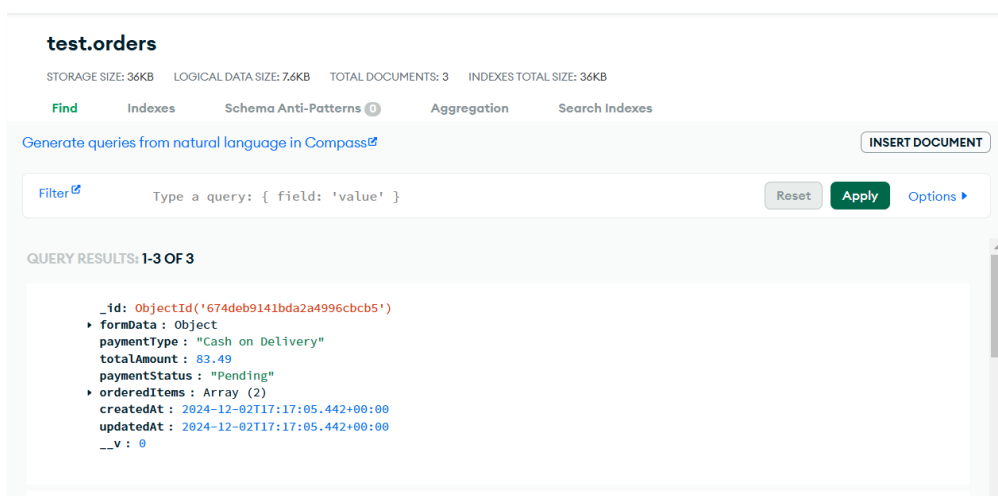


Fig 17 Orders Collection - MongoDB Atlas

# KNOWN ISSUES

## Performance Under High Traffic

During peak usage, such as sales or holiday seasons, the app may face downtime or slow load times due to insufficient server resources or unoptimized frontend rendering. Addressing this with horizontal scaling, CDN usage, and efficient frontend asset management is essential.

## Accessibility

The app may lack proper accessibility features, such as screen reader support, keyboard navigation, and adequate contrast for visually impaired users. Following WCAG (Web Content Accessibility Guidelines) can help make the app more inclusive.

## Mobile Responsiveness

Certain pages or components may not render properly on smaller screens, affecting usability on mobile devices. Consistent testing across multiple devices and frameworks like Tailwind CSS or Material-UI can ensure mobile responsiveness.

# FUTURE ENHANCEMENTS

## User Features:

- User Profiles: Save preferences, order history, etc.
- Subscription Model: Regular delivery for frequently purchased items.

## Product Management:

- Real-Time Inventory: Display up-to-date stock availability.
- Product Reviews: User ratings and reviews.
- AI Recommendations: Personalized product suggestions.

## Checkout & Payment:

- Multiple Payment Options: Support for PayPal, Stripe, etc.
- Cash on Delivery (COD): Additional payment flexibility.

## Admin Enhancements:

- Advanced Dashboard: Inventory management and sales analytics.
- Automated Order Management: Streamline backend processes.

## Delivery Features:

- Real-Time Tracking: Track orders live.
- Multiple Delivery Options: Same-day, scheduled, or pick-up.

## CONCLUSION

The Fresh Mart Grocery Web App demonstrates the potential of full-stack development to create innovative and efficient solutions for modern-day challenges. By integrating React.js, Node.js, Express.js, and MongoDB, the project has successfully delivered an intuitive platform tailored for both customers and administrators. The implementation of features such as secure authentication, responsive design, and seamless database interactions has resulted in a robust, scalable solution for online grocery shopping.

Additionally, the project addresses real-world issues like user-friendly shopping experiences and efficient inventory management, setting a solid foundation for future enhancements such as AI-powered product recommendations and improved mobile compatibility. This journey has not only strengthened our technical expertise but also highlighted the importance of teamwork and adaptability in problem-solving, paving the way for future accomplishments in web development and software engineering.