**Frontend Development with React.js**

**1. Introduction**

**Project Title:** Inventory Management Frontend

**Team Members:**

* Dharshini S – Frontend Developer & Presentation
* Sandhiya P – Project Coordinator/ Tester
* Sujitha P – Video created
* Varshana J – Documentation

**2. Project Overview**

**Purpose:**  
The Inventory Management Frontend is a React-based web application designed to manage products, sales, and inventory in a retail or warehouse environment. It enables users to track stock levels, process sales, and maintain an up-to-date product catalog with an intuitive and responsive interface.

**Features:**

* Add, update, and view product details.
* Manage and track sales transactions.
* Shopping cart with add/remove/update quantity functionality.
* Inventory dashboard with stock updates.
* Data persistence via local storage for offline-friendly usage.

**3. Architecture**

**Component Structure:**

* **NavBar** – Main navigation across pages.
* **Product Components** (ProductCatalog, ProductList, Product) – Display product information.
* **Cart Components** (Cart, CartItem) – Manage user’s cart items.
* **Inventory Components** (Inventory, Product) – Show and edit inventory details.
* **Sales Components** (Sales, SaleRecord) – View and log sales.
* **Pages** (NotFound) – 404 fallback route.

**State Management:**

* **Context API** used with separate contexts:
  + InventoryContext – Global product and stock data.
  + CartContext – Shopping cart data and actions.
  + SalesContext – Sales tracking and history.

**Routing:**

* Likely handled within App.js (React Router can be added if needed).
* Pages and components are conditionally rendered via navigation.

**4. Setup Instructions**

**Prerequisites:**

* Node.js (v14 or later)
* npm or yarn package manager

**Installation:**

# Clone repository

git clone <repo-url>

cd inventory-management-frontend

# Install dependencies

npm install

# Start development server

npm start

**5. Folder Structure**

**Client**

inventory-management-frontend/

package.json

package-lock.json

tailwind.config.js

src/

App.js

App.css

index.js

index.css

Layout.jsx

logo.svg

reportWebVitals.js

setupTests.js

App.test.js

components/

AddProduct.jsx

NavBar.jsx

Cart/

Cart.jsx

CartItem.jsx

Product/

ProductCatalog.jsx

Product.jsx

ProductList.jsx

Sales/

SaleRecord.jsx

Sales.jsx

Inventory/

Product.jsx

Inventory.jsx

pages/

NotFound.jsx

context/

SalesContext.js

CartContext.js

InventoryContext.js

public/

index.html

favicon.ico

logo192.png

logo512.png

inventoryLogo.png

cartImage.png

manifest.json

robots.txt

**Utilities**

src/

hooks/

useLocalStorage.js # Custom hook for persistent state

**6. Running the Application**

To run our React application:

1. Open the project folder in Visual Studio Code.

2. Install dependencies using:

npm install

3. Start the development server using:

npm start

4. After compilation, the project runs at:

Localhost: http://localhost:3000

Network: http://192.168.xx.xx:3000

**7. Component Documentation**

**Key Components:**

* AddProduct – Form for adding new products to inventory.
* ProductCatalog – Displays a grid/list of available products.
* Cart – Shows products added to the shopping cart.
* Sales – Displays past sales and records new ones.
* Inventory – Displays stock levels and allows modifications.

**Reusable Components:**

* NavBar – Used for navigation across all pages.
* CartItem – Represents an individual product in the cart.

**8. State Management**

**Global State:**

* Managed via Context API (CartContext, InventoryContext, SalesContext).
* useLocalStorage custom hook ensures persistence across reloads.

**Local State:**

* Managed using React’s use. State within individual components for form inputs, UI toggles, etc.

**9. User Interface**

* Built with **Tailwind CSS** for responsive and utility-first styling.
* Minimalist, mobile-friendly design.
* Icons and images stored in public/.

**10. Styling**

**Framework:** Tailwind CSS  
**Customizations:** Component-specific styling in App.css and index.css.

**11. Testing**

**Testing Strategy:**

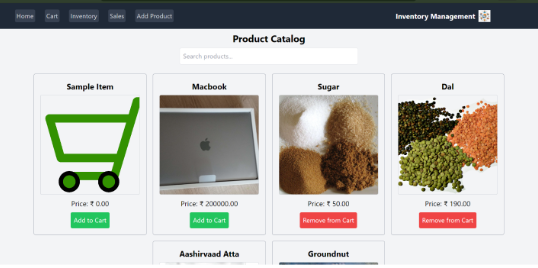
Our project uses manual testing as well as basic automated testing to ensure functionality. Each feature was tested step by step after development to confirm that it worked as expected. For React components, we could use tools like Jest or React Testing Library, but in this project, the focus was mainly on manual verification of UI, forms, and data handling.

**Code Coverage:**

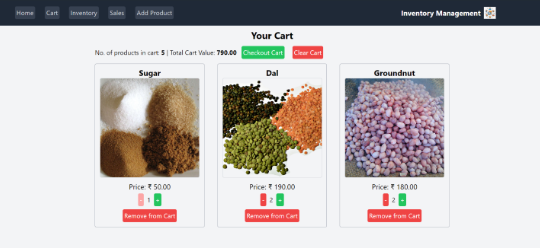
Since this is a small-scale project, code coverage tools were not fully implemented. Instead, testing focused on ensuring that all core features — such as adding, updating, and deleting items — work properly without errors. Future versions of this project can integrate Jest for unit testing and Cypress for end-to-end testing to improve code coverage.

**12. Screenshots or Demo**

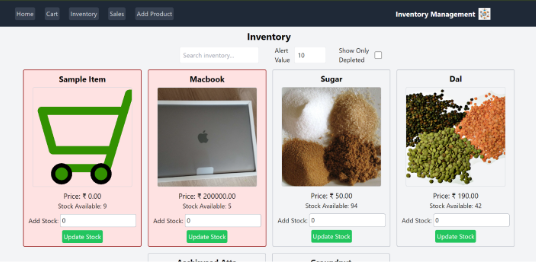
**Home :**



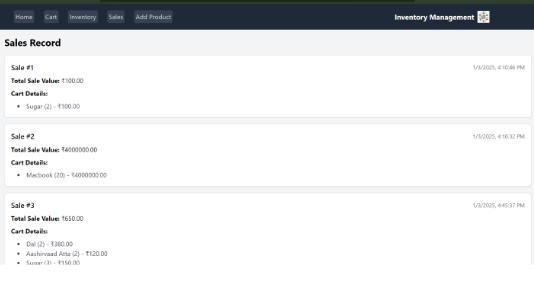
Cart



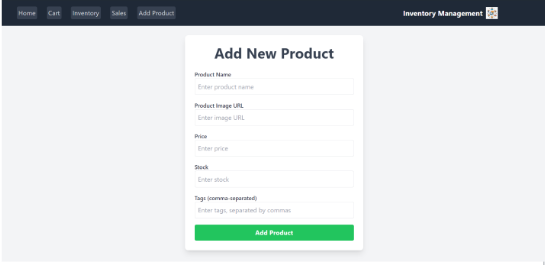
**Inventory :**



**Sales Record :**



**Add Products :**



**13. Known Issues**

* No backend integration yet; all data stored locally.
* No authentication or user management.

**14. Future Enhancements**

* Integrate with backend API for real-time updates.
* Add authentication & role-based access control.
* Implement search, filtering, and sorting in inventory.
* Improve UI with animations and better dashboard visualizations.