

DOCUMENTATION

Frontend Development with React.js

1. Introduction

- **Project Title:** RetailReady!
- **Project Topic:** Store Manager: Keep Track Of Inventory
- **NM ID:** NM2025TMID37846
- **Team Leader:** Priyadharshini V
- **Team Members:**
 - Priyadharshini V – Development and Demo video
 - Priyadharshini R – Testing and Documentation
 - Shubhasshree J – Resources and Documentation
 - Tharani G – Documentation

2. Project Overview

- **Purpose:** The **RetailReady!** application is designed to help small and medium retail stores efficiently manage their inventory and sales process. Its primary goals are to:
 - Maintain accurate stock levels with options for the addition and removal of stocks at will.
 - Track sales record with date and time.
 - Streamline the checkout process.
 - Provide insights into product performance
 - By offering a user-friendly interface, the system ensures that store staff can quickly update inventory, process customer sales, and monitor the stock.
- **Features:** Key frontend features include:
 - **Inventory Management:** Add, edit, and remove products with name, price, stock, tags, and image.
 - **Stock Updates:** Automatically update stock on checkout and when adding new stock.
 - **Cart & Checkout:** Add products to a cart, set quantities, process checkout to update inventory, and generate sales records.
 - **Depleting Stock Alerts:** Highlight low-stock items in red and maintain an alert count, with a filter to show only depleted items.
 - **Search Functionality:** Search for products in both the inventory and catalog.
 - **Sales Records:** Maintain a log of all sales with timestamp, products sold, and total sale amount.

3. Architecture

- **Component Structure:** The React application follows a modular component-based structure for maintainability:
 - **App.js and App.css** — Root component; contains layout and routes.
 - **Navbar.jsx** — Top navigation bar and branding.
 - **Cart.jsx** — Handles cart items, quantities, and checkout logic with the help of **CartItem.jsx**, **CartContext.js**, and **InventoryContext.js**
 - **ProductCatalog.jsx** — Displays products available for sale, includes search.
 - **SaleRecord.jsx** — Shows all past sales.

- **Inventory.jsx** – Manages stock updates and stock alerts with the help of **Product.jsx** and **InventoryContext.js**.
- **AddProduct.jsx** – Allows users to add new products to the inventory and product catalog.

State Management:

- The RetailReady! website uses React Context API for global state management, ensuring synchronized updates across components like Inventory and Catalog.
- The CartContext, InventoryContext, and SalesContext hold data for inventory, cart, and sales.
- Components consume this context using the useContext hook to read and update shared state.
- Local state is managed using useState inside individual components for form inputs, modal states, etc.
- localStorage is used to persist data across page reloads.
- **Routing:**
 - **react-router-dom** is used for client-side routing.
 - Routes are structured as:
 - / → Home(Catalog + Cart)
 - /inventory → Inventory
 - /sales → Sales Records
 - /alerts → Stock Alerts
 - BrowserRouter, Routes, and Route components define page navigation without full reloads.

4. Setup Instructions

- **Prerequisites:**
 - **Node.js** (v16 or above)
 - **npm** (comes with Node)
 - A code editor (VS Code)
 - Git installed on your system
- **Installation:**
 - **Clone the Repository:**
 - git clone
 - cd store-manager
 - **Install Dependencies:**
 - npm install
 - **Configure Environment Variables:**
 - Create a .env file in the root folder
 - Add any required variables like API URLs or keys
 - **Run the Development Server:**
 - Then, http://localhost:3000 if using CRA.

5. Folder Structure

- **Client:** The **React application** follows a clean and modular folder organization to keep the codebase maintainable and scalable:

Inventory_management/

|

|— public/ # Static files (favicon, index.html, etc.)

|

|— src/ # Main application source code

| |

| |— components/ # Reusable UI components

| | |— Cart/

| | |— Cart.jsx

| | |— CartItem.jsx

| | |— Inventory/

| | |— Inventory.jsx

| | |— Product.jsx

| | |— Product/

| | |— Product.jsx

| | |— ProductCatalog.jsx

| | |— ProductList.jsx

| | |— Sales/

| | | |—AddProduct.jsx

| | | |—NavBar.jsx

| |— context / # Context API for global state

| | |— SalesContext.jsx

| | |— InventoryContext.jsx

| | |— CartContext.jsx

| |— hooks/ # Custom React hooks (if any)

| | |— useLocalStorage.js

| |— styles/ # CSS modules or global stylesheets

| | |— App.css

| |

| |— App.jsx # Root component

| |— App.js # Centralized route definitions

| |— index.css #sets up Tailwind CSS and some basic global
Styles.

| |— index.js # Entry point that renders <App />

| |— Layout.jsx #makes sure navigation bar is present
consistently on every page.

| |— reportWebVitals.js

| |— setupTests.js

| |— package-lock.json

| |— package.json

| |— README.md #to provide more information about the project.

|— tailwind.config.js

- **Explanation:**

- **components/** — Small building blocks of UI used across pages.
- **context/** — Holds the Context API logic for sharing global state.
- **hooks/** — Contains reusable custom hooks for encapsulating logic (like localStorage sync).

6. Utilities: The project includes **helper utilities and custom hooks** to simplify repetitive tasks and improve code readability:

- **Custom Hook — useLocalStorage.js**

- Wraps localStorage logic so components can easily persist and retrieve data.
- Syncs state with localStorage automatically whenever it changes.

- **Utility Function:**

- **toLocaleString()**— Formats Date objects into a human-readable DD/MM/YYYY hh: mm format for sale records.
- Though the function is a JavaScript method, it is used multiple times in the program.

7. Running the Application:

- Navigate to the main project folder.
- Visual Studio Code terminal command - npm start.
- Access at <http://localhost:3000>

8. Component Documentation:

- **NavBar.js** provides navigation links to Home, Cart, Inventory, Sales, and Add Product.
- **Add Product.js**- Accepts product details as input.
- **Inventory.js** -Lists all products with their details. Cart.js - Shows selected products for checkout.
- **Cart.js shows** selected products for checkout.

9. Styling:

- Tailwind CSS is used for responsive layout.
- Predefined classes are used for grids, navigation, and styling.

10. State Management:

- The application uses React's useState and useEffect hooks for managing local state(e.g., product details, cart items).

11. User Interface

- **Home page:** The home page, categories, and prices. displays a Product Catalog, where all available products are listed.
- **Cart:** Displays the items that the user has added for checkout.
- **Inventory:** Used to view and manage product stock levels.
- **Sales:** Shows sales history and transaction details.
- **Add Product:** Allows users to add new products.

12. Demo Video:

The demo video for the project can be accessed through the Google Drive link given below:

<https://drive.google.com/file/d/1OVe9ztC-Q6rMzTxzKGnmNmnKNu5MSZtG/view?usp=sharing>

13. Testing :

- The program has been developed using HTML, CSS, JavaScript, and React components. Since there is no backend for this project, no database management system has been used in development.
- The tools used for testing the Project are as follows:
 - node.js
 - localhost:3000 server
 - VS Code
 - Web Browser (Google Chrome)

14. Known Issues:

- The **products** added to inventory **cannot be removed permanently**. We can prevent the products from showing in the Product Catalog, but we cannot delete said items from inventory.
- The **New Products** that are added **cannot be edited** after **successful addition**. If the image does not show, we cannot remove the image or add a new one. The product rate cannot be altered as well.

15. Future Enhancements:

- A new feature called **discount** can be added to give **discounts to customers**.
- The help and report features can let the customers contact the Store Manager in case of any error.
- A separate page for **Bulk Offers** can be created for providing **Offers** to the customers during special occasions and festive seasons.
- An **Expiry Date** Feature can be added to notify which **stocks need clearance**.

