Store Manager: Keep Track of Inventory

Here are some key features and a description of an inventory management system:

- **Inventory Management:** Inventory Management helps maintain healthy stock levels in a store and acquire them in time.
- **Stock Updates:** Stock will automatically update on sale of products, and it can be updated on adding new stock.
- **Cart:** Products can be added to cart for a particular sale and quantity can be added to each product.
- **Checkout at Cart**: Upon checkout, cart is cleared, inventory is updated, and a sale record is made.
- **Adding New Products to Inventory:** New products can be added to the inventory by providing product name, image URL, price, stock, tags.
- **Alert View for Depleting Stock:** Depleting stocks are shown in red background, and alert count can be updated as per requirement.

- Search Functionality for Products: Products in inventory and product catalog can be searched.

Sale Records: All sale records are stored with sale value, products and datetime.

PRE-REQUISITES

Here are the key prerequisites for developing a frontend application using React.js:

- Node.js and npm:

Node.js is a powerful JavaScript runtime environment that allows you to run JavaScript code on the local environment. It provides a scalable and efficient platform for building network applications.

Install Node.js and npm on your development machine, as they are required to run JavaScript on the server-side.

- Download: https://nodejs.org/en/download/
- Installation instructions: https://nodejs.org/en/download/package-manager/

- React.js:

React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications.

Install React.js, a JavaScript library for building user interfaces.

- Create a new React app:

npx create-react-app my-react-app

Replace my-react-app with your preferred project name.

- Navigate to the project directory:

cd my-react-app

- Running the React App:

With the React app created, you can now start the development server and see your React application in action.

- Start the development server:

npm start

This command launches the development server, and you can access your React app at http://localhost:3000 in your web browser.

- HTML, CSS, and JavaScript: Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.

- Version Control: Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

• Git: Download and installation instructions can be found at: https://git-scm.com/downloads

- Development Environment: Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.

• Visual Studio Code: Download from https://code.visualstudio.com/download

• Sublime Text: Download from https://www.sublimetext.com/download

• WebStorm: Download from https://www.jetbrains.com/webstorm/download

To get the Application project from drive:

Follow below steps:

Install Dependencies:

• Navigate into the cloned repository directory and install libraries:

cd store

npm install

- Start the Development Server:

• To start the development server, execute the following command:

npm start

Access the App:

• Open your web browser and navigate to http://localhost:3000.

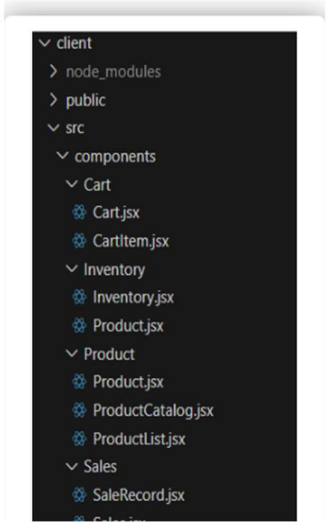
• You should see the application's homepage, indicating that the installation and setup were successful.

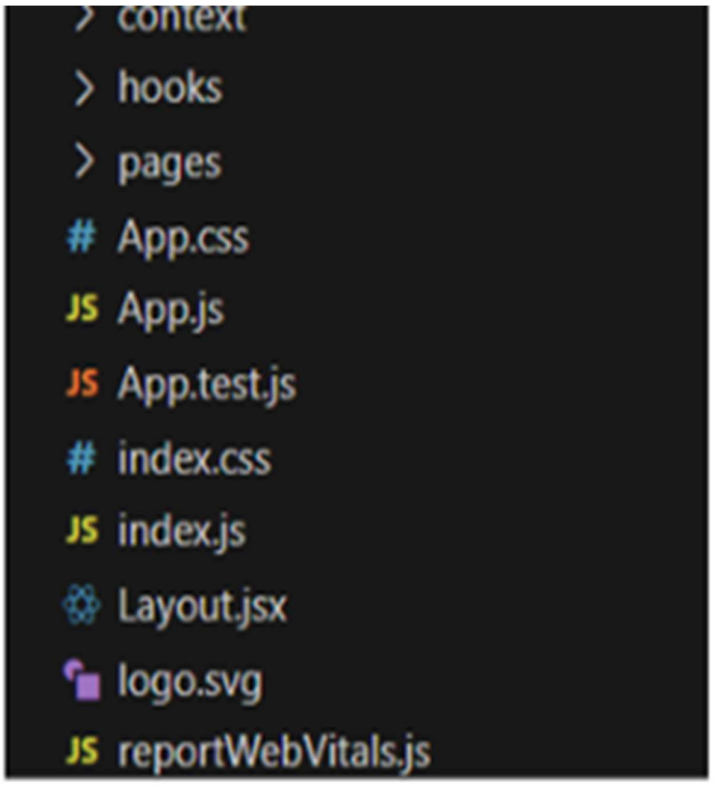
You have successfully installed and set up the application on your local machine. You can now proceed with further customization, development, and testing as needed.

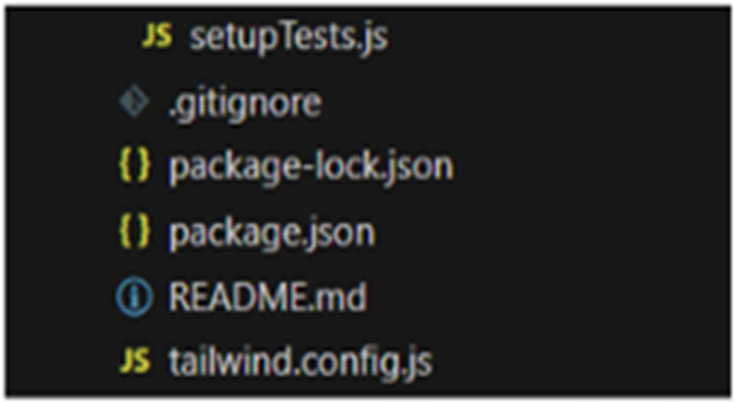
Project Structure

Project Structure

The image is of the folder structure which shows all the files and folders that have been used in project development.

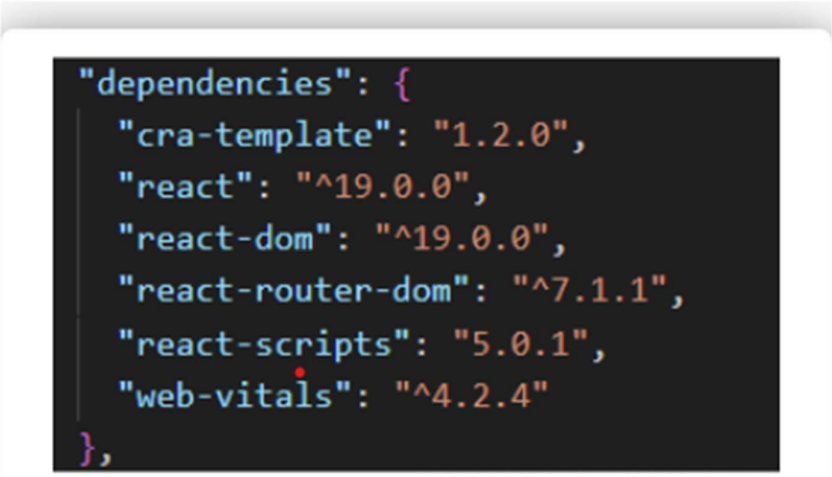


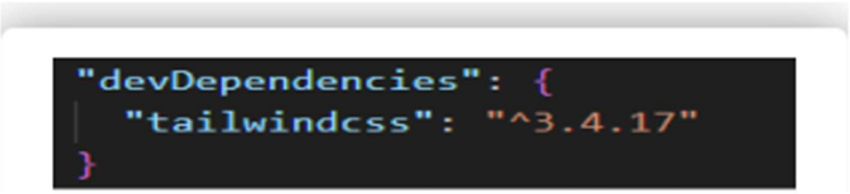

context
hooks
pages
App.css
App.js
App.test.js
index.css
index.js
Layout.jsx
logo.svg
reportWebVitals.js


setupTests.js
.gitignore
package-lock.json
package.json
README.md
tailwind.config.js

Project setup and configuration

- Project Setup
    - Step 1: Initialize a new React application.
        - Use create-react-app or Vite for project setup.
    - Step 2: Install dependencies.
        - Add tailwindcss, react, react-dom, and other necessary packages.





- Step 3: Configure TailwindCSS.
    - Set up the tailwind.config.js file.
    - Add the required styles in index.css.
- Step 4: Create a basic folder structure.
    - Example:

```
css
src/
??? components/
??? context/
??? hooks/
??? pages/
```

- For further reference, use the following resources
- https://react.dev/learn/installation
- https://react-bootstrap-v4.netlify.app/getting-started/introduction/
- https://axios-http.com/docs/intro
- https://reactrouter.com/en/main/start/tutorial

Project Development

- Setting Up Context and Reducers
    - Step 1: Create context files for Inventory, Cart, and Sales.
        - Use createContext() and define default states.
    - Step 2: Define reducers for state management.
        - Inventory Reducer: Handle stock addition, updates, and sales.
        - Cart Reducer: Manage adding/removing items and quantities.
        - Sales Reducer: Track sales records.
    - Step 3: Set up ContextProvider components.
        - Wrap the app in these providers in index.js or App.js.
    - Step 4: Persist data in localStorage.
        - Save and load context states to/from localStorage.

Milestone 3- Developing Core Components

- Step 1: Build the Inventory Component.
- Display a list of products with stock details.
- Implement search and alert value input.
- Step 2: Build the Product Component.
- Show product details, including an image, price, and stock.
- Add functionality to update stock.
- Step 3: Build the Cart Component.
- Display items added to the cart with quantity and total value.
- Add buttons for incrementing, decrementing, and removing items.
- Include checkout functionality to update inventory and create sales records.
- Step 4: Build the Sales Component.

- Display a list of sales records with details like date, cart items, and total sale value.
- Sort records from the latest to the oldest.

Milestone 4- Implementing Utilities

- Step 1: Create reusable helper functions.
  - Format currency.
  - Sort or filter data.
- Step 2: Add input validation.
  - Prevent invalid values for stock or cart quantities.
- Step 3: Handle edge cases.
  - Avoid negative stock updates.
  - Restrict checkout when the cart is empty.

Milestone 5- Styling with TailwindCSS

- Step 1: Create responsive layouts.
    - Use flexbox (flex, flex-wrap) and grid utilities.
- Step 2: Style individual components.
    - Inventory: Add hover effects and responsive search bar.
    - Cart: Highlight selected products and display alerts on low stock.
    - Sales: Use a clean table or card layout for sale records.
- Step 3: Ensure consistency.
    - Apply a theme or consistent color palette.

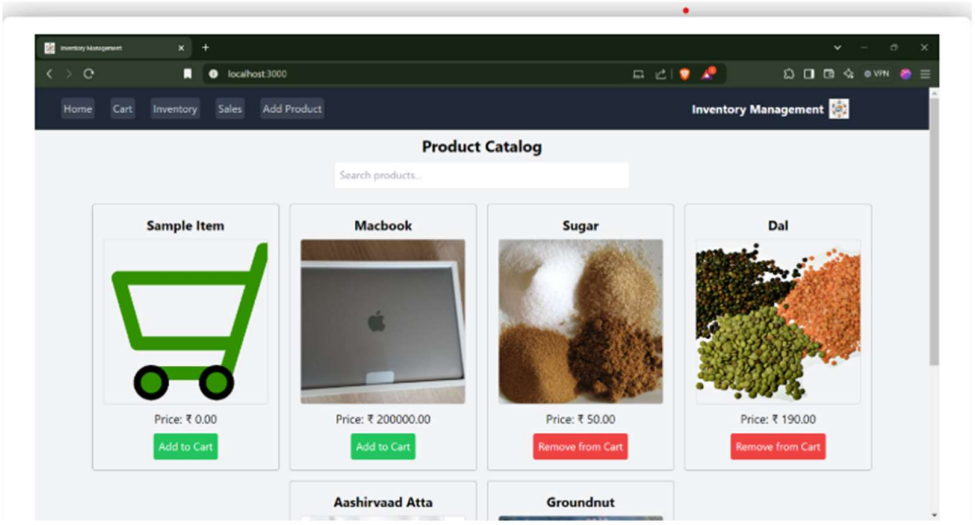
Milestone 6- Testing and Debugging

- Step 1: Test individual components.
    - Ensure correct rendering and functionality.

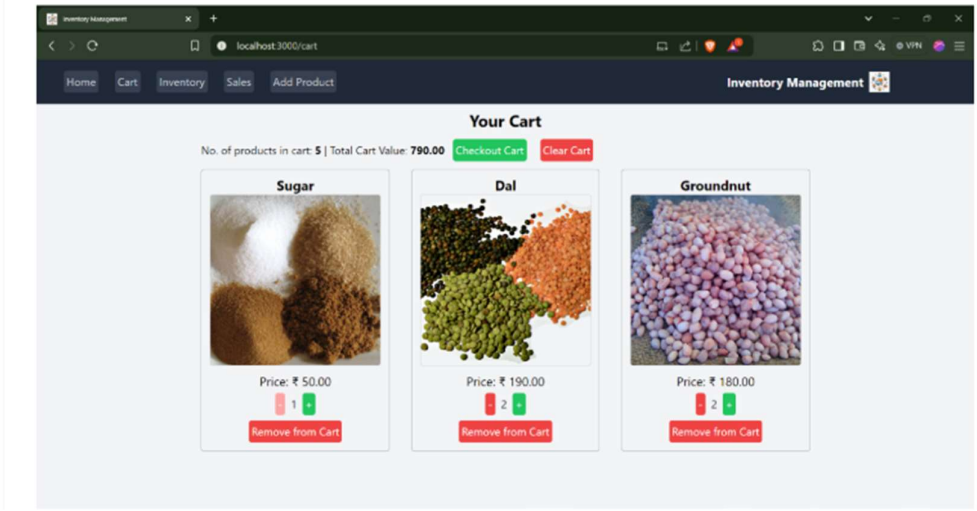
- Step 2: Test context and reducers.
    - Verify state updates and interactions with localStorage.
- Step 3: Debug UI/UX issues.
    - Check for responsiveness and usability.

Project Implementation & Execution

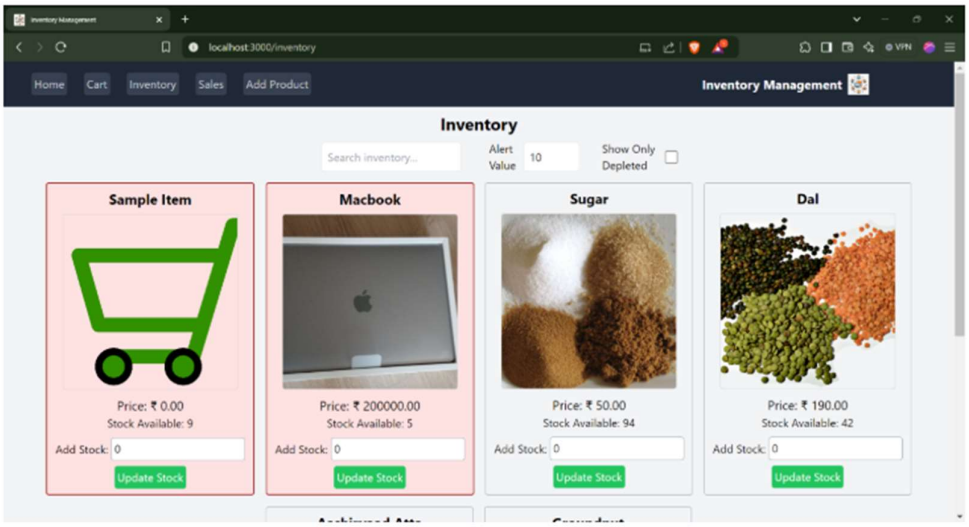
Product Catalog:



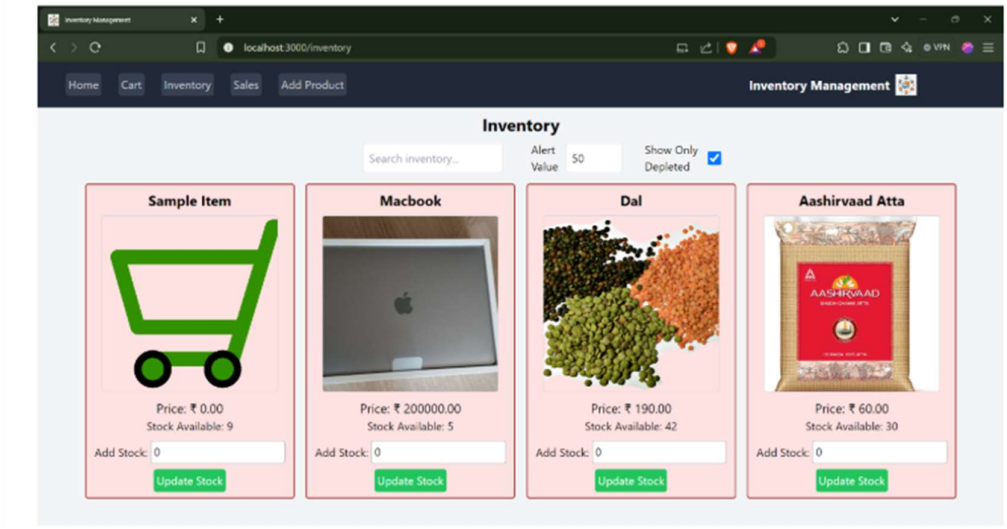
Cart:



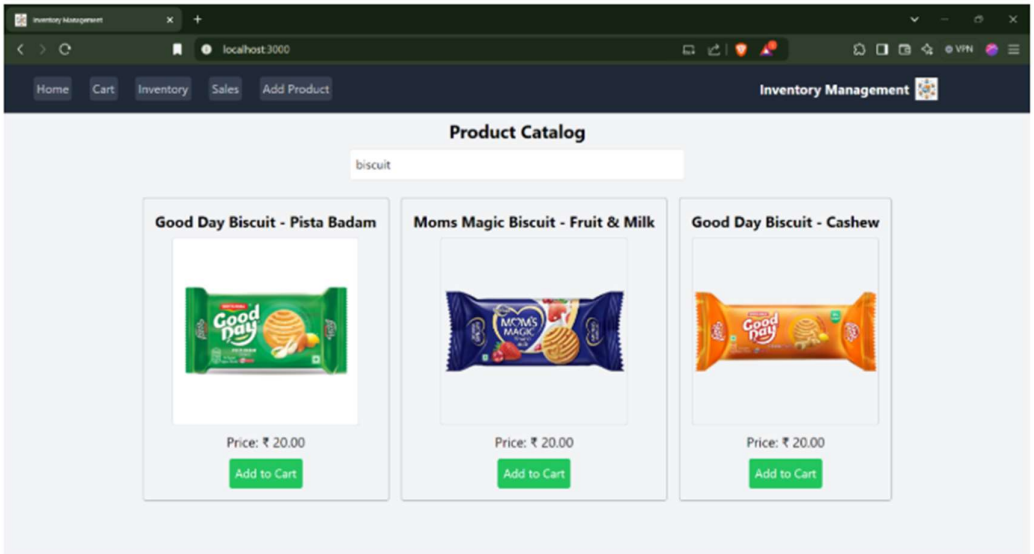
Inventory:



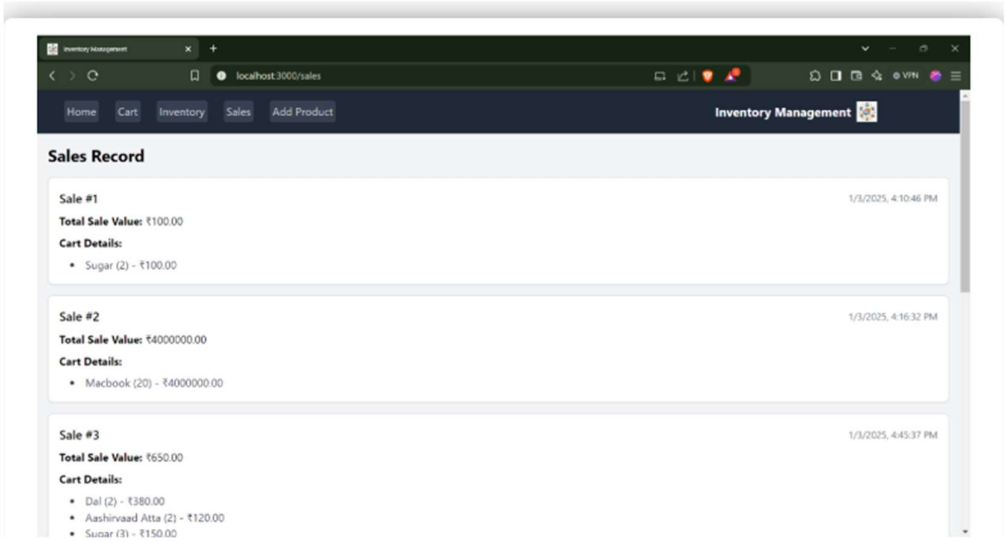
Depleted Stock:



Search Functionality:



Sales:



Add Product: