Frontend Development with React.js Project Documentation format

1. Introduction

* Project title: [Store manager]
* Team members

R.RAMYA

R.RAMYA

V.RANJITHA

V.RITHIKA

P.RITHICKA

2. Project Overview

Purpose: The purpose of the Store Manager Project is to develop a system that simplifies and automates the management of store operations. It is designed to support store owners, managers, and employees by providing an organized way to track inventory, sales, employees, and customer information. By replacing manual processes with a digital solution, the project aims to reduce errors, save time, and improve overall efficiency in store management.

Goal of the Store Manager Project:

The primary goal of the Store Manager Project is to ensure smooth and effective store operation by:

* Maintaining accurate records of products, stock levels, and transactions,
* Automating routine tasks to minimize manual effort and human error.
* Providing useful reports and insights for better decision-making.
* Enhancing customer satisfaction by ensuring product availability.
* Supporting business growth through improved control and resource management.

1. Architecture

Outline the structure of major components

[15-09-2025 19:35] Siva Kumar 3019: store-manager/

├── S R C/

│ ├── A P I/ # API service functions (fetch)

│ ├── components/ # Reusable UI components (Button, Table, Modal)

│ ├── context/ # Context providers (Inventory)

│ ├── hooks/ # Custom hooks (use Products)

│ ├── layouts/ # Layout components (Header, Sidebar, Footer)

│ ├── pages/ # Feature pages (Dashboard, Products, Orders, etc.)

│ ├── store/ # Re dux slices (if using Re dux)

│ ├── u tils/ # Helper functions (format Date, currency)

│ ├── App. J s x # Main app with routes

│ └── index.js # Entry point

├── Public / # Static assets

└── Package. j son

[15-09-2025 20:33] Siva Kumar 3019:

Frontend

Handles UI & client-side-logic.

* Component Layer

Pages/Containers – high-level views

(Dashboard Page, Products Page, etc.)

Layout – Header, Sidebar, Footer

UI Component – Button, Modal, Table, Card

* State Management

Local state (use State, use Reducer)

Global state (Context API)

Data fetching with React Query / Fetch API

* Reports Service

Analytics (sales, inventory, revenue)

Database Layer

* Stores business data:

Products (id, name, price, stock, category)

Orders (id, items, customer, total, status)

Customers (id, name, contact, history)

Users (id, username, password, role)

* Common DB choices:

SQL (PostgreSQL / MySQL)

NoSQL (MongoDB) for flexibility

Integration Layer

* Optional – connects to external systems:

Payment Gateway (Stripe, PayPal)

Email/SMS notifications

Third-party inventory systems

Cross-Cutting Concerns

* Error handling – API errors surfaced in UI via Toast/Modal
* Security – JWT validation, role-based UI
* Logging & Monitoring – track inventory updates, sales reports
* Testing – unit tests (Jest/React Testing Library), integration test

State Management:

* Context API (Simple & Built-in)

Best for lightweight apps or where global state isn’t too complex.

Uses React. Create context + use reducer or use state

Example use cases:

Authentication – current user, roles, login/logout

Theme (Theme Context) – dark/light mode

* Re dux (Scalable & Structured)

Best for larger apps with complex state and many interactions.

Centralized store where slices manage different domains.

* Example store slices:

Product Slice – product list, filters, stock updates.

Order Slice – order list, statuses, history.

Customer Slice – customer records, purchase history.

Top-Level Routes:

Import {Browser Router, Routes,} from "react-router-domain";

Import Layout from "./layouts/Layout";

Import Dashboard Page from "./pages/Dashboard Page";

Import Products Page from "./pages/Products Page";

Import Orders Page from "./pages/Orders Page";

Import Customers Page from "./pages/Customers Page";

Import Reports Page from "./pages/Reports Page";

Import Login Page from "./pages/Login Page";

Import Not Found Page from "./pages/Not Found Page";

Function App ()

<Browser Router>

<Routes>

{/\* Public route \*/}

<Route path="/login" element= {<Login Page />} />

{/\* Protected routes \*/}

<Route element= {<Layout />}>

<Route path="/" element= {<Dashboard Page />} />

<Route path="/products" element= {<Products Page />} />

<Route path="/orders" element= {<Orders Page />} />

<Route path="/customers" element= {<Customers Page />} />

<Route path="/reports" element= {<Reports Page />} />

</Route>

{/\* Fallback \*/}

<Route path="\*" element= {<Not Found Page />} />

</Routes>

</Browser Router>

);

}

Authentication: Login, Register

Dashboard: Overview of sales, products, staff, etc.

Products: Lists, Roles, Permissions

Reports: Sales, Inventory, Performance

Settings: Profile, Store preferences

1. Setup Instructions

Node.js (LTS version, >= 18.x) – runtime environment for JavaScript

Yarn – package manager for installing project dependencies

GIT – version control system

A database server (depends on project):

MySQL / PostgreSQL (common for relational storage), or MongoDB (if using NoSQL)

* React (or Angular depending on implementation)
* AXIOS – for making API requests
* Re dux / context API – state management
* Tailwind CSS / Bootstrap / Material UI – styling

Installation

Clone the repository

Clone https://github.com/<username>/store-manager.git

Cd store-manager

Install dependencies

N pm install

Configure environment variables

C p .e n v. Example .e n v

# then edit .e n v with DB credentials & JWT secret

Folder Structure

Explanation of Folders

* All static resources – assets (images, icons, styles)
* Reusable UI elements – components (buttons, modals, charts)
* State management – store (re dux slices, context)
* Handles API calls – services (e.g., fetching products, posting sales)
* Custom logic reusable across components – hooks
* Helper functions – u tils (date formatting, currency conversion)
* Route definitions for navigation – routes
* App-wide contexts – context (theme, settings)

This setup ensures scalability, reusability, and maintainability, making it easy to add new features without cluttering the project.

Store -manager/

│── Public/ # Static files (index.html, favicon, etc.)

│

│── S r c/ # Main source code

│ │── assets/ # Images, icons, fonts, styles

│ │ ├── images/

│ │ ├── icons/

│ │ └── styles/ # Global CSS/SCSS or Tailwind

│ │

│ │── components/ # Reusable UI building blocks

│ │ ├── common/ # Buttons, Inputs, Modals, Loaders

│ │ ├── layout/ # Navigation bar, Sidebar, Footer

│ │ └── charts/ # Reusable chart components (Sales Chart, Inventory Chart)

│ │

│ │── pages/ # Page-level components (mapped to routes)

│ │ ├── Dashboard/

│ │ ├── Products/

│ │ ├── Sales/

│ │ ├── Inventory/

│ │ ├── Employees/

│ │ └── Settings/

│ │

│ │── store/ # State management (Re dux / Context API)

│ │ ├── slices/ # Re dux slices (productsSlice.js, salesSlice.js)

│ │ ├── actions/ # A sync actions (API calls)

│ │ └── index.js # Store setup

│ │

│ │── services/ # API calls

│ │ ├── apiClient.js

│ │ ├── productService.js

│ │ └── salesService.js

│ │

│ │── hooks/ # Custom hooks (e.g., Fetch)

│ │

│ │── u tils/ # Helper functions (format Date, calculate Totals)

│ │

│ │── routes/ # Central route definitions

│ │ └── AppRoutes.js

│ │

│ │── context/ # React Context (if used for theme, etc.)

│ │

│ │── App.js # Root app component

│ │── main.js # Entry point (React DOM .render / create Root)

│

│── .e n v # Environment variables

│── Package .j son

│── tailwind.config.js / # Configuration (if using Tailwind or other frameworks)

│── vite.config.js / webpack.config.js

Utilities

In store manager project, the utilities are small, reusable pieces of logic that make the code base cleaner, more maintainable, and prevent duplication.

File Handling Utilities

Reads from inventory. Csv and loads data into a python

Dictionary

Utility role: abstracts away file reading logic, so main code

Does not deal with CSV parsing directly.

Inventory Management Utilities

Add a new item or increases quantity if it already exists

Encapsulates “add” logic, so we don’t repeat the same steps

Everywhere

Updates an item’s quantity

Removes an item completely from the inventory

Prints all items and their quantities neatly