**Project Documentation**

**1.Introduction**

**Project** **Title**: **Store Management System**

**Team ID: NM2025TMID37964.**

**Team Leader**: C. ILAKKIYA (Coding &Output)

[ilakkiya06cp@gmail.com](mailto:ilakkiya06cp@gmail.com)

**Team Members:**

1. Z. DHIVEYASREE (Documentation)

[krishnaraj2102@gmail.com](mailto:krishnaraj2102@gmail.com)

2.S. ESTHERRANI ( documentation)

[esther03058@gmail.com](mailto:esther03058@gmail.com)

3.NV. JAYADHARSHANI ( Demo video)

[dharshine47@gmail.com](mailto:dharshine47@gmail.com)

4.K. YAMINI (demo video)

[yaminiyashni@gmail.com](mailto:yaminiyashni@gmail.com)

The Store Management System is a web-based application designed to simplify the process of managing products, inventory, customers, and sales in retail stores. It provides a structured, digital solution to replace traditional manual methods, reducing errors and increasing efficiency. Built using React.js for the frontend, the application is user-friendly, scalable, and secure.

**2.Project Overview**

* **Purpose:**

The main purpose of this system is to automate store operations. It helps store managers and staff keep track of inventory, handle billing, manage customer data, and generate sales reports efficiently. By moving to a digital platform, store owners gain real-time visibility of their business.

**Key Goals**:

1. Replace manual record-keeping with a digital solution.
2. Track inventory levels and send low-stock alerts.
3. Simplify billing and receipt generation.
4. Maintain a customer database for better service.
5. Provide dashboards and reports for data-driven decisions.

* **Features:**

1. **Product Management:** Add, edit, delete, and search products with details like name, price, and quantity.
2. **Inventory Management:** Monitor stock availability; low stock notifications.
3. **Sales & Billing:** Generate invoices, calculate totals, and track transactions.
4. **Customer Management:** Store customer contact details and purchase history.
5. **Reports & Analytics:** View sales reports by day, month, or product.
6. **User Authentication:** Secure login and role-based access for staff.

**3.Architecture**

**Frontend**: Built with React.js for a dynamic and responsive UI.

**Backend**: Node.js with Express.js (for APIs and data management).

**Database**: MongoDB (storing products, customers, and sales).

**State Management**: Context API with React Hooks for global state handling.

* **Component Structure (Frontend):**

1. App.js – Root component handling routes.
2. Navbar.js – Navigation bar for different sections.
3. Dashboard.js – Displays sales stats, revenue, and alerts.
4. Products.js – Manage product catalog.
5. Inventory.js – Track stock levels.
6. Sales.js – Handle billing process.
7. Customers.js – Manage customer details.
8. Reports.js – Generate and display reports.
9. Auth.js – Login/Signup forms.

* **State Management:**

the state management is mainly handled using React Context API along with React Hooks.

1. **Context API:**

This is used for global state management. It allows data like products, customers, sales, and authentication details to be shared across the entire application without passing props manually through multiple components.

1. **React Hooks (useState, useEffect):**

These are used for local state management. Temporary values such as form inputs, search filters, or modal visibility are handled inside individual components.

1. **Redux (Not Used):**

Redux is another powerful state management library for React, but for this project it was not required, since Context API is simpler and sufficient for the application’

* **Routing**:

Using react-router-dom, the routes are:

/ – Login page

/dashboard – Dashboard with stats

/products – Product management

/inventory – Inventory overview

/sales – Billing system

/customers – Customer details

/reports – Reports section

**4.Setup Instructions:**

* **Prerequisites**:

Node.js (v16+)

MongoDB (for backend storage)

Git (for version control)

Visual Studio Code

* **Installation Steps**:

1. Clone repository:

Git clone <https://github.com/your-repo/store-management.git>

1. Install frontend dependencies:

Cd client

Npm install

1. Install backend dependencies:

Cd ../server

Npm install

1. Start frontend:

Cd client

Npm start

1. Start backend:

Cd server

Npm start

**5.Folder Structure:**

Store-management/

│── client/ # React frontend

│ ├── src/

│ │ ├── components/

│ │ │ ├── Navbar.js

│ │ │ ├── ProductCard.js

│ │ ├── pages/

│ │ │ ├── Dashboard.js

│ │ │ ├── Products.js

│ │ │ ├── Inventory.js

│ │ │ ├── Sales.js

│ │ │ ├── Customers.js

│ │ │ ├── Reports.js

│ │ ├── context/

│ │ │ ├── StoreContext.js

│ │ ├── utils/

│ │ │ ├── api.js

│ │ ├── App.js

│ │ ├── index.js

│

│── server/ # Node.js backend

│ ├── routes/

│ ├── models/

│ ├── controllers/

│── package.json

**6.Running the Application:**

* Frontend:

Cd client

Npm start

* Backend:

Cd server

Npm start

* Open browser: <http://localhost:3000>

**7.Component Documentation:**

* Navbar: Provides navigation between modules.
* Dashboard: Displays summary (total sales, stock alerts).
* Products: Allows staff to add, update, and delete products.
* Inventory: Shows stock count, warns when low.
* Sales: Billing interface with real-time price calculation.
* Customers: Saves customer records.
* Reports: Generates sales/inventory reports.
* Auth: Login/Signup with secure authentication.

**8.State Management:**

the State Management section has two parts:

* 1. Global State – how the whole application’s data is shared.
  2. Local State – how small, temporary data is handled inside components.
* **Global State**:

1. The Store Management System uses React Context API to handle global state.
2. The StoreContext provides access to data like products, inventory, sales, and customer details across all components.
3. For example, when a new product is added, the product list is updated globally and shown immediately on the Products page, Inventory page, and Reports page.
4. Authentication details (logged-in user) are also stored in global state, so that access is controlled throughout the app.

* **Local State**:

1. For temporary and component-specific data, the system uses React Hooks such as useState and useEffect.
2. Example: In the Sales page, when creating a bill, the quantity and price entered by the user are stored in local state. Once the bill is generated, the data is either sent to the global state or cleared.
3. On the Product form, local state is used to capture input values before saving them to the global context.

* **Flow of State**:

1. Global state is used for shared data like products, customers, and sales history.
2. Local state is used for temporary actions like filling forms, search filters, or pop-up modals.
3. This combination ensures efficiency and keeps the application structured.

**9.User Interface:**

* **Dashboard:** Graphs and statistics.
* **Product Page:** Grid of product cards with CRUD operations.
* **Sales Page**: Simple billing form with auto-total.
* **Customer Page:** Table format for customer details.
* **Reports Page:** Charts showing sales and stock levels.

**10.Styling:**

1. **Framework:** Tailwind CSS for responsiveness.
2. **Design System:** Custom reusable buttons, forms, and cards.
3. **Theming:** Light mode with consistent color palette.

**11.Testing:**

**Testing Strategy:**

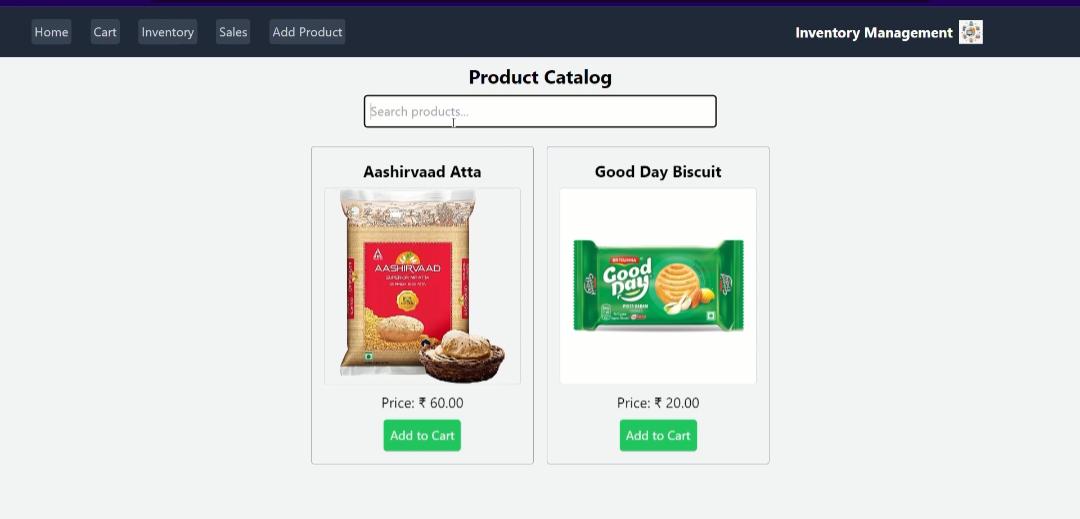
1. **Unit Testing:** Jest for core functions.
2. **Component Testing:** React Testing Library for UI behavior.
3. **API Testing:** Postman for backend endpoints.
4. **Code Coverage:** Reports generated to ensure all modules are tested.

**12.Screenshots or Demo:**

1. Dashboard showing sales summary.
2. Product management screen.
3. Billing form for sales.
4. Reports with charts and tables.

 **THE** **STORE:**

1. Need product and it’s prices:



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**13.Known Issues:**

1.No offline support.

2.Data refresh required for new reports.

3.Limited to single-store usage.

**14.Future Enhancements:**

1. Barcode scanner for faster billing.
2. Mobile app version using React Native.
3. Cloud-based sync for multi-branch stores.
4. Role-based access (Admin, Cashier, Manager).
5. AI-based sales prediction.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_