Frontend Development with React.js

Project Documentation format

1. **Introduction**
   - **Project Title**:Store manager:Keep track of inventory
   - **Team Members**: Sanjana.V(Development)
   - Pushpanandhini (Documentation)
   - Madelene (Video)
   - Jayasree. K (Coding)
2. **Project Overview**
   - **Purpose**:
   - Oversee daily operations – managing sales, stock, and customer service.
   - Achieve sales and profit goals – ensuring the store meets business targets.
   - Provide good customer service – ensuring customers are satisfied and complaints are handled.
   - Supervise and train staff – guiding employees to perform effectively.
   - Ensure store presentation – keeping the store clean, organized, and attractive.
   - **Features**:
   - Inventory Control – Keeps track of stock levels, prevents overstocking or stock-outs, and ensures goods are stored properly
   - Record Keeping – Maintains accurate records of receipts, issues, balance of stock, and movement of goods.
   - Storage & Safety – Ensures proper storage methods, labeling, preservation, and safety of goods.
   - Cost Control – Helps reduce storage costs by avoiding wastage, damage, and pilferage.
   - Supervision & Staff Management – Guides store staff, allocates duties, and ensures efficiency.
3. **Architecture**
   - **Component Structure:**
   - Purchasing – Ensures goods are ordered and stocked.
   - Receiving – Checks and records incoming materials.
   - Storage – Safely stores goods in proper conditions.
   - Stock Control – Monitors inventory levels to avoid shortages/overstock.
   - Security – Protects materials against theft, damage, or loss.
   - **State Management**:
   - Context API – Used to share data (e.g., cart items, stock levels) globally without passing props manually at each level.
   - Redux – A predictable state container that stores the entire application state in a single store. It uses actions and reducers to update the state, making it easier to debug and manage complex store operations.
   - API Integration – Data can also be managed using APIs where the backend provides updated state (like inventory count or order status) to the frontend.
   - **Routing**:
   - Ensures smooth transition between different store modules without reloading the whole system.
   - Supports modular design, where each function (inventory, billing, reports, customers) has its own route.
   - Makes the application scalable, since new features can be added with new routes.
   - Provides better control of navigation, reducing errors while accessing store operations.
   - Helps in tracking user activity by maintaining navigation history.
   - Works with role-based routing, showing different routes for admin, cashier, and store manager.
   - Improves user experience by offering clear and predictable navigation.
4. **Setup Instructions**
   - **Prerequisites**: Educational Qualification – Preferably graduate in Business/Commerce/Management.
   - Retail Knowledge – Understanding of sales, merchandising, and store operations.
   - Computer Literacy – Knowledge of POS systems, MS Office, and inventory software.
   - Work Experience – Prior experience in retail, sales, or supervisory roles.
   - Leadership Skills – Ability to manage staff, delegate tasks, and motivate team.
   - Customer Service Skills – Handling complaints, improving customer satisfaction.
5. **Folder Structure**
   - **Client**:
   - /src/components – Contains reusable UI components (buttons, forms, modals, product cards)
   - /src/pages – Holds main pages/screens like Dashboard, Products, Sales, Inventory, Reports.
   - /src/assets – Stores static files such as images, icons, CSS, fonts, and logos.

- /src/utils – Contains helper functions, utility classes, and custom hooks used across the app.
- /src/services (or api) – Manages API calls and communication with backend (e.g., fetching products, updating stock, sales entries).
- **Utilities**:
- Validations → Prevents wrong entries (e.g., no negative stock, valid email).
- Formatters → Formats sales price into currency format.
- Constants → Defines user roles like Admin, Manager, Staff in one place.
- Helpers → Provides reusable functions (e.g., calculate total sales, filter out-of-stock items).
- Logger → Captures and logs errors for debugging.
- Custom Hooks → Example: useFetchProducts() to fetch product data from API.

6. **Running the Application**

- npm install in the terminal
- npm start in the client directory.

7. **Component Documentation**
   - **Key Components**:
   - User Interface (UI) – Provides an easy and interactive way for store staff/admins to manage the system.
   - Product Management – Handles adding, updating, deleting, and categorizing products.
   - Inventory Management – Tracks stock levels, availability, and stock alerts,
   - Order Management – Manages customer orders, billing, and order history.
   - Customer Management – Stores customer details, purchase history, and preferences.
   - Database – Central storage for products, customers, orders, and transactions.
   - **Reusable Components**:
   - Header / Navbar Component – Common navigation bar used across all pages (Products, Orders, Customers).
   - Sidebar / Menu Component – Provides easy access to different modules (Inventory, Sales, Reports).
   - Form Component – Reusable input forms for product entry, customer details, and order creation.
   - Table / List Component – Displays products, orders, and customer data in a structured format.
   - Search & Filter Component – Helps in quickly finding products, customers, or orders.
   - Notification / Alert Component – Provides feedback for success, error, or warning messages.
8. **State Management**
   - **Global State**:
   - Refers to data that is shared across the entire store manager application.
   - Example: Logged-in user info, authentication status, product catalog, inventory levels, and order details.
   - Global state ensures consistency so that changes (like updating stock after a sale) reflect across all modules (Products, Orders, Reports, etc.).
   - Managed using tools like Redux,Context API, or MobX.
   - **Local State**:
   - Refers to data handled inside a specific component only.
   - Example: Input values in a product form, modal open/close state, search bar text, or filter selection.
   - Local state improves performance by isolating temporary data within components without affecting the whole system.
   - Managed using React's useState/useReducer hooks or similar methods.
9. **User Interface**:

- Dashboard – Displays an overview of sales, stock levels, revenue, and important notifications.
- Product Management Screen – Add, update, delete, or search products with details like name, price, and stock.
- Billing/Checkout Interface – Easy interface for generating bills, applying discounts, and processing payments.
- Inventory Screen – Shows available stock, alerts for low stock, and options to reorder items.
- Customer Management – Interface for managing customer profiles, purchase history, and loyalty programs.
- Security & Access Control – Role-based UI (admin, cashier, manager) so each user.

10. **Styling**

- **CSS Frameworks/Libraries**:
- For the Store Manager application, styling is handled using Bootstrap and custom CSS. Bootstrap provides responsive design, pre-built components (like buttons, tables, modals), and a grid system that ensures the application works well across desktops, tablets, and mobiles. Additionally, some utility classes from Tailwind CSS can be used for faster prototyping and consistency.
- Pre-processors (if used):
- SCSS (Sass) can be used to manage variables, nesting, and mixins for more maintainable and scalable styles.

- **Theming**:
- The application implements a simple theme system — consistent colors, typography, and layout are used across all pages (e.g., dashboard, inventory, sales reports). Custom styles are applied to highlight key elements like low-stock alerts, sales charts, and navigation menus. If required, dark and light themes can also be added for better user experience.

11. **Testing**

- **Testing Strategy**:
- Testing Strategy:
- In the Store Manager application, testing is done at multiple levels:
- Unit Testing: Ensures that individual components such as product listing, billing, and stock updates work correctly.
- Integration Testing: Checks if modules like sales, inventory, and reports interact smoothly
- End-to-End Testing: Simulates real user workflows (e.g., adding stock, generating invoices, checking reports) to confirm the system behaves as expected.
- **Code Coverage**: Tools like Jest and React Testing Library can be used to measure how much of the application is tested. This helps in identifying untested features and ensures reliable performance.
- Outcome: Testing improves accuracy, prevents errors in billing/inventory, and guarantees a smooth user experience for store managers.

Screenshots or Demo

1. **Known Issues**

- Occasional performance lags when handling large volumes of inventory data.
- Limited support for integration with certain third-party applications.
- Some UI components may not display correctly on smaller screen devices.
- Minor bugs in advanced search and filter functions.
- Synchronization delays may occur in multi-user environments.

2. **Future Enhancements**

- Adding advanced analytics and reporting dashboards for better decision making.
- Integration with more payment gateways and third-party services.
- Introduction of AI-based inventory forecasting and sales