STORE MANAGER

Project Documentation

1.INTRODUCTION:

Project Title: STORE MANAGER-Keep track of inventory

Team ID:NM2025TMID35392

- Team member: Vinitha V
- Team member: Pasamalar G
- Team member: Varthani R
- Team member: Suriya R

1. Introduction

1.1 Project Overview

The **"Store Manager Naan Mudhalvan"** project aims to create a comprehensive store management system that helps store owners or managers efficiently manage inventory, sales, and customer relations. This platform will also provide tools for managing employees, analyzing sales data, and generating reports. The goal is to simplify operations and improve the overall efficiency and profitability of a retail or e-commerce business.

1.2 Project Objectives

- To provide a centralized platform for managing inventory, orders, and customer interactions.
- To offer real-time tracking of sales, stock levels, and financial data.
- To enable easy generation of reports and insights for better decision-making.
- To improve employee management and streamline communication within the store.
- To ensure seamless integration with existing POS (Point of Sale) systems, online stores, and accounting tools.

1.3 Target Audience

- **Store Managers**: Those responsible for overseeing the day-to-day operations of the store.
- **Retailers**: Small to large-scale businesses in retail (grocery stores, clothing, electronics, etc.).
- **E-commerce Businesses**: Companies that sell products online and need to manage orders, inventory, and shipments.
- **Warehouse Managers**: Managing inventory for large stockpiles, ensuring timely product availability.

---

2. Functional Requirements

2.1 Features

- **Inventory Management**:
    - Track product stock levels in real-time.
    - Set automatic low-stock alerts and reorder reminders.
    - Manage product categories, SKUs, and attributes (e.g., size, color).
    - Support for batch/barcode scanning for quick inventory updates.
- **Sales Management**:
    - Record sales transactions and process payments (cash, card, mobile payments).
    - Generate and issue invoices, receipts, and refunds.
    - Apply discount codes, promotions, or loyalty points at checkout.
    - Sales tax calculation based on region.
- **Customer Management**:
    - Store customer profiles and order history.
    - Loyalty program integration (e.g., reward points, special discounts).
    - Send promotional offers and marketing messages to customers.
    - Provide customer support through chat or email.
- **Order Management**:
    - Track orders from customers (online or in-store).
    - Update order status (processing, shipped, delivered, cancelled).
    - Generate packing slips and shipping labels for fulfillment.
- **Employee Management**:
    - Track work hours and manage employee schedules.
    - Provide access control for employees (e.g., cashier, manager, admin).
    - Generate payroll reports based on employee hours worked.
- **Sales Analytics & Reporting**:
    - Generate real-time reports on sales, inventory turnover, and revenue.
    - Analyze customer behavior and buying patterns (e.g., most popular items, average spend).
    - Profit and loss reports, tax reports, and other financial analytics.
    - Forecast demand and inventory needs based on historical data.

- **Integration with POS Systems**:
    - Integration with existing POS systems to sync sales and inventory data automatically.
    - Integration with popular e-commerce platforms (e.g., Shopify, WooCommerce, Magento).
- **Mobile App**:
    - Mobile support for store managers and staff to access and update inventory or sales data remotely.
    - Barcode scanning for inventory management using smartphones or tablets.
- **Multi-Store Management (for chains/franchisees)**:
    - Manage multiple stores under a single account.
    - Track performance and stock levels across all locations.
    - Centralized inventory updates for all stores.

---

3. Non-Functional Requirements

3.1 Performance

- The platform should load within 3 seconds to ensure smooth user experience.
- Real-time inventory updates and sales tracking with minimal delay.
- Scalability to handle increasing amounts of data as business grows.

3.2 Scalability

- The system should handle increasing data, including product catalogs, sales, and customer records, without degrading performance.
- Support for adding new stores, locations, and employees as needed.

3.3 Security

- Role-based access control to limit user permissions (e.g., manager, employee).
- Secure payment processing (PCI-DSS compliant for handling card payments).
- Data encryption to protect sensitive customer and business information.
- Regular security audits to prevent unauthorized access or data breaches.

3.4 Usability

- Intuitive user interface (UI) for both novice and experienced users.
- Easy-to-follow navigation for inventory management, sales tracking, and reporting.
- Mobile-optimized design for on-the-go management.

---

4. Technical Architecture

4.1 Technology Stack

- **Frontend**: HTML, CSS, JavaScript, React.js or Vue.js (for interactive, dynamic user interfaces)
- **Backend**: Node.js with Express.js or Django (for API services and business logic)
- **Database**: PostgreSQL or MySQL (for relational data storage of products, customers, and sales transactions)
- **Payment Gateway**: Stripe, PayPal, or Razorpay (for handling transactions securely)
- **Cloud**: AWS, Google Cloud, or Azure (for hosting the platform and ensuring scalability)
- **Authentication**: OAuth 2.0, JWT (JSON Web Tokens) for secure login and user management
- **Barcode Scanning**: Integration with APIs or hardware for barcode/QR scanning for inventory management (e.g., using ZXing or a similar library).

4.2 Database Design

- **Tables/Entities**:
    - **Products**: Product ID, name, description, SKU, price, stock level, category, supplier.
    - **Customers**: Customer ID, name, email, phone number, address, loyalty points.
    - **Orders**: Order ID, customer ID, order date, order status, shipping address, total amount.
    - **Sales**: Sales ID, product ID, quantity, sale date, payment method.
    - **Employees**: Employee ID, name, role, shift timings, salary.
    - **Inventory Movements**: Movement ID, product ID, quantity, movement type (purchase, sale, return).

4.3 System Architecture

- The system will follow a **client-server** architecture:
    - **Frontend**: A React.js or Vue.js web interface that interacts with the backend through RESTful APIs.
    - **Backend**: A Node.js/Express or Django backend handling business logic, data management, and authentication.
    - **Database**: A relational database like MySQL/PostgreSQL will store product, order, and customer data.
    - **Payment Integration**: Secure third-party APIs like Stripe/PayPal to process transactions.

---

5. User Interface Design

5.1 Wireframes

- **Dashboard**: Displays sales performance, inventory levels, and key metrics at a glance.
- **Inventory Management Page**: A list of all products with stock levels, sales, and low stock alerts.
- **Sales Transactions Page**: A page showing daily sales, payment types, and pending orders.
- **Customer Profile Page**: View customer details, order history, and loyalty points.
- **Employee Management Page**: Manage employee schedules, track work hours, and manage payroll.
- **Reports Page**: Generate and view reports on sales, inventory turnover, profits, and more.

5.2 User Flow

1. **User Login**: Manager/Employee logs into the system.
2. **Dashboard**: Overview of sales, inventory, and key metrics.
3. **Inventory Management**: Add or update product details, adjust stock levels, and view product performance.
4. **Sales Processing**: Record sales transactions, generate receipts, and manage payments.
5. **Order Fulfillment**: Track and process incoming orders, update status, and generate shipping labels.
6. **Reports**: Generate real-time financial, sales, or inventory reports.

---

6. Project Timeline

6.1 Milestones

1. **Phase 1**: Requirements gathering and system design (2 weeks)
2. **Phase 2**: UI/UX design and prototyping (3 weeks)
3. **Phase 3**: Backend and API development (6 weeks)
4. **Phase 4**: Frontend development and integration with backend (5 weeks)
5. **Phase 5**: Testing and Quality Assurance (4 weeks)
6. **Phase 6**: Deployment, training, and final feedback (2 weeks)

6.2 Risk Management

- **Risks**:
    - Integration issues with existing POS systems.
    - Delays in payment gateway integration.
    - Lack of user engagement or feedback during early stages.
- **Mitigation Strategies**:

- Early testing of payment systems and POS integrations.
- Regular check-ins with the development team and key stakeholders.
- Collect user feedback via beta testing and continuously improve the system.

---

7. Testing and Quality Assurance

7.1 Testing Strategies

- **Unit Testing**: Test individual backend functions (e.g., payment processing, inventory updates).
- **Integration Testing**: Test how the frontend and backend work together (e.g., order processing, inventory updates).
- **User Acceptance Testing (UAT)**: Have real users test the system with their daily workflows.
- **Load Testing**: Ensure the system can handle high volumes of transactions and users during peak times (e.g., Black Friday, sale events).